Linux

- **“cd”**: The cd command changes your current directory
    - “/”: root directory
    - “~”: home directory
    - “.”: current directory
    - “..”: the parent of current directory
    - “-”: move to previous directory
- **“mv”:** move or rename a file:
    - “mv <file> <new file path>”: move a file to the new location
    - “mv <filename> <new filename>”: rename a file to a new filename
- **“ls”:** view the contents of a directory
    - ls -R will list all the files in the sub-directories as well
    - ls -a will show the hidden files
    - ls -al will list the files and directories with detailed information like the permissions, size, owner, etc.
- **“chmod”**: The chmod command sets the file permissions flag on a file or folder. The flags define who can read, write to or execute the file. When you list files with the “ls -l” (long format) option you’ll see a string of characters that look like “-rwxrwxrwx”. One way to use chmod is to provide the permissions you wish to give to the owner, group, and others as a 3 digit number. The leftmost digit represents the owner. The middle digit represents the group. The rightmost digit represents the others.

  “chmod 765 example.text”
- **“chown”**: The chown command allows you to change the owner and group owner of a file.
-
- **“find”**: searches for files and directories within a given directory
    - To find files in the current directory use: “find . -name notes.txt”
- **“grep”**: search through all the text in a given file. To illustrate, “grep blue notepad.txt” will search for the word blue in the notepad file. Lines that contain the searched word will be displayed fully.
- **“ping”**: check your connectivity status to a server
- “**ssh**”: connect to a remote server
- 终止一个进程 kill -9 pid

Python/C++ 基础

C++:

- **Virtual Function**

A virtual function is a member function which is declared within a base class and is re-defined(Overridden) by a derived class. When we use a pointer from the base class to refer to a derived class object, we can call a virtual function for that object and execute the derived class's version of the function.
"Person *p = new Student();"

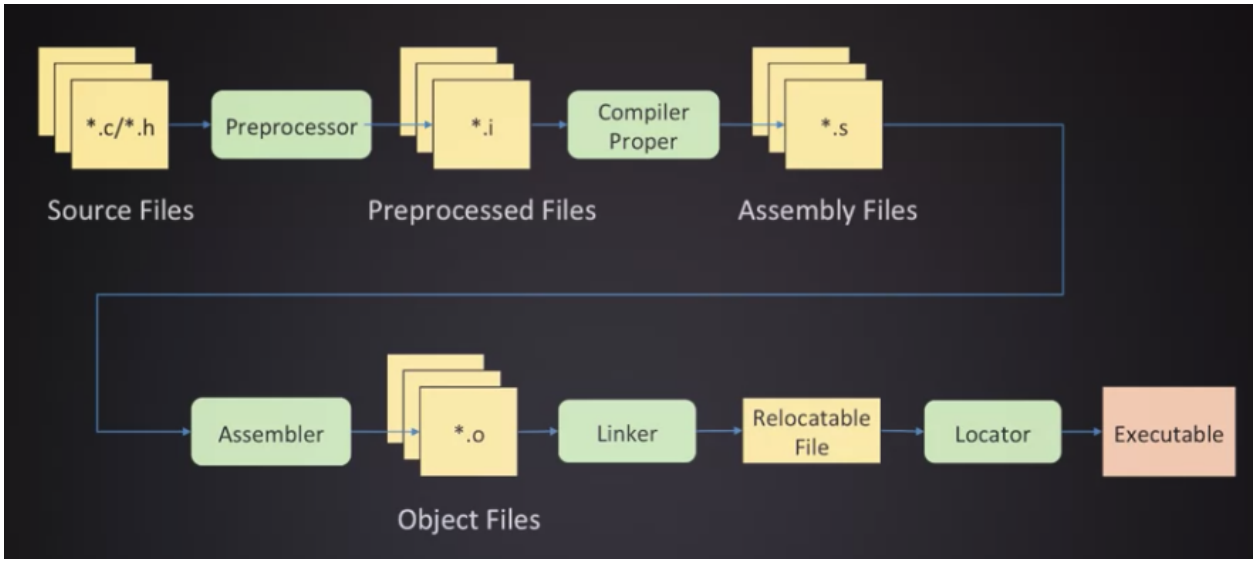
- 虚函数和纯虚函数区别？

使用一个基类类型的指针或者引用，来指向子类对象，进而调用由子类复写的个性化的虚函数
- 虚函数，在类成员方法的声明（不是定义）语句前加"virtual", 如 virtual voidfunc()
- 纯虚函数，在虚函数后加"=0"，如 virtual void func()=0
- 对于虚函数，子类可以（也可以不）重新定义基类的虚函数，该行为称之为复写Override。
- 对于纯虚函数，子类必须提供纯虚函数的个性化实现。

子类如果不提供虚函数的实现，将会自动调用基类的缺省虚函数实现，作为备选方案；
子类如果不提供纯虚函数的实现，编译将会失败。

- **Constructor / Deconstructor**
  The constructor of a class is a member function which will be automatically called once the object is created.
  The destructor cleans up the memory allocated within the class once the object is deleted. This function will be called automatically once an object is destroyed.
- **Pointer / Reference**
  A pointer holds the address of a variable
  A reference is another name(alias) of a pre-existing object and it doesn't have memory for its own.
- **Marco**
  Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro.
- **Smart Pointer**
  A smart pointer is a data type, usually implemented with templates, that simulates a pointer while also providing automatic garbage collection. It automatically counts the number of references to a SmartPointer<T*> object and frees the object of type T when the reference count hits zero.
- **Template**
  Templates are a way of reusing code to apply the same class to different data types. For example, we might have a list-like data structure which we would like to use for lists of various types.

Compilation



It consists of five main steps: Pre-processing, compiling, assembling, linking and locating

The preprocessor initially reads through source code and prepares it for compilation through three tasks.

- First, the preprocessor removes all comments from the code, those lines specified in C by /* */ or //.
- Second, the preprocessor will include any header files linked at the beginning of C files through the syntax #include "example_header.h".
- Finally, any and all macro variables defined in the file are replaced by their specified values.

Compiler: After preprocessing, the newly-filtered file is passed to the compiler. The compiler takes the preprocessed file and uses it to generate corresponding assembly code.
Assembler: From the compiler, the new assembly code is passed to the assembler, which assembles the code into object code. Where assembly code represents a correspondence between program and machine code, object code represents pure machine code (ie. binary)
Linker: Since one program will have multiple source code “xx.c” files, which means we will have multiple object files. The linker will take all object codes and libraries passed to it and link them together into a single executable file.

Python 的内存池机制

为什么要引入内存池:当创建大量小内存的对象的时候，频繁调用new和malloc会导致大量的内存碎 片。内存池首先在内存中申请一定数量等大小的内存块备用，当有新的内存需求的时候就分配而不是单 独再开辟内存。这样可以提升运行效率

Python的内存管理主要分为4层。

- 第0层是Cpython解释器中调用的C语言的malloc.

- 第一层是当申请内存大于256K的时候，用Python的raw_memory_allocator来分配。
- 第二层是当申请的内存小于256K 的时候，由Python的object allocator来分配。
- 第三层是python内置对象的独立内存池，各对象之间的 内存池不共享。对于常用的整数 -5, 256会直接被存放在缓冲池里。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。

Python元组和列表的区别

相同点:
- 都是序列
- 都可以存储任何数据类型
- 可以通过索引访问

不同点:
- 元组是不可变的，列表是可变的. tuple一旦初始化就不能修改，而且没有append() insert() 这些方法
- 列表不能当作字典的key, 而元组可以 （ Hashable)

面向过程和面向对象的语言区别

Reference
面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了；
面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为

反射

Reference
反射就是通过字符串的形式，导入模块；通过字符串的形式，去模块寻找指定函数，并执行。利用字符串的形式去对象（模块）中操作（查找/获取/删除/添加）成员,种基于字符串的事件驱动！
在python中主要是以getattr 和setattr两个函数实现的。

Python的数据结构

list, dict, tuple, linked list, str, set, queue, stack

Dict 和 list 的底层实现

Reference
dict的底层是哈希表。对插入字典的键，首先用哈希函数计算出一个哈希值，再以下标的形式映射到一个数组中，例如mod n。数组是一个二维数组，每一项保存了当前键的哈希值，和键值对。
python采用开放寻址法，同时如果数组中的已有的值占数组的2/3，就会开辟2倍的空间，用哈希

值 重新计算下标。这种方式的字典是无序的。

在python3.7之后，字典变成了有序的，可以记录插入的顺序。它的底层有两个数组，分别是indices和entries数组。插入一个值的时候，先计算出下标i，然后把indices数组下标为i的值改成当前entries数组的长度，然后将哈希值和键值对插入entries数组的末尾。这样entries数组就是按照插入顺序来构建的。查找的时候，对键计算哈希值，在indices数组对应的位置找到键值对在entries数组中的下标，然后再去entries数组中取值。这样的好处还有节约空间，以及加快了遍历速度。Python中所有不可变的内置类型都是可哈希的。可变类型（如列表，字典和集合）就是不可哈希的，因此不能作为字典的键。

list底层是一个可变长的数组，保存的都是对象的引用。当插入的时候，列表已满，就扩张。当删除 的时候，列表的长度小于分配大小的一半，就将分配大小减半

Python 的装饰器，作用，用法

Reference
装饰器本质上是一个 Python 函数或类，它可以让其他函数或类在不需要做任何代码修改的前提下增加额外功能，装饰器的返回值也是一个函数/类对象。它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景，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与函数功能本身无关的雷同代码到装饰器中并继续重用。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。

编译型语言和解释型语言 解释型语言的优点

计算机是不能够识别高级语言的，所以当我们运行一个高级语言程序的时候，就需要一个“翻译机”来从事把高级语言转变成计算机能读懂的机器语言的过程。这个过程分成两类，第一种是编译，第二种是解释。

- 编译型语言在程序执行之前，有一个单独的编译过程，将程序翻译成机器语言就不用再进行翻译了。
- 解释型语言，是在运行的时候将程序翻译成机器语言，所以运行速度相对慢。C/C++ 等都是编译型语言，而Java，C#等都是解释型语言。虽然Java程序在运行之前也有一个编译过程，但是并不是将程序编译成机器语言，而是将它编译成字节码（可以理解为一个中间语言）。在运行的时候，由JVM将字节码再翻译成机器语言。

编译型程序是面向特定平台的因而是平台依赖的。
编译型程序不支持代码中实现安全性——例如，一个编译型的程序可以访问内存的任何区域，并且可以对你的PC做它想做的任何事情（大部分病毒是使用编译型语言编写的）
由于松散的安全性和平台依赖性，编译型语言不太适合开发因特网或者基于Web的应用

解释型语言最大的优势之一是其平台独立性
解释型语言也可以保证高度的安全性——这是互联网应用迫切需要的
解释型语言存在一些严重的缺点。解释型应用占用更多的内存和CPU资源

垃圾回收机制

Reference
Python自带的解释器CPython主要使用了三种垃圾回收机制：是引用计数为主，标记-清除和分代回收两种机制为辅的策略

- 引用计数
  - 引用计数法Reference Counting的原理是，每个对象都维护一个引用计数字段，记录这个对象被引用的次数。如果有新的引用指向对象，对象引用计数就加一，引用被销毁时，对象引用计数减一，当用户的引用计数为0时，该内存被释放。
  - 引用计数法无法解决循环引用的问题（有一组对象的引用计数不为0，但是这组对象实际上并没有被变量引用，它们之间是相互引用，而且也不会有其他的变量再去引用这组对象，最终导致如果使用引用计数法这些对象占用的内存永远不会被释放，从而导致内存泄露）

- 标记清除：
  - Python会循环遍历零代链表上的每个对象，检查链表中每个互相引用的对象，根据规则减掉其引用计数，这一步是检测循环引用
  - Python中存在一个Garbage collection阈值，当被分配对象的计数值与被释放对象的计数值之间的差异累计超过某个阈值，则Python的收集机制就启动了，并且触发上述的标记-清除机制
- 分代回收：
  - 当执行标记-清除后，剩余的对象都是真实被引用的，而这些对象都会被移入一代链表。所谓一代链表就是零代链表执行标记-清除之后的剩余对象组成的链表。同样的，二代链表就是一代链表执行标记-清除之后的剩余对象组成的链表
  - Python采用分代回收的机制，实际上是基于弱代假说(weak generational hypothesis)提出的：这个假说由两个观点构成：年轻的对象通常“死”得也快，老对象则很有可能存活更长的时间。分代回收的意义在于：通过频繁的处理零代链表中的新对象，Python的垃圾收集器将把时间花在更有意义的地方

GIL锁

Reference
GIL锁指的是python的全局解释器锁（Global Interpreter Lock)。也就是在同一时间只能有一个线程执行python字节码。主要 原因是 **CPython**解释器的内存管理不是线程安全 的。某个线程想要执行，必须先拿到GIL，我们可以把GIL看作是“通行证”，并且在一个python进程中，GIL只有一个。拿不到通行证的线程，就不允许进入CPU执行。

GIL在两种情况的时候会被释放：

- 第一个是IO 操作
- 或者ticks计数达到100（ticks可以看作是python自身的一个计数器，专门做用于GIL，每次释放后归零，这个计数可以通过 sys.setcheckinterval 来调整），进行释放。

而每次释放GIL锁，线程进行锁竞争、切换线程，会消耗资源。并且由于GIL锁存在，python里一个进程永远只能同时执行一个线程(拿到GIL的线程才能执行)，这就是为什么在多核CPU上，**python的多线程效率并不高**。

避免GIL锁的话可以实现多进程: 原因是每个进程有各自独立的GIL，互不干扰，这样就可以真正意义上的并行执行，所以在python中，多进程的执行效率优于多线程(仅仅针对多核CPU而言)。

is和==

如果想判 断两个变量是否引用了同一个对象，需要使用is.
== 符号只能判断对象的值是 否相等

__new__和__init__区别

链接 :https://www.nowcoder.com/questionTerminal/ab4b9c7553c3491ca84c0a62989260c0

- __init__是当实例对象创建完成后被调用的，然后设置对象属性的一些初始值。
- __new__是在实例创建之前被调用的，因为它的任务就是创建实例然后返回该实例，是个静态方法。即，__new__在__init__之前被调用，__new__的返回值（实例）将传递给__init__方法的第一个参数，然后__init__给这个实例设置一些参数。

用python写出多线程，循环打印123

```
import threading
def showa():
 while True:
        lockc.acquire()
        print('a')
        locka.release()
def showb():
    while True:
        locka.acquire()
        print('b')
        lockb.release()
def showc():
    while True:
        lockb.acquire()
        print('c')
        lockc.release()
if __name__ == '__main__':
    locka = threading.Lock()
    lockb = threading.Lock()
    lockc = threading.Lock()

    t1=threading.Thread(target=showa)
    t2=threading.Thread(target=showb)
    t3=threading.Thread(target=showc)

    locka.acquire()
    lockb.acquire()

    t1.start()
    t2.start()
    t3.start()
```

操作系统 / Operating System

互斥锁，自旋锁，读写锁，悲观锁，乐观锁

Reference

互斥锁（Mutex):互斥锁是一种「独占锁」，比如当线程 A 加锁成功后，此时互斥锁已经被线程 A 独占了，只要线程 A 没有释放手中的锁，线程 B 加锁就会失败，于是就会释放 CPU 让给其他线程，既然线程 B 释放掉了 CPU，自然线程 B 加锁的代码就会被阻塞.

