1.jwt 的工作原理？

答：1.浏览器发送登录 2.服务端验证身份，根据算法，将用户表示符打包生成 token，并且返回给浏览器，3.浏览器发起请求获取用户资料，把刚刚拿到的 token 一起发送给服务器，4.服务器发送数据有 token 验明正身，5.服务器返回该用户资料。

jwt 三部分：header，jwt 的头部承载两部分信息，一是声明类型，这里是 jwt 二是声明算法，

payload：载荷就是存放有效信息的地方，

signature：jwt 的第三部分是一个签证信息，信息由三部分组成：header（base64） payload（base64）secret 这个部分需要 base64 加密后端 headers 和 base64 加密后的 payload 使用，连接成字符串，然后通过 header 中声明的加密方式进行加盐 secret 组合加密，然后组成了 jwt 第三部分

注意：secret 是保存在服务端的，jwt 的签发生成也是在服务端的，secret 就是用来进行 jwt 的签发和 jwt 的验证，所以，他就是你的服务端的私钥，在任何场景都不应该泄漏出去，一旦客户端得知这个 secret 那就意味着客户端是可以自我签发 jwt 了

2.jwt 和 session 相比有什么区别？各自有什么优势和缺点？

答：1.session 存储在服务端占用服务器资源，而 jwt 存在客户端 2.session 存储在 Cookie 中，存在伪造跨域站请求伪造攻击的风险， 3.session 需要服务

端存储一份，而 jwt 不需要， 4.session 只存储在一台服务器上，那么下次请求就必须请求这台服务器，不利于分布式应用， 5.存储在客户端的 jwt 比存储在服务端的 session 更具有扩展性。

3.什么是 base64 ？

答：Base64 编码是一种“防君子不防小人”的编码方式。广泛应用于 MIME 协议，作为电子邮件的传输编码，生成的编码可逆，后一两位可能有“=”，生成的编码都是 ascii 字符。

4.，如何使用 uuid？为什么要用 uuid？

答：uuid 是通用的唯一识别码的缩写，是一种软件构建的标准，其目的是让，分布式系统中所有的元素，都能有唯一的识别信息，而不需要通过中央控制端来辨别信息的指定，如此一来，每个人都可以创建不与其他人冲突的 uuid，在这样情况下，就不需要考虑数据库创建时的名称重复问题。

ajax 异步发送的时候，拼接到网址字符串 ，发送到后台，用来存储验证码的内容，可以设置 uuid 的存活时间。

5.二分查找？

答：也叫拆半查找称为二分查找，它充分利用了元素件的次序关系，采用分治策略，可在最坏的情况下用 O（log n） 完成搜索任务，它的基本思想是，将 n 个元素分成个数大致相同的两半，取 a[n/2]与欲查找的 x 作比较，如果 x=a[n/2]则找到 x，算法终止，如果 x<a[n/2],则我们只要在数组啊的作伴不继

续搜索 x（这里假设数据元素成升序排列）如果 x>a[n/2] ,则我们只要在数组 a 的右半步继续搜索 x。

6.celery 的用法？为甚要用 celery？

在程序运行的时候，我们经常会遇到一些耗时耗资源的操作，为了避免阻塞主程序，我们会采用异步或者多线程来处理任务，比如在主程序中调用一个函数，并从该函数中获取韩式的返回值，如果这个函数不能很快执行完并返回，那么主程序就会阻塞，直到函数返回，celery 是一个强大分布式队列，它可以让人物完全脱离主程序，甚至可以被分配到其他主机上运行。

celery 包含如下几个模块：任务模块：主要包异步任务和定时任务，异步任务通常在业务逻辑中被触发并发送到对列中，或者定时任务由 celery Beat 进程周期性的将任务发往任务队列。

消息中间件 broker：broker 就是任务调度队列，接收任务生产者发送来的消息，将任务存入队列，之所以需要中间的人原因是 celery 本身是不提供消息队列的服务，所以需要第三方组件实现。

