测试理论

1 什么是软件测试?

验证程序正常运行 2.验证满足用户需求 3.发现缺陷

2 软件测试的目的?

发现软件缺陷，提高软件质量。

3 什么是需求文档测试：

1. 需求文档是否符合用户需求；2、需求文档是否符合逻辑；3、技术上是否能实现；

4 什么是设计文档测试?

测试设计是否符合全部需求以及设计是否合理。

5 什么是α测试?

Alpha 测试是指软件开发公司组织内部人员模拟各类用户对即将面市的软件产品（称为α版本）进行测试，试图发现错误并修正。α测试的关键在于尽可能逼真的模拟实际运行环境和用户对软件产品的操作尽最大努力涵盖所有可能的用户操作方式。（公司内部测）

6 什么是β测试?

β测试是由潜在用户/客户在开发环境之外多个用户在一个或多个用户的实际使用环境下进行的测试 没有开发工程师参与下进行的软件测试（内（用户）测）

7 什么是驱动模块？

驱动模块在大多数场合称为"主程序"，它接收测试数据并将这些数据传递到被测试模块.单元测试一个函数单元时，被测单元本身是不能独立运行的，需要为其传送数据，为此写驱动  
驱动模块主要完成以下事情：  
1、接受测试输入;  
2、对输入进行判断;  
3、将输入传给被测单元，驱动被测单元执行;  
4、接受被测单元执行结果，并对结果进行判断;  
5、将判断结果作为用例执行结果输出测试报告。

8 什么是桩模块?

桩模块（Stub）是指模拟被测试的模块所调用的模块，而不是软件产品的组成的部分。主模块作为驱动模块,与之直接相连的模块用桩模块代替。在集成测试前要为被测模块编制一些模拟其下级模块功能的“替身”模块，以代替被测模块的接口，接受或传递被测模块的数据，这些专供测试用的“假”模块称为被测模块的桩模块。

9 什么是白盒测试?

白盒测试(White-box Testing，又称逻辑驱动测试,结构测试),它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能，白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。  
对开发语言的支持：白盒测试工具是对源代码进行的测试，测试的主要内容包括词法分析与语法分析、静态错误分析、动态检测等。目前测试工具主要支持的开发语言包括：标准C、C++、Visual C++、Java、Visual J++等。

10 什么是静态测试?

通过运行程序测试软件称为动态测试. 通过评审文档、阅读代码等方式测试软件称为静态测试,在动态测试中,通常使用白盒测试和黑盒测试从不同的角度设计测试用例,查找软件代码中的错误. 静态测试方法是指不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性。对需求规格说明书、软件设计说明书、源程序做结构分析、流程图分析、符号执行来找错。静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错，并为测试用例选取提供指导。

11 什么是回归测试?

回归测试的目的是在程序有修改的情况下，保证原有功能正常的一种测试策略和方法。  
说白了就是，我们测试人员在对程序进行测试时发现bug，然后返还程序员修改，程序员修改后发布新的软件包或新的软件补丁包给我们测试人员，我们就要重新对这个程序测试，已保证程序在修正了以前bug的情况下，正常运行，且不会带来新的错误的这样一个过程。 一般情况下是不需要全面测试的，而是根据修改的情况进行有效的测试。

12 白盒测试有哪几种方法?

白盒测试分为静态和动态两类： 静态：不执行程序，执行静态结构分析，代码走查，静态质量度量等 动态：基本路径，逻辑覆盖（语句覆盖，判断覆盖，条件覆盖，判断-条件覆盖，条件组合覆盖，路径覆盖）等   
1.语句覆盖：可执行语句至少被执行一次；  
2.判断覆盖：每个判断的取真分支和取假分支至少经历一次；  
3.条件覆盖：每个条件的取值至少满足一次；

4.判断条件覆盖：判断和条件都满足；  
5.条件组合覆盖：每个条件的所有可能都至少出现一次，并且判定结果至少出现一次 ；  
他与条件覆盖的区别：他不是简单要求每个条件出现“真”和“假”两种结果，而是要求这些结果所有可能至少出现一次

13 软件的缺陷等级应如何划分?

软件缺陷的等级可以用严重性和优先级来描述；  
严重性：衡量缺陷对客户满意度影响的满意程度，分为  
1.致命错误，可能导致本模块以及其他相关的模块异常，死机等问题；  
2.严重错误，问题局限在本模块，导致模块功能失常或异常退出；  
3.一般错误，模块功能部分失效；  
4.建议模块，有问题提出人对测试模块的改进建议；  
优先级：缺陷被修复的紧急程度；  
1.立即解决（P1级）：缺陷导致系统功能几乎不能使用或者测试不能继续，需立即修复；  
2.高优先级（P2级）：缺陷严重，影响测试，需优先考虑；  
3.正常排队（P3级）：缺陷需要正常排队等待修复；  
4.低优先级（P4级）：缺陷可以在有时间的时候被纠正；

14 如果能够执行完美的黑盒测试，还需要进行白盒测试吗?(白盒与黑盒的区别)

任何工程产品都可以使用以下两种方法之一进行测试。  
黑盒测试：已知产品的功能设计规格，可以进行测试证明每个实现了的功能是否符合要求。 白盒测试：已知产品的内部工作过程，可以通过测试证明每种内部操作是否符合设计规格要求，所有内部成分是否以经过检查。  
软件的黑盒测试意味着测试要在软件的接口处进行。这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。因此黑盒测试又叫功能测试或数据驱动测试。

黑盒测试主要是为了发现以下几类错误：  
1、是否有不正确或遗漏的功能?  
2、在接口上，输入是否能正确的接受?能否输出正确的结果?  
3、是否有数据结构错误或外部信息(例如数据文件)访问错误?  
4、性能上是否能够满足要求?  
5、是否有初始化或终止性错误?  
软件的白盒测试是对软件的过程性细节做细致的检查。这种方法是把测试对象看做一个打开的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序状态，确定实际状态是否与预期的状态一致。因此白盒测试又称为结构测试或逻辑驱动测试。白盒测试主要是想对程序模块进行如下检查：  
1、对程序模块的所有独立的执行路径至少测试一遍。  
2、对所有的逻辑判定，取“真”与取“假”的两种情况都能至少测一遍。  
3、在循环的边界和运行的界限内执行循环体。  
4、测试内部数据结构的有效性等等。  
以上事实说明，软件测试有一个致命的缺陷，即测试的不完全、不彻底性。由于任何程序只能进行少量(相对于穷举的巨大数量而言)的有限的测试，在未发现错误时，不能说明程序中没有错误。

从以上可以看出就算执行了完美的黑盒测试也是无法测试程序内部特定部位，另外当规格说明本身有误，也不能发现问题。而白盒测试能对程序的内部特定部位进行覆盖测试，所以黑盒和白盒测试为互补关系，结合起来进行测试用例的设计更为合理。

经验表明，通常在进行单元测试时采用白盒测试方法，集成测试采用灰盒测试方法，系统测试采用黑盒测试方法

15 软件测试应该划分几个阶段?简述各个阶段应重点测试的点?各个阶段的含义?

软件测试有五个阶段：单元测试、集成测试、系统测试、验收测试、回归测试

单元测试是对软件中的基本组成单元进行测试，如一个模块、一个过程等等，它是软件动态测试的最基本的部分，也是最重要的部分之一，其目的是检验软件最基本组成单元的正确性。

集成测试是在软件系统集成过程中所进行的测试，其主要目的是检验软件各单位之间的接口是否正确。

系统测试是对已经集成好的软件系统进行彻底的测试，已验证软件系统的正确性和性能等满足其规约所指定的要求，检查软件的行为和输出是否正确。

验收测试旨在向软件的购买者展示该软件满足其用户的需求。它的测试数据通常是系统测试的测试数据的子集。

回归测试是在软件的维护阶段，对软件进行修改之后进行的测试，其目的是检验对软件的修改是否正确。

16 针对缺陷采取怎样的管理措施?

1. 要更好的管理缺陷，必须引入缺陷管理工具，商用的或者开源的都可。  
2. 根据缺陷的生命周期，考虑缺陷提交的管理、缺陷状态的管理和缺陷分析的管理。  
3. 所有发现的缺陷都必须全部即时的、准确的提交到缺陷管理工具中，这是缺陷提交的管理。  
4. 缺陷提交后，需要即时的指派给相应的开发人员，提交缺陷的人需要密切注意缺陷的状态， 帮助缺陷的尽快解决。缺陷解决后需要即时对缺陷的修复进行验证。这样的目的有两个：一个是让缺陷尽快解决;二是方便后面缺陷的分析，这是缺陷状态的管理。

5. 为了更好的改进开发过程和测试过程，需要对缺陷进行分析，总结如缺陷的类别、缺陷的龄期分布等信息，这是缺陷分析的管理。

17 单元测试、集成测试、系统测试的侧重点是什么?

单元测试是在软件开发过程中要进行的最低级别的测试活动，在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试，测试重点是系统的模块，包括子程序的正确性验证等。

集成测试，也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求，组装成为子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。测试重点是模块间的衔接以及参数的传递等。  
系统测试是将经过测试的子系统装配成一个完整系统来测试。它是检验系统是否确实能提供系统方案说明书中指定功能的有效方法。测试重点是整个系统的运行以及与其他软件的兼容性。

18 设计用例的方法、依据有那些?

白盒测试用例设计方法:

基本路径测试\边界值分析\覆盖测试\循环测试\数据流测试\程序插桩测试\变异测试.

依据就是详细设计说明书及其代码结构  
黑盒测试用例设计方法:

基于用户需求的测试\功能图分析方法\等价类划分方法\边界值分析方法\错误推测方法（根据经验或直觉推测程序中可能存在的各种错误）\因果图方法（因果图法是一种适合于描述对于多种输入条件组合的测试方法，根据输入条件的组合、约束关系和输出条件的因果关系，分析输入条件的各种组合情况，从而设计[测试用例](https://baike.so.com/doc/150233-158747.html)的方法，它适合于检查程序输入条件涉及的各种组合情况。）\判定表驱动分析方法\正交实验设计方法.

依据是用户需求规格说明书,详细设计说明书。

19 描述使用bugzilla缺陷管理工具对软件缺陷(BUG)跟踪的管理的流程

1) 测试人员或开发人员发现bug后，判断属于哪个模块的问题，填写bug报告后，系统会自动通过Email通知项目组长或直接通知开发者。  
2) 经验证无误后，修改状态为VERIFIED.待整个产品发布后，修改为CLOSED.  
3) 还有问题，REOPENED，状态重新变为“New"，并发邮件通知。  
4) 项目组长根据具体情况，重新reassigned分配给bug所属的开发者。  
5) 若是，进行处理，resolved并给出解决方法。(可创建补丁附件及补充说明)  
6) 开发者收到Email信息后，判断是否为自己的修改范围。  
7) 若不是，重新reassigned分配给项目组长或应该分配的开发者。  
8) 测试人员查询开发者已修改的bug，进行重新测试。

20 请你分别介绍一下单元测试、集成测试、系统测试、验收测试、回归测试

1、单元测试：完成最小的软件设计单元（模块）的验证工作，目标是确保模块被正确的编码，使用过程设计描述作为指南，对重要的控制路径进行测试以发现模块内的错误，通常情况下是白盒的，对代码风格和规则、程序设计和结构、业务逻辑等进行静态测试，及早的发现和解决不易显现的错误。

2、集成测试：通过测试发现与模块接口有关的问题。目标是把通过了单元测试的模块拿来，构造一个在设计中所描述的程序结构，应当避免一次性的集成（除非软件规模很小），而采用增量集成。

自顶向下集成：模块集成的顺序是首先集成主模块，然后按照控制层次结构向下进行集成，隶属于主模块的模块按照深度优先或广度优先的方式集成到整个结构中去。

自底向上集成：从原子模块开始来进行构造和测试，因为模块是自底向上集成的，进行时要求所有隶属于某个给顶层次的模块总是存在的，也不再有使用稳定测试桩的必要。

3、系统测试：是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不相符合或与之矛盾的地方。系统测试的对象不仅仅包括需要测试的产品系统的软件，还要包含软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等。因此，必须将系统中的软件与各种依赖的资源结合起来，在系统实际运行环境下来进行测试。

4、回归测试：回归测试是指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。理论上，软件产生新版本，都需要进行回归测试，验证以前发现和修复的错误是否在新软件版本上再次出现。根据修复好了的缺陷再重新进行测试。回归测试的目的在于验证以前出现过但已经修复好的缺陷不再重新出现。一般指对某已知修正的缺陷再次围绕它原来出现时的步骤重新测试。

5、验收测试：验收测试是指系统开发生命周期方法论的一个阶段，这时相关的用户或独立测试人员根据测试计划和结果对系统进行测试和接收。它让系统用户决定是否接收系统。它是一项确定产品是否能够满足合同或用户所规定需求的测试。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。

21 请你回答一下单元测试、集成测试、系统测试、验收测试、回归测试这几步中最重要的是哪一步

这些测试步骤分别在软件开发的不同阶段对软件进行测试，我认为对软件完整功能进行测试的系统测试很重要，因为此时单元测试和集成测试已完成，能够对软件所有功能进行功能测试，能够覆盖系统所有联合的部件，是针对整个产品系统进行的测试，能够验证系统是否满足了需求规格的定义，因此我认为系统测试很重要。

22 请回答集成测试和系统测试的区别，以及它们的应用场景主要是什么？

区别：

1、计划和用例编制的先后顺序：从V模型来讲，在需求阶段就要制定系统测试计划和用例，HLD的时候做集成测试计划和用例，有些公司的具体实践不一样，但是顺序肯定是先做系统测试计划用例，再做集成。

2、用例的粒度：系统测试用例相对很接近用户接受测试用例，集成测试用例比系统测试用例更详细，而且对于接口部分要重点写，毕竟要集成各个模块或者子系统。

3、执行测试的顺序：先执行集成测试，待集成测试出的问题修复之后，再做系统测试。

应用场景：

集成测试：完成单元测试后，各模块联调测试；集中在各模块的接口是否一致、各模块间的数据流和控制流是否按照设计实现其功能、以及结果的正确性验证等等；可以是整个产品的集成测试，也可以是大模块的集成测试；集成测试主要是针对程序内部结构进行测试，特别是对程序之间的接口进行测试。集成测试对测试人员的编写脚本能力要求比较高。测试方法一般选用黑盒测试和白盒测试相结合。

系统测试：针对整个产品的全面测试，既包含各模块的验证性测试（验证前两个阶段测试的正确性）和功能性（产品提交个用户的功能）测试，又包括对整个产品的健壮性、安全性、可维护性及各种性能参数的测试。系统测试测试软件《需求规格说明书》中提到的功能是否有遗漏，是否正确的实现。做系统测试要严格按照《需求规格说明书》，以它为标准。测试方法一般都使用黑盒测试法。

23 请问测试开发需要哪些知识？需要具备什么能力？

需要的知识：

软件测试基础理论知识，如黑盒测试、白盒测试等；

编程语言基础，如C/C++、java、python等；

自动化测试工具，如Selenium、Appium、Robotium等；

计算机基础知识，如数据库、Linux、计算机网络等；

测试框架，如JUnit等。

需要具备的能力：

业务分析能力，分析整体业务流程、分析被测业务数据、分析被测系统架构、分析被测业务模块、分析测试所需资源、分析测试完成目标；

缺陷洞察能力，一般缺陷的发现能力、隐性问题的发现能力、发现连带问题的能力、发现问题隐患的能力、尽早发现问题的能力、发现问题根源的能力；

团队协作能力，合理进行人员分工、协助组员解决问题、配合完成测试任务、配合开发重现缺陷、督促项目整体进度、出现问题勇于承担；

专业技术能力，掌握测试基础知识、掌握计算机知识、熟练运用测试工具；

逻辑思考能力，判断逻辑的正确性、对可行性逻辑分析、站在客观角度思考；

问题解决能力，技术上的问题、工作中的问题、沟通问题；

沟通表达能力，和技术人员、产品人员、上下级的沟通；

宏观把控能力，有效控制测试时间、有效控制测试成本、有效制定测试计划、有效进行风险评估、有效控制测试方向。

24 请说一说黑盒与白盒的测试方法

常用的黑盒测试方法有：等价类划分法；边界值分析法；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1.语句覆盖每条语句至少执行一次。

2.判定覆盖每个判定的每个分支至少执行一次。

3.条件覆盖每个判定的每个条件应取到各种可能的值。

4.判定/条件覆盖同时满足判定覆盖条件覆盖。

5.条件组合覆盖每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖使程序中每一条可能的路径至少执行一次。

25 请说一下手动测试与自动化测试的优缺点

手工测试缺点：

1、重复的手工回归测试，代价昂贵、容易出错。

2、依赖于软件测试人员的能力。

手工测试优点：

1、测试人员具有经验和对错误的猜测能力。

2、测试人员具有审美能力和心理体验。

3、测试人员具有是非判断和逻辑推理能力。

自动化测试的优点：

1、对程序的回归测试更方便。这可能是自动化测试最主要的任务，特别是在程序修改比较频繁时，效果是非常明显的。由于回归测试的动作和用例是完全设计好的，测试期望的结果也是完全可以预料的，将回归测试自动运行，可以极大提高测试效率，缩短回归测试时间。

2、可以运行更多更繁琐的测试。自动化的一个明显的好处是可以在较少的时间内运行更多的测试。

3、可以执行一些手工测试困难或不可能进行的测试(压力测试)（负载测试）。比如，对于大量用户的测试，不可能同时让足够多的测试人员同时进行测试，但是却可以通过自动化测试模拟同时有许多用户，从而达到测试的目的。

4、更好地利用资源。将繁琐的任务自动化，可以提高准确性和测试人员的积极性，将测试技术人员解脱出来投入更多精力设计更好的测试用例。有些测试不适合于自动测试，仅适合于手工测试，将可自动测试的测试自动化后，可以让测试人员专注于手工测试部分，提高手工测试的效率。

5、测试具有一致性和可重复性。由于测试是自动执行的，每次测试的结果和执行的内容的一致性是可以得到保障的，从而达到测试的可重复的效果。

6、测试的复用性。由于自动测试通常采用脚本技术，这样就有可能只需要做少量的甚至不做修改，实现在不同的测试过程中使用相同的用例。

7、增加软件信任度。由于测试是自动执行的，所以不存在执行过程中的疏忽和错误，完全取决于测试的设计质量。一旦软件通过了强有力的自动测试后，软件的信任度自然会增加。

自动化测试的缺点：

1、不能取代手工测试

2、手工测试比自动测试发现的缺陷更多

3、对测试质量的依赖性极大

4、测试自动化不能提高有效性

5、测试自动化可能会制约软件开发。由于自动测试比手动测试更脆弱，所以维护会受到限制，从而制约软件的开发。

6、工具本身并无想像力

26 请问你怎么看待软件测试的潜力和挑战

主要潜力：随着互联网和软件行业的日渐成熟，测试也逐渐的被各个公司所重视。而且软件测试行业比较稳定，在未来的5-10年内发展会很快，人才缺口极大。软件测试的前景只会是越来越好，对专业素质要求也会越来越高。总结来说就是前景非常不错的。

有十大挑战： 1.软件测试人员被认为低人一等； 2.软件测试时间永远不够的； 3.缺乏简单易用的软件测试辅助工具； 4.缺乏具体通用的软件测试技术； 5.很难清楚了解用户需求和期望； 6.缺乏可明确衡量测试质量达标的度量； 7.很难确定一个测试实例是否执行完毕； 8.很难找出时间做软件自动化测试； 9.软件测试所需要文档经常不全面； 10.很多任务在身，很难保障质量。

27 你觉得软件测试的核心竞争力是什么

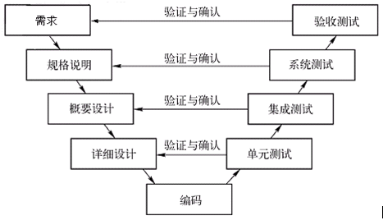
测试人员的核心竞争力在于提早发现问题，并能够发现别人无法发现的问题。

1、早发现问题：问题发现的越早，解决的成本越低。如果一个需求在还未实现的时候就能发现需求的漏洞，那么这种问题的价值是最高的。

2、发现别人无法发现的问题：所有人都能发现的问题，你发现了，那就证明你是可以被替代的。别人发现不了，而你可以发现，那么你就是无法被替代。

28 你觉得测试和开发需要怎么结合才能使软件的质量得到更好的保障

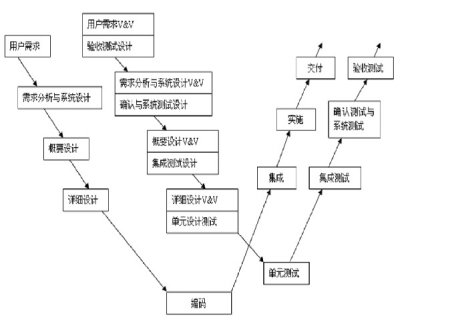
测试和开发应该按照W模型的方式进行结合，测试和开发同步进行，能够尽早发现软件缺陷，降低软件开发的成本。



在V模型中，测试过程被加在开发过程的后半部分，单元测试所检测代码的开发是否符合详细设计的要求。集成测试所检测此前测试过的各组成部分是否能完好地结合到一起。系统测试所检测已集成在一起的产品是否符合系统规格说明书的要求。而验收测试则检测产品是否符合最终用户的需求。V模型的缺陷在于仅仅把测试过程作为在需求分析、系统设计及编码之后的一个阶段，忽视了测试对需求分析、系统设计的验证，因此需求阶段的缺陷很可能一直到后期的验收测试才被发现，此时进行弥补将耗费大量人力物力资源。

相对于V模型，W模型增加了软件各开发阶段中应同步进行的验证和确认活动。W模型由两个V字型模型组成，分别代表测试与开发过程，图中明确表示出了测试与开发的并行关系。

W模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。W模型有利于尽早地全面的发现问题。例如，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。



W模型中测试的活动与软件开发同步进行，测试的对象不仅仅是程序，还包括需求和设计，因此能够尽早发现软件缺陷，降低软件开发的成本

29 你觉得单元测试可行吗

可行，单元测试可以有效地测试某个程序模块的行为，是未来重构代码的信心保证。事前可以保证质量，事后可以快速复现问题，并在修改代码后做回归自测。可行性考虑的是要用一些可行的方法做到关键的代码可测试，如通过边界条件、等价类划分、错误、因果，设计测试用例要覆盖常用的输入组合、边界条件和异常。

30 你觉得自动化测试有什么意义，都需要做些什么

自动化测试的意义在于

1、可以对程序的新版本自动执行回归测试

2、可以执行手工测试困难或者不可能实现的测试，如压力测试，并发测试，

3、能够更好的利用资源，节省时间和人力

执行自动化测试之前首先判断这个项目是不适合推广自动化测试，然后对项目做需求分析，指定测试计划，搭建自动化测试框架，设计测试用例，执行测试，评估

31 请你回答一下测试的相关流程是什么？

测试最规范的过程如下

需求测试->概要设计测试->详细设计测试->单元测试->集成测试->系统测试->验收测试

（W模型）

32 请你说一下如何写测试用例

1、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

2、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

3、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

4、找到需求相关的一些特性，补充测试用例

5、根据自己的经验分析遗漏的测试场景

6、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

7、书写格式一定要清晰

33 请问你觉得测试项目具体工作是什么？

搭建测试环境

撰写测试用例

执行测试用例

写测试计划，测试报告，测试，并提交BUG表单，跟踪bug修改情况

执行自动化测试，编写脚本，执行，分析，报告

进行性能测试，压力测试等其他测试，执行，分析，调优，报告

34 请问如果想进行bug的测评，怎么去评测bug？

Bug的priority（）和severity（）是两个重要属性，通常人员在提交bug的时候，只定义severity，而将priority交给leader定义，通常bug管理中，severity分为四个等级blocker、critical、major、minor/trivial，而priority分为五个等级immediate、urgent、high、normal、low。

Severity：

1、blocker：即系统无法执行，崩溃，或严重资源不足，应用模块无法启动或异常退出，无法测试，造成系统不稳定。常见的有严重花屏、内存泄漏、用户数据丢失或破坏、系统崩溃/死机/冻结、模块无法启动或异常退出、严重的数值计算错误、功能设计与需求严重不符、其它导致无法测试的错误， 如服务器500错误。

2、critical：即影响系统功能或操作，主要功能存在严重缺陷，但不会影响到系统稳定性。常见的有：功能未实现，功能错误、系统刷新错误、数据通讯错误、轻微的数值计算错误、影响功能及界面的错误字或拼写错误。

3、major：即界面、性能缺陷、兼容性，常见的有：操作界面错误，边界条件错误，提示信息错误，长时间操作无进度提示，系统未优化，兼容性问题。

4、minor/trivial：即易用性及建议性问题。

Priority

1、immediate：即马上解决，

2、urgent：急需解决

3、high：高度重视，有时间要马上解决

4、low：在系统发布前解决，或确认可以不用解决。

35 请你说一说测试用例的边界

边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充，这种情况下，其测试用例来自等价类的边界。

常见的边界值

1)对16-bit 的整数而言 32767 和 -32768 是边界

2)屏幕上光标在最左上、最右下位置

3)报表的第一行和最后一行

4)数组元素的第一个和最后一个

5)循环的第 0 次、第 1 次和倒数第 2 次、最后一次

36 请你说一下软件质量的六个特征

按照软件质量国家标准GB-T8566--2001G，软件质量可以用下列特征来评价：

a.功能特征：与一组功能及其指定性质有关的一组属性，这里的功能是满足明确或隐含的需求的那些功能。

b.可靠特征：在规定的一段时间和条件下，与软件维持其性能水平的能力有关的一组属性。

c.易用特征：由一组规定或潜在的用户为使用软件所需作的努力和所作的评价有关的一组属性。

d.效率特征：与在规定条件下软件的性能水平与所使用资源量之间关系有关的一组属性。

e.可维护特征：与进行指定的修改所需的努力有关的一组属性。

f.可移植特征：与软件从一个环境转移到另一个环境的能力有关的一组属性。

37 请你说一下设计测试用例的方法

黑盒测试：

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。等价类可以划分为有效等价类和无效等价类，设计测试用例的时候要考虑这两种等价类。

2.边界值分析法

边界值分析法是对等价类划分的一种补充，因为大多数错误都在输入输出的边界上。边界值分析就是假定大多数错误出现在输入条件的边界上，如果边界附件取值不会导致程序出错，那么其他取值出错的可能性也就很小。

边界值分析法是通过优先选择不同等价类间的边界值覆盖有效等价类和无效等价类来更有效的进行测试，因此该方法要和等价类划分法结合使用。

3.正交试验法

正交是从大量的试验点中挑选出适量的、有代表性的点。正交试验设计是研究多因素多水平的一种设计方法，他是一种基于正交表的高效率、快速、经济的试验设计方法。

4.状态迁移法

状态迁移法是对一个状态在给定的条件内能够产生需要的状态变化，有没有出现不可达的状态和非法的状态，状态迁移法是设计足够的用例达到对系统状态的覆盖、状态、条件组合、状态迁移路径的覆盖。

5.流程分析法

流程分析法主要针对测试场景类型属于流程测试场景的测试项下的测试子项进行设计，这是从白盒测试中路径覆盖分析法借鉴过来的一种很重要的方法。

6.输入域测试法

输入域测试法是针对输入会有各种各样的输入值的一个测试，他主要考虑 极端测试、中间范围测试，特殊值测试 。

7.输出域分析法

输出域分析法是对输出域进行等价类和边界值分析，确定是要覆盖的输出域样点，反推得到应该输入的输入值，从而构造出测试用例，他的目的是为了达到输出域的等价类和边界值覆盖。

8.判定表分析法

判定表是分析和表达多种输入条件下系统执行不同动作的工具，他可以把复杂的逻辑关系和多种条件组合的情况表达的即具体又明确；

9.因果图法

因果图是用于描述系统输入输出之间的因果关系、约束关系。因果图的绘制过程是对被测系统的外部特征的建模过程，根据输入输出间的因果图可以得到判定表，从而规划出测试用例。

10.错误猜测法

错误猜测法主要是针对系统对于错误操作时对于操作的处理法的猜测法，从而设计测试用例

11.异常分析法

异常分析法是针对系统有可能存在的异常操作，软硬件缺陷引起的故障进行分析，分析发生错误时系统对于错误的处理能力和恢复能力依此设计测试用例。

白盒测试：

白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了一些与数据相关的错误。

白盒测试需要遵循的原则有：1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1.语句覆盖每条语句至少执行一次。

2.判定覆盖每个判定的每个分支至少执行一次。

3.条件覆盖每个判定的每个条件应取到各种可能的值。

4.判定/条件覆盖同时满足判定覆盖条件覆盖。

5.条件组合覆盖每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖使程序中每一条可能的路径至少执行一次。

39 请你说一下app性能测试的指标

1、内存：内存消耗测试节点的设计目标是为了让应用不占用过多的系统资源，且及时释放内存，保障整个系统的稳定性。当然关于内存测试，在这里我们需要引入几个概念：空闲状态、中等规格、满规格。

空闲状态指打开应用后，点击home键让应用后台运行，此时应用处于的状态叫做空闲；中等规格和满规格指的是对应用的操作时间的间隔长短不一，中等规格时间较长，满规格时间较短。

内存测试中存在很多测试子项，清单如下：

●空闲状态下的应用内存消耗；

●中等规格状态下的应用内存消耗；

●满规格状态下的应用内存消耗；

●应用内存峰值；

●应用内存泄露；

●应用是否常驻内存；

●压力测试后的内存使用。

2、CPU：

使用Android提供的view plaincopy在CODE上查看代码片派生到我的代码片

Adb shell dumpsys CPUinfo |grep packagename >/address/CPU.txt来获取；

使用top命令view plaincopy在CODE上查看代码片派生到我的代码片

Adb shell top |grep packagename>/address/CPU.txt来获取。

3、流量：

网络流量测试是针对大部分应用而言的，可能还有部分应用会关注网速、弱网之类的测试。

流量测试包括以下测试项：

应用首次启动流量提示；

应用后台连续运行2小时的流量值；

应用高负荷运行的流量峰值。

4、电量：

●测试手机安装目标APK前后待机功耗无明显差异；

●常见使用场景中能够正常进入待机，待机电流在正常范围内；

●长时间连续使用应用无异常耗电现象。

5、启动速度：

第一类：首次启动--应用首次启动所花费的时间；

第二类：非首次启动--应用非首次启动所花费的时间；

第三类：应用界面切换--应用界面内切换所花费的时间。

6、滑动速度、界面切换速度

7、与服务器交互的网络速度

40 请你说一说app测试的工具

功能测试自动化

a) 轻量接口自动化测试 jmeter，

b) APP UI层面的自动化

android：UI Automator Viewer，Android Junit，Instrumentation，UIAutomator，

iOS:基于Instrument的iOS UI自动化，

性能测试

a) Web前端性能测试

网络抓包工具：Wireshark

网页文件大小

webpagetest

pagespeed insight

chrome adb

b) APP端性能测试

Android内存占用分析：MAT

iOS内存问题分析：ARC模式

Android WebView性能分析：

iOS WebView性能分析

c) 后台服务性能测试

负载，压力，耐久性

可拓展性，基准

工具：apacheAB，Jmeter，LoadRunner，

专项测试

a) 兼容性测试

手工测试：操作系统，分辨率，rom，网络类型

云平台：testin，脚本编写，Android。

b) 流量测试

Android自带的流量管理，

iOS自带的Network

tcpdump抓包

WiFi代理抓包：Fiddler

流量节省方法：压缩数据，json优于xml；WebP优于传统的JPG，PNG；控制访问的频次；只获取必要的数据；缓存；

c) 电量测试

基于测试设备的方法，购买电量表进行测试。

GSam Battery Monitoe Pro

iOS基于Instrument Energy工具

d) 弱网络测试

手机自带的网络状况模拟工具

基于代理的弱网络的模拟:

工具：windows：Network Delay Simulator

Mac：Network Link Conditioner

41 请你说一说bug的周期，以及描述一下不同类别的bug

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“

不同类别的bug：

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

42 请你说一说PC网络故障，以及如何排除障碍

(1)首先是排除接触故障，即确保你的网线是可以正常使用的。

然后禁用网卡后再启用，排除偶然故障。打开网络和共享中心窗口，单击窗口左上侧“更改适配器设置”右击其中的“本地连接“或”无线网络连接”，单击快捷菜单中的“禁用”命令，即可禁用所选网络。接下来重启网络，只需右击后单击启用即可。

(2)使用ipconfig查看计算机的上网参数

1、win+R输入cmd，打开命令提示符窗口

2、输入ipconfig，按Enter确认，可以看到机器的配置信息，输入ipconfig/all,可以看到IP地址和网卡物理地址等相关网络详细信息。

(3)使用ping命令测试网络的连通性，定位故障范围

在命令提示符窗口中输入”ping 127.0.0.1“，数据显示本机分别发送和接受了4个数据包，丢包率为零，可以判断本机网络协议工作正常，如显示”请求超时“，则表明本机网卡的安装或TCP/IP协议有问题，接下来就应该检查网卡和TCP/IP协议，卸载后重装即可。

(4)ping本机IP

在确认127.0.0.1地址能被ping通的情况下，继续使用ping命令测试本机的IP地址能否被ping通，如不能，说明本机的网卡驱动程序不正确，或者网卡与网线之间连接有故障，也有可能是本地的路由表面收到了破坏，此时应检查本机网卡的状态是否为已连接，网络参数是否设置正确，如果正确可是不能ping通，就应该重新安装网卡驱动程序。丢失率为零，可以判断网卡安装配置没有问题，工作正常。