上下文开销成本大:

- 当线程加锁失败时，内核会把线程的状态从「运行」状态设置为「睡眠」状态，然后把 CPU 切换给其他线程运行；
- 接着，当锁被释放时，之前「睡眠」状态的线程会变为「就绪」状态，然后内核会在合适的时间，把 CPU 切换给该线程运行。

如果你能确定被锁住的代码执行时间很短，就不应该用互斥锁，而应该选用自旋锁，否则使用互斥锁

自旋锁（Spin Lock) :一般加锁的过程，包含两个步骤：第一步，查看锁的状态，如果锁是空闲的，则将锁设置为当前线程持有。当发生多线程竞争锁的情况，加锁失败的线程会「忙等待」，直到它拿到锁。自旋锁是最比较简单的一种锁，一直自旋，利用 CPU 周期，直到锁可用。

自旋锁开销少，在多核系统下一般不会主动产生线程切换，适合异步、协程等在用户态切换请求的编程方式，但如果被锁住的代码执行时间过长，自旋的线程会长时间占用 CPU 资源，所以自旋的时间和被锁住的代码执行的时间是成「正比」的关系，我们需要清楚的知道这一点。

读写锁:读锁是一个共享锁，写锁是互斥锁。

- 当「写锁」没有被线程持有时，多个线程能够并发地持有读锁，这大大提高了共享资源的访问效率，因为「读锁」是用于读取共享资源的场景，所以多个线程同时持有读锁也不会破坏共享资源的数据。
- 但是，一旦「写锁」被线程持有后，读线程的获取读锁的操作会被阻塞，而且其他写线程的获取写锁的操作也会被阻塞。

所以说，写锁是独占锁，因为任何时刻只能有一个线程持有写锁，类似互斥锁和自旋锁，而读锁是共享锁，因为读锁可以被多个线程同时持有。读写锁在读多写少的场景，能发挥出优势

乐观锁:乐观锁做事比较乐观，它假定冲突的概率很低，它的工作方式是：先修改完共享资源，再验证这段时间内有没有发生冲突，如果没有其他线程在修改资源，那么操作完成，如果发现有其他线程已经修改过这个资源，就放弃本次操作。

悲观锁:前面提到的互斥锁、自旋锁、读写锁，都是属于悲观锁。悲观锁做事比较悲观，它认为多线程同时修改共享资源的概率比较高，于是很容易出现冲突，所以访问共享资源前，先要上锁。

什么是死锁，死锁的条件

A deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.
1. Mutual exclusion: Only one process can use the resource at any given instant of time
2. Hold and wait: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. No preemption: a resource can be released only voluntarily by the process holding it.
4. Circular Wait:

当线程A持有独占锁a，并尝试去获取独占锁b的同时，线程B持有独占锁b，并尝试获取独占锁a的情况下，就会发生AB两个线程由于互相持有对方需要的锁，而发生的阻塞现象，我们称为死锁。
（1）互斥条件：一个资源每次只能被一个进程使用。
（2）请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
（3）不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。
（4）循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

如何预防死锁，避免死锁，死锁发生的检查

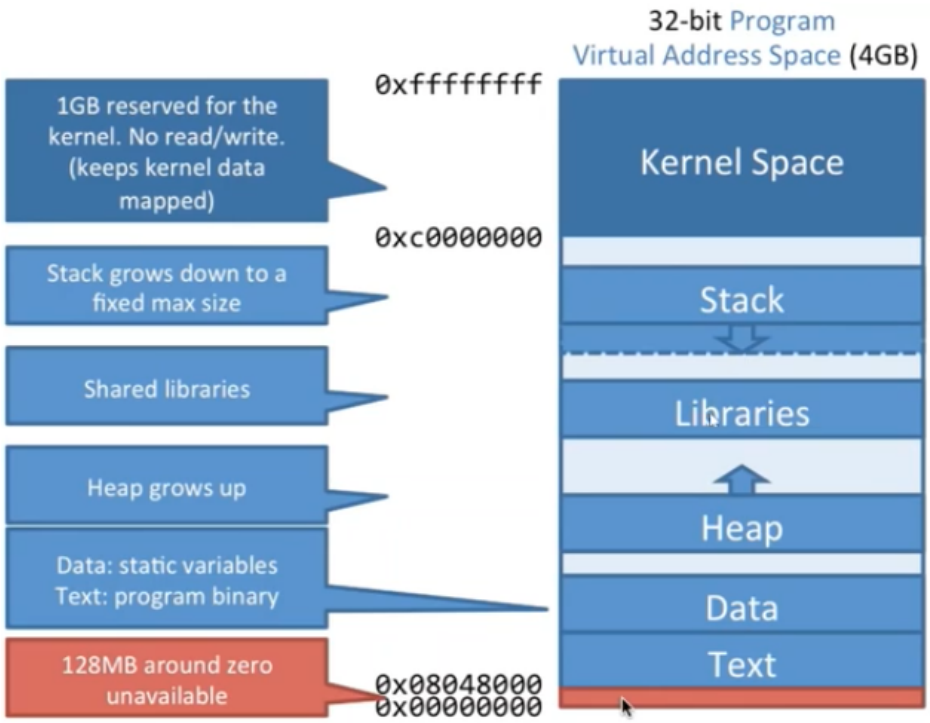
预防死锁:破坏死锁的四个条件:

- 把只能互斥访问的资源改造为允许共享使用
- 允许高优先级的进程剥夺低优先级的进程保持的资源
- 静态资源分配:进程一次性获取全部所需资源再运行
- 给资源的优先级编号，必须按增序获取

堆和栈的区别以及存储模式有什么区别

Reference
栈是由编译器自动分配和释放，存放函数的参数和局部变量的值，它的访问速度较快。栈上数据的释放是随着出栈自动销毁了。栈的生长方向向下，内存地址由高到低。

堆是程序运行时显式申请和释放的内存空间，程序结束后由操作系统回收，容易产生内存碎片。堆的空间比较灵活，可以不连续，通常比栈要大。堆的生长方向向上，内存地址由低到高。



内存泄漏和内存溢出

内存溢出**(out of memory):** 是指程序在申请内存时，没有足够的内存空间供其使用，出现outof memory;比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。比如栈满的时候入栈，栈空的时候出栈。

内存泄露 **(memory leak)**:是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。内存泄漏是指你向系统申请分配内存进行使用(new)，可是使用完了以后却不归还(delete)，结果你申请到的那块内存你自己也不能再访问（也许你把它的地址给弄丢了），而系统也不能再次将它分配给需要的程序

计算机内存管理的方式

Video
Reference
常见的内存管理方式有块式管理、页式管理、段式和段页式管理。
(1)块式管理：把主存分为一大块一大块的，当所需的程序片段不在主存时就分配一块主存空间，把程序片段load入主存，就算所需的程序片段只有几个字节也只能把这一块分配给它。这样会造成很大的浪费，平均浪费了50%的内存空间，但是易于管理。

(2)页式管理：把主存分为一页一页的，每一页的空间要比一块一块的空间小很多，这种方法的空间利用率要比块式管理高很多。程序的逻辑内存分为多个页（page），通过页表（page table）来寻找物理内存的映射，即相应的页帧（page frame）。如果对应的帧号是磁盘，那就会进入内核态，内

核会到磁盘中找到对应的数据然后加载到物理内存的帧中，然后将帧号填写到页表中。 如果物理内存已经满了，那就会使用页置换算法把不常用的页帧换到磁盘中。

- 时间优化：将常访问的几个页表项存到访问速度更快的硬件中，， 一般是MMU（内存管理单元），这个小表名称叫做TLB(Translation Lookaside Buffer)，可以称其为快表。寻址时先查找TLB，miss后再查page table。

(3)段式管理:把主存分为一段一段的，每一段的空间又要比一页一页的空间小很多，这种方法在空间利用率上又比页式管理高得多，但是也有另外一个缺点。一个程序片段可能会被分为几十段，这样很多时间就会被浪费在计算每一段的物理地址上。

(4)段页式管理：结合了段式管理和页式管理的优点。把主存先分成若干段，每个段又分成若干页。段页式管理每取一护具，要访问3次内存。

虚拟内存

用于扩充内存的。 它使得应用程序认为它拥有连续可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。虚拟内存的重要意义是它定义了一个连续的虚拟地址空间，并且把内存扩展到硬盘空间

如果程序很大，可以将经常用的部分装入内存，暂时用不到的部分留在外存。如果内存空间不足，操作系统会将暂时用不到的信息放入外存。当访问的信息不在内存中时，操作系统会将外中存的信息移入内存。

线程和进程的区别是什么

Reference
根本区别：进程是操作系统资源分配的基本单位，而线程是处理器任务调度和执行的基本单位

- 资源开销：每个进程都有独立的代码和数据空间（程序上下文），程序之间的切换会有较大的开销；线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小。

- 包含关系:如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的;线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程。

- 内存分配:同一进程的线程共享本进程的地址空间和资源，而进程之间的地址空间和资源是相互独立的

- 影响关系: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。

- 执行过程：每个独立的进程有程序运行的入口、顺序执行序列和程序出口。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，两者均可并发执行

多线程和多进程，以及应用场景

多进程多线程主要是为了实现并行计算。

- 多进程的话各个进程需要在操作系统层面进行进程的通信。
- 多线程共享进程的数据段和程序段，节约内存，可以直接操作共享内存，会涉及到同步问题。
- 多进程的新建进程开销，进程调度开销和通信开销要比多线程大。此外多进程的可靠性也强于多线程，因为多线程中的一个线程死亡可能导致整个进程异常，比如该线程没有释放一个锁。
- 多线程适合线程间需要数据共享的计算，多进程适合高性能要求的计算，以及分布式的系统。
- 在python中由于全局解释器锁的存在，多线程无法利用多核cpu实现并行化，所以python多线程适合IO密集型操作，多进程适合计算密集型操作

进程通信方式

Reference

1. 匿名管道:只支持单向数据流；只能用于具有亲缘关系的进程之间；没有名字；管道的缓冲区是有限的

2. 有名管道:有名管道不同于匿名管道之处在于它提供了一个路径名与之关联，以有名管道的文件形式存在于文件系统中，这样，即使与有名管道的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过有名管道相互通信

3. 信号(Signal): 信号是Linux系统中用于进程间互相通信或者操作的一种机制，信号可以在 任何时候发给某一进程，而无需知道该进程的状态。
    - 如果该进程当前并未处于执行状态，则该信号就由内核保存起来，直到该进程恢复执行并传递给它为止。
    - 如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消是才被传递给进程。

4. 消息队列(Message): 与管道（无名管道：只存在于内存中的文件；命名管道：存在于实际的磁介质或者文件系统）不同的是消息队列存放在内核中，只有在内核重启(即，操作系统重启)或者显示地删除一个消息队列时，该消息队列才会被真正的删除。

5. 共享内存(Shared Memory):
    - 使得多个进程可以可以直接读写同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。
    - 为了在多个进程间交换信息，内核专门留出了一块内存区，可以由需要访问的进程将其映射到自己的私有地址空间。进程就可以直接读写这一块内存而不需要进行数据的拷贝，从而大大提高效率。
    - 由于多个进程共享一段内存，因此需要依靠某种同步机制（如信号量）来达到进程间的同步及互斥。

6. 信号量(Semaphore): 信号量是一个计数器，用于多进程对共享数据的访问，信号量的意图在于进程间同步。为了获得共享资源，进程需要执行下列操作：
(1)创建一个信号量：这要求调用者指定初始值，对于二值信号量来说，它通常是1，也可是0
(2)等待一个信号量:该操作会测试这个信号量的值，如果小于0，就阻塞。也称为P操作。
(3)挂出一个信号量:该操作将信号量的值加1，也称为V操作。
7. 套接字(Socket): 套接字是一种通信机制，凭借这种机制，客户/服务器（即要进行通信的进程）系统的开发工作既可以在本地单机上进行，也可以跨网络进行。也就是说它可以让不在同一台计算机但通过网络连接计算机上的进程进行通信。