任务执行单元 worker ：worker 是执行任务单元，它实时监控消息队列，获取队列中的调度任务，并执行它。

任务存储结果：backend 用于存储任务执行的结果，以供查询，同消息中间件一样，存储也可使用 rabbitq。redis，MongoDB 等。

celery 的使用：1.定义任务函数，2.运行 celery 服务，3.客户应用程序的调用。

cd 到需要运行的任务文件下，

运行 celery 的命令：celery -A tasks worker --loglevel

调用 delay 函数即可启动要执行的任务，这个函数的效果是发送一条消息到 broker 中，这个消息包括要执行的函数，函数的参数以及其他的信息，这个时候 worker 会等待 broker 中的消息，一旦收到消息会立刻执行消息。

注意：如果把返回值赋值给一个变量，那么原来的应用程序也会被阻塞，需要等待的异步任务返回结果，因此实际使用中不需要把结果赋值。

celery 发布周期性任务，以上描述的发布异步任务：

只需要设置 celery 对象的 CELERYBEAT_SCHEDULE 属性即可，!/usr/bin/env python

-- coding:utf-8 –

from future import absolute_import

CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/5'BROKER_URL = 'redis://127.0.0.1:6379/6'

CELERY_TIMEZONE = 'Asia/Shanghai'

from datetime import timedelta

CELERYBEAT_SCHEDULE = { 'add-every-30-seconds': { 'task': 'proj.tasks.add', 'schedule': timedelta(seconds=30), 'args': (16, 16) },}

注意配置文件的需要指定时区，这段代码表示每隔 30 秒执行 add 函数，一旦使用了 scheduler，启动 celery 需要加上-B 参数 celery -A proj worker -B -l info

7.Django 的中间件：

Django 中的中间件是一个轻量级、底层插件系统，可以介入 Django 的请求和响应处理过程，修改 Django 的输入和输出。中间件的设计为开发者提供了一种无侵入的开发方式，增强了 Django 的健壮性。我们可以使用中间件，在 Django 处理视图的不同阶段对输入或输出进行干预（相当于 flask 的请求钩子）中间件的使用装饰器来实现，使用装饰器来装饰视图函数，完成中间件的功能。

使用中间件的方法：1.定义中间件 2.定义好中间件后，需要在 setting.py 文件中添加注册中间件

3.测试定义的中间件对每个视图函数都有作用，一旦视图函数被调用，中间件就被触发，可以在控制台中查看打印效果。

中间件执行顺序：在请求视图被处理前，中间件由上至下依次执行

在请求视图被处理后，中间件由下至上依次执行

8.Flask 的请求钩子：

1.before_first_request:在第一次请求之前运行，只需运行一次，如连接数据库

2.before_request:在每次请求都会执行，可以在这里权限校验操作，比如说：某用户是黑名单用户，黑名单用户登录系统将遭到拒绝访问，可以使用 before_request 进行权限校验。

3.after_request:在请求之后运行，会接受一个参数，这个参数就是前面的请求处理完毕之后，返回的响应数据，如果需要对响应做额外的处理，可以再这里进行。

4.teardown_request:每一次请求之后都会调用，会接受一个参数，参数是服务器出现的错误信息。

9.COOKIE 的使用：

有时候我们需要保持用户的浏览状态，比如说用户是否登录过，浏览过哪些商品等。

具体使用：服务端先设置 cookie 信息，并在客户端请求是把这个 cookie 信息发送给客户端，客户端会自动保存 cookie 的 key/value 值，下次向服务端发送请求是，客户端会自动带上 cookie 信息，服务端会根据 cookie 来识别状态，（之前是否访问过）

http 是一种无状态协议，浏览器请求服务器是无状态的。

无状态是指一次用户请求是，浏览器、服务器无法知道之前这个用户做过什么，每次请求都是一次新的请求。

无状态的原因：浏览器与服务器是使用 socket 套接字进行通信的，服务器将请求结果返回非浏览器之后，会关闭当前的 socket 连接，而且服务器也会在处理页面之后销毁页面对象。