(5)ping网关 网关地址能被ping通的话，表明本机网络连接以及正常，如果命令不成功，可能是网关设备自身存在问题，也可能是本机上网参数设置有误，检查网络参数。

45 请你说一说你知道的自动化测试框架

Junit TestNG Selenuim Xunit

1、模块化测试框架

模块化测试脚本框架（TEST MODulARITY FRAMEWORK）需要创建小而独立的可以描述的模块、片断以及待测应用程序的脚本。这些树状结构的小脚本组合起来，就能组成能用于特定的测试用例的脚本。在五种框架中，模块化框架是最容易掌握和使用的。在一个组件上方建立一个抽象层使其在余下的应用中隐藏起来，这是众所周知的编程技巧。这样应用同组件中的修改隔离开来，提供了程序设计的模块化特性。模块化测试脚本框架使用这一抽象或者封装的原理来提高自动测试组合的可维护性和可升级性。

2、测试库框架

测试库框架（Test Library Architecture）与模块化测试脚本框架很类似，并且具有同样的优点。不同的是测试库框架把待测应用程序分解为过程和函数而不是脚本。这个框架需要创建描述模块、片断以及待测应用程序的功能库文件。

3、关键字驱动或表驱动的测试框架

对于一个独立于应用的自动化框架，关键字驱动（KEYWORD DRIVEN）I9LJJ试和表驱动（TABLE DRIVEN）测试是可以互换的术语。这个框架需要开发数据表和关键字。这些数据表和关键字独立于执行它们的测试自动化工具，并可以用来“驱动＂待测应用程序和数据的测试脚本代码，关键宇驱动测试看上去与手工测试用例很类似。在一个关键字驱动测试中，把待测应用程序的功能和每个测试的执行步骤一起写到一个表中。这个测试框架可以通过很少的代码来产生大量的测试用例。同样的代码在用数据表来产生各个测试用例的同时被复用。

4、数据驱动测试框架

数据驱动（DATA DRIVEN），LJ试是一个框架。在这里测试的输入和输出数据是从数据文件中读取（数据池，ODBC源，CSV文件，EXCEL文件，ADO对象等）并且通过捕获工具生成或者手工生成的代码脚本被载入到变量中。在这个框架中，变量不仅被用来存放输入值还被用来存放输出的验证值。整个程序中，测试脚本来读取数值文件，记载测试状态和信息。这类似于表驱动测试，在表驱动测 试中，它的测试用例是包含在数据文件而不是在脚本中，对于数据而言，脚本仅仅是一个“驱动器”，或者是一个传送机构。然而，数据驱动测试不同于表驱动测试，尽管导航数据并不包含在表结构中。在数据驱动测试中，数据文件中只包含测试数据。这个框架意图减少需要执行所有测试用例所需要的总的测试脚本数。数据驱动需要很少的代码来产生大量的测试用例，这与表驱动极其类似。

5、混合测试自动化（Hybrid Test Automation）框架

最普遍的执行框架是上面介绍的所有技术的一个结合，取其长处，弥补其不足。这个混合测试框架是由大部分框架随着时间并经过若干项目演化而来的

46 请你说一说web测试和app测试的不同点

系统架构方面：

web项目，一般都是b/s架构，基于浏览器的

app项目，则是c/s的，必须要有客户端，用户需要安装客户端。

web测试只要更新了服务器端，客户端就会同步会更新。App项目则需要客户端和服务器都更新。

性能方面:

web页面主要会关注响应时间

而app则还需要关心流量、电量、CPU、GPU、Memory这些。

它们服务端的性能没区别，都是一台服务器。

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

app测试则要看分辨率，屏幕尺寸，还要看设备系统。

web测试是基于浏览器的所以不必考虑安装卸载。

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 。

此外APP还有一些专项测试：如网络、适配性。

49 请问你怎么看待测试，知道哪些测试的类型，有用过哪些测试方法？

测试是软件开发中不可或缺的一环，测试通过经济，高效的方法，捕捉软件中的错误，从而达到保证软件内在质量的目的。

测试分为功能测试和非功能测试，非功能测试又可以分为性能测试、压力测试、容量测试、健壮性测试、安全性测试、可靠性测试、恢复性测试、备份测试、协议测试、兼容性测试、可用性测试、配置测试、GUI测试。

测试方法用过等价划分法、边值分析法、错误推测法、因果图法。

50 请问你怎么测试网络协议

协议测试包括四种类型的测试

1、一致性测试：检测协议实现本身与协议规范的符合程度

2、互操作性测试：基于某一协议检测不同协议实现间互操作互通信的能力

3、性能测试：检测协议实现的性能指标，比如数据传输速度，连接时间，执行速度，吞吐量，并发度

4、健壮性测试：检测协议是现在各种恶劣环境下运行的能力，比如注入干扰报文，通信故障，信道被切断

52 给你一个字符串，你怎么判断是不是ip地址？手写这段代码，并写出测试用例

IP的格式：(1~255).(0~255).(0~255).(0~255)

测试用例：

等价类划分：

|  |
| --- |
| 有效可用的IP地址 |
| A类 | 1.0.0.0 -126.255.255.254 |
| A私有 | 10.0.0.0 -10.255.255.254 |
| B类 | 128.0.0.0 -191.255.255.254 |
| B私有 | 172.16.0.0 -172.31.255.254 |
| C类 | 192.0.0.0 -223.255.255.254 |
| C私有 | 192.168.0.0-192.168.255.254 |
| windows自动分配 | 169.254.0.0-169.254.255.254 |
| 有效但不可用的IP地址 |  |
| D | 224.0.0.0 -239.255.255.254 |
| E | 240.0.0.0 -255.255.255.254 |
| 全网 | 0.x.x.x, x.x.x.0 |
| 广播 | x.x.x.255 |

|  |  |
| --- | --- |
| 回环 | 127.0.0.0 -127.255.255.254 |

|  |  |
| --- | --- |
| 输入 | 结果 |
| 64.11.22.33 | 有效可用 |
| 10.12.13.14 | 有效可用，不能直接访问公网 |
| 151.123.234.56 | 有效可用 |
| 172.20.123.56 | 有效可用，不能直接访问公网 |
| 192.127.35.65 | 有效可用 |
| 192.168.128.128 | 有效可用，不能直接访问公网 |
| 169.254.15.200 | 有效可用，不能直接访问公网 |
| 224.1.2.3 | 有效不可用，超过有效范围（D类） |
| 250.11.22.33 | 有效不可用，超过有效范围（E类） |
| 0.200.3.4 | 有效不可用，全网地址 |
| 64.11.22.0 | 有效不可用，全网地址 |
| 10.12.13.255 | 有效不可用，广播地址 |
| 127.50.60.70 | 有效不可用，回环地址 |

53 请进行测试用例设计：一串数字，闰年的判别

判断闰年的标准是:能整除4且不能整除100，能整除400。设定合法的年份为1-9999

public class Test2 {

public static void main(String[] args) {

Scanner in = new Scanner (System.in);

int year=in.nextInt();

if(year<=0||year>9999)

{

System.out.println("请输入正确的年份");

}

if((year%4==0&&year%100!=0)||year%400==0)

{

System.out.println("闰年");

}else

{

System.out.println("不是闰年");

}

}

}



54 请你说一说简单用户界面登陆过程都需要做哪些分析

一、功能测试

1.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。

2.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。

3.登录成功后能否跳转到正确的页面

4.用户名和密码，如果太短或者太长，应该怎么处理

5.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

6.记住用户名的功能

7.登陆失败后，不能记录密码的功能

8.用户名和密码前后有空格的处理

9.密码是否非明文显示，使用星号圆点等符号代替。

10.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

11.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

12.输入密码的时候，大写键盘开启的时候要有提示信息。

13.什么都不输入，点击提交按钮，检查提示信息。

二、界面测试

1.布局是否合理，test box和按钮是否整齐。

2.testbox和按钮的长度，高度是否符合要求。

3. 界面的设计风格是否与UI的设计风格统一。

4. 界面中的文字简洁易懂，没有错别字。

三、性能测试

1.打开登录页面，需要的时间是否在需求要求的时间内。

2.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。

3.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。

四、安全性测试

1.登录成功后生成的Cookie，是否是http only (否则容易被脚本盗取)。

2.用户名和密码是否通过加密的方式，发送给Web服务器。

3.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。

4.用户名和密码的输入框，应该屏蔽SQL注入攻击。

5.用户名和密码的的输入框，应该禁止输入脚本 （防止XSS攻击）。

6.防止暴力破解，检测是否有错误登陆的次数限制。

7. 是否支持多用户在同一机器上登录。

8. 同一用户能否在多台机器上登录。

五、可用性测试

1. 是否可以全用键盘操作，是否有快捷键。

2. 输入用户名，密码后按回车，是否可以登陆。

3. 输入框能否可以以Tab键切换。

六、兼容性测试

1.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）。

2.同种浏览器不同版本下能否显示正常且功能正常。

2.不同的平台是否能正常工作，比如Windows, Mac。

3.移动设备上是否正常工作，比如Iphone, Andriod。

4.不同的分辨率下显示是否正常。

七、本地化测试

1. 不同语言环境下，页面的显示是否正确。

55 请对这个系统做出测试用例：一个系统，多个摄像头，抓拍车牌，识别车牌，上传网上，网上展示

功能：

1.每个摄像头都能抓拍车牌；

2.每个摄像头抓拍到的车牌能正常交给系统处理；

3.系统能够正确识别车牌；

4.系统能够将识别出的车牌上传；

5.上传至网络的车牌能够正常展示出来；

一、功能测试

1.使用正常的车牌，保持车牌静止，检查每个摄像头是否能抓拍车牌；

2.使用类似非车牌的写有字的纸板，检查每个摄像头是否抓拍；

3.使用正常的车牌，保持车牌较高速移动，检查每个摄像头是否能抓拍车牌；

4.在多种情况下检查每个摄像头抓拍到的车牌能否正常交给系统处理，如临时断电、断网后能否正常将数据交给系统；

5.使用抓拍到的正常的车牌，交由系统处理，检查系统能否识别车牌；

6.使用非车牌的其他图片，交由系统处理，检查系统能否识别；

7.在多种情况下检查系统能否将正常识别出的车牌进行上传，如临时断电、断网后未上传数据是否能继续上传；

8.构造非车牌的其他内容的数据，检查系统能否将异常内容进行上传；

9.检查上传至网络的车牌能否正常展示出来；

10.上传非车牌的其他内容的数据，检查能否正常显示出来。

二、性能测试

1.同时向一个摄像头展示多个静止的车牌，检查摄像头能否抓拍到多个车牌；

2.同时向一个摄像头展示多个较高速运动的车牌，检查摄像头能否抓拍到多个车牌；

3.抓拍后，检查系统识别车牌的时间是否在需求要求的时间内；

4.模拟大量抓拍照片同时交由系统处理，检查一定压力下系统能否正常识别车牌；

5.模拟大量车牌同时上传，检查一定压力下能否上传成功。

三、安全性测试

1.检查是否能够通过给车牌加装饰物等方法，使摄像头无法抓拍或抓拍后系统无法正常识别车牌

56 请你对吃鸡游戏进行压力测试

一.首先明确需要测试压力的内容:

1.游戏服务器硬件

a.硬盘I/o

b.内存

c.CPU

2.网络压力

a.长连接

a1.最大连接数

a2.流量(内网、外网、进、出)

b.长连接短周期(类似Http的TCP应用，这个比较特殊的一个需求，专门针Login Agent)

b1.每秒建立的连接数

b2.实际处理能力

3.数据库

a.每秒事务数

b.每秒锁等待数

c.平均延时(ms)

d.CPU暂用

4.多线程的最优线程数

a.数据库执行的多线程

b.多连接处理

二.Windows Server环境测试方式

1.服务器性能监测

使用Server自带的性能监测器设置各个进程的监测参数。Window的这个自动工具做的相当强大。大家自己摸一摸基本就会用了。每个参数都由详细的说明。

2.案例设计注意

a.对于数据库的性能测试上，现在由于所有的游戏服务器构架在DB前面都有一个实现DB缓冲功能的进程，以减少数据库频繁的读写操作。所以其实数据库的读是一个轻量级的数量；而数据库的写操作是一个周期性能过程。案例设计一定要能够驱动这种周期性能过程。比如我们游戏的战斗，导致游戏玩家数据的改变，或驱动所有在线玩家数据的周期性存储。

b.选择具有代表性，并且最频繁的游戏操作。用于进行最高用户在线的各种性能指标采集。如，开枪、道具拾取、道具使用、移动、聊天

c.聊天性能测试

广播聊天是最为考验游戏信息发送能力的功能。通过进行全局广播的压力测试。我们可以获取服务器进程发送信息到客户端的最高承载量。进而可以对我们的各种广播功能进行一个预估和频率限制。

d.同屏玩家的移动测试

移动+广播。这两种信息，基本是网络游戏流量的70-80%左右。同屏玩家数量，将会增加各种数据的广播需求，非常影响游戏性能。所以同屏的移动测试也是广播测试的一个必要环节。需要根据实际结果进行适当的优化。

e.大量玩家同时登录测试

玩家登录时，有大量的信息需要进行分配和初始化；同时也有大量的数据需要下传客户端。服务器需要进行大量的TCP连接建立。所以是一个比较关键的过程。这个测试案例是一个比较特殊，但是运营是肯定会碰到的案例。

f.由于线程池处理事务，随着事务的时耗，存在一个最优线程数的问题。过多的线程反而会降低服务器效率

3.细节问题

a.进行测试需要仔细思考客户端性能影响服务器最后表现的可能性。比如

a1.模拟客户端的性能无法有效处理服务器返回信息,可能就导致服务器发送的信息缓存在服务器系统缓存，从而表现出服务器内存不断增加。表现为服务器发送能力不足，其实可能根本就是客户端的性能问题

a2.客户端性能问题，导致发起的请求数过少，从而导致单位时间内服务器处理的请求过少。表现为服务器性能不足，其实根本就是客户端的请求能力不足。

b.网络带宽导致最后表现不足

b1.确认服务器的各个网卡，以及相互的带宽。不然可能因为相互带宽，导致服务器对于客户端请求的处理延时。表现为服务器卡机

b2.客户端模拟多个玩家，比如1000个玩家。而客户端的网卡或者客户端与服务器之间的中转服务器带宽过小，导致服务器数据发送不出，内存不断增加。表现为服务器发送能力不足，其实是中间带宽问题。

c.debug i/o导致服务器性能下降

c1.进行性能测试，一定要取消debug用的同步的i/o.比如我们服务器的debuginternalLog.同步i/o是非常影响性能的，特别在压力测试下可能导致每秒上千上万甚至几十万次的执行。一处的文件写入操作就可以导致几十万次的处理能力变成几千次的处理能力。

c2.客户端避免进行阻塞操作导致模拟多用户性能下降，导致服务器表现性能下降

d.流量需要区分内网网

内、外网流量在游戏正式运行时是完全分开的。价格也是完全不同的。一个千M的外网是一个无法想象的运营成本，而kmbps/s现在已经是一个可以接受的代价。游戏进程需要进行不同网卡的配置和绑定。确定内外网流量。

57 请你根据微信登录界面设计测试用例

一、功能测试

1.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。

2.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。

3.登录成功后能否能否跳转到正确的页面

4.检查能否选择不同登录方式进行登录，如使用手机号登录、使用微信号登录或扫码登录。

5.记住用户名的功能

6.登陆失败后，不能记录密码的功能

7.密码是否非明文显示显示，使用星号圆点等符号代替。

8.有验证码时，还要考虑文字是否扭曲过度导致辨认难度大，考虑颜色、刷新或换一个按钮是否好用

9.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

10.输入密码的时候，大写键盘开启的时候要有提示信息。

11.什么都不输入，点击提交按钮，检查提示信息。

二、界面测试

1.布局是否合理，testbox和按钮是否整齐。

2.testbox和按钮的长度，高度是否符合要求。

3. 界面的设计风格是否与UI的设计风格统一。

4. 界面中的文字简洁易懂，没有错别字。

三、性能测试

1.打开登录页面，需要的时间是否在需求要求的时间内。

2.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。

3.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。

四、安全性测试

1.登录成功后生成的Cookie，是否是httponly (否则容易被脚本盗取)。

2.用户名和密码是否通过加密的方式，发送给Web服务器。

3.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。

4.用户名和密码的输入框，应该屏蔽SQL注入攻击。

5.用户名和密码的的输入框，应该禁止输入脚本 （防止XSS攻击）。

6.防止暴力破解，检测是否有错误登陆的次数限制。

7.是否支持多用户在同一机器上登录。

8.同一用户能否在多台机器上登录。

五、兼容性测试

1.不同移动平台或PC环境下下能否显示正常且功能正常

2.同种平台下不同微信版本下能否显示正常且功能正常。

3.不同的分辨率下显示是否正常。

七、本地化测试

1. 不同语言环境下，页面的显示是否正确。

58 请你对朋友圈点赞功能进行测试

1.是否可以正常点赞和取消；

2.点赞的人是否在可见分组里；

3.点赞状态是否能即时更新显示；

4.点赞状态，共同好友是否可见；

6.性能检测，网速快慢对其影响；

7.点赞显示的是否正确，一行几个；

8.点赞是否按时间进行排序，头像对应的是否正确；

9.是否能在消息列表中显示点赞人的昵称、

10.不同手机，系统显示界面如何；备注；

11.可扩展性测试，点赞后是否能发表评论；

12.是否在未登录时可查看被点赞的信息。

59 如果做一个杯子的检测，你如何测试

1.功能

（1）水倒水杯容量的一半

（2）水倒规定的安全线

（4）水杯容量刻度与其他水杯一致

（5）盖子拧紧水倒不出来

（6）烫手验证

2.性能

（1）使用最大次数或时间

（2）掉地上不易损坏

（3）盖子拧到什么程度水倒不出来

（4）保温时间长

（5）杯子的耐热性

（6）杯子的耐寒性

（7）长时间放置水不会漏

（8）杯子上放置重物达到什么程度杯子会被损坏

3.界面

（1）外观完整、美观

（2）大小与设计一样（高、宽、容量、直径）

（3）拿着舒服

（4）材质与设计一样

（5）杯子上的图案掉落

（6）图案遇水溶解

4.安全

（1）杯子使用的材质毒或细菌的验证

（2）高温材质释放毒性

（3）低温材质释放毒性

5.易用性

（1）倒水方便

（2）喝水方便

（3）携带方便

（4）使用简单，容易操作

（5）防滑措施

6.兼容性

（1）杯子能够容纳果汁、白水、酒精、汽油等。

7.震动测试

（1）杯子加包装（有填充物），六面震动，检查产品是否能应对铁路/公路/航空运输。

8.可移植性

（1）杯子在不同地方、温度环境下都可以正常使用。

60 如何对一个页面进行测试

1、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。

2、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。

3、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？

4、兼容性测试

5、性能测试

61 如何对水壶进行测试

1.功能

（1）水倒水壶容量的一半

（2）水倒规定的安全线

（4）水壶容量刻度与其他水壶一致

（5）盖子拧紧水倒不出来

（6）烫手验证

2.性能

（1）使用最大次数或时间

（2）掉地上不易损坏

（3）盖子拧到什么程度水倒不出来

（4）保温时间长

（5）壶的耐热性

（6）壶的耐寒性

（7）长时间放置水不会漏

（8）壶上放置重物达到什么程度壶会被损坏

3.界面

（1）外观完整、美观

（2）大小与设计一样（高、宽、容量、直径）

（3）拿着舒服

（4）材质与设计一样

（5）壶上的图案掉落

（6）图案遇水溶解

4.安全

（1）壶使用的材质毒或细菌的验证

（2）高温材质释放毒性

（3）低温材质释放毒性

5.易用性

（1）倒水方便

（2）喝水方便

（3）携带方便

（4）使用简单，容易操作

（5）防滑措施

6.兼容性

（1）壶能够容纳果汁、白水、酒精、汽油等。

7.震动测试

（1）壶加包装（有填充物），六面震动，检查产品是否能应对铁路/公路/航空运输。

8.可移植性

（1）壶在不同地方、温度环境下都可以正常使用。

62 如何对淘宝搜索框进行测试

一功能测试：

1. 输入关键字，查看: 返回结果是否准确，返回的文本长度需限制

1.1输入可查到结果的正常关键字、词、语句，检索到的内容、链接正确性；

1.2输入不可查到结果的关键字、词、语句；

1.3输入一些特殊的内容，如空、特殊符、标点符、极限值等，可引入等价类划分的方法等；

2. 结果显示：标题，卖家，销售量，单行/多行，是否有图片

3. 结果排序：价格 销量 评价 综合

4.返回结果庞大时，限制第一页的显示量，需支持翻页

5.多选项搜索：关键字 品牌 产地 价格区间 是否天猫 是否全国购

6.是否支持模糊搜索，支持通配符的查询

7.网速慢的情况下的搜索

8.搜索结果为空的情况

9.未登录情况和登录情况下的搜索（登录情况下存储用户搜索的关键字/搜索习惯）

二.性能测试：

1压力测试：在不同并发用户数压力下的表现（评价指标如响应时间等）

2负载测试：看极限能承载多大的用户量同时正常使用

3稳定性测试：常规压力下能保持多久持续稳定运行

4内存测试：有无内存泄漏现象

5大数据量测试：如模拟从庞大的海量数据中搜索结果、或搜索出海量的结果后列示出来，看表现如何等等。

三. 易用性：交互界面的设计是否便于、易于使用

1依据不同的查询结果会有相关的人性化提示，查不到时告知？查到时统计条数并告知？有疑似输入条件错误时提示可能正确的输入项等等处理；

2查询出的结果罗列有序，如按点击率或其他排序规则，确保每次查询出的结果位置按规则列示方便定位，显示字体、字号、色彩便于识别等等；

3标题查询、全文检索、模糊查询、容错查询、多关键字组织查询（空格间格开）等实用的检索方式是否正常？

4输入搜索条件的控件风格设计、位置摆放是否醒目便于使用者注意到，有否快照等快捷查看方式等人性化设计？

四. 兼容性

1WINDOWS/LINUX/UNIX等各类操作系统下及各版本条件下的应用

2IE/FIREFOX/GOOGLE/360/QQ等各类浏览器下及各版本条件下、各种显示分辨率条件下的应用

3SQL/ORACLE/DB2/MYSQL等各类数据库存储情况下的兼容性测试

4简体中文、繁体中文、英文等各类语种软件平台下的兼容性测试

5IPHONE/IPAD、安卓等各类移动应用平台下的兼容性测试

6与各相关的监控程序的兼容性测试，如输入法、杀毒、监控、防火墙等工具同时使用

五. 安全性

1被删除、加密、授权的数据，不允许被SQL注入等攻击方式查出来的，是否有安全控制设计；

2录入一些数据库查询的保留字符，如单引号、%等等，造成查询SQL拼接出的语句产生漏洞，如可以查出所有数据等等，这方面要有一些黑客攻击的思想并引入一些工具和技术，如爬网等。

3通过白盒测试技术，检查一下在程序设计上是否存在安全方面的隐患；

4对涉及国家安全、法律禁止的内容是否进行了相关的过滤和控制；

63 如何对一瓶矿泉水进行测试

界面测试:查看外观是否美观

功能度：查看水瓶漏不漏；瓶中水能不能被喝到

安全性：瓶子的材质有没有毒或细菌

可靠性：从不同高度落下的损坏程度

可移植性：在不同的地方、温度等环境下是否都可以正常使用

兼容性：是否能够容纳果汁、白水、酒精、汽油等

易用性：是否烫手、是否有防滑措施、是否方便饮用

用户文档：使用手册是否对的用法、限制、使用条件等有详细描述

疲劳测试：将盛上水（案例一）放24小时检查泄漏时间和情况；盛上汽油（案例二）放24小时检查泄漏时间和情况等

压力测试：用根针并在针上面不断加重量，看压强多大时会穿透

跌落测试:测试在何种高度跌落会破坏水瓶

64 请你说一下jmeter

Jmeter：Apache JMeter是Apache组织开发的基于Java的压力测试工具。用于对软件做压力测试，它最初被设计用于Web应用测试，但后来扩展到其他测试领域。 它可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器等等。JMeter 可以用于对服务器、网络或对象模拟巨大的负载，来自不同压力类别下测试它们的强度和分析整体性能。另外，JMeter能够对应用程序做功能/回归测试，通过创建带有断言的脚本来验证你的程序返回了你期望的结果。为了最大限度的灵活性，JMeter允许使用正则表达式创建断言。

为什么使用Jmeter：

•    开源免费，基于Java编写，可集成到其他系统可拓展各个功能插件

•    支持接口测试，压力测试等多种功能，支持录制回放，入门简单

•    相较于自己编写框架或其他开源工具，有较为完善的UI界面，便于接口调试

•    多平台支持，可在Linux，Windows，Mac上运行

用例生成与导出：

Jmeter的用例格式为jmx文件，实际为xml格式，感兴趣可以学习下自己定制生成想要的jmx文件。

生成原则：

每个功能模块为一个独立的jmx文件。增加可维护性。（尽量不要将一个jmx文件放入太多功能，后期维护成本会很高。）

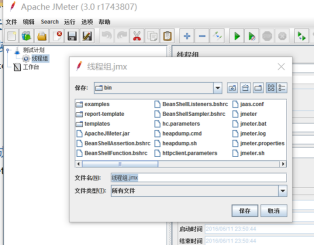
模块的私有变量保存在模块中，多模块共有的（例如服务器ip端口等）可以考虑存在单独的文件中读取。

接口测试不要放太多线程，毕竟不是做压力测试，意义也不大。

导出方法：

编写测试用例

文件——保存为——确定：



Jmeter运行模式及参数

GUI模式

打开已有的jmx文件（文件——打开）

点击启动按钮运行

命令行模式

依赖：

配置jmeter环境变量（windows下为将${jmeterhome}/bin加入Path变量）

如果未加入环境变量，在执行的时候可以直接给出全路径或在${jmeterhome}/bin下执行

命令：

jmeter -n -t <testplan filename> -l <listener filename>

参数：

-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 jtl文件 -> 记录结果的文件

-r 远程执行 -> 启动远程服务

-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

-j 日志文件->设置JMeter日志文件的名称

实例：

JMeter -n -t my\_test.jmx -l log.jtl -H my.proxy.server -P 8000

执行步骤：

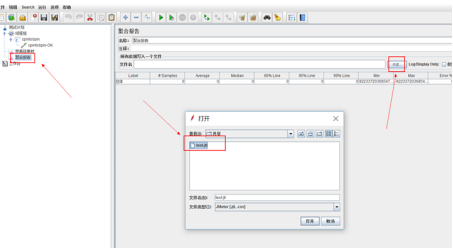
JMeter 默认去当前目录寻找脚本文件，并把日志记录在当前目录。比如你在 C:\tools\apache-jmeter-2.11\bin 目录下执行以上命令，JMeter 会去该目录下寻找 test.jmx 脚本并把执行结果放在该目录。如果你的脚本在其他目录，而且想要把执行结果放在另外文件夹，可以使用绝对路径告诉 JMeter。

执行结果查看：

GUI界面打开聚合报告

在GUI界面创建一个聚合报告

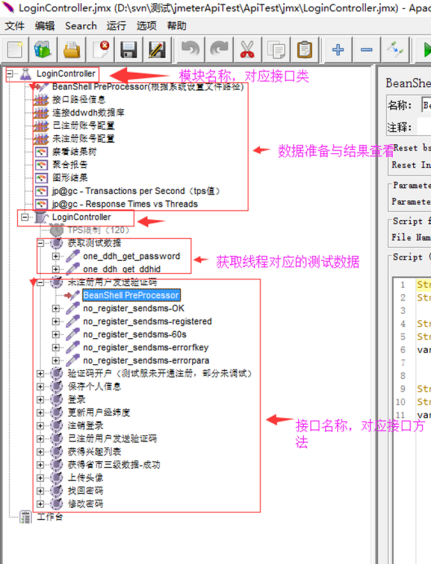
聚合报告界面点击浏览，选中生成的.jtl文件，打开



Jmeter使用

Jmeter创建接口测试计划实例

测试用例应该作为测试的基础内容，而用例的结构可能划分，则是用例的基础（忽然在这里想说一下，用例仅仅是一项测试活动的纲要，有最好，没有的话能保证质量也OK。更不用说用例的格式问题，无论是表格还是导图，其实都无所谓！本文的用例是指jmx文件中的控件结构）。



•    模块名称（测试计划）：每个模块独立划分为一个jmx文件（例如登陆模块），最好与接口类一一对应。对应的服务器信息，数据库信息等可存在这里。

•    数据准备：用于测试数据的准备（例如账号信息）。

•    结果查看：用于放置需要查看结果的控件（例如结果树）。

•    线程组：所有的接口测试用例放在线程组下，集中定义线程等信息

•    获取线程对应测试数据：用于获取针对独立线程的测试数据，例如在数据准备里面获得了账号信息，在这里根据账号信息去数据库获取对应的名称，ID等信息。

•    请求名称：用简单控制器为文件夹，内有不同的请求。简单控制器为一个独立的接口，不同请求对应不同的代码路径（例如成功请求，失败请求等）。建议请求名称最好用英文形式，否则后期持续集成或许会出现问题（no zuo no die！）。

•    在每条请求内放置正则匹配（用于应对需要返回值作为下次请求的参数的情况）以及断言。

65 请你来说一下购物车的测试用例

1.界面测试

•    界面布局、排版是否合理；文字是否显示清晰；不同卖家的商品是否区分明显。

2.功能测试

未登录时：

•    将商品加入购物车，页面跳转到登录页面，登录成功后购物车数量增加；

•    点击购物车菜单，页面跳转到登录页面。

登录后：

•    所有链接是否跳转正确；

•    商品是否可以成功加入购物车；

•    购物车商品总数是否有限制；

•    商品总数是否正确；

•    全选功能是否好用；

•    删除功能是否好用；

•    填写委托单功能是否好用；

•    委托单中填写的价格是否正确显示；

•    价格总计是否正确；

•    商品文字太长时是否显示完整；

•    店铺名字太长时是否显示完整；

•    创新券商品是否打标；

•    购物车中下架的商品是否有特殊标识；

•    新加入购物车商品排序（添加购物车中存在店铺的商品和购物车中不存在店铺的商品）；

•    是否支持TAB、ENTER等快捷键；

•    商品删除后商品总数是否减少；

•    购物车结算功能是否好用。

3.兼容性测试

•    不同浏览器测试。

4.易用性测试

•    删除功能是否有提示；是否有回到顶部的功能；商品过多时结算按钮是否可以浮动显示。

5.性能测试

•    压力测试；并发测试。

66 你写的测试程序是怎么样的，你写过前端、后端程序吗？

开发测试驱动程序一般分为4步：

1，指出需要的新特性。可以记录下来，然后为其编写一个测试。

2，编写特性的概要代码，这样程序就可以运行而没有任何语法等方面的错误，但是测试会失败。看到测试失败是很重要的，这样就能确定测试可以失败。如果测试代码中出现了错误，那么就有可能出现任何情况，测试都会成功，这样等于没测试任何东西。再强调一遍：在试图测试成功之前，先要看到它失败。

3，为特性的概要编写虚设代码，能满足测试要求就行。不用准确的实现功能，只要保证测试可以通过即可。这样一来就可以保证在开发的时候总是通过测试了，（除了第一次测试的时候）甚至在最初实现功能时亦是如此。

4，现在重写（或者重构）代码，这样它就会做自己应该做的事，从而保证测试一直成功。

在编码完成时，应该保证代码处于健康状态--不要遗留下任何测试失败。

67 请问你有没有写过web测试，怎么写的？

Web测试主要从下面几个大方向考虑

功能测试，主要做链接测试，表单测试，cookies测试，设计语言测试等

性能测试，考虑连接速度测试，以及负载测试，例如：Web应用系统能允许多少个用户同时在线？如果超过了这个数量，会出现什么现象？Web应用系统能否处理大量用户对同一个页面的请求？还有压力测试

可用性测试，比如导航测试，图形测试，内容测试，整体界面测试等

兼容性测试，市场上有很多不同的操作系统类型，最常见的有Windows、Unix、Macintosh、Linux等。Web应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置。这样，就可能会发生兼容性问题，同一个应用可能在某些操作系统下能正常运行，但在另外的操作系统下可能会运行失败。因此，在Web系统发布之前，需要在各种操作系统下对Web系统进行兼容性测试。

安全性测试，

（1）现在的Web应用系统基本采用先注册，后登陆的方式。因此，必须测试有效和无效的用户名和密码，要注意到是否大小写敏感，可以试多少次的限制，是否可以不登陆而直接浏览某个页面等。

（2）Web应用系统是否有超时的限制，也就是说，用户登陆后在一定时间内（例如15分钟）没有点击任何页面，是否需要重新登陆才能正常使用。

（3）为了保证Web应用系统的安全性，日志文件是至关重要的。需要测试相关信息是否写进了日志文件、是否可追踪。

（4）当使用了安全套接字时，还要测试加密是否正确，检查信息的完整性。

（5）服务器端的脚本常常构成安全漏洞，这些漏洞又常常被黑客利用。所以，还要测试没有经过授权，就不能在服务器端放置和编辑脚本的问题。

68 请问测试路由器怎么测，用命令行还是界面？

可以采用lperf这个命令

Lperf是一个网络性能测试工具，可以测量最大tcp和udp带宽，具有多种参数和特性，可以记录带宽，延迟抖动，数据包丢失，通过这些信息可以发现网络问题，检查网络质量，定位网络瓶颈。

iperf的使用非常简单，测试的原理是在wan口连接一台PC机，在LAN口连接一台PC，两边分别运行iperf服务端和客户端模式，用来测量LAN->WAN和WAN->LAN性能。具体命令如下：