进程调度方法

Reference:
1. 非剥夺方式（非抢占方式）一旦占用CPU，直至完成或阻塞。不利用实时任务，不利用短作业；使用于批处理系统
2. 剥夺方式（抢占方式）在一定情况下，可剥夺一进程占有的处理机。抢占的原则有：短作业（进程）优先原则、时间片原则、优先权原则。

- 先来先服务调度算法（FCFS）
- 短作业优先调度算法（SJF）
- 时间片轮转调度算法（RR）
- 高响应比优先调度算法（HRRF）： 响应时间 / 运行时间
- 最高优先级优先调度算法

什么是Linux用户态和内核态

Reference
指的是处理器处于的两种状态。

- 用户态的程序只能访问用户空间的资源，执行非特权指令，对于特权指令要通过系统调用的方式进行。
- 运行在内核态的进程控制着操作系统内核程序和计算机的硬件资源，可以执行特权指令和非特权指令。

用户态就是提供应用程序运行的空间，为了使应用程序访问到内核管理的资源例如CPU，内存，I/O。内核必须提供一组通用的访问接口，这些接口就叫系统调用

从用户态到内核态切换可以通过三种方式：

1. 系统调用: 其实系统调用本身就是中断，但是软件中断，跟硬中断不同。
2. 异常:如果当前进程运行在用户态，如果这个时候发生了异常事件，就会触发切换。例如：缺页异常。
3. 外设中断：当外设完成用户的请求时，会向CPU发送中断信号。

进程的状态，进程状态就绪和等待状态的区别是什么

就绪，执行，阻塞
阻塞和就绪的区别：阻塞是等待除CPU以外的资源，而就绪等待的是CPU资源。
1)就绪——执行：对就绪状态的进程，当进程调度程序按一种选定的策略从中选中一个就绪进程，为之分配了处理机后，该进程便由就绪状态变为执行状态；
2)执行——阻塞：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如进程提出输入/输出请求而变成等待外部设备传输信息的状态，进程申请资源（主存空间或外部设备）得不到满足时变成等待资源状态，进程运行中出现了故障（程序出错或主存储器读写错等）变成等待干预状态等等；
当进程进入阻塞状态，是不占用CPU资源的。
3)阻塞——就绪：处于阻塞状态的进程，在其等待的事件已经发生，如输入/输出完成，资源得到满足或错误处理完毕时，处于等待状态的进程并不马上转入执行状态，而是先转入就绪状态，然后再由系统进程调度程序在适当的时候将该进程转为执行状态；
4)执行——就绪：正在执行的进程，因时间片用完而被暂停执行，或在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行而被迫让出处理机时，该进程便由执行状态转变为就绪状态。

线程池

线程池就是事先将若干个进程放入一个容器中，当需要的时候从线程池中获取线程，使用完毕的时候将线程放回线程池。这样可以减少多任务的时候创建和销毁线程的开销。这是一种多线程实现方式。

线程安全的实现方式

Reference
线程安全定义：当多个线程访问某个类, 不管运行环境采用何种调度方式或者这些线程如何交替执行, 并且在主调代码中不需要任何额外的同步或协同, 这个类都能表现出正确的行为, 那么就称这个类为线程安全的

- Synchronized: synchronized关键字，就是用来控制线程同步的，保证我们的线程在多线程环境下，不被多个线程同时执行，确保我们数据的完整性，使用方法一般是加在方法上。当synchronized锁住一个对象之后，别的线程如果想要获取锁对象，那么就必须等这个线程执行完释放锁对象之后才可以，否则一直处于等待状态。
- Lock:就是我们在需要的时候去手动的获取锁和释放锁，甚至我们还可以中断获取以及超时获取的同步特性，但是从使用上说Lock明显没有synchronized使用起来方便快捷。

进程和线程的上下文切换

Context Switches: 上下文切换就是从当前执行任务切换到另一个任务执行的过程。但是，为了确保下次能从正确的位置继续执行，在切换之前，会保存上一个任务的状态。

进程上下文切换与线程上下文切换最主要的区别就是线程的切换虚拟空间内存是相同的（因为都是属于自己的进程），但是，进程切换的虚拟空间内存则是不同的。
同时，这两种上下文切换的处理都是通过操作系统内核来完成的。内核的这种切换过程伴随的最显著的性能损耗是将寄存器中的内容切换出。线程上下文切换比进程上下文切换快的多。

协程

Reference
协程的特点

1. 线程的切换由操作系统负责调度，协程由用户自己进行调度，因此减少了上下文切换，提高了效率。
2. 线程的默认Stack大小是1M，而协程更轻量，接近1K。因此可以在相同的内存中开启更多的协程。
3. 由于在同一个线程上，因此可以避免竞争关系而使用锁。
4. 适用于被阻塞的，且需要大量并发的场景。但不适用于大量计算的多线程，遇到此种情况，更好实用线程去解决

协程的原理：当出现IO阻塞的时候，由协程的调度器进行调度，通过将数据流立刻yield掉（主动让出），并且记录当前栈上的数据，阻塞完后立刻再通过线程恢复栈，并把阻塞的结果放到这个线程上去跑，这样看上去好像跟写同步代码没有任何差别，这整个流程可以称为coroutine

协程是轻量级线程，拥有自己的上下文和栈，在调度切换的时候可以恢复上一次调用时的状态，也 就是上一次离开逻辑流的位置。适合IO密集型程序和高并发程序

并发和并行区别

- 并行(parallel)：指在同一时刻，有多条指令在多个处理器上同时执行。所以无论从微观还是从宏观来看，二者都是一起执行的。
- 并发(concurrency)：指在同一时刻只能有一条指令执行，但多个进程指令被快速的轮换执行，使得在宏观上具有多个进程同时执行的效果，但在微观上并不是同时执行的，只是把时间分成若干段，使多个进程快速交替的执行。

数据库

数据库引擎及他们之间的区别

MyISAM
MyISAM is best suited for high “select” query rate, and non transactional operations.
优势是：

- 占用空间小，处理速度快，有较高的插入和查询速度
- 索引的方式：MyISAM也是使用B+tree索引但是和Innodb的在具体实现上有些不同。

缺点是:

- 不支持事务的完整性和并发性

- 只支持表锁，不支持行锁
- MyISAM is the worst for high “insert/update” query rate. (**because of table level locking**)

InnoDB:
InnoDB is best suited for parallel insert/update/delete operations (**because of row level locking**), and transactional operations (because of roll back feature).
优势在于:
- 提供了良好的事务处理、崩溃修复能力和并发控制（锁 是行级别）
- innodb存储引擎支持外键（foreign key)
- 为巨大数据量设计，表的可用空间很大
- innodb存储引擎索引使用的是B+Tree。

缺点是:
- 读写效率较差，占用的数据空间相对较大。

Memory:
优点是：
- Memory engine (HEAP) is best suited for on the fly fast data access, as everything is stored in the RAM.
- memory存储引擎默认使用哈希（HASH）索引，其速度比使用B-+Tree型要快

缺点是:
- Memory engine is worst for long term usage (because Of data integrity issues) and transactional operations.

(多次)查询速度慢的原因，如何解决

首先判断执行很慢出现的频率，是偶尔执行很慢，还是每次都很慢。如果是偶尔很慢的情况，可能是数据库的redo log 被写满，正在刷新脏页，也就是将内存中修改的数据同步到磁盘中。此时可能因为磁盘占用会发生数据库性能的短暂下降。也有可能是查询的这个表或表项被加锁了，无法获取。

如果每次都很慢，那么也有几种情况，可以通过explain语句排查
- 对查询的条件字段没有加索引，只能进行全表扫描
- 加了索引但是sql语句没有利用到索引
  例如查询的时候对字段使用了函数操作，或者查询的时候执行了潜在的类型转换
- 数据库没有利用索引
  mysql中有查询优化器的机制。根据我们查询的内容，mysql会找出所有可用的索引，并预测使用各种索引以及执行全表扫描的代价。最后会使用代价最低的查询方式。假如这个字段的区分度比较高，意味着符合要求的行数会比较少，此时走索引的效率会更高。而mysql对于字段的区分度是通过抽样的方式进行的。也许这个抽样的结果偏差比较大，系统预测字段的区分度比较低而选择不走索引，而是全表扫描。这种情况可以用过强制走索引查询来验证：
  ```
  select * from t force index(a) where c < 100 and c < 100000;
  ```
- 也有可能是索引的设计问题

比如我们对一个比较长的字段进行了前缀索引，而这个字段恰好前缀的重复字符比较多（比如手机号或者是身份证号），因此索引没有办法很好地区分。这种情况可以通过增加前缀索引的字符数来提高区分度，或者利用倒序保存的技巧来让前缀尽可能有区分度
还有可能是因为我们查询的索引是非主键索引，因此产生了回表的开销。这种情况可以通过覆盖索引的方式解决。例如针对需要查询和筛选的字段建立联合索引，这样可以通过索引本身来获取需要查找的数据，而不用回表。

- sql语言的设计问题
  例如把两个大表执行了连接操作或者进行了全连接，这样会生成拥有大量数据的中间表。拖累查询速度。这种方法可以通过先对一个表进行条件查找，再进行连接。此外就是对查询结果使用limit操作。

数据库的事务是什么？怎么使用

数据库事务是指作为单个逻辑单元执行的一系列操作，要么完全执行，要么完全地不执行。事务的四个特性(ACID):原子性、一致性、隔离性和持久性。

mysql中执行insert delete和update的时候，就开启了一个事务。或者用start transaction手动开启事务，commit之后提交成功。rollback回滚。

事务的特性（ACID），讲一下每个特性的意思

一般来说，事务是必须满足4个条件（ACID）：：原子性（Atomicity，或称不可分割性）、(一致性Consistency)、隔离性(Isolation，又称独立性)、持久性(Durability)。

原子性:一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性:在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

隔离性:数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted)、读提交（read committed）、可重复读（repeatable read）和串行化(Serializable)。

持久性:事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

数据库隔离等级

由低到高:
1. 读取未提交(Read Uncommitted)：所有事物可看到其他未提交事物的结果。也就是脏读

2. 读取已提交(Read Committed)：一个事物可以看见已经提交的事物的结果。支持不可重复读：一个事物的处理期间，可能其他事物提交了更改。
3. 可重复读(Repeatable Read)：保证了一个事物读取数据的时候看到的是同一个数据。导致幻读：一个事务的写入改变了另一个事务的查询结果的正确性。
4. 串行化(Serializable)：事务只能串行执行，不能并发执行。

主键、外键、索引的各自的含义以及区别

定义：
主键：唯一标识一条记录，不能有重复的，不允许为空
外键：表的外键是另一表的主键, 外键可以有重复的, 可以是空值
索引:是对数据库表中一列或多列的值进行排序的一种结构

作用:
主键：用来保证数据完整性
外键：用来和其他表建立联系用的
索引:是提高查询排序的速度

索引的类型

从物理存储角度:
聚簇索引和非聚簇索引

从数据结构角度:
B+树索引、hash索引

从逻辑角度 :
1.普通索引: 是最基本的索引，它没有任何限制
2.唯一索引: 与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一
3.主键索引: 是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值
4.组合索引: 复合索引指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用复合索引时遵循最左前缀集合
5.全文索引: 主要用来查找文本中的关键字，而不是直接与索引中的值相比较。fulltext索引跟其它索引大不相同，它更像是一个搜索引擎，而不是简单的where语句的参数匹配。fulltext索引配合match against操作使用，而不是一般的where语句加like。它可以在create table，alter table，create index使用，不过目前只有char、varchar，text 列上可以创建全文索引

什么是聚簇索引、什么是非聚簇索引

聚集索引与非聚集索引的区别是：叶节点是否存放一整行记录

聚集索引: InnoDB使用的是聚簇索引，将主键组织到一棵B+树中，而行数据就储存在叶子节点上，若使用"where id = 14"这样的条件查找主键，则按照B+树的检索算法即可查找到对应的叶节点，之后获得行数据。
若对Name列进行条件搜索，则需要两个步骤：第一步在辅助索引B+树中检索Name，到达其叶子节点获取对应的主键。第二步使用主键在主索引B+树种再执行一次B+树检索操作，最终到达叶子节点即可获取整行数据。（重点在于通过其他键需要建立辅助索引）

非聚簇索引: MyISM使用的是非聚簇索引，非聚簇索引的两棵B+树看上去没什么不同，节点的结构完全一致只是存储的内容不同而已，主键索引B+树的节点存储了主键，辅助键索引B+树存储了辅助键。表数据存储在独立的地方，这两颗B+树的叶子节点都使用一个地址指向真正的表数据，对于表数据来说，这两个键没有任何差别。由于索引树是独立的，通过辅助键检索无需访问主键的索引树。

辅助索引使用主键作为"指针"而不是使用地址值作为指针的好处是，减少了当出现行移动或者数据页分裂时辅助索引的维护工作，使用主键值当作指针会让辅助索引占用更多的空间，换来的好处是InnoDB在移动行时无须更新辅助索引中的这个"指针"

索引的数据结构是什么

b+树。b+树是一种搜索树。它有两种节点，一种是叶子节点，一种是非叶节点。节点可以包含多个数据和指向多个子节点的指针。叶子节点之间以链表的形式连接在一起。所有的查询的数据都保存在叶子节点中。非叶节点是以一个索引值一个指针的方式存储的。查询的时候遍历非叶节点的索引值，找到查询数据所属的范围，再通过指针指向下一层节点，直到查询到叶子节点。