当浏览器请求某网站的时候，会将本网站下所有的 cookie 信息提交给服务器，所以在 request 中可以读取 cookie 信息。

10.redis 和 session 在 python 框架的使用？

答：我们知道 session 其实是在 cookie 中保存了一个 sessionid，用户每次访问都讲 sessionid 发给服务器，服务器通过 id 查找用户对应的状态数据，在这里我的处理方式也是在 cookie 中定义一个 sessionid，程序需要取得用户状态是将 sessionid 作为 key 在 Redis 中查找。

同时 session 支持用户在一定时间不访问将 session 回收。 借用 Redis 中 key 支持过期时间的特性支持这个功能，但是续期方面徐亚程序自行拦截请求调用这个方法

11.第三方 qq 登录实现？

答：1.注册成为 QQ 互联平台开发者， http://connect.qq.com

2.准备一个可访问的域名，3.创建网页应用，配置必要信息，其中包括域名以及回调地址，其中域名需要验证，需确保对域名主机有足够的控制权限，获取应用 appID、appKey 进行开发 。

整体流程为 A:用户打开客户端以后，客户端要求用户给予授权

B 用户同意给客户端授权 C 客户端使用上一步获得的授权，向认证服务器申请令牌

D 认证服务器对客户端进行认证后，确认无误，同意发放令牌。

E 客户端使用令牌，向资源服务器申请资源，

F 资源服务器确认令牌无误后，同意向客户端开发资源。

12.Fastdfs 的介绍及使用？

答：Fastdfs 是一个开源的轻量级分布式文件系统，它对文件进行管理，功能包括：文件存储，文件同步，文件访问（文件上传和下载），解决的大量存储和负载均衡的问题，特别适合文件为载体的在线服务，如相册网站，视频网站等。

跟踪器和存储节点都可以由一台或多台服务器构成。跟踪器和节点中的服务器可以随时增加或下线而不会影响线上服务器，其中跟踪器中的服务器都是对等的，可以根据服务器的压力情况随时增加或减少。为了支持大容量，存储节点（服务器）采用了分卷的组织方式，存储系统由一个或多个卷组成，卷与卷之间的文件是相互独立的，所有文件容量累加就是整个文件存储系统中文件容量，一个卷可以由一台或者多台存储服务器组成，一个卷下的存储服务器的文件都是相同的，卷中的多台存储服务器器到了冗余备份和负载均衡的作用。

在卷中增加服务器时，同步已有的文件由系统自动完成，同步完成后，系统自动将新增服务器切换到线上提供服务。

当存储空间不够的时或即将耗尽时，可以动态添加卷，只需要增加一台或多台服务器，并将它们配置为一个新的卷，这样就扩大了存储系统的容量。

上传交互过程：1.client 询问 tracker 上传到的 storage，不需要参数

2.tracker 返回一台可用的 storage 3.client 直接和 storage 直接通讯完成文件上传

下载交互过程：1.client 询问 tracker 下载文件的 storage，参数为文件标识（卷名和文件名）

2.tracker 返回一台可用的 storage 3.client 直接和 storage 通讯完成文件下载。

13.docker 的介绍

答：docker 是一个开源的应用容器引擎，让开发者可用打包他们的应用以及依

赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口。

首先了解 docker 就要知道什么什么是虚拟化，虚拟化是一种资源管理技术，是将计算机的各种实体资源，比如服务器，，网络，内存及存储等，予以抽象、转换出来，打破实体结构间的不可切割的障碍，使用户可以比原本的组态更好的方式来应用这些资源，这些资源的新虚拟部分不受现有资源的假设方式，地域物理组态所限制，一般所指的虚拟化资源包括技算能力和资料存储。