服务端：iperf -s -w 1m  
客户端：iperf -c <server ip> -w 1m -t 20 -P 10

含义是TCP wndowsize 为1MByte，测试时间是20s，线程是10。

69 请你回答一下如何测试手机开机键？

功能测试： 按下开机键，屏幕能否亮起

性能测试：按下开机键，屏幕能否在规定时间内亮起

压力测试：连续多次按下开机键，观察屏幕是否能一直亮起，到多久时间失灵

健壮性测试：给定一个中了病毒的手机或者是淘汰许久的老机子，按下开机键观察屏幕能否亮起

可靠性测试：连续按下开机键有限次数，比如1万次，记录屏幕未亮起的次数

可用性测试：开机键按下费不费力，开机键的形状设计是否贴合手指，开机键的位置设计是否方便

70 请你回答一下性能测试有哪些指标，对一个登录功能做性能测试，有哪些指标，怎么测出可同时处理的最大请求数量

性能测试常用指标：

从外部看，主要有

1、吞吐量：每秒钟系统能够处理的请求数，任务数

2、响应时间：服务处理一个请求或一个任务的耗时

3、错误率：一批请求中结果出错的请求所占比例

从服务器的角度看，性能测试关注CPU，内存，服务器负载，网络，磁盘I/Ｏ

对登录功能做性能测试

单用户登陆的响应界面是否符合预期

单用户登陆时后台请求数量是否过多

高并发场景下用户登录的响应界面是否符合预期

高并发场景下服务端的监控指标是否符合预期

高集合点并发场景下是否存在资源死锁和不合理的资源等待

长时间大量用户连续登录和登出，服务器端是否存在内存泄漏

怎么测出可同时处理的最大请求数量

可以采用性能测试工具([WeTest服务器性能](https://link.zhihu.com/?target=http%3A/wetest.qq.com/gaps/%3Ffrom%3Dcontent_zhihu" \t "_blank))，该工具是腾讯wetest团队出品，使用起来很简单方便，但测试功能相当强大，能提供10w+以上的并发量，定位性能拐点，测出服务器模型最大并发

71 对于有系统大量并发访问，你会如何做测试，有什么建议

如何做高并发系统的测试，一般而言，整体的测试策略是：先针对部分系统进行性能测试及压力测试，得到各部分的峰值处理性能，**再模拟整体流程测试**，重点测试整体业务流程以及业务预期负荷，着重测试以下几点：

1、不同省份，不同运营商CDN节点性能，可采用典型压力测试方案

2、核心机房BGP网络带宽，此部分重点在于测试各运行商的BGP网络可靠性，实际速率，一般采用smokeping,lxChariot等工具

3、各类硬件设备性能，一般采用专业的网络设备测试工具

4、各类服务器并发性能，分布式处理能力，可采用压力测试方案工具

5、业务系统性能，采用业务系统压力测试方案

6、数据库处理性能，这部分需要结合业务系统进行测试，以获取核心业务场景下的数据库的TPS/QPS，

7、如果有支付功能，需要进行支付渠道接口及分流测试，此部分相对而言可能是最大的瓶颈所在，此外还涉及备份方案，容灾方案，业务降级方案的测试。

72 请你说一下多态

什么是多态？

多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

Java实现多态有三个必要条件：继承、重写、向上转型。

继承：在多态中必须存在有继承关系的子类和父类。

重写：子类对父类中某些方法进行重新定义，在调用这些方法时就会调用子类的方法。

向上转型：在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。

Java中有两种形式可以实现多态，继承和接口：

基于继承的实现机制主要表现在父类和继承该父类的一个或多个子类对某些方法的重写，多个子类对同一方法的重写可以表现出不同的行为。

基于接口的多态中，指向接口的引用必须是指定这实现了该接口的一个类的实例程序，在运行时，根据对象引用的实际类型来执行对应的方法。

java中产生死锁可能性的最根本原因是：1）是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环；2）默认的锁申请操作是阻塞的。

如，线程在获得一个锁L1的情况下再去申请另外一个锁L2，也就是锁L1想要包含了锁L2，在获得了锁L1，并且没有释放锁L1的情况下，又去申请获得锁L2，这个是产生死锁的最根本原因。

2、避免死锁：

•    方案一：破坏死锁的循环等待条件。

•    方法二：破坏死锁的请求与保持条件，使用lock的特性，为获取锁操作设置超时时间。这样不会死锁（至少不会无尽的死锁）

•    方法三：设置一个条件遍历与一个锁关联。该方法只用一把锁，没有chopstick类，将竞争从对筷子的争夺转换成了对状态的判断。仅当左右邻座都没有进餐时才可以进餐。提升了并发度。

3、linux中查看死锁进程状态

使用pstack 和 gdb 工具对死锁程序进行分析

pstack 进程号 查看各个线程的堆栈信息

当进程吊死的时候，多次使用，死锁的线程将一直处于等锁的状态，确定某些线程一直没有变化，一直处于等锁的状态。那么这些线程很可能是死锁了。如果怀疑哪些线程发生死锁了，可以采用gdb 进一步attach线程并进行分析。

执行命令gdb attach 进程号，进入gdb调试终端

运行：(gdb) info thread

73 请问你怎么理解多态，他有什么好处

所谓多态，就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行时确定，即一个引用变量到底会指向哪个类的实例对象，调用哪个类的实现方法，由程序运行期间才确定，这样不用修改程序源代码就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是[多态性](https://www.baidu.com/s?wd=%E5%A4%9A%E6%80%81%E6%80%A7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。

74 Python自动化测试框架有哪些？

Robot Framework

Robot Framework是最流行的Python自动化测试框架。它是完全用Python开发的，对于验收测试非常有用。这个框架可以运行在Java和.NET环境。它还支持跨平台，例如Windows、MacOS和Linux。这个产品是由世界上一些著名的测试人员创建的，拥有关键词驱动方案。它有如此多可用的工具和库，使得这个框架非常先进和健壮。

它是一个开源框架，并且留有API扩展空间。表格式的测试数据语法和关键词驱动测试使得它在全球的测试人员中非常流行。它无疑是最易用的自动化测试框架，并且允许你进行并行测试。

Pytest

如果你的项目比较小、复杂度比较低，Pytest是最适合的自动化测试平台。非常多的Python开发者都喜欢它，他们大部分用它来进行单元测试。它也具有Robot Framework所闻名的验收测试能力。Pytest最好的特性之一是，它提供了测试用例的详细失败信息，使得开发者可以快速准确地改正问题。它兼容最新版本的Python。它还兼容unittest、doctest和nose，开箱即用。它还有包含更多功能的插件和多样化的现有测试技术和测试用例。事实上，有300多个可用插件来自它的活跃社区。该平台设计用于编写更简单错误率更小的代码。你可以将Pytest与诸如Selenium和Splinter之类的图形用户界面一起使用，来使得测试工作更轻松。

75 unittest工作原理

unittest中最核心的四部分是：TestCase（测试案例），TestSuite（测试套件），TestRunner（运行器），TestFixture（准备和善后工作）

（1）一个TestCase的实例就是一个测试用例。测试用例就是指一个完整的测试流程，包括测试前准备环境的搭建（setUp），执行测试代码（run），以及测试后环境的还原（tearDown）。元测试（unit test）的本质也就在这里，一个测试用例是一个完整的测试单元，通过运行这个测试单元，可以对某一个问题进行验证。

（2）而多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。

（3）TestLoader是用来加载TestCase到TestSuite中的。

（4）TextTestRunner是来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的run(result)方法

（5）测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

综上，整个流程就是首先要写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中，整个过程集成在unittest.main模块中。

76 如何测试一个电梯

需求测试：查看电梯使用说明书、安全说明书等

界面测试：查看电梯外观

功能测试：

1.测试电梯能否实现正常的上升和下降功能。

2.电梯的按钮是否都可以使用。

3.电梯门的打开，关闭是否正常。

4.报警装置是否可用。

5.与其他电梯之间是否协作良好。

6.通风状况如何。

7.突然停电时的情况。

8.上升途中的响应。1）电梯本来在1楼，如果有人按18楼，那么电梯在上升到5楼的时候，有人按了10楼，这时候是否会在10楼先停下来

2）电梯下降到10层时显示满员，此时若8层有人等待电梯，是否在8层停。

9.是否有手机信号

可靠性测试：

1.门关上的一刹那出现障碍物。

2.同时按关门和开门按钮。

3.点击当前楼层号码

4.多次点击同一楼层号码

5.同时按上键和下键

易用性：电梯的按钮的设计符合一般人的习惯吗

用户文档：使用手册是否对电梯的用法、限制、使用条件等有详细的描述

压力测试：1.看电梯的最大承重量，在负载过重时报警装置是否有提醒

稳定性测试：看垫底在最大负载下平行运行的最长时间

77 web测试和APP测试的区别。

https://www.cnblogs.com/laoluoits/p/5673291.html

计算机网络

1.Http和Https的区别(应用层)

Http协议运行在TCP之上，明文传输，客户端与服务器端都无法验证对方的身份；Https是身披SSL(Secure Socket Layer)外壳的Http，运行于SSL上，SSL运行于TCP之上，是添加了加密和认证机制的HTTP。二者之间存在如下不同：

端口不同：Http与Https使用不同的连接方式，用的端口也不一样，前者是80，后者是443；

资源消耗：和HTTP通信相比，Https通信会由于加减密处理消耗更多的CPU和内存资源；

开销：Https通信需要CA机构颁发的SSL证书，Http无需证书；

Https的加密机制是一种共享密钥加密和公开密钥加密并用的混合加密机制。

**一、HTTP和HTTPS的基本概念**

HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

HTTPS：是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。

HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

**二、HTTP与HTTPS有什么区别？**

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

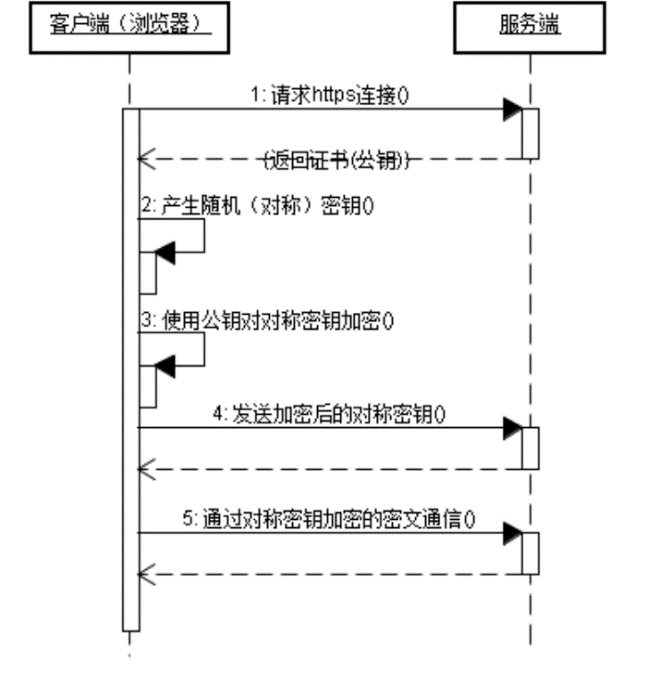
2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

2 HTTPS原理





SSL

SSL(Secure Sockets Layer[安全套接层](https://baike.baidu.com/item/%E5%AE%89%E5%85%A8%E5%A5%97%E6%8E%A5%E5%B1%82)),及其继任者[传输层安全](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82%E5%AE%89%E5%85%A8" \t "_blank)（Transport Layer Security，TLS）是为[网络通信](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1/9636548" \t "_blank)提供安全及[数据完整性](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AE%8C%E6%95%B4%E6%80%A7/110071)的一种安全协议。TLS与SSL在[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536" \t "_blank)对网络连接进行加密。SSL协议所采用的加密技术既有对称密钥加密技术，也有非对称密钥加密技术。

根据SSL安全强度，CA一般将SSL证书分为5种类型：

1 DVSSL：即域名型SSL安全证书，无需人工审核，快速颁发。

2 OVSSL：即企业型SSL安全证书，需要人工审核，证书包含企业认证信息。

3 EVSSL：即增强型SSL安全证书，人工审核严格，颁发之后可以使网站域名在浏览器地址栏变成绿色。

SSL的工作原理：

数据传输的机密性：利用对称密钥算法对传输的数据进行加密。

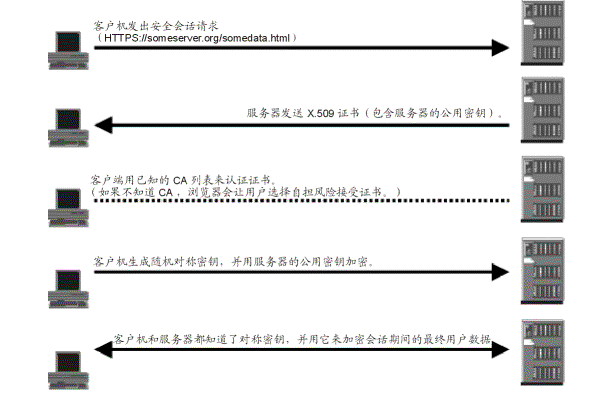
身份验证机制：基于证书利用数字签名方法对服务器和客户端进行身份验证，其中客户端的身份验证是可选的。

消息完整性验证：消息传输过程中使用MAC算法来检验消息的完整性。

3 对称加密与非对称加密

对称密钥加密是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方；而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。

由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，它非常的慢，所以我们还是要用对称加密来传送消息，但对称加密所使用的密钥我们可以通过非对称加密的方式发送出去。(非对称密钥算法可以用来实现数字签名)



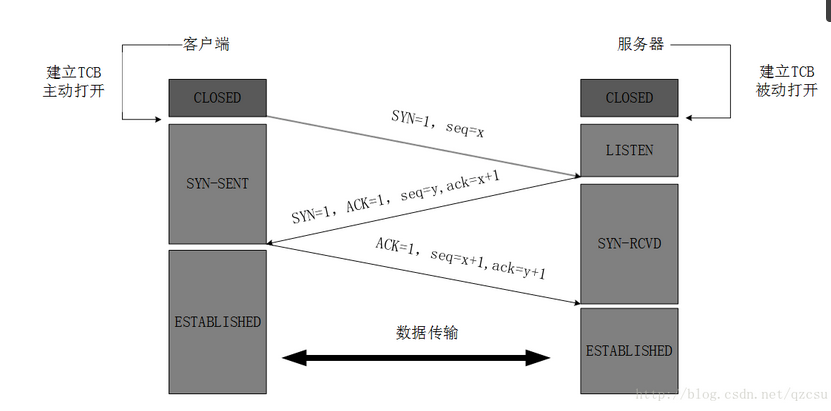
4 三次握手与四次挥手

(1). 三次握手(我要和你建立链接，你真的要和我建立链接么，我真的要和你建立链接，成功)：

第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。**同步位SYN=1，初始序号seq=x**

第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。**同步位SYN=1，确认号ACK=1，初始序号seq=y，确认号ack=x+1**

第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。**ACK=1，seq=x+1（上一个报文段的ack），ack = y+1（上一个报文段的seq+1，因为消耗了一个序号）**



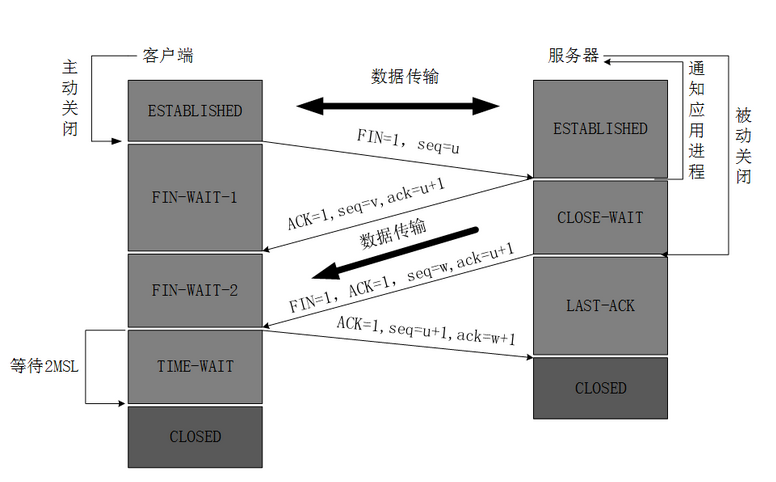
(2). 四次挥手(我要和你断开链接；好的，断吧。我也要和你断开链接；好的，断吧)：

第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。

第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。此时TCP链接处于半关闭状态，即客户端已经没有要发送的数据了，但服务端若发送数据，则客户端仍要接收。

第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。

第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。



什么是2MSL？MSL即Maximum Segment Lifetime，也就是报文最大生存时间

原因：其一，保证发送的ACK会成功发送到对方

其二，报文可能会被混淆，意思是说，其他时候的连接可能会被当作本次的连接。(主要防止已经失效的连接请求报文突然又传送到了服务器，从而产生错误。)

5 为什么TCP链接需要三次握手，两次不可以么，为什么？

为了防止已失效的链接请求报文突然又传送到了服务端，因而产生错误。

客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达Server。这是，Server误以为这是Client发出的一个新的链接请求，于是就向客户端发送确认数据包，同意建立链接。若不采用“三次握手”，那么只要Server发出确认数据包，新的链接就建立了。由于client此时并未发出建立链接的请求，所以其不会理睬Server的确认，也不与Server通信；而这时Server一直在等待Client的请求，这样Server就白白浪费了一定的资源。若采用“三次握手”，在这种情况下，由于Server端没有收到来自客户端的确认，则就会知道Client并没有要求建立请求，就不会建立链接。

6 TCP协议如何来保证传输的可靠性

TCP提供一种面向连接的、可靠的字节流服务。其中，面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个TCP连接。在一个TCP连接中，仅有两方进行彼此通信；而字节流服务意味着两个应用程序通过TCP链接交换8bit字节构成的字节流，TCP不在字节流中插入记录标识符。

对于可靠性，TCP通过以下方式进行保证：

数据包校验：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时TCP发送数据端超时后会重发数据；

对失序数据包重排序：既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。TCP将对失序数据进行重新排序，然后才交给应用层；

丢弃重复数据：对于重复数据，能够丢弃重复数据；

应答机制：当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒；

超时重发：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段；

流量控制：TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据，这可以防止较快主机致使较慢主机的缓冲区溢出，这就是流量控制。TCP使用的流量控制协议是可变大小的滑动窗口协议。

7 客户端不断进行请求链接会怎样？DDos(Distributed Denial of Service)攻击？

服务器端会为每个请求创建一个链接，并向其发送确认报文，然后等待客户端进行确认

1)、DDos 攻击

客户端向服务端发送请求链接数据包

服务端向客户端发送确认数据包

客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认

2)、DDos 预防 ( 没有彻底根治的办法，除非不使用TCP )

限制同时打开SYN半链接的数目

缩短SYN半链接的Time out 时间

关闭不必要的服务

DOS：中文名称是拒绝服务，一切能引起DOS行为的攻击都被称为DOS攻击。该攻击的效果是使得计算机或网络无法提供正常的服务。常见的DOS攻击有针对计算机网络带宽和连通性的攻击。 DOS是单机于单机之间的攻击。

DOS攻击的原理：首先攻击者向被攻击的服务器发送大量的虚假IP请求，被攻击者在收到请求后返回确认信息，等待攻击者进行确认，（此处需要拥有HTTP协议工作方式和TCP三次握手的基本知识）该过程需要TCP的三次握手，由于攻击者发送的请求信息是虚假的，所以服务器接收不到返回的确认信息，在一段时间内服务器会处与等待状态，而分配给这次请求的资源却被有被释放。当被攻击者等待一定的时间后，会因连接超时而断开，这时攻击者在次发送新的虚假信息请求，这样最终服务器资源被耗尽，直到瘫痪。

DDOS：中文名称是分布式拒绝服务攻击。指的是攻击者控制多台主机同时向同一主机或网络发起DOS攻击

解决：带宽资源充足 服务器的硬件配置 规范自身操作 防火墙的设置

8 Get与POST的区别

GET与POST是我们常用的两种HTTP Method，二者之间的区别主要包括如下五个方面：

(1). 从功能上讲，GET一般用来从服务器上获取资源，POST一般用来更新服务器上的资源；

(2). 从REST服务角度上说，GET是幂等的，即读取同一个资源，总是得到相同的数据，而POST不是幂等的，因为每次请求对资源的改变并不是相同的；进一步地，GET不会改变服务器上的资源，而POST会对服务器资源进行改变；

(3). 从请求参数形式上看，GET请求的数据会附在URL之后，即将请求数据放置在HTTP报文的 请求头 中，以?分割URL和传输数据，参数之间以&相连。特别地，如果数据是英文字母/数字，原样发送；否则，会将其编码为 application/x-www-form-urlencoded MIME 字符串(如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII)；而POST请求会把提交的数据则放置在是HTTP请求报文的 请求体 中。

(4). 就安全性而言，POST的安全性要比GET的安全性高，因为GET请求提交的数据将明文出现在URL上，而且POST请求参数则被包装到请求体中，相对更安全。

(5). 从请求的大小看，GET请求的长度受限于浏览器或服务器对URL长度的限制，允许发送的数据量比较小，而POST请求则是没有大小限制的。

1). GET请求中URL编码的意义

我们知道，在GET请求中会对URL中非西文字符进行编码，这样做的目的就是为了 避免歧义。看下面的例子，

针对“name1=value1&name2=value2”的例子，我们来谈一下数据从客户端到服务端的解析过程。首先，上述字符串在计算机中用ASCII吗表示为：

6E616D6531 3D 76616C756531 26 6E616D6532 3D 76616C756532

6E616D6531：name1

3D：=

76616C756531：value1

26：&

6E616D6532：name2

3D：=

76616C756532：value2

服务端在接收到该数据后就可以遍历该字节流，一个字节一个字节的吃，当吃到3D这字节后，服务端就知道前面吃得字节表示一个key，再往后吃，如果遇到26，说明从刚才吃的3D到26子节之间的是上一个key的value，以此类推就可以解析出客户端传过来的参数。

现在考虑这样一个问题，如果我们的参数值中就包含=或&这种特殊字符的时候该怎么办？比如，“name1=value1”，其中value1的值是“va&lu=e1”字符串，那么实际在传输过程中就会变成这样“name1=va&lu=e1”。这样，我们的本意是只有一个键值对，但是服务端却会解析成两个键值对，这样就产生了歧义。

那么，如何解决上述问题带来的歧义呢？解决的办法就是对参数进行URL编码：例如，我们对上述会产生歧义的字符进行URL编码后结果：“name1=va%26lu%3D”，这样服务端会把紧跟在“%”后的字节当成普通的字节，就是不会把它当成各个参数或键值对的分隔符。更多关于 URL编码 的内容，请参考我的博文《使用 URLDecoder 和 URLEncoder 对中文字符进行编码和解码》，此不赘述。

9 TCP与UDP的区别

TCP (Transmission Control Protocol)和UDP(User Datagram Protocol)协议属于传输层协议，它们之间的区别包括：

(1)TCP协议是面向连接的，而UDP是无连接的

(2)TCP协议是可靠传输的，但是UDP是不可靠传输。

(3)TCP协议所需资源多，UDP对系统要求较少。

(4)TCP有流量控制和拥塞控制，UDP没有

(5)TCP是一对一的连接，而UDP则可以支持一对一，多对多，一对多的通信。

(6)TCP面向的是字节流的服务，UDP面向的是报文的服务。

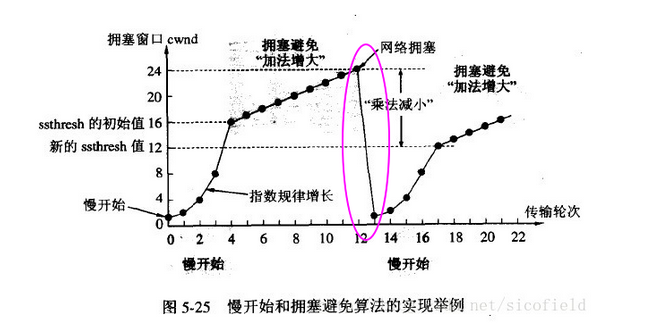
(7)UDP具有较好的实时性，工作效率比TCP高

10 TCP的拥塞处理

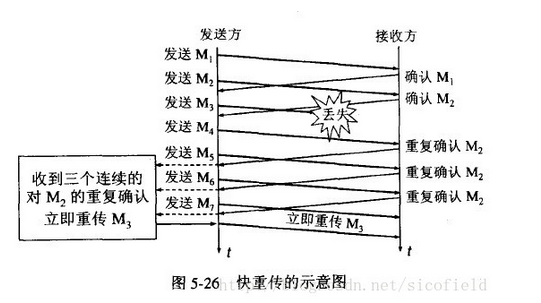
计算机网络中的带宽、交换结点中的缓存及处理机等都是网络的资源。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就会变坏，这种情况就叫做拥塞。拥塞控制就是 防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。注意，拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。拥塞控制的方法主要有以下四种：

1). 慢启动：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小;

2). 拥塞避免：拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍，这样拥塞窗口按线性规律缓慢增长。



3). 快重传：快重传要求接收方在收到一个 失序的报文段 后就立即发出 重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。



4). 快恢复：快重传配合使用的还有快恢复算法，当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半，但是接下去并不执行慢开始算法：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。



流量控制（flow control）所说的端到端（end to end）针对的是发送方和接收方速度不匹配的问题（比如经典的fast sender and slow receiver问题，接收方缓存大小与发送速率不匹配），提供一种速度匹配服务遏制发送速率使接收方应用程序的读取速率与之相适应。主要的方法有：

(1)Stop-and-wait 这个也是最简单粗暴的，发一个分组就等对方一个ACK回应然后再发，收不到ACK就不发了

(2)滑动窗口（包括go back N和选择重传） 接收方在一个窗口都满了以后才会发送ACK确认并要求发送下一个窗口。Go back N中接收方发送NAK表示接收前N-1个分组失败然后由发送方退回N步继续发送；选择重传双方都维护窗口，引入序列号，发现超时未收到序列号则由发送方重传。

流量控制主要针对的是端到端传输中控制流量大小并保证传输可靠性（未收到ack就不滑动）。

窗口滑动协议：该协议允许发送方在停止并等待确认前发送多个数据分组。由于发送方不必每发送一个分组就停下来等待确认，所以该协议可以加速数据的传输，提高网络吞吐量。

超时重传：超时重传是TCP协议保证数据可靠性的另一个重要机制，其原理是在发送某一个数据以后就开启一个计时器，在一定时间内如果没有得到发送的数据报的ACK[报文](https://baike.baidu.com/item/%E6%8A%A5%E6%96%87/3164352)，那么就重新发送数据，直到发送成功为止。

11 从输入网址到获得页面的过程

(1). 浏览器查询 DNS，获取域名对应的IP地址:具体过程包括浏览器搜索自身的DNS缓存、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析（此解析不具有权威性）。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询；

(2). 浏览器获得域名对应的IP地址以后，浏览器向服务器请求建立链接，发起三次握手；

(3). TCP/IP链接建立起来后，浏览器向服务器发送HTTP请求；

(4). 服务器接收到这个请求，并根据路径参数映射到特定的请求处理器进行处理，并将处理结果及相应的视图返回给浏览器；

(5). 浏览器解析并渲染视图，若遇到对js文件、css文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；

(6). 浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

用户输入URL -> 浏览器解析地址 -> DNS查找域名对应ip -> 服务器响应 ->浏览器拿到响应渲染

按OSI七层模型进行理解：每次都请求都会经过客户端的应用层（http协议）-->客户端的传输层（tcp或udp协议） -->客户端的网络层（ip协议）--> 客户端的链路层（网卡，路由器等）--> ------------------经过dns解析，穿越多个isp（互联网服务提供商，移动，联通，电信等），各种数据交换，找到了服务器------------------- 服务器的链路层 -->服务器的网络层 -->服务器的传输层 -->服务器的应用层。 这个请求完成了。

PING：通过发送回送请求报文和回送回答报文来检测源主机到目的主机的链路是否有问题，目的地是否可达，以及通信的延迟情况。

**二、Ping工作过程**

假定主机A的IP地址是192.168.1.1，主机B的IP地址是192.168.1.2，都在同一子网内，则当你在主机A上运行“Ping 192.168.1.2”后，都发生了些什么呢?

（1）Ping命令会构建一个固定格式的ICMP请求数据包，然后由ICMP协议将这个数据包连同地址“192.168.1.2”一起交给IP层协议 ping能够计算往返时间RTT，它在报文的数据部分插入发送时间。

（2）IP层协议将以地址“192.168.1.2”作为目的地址，本机IP地址作为源地址，加上一些其他的控制信息，构建一个IP数据包，并在一个映射表（ARP实现IP地址转成Mac地址的协议）中查找出IP地址192.168.1.2所对应的MAC地址（这是数据链路层协议构建数据链路层的传输单元——帧所必需的），一并交给数据链路层。

（3）数据链路层构建一个数据帧，目的地址是IP层传过来的MAC地址，源地址则是本机的物理地址，还要附加上一些控制信息，依据以太网的介质访问规则，将它们传送出去。

（4）主机B收到这个数据帧后，先检查它的目的地址，并和本机的物理地址对比，如符合，则接收；否则丢弃。接收后检查该数据帧，将IP数据包从帧中提取出来，交给本机的IP层协议。同样，IP层检查后，将有用的信息提取后交给ICMP协议，后者处理后，马上构建一个ICMP应答包，发送给主机A，其过程和主机A发送ICMP请求包到主机B一模一样。即先由IP地址，在网络层传输，然后再根据mac地址由数据链路层传送到目的主机

12 TCP重发机制

TCP协议是一个可靠的协议。它通过重新发送(retransmission)来实现TCP片段传输的可靠性。简单的说，TCP会不断重复发送TCP片段，直到片段被正确接收。

13 Session、Cookie 与 Application

cookies:是针对每一个网站的信息，每一个网站只对应一个，其它网站不能访问，这个文件是保存在客户端的，每次你打相应网站，浏览器会查找这个网站的 cookies，如果有就会将这个文件起发送出去。Cookies文件的内容大致包函这些信息如用户名，密码，设置等。——自动登陆

session： 是针对每一个用户的，只有客户机访问，程序就会为这个客户新增一session。session 里主要保存的是用户的登录信息，操作信息等。这个session在用户访问结束后会被自动消失（如果超时也会）。

Cookie和Session都是客户端与服务器之间保持状态的解决方案，具体来说，cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

(1). Cookie及其相关API

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie，而客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器，服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

(2). Session及其相关API

同样地，会话状态也可以保存在服务器端。客户端请求服务器，如果服务器记录该用户状态，就获取Session来保存状态，这时，如果服务器已经为此客户端创建过session，服务器就按照sessionid把这个session检索出来使用；如果客户端请求不包含sessionid，则为此客户端创建一个session并且生成一个与此session相关联的sessionid，并将这个sessionid在本次响应中返回给客户端保存。保存这个sessionid的方式可以采用 cookie机制 ，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器；若浏览器禁用Cookie的话，可以通过 URL重写机制 将sessionid传回服务器。

(3). Session 与 Cookie 的对比

实现机制：Session的实现常常依赖于Cookie机制，通过Cookie机制回传SessionID；

大小限制：Cookie有大小限制并且浏览器对每个站点也有cookie的个数限制，Session没有大小限制，理论上只与服务器的内存大小有关；

安全性：Cookie存在安全隐患，通过拦截或本地文件找得到cookie后可以进行攻击，而Session由于保存在服务器端，相对更加安全；

服务器资源消耗：Session是保存在服务器端上会存在一段时间才会消失，如果session过多会增加服务器的压力。

Application（ServletContext）：与一个Web应用程序相对应，为应用程序提供了一个全局的状态，所有客户都可以使用该状态。

14 SQL 注入

SQL注入就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

1). SQL注入攻击的总体思路

(1). 寻找到SQL注入的位置

(2). 判断服务器类型和后台数据库类型

(3). 针对不通的服务器和数据库特点进行SQL注入攻击

2). SQL注入攻击实例

比如，在一个登录界面，要求输入用户名和密码，可以这样输入实现免帐号登录：



用户一旦点击登录，如若没有做特殊处理，那么这个非法用户就很得意的登陆进去了。这是为什么呢?下面我们分析一下：从理论上说，后台认证程序中会有如下的SQL语句：String sql = “select \* from user\_table where username=’ “+userName+” ’ and password=’ “+password+” ‘”; 因此，当输入了上面的用户名和密码，上面的SQL语句变成：SELECT \* FROM user\_table WHERE username=’’or 1 = 1 – and password=’’。分析上述SQL语句我们知道，username=‘ or 1=1 这个语句一定会成功；然后后面加两个-，这意味着注释，它将后面的语句注释，让他们不起作用。这样，上述语句永远都能正确执行，用户轻易骗过系统，获取合法身份。

3). 应对方法

(1). 参数绑定

使用预编译手段，绑定参数是最好的防SQL注入的方法。目前许多的ORM框架及JDBC等都实现了SQL预编译和参数绑定功能，攻击者的恶意SQL会被当做SQL的参数而不是SQL命令被执行。在mybatis的mapper文件中，对于传递的参数我们一般是使用#和$来获取参数值。当使用#时，变量是占位符，就是一般我们使用javajdbc的PrepareStatement时的占位符，所有可以防止sql注入；当使用$时，变量就是直接追加在sql中，一般会有sql注入问题。

(2). 使用正则表达式过滤传入的参数

15 XSS 攻击

XSS是一种经常出现在web应用中的计算机安全漏洞，与SQL注入一起成为web中最主流的攻击方式。XSS是指恶意攻击者利用网站没有对用户提交数据进行转义处理或者过滤不足的缺点，进而添加一些脚本代码嵌入到web页面中去，使别的用户访问都会执行相应的嵌入代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。

1). XSS攻击的危害

盗取各类用户帐号，如机器登录帐号、用户网银帐号、各类管理员帐号

控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力

盗窃企业重要的具有商业价值的资料

非法转账

强制发送电子邮件

网站挂马

控制受害者机器向其它网站发起攻击

2). 原因解析

主要原因：过于信任客户端提交的数据！

解决办法：不信任任何客户端提交的数据，只要是客户端提交的数据就应该先进行相应的过滤处理然后方可进行下一步的操作。

进一步分析细节：客户端提交的数据本来就是应用所需要的，但是恶意攻击者利用网站对客户端提交数据的信任，在数据中插入一些符号以及javascript代码，那么这些数据将会成为应用代码中的一部分了，那么攻击者就可以肆无忌惮地展开攻击啦，因此我们绝不可以信任任何客户端提交的数据！！！

3). XSS 攻击分类

(1). 反射性XSS攻击 (非持久性XSS攻击)

漏洞产生的原因是攻击者注入的数据反映在响应中。一个典型的非持久性XSS攻击包含一个带XSS攻击向量的链接(即每次攻击需要用户的点击)，例如，正常发送消息：接收者将会接收信息并显示Hello,World；但是，非正常发送消息：接收者接收消息显示的时候将会弹出警告窗口！

(2). 持久性XSS攻击 (留言板场景)

XSS攻击向量(一般指XSS攻击代码)存储在网站数据库，当一个页面被用户打开的时候执行。也就是说，每当用户使用浏览器打开指定页面时，脚本便执行。与非持久性XSS攻击相比，持久性XSS攻击危害性更大。从名字就可以了解到，持久性XSS攻击就是将攻击代码存入数据库中，然后客户端打开时就执行这些攻击代码。

正常操作流程是：用户是提交相应留言信息 —— 将数据存储到数据库 —— 其他用户访问留言板，应用去数据并显示；而非正常操作流程是攻击者在value填写:并将数据提交、存储到数据库中；当其他用户取出数据显示的时候，将会执行这些攻击性代码。

4). 修复漏洞方针

漏洞产生的根本原因是 太相信用户提交的数据，对用户所提交的数据过滤不足所导致的，因此解决方案也应该从这个方面入手，具体方案包括：

将重要的cookie标记为http only, 这样的话Javascript 中的document.cookie语句就不能获取到cookie了（如果在cookie中设置了HttpOnly属性，那么通过js脚本将无法读取到cookie信息，这样能有效的防止XSS攻击）；

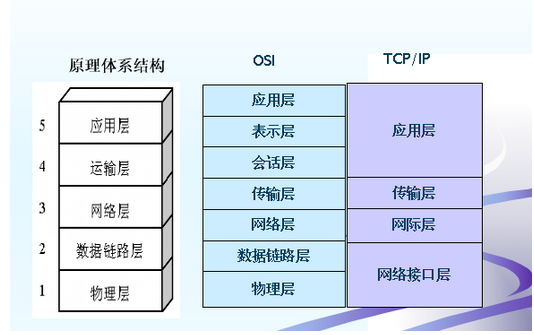
表单数据规定值的类型，例如：年龄应为只能为int、name只能为字母数字组合.