为什么要用b+树索引而不用hash or b-树 or 红黑树

reference
1. Hash 索引仅仅能满足”=”,”IN”和”<=>”查询，不能使用范围查询。
2. Hash 索引无法被用来避免数据的排序操作
3. 由于不同索引键存在相同 Hash 值，所以即使取满足某个 Hash 键值的数据的记录条数，也无法从 Hash 索引中直接完成查询，还是要通过访问表中的实际数据进行相应的比较，并得到相应的结果。
4. Hash 索引遇到大量Hash值相等的情况后性能并不一定就会比B+Tree索引高。
对于选择性比较低的索引键，如果创建 Hash 索引，那么将会存在大量记录指针信息存于同一个 Hash 值相关联。这样要定位某一条记录时就会非常麻烦，会浪费多次表数据的访问，而造成整体性能低下

reference
B+树只有叶节点存放数据，其余节点用来索引，而B-树是每个索引节点都会有Data域.
另一个优点是： B+树所有的Data域在叶子节点，一般来说都会进行一个优化，就是将所有的叶子节点用指针起来。这样遍历叶子节点就能获得全部数据，这样就能进行区间访问啦

在大规模数据存储的时候，红黑树往往出现由于树的深度过大而造成磁盘IO读写过于频繁，进而导致效率低下的情况

前缀索引

当要索引的列字符很多时 索引则会很大且变慢 (可以只索引列开始的部分字符串 节约索引空间 从而提高索引效率)
很长的字符序列，为了减少内存占用，和提高搜索的效率，以序列的前若干个字符作为索引。为了降低索引的重复率，建索引要先判断数据的分布，也就是前缀的数据重复性。

如何确定前缀索引长度？
可以通过计算选择性来确定前缀索引的选择性，计算方法如下
全列选择性：
SELECT COUNT(DISTINCT column_name) / COUNT(*) FROM table_name;
某一长度前缀的选择性
SELECT COUNT(DISTINCT LEFT(column_name, prefix_length)) / COUNT(*) FROM table_name;
当前缀的选择性越接近全列选择性的时候，索引效果越好。

数据库有几种表之间的连接形式（左连接，右连接，内连接，完全连接）

三大范式

https://zhuanlan.zhihu.com/p/63146817
1. 第一范式(确保每列保持原子性)
第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。
2. 第二范式(确保表中的每列都和主键相关): 也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。
3. 第三范式(确保每列都和主键列直接相关,而不是间接相关)

redis vs memcached

https://www.imaginarycloud.com/blog/redis-vs-memcached/
Memcache:
1. String is the only storing data type, which consumes comparatively less memory resources for metadata
2. Memcached is multithreaded, you can easily scale up by giving it more computational resources
3. If we use Memcached to modify a single field in the object, the string has to be loaded, deserialized, the object field edited, serialized and stored.

Redis:
1. Support 5 data types: string, hash table, set, sorted set, List
2. Support more eviction policies while Memcaches uses LRU mostly
3. High available by providing the persistence (RDB snapshot and AOF)
4. Redis is that the data it stores isn't opaque, so the server can manipulate it directly. These built-in commands and user scripts give you the flexibility of handling data processing tasks directly in Redis without having to ship data across the network to another system for processing
5. It is single threaded and grows well horizontally

Redis 是怎么实现永久保存的

https://docs.redislabs.com/latest/rc/concepts/data-persistence/
RDB snapshot: Is a point-in-time snapshot of all your dataset, that is stored in a file in disk and performed at specified intervals. This way, the dataset can be restored on startup.
If the dataset stored in Redis is too big, the RDB file will take some time to be created, which has an impact on the response time. On the other hand, it will be faster to load on boot up compared to the AOF log.

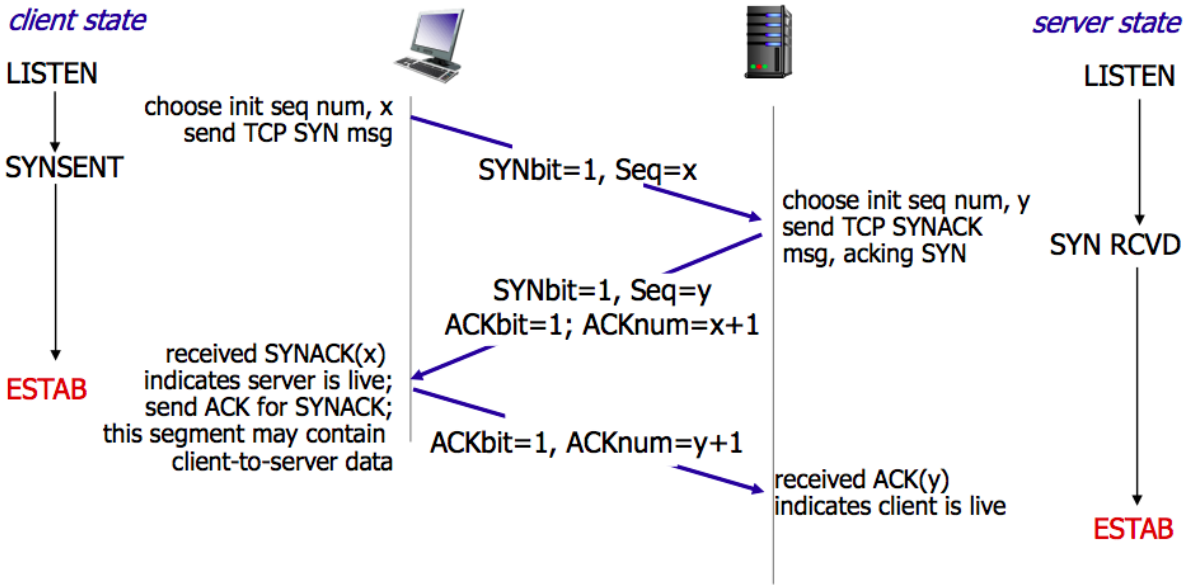
AOF log: Is an Append Only File log of all the write commands performed in the Redis server. This file is also stored in disk, so by re-running all the commands in their order, a dataset can be restored on startup. The AOF log is better if data loss is not acceptable at all, as it can be updated at every command. It also has no corruption issues since it's an append-only file. However, it can grow much larger than an RDB snapshot.

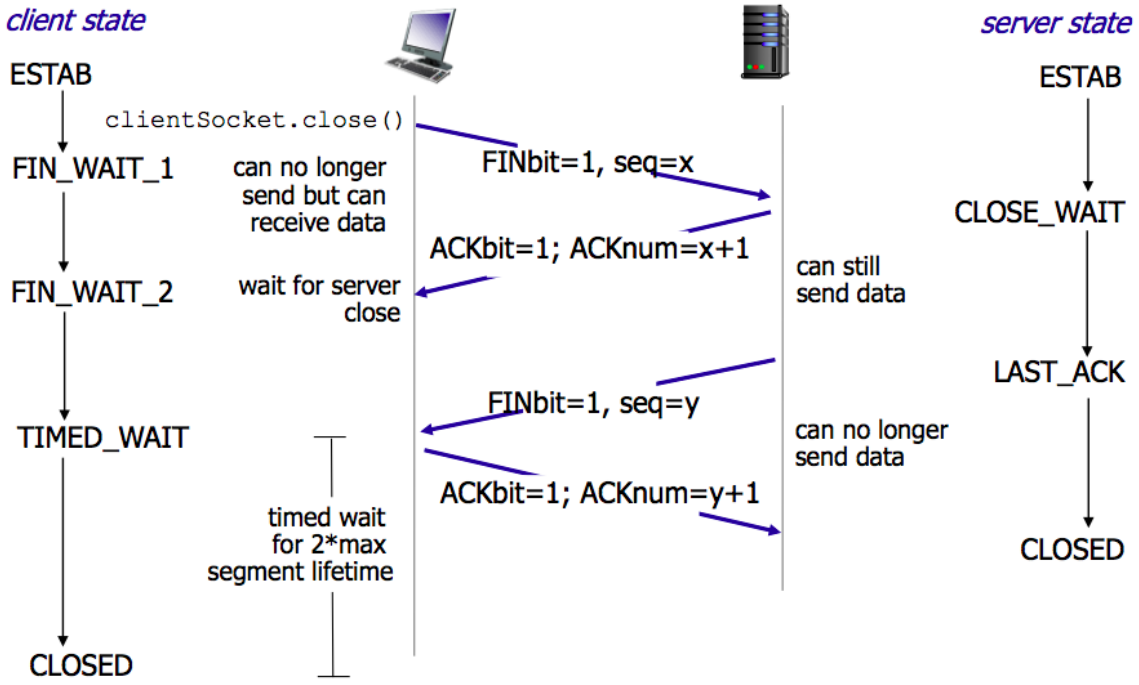
Database Read Phenomena

https://en.wikipedia.org/wiki/Isolation_(database_systems)
1. Dirty read: occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.
2. Non-repeatable read: during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.
3. Phantom read: in the course of a transaction, new rows are added or removed by another transaction to the records being read.

网络

TCP三次握手，四次挥手的过程





为什么客户端要在TIME_WAIT状态等待一段时间

这TIME_WAIT状态的等待时间就是为了防止服务端没有收到客户端发送的第四次挥手的ACK数据包，从而向客户端重新发送第三次挥手的FIN数据包时，客户端能正常接收到FIN数据包并且向服务端发送第四次挥手的ACK应答数据包。

TIME_WAIT状态一般等待的是2MSL的时长

CLOSING状态

Reference
CLOSING: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

如果server端没有收到第三次ack，但是收到了client端发送的数据，server端会怎么处理

由于Server没有收到ACK确认，因此会重发之前的SYN+ACK（默认重发五次，之后自动关闭连接），Client收到后会重新传ACK给Server。如果 **Client**向服务器发送数据，服务器会以**RST**包**(**用于强制关闭**tcp**连接**)**响应。

讲一讲get和post

- GET产生的URL地址可以被Bookmark，而POST不可以。
- GET请求会被浏览器主动cache，而POST不会，除非手动设置。
- GET请求只能进行url编码，而POST支持多种编码方式。
- GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
- GET请求在URL中传送的参数是有长度限制的，而POST么有。
- 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
- GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
- GET参数通过URL传递，POST放在Request body中。

Http状态码

- 1xx：指示信息--表示请求已接收，继续处理。
- 2xx：成功--表示请求已被成功接收、理解、接受。
- 3xx：重定向--要完成请求必须进行更进一步的操作。
- 4xx：客户端错误--请求有语法错误或请求无法实现。

- 5xx：服务器端错误--服务器未能实现合法的请求。

常见状态代码、状态描述的说明如下。

- 200 OK：客户端请求成功。
- 400 Bad Request：客户端请求有语法错误，不能被服务器所理解。
- 401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
- 403 Forbidden：服务器收到请求，但是拒绝提供服务。
- 404 Not Found：请求资源不存在，举个例子：输入了错误的URL。
- 500 Internal Server Error：服务器发生不可预期的错误。
- 502 网关错误 (Bad gateway)、504 Gateway Time-out。
- 503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常

TCP和UDP

- TCP的主要特点是：
    - 面向连接。
    - 每一条TCP连接只能是点对点的（一对一）。
    - 提供可靠交付的服务(无差错，不丢失，不重复，且按序到达)(校验和、重传控制、序号标识、滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。)。
    - 有拥塞控制。
    - 提供全双工通信。
    - In-order delivery
    - 面向字节流。
    - Header size is 20 bytes.
- UDP的主要特点是：
    - 无连接。
    - 尽最大努力交付(不保证可靠交付)。
    - 面向报文。
    - 无拥塞控制。
    - No guarantee for in order packet delivery
    - 支持一对一、一对多、多对一和多对多的交互通信。
    - Header size is 8 bytes. 首部开销小（只有四个字段：源端口、目的端口、长度、验检和）。

采用TCP，一旦发生丢包，TCP会将后续的包缓存起来，等前面的包重传并接收到后再继续发送，延时会越来越大。

UDP对实时性要求较为严格的情况下，采用自定义重传机制，能够把丢包产生的延迟降到最低，尽量减少网络问题对游戏性造成影响

一次HTTP请求的过程（输入URL到返回的全过程）

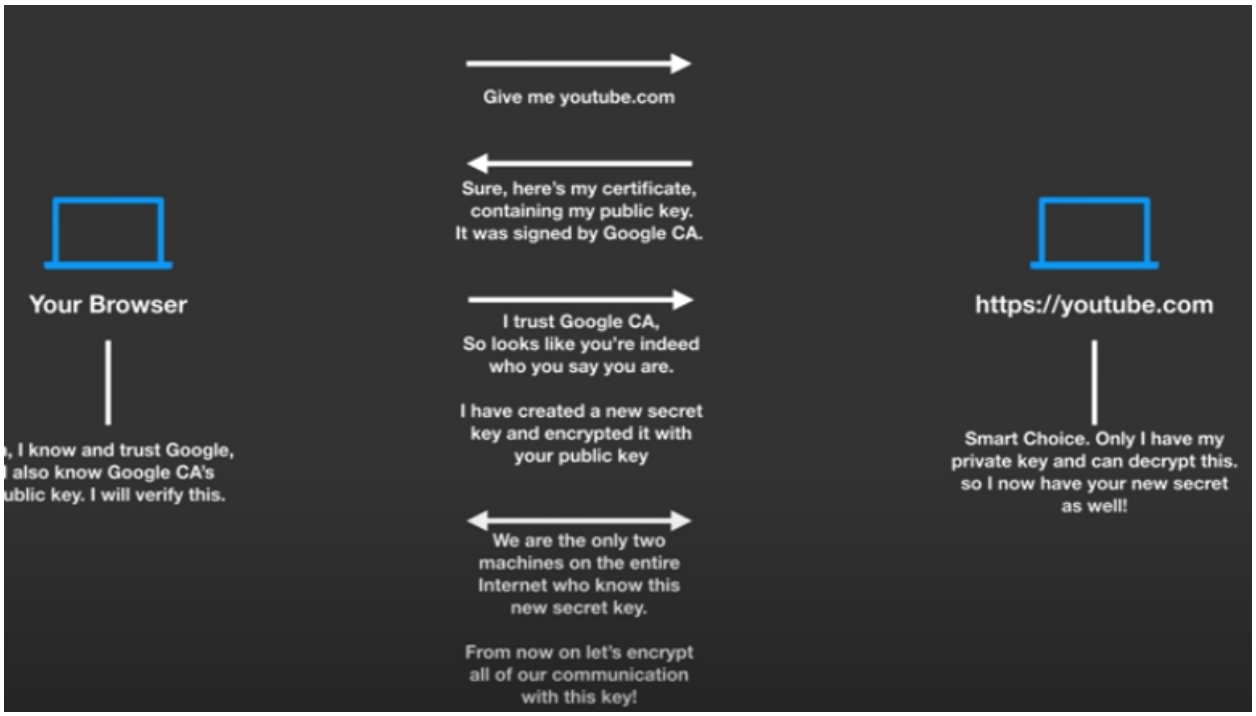
- 根据域名，进行DNS的域名解析
- 得到解析的IP地址，建立TCP连接
- 向目标地址发送HTTP请求
- 服务器处理请求
- 服务器返回响应结果
- 关闭TCP连接
- 浏览器解析HTML响应并对页面进行渲染