docker 的优点：简化程序 避免选择恐惧症 节省开支

14.单例模式在工作中的应用场景？

答：单例模式只允许创建一个对象，因此节省开支，加快对象的访问速度，因此对象需要被公用的场合适用，如多个模块使用同一个数据源连接对象等等。如：

1.需要频繁的实例化然后销毁对象。

2.创建对象时耗时过多或者耗资源过多，但又经常用到的对象。

3.有状态的工具类对象， 4.频繁访问数据库或文件的对象

经典使用场景：1.资源共享的情况下，避免由于资源操作是导致的性能损耗等，如上述的日志文件应用配置等。

15.购物车有满就减活动，数据库是怎么操作的？

答：购物车正常进行增删改查操作，只不过前端把数据进行修改，传给后端。

16.多进程、多线程在项目中哪些地方使用？

答：需要进行大量计算的优先使用线程，大量计算当然很消耗 cpu，切换频繁了这种情况线程最合适，常见的有图像处理，算法处理。

16.怎么解决或者实现高并发？

答：一。Nginx 要做负责均衡 二、程序层面多做线程，锁等机制。

三、数据库层面处理 四、服务器配置要尽量高

17.scrapy 的流程分析？

答：1.构建请求信息（url，method，headers，params，data）

2.发起 HTTP/HTTPS 请求，获取 HTTP/HTTPs 请求信息

3.解析响应，分析响应数据的数据结构或者页面结构提取需要的数据

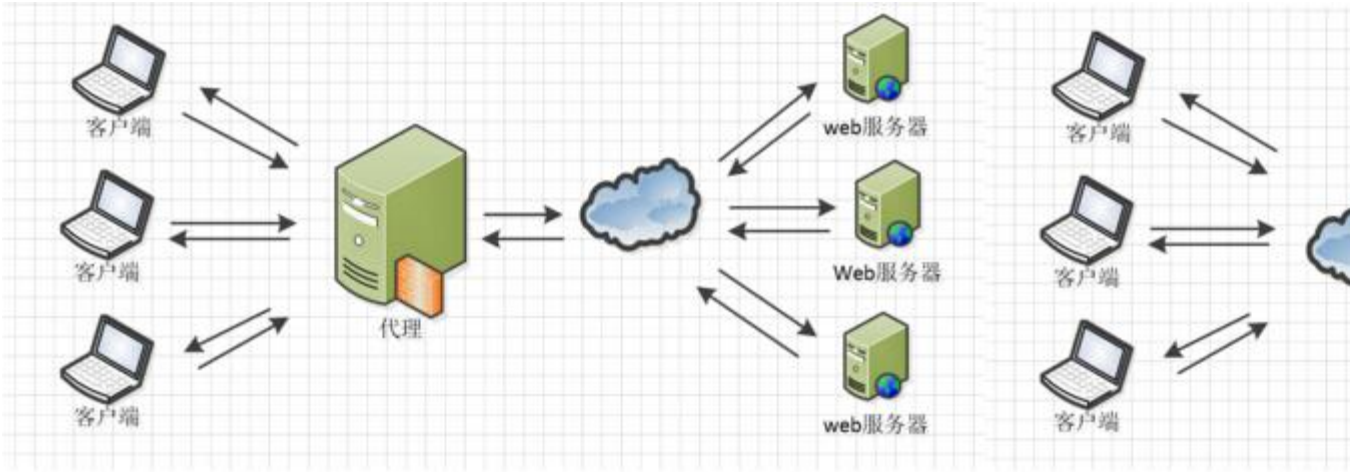
提取想要的数据， 提取新的请求地址

4.对数据进行存储/对新的请求地址重复前面的步骤。

18.Nginx 的讲解？

答：Nginx 功能丰富可以作为 http 服务器，也可以作为反向代理服务器并支持很多第三方模块扩展。1.Nginx 的稳定性，功能集，示例配置文件和低系统资源的消耗。