对数据进行Html Encode 处理过滤或移除特殊的Html标签，例如: <script>, <iframe> , < for <, > for>, &quot for

过滤JavaScript 事件的标签，例如 “οnclick=”, “onfocus” 等等。

需要注意的是，在有些应用中是允许html标签出现的，甚至是javascript代码出现。因此，我们在过滤数据的时候需要仔细分析哪些数据是有特殊要求（例如输出需要html代码、javascript代码拼接、或者此表单直接允许使用等等），然后区别处理！

16 OSI网络体系结构与TCP/IP协议模型



1). 物理层

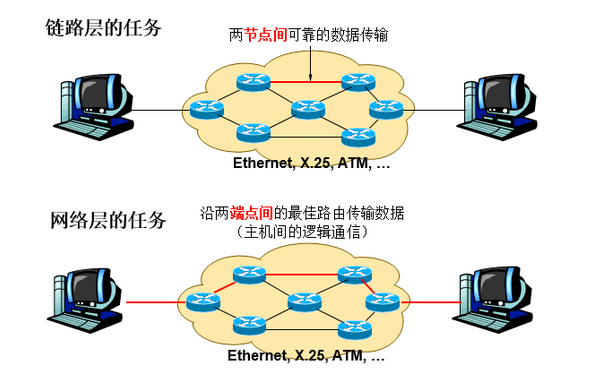
参考模型的最低层，也是OSI模型的第一层，实现了相邻计算机节点之间比特流的透明传送，并尽可能地屏蔽掉具体传输介质和物理设备的差异，使其上层(数据链路层)不必关心网络的具体传输介质。

2). 数据链路层（data link layer）

接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层。这一层在物理层提供的比特流的基础上，通过差错控制、流量控制方法，使有差错的物理线路变为无差错的数据链路，即提供可靠的通过物理介质传输数据的方法。

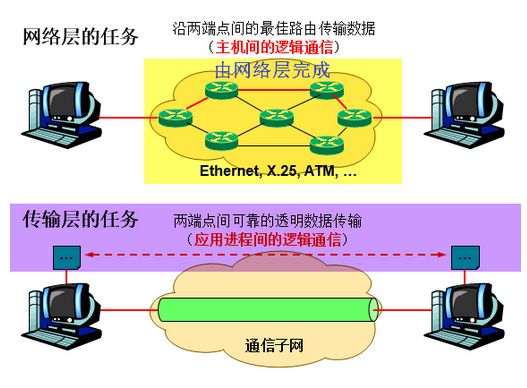
3). 网络层

将网络地址翻译成对应的物理地址，并通过路由选择算法为分组通过通信子网选择最适当的路径。



4). 传输层（transport layer）

在源端与目的端之间提供可靠的透明数据传输，使上层服务用户不必关系通信子网的实现细节。在协议栈中，传输层位于网络层之上，传输层协议为不同主机上运行的进程提供逻辑通信，而网络层协议为不同主机提供逻辑通信，如下图所示。



实际上，网络层可以看作是传输层的一部分，其为传输层提供服务。但对于终端系统而言，网络层对它们而言是透明的，它们知道传输层的存在，也就是说，在逻辑上它们认为是传输层为它们提供了端对端的通信，这也是分层思想的妙处。

5). 会话层（Session Layer）

会话层是OSI模型的第五层，是用户应用程序和网络之间的接口，负责在网络中的两节点之间建立、维持和终止通信。

6). 表示层（Presentation Layer）：数据的编码，压缩和解压缩，数据的加密和解密

表示层是OSI模型的第六层，它对来自应用层的命令和数据进行解释，以确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。

7). 应用层（Application layer）：为用户的应用进程提供网络通信服务

17 TCP和UDP分别对应的常见应用层协议

1). TCP对应的应用层协议

FTP：定义了文件传输协议，使用21端口。常说某某计算机开了FTP服务便是启动了文件传输服务。下载文件，上传主页，都要用到FTP服务。

Telnet：它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于DOS模式下的通信服务。如以前的BBS是-纯字符界面的，支持BBS的服务器将23端口打开，对外提供服务。

SMTP：定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置-中常看到有这么SMTP端口设置这个栏，服务器开放的是25号端口。

POP3：它是和SMTP对应，POP3用于接收邮件。通常情况下，POP3协议所用的是110端口。也是说，只要你有相应的使用POP3协议的程序（例如Fo-xmail或Outlook），就可以不以Web方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是163邮箱就没有必要先进入网易网站，再进入自己的邮-箱来收信）。

HTTP：从Web服务器传输超文本到本地浏览器的传送协议。

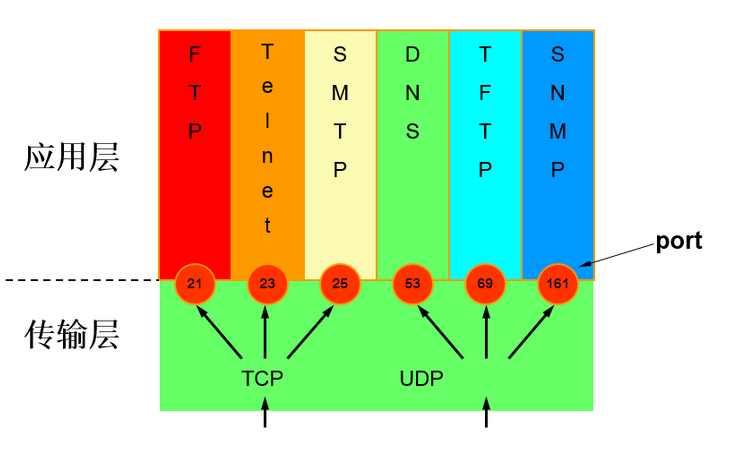
2). UDP对应的应用层协议

DNS：用于域名解析服务，将域名地址转换为IP地址。DNS用的是53号端口。

SNMP：简单网络管理协议，使用161号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

TFTP(Trival File Transfer Protocal)：简单文件传输协议，该协议在熟知端口69上使用UDP服务。

3). 图示



18 网络层的ARP（地址解析协议）协议工作原理

网络层的ARP协议完成了IP地址与物理地址的映射。首先，每台主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址的对应关系。当源主机需要将一个数据包要发送到目的主机时，会首先检查自己ARP列表中是否存在该IP地址对应的MAC地址：如果有，就直接将数据包发送到这个MAC地址；如果没有，就向本地网段发起一个ARP请求的广播包，查询此目的主机对应的MAC地址。此ARP请求数据包里包括源主机的IP地址、硬件地址、以及目的主机的IP地址。网络中所有的主机收到这个ARP请求后，会检查数据包中的目的IP是否和自己的IP地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的MAC地址和IP地址添加到自己的ARP列表中，如果ARP表中已经存在该IP的信息，则将其覆盖，然后给源主机发送一个ARP响应数据包，告诉对方自己是它需要查找的MAC地址；源主机收到这个ARP响应数据包后，将得到的目的主机的IP地址和MAC地址添加到自己的ARP列表中，并利用此信息开始数据的传输。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

**生存时间：**ARP对每一个映射地址的项目都设有生存时间（例如10-20分钟），这是为了防止该局域网中某些主机的硬件地址发生改变的情况。

19 IP地址的分类

IP地址是指互联网协议地址，是IP协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。IP地址编址方案将IP地址空间划分为A、B、C、D、E五类，其中A、B、C是基本类，D、E类作为多播和保留使用，为特殊地址。

每个IP地址包括两个标识码（ID），即网络ID和主机ID。同一个物理网络上的所有主机都使用同一个网络ID，网络上的一个主机（包括网络上工作站，服务器和路由器等）有一个主机ID与其对应。A~E类地址的特点如下：

A类地址：以0开头，第一个字节范围：0~127；

B类地址：以10开头，第一个字节范围：128~191；

C类地址：以110开头，第一个字节范围：192~223；

D类地址：以1110开头，第一个字节范围为224~239

E类地址：以1111开头，保留地

1). A类地址：1字节的网络地址 + 3字节主机地址，网络地址的最高位必须是“0”

一个A类IP地址是指， 在IP地址的四段号码中，第一段号码为网络号码，剩下的三段号码为本地计算机的号码。如果用二进制表示IP地址的话，A类IP地址就由1字节的网络地址和3字节主机地址组成，网络地址的最高位必须是“0”。A类IP地址中网络的标识长度为8位，主机标识的长度为24位，A类网络地址数量较少，有126个网络，每个网络可以容纳主机数达1600多万台。

A类IP地址的地址范围1.0.0.0到127.255.255.255（二进制表示为：00000001 00000000 00000000 00000000 - 01111110 11111111 11111111 11111111），最后一个是广播地址。A类IP地址的子网掩码为255.0.0.0，每个网络支持的最大主机数为256的3次方-2=16777214台。

2). B类地址: 2字节的网络地址 + 2字节主机地址，网络地址的最高位必须是“10”

一个B类IP地址是指，在IP地址的四段号码中，前两段号码为网络号码。如果用二进制表示IP地址的话，B类IP地址就由2字节的网络地址和2字节主机地址组成，网络地址的最高位必须是“10”。B类IP地址中网络的标识长度为16位，主机标识的长度为16位，B类网络地址适用于中等规模的网络，有16384个网络，每个网络所能容纳的计算机数为6万多台。

B类IP地址地址范围128.0.0.0-191.255.255.255（二进制表示为：10000000 00000000 00000000 00000000—-10111111 11111111 11111111 11111111），最后一个是广播地址。B类IP地址的子网掩码为255.255.0.0，每个网络支持的最大主机数为256的2次方-2=65534台。

3). C类地址: 3字节的网络地址 + 1字节主机地址，网络地址的最高位必须是“110”

一个C类IP地址是指，在IP地址的四段号码中，前三段号码为网络号码，剩下的一段号码为本地计算机的号码。如果用二进制表示IP地址的话，C类IP地址就由3字节的网络地址和1字节主机地址组成，网络地址的最高位必须是“110”。C类IP地址中网络的标识长度为24位，主机标识的长度为8位，C类网络地址数量较多，有209万余个网络。适用于小规模的局域网络，每个网络最多只能包含254台计算机。

C类IP地址范围192.0.0.0-223.255.255.255（二进制表示为: 11000000 00000000 00000000 00000000 - 11011111 11111111 11111111 11111111）。C类IP地址的子网掩码为255.255.255.0，每个网络支持的最大主机数为256-2=254台。

4). D类地址:多播地址，用于1对多通信，最高位必须是“1110”

D类IP地址在历史上被叫做多播地址(multicast address)，即组播地址。在以太网中，多播地址命名了一组应该在这个网络中应用接收到一个分组的站点。多播地址的最高位必须是“1110”，范围从224.0.0.0到239.255.255.255。

5). E类地址:为保留地址，最高位必须是“1111”

18、IP地址与物理地址

物理地址是数据链路层和物理层使用的地址，IP地址是网络层和以上各层使用的地址，是一种逻辑地址，其中ARP协议用于IP地址与物理地址的对应。

20 简述ICMP协议和IGMP协议。

**ICMP：**

* Internet Control Message Protocol，网际控制报文协议。
* **功能：** 为了更加有效的转发IP数据报和提高交付成功的机会，ICMP允许主机或路由器报告差错情况和提供有关异常情况的报告。

**IGMP：**

* Internet Group Management Protocol，网际组管理协议。
* **功能：** IGMP协议是让连接在本局域网上的多播路由器知道本局域网上是否有主机（严格来讲是主机上的某个进程）参加或者退出了某个多播组。

21 HTTP请求方法,请求头,响应头，响应状态码

1. HTTP请求头

accept：浏览器通过这个头告诉服务器，它所支持的数据类型。如：text/html, image/jpeg  
accept-Charset：浏览器通过这个头告诉服务器，它支持哪种字符集。  
accept-encoding：浏览器通过这个头告诉服务器，它支持哪种压缩格式。  
accept-language：浏览器通过这个头告诉服务器，它的语言环境。  
host：浏览器通过这个头告诉服务器，它想访问哪台主机。  
if-modified-since：浏览器通过这个头告诉服务器，缓存数据的时间  
referer：浏览器通过这个头告诉服务器，客户机是哪个页面来的(防盗链)。  
Connection：浏览器通过这个头告诉服务器，请求完后是断开链接还是维持链接。

2. HTTP响应头

location：服务器通过这个头告诉浏览器跳到哪里。  
server：服务器通过这个头告诉浏览器服务器的型号。  
content-encoding：服务器通过这个头告诉浏览器数据的压缩格式。  
content-length：服务器通过这个头告诉浏览器回送数据的长度。  
content-language：服务器通过这个头告诉浏览器语言环境。  
content-type：服务器通过这个头告诉浏览器回送数据的类型。  
refresh：服务器通过这个头告诉浏览器定时刷新。  
content-disposition：服务器通过这个头告诉浏览器以下载方式打开数据。  
transfer-encoding：服务器通过这个头告诉浏览器数据是以分块方式回送的  
以下三个表示服务器通过这个头告诉浏览器不要缓存  
expires：-1  
cache-control：no-cache  
pragma：no-cache

3. HTTP响应状态码

100-199：表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程  
200-299：表示成功接收请求并已完成整个处理过程，常用200  
300-399：未完成请求，客户需进一步细化请求，常用302,307,304  
400-499：客户端的请求有错误，常用404  
500-599：服务器端出现错误，常用500

1×× : 请求处理中，请求已被接受，正在处理

2×× : 请求成功，请求被成功处理

200 OK

3×× : 重定向，要完成请求必须进行进一步处理

301 : 永久性转移

302 ：暂时性转移

304 ： 已缓存

4×× : 客户端错误，请求不合法

400：Bad Request,请求有语法问题

403：拒绝请求

404：客户端所访问的页面不存在

5×× : 服务器端错误，服务器不能处理合法请求

500 ：服务器内部错误

503 ：服务不可用，稍等

4. HTTP请求方法

|  |  |  |
| --- | --- | --- |
| 1 | GET | 请求指定的页面信息，并返回实体主体。 |
| 2 | HEAD | 类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| 3 | POST | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 |
| 4 | PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| 5 | DELETE | 请求服务器删除指定的页面。 |
| 6 | CONNECT | HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。 |
| 7 | OPTIONS | 允许客户端查看服务器的性能。 |
| 8 | TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |
| 9 | PATCH | 是对 PUT 方法的补充，用来对已知资源进行局部更新 。 |

说完原理性的问题，我们从表面上来看看GET和POST的区别：

1. get是从服务器上获取数据，post是向服务器传送数据。

get 和 post只是一种传递数据的方式，get也可以把数据传到服务器，他们的本质都是发送请求和接收结果。只是组织格式和数据量上面有差别，http协议里面有介绍

2. get是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTP post机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。 因为get设计成传输小数据，而且最好是不修改服务器的数据，所以浏览器一般都在地址栏里面可以看到，但post一般都用来传递大数据，或比较隐私的数据，所以在地址栏看不到，能不能看到不是协议规定，是浏览器规定的。

3. 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。

没明白，怎么获得变量和你的服务器有关，和get或post无关，服务器都对这些请求做了封装

4. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80KB，IIS5中为100KB。 post基本没有限制，我想大家都上传过文件，都是用post方式的。只不过要修改form里面的那个type参数

5. get安全性非常低，post安全性较高。 如果没有加密，他们安全级别都是一样的，随便一个监听器都可以把所有的数据监听到。

22 http请求报文

一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成，下图给出了请求报文的一般格式。

**1 请求行**

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如，GET /index.html HTTP/1.1。

**2.请求头部**

请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端请求的信息，

**3.空行**

最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。

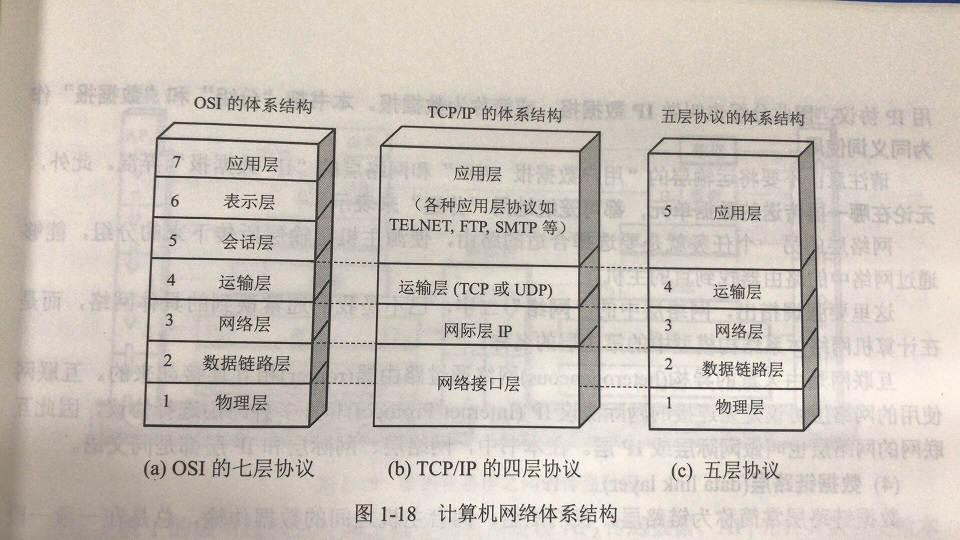
**4.请求数据**

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

23 socket有什么作用

Socket利用网间网通信设施实现进程通信,但它对通信设施的细节毫不关心,只要通信设施能提供足够的通信能力,它就满足了。至此,我们对Socket进行了直观的描述。抽象出来,Socket实质上提供了进程通信的端点。进程通信之前,双方首先必须各自创建一个端点,否则是没有办法建立联系并相互通信的。正如打电话之前,双方必须各自拥有一台电话机一样。

24 OSI、TCP/IP、五层协议体系结构是什么样的？每层的作用是什么？每层的协议有哪些？使用了哪些硬件设备？



**OSI七层协议体系：** 概念清楚，理论也比较完整，但是它既复杂也不实用。

物理层：

* + 计算机网络体系中最底层，为所有网络/数据数据通信提供物理的通信线路。另外通信线路可以通过信道复用方式在一条物理线路中划分多条信道。默认情况下，一条物理线路就是一条信道。
  + **常用协议：** ISO2110，IEEE802，IEEE802.2
  + **常用硬件：** 中继器、集线器、还有我们通常说的双绞线也工作在物理层
* 数据链路层：
  + 为同一局域网内部的网络/数据通信提供点到点的数据传输通道，通过MAC地址寻址把数据传输到目的节点。
  + 数据链路层的通信只能在同一网段进行，要在不同网段间进行，必须依靠网络层和传输层。
  + 数据链路层提供的不是物理线路，而是在物理层的物理线路基础之上，通过数据链路层协议构建的虚拟数据传输通道，并且只能在同一段内进行数据转发。
  + **常用协议：**SLIP，CSLIP，PPP，ARP，RARP，MTU。**（在OSI模型中ARP协议属于链路层；而在TCP/IP模型中，ARP协议属于网络层。）**
  + **常用硬件：**网桥(网桥（Bridge）是一个局域网与另一个局域网之间建立连接的桥梁。)、以太网交换机、网卡（其实网卡是一半工作在物理层、一半工作在数据链路层,网卡是电脑和[网络连接](http://www.so.com/s?q=%E7%BD%91%E7%BB%9C%E8%BF%9E%E6%8E%A5&ie=utf-8&src=internal_wenda_recommend_textn)的[接口](http://www.so.com/s?q=%E6%8E%A5%E5%8F%A3&ie=utf-8&src=internal_wenda_recommend_textn)）
* 网络层：
  + 为不同网段之间的数据转发提供路径选择，通过IP地址（也有可能是其他网络地址）把数据包转发到目的节点。网络层这种寻址功能我们称为“路由寻址”，就是选择哪一条路径到达下一个节点。
  + 网络层的功能仅起到不同网络间转发数据包的功能，最终数据还是要在目的网络的数据链路层进行传输，在到达下一个节点设备（如路由器）再进行转发。
  + **常用协议：**IP，ICMP，RIP，OSPF，BGP，IGMP
  + **常用硬件：**路由器(路由器就是连接两个以上各别网络的设备)
* 运输层：
  + 是在下面三层构建的网络平台基础上专门为通信双方构建端到端（不是点到点）的数据传输通道，使通信双方就像在直接进行数据传输一样。这个端到端的传输通道是可以跨网络的。
  + **常用协议：** TCP，UDP
  + **常用硬件：**网关(网关既可以用于广域网互连，也可以用于局域网互连。)
* 会话层：
  + 为具体的用户建立会话进程。
  + **常用协议：** 无
  + **常用硬件：**网关
* 表示层：
  + 是对用户网络应用数据的具体解释，包括在网络通信中可采用的信息技术、可采用的加密方式等。
  + **常用协议：** 无
  + **常用硬件：**网关
* 应用层：
  + 用户进行具体网络应用的层次，是具体网络应用的体现者，负责接受用户的各种网络应用进程的调用。
  + **常用协议：** TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet
  + **常用硬件：**网关

25 路由器、中继器、集线器、交换机、网关、网桥各自的概念以及作用？

* 路由器（Router）：
  + 工作范围：网络层，不同网络之间
  + 功能：是一种计算机网络设备，提供了**路由与转送**两种重要机制，可以决定数据包从来源端到目的端所经过的路由路径（host到host之间的传输路径），这个过程称为路由；将路由器输入端的数据包移送至适当的路由器输出端(在路由器内部进行)，这称为转送。路由工作在OSI模型的第三层——即网络层，例如网际协议。
* 中继器：
  + 工作范围：物理层
  + 功能：中继器是将一条电缆传输过来的信号经过放大和调整以后再传输给另一个电缆。因此，它是物理层上的网络延长设备。一般而言，中继器连接的双方是相同的物理介质，但是有些也可以完成不同媒介之间的连接。但是中继器只是完成了信号的转换，而不能完成转发。也就是说，中继器不能再传输速度不同的介质之间完成转发。
* 集线器：
  + 工作范围：物理层
  + 功能：如果每个设备只有一个对外接口，那么意味着只能建立一对一的点到点的通信。为了能够让通信“一对多”，需要将信号复制广播，于是，产生了集线器。集线器的主要功能是对接收到的信号进行再生整形放大，以扩大网络的传输距离，同时把所有节点集中在以它为中心的节点上。
* 交换机：
  + 工作范围：数据链路层
  + 功能：交换机内部的CPU会在每个端口成功连接时，通过ARP协议学习它的MAC地址，保存成一张ARP表。在今后的通讯中，发往该MAC地址的数据包将仅送往其对应的端口，而不是所有的端口。也就是说，交换机也有一张表，记录的是port-mac。按照传输数据包的层次，交换机可以分为一、二.....七层交换机。
* 网桥：
  + 工作范围：数据链路层
  + 功能：网桥能够连接两个不同的数据链路，即使他们的传输速度不同。它能够识别数据链路中的数据帧，并将这些数据帧临时存储在内存中，然后在重新生成一个新的数据帧转发给另外一个网段。网桥还具有检错功能。因此，网桥有时候也会被称为2层交换机。（数据链路层设备）有些网桥能够记住曾今通过自己转发的数据帧的MAC地址，并保存在内部的内存表中。这种网桥被称为自学式网桥。
* 网卡：
  + 工作范围：物理层、数据链路层
  + 功能：又称**网络接口控制器**，**网络适配器**（network adapter），**网卡**（network interface card），或**局域网接收器**（LAN adapter），是一块被设计用来允许计算机在计算机网络上进行通讯的计算机硬件。网卡的作用是负责接收网络上的数据包，通过和自己本身的物理地址相比较决定是否为本机应接信息，解包后将数据通过主板上的总线传输给本地计算机，另一方面将本地计算机上的数据打包后送出网络。
    - 实现与主机总线的网络通讯连接，网卡相当于桥梁设备
    - 实现主机与介质之间的电信号匹配
    - 提供数据缓冲能力
* 网关：
  + 工作范围：运输层、应用层
  + 功能：网关（Gateway）负责第三层（网络层）以上的数据中继，实现不同体系结构的网络协议转换，它通常采用软件的方法实现，并且与特定的应用服务一一对应。比如：OSI的文件传输服务FTAM和TCP/IP的文件传输服务FTP，尽管二者都是文件传输但是由于所执行的协议不同不能直接进行通信，而需要网关将两个文件传输系统互连，达到相互进行文件传输的目的。

26 端口以及对应的服务？

**端口号分类：**

* 服务器端使用的端口号。

**熟知端口号**：0~1023。IANA把一些端口号指派给了TCP/IP最重要的一些程序。当有一些新的应用程序出现后，IANA必须给它指定 一个熟知端口号，否则互联网上的其他应用进程无法和它进行通信。

| **应用程序** | **FTP** | **TELNET** | **SMTP** | **DNS** | **TFTP** | **HTTP** | **SNMP** | **SNMP(trap)** | **HTTPS** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 熟知端口号 | 21 | 23 | 25 | 53 | 69 | 80 | 161 | 162 | 443 |

**登记端口号：**1024~49151。这类端口号是为没有熟知端口号的应用程序使用。这类端口号必须在IANA中按照规定的手续进行登记，以防止重复。

* 客户端使用的端口号：49152~65535。只有在客户端进程运行时才动态选择，因此又叫短暂端口号。这类端口号留给客户进程选择暂时使用，通信结束后就释放，不再占用。

操作系统

1 进程和线程以及它们的区别

https://blog.csdn.net/weixin\_40271838/article/details/79998327

* 地址空间：同一进程的线程共享本进程的地址空间，而进程之间则是独立的地址空间。
* 资源拥有：同一进程内的线程共享本进程的资源如内存、I/O、cpu等，但是进程之间的资源是独立的。

一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程都死掉。所以多进程要比多线程健壮。

进程切换时，消耗的资源大，效率高。所以涉及到频繁的切换时，使用线程要好于进程。同样如果要求同时进行并且又要共享某些变量的并发操作，只能用线程不能用进程

* 执行过程：每个独立的进程有一个程序运行的入口、顺序执行序列和程序入口。但是线程不能独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。
* 线程是处理器(CPU)调度的基本单位，但是进程不是。
* 两者均可并发执行。

2 并发和并行的区别：

并发是指一个处理器同时处理多个任务。   
并行是指多个处理器或者是多核的处理器同时处理多个不同的任务。   
并发是逻辑上的同时发生（simultaneous），而并行是物理上的同时发生。

3 进程间的通信的几种方式

管道（pipe）及命名管道（named pipe）：管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生；

消息队列：消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有写权限得进程可以按照一定得规则向消息队列中添加新信息；对消息队列有读权限得进程则可以从消息队列中读取信息；

共享内存：可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等；

信号量：主要作为进程之间及同一种进程的不同线程之间得同步和互斥手段；

套接字：这是一种更为一般得进程间通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛

4 线程通信的方法：

①同步：多个线程通过synchronized关键字这种方式来实现线程间的通信。

②while轮询的方式

③wait/notify机制

④管道通信就是使用java.io.PipedInputStream 和 java.io.PipedOutputStream进行通信

5 线程之间通信的手段

使用全局变量

主要由于多个线程可能更改全局变量，因此全局变量最好声明为volatile

使用消息实现通信

在Windows程序设计中，每一个线程都可以拥有自己的消息队列（UI线程默认自带消息队列和消息循环，工作线程需要手动实现消息循环），因此可以采用消息进行线程间通信sendMessage,postMessage。

使用事件CEvent类实现线程间通信

Event对象有两种状态：有信号和无信号，线程可以监视处于有信号状态的事件，以便在适当的时候执行对事件的操作。

6 线程同步的方式

互斥量 Synchronized/Lock：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问

信号量 Semphare：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量

事件(信号)，Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作

7 请你说一下线程的同步和互斥以及应用常见

互斥：指在某一时刻指允许一个进程运行其中的程序片，具有排他性和唯一性。

对于线程A和线程B来讲，在同一时刻，只允许一个线程对临界资源进行操作，即当A进入临界区对资源操作时，B就必须等待；当A执行完，退出临界区后，B才能对临界资源进行操作。

同步：指的是在互斥的基础上，实现进程之间的有序访问。假设现有线程A和线程B，线程A需要往缓冲区写数据，线程B需要从缓冲区读数据，但他们之间存在一种制约关系，即当线程A写的时候，B不能来拿数据；B在拿数据的时候A不能往缓冲区写，也就是说，只有当A写完数据（或B取走数据），B才能来读数据（或A才能往里写数据）。这种关系就是一种线程的同步关系。

应用常见：多线程编程中，难免会遇到多个线程同时访问临界资源的问题，如果不对其加以保护，那么结果肯定是不如预期的，因此需要线程同步与互斥。

8 多线程

最开始，线程只是用于分配单个处理器的处理时间的一种工具。但假如操作系统本身支持多个处理器，那么每个线程都可分配给一个不同的处理器，真正进入“并行运算”状态。从程序设计语言的角度看，多线程操作最有价值的特性之一就是程序员不必关心到底使用了多少个处理器。程序在逻辑意义上被分割为数个线程;假如机器本身安装了多个处理器，那么程序会运行得更快，毋需作出任何特殊的调校。根据前面的论述，大家可能感觉线程处理非常简单。但必须注意一个问题：共享资源!如果有多个线程同时运行，而且它们试图访问相同的资源，就会遇到一个问题。举个例子来说，两个线程不能将信息同时发送给一台打印机。为解决这个问题，对那些可共享的资源来说(比如打印机)，它们在使用期间必须进入锁定状态。所以一个线程可将资源锁定，在完成了它的任务后，再解开(释放)这个锁，使其他线程可以接着使用同样的资源。

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

一个采用了多线程技术的应用程序可以更好地利用系统资源。其主要优势在于充分利用了CPU的空闲时间片，可以用尽可能少的时间来对用户的要求做出响应，使得进程的整体运行效率得到较大提高，同时增强了应用程序的灵活性。更为重要的是，由于同一进程的所有线程是共享同一内存，所以不需要特殊的数据传送机制，不需要建立共享存储区或共享文件，从而使得不同任务之间的协调操作与运行、数据的交互、资源的分配等问题更加易于解决。