Http和https

http是超文本传输协议，是一种从服务器传输html文件到本地浏览器的协议。http有几个安全性问题明文传输，容易被截取;没有数据完整性的校验;也没有对服务器身份的校验。

https是为了解决这些问题而产生的，它本质上还是http协议，但是使用ssl证书来验证服务器的身份并为客户端与服务器之间的通信进行加密。

首先客户端与服务端建立ssl连接。服务端会返回一个网站的证书给客户端。客户端通过证书中心 CA的公钥开数字证书，就可以验证证书是否合法，有没有过期，以及网址是否是当前访问的网 址。如果验证通过后，浏览器会生成一个随机数，并用证书解密得到的公钥来加密这个数，然后发 送给服务器。服务器用自己的私钥解密之后，就可以将之后与客户端的通信内容都用这个随机数秘 钥的对称加密算法来加密。



无状态和无连接

1. 无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间. HTTP 的设计者有意利用这种特点将协议设计为请求时建连接、请求完释放连接，以尽快将资源释放出来服务其他客户端。随着时间的推移，网页变得越来越复杂，里面可能嵌入了很多图片，这时候每次访问图片都需要建立一次 TCP 连接就显得很低效。后来，Keep-Alive 被提出用来解决这效率低的问题。Keep-Alive 功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接

2. 无状态是指协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。即我们给服务器发送 HTTP 请求之后，服务器根据请求，会给我们发送数据过来，但是，发送完，不会记录任何信息。HTTP 是一个无状态协议，这意味着每个请求都是独立的

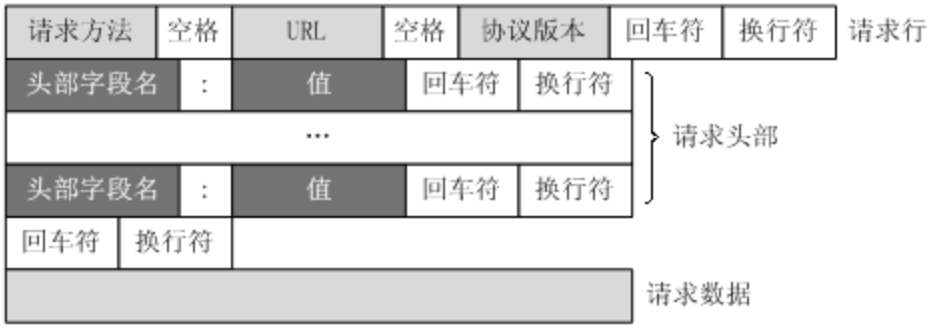
OSI七层模型，各层有哪些协议

OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	HTTP、TFTP, FTP, NFS, WAIS、SMTP
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	数据链路层	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

HTTP协议 请求报文结构

Reference

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.请求头: 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。
HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT
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，典型的请求头有：
User-Agent:产生请求的浏览器类型。
Accept:客户端可识别的内容类型列表。
Host:请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。
3. 空行
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

TCP如何保证可靠传输，丢包怎么办

为了满足TCP协议不丢包。TCP协议有如下规定：

1. 数据分片:发送端对数据进行分片，接受端要对数据进行重组，由TCP确定分片的大小并控制分片和重组
2. 到达确认：接收端接收到分片数据时，根据分片数据序号向发送端发送一个确认
3. 超时重发：发送方在发送分片时设置超时定时器，如果在定时器超时之后没有收到相应的确认，重发分片数据
4. 滑动窗口：TCP连接的每一方的接受缓冲空间大小固定，接收端只允许另一端发送接收端缓冲区所能接纳的数据，TCP在滑动窗口的基础上提供流量控制，防止较快主机致使较慢主机的缓冲区溢出
5. 失序处理：作为IP数据报来传输的TCP分片到达时可能会失序，TCP将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层;
6. 重复处理：作为IP数据报来传输的TCP分片会发生重复，TCP的接收端必须丢弃重复的数据;
7. 数据校验：TCP将保持它首部和数据的检验和，这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到分片的检验或有差错，TCP将丢弃这个分片，并不确认收到此报文段导致对端超时并重发

UDP想要可靠怎么实现 RUDP

- 增加确认报文ACK

- 增加发送队列和接收队列
- 增加超时重传和请求重传的机制，选择重传(XOR的冗余包)
- 增加拥塞控制
- 优化传输路径(传输前本地计算中间节点的延迟或阻塞程度，动态选择最佳的传输链路)

滑动窗口

Reference

滑动窗口控制: 窗口的大小就是无需等待ACK包的情况下还能发送的最大数据量。利用缓冲区 保存已经发送数据。利用下一次的ACK包可以确定接受端有没有收到之前的数据，如果有就清除缓冲区内的数据，

半连接攻击

半连接:
在三次握手过程中，服务器发送SYN-ACK之后，收到客户端的ACK之前的TCP连接称为半连接(half-open connect).此时服务器处于Syn_RECV状态.当收到ACK后，服务器转入ESTABLISHED状态.

SYN攻击属于DOS攻击的一种，它利用TCP协议缺陷，通过发送大量的半连接请求，耗费CPU和内存资源。

SYN攻击利用TCP协议三次握手的原理，大量发送伪造源IP的SYN包也就是伪造第一次握手数据包，服务器每接收到一个SYN包就会为这个连接信息分配核心内存并放入半连接队列，如果短时间内接收到的SYN太多，半连接队列就会溢出，操作系统会把这个连接信息丢弃造成不能连接，当攻击的SYN包超过半连接队列的最大值时，正常的客户发送SYN数据包请求连接就会被服务器丢弃。目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。每种操作系统半连接队列大小不一样所以抵御SYN攻击的能力也不一样。
Syn攻击是一个典型的DOS攻击。检测SYN攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击

防御:
- 缩短超时（SYN Timeout）时间
- 增加最大半连接数
- 修改重试次数： 重传次数设置为0，只要收不到客户端的响应，立即丢弃该连接，默认设置为5次
- 过滤网关防护
- 限制单IP并发数： 使用iptables限制单个地址的并发连接数量
- SYN cookies技术

TCP粘包和拆包

拆包和粘包是在socket编程中经常出现的情况，在socket通讯过程中，

- 粘包：如果通讯的一端一次性连续发送多条数据包，tcp协议会将多个数据包打包成一个tcp报文发送出去。
- 拆包：而如果通讯的一端发送的数据包超过一次tcp报文所能传输的最大值时，就会将一个数据包拆成多个最大tcp长度的tcp报文分开传输。

粘包指的是接收端的应用程序取缓存时无法区分出两个包各自的大小。TCP中是TCP是面向流的，包和包之间没有明显界限。原因有两种，

- 一种是发送端的第一个包过小，系统自动合成两个包为一 个包发送。
- 第二个是接收端的读缓存不及时。

拆包指的是发送端的一个包在接收端的时候被拆成了两个包。原因可能是：

- 发送的数据大于以太网包允许的最大值。
- 还可能是因为发送端发送的数据超过了缓冲区的剩余大小。

粘包和拆包的本质原因都是无法区分包的大小界限。解决有以下几种方式:

- 消息数据定长，不足补空格。接受放解析定长数据后就是完整的数据。
- 采用特定的分割符号做为分割。
- 把TCP包消息数据区分消息头和消息体，消息头中保存了消息的长度

Cookie和session是什么，为什么要用cookie和session

Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；
Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式

原因：由于**HTTP**协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session.典型的场景比如购物车，当你点击下单按钮时，由于HTTP协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的Session，用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。这个Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有

思考一下服务端如何识别特定的客户？这个时候Cookie就登场了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户

不同点：

- Cookie在客户端，Session在服务器端
- Cookie只能存储较少的ASCII数据，Session可以存任意数据。

- Cookie有效期比较长，在浏览器中一般长期保存。Session有效期比较短，在用户会话结束之 后立刻失效。
- Cookie的安全性比较差，因为存放在用户端。

DNS协议和作用

DNS协议是用来将域名转换为IP地址

DNS的查询方式

第一种是本地解析，就是客户端可以使用缓存信息就地应答，这些缓存信息是通过以前的查询获得的；
第二种是直接解析，就是直接由所设定的DNS服务器解析，使用的是该DNS服务器的资源记录缓存或者其权威回答（如果所查询的域名是该服务器管辖的）；
第三种是递归查询，即设定的DNS服务器代表客户端向其他DNS服务器查询，以便完全解析该名称，并将结果返回至客户端。
第四种是迭代查询，即设定的DNS服务器向客户端返回一个可以解析该域名的其他DNS服务器，客户端再继续向其他DNS服务器查询。

ARP协议作用、工作方式

reference
ARP协议作用是把对方主机的IP地址解析为MAC地址
工作原理是发送ARP广播来解析对方主机的MAC地址，解析的结果存储到本机ARP缓存列表中，先查看ARP缓存表，在进行ARP广播查找
(1)每台主机都会在自己的ARP缓冲区中 建立一个 ARP列表（地址转换表），以表示IP地址和MAC地址的对应关系；

(2)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；

(3)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

(4)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。

三次握手过程中是否存在安全问题？描述一下存在什么样的安全问题？针对这样的安全问题如何防御？

1）SYN flood 泛洪攻击，伪装的IP向服务器发送一个SYN请求建立连接，然后服务器向该IP回复SYN和ACK，但是找不到该IP对应的主机，当超时时服务器收不到ACK会重复发送。当大量的攻击者请求建立连接时，服务器就会存在大量未完成三次握手的连接，服务器主机backlog被耗尽而不能响应其它连接。即SYN flood泛洪攻击

防范措施：

1、降低SYN timeout时间，使得主机尽快释放半连接的占用

2、采用SYN cookie设置，如果短时间内连续收到某个IP的重复SYN请求，则认为受到了该IP的攻击，丢弃来自该IP的后续请求报文

3、在网关处设置过滤，拒绝将一个源IP地址不属于其来源子网的包进行更远的路由

2）Land 攻击，当一个主机向服务器发送SYN请求连接，服务器回复ACK和SYN后，攻击者截获ACK和SYN。然后伪装成原始主机继续与服务器进行通信，目标地址和源地址都是目标本身,自己联系自己

防火墙防御

CSRF攻击

CSRF攻击:浏览器无法区别发送请求的是不是用户本人。浏览器存储了网站A的Cookie信息，然后 用户在钓鱼网站上点击了某个标签，这个标签发送了一个向网站A的请求。

解决方法:验证码，Token，请求来源限制，自定义HTTP头熟悉

TCP如何进行拥塞控制？拥塞控制如何判断发生拥塞？

https://zhuanlan.zhihu.com/p/37379780

- **慢开始(Slow Start)**：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(Congestion Window)的大小。也就是每轮发送后将拥塞窗口加倍。为了防止cwnd增长过大引起网络拥塞，还需设置一个慢开始门限ssthresh状态变量。ssthresh的用法如下：
  - 当cwnd<ssthresh时，使用慢开始算法。
  - 当cwnd>ssthresh时，改用拥塞避免算法。
  - 当cwnd=ssthresh时，慢开始与拥塞避免算法任意
- 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。
- 快重传算法：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，而不必继续等待为其设置的重传计时器到期。无论是在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞，就把慢开始门限ssthresh设置为出现拥塞时的发送窗口大小的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为ssthresh减半后的值，执行拥塞避免算法

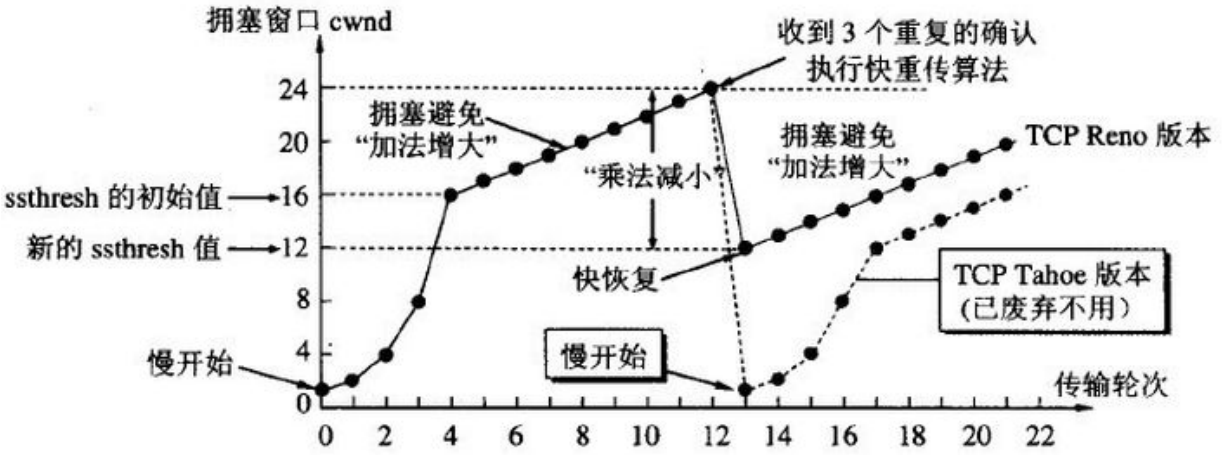


图 5-27　从连续收到三个重复的确认转入拥塞避免

如何得知发生拥塞：发生丢包。如果接收端接收到了不顺次的数据包，ACK中会指明当前需要的下一个包的序号。当发送方收到了三次同样的ACK，就知道发生了丢包，触发重传机制。

TCP快重传如何判断丢失？

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，可提高网络吞吐量约20%）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

CDN?

content distribution network (CDN), is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and performance by distributing the service spatially relative to end users.