2.反向代理

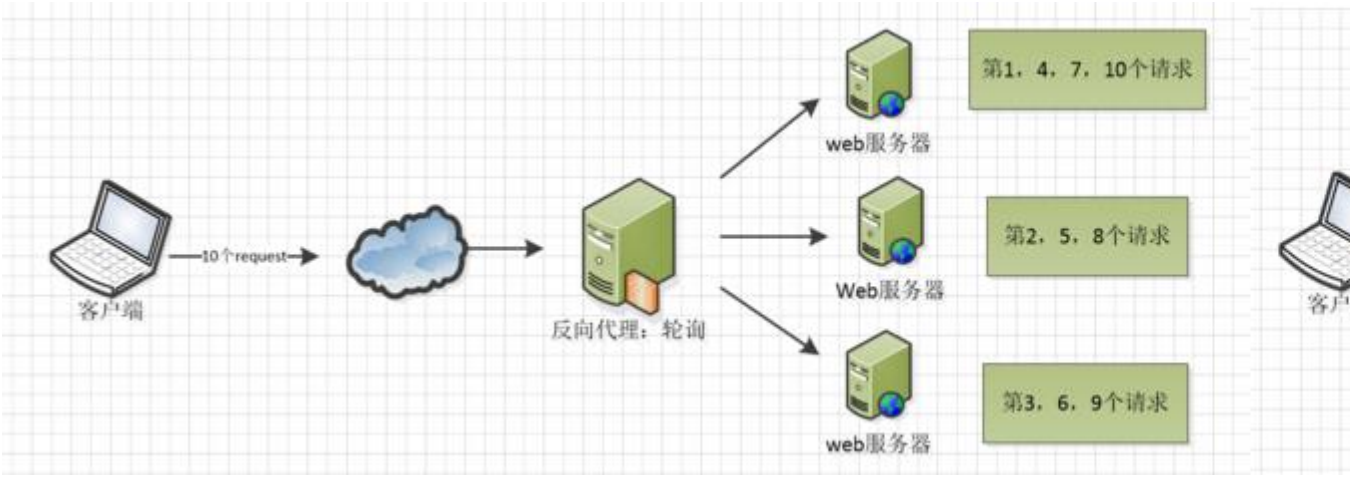


Nginx 可以根据不同的正则匹配，采取不同的转发策略，比如图片文件结尾的走文件服务器，动态页面走 web 服务器，只要你正则写正确，又有相对应的服务器解决方案，并且对 Nginx 的返回结果进行错误也跳转，异常判断等，如果被分发的服务器存在异常，它可以将请求重新转发 给另一台服务器，然后自动去除异常服务器，

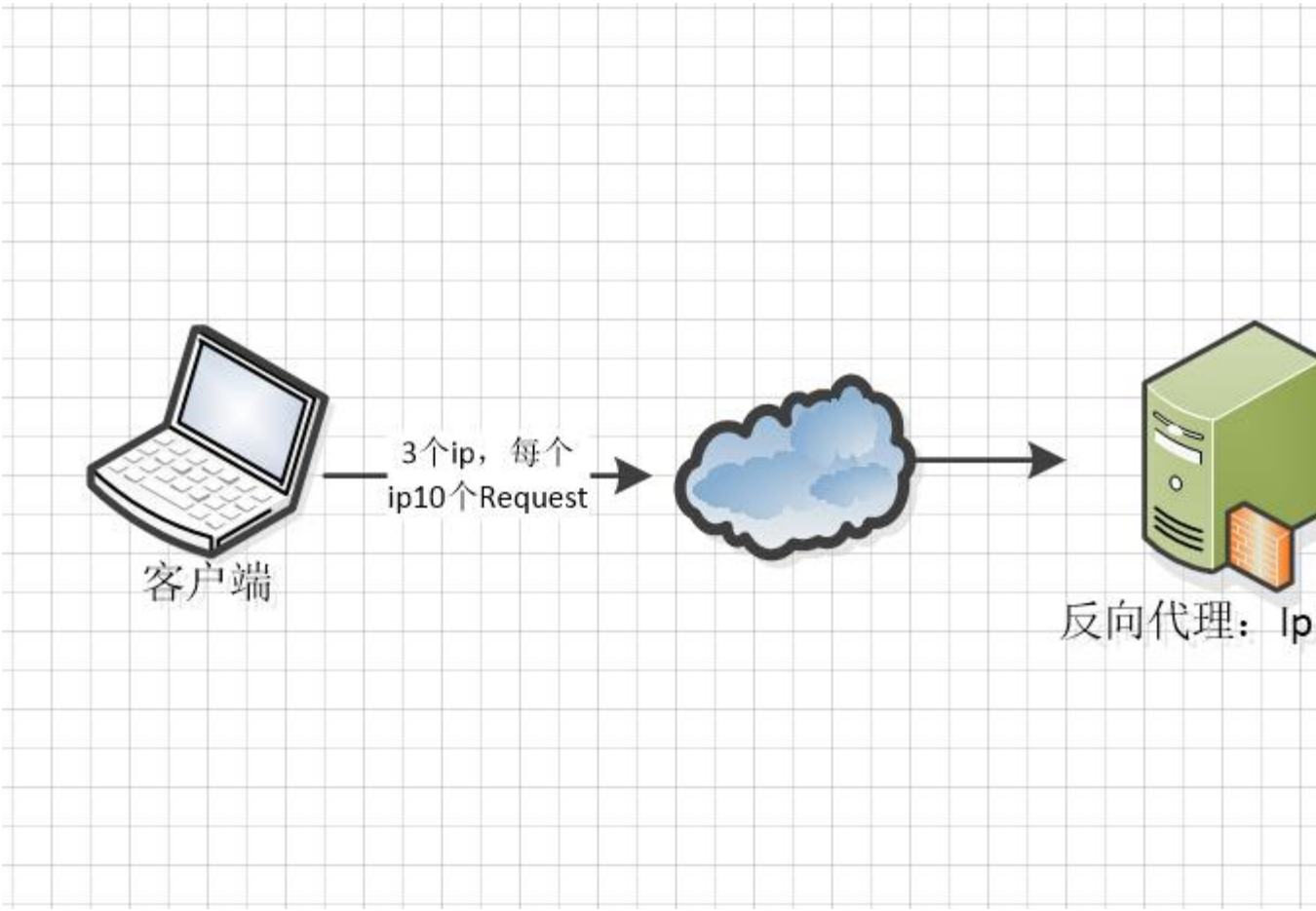
此方法最为便利，前后端代码无需做特殊的改变，只需在 nginx。conf 做配置，将本地请求地址转向真正实现的请求的 url 地址。