9 说一说进程同步有哪几种机制

原子操作、信号量机制、自旋锁管程、会合、分布式系统

10 进程同步的方法

1、临界区（Critical Section）:通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。

优点：保证在某一时刻只有一个线程能访问数据的简便办法

缺点：虽然临界区同步速度很快，但却只能用来同步本进程内的线程，而不可用来同步多个进程中的线程。

2、互斥量（Mutex）:为协调共同对一个共享资源的单独访问而设计的。

互斥量跟临界区很相似，比临界区复杂，互斥对象只有一个，只有拥有互斥对象的线程才具有访问资源的权限。

优点：使用互斥不仅仅能够在同一应用程序不同线程中实现资源的安全共享，而且可以在不同应用程序的线程之间实现对资源的安全共享。

缺点：①互斥量是可以命名的，也就是说它可以跨越进程使用，所以创建互斥量需要的资源更多，所以如果只为了在进程内部是用的话使用临界区会带来速度上的优势并能够减少资源占用量。因为互斥量是跨进程的互斥量一旦被创建，就可以通过名字打开它。

②通过互斥量可以指定资源被独占的方式使用，但如果有下面一种情况通过互斥量就无法处理，比如现在一位用户购买了一份三个并发访问许可的数据库系统，可以根据用户购买的访问许可数量来决定有多少个线程/进程能同时进行数据库操作，这时候如果利用互斥量就没有办法完成这个要求，信号量对象可以说是一种资源计数器。

3、信号量（Semaphore）:为控制一个具有有限数量用户资源而设计。它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目。互斥量是信号量的一种特殊情况，当信号量的最大资源数=1就是互斥量了。

优点：适用于对Socket（套接字）程序中线程的同步。（例如，网络上的HTTP服务器要对同一时间内访问同一页面的用户数加以限制，只有不大于设定的最大用户数目的线程能够进行访问，而其他的访问企图则被挂起，只有在有用户退出对此页面的访问后才有可能进入。）

缺点：①信号量机制必须有公共内存，不能用于分布式操作系统，这是它最大的弱点；

②信号量机制功能强大，但使用时对信号量的操作分散， 而且难以控制，读写和维护都很困难，加重了程序员的编码负担；

③核心操作P-V分散在各用户程序的代码中，不易控制和管理，一旦错误，后果严重，且不易发现和纠正。

4、事件（Event）: 用来通知线程有一些事件已发生，从而启动后继任务的开始。

优点：事件对象通过通知操作的方式来保持线程的同步，并且可以实现不同进程中的线程同步操作。

11 什么是死锁？死锁产生的条件？

1). 死锁的概念

在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。

2). 死锁产生的四个必要条件

互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；

请求与保持条件：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有；

非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放

循环等待：若干进程之间形成一种头尾相接的环形等待资源关系

3). 死锁的处理基本策略和常用方法

解决死锁的基本方法主要有 预防死锁、避免死锁、检测死锁、解除死锁 、鸵鸟策略 等。

(1). 死锁预防

死锁预防的基本思想是 只要确保死锁发生的四个必要条件中至少有一个不成立，就能预防死锁的发生，具体方法包括：

打破互斥条件：允许进程同时访问某些资源。但是，有些资源是不能被多个进程所共享的，这是由资源本身属性所决定的，因此，这种办法通常并无实用价值。

打破占有并等待条件：可以实行资源预先分配策略(进程在运行前一次性向系统申请它所需要的全部资源，若所需全部资源得不到满足，则不分配任何资源，此进程暂不运行；只有当系统能满足当前进程所需的全部资源时，才一次性将所申请资源全部分配给该线程)或者只允许进程在没有占用资源时才可以申请资源（一个进程可申请一些资源并使用它们，但是在当前进程申请更多资源之前，它必须全部释放当前所占有的资源）。但是这种策略也存在一些缺点：在很多情况下，无法预知一个进程执行前所需的全部资源，因为进程是动态执行的，不可预知的；同时，会降低资源利用率，导致降低了进程的并发性。

打破非抢占条件：允许进程强行从占有者哪里夺取某些资源。也就是说，但一个进程占有了一部分资源，在其申请新的资源且得不到满足时，它必须释放所有占有的资源以便让其它线程使用。这种预防死锁的方式实现起来困难，会降低系统性能。

打破循环等待条件：实行资源有序分配策略。对所有资源排序编号，所有进程对资源的请求必须严格按资源序号递增的顺序提出，即只有占用了小号资源才能申请大号资源，这样就不回产生环路，预防死锁的发生。

(2). 死锁避免的基本思想

死锁避免的基本思想是动态地检测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态。所谓安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态是安全的，换句话说就是，如果存在一个安全序列，那么系统处于安全状态。资源分配图算法和银行家算法是两种经典的死锁避免的算法，其可以确保系统始终处于安全状态。其中，资源分配图算法应用场景为每种资源类型只有一个实例(申请边，分配边，需求边，不形成环才允许分配)，而银行家算法应用于每种资源类型可以有多个实例的场景。

2、避免死锁：

•    方案一：破坏死锁的循环等待条件。

•    方法二：破坏死锁的请求与保持条件，使用lock的特性，为获取锁操作设置超时时间。这样不会死锁（至少不会无尽的死锁）

•    方法三：设置一个条件遍历与一个锁关联。该方法只用一把锁，没有chopstick类，将竞争从对筷子的争夺转换成了对状态的判断。仅当左右邻座都没有进餐时才可以进餐。提升了并发度。

(3). 死锁解除

死锁解除的常用两种方法为进程终止和资源抢占。所谓进程终止是指简单地终止一个或多个进程以打破循环等待，包括两种方式：终止所有死锁进程和一次只终止一个进程直到取消死锁循环为止；所谓资源抢占是指从一个或多个死锁进程那里抢占一个或多个资源，此时必须考虑三个问题：

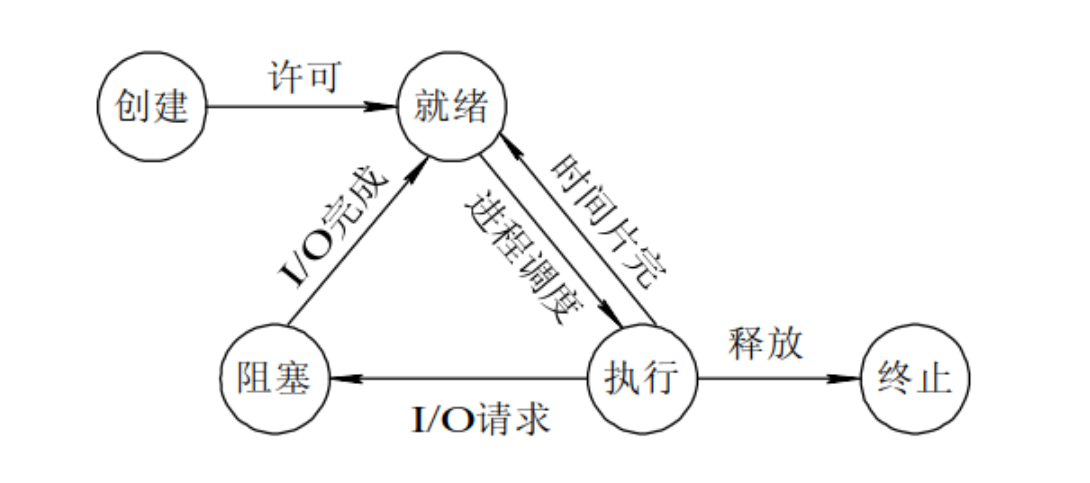
(I). 选择一个牺牲品

(II). 回滚：回滚到安全状态

(III). 饥饿（在代价因素中加上回滚次数，回滚的越多则越不可能继续被作为牺牲品，避免一个进程总是被回滚）

12 进程有哪几种状态？

* 就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源；
* 运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数；
* 阻塞状态： 进程等待某种条件，在条件满足之前无法执行；



13 线程有几种状态？

在 Java虚拟机 中，线程从最初的创建到最终的消亡，要经历若干个状态：创建(new)、就绪(runnable/start)、运行(running)、阻塞(blocked)、等待(waiting)、时间等待(time waiting) 和 消亡(dead/terminated)。在给定的时间点上，一个线程只能处于一种状态，

14 分页和分段有什么区别（内存管理）？

段式存储管理是一种符合用户视角的内存分配管理方案。在段式存储管理中，将程序的地址空间划分为若干段（segment），如代码段，数据段，堆栈段；这样每个进程有一个二维地址空间，相互独立，互不干扰。段式管理的优点是：没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）

页式存储管理方案是一种用户视角内存与物理内存相分离的内存分配管理方案。在页式存储管理中，将程序的逻辑地址划分为固定大小的页（page），而物理内存划分为同样大小的帧，程序加载时，可以将任意一页放入内存中任意一个帧，这些帧不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

两者的不同点：

目的不同：分页是由于系统管理的需要而不是用户的需要，它是信息的物理单位；分段的目的是为了能更好地满足用户的需要，它是信息的逻辑单位，它含有一组其意义相对完整的信息；

大小不同：页的大小固定且由系统决定，而段的长度却不固定，由其所完成的功能决定；

地址空间不同： 段向用户提供二维地址空间；页向用户提供的是一维地址空间；

信息共享：段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制；

内存碎片：页式存储管理的优点是没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）；而段式管理的优点是没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）。

15 操作系统中进程调度策略有哪几种？

FCFS(先来先服务，队列实现，非抢占的)：先请求CPU的进程先分配到CPU

SJF(最短作业优先调度算法)：平均等待时间最短，需要预知未来，但难以知道下一个CPU区间长度

优先级调度算法(可以是抢占的，也可以是非抢占的)：优先级越高越先分配到CPU，相同优先级先到先服务，存在的主要问题是：低优先级进程无穷等待CPU，会导致无穷阻塞或饥饿；解决方案：老化

时间片轮转调度算法(可抢占的)：队列中没有进程被分配超过一个时间片的CPU时间，除非它是唯一可运行的进程。如果进程的CPU区间超过了一个时间片，那么该进程就被抢占并放回就绪队列。

多级队列调度算法：将就绪队列分成多个独立的队列，每个队列都有自己的调度算法，队列之间采用固定优先级抢占调度。其中，一个进程根据自身属性被永久地分配到一个队列中。

多级反馈队列调度算法：与多级队列调度算法相比，其允许进程在队列之间移动：若进程使用过多CPU时间，那么它会被转移到更低的优先级队列；在较低优先级队列等待时间过长的进程会被转移到更高优先级队列，以防止饥饿发生。

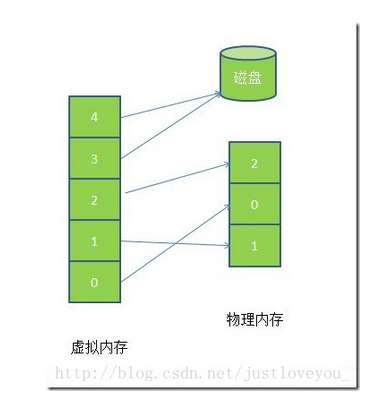
16 什么是虚拟内存？

1).内存的发展历程

没有内存抽象(单进程，除去操作系统所用的内存之外，全部给用户程序使用) —> 有内存抽象（多进程，进程独立的地址空间，交换技术(内存大小不可能容纳下所有并发执行的进程)）—> 连续内存分配(固定大小分区(多道程序的程度受限)，可变分区(首次适应，最佳适应，最差适应)，碎片) —> 不连续内存分配（分段，分页，段页式，虚拟内存）

2).虚拟内存

虚拟内存允许执行进程不必完全在内存中。虚拟内存的基本思想是：每个进程拥有独立的地址空间，这个空间被分为大小相等的多个块，称为页(Page)，每个页都是一段连续的地址。这些页被映射到物理内存，但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻进行必要的映射；当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的命令。这样，对于进程而言，逻辑上似乎有很大的内存空间，实际上其中一部分对应物理内存上的一块(称为帧，通常页和帧大小相等)，还有一些没加载在内存中的对应在硬盘上，如图5所示。



注意，请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。

由图5可以看出，虚拟内存实际上可以比物理内存大。当访问虚拟内存时，会访问MMU（内存管理单元）去匹配对应的物理地址（比如图5的0，1，2）。如果虚拟内存的页并不存在于物理内存中（如图5的3,4），会产生缺页中断，从磁盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将磁盘中的页换出。

3). 页面置换算法(4)

FIFO先进先出页面置换算法：在操作系统中经常被用到，比如作业调度（主要实现简单，很容易想到）；

LRU（Least recently use）最近最久未被访问页面置换算法：根据使用时间到现在的长短来判断；

LFU（Least frequently use）最少使用次数页面置换算法：根据使用次数来判断；

OPT（Optimal replacement）最优页面置换算法：理论的最优，理论；就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。（尽可能的减少页面的换进换出次数）

Clock 算法

4). 虚拟内存的应用与优点

虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。虚拟内存的使用可以带来以下好处：

在内存中可以保留多个进程，系统并发度提高

解除了用户与内存之间的紧密约束，进程可以比内存的全部空间还大

17 颠簸

颠簸本质上是指频繁的页调度行为，具体来讲，进程发生缺页中断，这时，必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立刻再次需要这个页。因此，会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称为颠簸（抖动）。

内存颠簸的解决策略包括：

如果是因为页面替换策略失误，可以修改替换算法来解决这个问题

如果是因为运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低多道程序的数量；

否则，还剩下两个办法：终止该进程或增加物理内存容量。

18 局部性原理—虚拟内存技术

(1). 时间上的局部性：最近被访问的页在不久的将来还会被访问；

(2). 空间上的局部性：内存中被访问的页周围的页也很可能被访问。

19 什么是缓冲区溢出？有什么危害？其原因是什么？

缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

危害有以下两点：

程序崩溃，导致拒绝额服务   
跳转并且执行一段恶意代码   
造成缓冲区溢出的主要原因是程序中没有仔细检查用户输入。

20 请你说一下java里内存泄漏和溢出的区别

1、内存泄漏memory leak :是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏似乎不会有大的影响，但内存泄漏堆积后的后果就是内存溢出。

2、内存溢出 out of memory :指程序申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储int类型数据的存储空间，但是你却存储long类型的数据，那么结果就是内存不够用，此时就会报错OOM,即所谓的内存溢出。

内存溢出常见原因：

1.内存中加载的数据量过于庞大，如一次从数据库取出过多数据；  
2.集合类中有对对象的引用，使用完后未清空，使得JVM不能回收；  
3.代码中存在死循环或循环产生过多重复的对象实体；  
4.使用的第三方软件中的BUG；  
5.启动参数内存值设定的过小

解决方案：

1、修改JVM参数，直接增加内存

2、检查错误日志，查看内存溢出错误前是否有其他异常错误

3、对代码进行走查分析，找出可能发生内存溢出的位置

21 操作系统的内存管理

https://www.cnblogs.com/niupan369/p/4076738.html

数据库原理

1. 数据库范式

第一范式：列不可分，eg:【联系人】（姓名，性别，电话），一个联系人有家庭电话和公司电话，那么这种表结构设计就没有达到 1NF；（所有属性的域都是原子的，则成为第一范式）

第二范式：有主键，保证完全依赖(指每个非主属性能被主属性唯一标识)。eg:订单明细表【OrderDetail】（OrderID，ProductID，UnitPrice，Discount，Quantity，ProductName），Discount（折扣），Quantity（数量）完全依赖（取决）于主键（OderID，ProductID），而 UnitPrice，ProductName 只依赖于 ProductID，不符合2NF；

第三范式：无传递依赖(非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况)，eg:订单表【Order】（OrderID，OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity）主键是（OrderID），CustomerName，CustomerAddr，CustomerCity 直接依赖的是 CustomerID（非主键列），而不是直接依赖于主键，它是通过传递才依赖于主键，所以不符合 3NF。

BCNF **符合3NF，并且，主属性不依赖于主属性**。

（1）所有非主属性对每一个码都是完全函数依赖；

（2）所有的主属性对于每一个不包含它的码，也是完全函数依赖；

（3）没有任何属性完全函数依赖于非码的任意一个组合。

R属于3NF，不一定属于BCNF，如果R属于BCNF，一定属于3NF。

**3**NF是要满足不存在非主属性对候选码的传递函数依赖，BCNF是要满足不存在任一属性（包含非主属性和主属性）对候选码的传递函数依赖。

2 关系型数据库和非关系型数据库的区别

关系型数据库最典型的数据结构是表，由二维表及其之间的联系所组成的一个数据组织

优点：1、易于维护：都是使用表结构，格式一致；2、使用方便：SQL语言通用，可用于复杂查询；3、复杂操作：支持SQL，可用于一个表以及多个表之间非常复杂的查询。

缺点：1、读写性能比较差，尤其是海量数据的高效率读写；2、固定的表结构，灵活度稍欠；3、高并发读写需求，传统关系型数据库来说，硬盘I/O是一个很大的瓶颈。

非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合，可以是文档或者键值对等。

优点：1、格式灵活：存储数据的格式可以是key,value形式、文档形式、图片形式等等，使用灵活，应用场景广泛，而关系型数据库则只支持基础类型。2、速度快：nosql可以使用硬盘或者随机存储器作为载体，而关系型数据库只能使用硬盘；3、高扩展性；4、成本低：nosql数据库部署简单，基本都是开源软件。

缺点：1、不提供sql支持，学习和使用成本较高；2、无事务处理；3、数据结构相对复杂，复杂查询方面稍欠。

3 sql中having用法

having经常与group by和聚合函数（COUNT, SUM, AVG, MIN, or MAX）一起使用，用来过滤由GROUP BY语句返回的记录集。

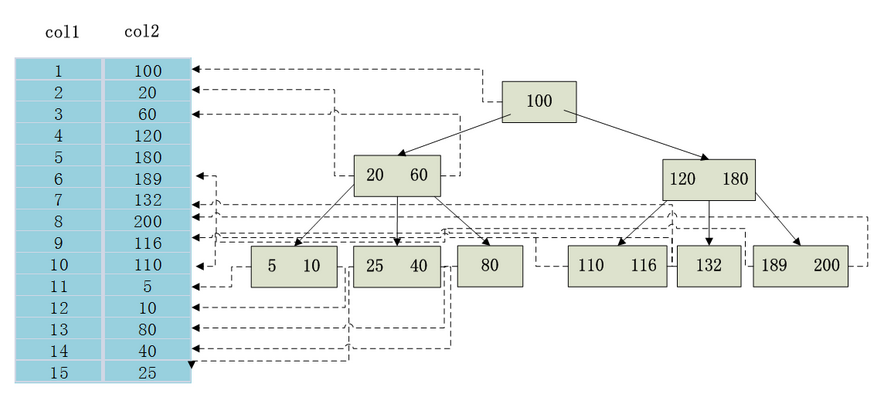
聚合函数通过作用于一组数据而只返回一个单个值，因此，在SELECT语句中出现的元素要么为一个聚合函数的输入值，要么为GROUP BY语句的参数，否则会出错。

HAVING语句的存在弥补了WHERE关键字不能与聚合函数联合使用的不足。

4 数据库索引

索引是对数据库表中一个或多个列的值进行排序的数据结构，以协助快速查询、更新数据库表中数据。(能够帮我们快速的检索数据库中的数据)

索引的实现通常使用B TREE及其变种和Hash索引。索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据指针快速寻找数据。



上图显示了一种索引方式。左边是数据库中的数据表，有col1和col2两个字段，一共有15条记录；右边是以col2列为索引列的B TREE索引，每个节点包含索引的键值和对应数据表地址的指针，这样就可以都过B TREE在 O(logn) 的时间复杂度内获取相应的数据，这样明显地加快了检索的速度。

1). 索引的底层实现原理和优化

在数据结构中，我们最为常见的搜索结构就是二叉搜索树和AVL树(高度平衡的二叉搜索树，为了提高二叉搜索树的效率，减少树的平均搜索长度)了。然而，无论二叉搜索树还是AVL树，当数据量比较大时，都会由于树的深度过大而造成I/O读写过于频繁，进而导致查询效率低下，因此对于索引而言，多叉树结构成为不二选择。特别地，B-Tree的各种操作能使B树保持较低的高度，从而保证高效的查找效率。

(1). B-Tree(平衡多路查找树)

B\_TREE是一种平衡多路查找树，是一种动态查找效率很高的树形结构。B\_TREE中所有结点的孩子结点的最大值称为B\_TREE的阶，B\_TREE的阶通常用m表示，简称为m叉树。一般来说，应该是m>=3。一颗m阶的B\_TREE或是一颗空树，或者是满足下列条件的m叉树：

树中每个结点最多有m个孩子结点；

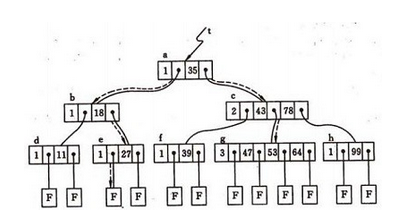
若根结点不是叶子节点，则根结点至少有2个孩子结点；

除根结点外，其它结点至少有(m/2的上界)个孩子结点；

结点的结构如下图所示，其中，n为结点中关键字个数，(m/2的上界)-1 <= n <= m-1；di(1<=i<=n)为该结点的n个关键字值的第i个，且di< d(i+1)；ci(0<=i<=n)为该结点孩子结点的指针，且ci所指向的节点的关键字均大于或等于di且小于d(i+1)；

所有的叶结点都在同一层上，并且不带信息（可以看作是外部结点或查找失败的结点，实际上这些结点不存在，指向这些结点的指针为空）。

下图是一棵4阶B\_TREE，4叉树结点的孩子结点的个数范围[2,4]。其中，有2个结点有4个孩子结点，有1个结点有3个孩子结点，有5个结点有2个孩子结点。



B\_TREE的查找类似二叉排序树的查找，所不同的是B-树每个结点上是多关键码的有序表，在到达某个结点时，先在有序表中查找，若找到，则查找成功；否则，到按照对应的指针信息指向的子树中去查找，当到达叶子结点时，则说明树中没有对应的关键码。由于B\_TREE的高检索效率，B-树主要应用在文件系统和数据库中，对于存储在硬盘上的大型数据库文件，可以极大程度减少访问硬盘次数，大幅度提高数据检索效率。

(2). B+Tree ：(InnoDB存储引擎的索引实现)

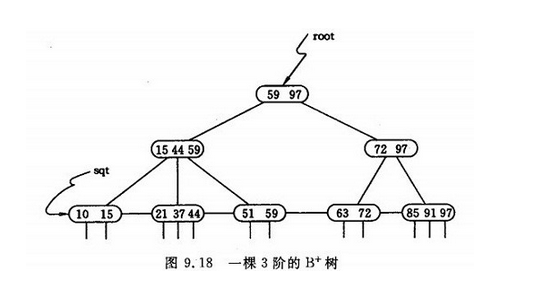
B+Tree是应文件系统所需而产生的一种B\_TREE树的变形树。一棵m阶的B+树和m阶的B\_TREE的差异在于以下三点：

n 棵子树的结点中含有n个关键码；

所有的叶子结点中包含了全部关键码的信息，及指向含有这些关键码记录的指针，且叶子结点本身依关键码的大小自小而大的顺序链接；

非终端结点可以看成是索引部分，结点中仅含有其子树根结点中最大（或最小）关键码。

下图为一棵3阶的B+树。通常在B+树上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点。因此可以对B+树进行两种查找运算：一种是从最小关键字起顺序查找，另一种是从根节点开始，进行随机查找。



在B+树上进行随机查找、插入和删除的过程基本上与B-树类似。只是在查找时，若非终端结点上的关键码等于给定值，并不终止，而是继续向下直到叶子结点。因此，对于B+树，不管查找成功与否，每次查找都是走了一条从根到叶子结点的路径。

(3). 为什么说B+-tree比B 树更适合实际应用中操作系统的文件索引和数据库索引？

B+tree的磁盘读写代价更低：B+tree的内部结点并没有指向关键字具体信息的指针(红色部分)，因此其内部结点相对B树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多，相对来说IO读写次数也就降低了；

B+tree的查询效率更加稳定：由于内部结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引，所以，任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当；

数据库索引采用B+树而不是B树的主要原因：B+树只要遍历叶子节点就可以实现整棵树的遍历，而且在数据库中基于范围的查询是非常频繁的，而B树只能中序遍历所有节点，效率太低。

B+树和B树的区别：

有关b树的一些特性，注意与后面的b+树区分：

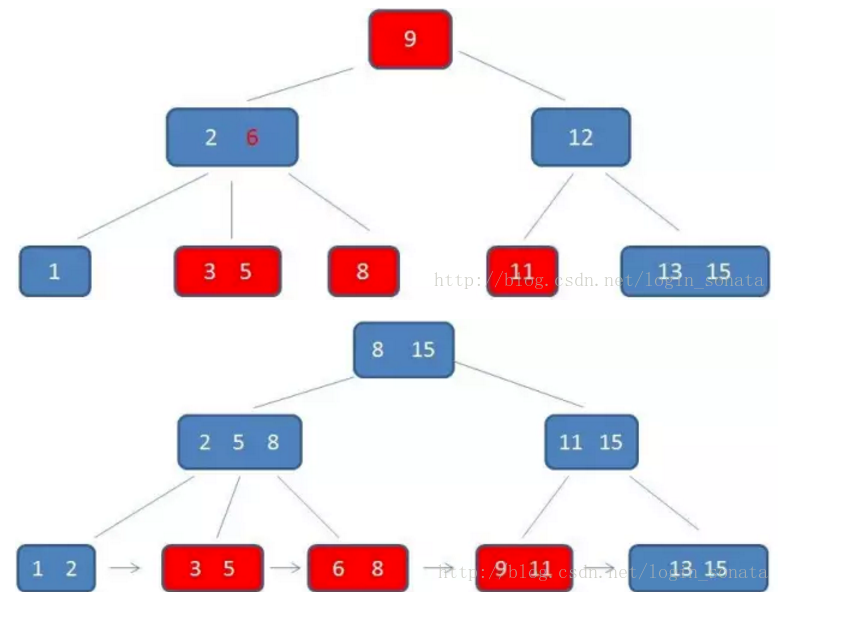
1. 关键字集合分布在整颗树中；
2. 任何一个关键字出现且只出现在一个结点中；
3. 搜索有可能在非叶子结点结束；
4. 其搜索性能等价于在关键字全集内做一次二分查找；

b+树，是b树的一种变体，查询性能更好。m阶的b+树的特征：

1. 有n棵子树的非叶子结点中含有n个关键字（b树是n-1个），这些关键字不保存数据，只用来索引，所有数据都保存在叶子节点（b树是每个关键字都保存数据）。
2. 所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。
3. 所有的非叶子结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。
4. 通常在b+树上有两个头指针，一个指向根结点，一个指向关键字最小的叶子结点。
5. 同一个数字会在不同节点中重复出现，根节点的最大元素就是b+树的最大元素

b+树相比于b树的查询优势：

1. b+树的中间节点不保存数据，所以磁盘页能容纳更多节点元素，更“矮胖”；
2. b+树查询必须查找到叶子节点，b树只要匹配到即可不用管元素位置，因此b+树查找更稳定（并不慢）；
3. 对于范围查找来说，b+树只需遍历叶子节点链表即可，b树却需要重复地中序遍历，如下两图：



(4). 文件索引和数据库索引为什么使用B+树?

文件与数据库都是需要较大的存储，也就是说，它们都不可能全部存储在内存中，故需要存储到磁盘上。而所谓索引，则为了数据的快速定位与查找，那么索引的结构组织要尽量减少查找过程中磁盘I/O的存取次数，因此B+树相比B树更为合适。数据库系统巧妙利用了局部性原理与磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入，而红黑树这种结构，高度明显要深的多，并且由于逻辑上很近的节点(父子)物理上可能很远，无法利用局部性。最重要的是，B+树还有一个最大的好处：方便扫库。B树必须用中序遍历的方法按序扫库，而B+树直接从叶子结点挨个扫一遍就完了，B+树支持range-query非常方便，而B树不支持，这是数据库选用B+树的最主要原因。

2). 索引的优点

大大加快数据的检索速度，这也是创建索引的最主要的原因；

加速表和表之间的连接；

在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间；

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；

3). 什么情况下设置了索引但无法使用？

以“%(表示任意0个或多个字符)”开头的LIKE语句，模糊匹配；

OR语句前后没有同时使用索引；

数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）；

对于多列索引，必须满足 最左匹配原则 (eg：多列索引col1、col2和col3，则 索引生效的情形包括 col1或col1，col2或col1，col2，col3)。

4). 什么样的字段适合创建索引？

经常作查询选择的字段

经常作表连接的字段

经常出现在order by, group by, distinct 后面的字段

5). 创建索引时需要注意什么？

非空字段：应该指定列为NOT NULL，除非你想存储NULL。在mysql中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用0、一个特殊的值或者一个空串代替空值；

取值离散大的字段：（变量各个取值之间的差异程度）的列放到联合索引的前面，可以通过count()函数查看字段的差异值，返回值越大说明字段的唯一值越多字段的离散程度高；

索引字段越小越好：数据库的数据存储以页为单位一页存储的数据越多一次IO操作获取的数据越大效率越高。

6). 索引的缺点

时间方面：创建索引和维护索引要耗费时间，具体地，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度；

空间方面：索引需要占物理空间。

7). 索引的分类

普通索引和唯一性索引：索引列的值的唯一性

单个索引和复合索引：索引列所包含的列数

聚簇索引与非聚簇索引：聚簇索引按照数据的物理存储进行划分的。对于一堆记录来说，使用聚集索引就是对这堆记录进行堆划分，即主要描述的是物理上的存储。正是因为这种划分方法，导致聚簇索引必须是唯一的。聚集索引可以帮助把很大的范围，迅速减小范围。但是查找该记录，就要从这个小范围中Scan了；而非聚集索引是把一个很大的范围，转换成一个小的地图，然后你需要在这个小地图中找你要寻找的信息的位置，最后通过这个位置，再去找你所需要的记录。

8). 主键、自增主键、主键索引与唯一索引概念区别

主键：指字段唯一、不为空值的列；

主键索引：指的就是主键，主键是索引的一种，是唯一索引的特殊类型。创建主键的时候，数据库默认会为主键创建一个唯一索引；

自增主键：字段类型为数字、自增、并且是主键；

唯一索引：索引列的值必须唯一，但允许有空值。主键是唯一索引，这样说没错；但反过来说，唯一索引也是主键就错误了，因为唯一索引允许空值，主键不允许有空值，所以不能说唯一索引也是主键。

5 mysql索引的类型

MySQL目前主要有以下几种索引类型：  
1.普通索引：仅加速查询  
2.唯一索引：加速查询 + 列值唯一（可以有null）  
3.主键索引：加速查询 + 列值唯一（不可以有null）+ 表中只有一个  
4.组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并  
5.全文索引：对文本的内容进行分词，进行搜索

6 唯一索引和主键索引的区别

1.主键为一种约束，唯一索引为一种索引，本质上就不同。

2.主键在表中只能有一个，唯一索引可以有多个。

3.主键创建后一定包含唯一性索引，而唯一索引不一定就是主键。

4.主键不能为null,唯一索引可以为null.

5.主键可以被其它表引用，唯一索引不能。

6.主键和索引都是键，主键是逻辑键，索引为物理键，即主键不实际存在。

7 主键就是聚集索引吗？主键和索引有什么区别？

主键是一种特殊的唯一性索引，其可以是聚集索引，也可以是非聚集索引。在SQLServer中，主键的创建必须依赖于索引，默认创建的是聚集索引，但也可以显式指定为非聚集索引。InnoDB作为MySQL存储引擎时，默认按照主键进行聚集，如果没有定义主键，InnoDB会试着使用唯一的非空索引来代替。如果没有这种索引，InnoDB就会定义隐藏的主键然后在上面进行聚集。所以，对于聚集索引来说，你创建主键的时候，自动就创建了主键的聚集索引。

8 数据库索引的优缺点及原理

**索引的原理：索引的原理大致概括为以空间换时间，数据库在未添加索引的时候进行查询默认的是进行全量搜索，也就是进行全局扫描，有多少条数据就要进行多少次查询，然后找到相匹配的数据就把他放到结果集中，直到全表扫描完。而建立索引之后，会将建立索引的KEY值放在一个n叉树上（BTree）。因为B树的特点就是适合在磁盘等直接存储设备上组织动态查找表，每次以索引进行条件查询时，会去树上根据key值直接进行搜索，次数约为log总条数，底数为页面存储数**

**索引的优点：**

①建立索引的列可以保证行的唯一性，生成唯一的rowId

②建立索引可以有效缩短数据的检索时间

③建立索引可以加快表与表之间的连接

④为用来排序或者是分组的字段添加索引可以加快分组和排序顺序

**索引的缺点：**

1.创建索引和维护索引需要时间成本，这个成本随着数据量的增加而加大

2.创建索引和维护索引需要空间成本，每一条索引都要占据数据库的物理存储空间，数据量越大，占用空间也越大（数据表占据的是数据库的数据空间）

3.会降低表的增删改的效率，因为每次增删改索引需要进行动态维护，导致时间变长

**什么样的表跟列要建立索引：**

1.总的来说就是数据量大的，经常进行查询操作的表要建立索引

2.表中字段建立索引应该遵循几个原则：越小的数据类型通常更好简单的数据类型更好,尽量避免NULL对非唯一的字段

3.表与表连接用于多表联合查询的约束条件的字段应当建立索引

4.用于排序的字段可以添加索引，用于分组的字段应当视情况看是否需要添加索引。

5.添加多列索引的时候，对应的多条件查询可以触发该索引的同时，索引最左侧的列的单条件查询也可以触发。

6.如果有些表注定只会进行查询所有，也就没必要添加索引，因为查询全部只能进行全量搜索即扫描全表。

9 哪些情况下不应该使用索引

1) 数据唯一性差的字段不要使用索引

2) 频繁更新的字段不要使用索引

3) 字段不在where语句出现时不要添加索引

4) 数据量少的表不要使用索引

10 哪些情况下索引会失效

1,<>

2,单独的>,<,(有时会用到，有时不会)

3,like "%\_" 百分号在前.