Socket 编程

Socket 是对 TCP/IP 协议族的一种封装，是应用层与TCP/IP协议族通信的中间软件抽象层。从设计模式的角度看来，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议

长连接和短连接以及他们分别适用的场景

测试

1.对测试开发的认识，为什么想做测试开发，测试测开的理解

我认为测试开发是在测试这个岗位上拥有一定开发能力和经验的工程师。在传统的开发流程中，测试更多地是承担着功能性验证和查错的部分。而目前的产品迭代的速度在不断加快，如何更高效地保证产品的质量是测试开发需要解决的问题。比如说通过自动化测试或者开发测试工具的方法来加快产品验证的速度。

2. 测试有哪些方法

- 从测试的设计方法上可以分为黑盒测试和白盒测试:
    - 黑盒测试主要是把被测产品作为一个整体，根据功能需求来设计测试用例。黑盒测试不关心软件内部的结构，只关心输入和输出的正确性。
      常用方法:等价类划分，边界值，因果图，决策表
    - 白盒测试是已知产品的内部工作的过程，根据程序的内部结构来检测其内部各个部件的交互是否正常。
      常用方法:逻辑覆盖(语句覆盖、分支覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖，路径覆 盖)

- 从目的的不同可以分为功能性测试，性能测试，可用性测试，兼容性测试，安全性测试
  https://zhuanlan.zhihu.com/p/34605365

3. 讲一讲接口测试

目的:检测外部系统与被测系统之间以及各个子系统之间数据的交互和控制的过程，以及他们逻辑
上的依赖关系。利用一些工具模拟发包和抓包的过程

4. 黑盒测试的方法

等价类划分，边界值，流程图，因果图，决策表

5. 白盒测试的方法

https://www.zhihu.com/question/385083557/answer/1128108136

- 语句覆盖 :对程序的逻辑覆盖很少，只关心判定表达式的值，是很弱的逻辑覆盖标准
- 判定覆盖: 判定覆盖比语句覆盖强一些，能发现一些语句覆盖无法发现的问题。但是往往一些判定条件都是由多个逻辑条件组合而成的，进行分支判断时相当于对整个组合的最终结果进行判断，这样就会忽略每个条件的取值情况，导致遗漏部分测试路径。
- 条件覆盖: 条件覆盖使得判定中的每一个条件都取到了不同的结果，这一点判定覆盖则无法保证。但条件覆盖也有缺陷，因为它只能保证每个条件都取到了不同结果，但没有考虑到判定结果，因此有时候条件覆盖并不能保证判定覆盖。
- 判定/条件覆盖:说白了就是我们设计的测试用例可以使得判断中每个条件所有的可能取值至少执行一次（条件覆盖），同时每个判断本身所有的结果也要至少执行一次（判定覆盖）。不难发现判定条件覆盖同时满足判定覆盖和条件覆盖，弥补了两者各自的不足，但是判定条件覆盖并未考虑条件的组合情况。
- 条件组合覆盖 : 条件组合覆盖，测试用例应该使得每个判定中的各个条件的各种可能组合都至少出现一次。显然，满足条件组合覆盖的测试用例一定是满足判定覆盖、条件覆盖和判定条件覆盖的。
- 路径覆盖:路径覆盖，意思是说我们设计的测试用例可以覆盖程序中所有可能的执行路径。这种覆盖方法可以对程序进行彻底的测试用例覆盖，比前面讲的五种方法覆盖度都要高。

中断测试

中断测试有人为中断、新任务中断以及意外中断等几种情况，主要从以下几个方面进行验证:
a.来电中断:呼叫挂断、被呼叫挂断、通话挂断、通话被挂断
b.短信中断:接收短信、查看短信
c.其他中断:蓝牙、闹钟、插拔数据线、手机锁定、手机断电、手机问题(系统死机、重启)

软件测试的流程

测试需求分析和文档审查 → 设计测试计划，并进行同行评审 → 测试设计(用例编写，测试脚本编写，开发，测试场景的编写)并进行同行评审 → 测试执行(包括执行测试的用例，执行测试的脚 本，进行测试的开发，对测试场景的执行) → 发现bug，进行处理 → 回归测试，重复再次执行上 述测试 → 出测试报告 → 测试验收 → 测试总结

微信红包

- 功能测试
  1.红包最多可以输入的金额；
  2.红包一次性可以发送的最大个数；
  3.在输入红包的钱数和个数时只能输入数字；
  4.当余额不足时，红包发送失败；
  5.发送的红包自己是否可以领取；
  6.发送的红包别人是否可以领取；
  7.红包超过24小时是否可以领取；
  8.红包超时未领取，是否退回原账户；

9.是否可以自己选择支付方式；
10.红包描述可以输入的最大字符；
11.余额不足时，是否可以切换支付方式；

- UI界面测试
  1.输入界面是否清晰可见；
  2.红包界面颜色搭配是否美观；
  3.输入金额界面是否有错别字；
- 性能测试
  1.发红包成功后的跳转时间；
  2.红包超时未领取后的退款时间；
  3.网速较差时，发红包的时间；
  4.耗电量， 消耗流量多少， 所占内存；
- 安全测试
  1.红包发送成功后，微信是否会收到通知；
  2.红包发送失败，余额不会产生变化；
- 弱网测试
  1.弱网下是否可以发送红包；
  2.弱网下发送红包的时间；
- 易用性测试
  1.是否可以选择默认支付方式；
  2.余额不足时，是否可以切换支付方式；
  3.是否支持密码支付和指纹支付；
- 兼容性测试
  1. 苹果手机和安卓手机
  2. 苹果手机的不同版本
  3. 安卓手机不同的机型
  4. 不同分辨率

微信评论功能

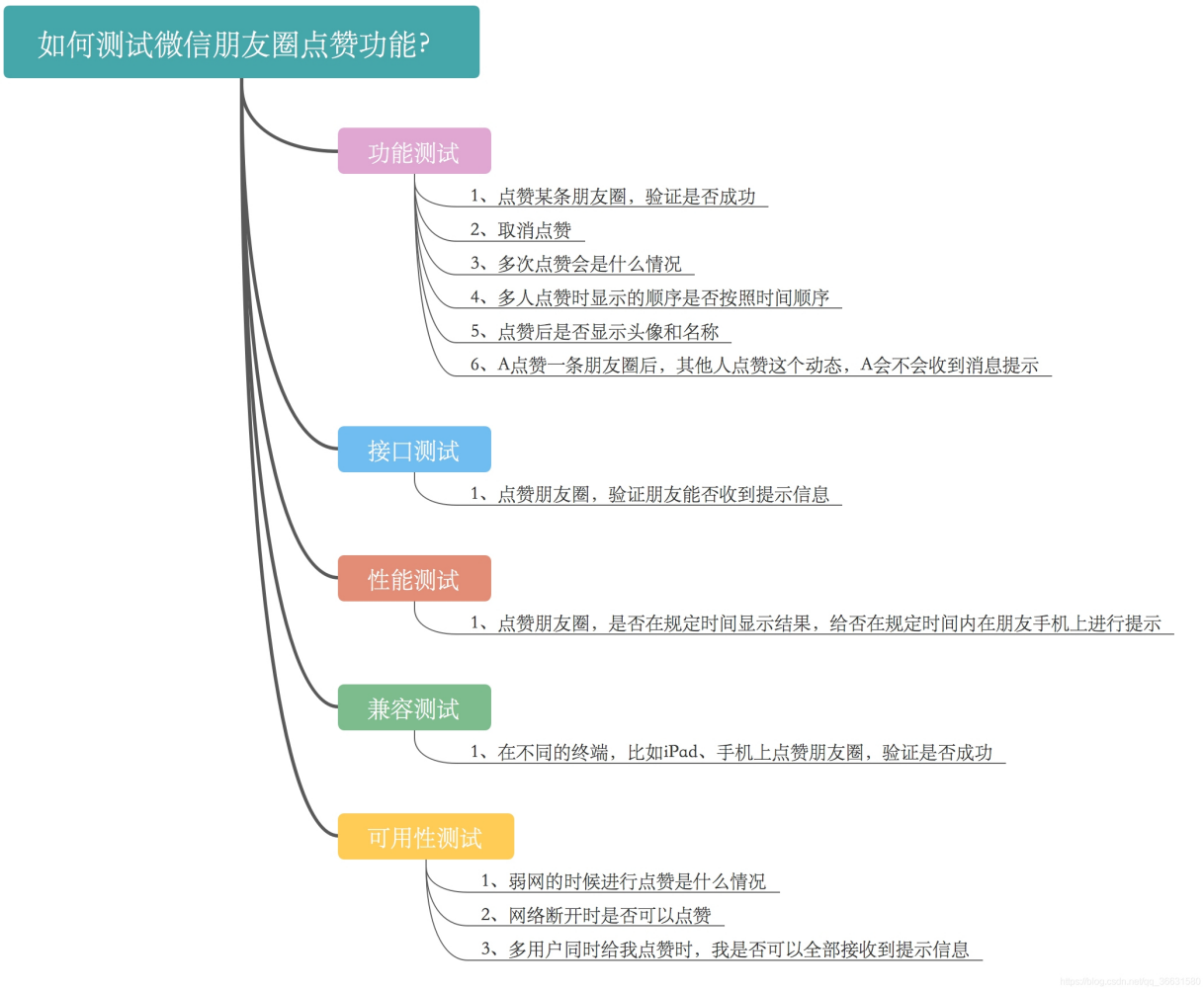
- 功能测试
  评论和删除评论功能
  能否回复评论
  评论能否按时间先后顺序显示
  评论是否有上限
  评论本身能否被其他人评论后者点赞
  能否及时刷新 共同好友能否看得到评论，非共同好友能否看到评论状态 以及能否回复
  a、评论长度：评论字数合理长度、评论超过字数上限
  b、评论类型：纯中文、纯数字、纯字母、纯字符、纯表情（微信表情/手机自带表情）、混合类型、包含url链接；
  c、评论是否支持复制粘贴
  d、为空验证
  e、发表评论后删除

f、评论回复操作

- 界面
  评论能否显示评论人的昵称，若能显示是否正确
  是否能将评论全部显示在朋友圈下面
- 性能
  被评论方和其共同好友多久收到动态消息提醒
  多用户同时评论，服务器相应时间
- 可靠性
  网速快慢对其影响
  弱网的时候进行评论是什么情况
  网络断开时是否可以评论
  评论时有短信或电话进来，能否显示点赞情况
  用户评论几秒后可以看到评论显示成功，取消同理
  多用户同时给我评论时，我是否可以全部接收到提示消息

- 安全性
  评论的人是否在可见分组里
  能否在未登陆时查看评论信息
  共同好友能否看得到评论，非共同好友能否看到评论状态
- 兼容性
  不同手机如何显示
  可扩展性
  是否可以既评论又点赞
  评论和点赞后是怎样现实的，分两次显示，还是一次显示

微信点赞



微信聊天

微信聊天框的主要功能就是发送消息和接收别人发过来的消息。
消息的分类：纯文字，图片，文件，表情，语音、视频，文字+表情
聊天的特殊功能:@符号，撤回功能，加好友功能，消息重发，发红包，转账，发送位置信息、发送名片、群聊等功能

- 功能测试：

  正常网络情况下，发送纯文字，图片，文件，表情，语音、视频，文字+表情消息，发送 和接收功能是否正常
  正常网络情况下，测试图片，文件，语音，视频，文字的最大值测试
  正常网络情况下，是否支持群发文件、群聊文件
  正常网络情况下，是否支持语音转文字

正常网络情况下，发送语音聊天、视频聊天时，是否有声音提示
正常网络情况下，发送失败后，是否支持消息重发
正常网络情况下，消息发送后，是否支持在一定时间范围内可以撤回功能，超出指定时间范围，是否不支持撤回功能
正常网络情况下，发送名片、发送位置信息功能是否正常
正常网络情况下，发送红包、转账功能是否正常
正常网络情况下，语音聊天、视频聊天相互转换功能是否正常
正常网络情况下，发送语音聊天、视频聊天时，长时间未接听，是否有提示
正常网络情况下，未加好友情况下，加好友功能是否正常
正常网络情况下，互相是好友的情况下，发消息功能是否正常
正常网络情况下，连续发消息消息统计数量功能正常
正常网络情况下，消息较多时，是否支持一键回到之前浏览位置
正常网络情况下，群聊：发送消息是否所有成员全部可见
正常网络情况下，群聊：@单个人，多个人，全部人时，对象是否会收到提醒
正常网络情况下，群聊：发起群视频，群语音时，所有被邀请的成员是否能加入群聊
正常网络情况下，群聊：群聊数量统计是否正确