3.负载均衡

Nginx 提供的负载均衡策略有两种：内置策略，和扩展策略。内置策略为轮询，加权轮询，扩展策略，就天马行空了，



IP hash 算法，对客户端请求的 ip 进行 hash 操作，然后根据 hash 结果将同一个客户端的 ip 的请求分发给同一台服务器进行处理，可以解决 session 不共享的问题



3.web 缓存

Nginx 可以对不同的文件做不同的缓存处理，配置灵活，并且支持 FastCGI_cache，主要用于对 FastCGI 的动态程序进行缓存，配合着第三方 ngx_cache_purge，对制定的 URL 缓存内容可以的进行增删管理

19.Nginx 和 uwisg 的关系

答：1.首先 nginx 是对外服务的接口，外部浏览器通过 url 访问 nginx，

2.nginx 接收到浏览器发送过来的 http 请求，将进行分析，分析 url，如果是静态文件请求就直接访问用户给 nginx 配置的静态文件目录，直接返回用户请求的静态文件，如果不是静态文件，而是一个动态请求，那么 nginx 就将请求转发给 uwsgi，uwsgi 接收到请求之后将包进行处理，处理成 wsgi 可以接收的格式，并发给 wsgi，wsg 根据请求调用的应用程序的某个文件，某个文件的某个函数，最后处理完成将返回值再次交给 wsgi，wsgi 将返回值进行打包，打包成 uwsgi 能接收的格式，uwsgi 接收 wsgi 发送的请求，并转发给 nginx，nginx 最终将返回值返回给浏览器。

3.要知道第一级的 nginx 并不是必须的，uwsgi 完全可以完成整个的和浏览器交互的流程，但是要考虑到某些情况

a.安全问题，程序不能直接被浏览器访问到，而是通过 nginx，nginx 只开放某个接口，uwsgi 本身是内网接口，这样运维人员在 nginx 上加上安全性的限制，可以达到保护程序的作用。

b。负载均衡问题，一个 uwsgi 很可能不够用，即使开课多个 work 也是不行，毕竟一台机器的 cpu 和内存都是有限的，有了 nginx 做代理，一个 nginx 可以代理多台 uwsgi 完成 uwsgi 的负载均衡。

c.静态问题，用 django 或是这种东西来负责静态文件的处理都是浪费行为，而且他们本身对文件处理也不如 nginx 好，所以整个静态文件的处理都是由 nginx 完成的静态文件的访问完全不经过 uwsgi 以及其后面的东西