4,表没分析.

5,单独引用复合索引里非第一位置的索引列.

6,字符型字段为数字时在where条件里不添加引号.

7,对索引列进行运算.需要建立函数索引.

8,not in ,not exist.

9,当变量采用的是times变量，而表的字段采用的是date变量时.或相反情况。

10, 索引失效。

11,基于cost成本分析(oracle因为走全表成本会更小)：查询小表,或者返回值大概在10%以上

12,有时都考虑到了 但就是不走索引,drop了重建试试在

13,B-tree索引 is null不会走,is not null会走,位图索引 is null,is not null都会走

14,联合索引 is not null 只要在建立的索引列（不分先后）都会走,in null 必须要和建立索引第一列一起使用,当建立索引第一位置条件是is null 时,其他建立索引的列可以是is null（但必须在所有列都满足is null的时候）,或者=一个值；当建立索引的第一位置是=一个值时,其他索引列可以是任何情况（包括is null =一个值）,以上两种情况索引都会走。其他情况不会走。

二、索引失效解决方法

1. 选用适合的Oracle优化器

2、重建索引

3、强制索引

11 数据库索引的实现方式（数据结构）

B+树：数据库索引采用B+树的主要原因是B+树在非叶子节点不保存数据，只在叶子节点保存。而B树在叶子节点和非叶子节点都会保存。这种结构如果用B树会增加磁盘IO次数，导致性能不如B+树

Hash索引：就是通过散列函数来定位的一种索引，不过很少有单独使用散列索引的，反而是散列文件组织用的比较多。

散列文件组织就是根据一个键通过散列计算把对应的记录都放到同一个槽中，这样的话相同的键值对应的记录就一定是放在同一个文件里了，也就减少了文件读取的次数，提高了效率。

散列索引就是根据对应键的散列码来找到最终的索引项的技术，其实和B树就差不多了，也就是一种索引之上的二级辅助索引，我理解散列索引都是二级或更高级的稀疏索引，否则桶就太多了，效率也不会很高。

位图索引：位图索引是一种针对多个字段的简单查询设计一种特殊的索引，适用范围比较小，只适用于字段值固定并且值的种类很少的情况，比如性别，只能有男和女，或者级别，状态等等，并且只有在同时对多个这样的字段查询时才能体现出位图的优势。

位图的基本思想就是对每一个条件都用0或者1来表示，如有5条记录，性别分别是男，女，男，男，女，那么如果使用位图索引就会建立两个位图，对应男的10110和对应女的01001,这样做有什么好处呢，就是如果同时对多个这种类型的字段进行and或or查询时，可以使用按位与和按位或来直接得到结果了。

B+树最常用，性能也不差，用于范围查询和单值查询都可以。特别是范围查询，非得用B+树这种顺序的才可以了。 HASH的如果只是对单值查询的话速度会比B+树快一点，但是ORACLE好像不支持HASH索引，只支持HASH表空间。位图的使用情况很局限，只有很少的情况才能用，一定要确定真正适合使用这种索引才用（值的类型很少并且需要复合查询），否则建立一大堆位图就一点意义都没有了。

B+ Tree索引和Hash索引区别 哈希索引适合等值查询，但是不无法进行范围查询 哈希索引没办法利用索引完成排序 哈希索引不支持多列联合索引的最左匹配规则 如果有大量重复键值得情况下，哈希索引的效率会很低，因为存在哈希碰撞问题.

多列联合索引的最左匹配规则：假设联合索引是state/city/zipCode

那么state就是第一关，city是第二关，zipCode就是第三关

必须匹配了第一关，才能匹配第二关，匹配了第一关和第二关，才能匹配第三关

你不能直接到第二关的

索引的格式就是第一层是state，第二层才是city

多列索引是先按照第一列进行排序，然后在第一列排好序的基础上再对第二列排序，如果没有第一列的话，直接访问第二列，那第二列肯定是无序的，直接访问后面的列就用不到索引了. 在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边

12 数据库事务

事务是一个不可分割的数据库操作序列，也是数据库并发控制的基本单位，其执行的结果必须使数据库从一种一致性状态变到另一种一致性状态。

(1). 事务的特征

数据库事物四大特性-ACID

原子性、一致性、隔离性，持久性

1 原子性（Atomicity) 指的是事物中包含的所有操作要么全做，要么全不做（all or

none）。

2 一致性（consistency）在事物开始以前，数据库处于一致性的状态，事物结束后，数据库也必须处于一致性的状态。

3 隔离性（Isolation) 事务的隔离性要求系统必须保证事务不受其他并发执行的事务的影响，也既要达到这样一种效果：对于任何一对事务T1和T2，在事务T1看来，T2要么在T1开始之前执行，要么在T1完成之后才开始执行，这样，每个事务都感觉不到系统中有并发事务执行。

4 持久性（Durability) 一个事务一旦成功完成，它对数据库的改变必须是永久的，即使是在系统遇到故障的情况下也不会丢失，数据的重要性决定了事物的持久性的重要。

(2). 事务并发带来的问题

脏读：一个事务读取了另一个事务未提交的数据；

不可重复读：不可重复读的重点是修改，同样条件下两次读取结果不同，也就是说，被读取的数据可以被其它事务修改；

幻读：幻读的重点在于新增或者删除，同样条件下两次读出来的记录数不一样。

(3). 隔离级别

1、未提交读(Read Uncommitted)自定义名称：修改级别隔离（只要修改了数据，则其他事务就可以看到）

可以理解为：A事务只要修改了数据，无论有没有提交，其他事务都能够读取到A事务修改后的结果。

潜在问题：【对于同一记录进行修改操作】：A事务修改之后未提交，B事务读取到修改之后的数据，然后在这个基础上进行操作修改并提交。然后A提交失败，数据回滚，则数据就出现异常。这就是所谓的脏读。

解决方案：A事务提交之后B事务才能读取到A修改之后的数据，不让读取修改但未提交的数据

2、提交读(Read Committed) 自定义名称：提交级别隔离（只有成功提交事务，则其他事务才能够看到）

可以理解为：A事务成功commit之后，其他事务才能够读取到A提交事务之后的最新数据，否则只能读取到A提交之前的原始数据。这样就解决了脏读问题。

潜在问题【对于同一记录进行修改操作】：A事务开启第一次读取数据，然后B事务开启对数据进行了修改并提交成功， 此时A事务还没有结束，又进行了一次读操作，然后便发现A事务中的两次读取的数据不一致。这就是所谓的不可重复读。同一个事务中两次读取的数据不一样。

解决方案：在一个事务对该行数据进行操作的时候，其他事务不允许进行修改。可以理解为添加一个行级别的锁

3、可重复读(Repeatable Read) 自定义名称：事务级别隔离（A事务中的数据不会受B事务中的修改操作影响，即使B事务提交之后，A事务再次查询的数据也不会发生变化）

可以理解为：在自己的事务中复制了一套相关的数据，该数据是私有的不受其他事务的影响。因此可以解决不可重复读的问题。这里虽然解决了不可重读的问题，但是却没有解决数据覆盖的问题，也就是B操作的数据是最原始的数据（和A未修改之前的数据一样），而并不是最新（A修改之后的数据）的数据。对于此隔离机制无法解决这个问题，需要使用行级锁来进行处理。

潜在问题 【对于同表进行插入或删除操作】：A事务插入/删除了一条记录，但是B事务中却查询不到插入的新记录/依旧能查到删除的记录。这就是所谓的幻读。幻读是对于记录条数而言的，不是针对于某一条记录的数据而言的。这里需要注意！

解决方案：在有事务对该表进行操作的时候，不允许其他事务操作该表。添加表级锁。

4、串行读(Serializable)自定义名称：表级别隔离（不允许同时有多个事务操作该表）

可以理解为：所有的事务串行操作该数据表。这样就可以解决幻读操作。

潜在问题：数据安全了 但是操作效率降低了。

(4)、mysql的事务支持

MySQL的事务支持不是绑定在MySQL服务器本身，而是与存储引擎相关：

MyISAM：不支持事务，用于只读程序提高性能；

InnoDB：支持ACID事务、行级锁、并发；

Berkeley DB：支持事务。

13 数据库引擎MyISAM和InnoDB

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：

InnoDB支持事务，MyISAM不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而MyISAM 就不可以了；

MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；

InnoDB 支持外键，MyISAM 不支持；

MyISAM 是默认引擎，InnoDB 需要指定；

InnoDB 不支持FULLTEXT 类型的索引；

InnoDB 中不保存表的行数，如select count() from table 时，InnoDB需要扫描一遍整个表来计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count()语句包含 where 条件时 MyISAM 也需要扫描整个表；

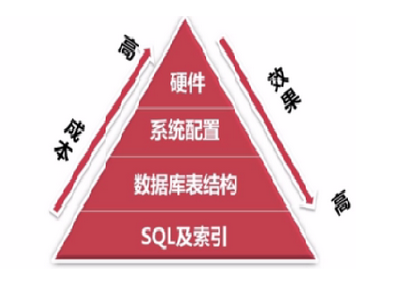
对于自增长的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM 表中可以和其他字段一起建立联合索引；

清空整个表时，InnoDB是一行一行的删除，效率非常慢。MyISAM 则会重建表；

InnoDB 支持行锁（某些情况下还是锁整表，如update table set a=1 where user like '%lee%'.

MyISAM的索引为非聚集索引，InnoDB的索引为聚集索引

14 **数据库优化**

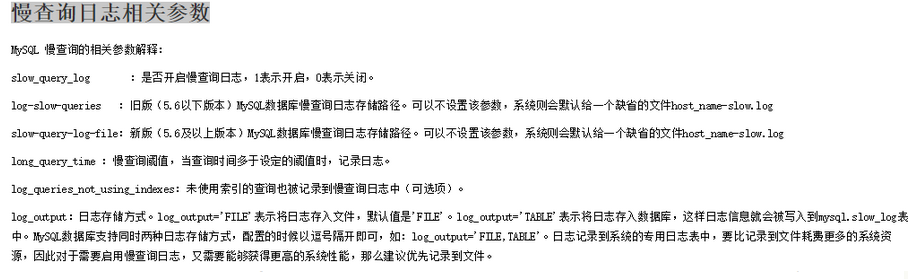


(1). SQL语句的优化

SQL语句的优化主要包括三个问题，即如何发现有问题的SQL、如何分析SQL的执行计划以及如何优化SQL，下面将逐一解释。

a. 怎么发现有问题的SQL?（通过MySQL慢查询日志对有效率问题的SQL进行监控）

MySQL的慢查询日志是MySQL提供的一种日志记录，它用来记录在MySQL中响应时间超过阀值的语句，具体指运行时间超过long\_query\_time值的SQL，则会被记录到慢查询日志中。long\_query\_time的默认值为10，意思是运行10s以上的语句。慢查询日志的相关参数如下所示：



通过MySQL的慢查询日志，我们可以查询出执行的次数多占用的时间长的SQL、可以通过pt\_query\_disgest(一种mysql慢日志分析工具)分析Rows examine(MySQL执行器需要检查的行数)项去找出IO大的SQL以及发现未命中索引的SQL，对于这些SQL，都是我们优化的对象。

b. 通过explain查询和分析SQL的执行计划

使用 EXPLAIN 关键字可以知道MySQL是如何处理你的SQL语句的，以便分析查询语句或是表结构的性能瓶颈。通过explain命令可以得到表的读取顺序、数据读取操作的操作类型、哪些索引可以使用、哪些索引被实际使用、表之间的引用以及每张表有多少行被优化器查询等问题。当扩展列extra出现Using filesort和Using temporay，则往往表示SQL需要优化了。

c. SQL语句的优化

优化insert语句：一次插入多值；

应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描；

应尽量避免在 where 子句中对字段进行null值判断，否则将导致引擎放弃使用索引而进行全表扫描；

优化嵌套查询：子查询可以被更有效率的连接(Join)替代；

很多时候用 exists 代替 in 是一个好的选择。

（2） 索引优化

建议在经常作查询选择的字段、经常作表连接的字段以及经常出现在order by、group by、distinct 后面的字段中建立索引。但必须注意以下几种可能会引起索引失效的情形：

以“%(表示任意0个或多个字符)”开头的LIKE语句，模糊匹配；

OR语句前后没有同时使用索引；

数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）；

对于多列索引，必须满足最左匹配原则(eg,多列索引col1、col2和col3，则索引生效的情形包括col1或col1，col2或col1，col2，col3)。

2). 数据库表结构的优化

数据库表结构的优化包括选择合适数据类型、表的范式的优化、表的垂直拆分和表的水平拆分等手段。

(1). 选择合适数据类型

使用较小的数据类型解决问题；

使用简单的数据类型(mysql处理int要比varchar容易)；

尽可能的使用not null 定义字段；

尽量避免使用text类型，非用不可时最好考虑分表；

(2). 表的范式的优化

一般情况下，表的设计应该遵循三大范式。

(3). 表的垂直拆分

把含有多个列的表拆分成多个表，解决表宽度问题，具体包括以下几种拆分手段：

把不常用的字段单独放在同一个表中；

把大字段独立放入一个表中；

把经常使用的字段放在一起；

这样做的好处是非常明显的，具体包括：拆分后业务清晰，拆分规则明确、系统之间整合或扩展容易、数据维护简单。

(4). 表的水平拆分

表的水平拆分用于解决数据表中数据过大的问题，水平拆分每一个表的结构都是完全一致的。一般地，将数据平分到N张表中的常用方法包括以下两种：

对ID进行hash运算，如果要拆分成5个表，mod(id,5)取出0~4个值；

针对不同的hashID将数据存入不同的表中；

3). 系统配置的优化

操作系统配置的优化：增加TCP支持的队列数

mysql配置文件优化：Innodb缓存池设置(innodb\_buffer\_pool\_size，推荐总内存的75%)和缓存池的个数（innodb\_buffer\_pool\_instances）

4). 硬件的优化

CPU：核心数多并且主频高的

内存：增大内存

磁盘配置和选择：磁盘性能

15 什么是存储过程？有哪些优缺点？

存储过程是事先经过编译并存储在数据库中的一段SQL语句的集合。进一步地说，存储过程是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。存储过程具有以下特点：

存储过程只在创建时进行编译，以后每次执行存储过程都不需再重新编译，而一般SQL语句每执行一次就编译一次，所以使用存储过程可提高数据库执行效率；

当SQL语句有变动时，可以只修改数据库中的存储过程而不必修改代码；

减少网络传输，在客户端调用一个存储过程当然比执行一串SQL传输的数据量要小；

通过存储过程能够使没有权限的用户在控制之下间接地存取数据库，从而确保数据的安全。

16 简单说一说drop、delete与truncate的区别

SQL中的drop、delete、truncate都表示删除，但是三者有一些差别：

Delete用来删除表的全部或者一部分数据行，执行delete之后，用户需要提交(commmit)或者回滚(rollback)来执行删除或者撤销删除， delete命令会触发这个表上所有的delete触发器；

Truncate删除表中的所有数据，这个操作不能回滚，也不会触发这个表上的触发器，TRUNCATE比delete更快，占用的空间更小；

Drop命令从数据库中删除表，所有的数据行，索引和权限也会被删除，所有的DML触发器也不会被触发，这个命令也不能回滚。

因此，在不再需要一张表的时候，用drop；在想删除部分数据行时候，用delete；在保留表而删除所有数据的时候用truncate。

1、在速度上，一般来说，drop>truncate>delete。

2、在使用drop和truncate时一定要注意，虽然可以恢复，但为了减少麻烦，还是要慎重。

3、如果想删除部分数据用delete，注意带上where子句，回滚段要足够大；如果想删除表，当然用drop；如果想保留表而将所有数据删除，如果和事务无关，用truncate即可；如果和事务有关，或者想触发trigger，还是用delete；如果是整理表内部的碎片，可以用truncate跟上reuse stroage，再重新导入/插入数据。

17 什么叫视图？游标是什么？

视图是一种虚拟的表，通常是有一个表或者多个表的行或列的子集，具有和物理表相同的功能，可以对视图进行增，删，改，查等操作。特别地，对视图的修改不影响基本表。相比多表查询，它使得我们获取数据更容易。

游标是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

在操作mysql的时候，我们知道MySQL检索操作返回一组称为结果集的行。这组返回的行都是与 SQL语句相匹配的行（零行或多行）。使用简单的 SELECT语句，例如，没有办法得到第一行、下一行或前 10行，也不存在每次一行地处理所有行的简单方法（相对于成批地处理它们）。有时，需要在检索出来的行中前进或后退一行或多行。这就是使用游标的原因。游标（cursor）是一个存储在MySQL服务器上的数据库查询，它不是一条 SELECT语句，而是被该语句检索出来的结果集。在存储了游标之后，应用程序可以根据需要滚动或浏览其中的数据。游标主要用于交互式应用，其中用户需要滚动屏幕上的数据，并对数据进行浏览或做出更改。

18 什么是触发器？

触发器是与表相关的数据库对象，在满足定义条件时触发，并执行触发器中定义的语句集合。触发器的这种特性可以协助应用在数据库端确保数据库的完整性。

19 数据库锁分类和总结

**锁**是网络数据库中的一个非常重要的概念，当多个用户同时对数据库并发操作时，会带来数据不一致的问题，所以，锁主要用于多用户环境下保证数据库完整性和一致性。

数据库锁出现的目的：处理并发问题

**并发控制的主要采用的技术手段**：乐观锁、悲观锁和时间戳。

从数据库系统角度分为三种：排他锁、共享锁、更新锁。  
从程序员角度分为两种：一种是悲观锁，一种乐观锁。

**悲观锁（Pessimistic Lock）**

顾名思义，很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人拿这个数据就会block（阻塞），直到它拿锁。

悲观锁（Pessimistic Lock）：正如其名，具有强烈的独占和排他特性。它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

传统的关系数据库里用到了很多这种锁机制，比如行锁、表锁、读锁、写锁等，都是在操作之前先上锁。

**悲观锁按使用性质划分**

共享锁（Share Lock）:S锁，也叫读锁，用于所有的只读数据操作。共享锁是非独占的，允许多个并发事务读取其锁定的资源。

**性质**

**1. 多个事务可封锁同一个共享页；**

**2. 任何事务都不能修改该页；**

**3. 通常是该页被读取完毕，S锁立即被释放。**

例如，执行查询语句“SELECT \* FROM my\_table”时，首先锁定第一页，读取之后，释放对第一页的锁定，然后锁定第二页。这样，就允许在读操作过程中，修改未被锁定的第一页。

例如，语句“SELECT \* FROM my\_table HOLDLOCK”就要求在整个查询过程中，保持对表的锁定，直到查询完成才释放锁定。

**排他锁（Exclusive Lock）**

X锁，也叫写锁，表示对数据进行写操作。如果一个事务对对象加了排他锁，其他事务就不能再给它加任何锁了。

**性质**

**1. 仅允许一个事务封锁此页；**

**2. 其他任何事务必须等到X锁被释放才能对该页进行访问；**

**3. X锁一直到事务结束才能被释放。**

**产生排他锁的SQL语句如下：select \* from ad\_plan for update;**

更新锁

U锁，在修改操作的初始化阶段用来锁定可能要被修改的资源，这样可以避免使用共享锁造成的死锁现象。

因为当使用共享锁时，修改数据的操作分为两步：

1. 首先获得一个共享锁，读取数据，

2. 然后将共享锁升级为排他锁，再执行修改操作。

这样如果有两个或多个事务同时对一个事务申请了共享锁，在修改数据时，这些事务都要将共享锁升级为排他锁。这时，这些事务都不会释放共享锁，而是一直等待对方释放，这样就造成了死锁。如果一个数据在修改前直接申请更新锁，在数据修改时再升级为排他锁，就可以避免死锁。

性质

1. 用来预定要对此页施加X锁，它允许其他事务读，但不允许再施加U锁或X锁；

2. 当被读取的页要被更新时，则升级为X锁；

3. U锁一直到事务结束时才能被释放。

**悲观锁按作用范围划分为：行锁、表锁。**

行锁：锁的作用范围是行级别。

表锁：锁的作用范围是整张表。

数据库能够确定那些行需要锁的情况下使用行锁，如果不知道会影响哪些行的时候就会使用表锁。

**乐观锁（Optimistic Lock）**

顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以，不会上锁。但是在更新的时候会判断一下在此期间别人有没有更新这个数据，可以使用版本号等机制。

乐观锁（ Optimistic Locking ）：相对悲观锁而言，乐观锁机制采取了更加宽松的加锁机制。悲观锁大多数情况下依靠数据库的锁机制实现，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。而乐观锁机制在一定程度上解决了这个问题。

乐观锁，大多是基于数据版本（ Version ）记录机制实现。

数据版本：为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据.

乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

乐观锁的实现方式：

版本号（记为version）：

就是给数据增加一个版本标识，在数据库上就是表中增加一个version字段，每次更新把这个字段加1，读取数据的时候把version读出来，更新的时候比较version，如果还是开始读取的version就可以更新了，如果现在的version比老的version大，说明有其他事务更新了该数据，并增加了版本号，这时候得到一个无法更新的通知，用户自行根据这个通知来决定怎么处理，比如重新开始一遍。这里的关键是判断version和更新两个动作需要作为一个原子单元执行，否则在你判断可以更新以后正式更新之前有别的事务修改了version，这个时候你再去更新就可能会覆盖前一个事务做的更新，造成第二类丢失更新，所以你可以使用update … where … and version=”old version”这样的语句，根据返回结果是0还是非0来得到通知，如果是0说明更新没有成功，因为version被改了，如果返回非0说明更新成功。

时间戳（使用数据库服务器的时间戳）

和版本号基本一样，只是通过时间戳来判断而已，注意时间戳要使用数据库服务器的时间戳不能是业务系统的时间。

待更新字段

和版本号方式相似，只是不增加额外字段，直接使用有效数据字段做版本控制信息，因为有时候我们可能无法改变旧系统的数据库表结构。假设有个待更新字段叫count,先去读取这个count,更新的时候去比较数据库中count的值是不是我期望的值（即开始读的值），如果是就把我修改的count的值更新到该字段，否则更新失败。java的基本类型的原子类型对象如AtomicInteger就是这种思想。

所有字段

和待更新字段类似，只是使用所有字段做版本控制信息，只有所有字段都没变化才会执行更新。

乐观锁几种方式的区别

新系统设计可以使用version方式和timestamp方式，需要增加字段，应用范围是整条数据，不论那个字段修改都会更新version,也就是说两个事务更新同一条记录的两个不相关字段也是互斥的，不能同步进行。旧系统不能修改数据库表结构的时候使用数据字段作为版本控制信息，不需要新增字段，待更新字段方式只要其他事务修改的字段和当前事务修改的字段没有重叠就可以同步进行，并发性更高。

并发控制会造成两种锁

活锁

死锁

并发控制会造成活锁和死锁，就像操作系统那样，会因为互相等待而导致。

活锁:定义：指的是T1封锁了数据R，T2同时也请求封锁数据R，T3也请求封锁数据R，当T1释放了锁之后，T3会锁住R，T4也请求封锁R，则T2就会一直等待下去。

解决方法：采用“先来先服务”策略可以避免。

死锁定义：就是我等你，你又等我，双方就会一直等待下去。比如：T1封锁了数据R1，正请求对R2封锁，而T2封住了R2,正请求封锁R1，这样就会导致死锁，死锁这种没有完全解决的方法，只能尽量预防。

预防方法：

1. 一次封锁法，指的是一次性把所需要的数据全部封锁住，但是这样会扩大了封锁的范围，降低系统的并发度；

2. 顺序封锁法，指的是事先对数据对象指定一个封锁顺序，要对数据进行封锁，只能按照规定的顺序来封锁，但是这个一般不大可能的。

系统判定死锁的方法：

超时法：如果某个事物的等待时间超过指定时限，则判定为出现死锁；

等待图法：如果事务等待图中出现了回路，则判断出现了死锁。

对于解决死锁的方法，只能是撤销一个处理死锁代价最小的事务，释放此事务持有的所有锁，同时对撤销的事务所执行的数据修改操作必须加以恢复。

20 查询每门课程成绩都大于80分学生的学号  
数据库 表 student  
name score course  
A 85  语文  
A 75  数学  
A 82  英语  
B   75  语文  
B   89  数学  
B   79  英语  
天使美眉90 语文  
天使美眉100 数学  
天使美眉100 英语  
请找出每门课程都超过80分的那个人名字的SQL语句

SQL1:  
select name from stu  
group by name  
having name not in (  
select name from stu  
where score <80)  
SQL2:  
select name from test.stu  
group by name  
having min(score)>=80

21 数据库分库分表策略，如何分库，如何分表？

切分策略

数据切分分为两种方式，纵向切分和水平切分

1. 纵向切分  
   常见有纵向分库纵向分表两种。  
   1）.纵向分库就是根据业务耦合性，将关联度低的不同表存储在不同的数据库，做法与大系统拆分为多个小系统类似，按业务分类进行独立划分。与“微服务治理”的做法相似，每个微服务使用单独的一个数据库。  
   2）.垂直分表是基于数据库中的列进行，某个表字段较多，可以新建一张扩展表，将不经常用或者字段长度较大的字段拆出到扩展表中。在字段很多的情况下，通过大表拆小表，更便于开发与维护，也能避免跨页问题，MYSQL底层是通过数据页存储的，一条记录占用空间过大会导致跨页，造成额外的开销。另外，数据库以行为单位将数据加载到内存中，这样表中字段长度越短且访问频次较高，内存能加载更多的数据，命中率更高，减少磁盘IO，从而提升数据库的性能。

* 垂直切分的优点：
  + 解决业务系统层面的耦合，业务清晰
  + 与微服务的治理类似，也能对不同业务的数据进行分级管理，维护，监控，扩展等。
  + 高并发场景下，垂直切分一定程度的提升IO，数据库连接数，单机硬件资源的瓶颈。
* 垂直切分的缺点
  + 部分表无法join，只能通过接口聚合方式解决，提升了开发的复杂度。
  + 分布式事处理复杂
  + 依然存在单表数据量过大的问题。

1. 水平切分  
   当一个应用难以再细粒度的垂直切分或切分后数据量行数依然巨大，存在单库读写，存储性能瓶颈，这时候需要进行水平切分。  
   水平切分为库内分表和分库分表，是根据表内数据内在的逻辑关系，将同一个表按不同的条件分散到多个数据库或多表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果。  
   库内分表只解决单一表数据量过大的问题，但没有将表分布到不同机器的库上，因些对于减轻mysql的压力来说帮助不是很大，大家还是竞争同一个物理机的CPU、内存、网络IO，最好通过分库分表来解决。

* 水平切分优点
  + 不存在单库数据量过大、高并发的性能瓶颈，提升系统稳定性和负载能力。
  + 应用端改造较小，不需要拆分业务模块。
* 水平切分缺点
  + 跨分片的事务一致性难以保证
  + 跨库的join关联查询性能较差
  + 数据多次扩展维度和维护量极大。

22 redis 数据库和mysql的关系

redis是Nosql数据库中使用较为广泛的非关系型内存数据库，redis内部是一个key-value存储系统。它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set –有序集合)和hash（哈希类型，类似于Java中的map）。

redis和mysql要根据具体业务场景去选型

* mysql：数据放在磁盘
* redis：数据放在内存

redis适合放一些频繁使用，比较热的数据，因为是放在内存中，读写速度都非常快，一般会应用在下面一些场景

* 排行榜
* 计数器
* 消息队列推送
* 好友关注，粉丝

首先要知道mysql存储在磁盘里，redis存储在内存里，redis既可以用来做持久存储，也可以做缓存，而目前大多数公司的存储都是mysql + redis，mysql作为主存储，redis作为辅助存储被用作缓存，加快访问读取的速度，提高性能  
那么为什么不直接全部用redis存储呢？  
我的看法是：因为redis存储在内存中，如果存储在内存中，存储容量肯定要比磁盘少很多，那么要存储大量数据，只能花更多的钱去购买内存，造成在一些不需要高性能的地方是相对比较浪费的，所以目前基本都是mysql(主) + redis(辅)，在需要性能的地方使用redis，在不需要高性能的地方使用mysql，好钢用在刀刃上

目前redis做数据库还不太靠谱。它支持的数据类型太少，而且查询功能太弱。redis并不是为了作为数据库使用的，它更多地是一个高速存取器，一般用作缓存和类似场景。

如果你想找一个关系型数据库如mysql的替代者，推荐使用mongodb，支持海量数据，查询功能强大，数据类型支持广泛。目前已有一些团队在后台完全使用mongodb作为数据库。

**对比memcache**

* memcache是一个分布式的内存对象缓存系统，并不提供持久存储功能，而redis拥有持久化功能
* memcache数据存储基于LRU(简单说：最近、最少使用key会被剔除)，而redis则可以永久保存(服务一直运行情况下)
* memcache是多线程的（这是memcache优势之一），也就意味着阻塞情况少，而redis是单线程的，阻塞情况相对较多
* 两者性能上相差不大
* memcache只支持简单的k、v数据存储，而redis支持多种数据格式存储(列表字典)。
* memcache是多线程、非阻塞IO复用网络模型，而redis是单线程IO复用模型

redis通过WATCH命令实现乐观锁，作为WATCH命令的参数的键会受到Redis的监控，Redis能够检测到它们的变化。在执行EXEC命令之前，如果Redis检测到至少有一个键被修改了，那么整个事务便会中止运行，然后EXEC命令会返回一个nil值，提醒用户事务运行失败。

**Redis持久化是如何工作的？**   
什么是持久化？简单来讲就是将数据放到断电后数据不会丢失的设备中，也就是我们通常理解的硬盘上。

二 、Redis提供了RDB持久化和AOF持久化(保证持久化)

RDB机制的优势和略施

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘。

AOF文件保存过程

redis会将每一个收到的写命令都通过write函数追加到文件中(默认是 appendonly.aof)。Redis如何保持数据一致性??