- 性能测试：发送图片、文字、视频、语音等消息，对方收到的时间，是否在需求时间之内

- 兼容性测试：不同输入法，不同手机型号，不同手机系统，不同浏览器，不同电脑型号与版本

- 界面测试：双方头像显示，消息显示是否正常， 聊天记录是否按时间顺序显示

- 弱网测试：在网络条件不好或者无网络的情况下各类功能是否正常，视频聊天、语音聊天是否有提示

- 场景组合测试：
  给网络条件不好或者无网络的好友发消息，恢复网络条件时，是否能接收正常
  给网络条件不好或者无网络的好友发语音、视频消息时，恢复网络条件时，是否有提示
  正在编辑文字消息时，语音、视频聊天中断结束后，是否回到正在编辑的聊天框
  正在语音、视频聊天时，电话或者短信进入，是否会有提示
  语音、视频聊天时，手机进入低电量模式，是否会有提示

微信搜索

- 功能测试：
  - 搜索内容为空、空格、关键字中间有空格
  - 在允许的字数范围内搜索、 字数范围外搜索，是否截取或其它正确处理；
  - 输入特殊字符，表情符号，url链接
  - 输入框复制和粘贴功能
  - 取消搜索:未输入取消，已输入内容取消
  - 输入敏感词是否有提示
  - 多次搜索同样的内容

    - 指定内容:搜索朋友圈、公众号、音乐、表情，结果验证
    - 语音：语音输入是否成功、语音输入是否准确，准确率是多少
    - 点击搜索后键盘是否收起
    - 查看搜索结果是否能正常跳转
    - 数据分页展示是否正确、无重复
    - 无网络时搜索界面是否能展示正确
    - 被删除的内容是否能被搜索出来
    - 关闭搜索功能后搜索入口消失

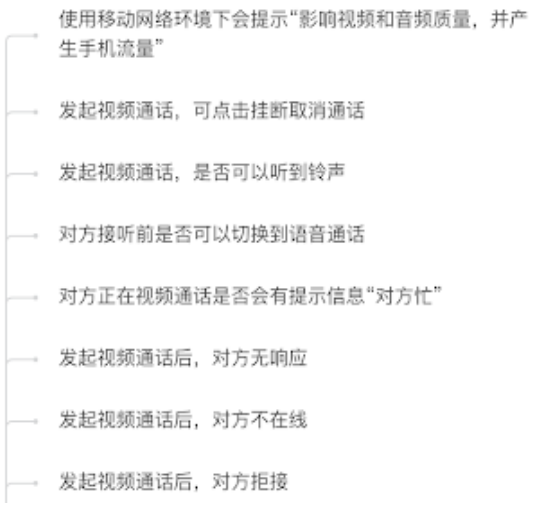
- 接口测试
    - 边界值字符长度限制，输入超出长度的字符
    - sql注入（SQL注入就是一种通过操作输入来修改后台SQL语句达到代码执行进行攻击目的的技术。）
    - 验证接口返回结果是否符合需求

- 性能测试：
    - 接口平均响应时间
    - 搜索数据量超大时响应时间测试
    - 并发测试，监控接口性能和服务器性能

- 兼容性测试：
    - 移动端：安卓、IOS\PAD
    - PC端:windows、mac
    - web端:浏览器兼容性
- 安全测试：
    - sql注入
    - 脚本禁用
    - 敏感词禁用

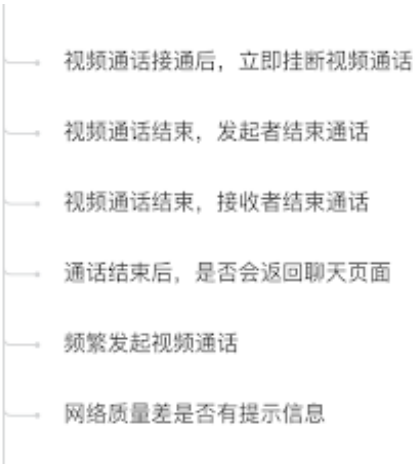
微信视频通话

- 功能性:
    - 发起视频

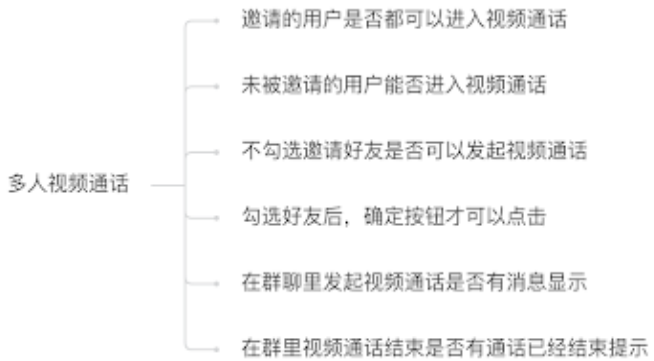


- 视频通话中

- 视频通话结束


视频通话接通后，立即挂断视频通话
视频通话结束，发起者结束通话
视频通话结束，接收者结束通话
通话结束后，是否会返回聊天页面
频繁发起视频通话
网络质量差是否有提示信息

- 多人视频


多人视频通话
邀请的用户是否都可以进入视频通话
未被邀请的用户能否进入视频通话
不勾选邀请好友是否可以发起视频通话
勾选好友后，确定按钮才可以点击
在群聊里发起视频通话是否有消息显示
在群里视频通话结束是否有通话已经结束提示

- 异常情况


异常
无网络能否发起视频通话
无网络能否接受视频通话
网络质量不好的情况下发起视频通话
网络质量不好的情况下接受视频通话
视频通话中有新的视频邀请
视频通话中断电
视频通话中断网
视频通话中来电话、短信
视频通话时手机重启
视频通话中手机死机
视频通话时微信版本升级

- 其他

微信发送朋友圈

- 功能测试
  - 只发送文本

    a、考虑文本长度：1-1500字符（该数据为百度数据）、超出最大字符长度

    b、考虑文本类型：纯中文、纯数字、纯字母、纯字符、纯表情（微信表情/手机自带表情）、混合类型、包含url链接；因为过长纯类型需要换行很容易出现超出边框问题，所以这里先考虑过长纯类型情况

c、文本是否支持复制粘贴

d、为空验证

- 只发送图片

a、本地相册选择/拍摄

b、图片数量验证：1-9张图片、超出9张

c、图片格式验证：常见图片格式jpg、png（以实际微信需求支持的格式为准）、动态gif图片、不支持的图片格式

d、图片尺寸验证：最大700*800像素(此为百度数据)、超出最大尺寸范围是否压缩

e、图片大小验证：1-300kb（此为百度数据）、超出300kb

f、图片的预览验证：点击支持预览大图、多张图片支持左右滑动预览

g、图片的增删改操作

h、为空验证

- 只发送视频

本地相册选择/拍摄

视频秒数验证：1-10s，超出10s

视频个数验证：1个，超出1个

视频格式验证：支持的视频格式，例mp4、不支持的视频格式

视频大小验证：苹果400kb以内、Android200-300kb（此为百度数据）、超出指定大小

视频预览增删改操作

为空验证

- 发送文本+图片：输入满足要求的文本、图片进行一次验证
- 发送文本+视频：输入满足要求的文本、视频进行一次验证
- 发送图片+视频：不支持发送
- 朋友圈发送内容是否有限制，例如涉及黄赌毒等敏感字
- 所在位置

a、不显示位置：发送到朋友圈动态不显示位置

b、选择对应位置：搜索支持、自动定位、手动编辑

C、点击取消，返回上一级页面

- 谁可以看

a、设置公开：所有朋友可见

b、设置私密（仅自己可见）：自己查看朋友圈-可见、好友查看朋友圈-不可见

c、设置部分可见（部分朋友可见）：选择的部分好友-可见、不被选择的好友-不可见、是否有人数上限

d、设置不给谁看（选中的朋友不可见）：不被选中的朋友-可见、被选中的朋友-不可见、是否有人数上限

e、点击取消，返回发送页面

- 提醒谁看

a、提醒单人/提醒多人：被提醒的朋友-收到消息提醒、未被提醒-未有消息提醒

b、是否有人数上限

c、点击取消，返回发送页面

12）取消发送朋友圈操作

a、选择相机，点击取消，返回朋友圈页面

b、进入朋友圈发送页面，选择文本图片，点击取消

- 朋友圈当天发送次数是否有上限限制

- 界面/易用性测试

  1、技术人员角度：页面布局设计是否跟产品原型图/ui效果图一致

  2、但除了考虑1之外，我们同样要考虑到用户使用：功能操作是否简便，页面布局排版风格是否美观合理，提示语相关信息是否易于理解

- 中断测试

  1、主要考虑：a)核心功能 b)当前功能存在实时数据交换，例发朋友圈、浏览朋友圈进行中断，是否容易出现崩溃

  2、中断包括:前后台切换、锁屏解锁、断网重连、app切换、来电话/来短信中断、插拔耳机线/数据线

- 网络测试

  1、三大运营商不同网络制式测试

  2、网络切换测试：WIFI/4G/3G/2G

  3、无网测试：对于缓存在本地的数据，部分朋友圈信息是否支持浏览

  4、弱网测试：

  a、延时：页面响应时间是否可接受、不同网络制式是否区分超时时长、出现请求超时，是否给予相应的提示

  b、丢包：有无超时重连机制、如果未响应，是否给予相应提示

  c、页面呈现的完整性验证

- 兼容性测试

  1、Android手机端、苹果手机端、pad版（主流）功能界面显示是否正常

  2、各平台朋友圈展示数据是否一致

- 安全测试

  发送朋友圈时，文本输入脚本代码，是否出现异常

- 性能测试

  1、服务器性能测试

  可通过loadrunner/jmeter工具实现，主要关注TPS、响应时间、吞吐量、CPU内存等

  2、app客户端性能测试

  可通过GT工具实现，运行时关注cpu、内存、流量、电量等占用率

  3、app压力稳定性测试

  通过monkey工具实现，频繁发送朋友圈，浏览朋友圈请求，是否容易发生崩溃

支付宝付款码

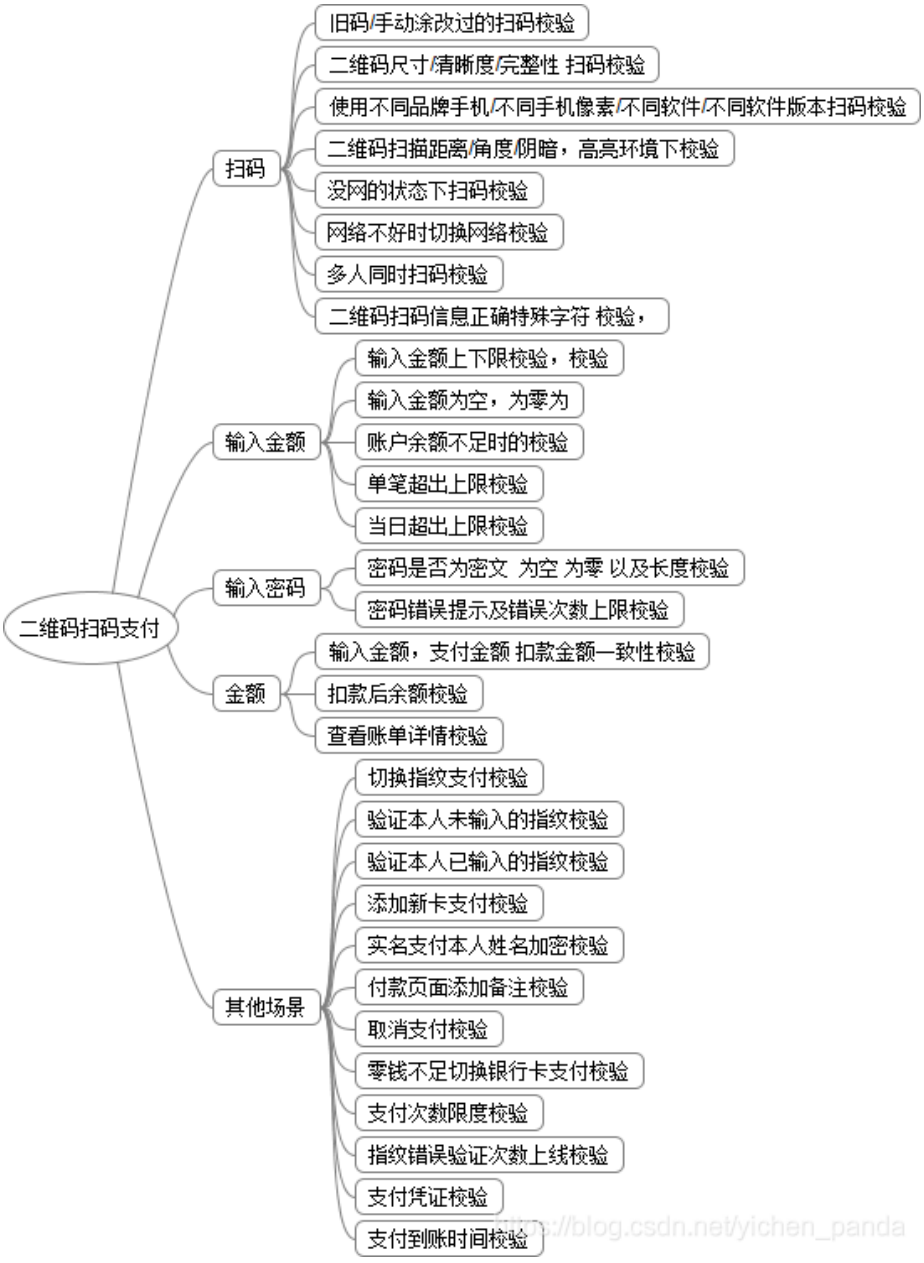
用户角度：

- 功能性测试：
  用户能否成功生成用于支付的二维码或二维码；
  二维码出现后屏幕能否变成增亮的模式；
  用户能否成功选取不同的付款方式，比如“花呗”、“账户余额”、“余额宝”、“银行账户；
  扫码完成后，用户能否收到支付成功的界面，并且界面能正确显示用户支付的金额，包括付款信息、是否使用优惠、折扣等；
- 界面测试：打开支付宝后，能否正确显示界面，二维码的界面是否正确，支付的每个步骤界面是否正确。不会出现前端界面错误，或者打开不了界面。
- 易用性测试：在整个用户支付的过程中，操作步骤是否简易方便。
- 兼容性:测试扫码支付功能，在不同手机品牌，不同操作系统下是否兼容。
- 安全测试：二维码如果超过安全时间后能否自动更新为新的二维码。测试整个支付流程的安全机制能否成功实现。
- 压力测试：持续的扫码，测试扫码支付功能在强压的状态下，工作状态如何。
- 网络测试：测试在不同网络环境下，不同网络信号强度的情况下，整个支付流程是否出现卡顿，卡顿的点容易出现在哪里。