23 MySQL主从复制原理

**主从形式**

mysql主从复制 灵活

* 一主一从
* 主主复制
* 一主多从---扩展系统读取的性能，因为读是在从库读取的；
* 多主一从---5.7开始支持
* 联级复制---

**用途及条件**

mysql主从复制用途

* 实时灾备，用于故障切换
* 读写分离，提供查询服务
* 备份，避免影响业务

主从部署必要条件：

* 主库开启binlog日志（设置log-bin参数）
* 主从server-id不同
* 从库服务器能连通主库

**主从原理**

mysql主从复制原理

从库生成两个线程，一个I/O线程，一个SQL线程；

i/o线程去请求主库的binlog，并将得到的binlog日志写到relay log（中继日志） 文件中；

主库会生成一个 log dump 线程，用来给从库 i/o线程传binlog；

SQL 线程，会读取relay log文件中的日志，并解析成具体操作，来实现主从的操作一致，而最终数据一致；

mysql主从复制存在的问题：

* 主库宕机后，数据可能丢失
* 从库只有一个sql Thread，主库写压力大，复制很可能延时

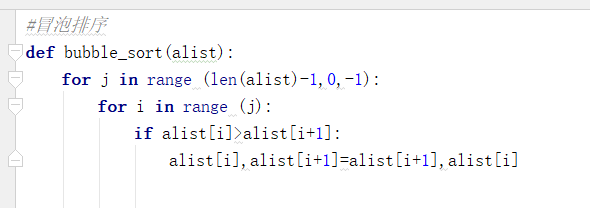
解决方法：

* 半同步复制---解决数据丢失的问题
* 并行复制----解决从库复制延迟的问题

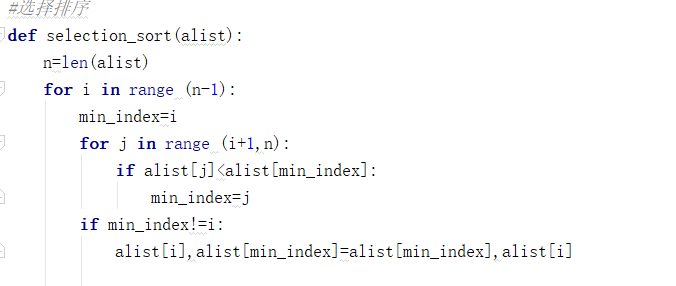
数据结构

1 排序：

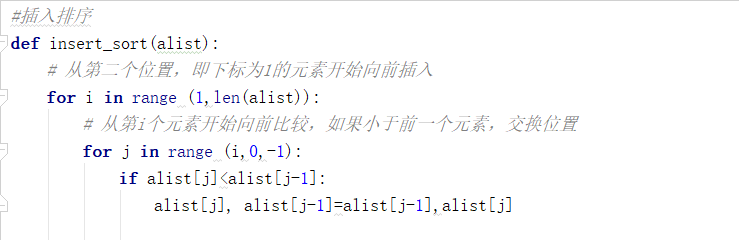
冒泡排序



选择排序



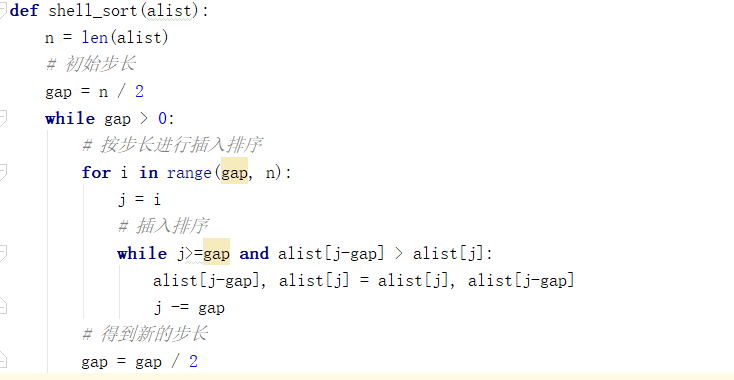
插入排序



快排

*#快排***def** quick\_sort(alist,start,end):  
 *# 递归的退出条件* **if** start>=end:  
 **return** *# 设定起始元素为要寻找位置的基准元素* mid=alist[start]  
 *# low为序列左边的由左向右移动的游标* low=start  
 *# high为序列右边的由右向左移动的游标* high=end  
 **while** low <high:  
 *# 如果low与high未重合，high指向的元素不比基准元素小，则high向左移动* **while** low<high **and** alist[high]>=mid:  
 high-=1  
 *# 将high指向的元素放到low的位置上* alist[low]=alist[high]  
 *# 如果low与high未重合，low指向的元素比基准元素小，则low向右移动* **while** low < high **and** alist[low]<mid:  
 low+=1  
 *# 将low指向的元素放到high的位置上* alist[high]=alist[low]  
 *# 将基准元素放到该位置* alist[low]=mid  
 *# 对基准元素左边的子序列进行快速排序* quick\_sort(alist,start,low-1)  
 *# 对基准元素右边的子序列进行快速排序* quick\_sort(alist,low+1,end)

希尔排序



归并排序

**def** merge\_sort(alist):  
 **if** len(alist) <= 1:  
 **return** alist  
 *# 二分分解* num = len(alist) / 2  
 left = merge\_sort(alist[:num]  
 right = merge\_sort(alist[num:])  
 *# 合并* **return** merge(left, right)  
**def** merge(left, right):  
 *'''合并操作，将两个有序数组left[]和right[]合并成一个大的有序数组'''  
 # left与right的下标指针* l, r = 0, 0  
 result = []  
 **while** l < len(left) **and** r < len(right):  
 **if** left[l] < right[r]:  
 result.append(left[l])  
 l += 1  
 **else**:  
 result.append(right[r])  
 r += 1  
 result += left[l:]  
 result += right[r:]  
 **return** result



**稳定性的定义**

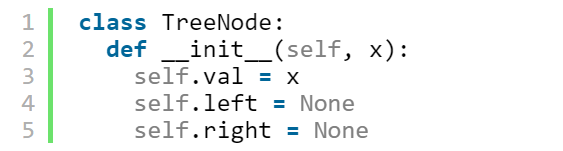
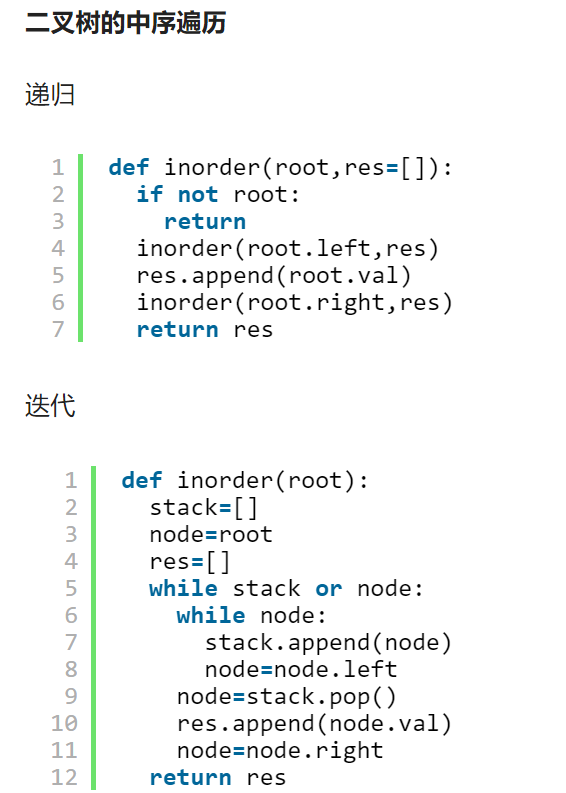
假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，ri=rj，且ri在rj之前，而在排序后的序列中，ri仍在rj之前，则称这种排序算法是稳定的；否则称为不稳定的。

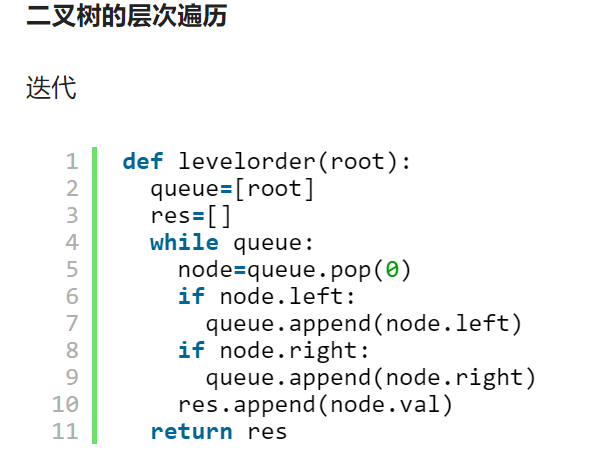
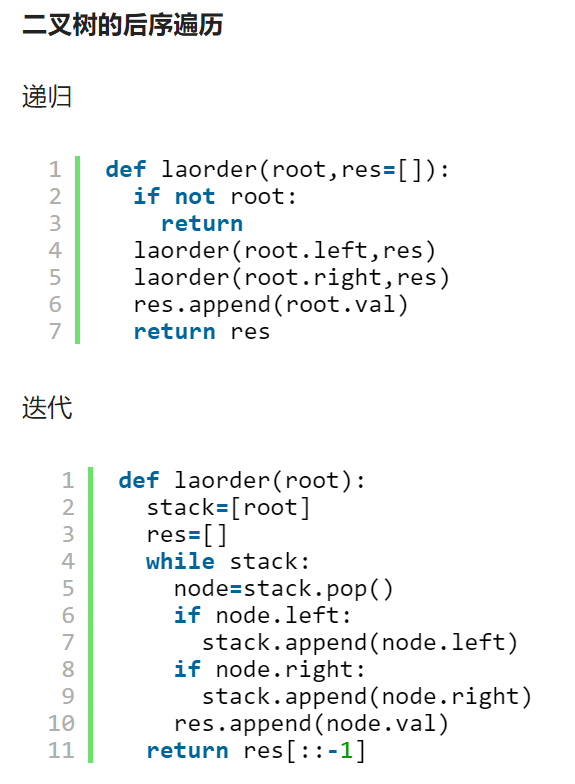
2 二分法查找

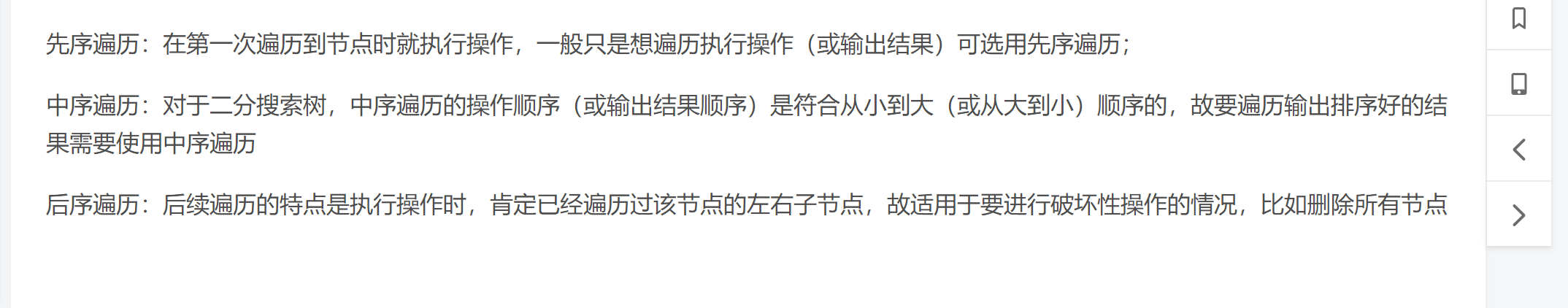
*# 非递归实现***def** binary\_search(alist, item):  
 first = 0  
 last = len(alist) - 1  
 **while** first <= last:  
 midpoint = (first + last) / 2  
 **if** alist[midpoint] == item:  
 **return True  
 elif** item < alist[midpoint]:  
 last = midpoint - 1  
 **else**:  
 first = midpoint + 1  
 **return False***#（递归实现）***def** binary\_search(alist, item):  
 **if** len(alist) == 0:  
 **return False  
 else**:  
 midpoint = len(alist) // 2  
 **if** alist[midpoint] == item:  
 **return True  
 else**:  
 **if** item < alist[midpoint]:  
 **return** binary\_search(alist[:midpoint], item)  
 **else**:  
 **return** binary\_search(alist[midpoint + 1:], item)

3 树算法

二叉树的定义：

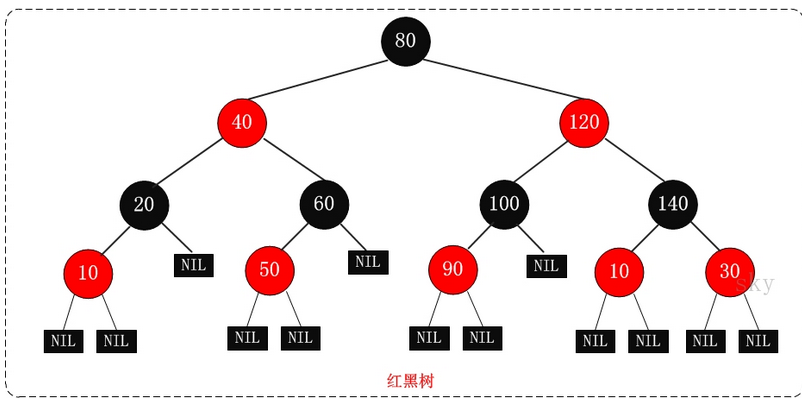




4 红黑树

**红黑树的特性**:  
**（1）每个节点或者是黑色，或者是红色。**  
**（2）根节点是黑色。**  
**（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]**  
**（4）如果一个节点是红色的，则它的子节点必须是黑色的。**  
**（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。**

**注意**：  
(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。  
(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

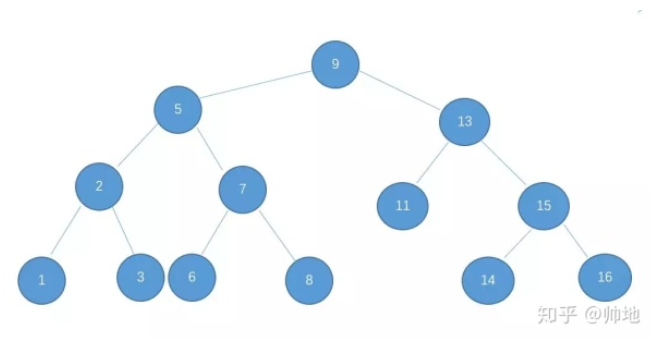


### 红黑树的应用

红黑树的应用比较广泛，主要是用它来存储有序的数据，它的时间复杂度是O(lgn)，效率非常之高。例如，Java集合中的[TreeSet](http://www.cnblogs.com/skywang12345/p/3311268.html)和[TreeMap](http://www.cnblogs.com/skywang12345/p/3310928.html)，C++ STL中的set、map，以及Linux虚拟内存的管理，都是通过红黑树去实现的。

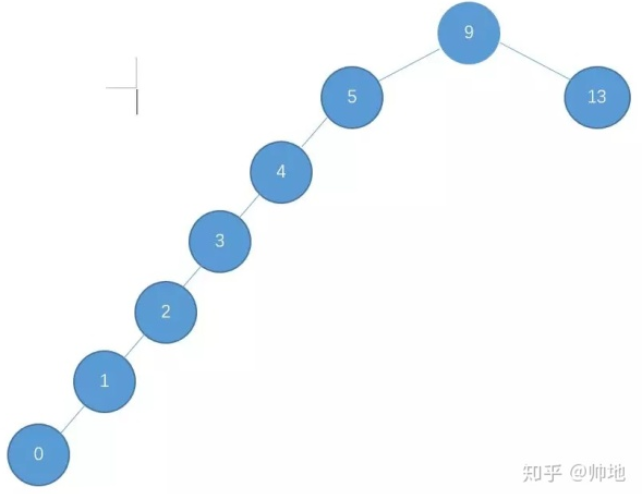
5 二叉搜索树

二叉查找树，相信大家都接触过，二叉查找树的特点就是左子树的节点值比父亲节点小，而右子树的节点值比父亲节点大，如图



基于二叉查找树的这种特点，我们在查找某个节点的时候，可以采取类似于**二分查找**的思想，快速找到某个节点。n 个节点的二叉查找树，正常的情况下，查找的时间复杂度为 O（logn）。

之所以说是**正常情况下**，是因为二叉查找树有可能出现一种极端的情况，例如



这种情况也是满足**二叉查找树**的条件，然而，此时的二叉查找树已经近似退化为一条链表，这样的二叉查找树的查找时间复杂度顿时变成了 O(n)，可想而知，我们必须不能让这种情况发生，为了解决这个问题，于是我们引申出了**平衡二叉树**。

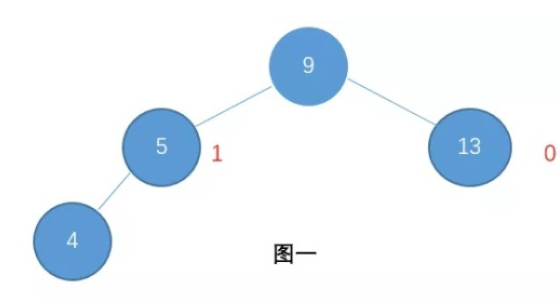
6 平衡二叉树

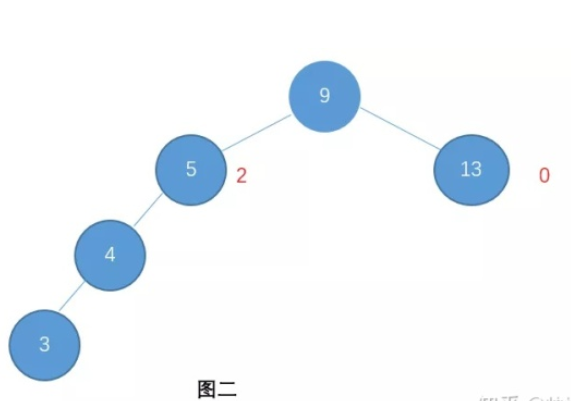
平衡二叉树就是为了解决二叉查找树退化成一颗链表而诞生了，平衡树具有如下特点

1、具有二叉查找树的全部特性。

2、每个节点的左子树和右子树的高度差至多等于1。

例如：图一就是一颗平衡树了，而图二则不是(节点右边标的是这个节点的高度)





对于图二，因为节点9的左孩子高度为2，而右孩子高度为0。他们之间的差值超过1了。对于图二，因为节点9的左孩子高度为2，而右孩子高度为0。他们之间的差值超过1了。

为什么有了平衡树还需要红黑树？

虽然平衡树解决了二叉查找树退化为近似链表的缺点，能够把查找时间控制在 O(logn)，不过却不是最佳的，因为平衡树要求**每个节点的左子树和右子树的高度差至多等于1**，这个要求实在是太严了，导致每次进行插入/删除节点的时候，几乎都会破坏平衡树的第二个规则，进而我们都需要通过**左旋**和**右旋**来进行调整，使之再次成为一颗符合要求的平衡树。

显然，如果在那种插入、删除很频繁的场景中，平衡树需要频繁着进行调整，这会使平衡树的性能大打折扣，为了解决这个问题，于是有了**红黑树**，红黑树具有如下特点：

1、具有二叉查找树的特点。

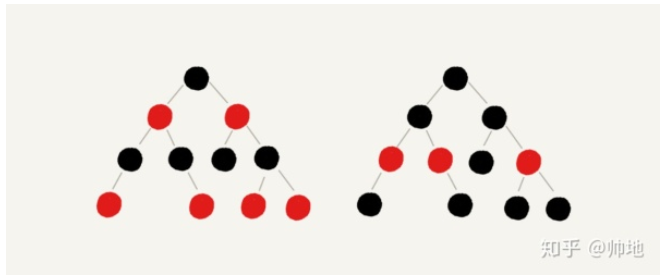
2、根节点是黑色的；

3、每个叶子节点都是黑色的空节点（NIL），也就是说，叶子节点不存数据。

4、任何相邻的节点都不能同时为红色，也就是说，红色节点是被黑色节点隔开的。

5、每个节点，从该节点到达其可达的叶子节点是所有路径，都包含相同数目的黑色节点。

例如下面的图片（注意，图片中黑色的、空的叶子节点没有画出）（图片来自极客时间）



不过，与平衡树不同的是，红黑树在插入、删除等操作，**不会像平衡树那样，频繁着破坏红黑树的规则，所以不需要频繁着调整**，这也是我们为什么大多数情况下使用红黑树的原因。

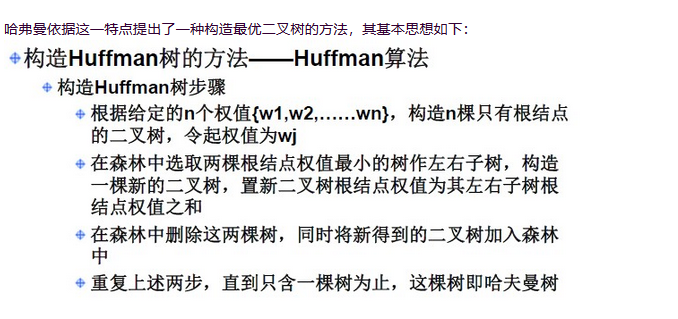
不过，如果你要说，单单在查找方面的效率的话，平衡树比红黑树快。

所以，我们也可以说，**红黑树是一种不大严格的平衡树**。也可以说是一个折中发方案。

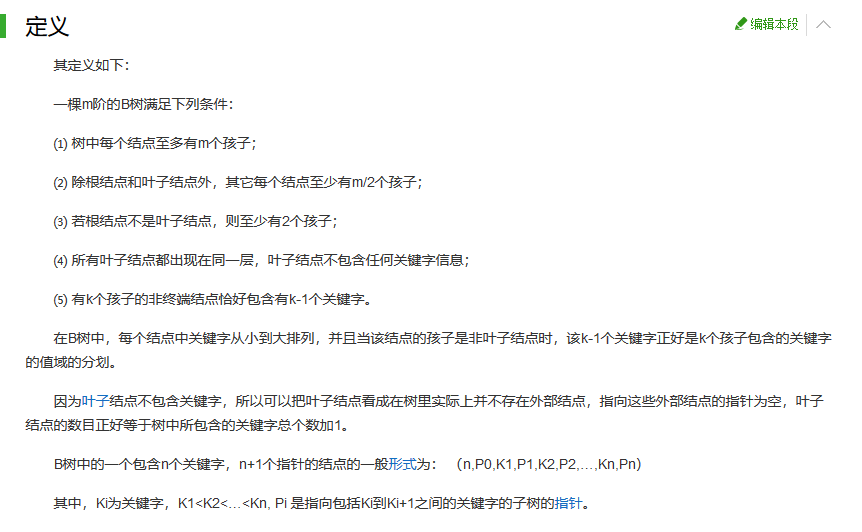
总结

所以，最后的答案是，平衡树是为了解决二叉查找树退化为链表的情况，而红黑树是为了解决平衡树在插入、删除等操作需要频繁调整的情况。

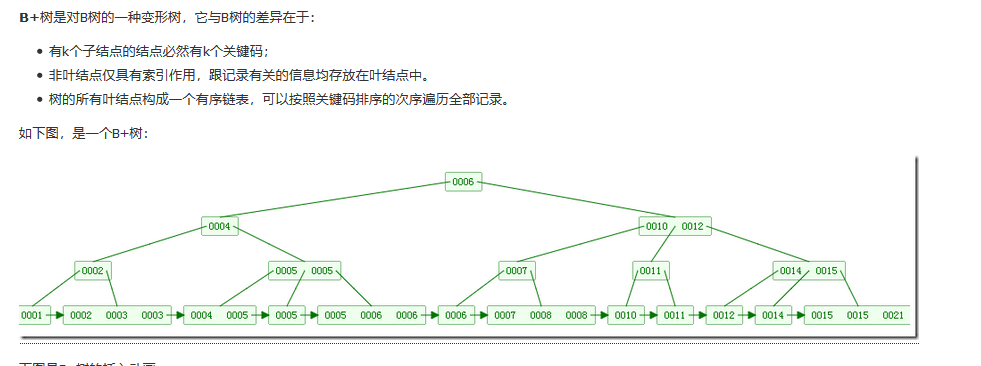
7 哈夫曼树



8 B树



9 B+树



B和B+树的区别在于，B+树的非叶子结点只包含导航信息，不包含实际的值，所有的叶子结点和相连的节点使用链表相连，便于区间查找和遍历。

B+ 树的优点在于：

* 由于B+树在内部节点上不包含数据信息，因此在内存页中能够存放更多的key。 数据存放的更加紧密，具有更好的空间局部性。因此访问叶子节点上关联的数据也具有更好的缓存命中率。
* B+树的叶子结点都是相链的，因此对整棵树的便利只需要一次线性遍历叶子结点即可。而且由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

但是B树也有优点，其优点在于，由于B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

10 数组和链表

* 数组是将元素在内存中连续存放，由于每个元素占用内存相同，可以通过下标迅速访问数组中任何元素。但是如果要在数组中增加一个元素，需要移动大量元素，在内存中空出一个元素的空间，然后将要增加的元素放在其中。同样的道理，如果想删除一个元素，同样需要移动大量元素去填掉被移动的元素。如果应用需要快速访问数据，很少或不插入和删除元素，就应该用数组。
* 链表恰好相反，链表中的元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起。比如：上一个元素有个指针指到下一个元素，以此类推，直到最后一个元素。如果要访问链表中一个元素，需要从第一个元素开始，一直找到需要的元素位置。但是增加和删除一个元素对于链表数据结构就非常简单了，只要修改元素中的指针就可以了。如果应用需要经常插入和删除元素你就需要用链表数据结构了。

11 解决哈希冲突方法

开放定址法

这种方法也称再散列法，其基本思想是：当关键字key的哈希地址p=H（key）出现冲突时，以p为基础，产生另一个哈希地址p1，如果p1仍然冲突，再以p为基础，产生另一个哈希地址p2，…，直到找出一个不冲突的哈希地址pi ，将相应元素存入其中。这种方法有一个通用的再散列函数形式：

Hi=（H（key）+di）% m   i=1，2，…，n

其中H（key）为哈希函数，m 为表长，di称为增量序列。增量序列的取值方式不同，相应的再散列方式也不同。主要有以下三种：

线性探测再散列

dii=1，2，3，…，m-1

这种方法的特点是：冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表。

二次探测再散列

di=12，-12，22，-22，…，k2，-k2    ( k<=m/2 )

这种方法的特点是：冲突发生时，在表的左右进行跳跃式探测，比较灵活。

伪随机探测再散列

di=伪随机数序列。

具体实现时，应建立一个伪随机数发生器，（如i=(i+p) % m），并给定一个随机数做起点。

例如，已知哈希表长度m=11，哈希函数为：H（key）= key  %  11，则H（47）=3，H（26）=4，H（60）=5，假设下一个关键字为69，则H（69）=3，与47冲突。

如果用线性探测再散列处理冲突，下一个哈希地址为H1=（3 + 1）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 + 2）% 11 = 5，还是冲突，继续找下一个哈希地址为H3=（3 + 3）% 11 = 6，此时不再冲突，将69填入5号单元。

如果用二次探测再散列处理冲突，下一个哈希地址为H1=（3 + 12）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 - 12）% 11 = 2，此时不再冲突，将69填入2号单元。

如果用伪随机探测再散列处理冲突，且伪随机数序列为：2，5，9，……..，则下一个哈希地址为H1=（3 + 2）% 11 = 5，仍然冲突，再找下一个哈希地址为H2=（3 + 5）% 11 = 8，此时不再冲突，将69填入8号单元。

再哈希法

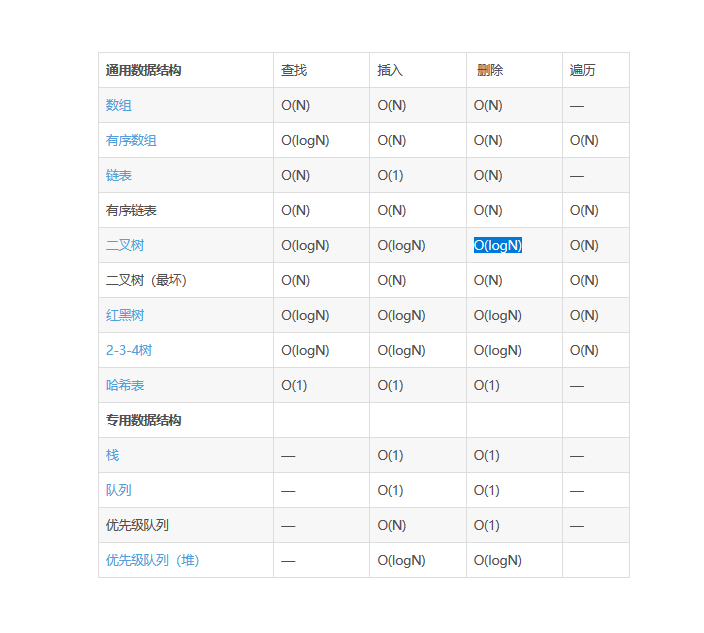
这种方法是同时构造多个不同的哈希函数：

Hi=RH1（key）  i=1，2，…，k

当哈希地址Hi=RH1（key）发生冲突时，再计算Hi=RH2（key）……，直到冲突不再产生。这种方法不易产生聚集，但增加了计算时间。

链地址法

这种方法的基本思想是将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。





PYTHON

1 Python的数据类型：

Python3 中有六个标准的数据类型：

* Number（数字：int、float、bool、complex（复数）：1、Python可以同时为多个变量赋值，如a, b = 1, 2。2、一个变量可以通过赋值指向不同类型的对象。3、数值的除法包含两个运算符：**/**返回一个浮点数，**//**返回一个整数。4、在混合计算时，Python会把整型转换成为浮点数。
* String（字符串）：Python中的字符串用单引号**'**或双引号**"**括起来，同时使用反斜杠**\**转义特殊字符。
* List（列表）：1、List写在方括号之间，元素用逗号隔开。2、和字符串一样，list可以被索引和切片。3、List可以使用+操作符进行拼接。4、List中的元素是可以改变的
* Tuple（元组）：1、与字符串一样，元组的元素不能修改。2、元组也可以被索引和切片，方法一样。3、注意构造包含 0 或 1 个元素的元组的特殊语法规则。4、元组也可以使用+操作符进行拼接。
* Set（集合）：集合（set）是由一个或数个形态各异的大小整体组成的，构成集合的事物或对象称作元素或是成员。基本功能是进行成员关系测试和删除重复元素。可以使用大括号**{ }**或者**set()**函数创建集合，注意：创建一个空集合必须用**set()**而不是**{ }**，因为**{ }**是用来创建一个空字典。
* Dictionary（字典）：1、字典是一种映射类型，它的元素是键值对。2、字典的关键字必须为不可变类型，且不能重复。3、创建空字典使用**{ }**。

Python3 的六个标准数据类型中：

* **不可变数据（3 个）：**Number（数字）、String（字符串）、Tuple（元组）；
* **可变数据（3 个）：**List（列表）、Dictionary（字典）、Set（集合）。

类型之间的转换  
 int(x,base=10)x字符串或数字，base进制数，默认十进制 浮点转为整数  
 float 整数转换为浮点型  
 complex(1,2) 转换为复数  
 str(10)将对象转换为字符串  
 repe()将对象转换为表达式字符串  
 repr(dict)将对象转换为表达式字符串  
 eval(str)用来计算在字符串中有效的python表达式，返回一个对象  
 tuple(listi)将列表转化为元组  
 list()将元组转换为列表  
 set转换集合

Python 字典(Dictionary) update()方法：Python 字典(Dictionary) update() 函数把字典dict2的键/值对更新到dict里。有相同的键会直接替换成 update 的值。

2 Python中is与==的使用区别详解

1.== 是比较两个对象的内容是否相等，即两个对象的“值“”是否相等，不管两者在内存中的引用地址是否一样。

2.is 比较的是两个实例对象是不是完全相同，它们是不是同一个对象，占用的内存地址是否相同。即is比较两个条件：1.内容相同。2.内存中地址相同

3 常用Python第三方库

sklern、numpy、pandas、画图的matplotlib

4 python 深拷贝和浅拷贝

1．深浅拷贝都是对源对象的复制，占用不同的内存空间。

2．不可变类型的对象，对于深浅拷贝毫无影响，最终的地址值和值都是相等的。

3．可变类型：

浅拷贝： 值相等，地址相等

copy浅拷贝：将对象的引用复制给另一个对象

deepcopy深拷贝：值相等，地址不相等 完全拷贝，完全独立于原对象

5 Python 中 pass 语句的作用是什么？

在编写代码时只写框架思路，具体实现还未编写就可以用 pass 进行占位，使程序不报错，不会进行任何操作。

6 什么是 lambda 函数？ 有什么好处？

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数

lambda 函数比较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用，连名字都很随意的情况下；

匿名函数，一般用来给 filter，map这样的函数式编程服务; 作为回调函数，传递给某些应用，比如消息处理

7 Python 定时任务：

1 第一种办法是最简单又最暴力。那就是在一个死循环中，使用线程睡眠函数 sleep()。

2 Python 标准库 threading 中有个 Timer 类。它会新启动一个线程来执行定时任务，所以它是非阻塞函式。使用标准库中sched模块。sched 是事件调度器，它通过 scheduler 类来调度事件，从而达到定时执行任务的效果。

3 PScheduler的全称是Advanced Python Scheduler。它是一个轻量级的 Python 定时任务调度框架。

4 APScheduler 支持三种调度任务：固定时间间隔，固定时间点（日期），Linux 下的 Crontab 命令。同时，它还支持异步执行、后台执行调度任务。

8 启动和关闭程序





9 python2和python3的区别？

print从语句变成函数

python2的range返回的是列表，但是Python3返回的是迭代器，不占用内存空间，这点和python2的xrange是一样的。

字符串：原来python2中是以8bit存储，但是现在python3中是以16bit存储

try except语句发生变化

打开文件发生变化：原来可以用file、open两种方式，现在只能用open这种方式

从键盘录入一个字符串，原来用raw\_input(),现在是input()，python2的input()输入的是一个int型数据。

python3字符串以unicode编码存储，当写入二进制文件时，字符串无法直接写入（或读取），必须以某种方式的编码为字节序列后，方可写入。字符串.encode("utf8")为bytes（字节）；bytes.decode("utf8")为字符串。

chr(k)与ord(c)

除法运算：python2.4.2以前，10/3=3；python3.0后，10/3=3.333333

import thread 问题：python3中需要在Python2的Thread模块前添加\_thread，否则会出错。

10 Python 两种执行方式——python和python -m 的区别

Python [xxx.py](http://xxx.py/) # 直接运行

python -m xxx # 相当于import，叫做当做模块来启动

两者主要区别在于sys.path不同：

直接运行会将该脚本所在目录添加至sys.path，

当做模块启动则会将当前运行命令的路径添加至sys.path

11 Python3 进程、线程、协程

进程是资源分配的基本单位，线程是cpu调度的基本单位，比如说，一个程序启动后就开启了一个进程，进程开启后会分配进程号和资源，一个进程里可以运行多个线程，这些线程之间共享资源。

在python中，因为有GIL锁的存在，它保证一个CPU一次只能运行一个线程，所以计算密集型的任务因为要用到CPU，所以用多线程没有作用，应该用多进程，对于IO密集型，可以用多线程，比如说网络请求，访问数据库数据这些，都可以用到多线程。

线程和协程？

协程是更轻量级的，一个线程可以有多个协程，线程是cpu层，是内核级的，而协程是由程序员自己控制的，是用户级的。协程可以由gevent模块和yield来控制。

12 什么是面向切面编程（AOP）？

面向切面编程（AOP是Aspect Oriented Program的首字母缩写），因为面向对象会将功能不同的模块分散到不同的对象中去，这样就降低了代码的复杂程度，使得不同的类可以重用，但是在分散代码的同时，也增加了代码的重复性。