商家角度：

- 功能性测试：扫码枪能否成功扫到用户手机中的二维码，扫码成功后能否收到钱，并且成功生成收款的界面。支付宝后台、商家后台、用户手机能否成功传输支付结果信息。
- 易用性测试：在不同光线，屏幕不同亮度的情况下，能否成功完成扫码收款的功能。

- 其他测试与用户测试差不多。



如何测试弹幕？？？？

淘宝购物车

- 基本功能
  购物车页面的所有连接是否正常。
  - 添加购物车
    从商品信息页面添加的商品能显示在购物车中；
    若未登录，点击购物车，跳转至登陆界面；若已登录，点击购物车，跳转至购物车结算页面
    购物车有无最大金额或者商品数量限制；
    若没有选择任何商品，点击结算，则提示用户“请添加要结算的商品”。

  - 管理购物车
    商品可以全选、取消全选、任意勾选等
    勾选商品后，已选商品的总价（和优惠满减活动）会显示。
    勾选商品，点击结算按钮后，进去确认订单信息页面。
    对于有限购的情况，选择的商品数量>上限时，会自动更改为上限值
    可以在购物车中重新修改商品规格和属性。
    购物车能添加的商品种类是有数量上限的。
    结算的时候商品可以全选，选择底部的全选按钮。
    可以在购物车页面对宝贝进行管理， 收藏等
    购物车中如果存在有商品降价、库存不足、限购件数等，在商品详情的下面，会有对应的字体展示。
- 性能测试
  打开购物车时间是否在已定的用户可以棘手的时间范围内。
  编辑购物车：删除、添加商品需要的时间。
  在购物车页面选择需要购买的商品进行结算的时候，结算金额可不可以实时显示。
  清空失效商品需要的时间。
- 兼容性测试
  iOS:不同型号，不同的iOS系统。

安卓:不同品牌，不同型号，不同的安卓系统。

- 界面测试
  打开淘宝购物车页面后，页面的布局是否合理，是否完整。
  不同卖家的商品在不同的table区域显示，区分明显。
  页面的功能按钮可以正常显示。
  商品的最下方显示失效宝贝。
  页面的最低端显示“你可能喜欢”
  向下滑动页面，在购物车顶端展示“购物车”。

- 网络环境
  3G、4G、WiFi网络环境下应用的各功能可正常运行。
  网络异常时，数据交换是否会有提醒。
  中途断网再很快连网，数据是否可以自动恢复，正常加载。
  只允许内网访问的APP，在连接到外网时是否会有提醒。
- 异常测试
  没有内存时，APP是否能够正常相应。
  横竖屏切换展示。
  APP运行时网络中断。
  反复操作某一个功能，不断点击和刷新，是否出现闪退。
  APP运行时接入电话、短信、社交软件的信息提示时，是否能够正常运行。

    -

抖音短视频

- 功能测试：
    - 视频的播放暂停是否正常
    - 视频的上下滑动是否正常
    - 视频是否画音同步
    - 视频播放过程中是否有锁屏
    - 已经加载过的视频，再次滑到，能否立即播放，或者是否需要一点时间才可以
    - 视频之间的切换是否有黑屏
    - 视频页面刷新时，视频是否正常刷新
    - 视频加载是否是先一张封面然后再加载视频还是别的效果？
    - 视频加载是否完全
- 界面测试：
    - 布局是否合理，高度是否符合要求，点赞按钮是否大范围遮挡了视频画面
    - 布局美感，是否符合人的审美标注
    - 界面中文字简洁易懂，无错别字。
- 性能测试：
    - 首帧响应时间：即用户首次加载视频，获取首个完整关键帧所需的时长，通俗的理解就是从用户点击播放按钮到出现第一帧视频画面所需要的时间
    - 视频清晰度

    - 播放流畅度：即视频播放过程1秒钟时间里显示的图片的帧数，也可以理解为图形处理器每秒钟能够刷新几次
    - 续航能力：即在手机满电的情况下，持续播放视频待机时长。待机时间越长越好
    - CPU指标：即视频播放器在启动和播放视频过程中，CPU所占用的情况
    - 播放稳定性：视频在播放过程中，不会因为播放时间长而导致视频播放质量的下降，主要包括音视频同步、画面的质量、流畅度、响应时间等指标。
- 安全性测试：
    - 视频内容监控
    - 被隐藏的视频是否会被泄漏
- 可用性测试：
    - 快捷键的使用是否正常，比如上下左右键
    - 输入用户名，密码后，回车是否可以登录。
    - 输入框是否能用Tab切换
- 兼容性测试：
    - 不同的平台下是否能正常工作，比如Windows，Mac
    - 移动设备上是否正常工作，比如phone，Android
    - 不同语言环境下，页面的显示是否正确

抖音app用户登录

- 功能测试：

  1.输入框的功能：
  在点击"我",界面
  点击遇到问题,是否可以跳转页面,点击返回是否可退出当前页面
  是否可以点击"x"关闭"我"界面
  点击手机号码登录是否成功跳转界面
  在手机号码登录界面时,点击国家代码框是否可以跳转页面,默认是否中国代码
  在选择国家代码页面,是否中国优先排列,其他是否按照英文字母顺序排列,上滑下拉到尽头是否有提示样式出现
  在选择国家代码页面中,是否点击"x"可以退出当前页面,
  在手机验证码登录页面,输入手机号,每4位是否有隔断,少输多输错输空格入是否有提示
  在手机验证码登录页面,是否可以点击手机号码输入框的"x"全部删除所输入内容
  在手机验证码登录页面,没有对手机号号码输入框进行编辑,"x"是否出现
  在手机验证码登录页面,没有输入手机号码或验证码时,是否输入框会有提示信息出现,比如"请输入号码""请输入验证码"
  当不输/输入/输错正确手机号码后,"获取验证码"颜色是否转变,是否具有实际功能,
  当点击获取验证码后,"获取验证码"是否转变读秒字体,且相应开始读秒,
  当点击获取验证码后,是否弹出"收不到短信?获取语音验证码",且"获取语音验证码"字体颜色转变,以提示用户
  点击获取语音验证码,是否具有实际功能,是否弹出对话框,提示"我们将以...注意接听",是否有关闭对话框按钮,关闭对话框按钮是否具有实际功能

是否默认或可以勾选"我已阅读并同意xxxxx"
不选择"我以默认xxxxxxx"是否可以输入正确号码及验证码后登录抖音
我已阅读并同意XXXXX,字段中相应字体内容是否变色,以提示用户重要性
点击"我已阅读并同意xxxxx"相应内容是否可以跳转页面,显示相关内容
当读秒结束后,点击登录按钮,是否可以成功登录,是否有提示"验证码过期,请重试"字体显示
当读秒结束后,读秒字体是否转变成"重新发送",
点击"重新发送"是否具有实际意义,字体是否转变读秒字体,
不输入内容,点击登录是否可以正常登录,是否有提示用户输入相应内容

点击密码登录(和验证码登录重复的case不写)
密码输入空格/错误/非规定内字符后,点击登录是否有相应提示出现
密码输入过长过短点击登陆后是否会有提示
不输入手机号,点击忘记密码,是否有提示出现
输入正确的手机号,点击忘记密码,是否跳转页面,
验证登录的次数是否有限制

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

- 界面测试：
  - 布局是否合理, 2个testbox和一个按钮是否对齐，高度是否符合要求
  - 布局美感, 是否符合人的审美标注
  - 界面中文字简洁易懂，无错别字。
- 性能测试：
  - 打开登录页面，需要几秒
  - 输入正确的用户名和密码后，登录成功跳转到新页面，花费多少时间
- 安全性测试：
  - 登录成功后生成的Cookie，是否是http only
  - 用户名和密码是否通过加密的方式，发送给Web服务器

    - 用户名和密码的验证，应该是用服务器端验证，而不能只在客户端JavaScript验证
    - 用户名和密码的输入框，应该屏蔽SQL注入攻击
    - 用户名和密码的输入框，应该禁止输入脚本（防止XSS攻击）
    - 错误登录的次数限制（防止暴力破解)
    - 考虑是否支持多用户在同一机器上登录
    - 考虑一用户在多台机器登录
- 可用性测试：
    - 快捷键的使用是否正常，比如上下左右键
    - 输入用户名，密码后，回车是否可以登录。
    - 输入框是否能用Tab切换
- 兼容性测试：
    - 主流的浏览器下是否显示正常，功能正常
    - 不同的平台下是否能正常工作，比如Windows，Mac
    - 移动设备上是否正常工作，比如phone，Android
    - 不同语言环境下，页面的显示是否正确

水瓶

矿泉水瓶的外观测试

- 对于矿泉水瓶的外观测试——主要是两方面，水瓶的大小、水瓶外观设计，那具体一点来说，我们可以列举出以下一些用例：
  1)瓶子的高度、底座是否符合设计要求；
  2)瓶子的口径是否符合设计要求；
  3)瓶身上的字体颜色、大小是否符合设计要求，是否有错别字；
  4)瓶身上的纹路是否符合设计要求；
  5)瓶身上图标位置、间距是否符合设计要求；
  6)瓶子是否有异味；

- 矿泉水瓶的功能测试：对于矿泉水瓶来说，功能测试主要就是测试水瓶的装水、喝水的功能，具体如下：
  1)检查水瓶在装少量水、半瓶水、装满水的情况下是否会漏水；
  2)检查水瓶在装少量水、半瓶水、装满水的情况下能否喝到水；
  3)在瓶盖拧紧不漏水的情况下，分别让成年男性、成年女性和小孩尝试拧开瓶盖，看是否成功；

- 矿泉水瓶的性能测试：对于矿泉水瓶来说，性能测试主要是测试水瓶的抗摔、抗压、抗高低温等情况，具体如下：
  1)将空瓶、半瓶水、满瓶水分别从不同的高度摔下来，看瓶子是否摔坏；
  2)成年人从各种角度按压空瓶，看瓶子是否漏水或破裂；
  3)将空瓶和半瓶水、满瓶水分别放置于冰箱、室内和太阳光下一段时间，观察瓶子是否漏水，瓶身是否破裂；

- 矿泉水瓶的安全性测试： 对于矿泉水瓶来说，安全性测试主要是测试水瓶使用时是否会伤害到人、是否产生对人体有害物质等，具体如下：
  1)用手去抚摸瓶身的内壁和外壁，是否感觉光滑舒适不刺手；
  2)用矿泉水瓶喝水，并转动瓶口，感受瓶口是否圆滑；
  3)空瓶长时间放置一段时间，看是否产生塑化剂或细菌；
  4)矿泉水瓶分别装满不同液体（水、碳酸饮料、果汁等），放置一段时间，看瓶身是否与液体之间发生化学反应，是否产生有毒物质或细菌；
  5)矿泉水瓶中装入热水（或放入微波炉），观察瓶子是否变形，是否有异味产生

- 矿泉水瓶的易用性测试
  1)用手轻拿装满水的瓶子，看是否轻易掉落，是否有防滑措施；
  2)矿泉水瓶分别装入不同温度的水，用手感受瓶身温度，看是否会烫手；
  3)分别将水瓶放在手中、口袋、包包、车上等不同位置，看是否方便携带

- 矿泉水瓶的兼容性测试
  1)瓶中分别装入碳酸饮料、果汁、咖啡、茶水等液体，方式一段时间后看是否变味；
  2)瓶中是否可以装入固体

电梯

1.功能:上升、下降、停止、开门、关门、梯内电话、灯光、指示灯等；
2.性能:速度、反应时间、关门时间等；
3.压力：超载、尖锐物碰撞电梯壁等；
4.安全:停电、报警装置、轿箱停靠位置、有人扒门时的情况等；
5.可用性:按键高度、操作是否方便、舒适程度等；
6.UI:美观程度、光滑程度、形状、质感等；
7.稳定性：长时间运行情况等；
8.兼容性:不同电压是否可工作、不同类型电话是否可安装等。

- 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
10.电梯门关闭过程中有人在外/在内按开

- 可靠性:
  1.门关上的一刹那出现障碍物（大中小）。
  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
- 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
  2.在一定时间内不断让电梯上升、下降
  3.电梯连续运行1天/1小时
- 稳定性测试：
  看电梯在最大负载下平稳运行的最长时间