比如两个类都需要日志，那么就需要在两个类中都加入日志。这样就重复了。所以我们就需要动态地将日志相关的代码切入到类的指定方法中，这就是面向切面的编程。

通过装饰器将一段代码插入进去，用于扩展原有的类的功能，就是面向切面的编程。

13 以及Python装饰器？

装饰器是一种设计模式，经常被用于有切面需求的场景，较为经典的有插入日志、性能测试、事务处理，授权等。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量函数中与函数功能本身无关的雷同代码并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能。

def use\_logging(func):

def wrapper():

logging.warn("%s is running" % func.\_\_name\_\_)

return func() # 把 foo 当做参数传递进来时，执行func()就相当于执行foo()

return wrapper

def foo():

print('i am foo')

foo = use\_logging(foo) # 因为装饰器 use\_logging(foo) 返回的时函数对象 wrapper，这条语句相当于 foo = wrapper

foo() # 执行foo()就相当于执行 wrapper()

use\_logging 就是一个装饰器，它一个普通的函数，它把执行真正业务逻辑的函数 func 包裹在其中，看起来像 foo 被 use\_logging 装饰了一样，use\_logging 返回的也是一个函数，这个函数的名字叫 wrapper。在这个例子中，函数进入和退出时，被称为一个横切面，这种编程方式被称为面向切面的编程。

**@ 语法糖**

如果你接触 Python 有一段时间了的话，想必你对 @ 符号一定不陌生了，没错 @ 符号就是装饰器的语法糖，它放在函数开始定义的地方，这样就可以省略最后一步再次赋值的操作。

def use\_logging(func):

def wrapper():

logging.warn("%s is running" % func.\_\_name\_\_)

return func()

return wrapper

@use\_logging

def foo():

print("i am foo")

foo()

如上所示，有了 @ ，我们就可以省去foo = use\_logging(foo)这一句了，直接调用 foo() 即可得到想要的结果。你们看到了没有，foo() 函数不需要做任何修改，只需在定义的地方加上装饰器，调用的时候还是和以前一样，如果我们有其他的类似函数，我们可以继续调用装饰器来修饰函数，而不用重复修改函数或者增加新的封装。这样，我们就提高了程序的可重复利用性，并增加了程序的可读性。装饰器在 Python 使用如此方便都要归因于 Python 的函数能像普通的对象一样能作为参数传递给其他函数，可以被赋值给其他变量，可以作为返回值，可以被定义在另外一个函数内。

一个函数还可以同时定义多个装饰器它的执行顺序是从里到外，最先调用最里层的装饰器，最后调用最外层的装饰器

13 迭代器与生成器？

迭代器：任何实现了\_\_iter\_\_和\_\_next\_\_()方法的对象都是迭代器，\_\_iter\_\_返回迭代器自身，\_\_next\_\_返回容器中的下一个值，如果容器中没有更多元素了，则抛出StopIteration异常。

Python标准库中存在着一些可迭代对象，例如:list, tuple, dict, set, str等。

class MyNumbers():  
 def \_\_iter\_\_(self):  
 self.a = 1  
 return self  
 def \_\_next\_\_(self):  
 if self.a<=20:  
 x= self.a  
 self.a+=1  
 return x  
 else:  
 raise StopIteration  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 my\_num = MyNumbers()  
 for i in my\_num:  
 print(i)

生成器：Python中的生成器主要分为两种类型：

生成器函数(generator function)返回得到的生成器:包含yield关键字的函数称为生成器函数：使用了 yield 的函数被称为生成器（generator）。在Python中，这种一边循环一边计算的机制，称为生成器：生成器是一个特殊的程序，可以被用作控制循环的迭代行为，python中生成器是迭代器的一种，使用yield返回值函数，每次调用yield会暂停，而可以使用next()函数和send()函数恢复生成器。

# 使用迭代器生成斐波那契数列  
def fibonacci(n):  
 a,b = 0,1  
 count = 0  
 while count<=n:  
 yield a  
 a,b = b,a+b  
 count+=1  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 f = fibonacci(1)  
 while True:  
 try:  
 print(next(f),end=" ")  
 except StopIteration:  
 sys.exit()

**利用生成器代替可迭代中的\_\_iter\_\_迭代器**

在迭代器模式章节中，我们在可迭代IterText中的\_\_iter\_\_返回迭代器IteratorText实例，然而使用生成器的方式会使代码更加优雅。

class IteraText:  
 def \_\_init\_\_(self,text):  
 self.text = text  
 def \_\_iter\_\_(self):  
 for letter in self.text:  
 yield letter

因为yield存在于\_\_iter\_\_，因此\_\_iter\_\_变成了生成器函数，调用它测返回一个生成器，同时生成器又实现了迭代器协议，因此IterText满足了可迭代的需求。

1. **Python中的鸭子类型？**

参考链接：<https://foofish.net/dynamic_type_and_duck_type.html>

python是动态语言，动态语言中经常提到鸭子类型，所谓鸭子类型就是：如果走起路来像鸭子，叫起来也像鸭子，那么它就是鸭子（If it walks like a duck and quacks like a duck, it must be a duck）。鸭子类型是编程语言中动态类型语言中的一种设计风格，一个对象的特征不是由父类决定，而是通过对象的方法决定的。例如，当一个对象有\_\_iter\_\_()和\_\_next\_\_()方法，并且提供了正确的实现，那么它就是一个迭代器；一个对象若有push和pop方法，并且这些方法提供了正确的实现，它就可以被当作栈来使用。一个对象如果有length属性，也可以依照下标来存取属性（最好还要拥有slice和splice等方法），这个对象就可以被当作数组来使用。这就是python的鸭子类型

1. Python代码执行原理

解释器执行，自上而下，解释一行，执行一行

16 Python导入一个包的过程？

根据sys.path查找导入的模块，默认从当前路径下查找.如果找到了就开辟一个命名空间，来执行这个模块把这个模块用到的名字都收录进入命名空间创建变量名指向这个命名空间（变量名和模块名称一致）

注意：

1，模块不能重复导入

2，原文件和导入模块的内存空间始终是隔离的

3，模块名称必须符合标识符命名规范

17 解释python的对象---python源码剖析！！！

python中一切皆对象，一个对象就是一片被分配的内存空间，这些内存可能是连续的，也可能是离散的，在这片内存空间上存储着一系列的数据以及可对这些数据进行修改或读取操作的一系列代码。

18 **什么是上下文管理器？**

参考链接：<https://www.cnblogs.com/wilber2013/p/4638967.html>

在使用Python编程中，可以会经常碰到这种情况：有一个特殊的语句块，在执行这个语句块之前需要先执行一些准备动作；当语句块执行完成后，需要继续执行一些收尾动作。

例如：当需要操作文件或数据库的时候，首先需要获取文件句柄或者数据库连接对象，当执行完相应的操作后，需要执行释放文件句柄或者关闭数据库连接的动作。

又如，当多线程程序需要访问临界资源的时候，线程首先需要获取互斥锁，当执行完成并准备退出临界区的时候，需要释放互斥锁。

对于这些情况，Python中提供了上下文管理器（Context Manager）的概念，可以通过上下文管理器来定义/控制代码块执行前的准备动作，以及执行后的收尾动作。

with关键字来使用上下文管理器。在一个类中实现了\_\_enter\_\_和\_\_exit\_\_两个内置方法，就是上下文管理器

19 什么是闭包？

在函数内部再定义一个函数，并且这个里面的函数用到了外面的函数的变量，如果要在内部更改外面函数的变量，需要用到nonlocal关键字，那么这个函数以及用到的一些变量称之为闭包。如果要更改全局变量，使用global关键字。函数装饰器是在闭包的基础上实现的。

20 Python中一般类都继承object，那么object的父类是什么？

是type

21 Python中的垃圾回收机制

Python中的垃圾回收是以引用计数为主，**标记-清除**和**分代收集**两种机制为辅的策略。引用计数的缺陷是循环引用的问题。  
在Python中，如果一个对象的引用数为0，Python虚拟机就会回收这个对象的内存。

由于存在循环引用问题，这样循环引用的对象在任何时候都不为0，那么就没办法使用引用计数法回收这些内存，因此引入了分代收集机制

Python将对象分为三代，一个代就是一个链表，实际上python维护了三个链表，第0代、第1代、第2代，一开始，对象在创建的时候，放在一代中，如果在一次一代的垃圾检查中，该对象存活下来，就会被放到二代中，同理在一次二代的垃圾检查中，该对象存活下来，就会被放到三代中。越是年老的一代，回收的越不频繁。

有三种情况会触发垃圾回收：

1. 调用gc.collect(),需要先导入gc模块。
2. 当gc模块的计数器达到阀值的时候。
3. 程序退出的时候。

gc模块

gc模块提供一个接口给开发者设置垃圾回收的选项。上面说到，采用引用计数的方法管理内存的一个缺陷是循环引用，而gc模块的一个主要功能就是解决循环引用的问题。

**标记清除**

标记清除（Mark—Sweep）』算法是一种基于追踪回收（tracing GC）技术实现的垃圾回收算法。它分为两个阶段：第一阶段是标记阶段，GC会把所有的『活动对象』打上标记，第二阶段是把那些没有标记的对象『非活动对象』进行回收。那么GC又是如何判断哪些是活动对象哪些是非活动对象的呢？

对象之间通过引用（指针）连在一起，构成一个有向图，对象构成这个有向图的节点，而引用关系构成这个有向图的边。从根对象（root object）出发，沿着有向边遍历对象，可达的（reachable）对象标记为活动对象，不可达的对象就是要被清除的非活动对象。根对象就是全局变量、调用栈、寄存器。 mark-sweepg 在上图中，我们把小黑圈视为全局变量，也就是把它作为root object，从小黑圈出发，对象1可直达，那么它将被标记，对象2、3可间接到达也会被标记，而4和5不可达，那么1、2、3就是活动对象，4和5是非活动对象会被GC回收。

标记清除算法作为Python的辅助垃圾收集技术主要处理的是一些容器对象，比如list、dict、tuple，instance等，因为对于字符串、数值对象是不可能造成循环引用问题。Python使用一个双向链表将这些容器对象组织起来。不过，这种简单粗暴的标记清除算法也有明显的缺点：清除非活动的对象前它必须顺序扫描整个堆内存，哪怕只剩下小部分活动对象也要扫描所有对象。

22 Python的单例设计模式

使用new关键字：

class Singleton(object):  
 \_instance = None  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if cls.\_instance is None:  
 cls.\_instance = object.\_\_new\_\_(cls)  
 return cls.\_instance  
 def \_\_init\_\_(self):  
 pass

23 json字符串

# json字符串  
import json  
# 将json字符串转化为字典  
dict = json.load("json字符串")  
# 将字典转化为json字符串  
json\_str = json.dumps(dict())

24 eval()函数

eval() 函数用来执行一个字符串表达式，并返回表达式的值。

# python的eval()函数用来执行一个字符串表达式，并返回表达式的值  
print(eval("1+2")) # 3

25 select、poll、epoll的区别？

select，poll，epoll都是IO多路复用的机制。I/O多路复用就通过一种机制，可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。**但select，poll，epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的**，而异步I/O则无需自己负责进行读写，异步I/O的实现会负责把数据从内核拷贝到用户空间。

**select的几大缺点：**

**（1）每次调用select，都需要把fd集合从用户态拷贝到内核态，这个开销在fd很多时会很大**

**（2）同时每次调用select都需要在内核遍历传递进来的所有fd，这个开销在fd很多时也很大**

**（3）select支持的文件描述符数量太小了，默认是1024**

poll除了没有了最大连接数的缺点，其他都和select一样

（1）select，poll实现需要自己不断轮询所有fd集合，直到设备就绪，期间可能要睡眠和唤醒多次交替。而epoll其实也需要调用epoll\_wait不断轮询就绪链表，期间也可能多次睡眠和唤醒交替，但是它是设备就绪时，调用回调函数，把就绪fd放入就绪链表中，并唤醒在epoll\_wait中进入睡眠的进程。虽然都要睡眠和交替，但是select和poll在“醒着”的时候要遍历整个fd集合，而epoll在“醒着”的时候只要判断一下就绪链表是否为空就行了，这节省了大量的CPU时间。这就是回调机制带来的性能提升。

（2）select，poll每次调用都要把fd集合从用户态往内核态拷贝一次，并且要把current往设备等待队列中挂一次，而epoll只要一次拷贝，而且把current往等待队列上挂也只挂一次（在epoll\_wait的开始，注意这里的等待队列并不是设备等待队列，只是一个epoll内部定义的等待队列）。这也能节省不少的开销。

26 列表和元组的区别？

元组一旦被赋值，值不可以被改变，一旦改变就会出错；但是列表可以任意的更改。其次的区别是他们用不同的符号表示，复制的时候，列表用方括号[]，而元组用小括号（）。

27 python之seek()和tell()详解

1、seek()函数是Python中操作文件游标移动操作的函数

用法如下

seek(offset,whence=0)

offset：开始的偏移量，也就是代表需要移动偏移的字节数

whence：给offset参数一个定义，表示要从哪个位置开始偏移；0代表从文件开头开始算起，1代表从当前位置开始算起，2代表从文件末尾算起。

1. tell()返回文件中的位置

readline(n):读入若干行，n代表读入的最长字节数。

readlines() :读入所有行的内容

read读入所有行的内容

28 Python解释器

当我们编写Python代码时，我们得到的是一个包含Python代码的以.py为扩展名的文本文件。要运行代码，就需要Python解释器去执行.py文件。

CPython: 这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。CPython是使用最广的Python解释器。

IPython

IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。好比很多国产浏览器虽然外观不同，但内核其实都是调用了IE。CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。

Jython:Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。

29 Python的内存管理机制：[引入计数](https://www.cnblogs.com/geaozhang/p/7111961.html" \l "yinyongjishu)、[垃圾回收](https://www.cnblogs.com/geaozhang/p/7111961.html#lajihuishou)、[内存池机制](https://www.cnblogs.com/geaozhang/p/7111961.html#neicunchijizhi)

一 垃圾回收：从基本原理上，当Python的某个对象的引用计数降为0时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾了。

二 在Python中，每个对象都有存有指向该对象的引用总数，即引用计数(reference count)。我们可以使用sys包中的getrefcount()，来查看某个对象的引用计数。

三 对象引用对象，是Python最基本的构成方式。即使是a = 1这一赋值方式，实际上是让词典的一个键值"a"的元素引用整数对象1。该词典对象用于记录所有的全局引用。该词典引用了整数对象1。我们可以通过内置函数globals()来查看该词典。

四 分代回收 Python同时采用了分代(generation)回收的策略。这一策略的基本假设是，存活时间越久的对象，越不可能在后面的程序中变成垃圾。我们的程序往往会产生大量的对象，许多对象很快产生和消失，但也有一些对象长期被使用。出于信任和效率，对于这样一些“长寿”对象，我们相信它们的用处，所以减少在垃圾回收中扫描它们的频率。

Python的内存机制以金字塔行，-1，-2层主要有操作系统进行操作，

第0层是C中的malloc，free等内存分配和释放函数进行操作；

第1层和第2层是内存池，有Python的接口函数PyMem\_Malloc函数实现，当对象小于256K时有该层直接分配内存；

第3层是最上层，也就是我们对Python对象的直接操作；

30 Python 中如何实现多线程？

线程是轻量级的进程，多线程允许一次执行多个线程。众所周知，Python 是一种多线程语言，它有一个多线程包。

GIL（全局解释器锁）确保一次执行单个线程。一个线程保存GIL并在将其传递给下一个线程之前执行一些操作，这就产生了并行执行的错觉。但实际上，只是线程轮流在 CPU 上。当然，所有传递都会增加执行的开销。

31 python 类的使用和初始化

虽然我们可以自由地给一个实例绑定各种属性，但是，现实世界中，一种类型的实例应该拥有相同名字的属性。例如，Person类应该在创建的时候就拥有 name、gender 和 birth 属性，怎么办？

在定义 Person 类时，可以为Person类添加一个特殊的\_\_init\_\_()方法，当创建实例时，\_\_init\_\_()方法被自动调用，我们就能在此为每个实例都统一加上以下属性：

class Person(object):

def \_\_init\_\_(self, name, gender, birth):

self.name = name

self.gender = gender

self.birth = birth

\_\_init\_\_() 方法的第一个参数必须是 self（也可以用别的名字，但建议使用习惯用法），后续参数则可以自由指定，和定义函数没有任何区别。

相应地，创建实例时，就必须要提供除 self 以外的参数：

xiaoming = Person('Xiao Ming', 'Male', '1991-1-1')

xiaohong = Person('Xiao Hong', 'Female', '1992-2-2')

有了\_\_init\_\_()方法，每个Person实例在创建时，都会有 name、gender 和 birth 这3个属性，并且，被赋予不同的属性值，访问属性使用.操作符：

print xiaoming.name

# 输出 'Xiao Ming'

print xiaohong.birth

# 输出 '1992-2-2'

要特别注意的是，初学者定义\_\_init\_\_()方法常常忘记了 self 参数：

>>> class Person(object):

... def \_\_init\_\_(name, gender, birth):

... pass

>>> xiaoming = Person('Xiao Ming', 'Male', '1990-1-1')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: \_\_init\_\_() takes exactly 3 arguments (4 given)

这会导致创建失败或运行不正常，因为第一个参数name被Python解释器传入了实例的引用，从而导致整个方法的调用参数位置全部没有对上。

任务

请定义Person类的\_\_init\_\_方法，除了接受 name、gender 和 birth 外，还可接受任意关键字参数，并把他们都作为属性赋值给实例

?不会了怎么办

要定义关键字参数，使用 \*\*kw；

除了可以直接使用self.name = 'xxx'设置一个属性外，还可以通过 set attr (self, 'name', 'xxx') 设置属性。

32 [python文件的打开和读取](https://www.cnblogs.com/hqd2008/p/7637577.html)

**(**1) f = open("文件名"，"打开方式")

centent = f.read(size)     f.readline(size)   f.readlines()

f.close()

打开方式: r-->只读 w-->只写 r+ -->读写 w+ -->读写 a -->追加写 a+ -->追加读写 rb 以二进制格式打开文件，用于只读

Wb 以二进制格式打开文件，用于写入

文件以w+打开后，读取出来的内容为空；而以r+打开，写入的内容为追加。

f.read() 读取全部文件内容

f.read(size) 读取size的字符 #size为读取的长度，byte为单位

f.readline(size)读取size行的字符

f.readlines()读取所有的字符，并按行返回list

当要读取内容并循环输出时，推荐使用：for i in f:

直接读取f，此时f为迭代器，可用于大内容的文件读取

(2) With open("文件名"，"打开方式",conding="编码格式")  as f:

Pass

with 的作用就是调用close（）方法

此方法不用关闭文件

33 with 的用法(上下文管理器)

**一、With语句是什么?**

有一些任务，可能事先需要设置，事后做清理工作。对于这种场景，Python的with语句提供了一种非常方便的处理方式。一个很好的例子是文件处理，你需要获取一个文件句柄，从文件中读取数据，然后关闭文件句柄。

**二 with如何工作?**

**紧跟with后面的语句被求值后，返回对象的 \_\_enter\_\_() 方法被调用，这个方法的返回值将被赋值给as后面的变量。   
当with后面的代码块全部被执行完之后，将调用前面返回对象的 \_\_exit\_\_()方法。**

**因此，Python的with语句是提供一个有效的机制，让代码更简练，同时在异常产生时，清理工作更简单。**

**三、相关术语**

**上下文管理协议（Context Management Protocol）：包含方法 \_\_enter\_\_() 和 \_\_exit\_\_()，支持该协议的对象要实现这两个方法。  
上下文管理器（Context Manager）：支持上下文管理协议的对象，这种对象实现了\_\_enter\_\_() 和 \_\_exit\_\_() 方法。上下文管理器定义执行 with 语句时要建立的运行时上下文，负责执行 with 语句块上下文中的进入与退出操作。通常使用 with 语句调用上下文管理器，也可以通过直接调用其方法来使用。  
运行时上下文（runtime context）：由上下文管理器创建，通过上下文管理器的 \_\_enter\_\_() 和\_\_exit\_\_() 方法实现，\_\_enter\_\_() 方法在语句体执行之前进入运行时上下文，\_\_exit\_\_() 在语句体执行完后从运行时上下文退出。with 语句支持运行时上下文这一概念。   
上下文表达式（Context Expression）：with 语句中跟在关键字 with 之后的表达式，该表达式要返回一个上下文管理器对象。   
语句体（with-body）：with 语句包裹起来的代码块，在执行语句体之前会调用上下文管理器的 \_\_enter\_\_() 方法，执行完语句体之后会执行\_\_exit\_\_() 方法。**

**34 python \* 和\*\* 的区别**

**1. 算数运算**

**\* 代表乘法**

**\*\* 代表乘方**

1. **函数形参**

\*args 和 \*\*kwargs 主要用于函数定义。

**你可以将不定数量的参数传递给一个函数。不定的意思是：预先并不知道, 函数使用者会传递多少个参数给你, 所以在这个场景下使用这两个关键字。其实并不是必须写成 \*args 和 \*\*kwargs。\*(星号) 才是必须的. 你也可以写成 \*ar 和 \*\*k 。而写成 \*args 和\*\*kwargs 只是一个通俗的命名约定。**

**python函数传递参数的方式有两种：**

**位置参数（positional argument）**

**关键词参数（keyword argument）**

**\*args 与 \*\*kwargs 的区别，两者都是 python 中的可变参数：**

**\*args 表示任何多个无名参数，它本质是一个 tuple**

**\*\*kwargs 表示关键字参数，它本质上是一个 dict**

**如果同时使用 \*args 和 \*\*kwargs 时，必须 \*args 参数列要在 \*\*kwargs 之前。**

**3. 函数实参**

**如果函数的形参是定长参数，也可以使用 \*args 和 \*\*kwargs 调用函数，类似对元组和字典进行解引用：**

**4. 序列解包**

[**序列解包**](https://blog.csdn.net/yilovexing/article/details/80576788)**往期博客有写过，这里只列出一个例子，序列解包没有 \*\*。**

\* 获取的值默认为 list # 获取剩余部分 # 获取中间部分 # 如果左值比右值要多，那么带 \* 的变量默认为空

35 [Python之面向过程和面向对象的区别](https://www.cnblogs.com/schut/p/8541547.html)

**一、面向过程**

**1、面向过程**：**核心是过程二字，过程指的是解决问题的步骤，好比如设计一条流水线，是一种机械式的思维方式。**

就是程序从上到下一步步执行，一步步从上到下，从头到尾的解决问题 。基本设计思路就是程序一开始是要着手解决一个大的问题，然后把一个大问题分解成很多个小问题或子过程，这些子过程再执行的过程再继续分解直到小问题足够简单到可以在一个小步骤范围内解决。

**2、优缺点**：

**优点：将复杂的问题流程化，进而简单化。**

**缺点：扩展性差**

**二、面向对象**

**1、面向对象：核心是对象二字，特征和技能的结合体。**

**2、优缺点：**

**优点：可扩展性高**

**缺点：编程复杂度高**

**3、应用场景**：用户需求经常变化，互联网应用，游戏，企业内应用等。

**4、关于面向对象的几个名词解释**

**类：**一个类即是对一系列对象相似的特征和技能的结合体，如同一个模板。在类中定义了这些对象的都具备的属性，共同的方法。

**属性：**人类包含很多特征，把这些特征用程序来描述的话，叫做属性，比如年龄、身高、性别、姓名等都叫做属性，一个类中，可以有多个属性。

**方法：**人类不止有身高、年龄、性别这些属性，还能做好多事情，比如说话、走路、吃饭等，相比较于属性是名词，说话、走路是动词，这些动词用程序来描述就叫做方法。

**实例(对象)：**一个对象即是一个类的实例化后的实例，一个类必须经过实例化后方可在程序中调用，一个类可以实例化多个对象，每个对象亦可以有不同的属性，就像人类是指所有人，每个人是指具体的对象，人与人之前有共性，亦有不同。

**实例化：**把一个类转变为一个对象的过程就叫实例化

**6、面向对象三大特性**

<1>**封装**（Encapsulation）：在类中对数据的赋值、内部调用对外部用户来说是透明不可见的，这使类变成了一个胶囊或容器，里面包含着类的数据和方法。

<2>**继承**（Inheritance）：一个类可以派生出子类，在这个父类里定义的属性、方法自动被子类继承。

<3>**多态**（Polymorphism）：多态是面向对象的重要特性,简单点说:“一个接口，多种实现”，指一个基类中派生出了不同的子类，且每个子类在继承了同样的方法名的同时又对父类的方法做了不同的实现，这就是同一种事物表现出的多种形态。

36 [Python 字符串操作（截取/替换/查找/分割）](https://www.cnblogs.com/baoendemao/p/3804664.html)

Python 截取字符串使用 变量[头下标:尾下标]，就可以截取相应的字符串，其中下标是从0开始算起，可以是正数或负数，下标可以为空表示取到头或尾。

Python 替换字符串使用 变量.replace("被替换的内容"，"替换后的内容"[，次数])，替换次数可以为空，即表示替换所有。要注意的是使用replace替换字符串后仅为临时变量，需重新赋值才能保存。

Python 查找字符串使用 变量.find("要查找的内容"[，开始位置，结束位置])，开始位置和结束位置，表示要查找的范围，为空则表示查找所有。查找到后会返回位置，位置从0开始算，如果没找到则返回-1。

Python 分割字符串使用 变量.split("分割标示符号"[分割次数])，分割次数表示分割最大次数，为空则分割所有(用逗号分割str字符串，并保存到列表)

37 交换变量

a, b = b, a

38 join函数

语法： 'sep'.join(seq)

参数说明  
sep：分隔符。可以为空  
seq：要连接的元素序列、字符串、元组、字典  
上面的语法即：以sep作为分隔符，将seq所有的元素合并成一个新的字符串

返回值：返回一个以分隔符sep连接各个元素后生成的字符串

39 正则表达式

用\d可以匹配一个数字，\w可以匹配一个字母或数字 .可以匹配任意字符

用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符 \s可以匹配一个空格

[0-9a-zA-Z\\_]可以匹配一个数字、字母或者下划线；

[0-9a-zA-Z\\_]+可以匹配至少由一个数字、字母或者下划线组成的字符串，比如'a100'，'0\_Z'，'Py3000'等等；

[a-zA-Z\\_][0-9a-zA-Z\\_]\*可以匹配由字母或下划线开头，后接任意个由一个数字、字母或者下划线组成的字符串，也就是Python合法的变量；

[a-zA-Z\\_][0-9a-zA-Z\\_]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）。

A|B可以匹配A或B，所以(P|p)ython可以匹配'Python'或者'python'。

^表示行的开头，^\d表示必须以数字开头。

$表示行的结束，\d$表示必须以数字结束。

re.match()方法判断是否匹配，如果匹配成功，返回一个Match对象，否则返回None

除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组（Group）。

40 python ‘//’ 取整，‘%’ 取余

41 Python count() 方法用于统计字符串里某个字符出现的次数。可选参数为在字符串搜索的开始与结束位置。

42 ceil() 函数返回数字的上入整数。ceil()是不能直接访问的，需要导入 math 模块，通过静态对象调用该方法。import math math.ceil( x )

43 try-finally 语句

try-finally 语句无论是否发生异常都将执行最后的代码

当在try块中抛出一个异常，立即执行finally块代码。

finally块中的所有语句执行后，异常被再次触发，并执行except块代码。

捕捉异常可以使用try/except语句。

try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。

如果你不想在异常发生时结束你的程序，只需在try里捕获它。

Linux

常用命令：

ls 显示文件或目录

-l 列出文件详细信息l(list)

-a 列出当前目录下所有文件及目录，包括隐藏的a(all)

mkdir 创建目录

p 创建目录，若无父目录，则创建p(parent)

cd 切换目录

touch 创建空文件

echo 创建带有内容的文件。

Cat 查看文件内容

Cp 拷贝

mv 移动或重命名

rm删除文件

-r 递归删除，可删除子目录及文件

-f 强制删除

find 在文件系统中搜索某文件

wc 统计文本中行数、字数、字符数

grep 在文本文件中查找某个字符串: grep 是查找匹配条件的行，find 是搜索匹配条件的文件。

rmdir 删除空目录

tree 树形结构显示目录，需要安装tree包

pwd 显示当前目录

ln 创建链接文件

more、less 分页显示文本文件内容: more功能类似 cat ，cat命令是整个文件的内容从上到下显示在屏幕上。 more会以一页一页的显示方便使用者逐页阅读，而最基本的指令就是按空白键（space）就往下一页显示，按 b 键就会往回（back）一页显示，而且还有搜寻字串的功能 。more命令从前向后读取文件，因此在启动时就加载整个文件。less 工具也是对文件或其它输出进行分页显示的工具，应该说是linux正统查看文件内容的工具，功能极其强大。Less 的用法比起 more 更加的有弹性。 在 more 的时候，我们并没有办法向前面翻，只能往后面看，但若使用了 less 时，就可以使用 [pageup] [pagedown] 等按 键的功能来往前往后翻看文件，更容易用来查看一个文件的内容！除此之外，在 less 里头可以拥有更多的搜索功能，不止可以向下搜，也可以向上搜。

head、tail 显示文件头、尾内容

ctrl+alt+F1 命令行全屏模式

**系统管理命令**

Stat 显示指定文件的详细信息，比ls更详细

who 显示在线登陆用户

whoami 显示当前操作用户

hostname 显示主机名

uname 显示系统信息

top 动态显示当前耗费资源最多进程信息

ps 显示瞬间进程状态 ps -aux 显示所有包含其他[使用者](http://www.so.com/s?q=%E4%BD%BF%E7%94%A8%E8%80%85&ie=utf-8&src=internal_wenda_recommend_textn" \t "_blank)的行程

du 查看目录大小 du -h /home带有单位显示目录信息

df   查看磁盘大小 df -h 带有单位显示磁盘信息

ifconfig 查看网络情况

ping 测试网络连通

netstat 显示网络状态信息

man 命令不会用了，找男人  如：man ls

clear 清屏

alias 对命令重命名 如：alias showmeit="ps -aux" ，另外解除使用unaliax showmeit

kil 杀死进程，可以先用ps 或 top命令查看进程的id，然后再用kill命令杀死进程。

lsof -i:端口号 查看使用端口号的进程

netstate -tunlp|grep 端口号 查看使用端口号的进程

**打包压缩相关命令**

gzip：

bzip2：

tar: 打包压缩

-c 归档文件

-x 压缩文件

-z gzip压缩文件

-j bzip2压缩文件

-v 显示压缩或解压缩过程 v(view)

-f 使用档名

tar -cvf /home/abc.tar /home/abc 只打包，不压缩

tar -zcvf /home/abc.tar.gz /home/abc 打包，并用gzip压缩

tar -jcvf /home/abc.tar.bz2 /home/abc 打包，并用bzip2压缩

**关机/重启机器**

shutdown

-r 关机重启

-h 关机不重启

now 立刻关机

halt 关机

reboot 重启

**Linux管道**

将一个命令的标准输出作为另一个命令的标准输入。也就是把几个命令组合起来使用，后一个命令除以前一个命令的结果

**vim使用**

vim三种模式：命令模式、插入模式、编辑模式。使用ESC或i或：来切换模式。

命令模式下：

:q 退出

:q! 强制退出

:wq 保存并退出

**用户及用户组**

1.查看所有用户：cut -d : -f 1 /etc/passwd

2.添加用户：useradd newuser

3. 删除用户：userdel 用户名

4. 添加分组：groupadd newgroup

5. 删除分组：groupdel 组名

6. 修改密码：passwd passwd 用户名

7. su切换用户：su – 用户名

8. chmod修改访问权限：chmod [who] [+ | - | =] [mode] 文件名

9 chown 用于更改某个文件或目录的属主和属组：chown [用户:组] 文件 chown oracle:dba text 该命令将 text 文件的属主和属组分别改为oracle 和dba。

PID的分配方式：

 pid分配上限的查询方式cat /proc/sys/kernel/pid\_max，Android系统一般默认为32768。

 对于pid<300的情况值允许分配一次，不可再改变。也就是进程pid分配范围为(300, 32768)；

 每个pid分配成功，便会把当前的pid设置到last\_pid， 那么下次pid的分配便是从last\_pid+1开始 往下查找。这就意味着当last\_pid+1或者附近的进程，刚被杀并回收该pid，此时再创建新进程，很有可能会复用 pid.

 位图法记录已分配和未分配pid,由于pid的最大上限为32768，故pidmap采用4KB大小的内存，每一位代表一个进程ID号，正好4K\*8=32K= 32768。

# Ctrl+c跟kill-9和kill-15的区别

ctrl+c是终止当前在终端窗口中运行的命令或脚本，  
kill -9 pid，是不顾后果的强制终止(如果的你的速度够快，有时候是和ctrl＋c是一样的)  
kill -15 pid，是先关闭和其有关的程序，再将其关闭

其他：

面向对象的三大特性

1)封装性

封闭性，也可以称为信息隐藏。就是将一个类的使用和实现分开，只保留有限的接口(方法)与外部联系。对于用到该类的开发人员，只要知道这个类如何使用即可，而不用去关心这个类是如何实现的。这样做可以让开发人员更好地把精力集中起来专注别的事情，同时也避免了程序之间的相互依赖而带来的不便。

2)继承性

继承性就是派生类(子类)自动继承一个或多个基类(父类)中的属性与方法，并可以重写或添加新的属性和方法。继承这个特性简化了对象和类的创建，增加了代码的可重性。继承分单继承和多继承，PHP所支持的是单继承，也就是说，一个子类有且只有一个父类。

3)多态性

多态性是指同一个类的不同对象，使用同一个方法可以获得不同的结果，这种技术称为多态性。多态性增强了软件的灵活性和重用性。

什么时候用多态？

当接口已经确定，但同一个接口在不同环境需要不同实现的时候。

面向对象和面向过程的区别

面向对象是相对于面向过程而言的。面向过程语言是一种基于功能分析的、以算法为中心的程序设计方法；而面向对象是一种基于结构分析的、以数据为中心的程序设计思想。在面向对象语言中有一个有很重要东西，叫做类。面向对象有三大特性：封装、继承、多态