4.后台跨域：前端向本地后台发出请求，后台在向 api 服务器请求数据，然后在将请求到的数据返回给前端。

20.网站如何解决高并发？

答：买服务器：使用高性能的服务器，高性能数据库，高效率编程语言，还有高性能的 web 容器；页面静态化，图片服务器分离，数据库集群，负载均衡。

shell 的基本命令

ls chmod ugo+rwx mylife 将 myfile 文件给 u，g，o 用户 r，w，x 的权限

touch 创建一个空文档 find 搜索文件 echo 显示文本行货变量，或者把字符串输入到文件

cat 显示文件 grep 允许对文本文件进行模式查找，如果找到匹配模式，grep 打印包含模式的所有行。

nginx 和 uwsgi 的配置

Git 使用

数据库的存储原理

储存过程是一个可编程的函数，他在数据库中创建并保存，它可以有 SQL 语句和一些特殊的控制结构组成，当希望在不同的应用程序或平台上执行相同的函数，或者封装特定的功能时，存储过程中是非常有用的，数据库的存储过程汇

总可以看做是对编程中面向对象的模拟，它允许控制数据的访问方式，存储过程中有已下优点：

1.存储过程能实现较快的执行速度 2.存储过程中允许标准组件是编程

3.存储过程可以用流程控制语句编写，有很强的灵活性，可以完成负杂的判断和比较复杂的运算

5.存储过程中可以被作为一种安全机制来充分利用

6.存储过程中能够见识网络流量。

数据库的事务特性

1.原子性：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不完成

2.一致性：几个并行的事务，其执行结果必须与按其某一顺序串行执行的结果相同。

3.隔离性：事务的执行不受其他事务的干扰。事务执行中间结果对其他事务必须是透明的。

4.持久性：对于任意已提交的事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障。

数据库的索引：

数据库的索引，是数据库管理系统中的一个排序数据结构，以协助快速查询、更新数据库表中的数据。索引的实现通常使用 B_TREE,B_TREE 索引加速了数据的访问，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。

数据库怎么优化查询效率

答：1.储存引擎选择：如果数据表需要事务处理，应该考虑使用 Innodb，因为它完全符合 ACID 特性，如果不需要事务处理，使用默认存储引擎 MyISAM 是比较明智的

2.分表分库，主从

3.对查询优化，要尽量避免全表扫描，首先考虑在 where 及 order by 涉及的列上建立索引。

mysql 集群的优缺点：

优点：99.%的高可用性 快速的自动失效切换 灵活的分布式体系结构 ，没有单点故障

高吞吐量和低延迟，可扩展性强，支持在线扩展

缺点：存在很多限制，比如不支持外键，部署、管理、配置负复杂

占用磁盘空间大，内存大，备份和恢复不方便 重启的时候，数据节点到内存将需要很长时间

你用的 Mysql 是哪个引擎，各引擎之间有什么区别？

答：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，这一点非常之重要，事务是一种高级处理方式，如在一些列增删改查中只要那个错还可以回滚还原，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的安全性较高的应用。 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，清空整个表示是 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

数据库的优化：

1 优化索引，sql 语句，分析慢查询

2.设计表的时候严格根据数据库的设计范式来设计数据库

3.使用缓存，把经常访问的数据而且不需要经常变话的数据放在缓存中，能节约磁盘 IO

4.优化硬件，采用 SSD，使用磁盘队列技术

5.采用 mysql 内部自带的表分区技术，把数据分成不同的文件，能够提高磁盘的读写效率

6.垂直分表，把一些不经常读的数据放在一张表里，节约磁盘 I/O

7.主从分离读写，采用主从复制把数据库的读操作和写入操作分离开来

8.分库分表分机器（数据量特别大）主要原理就是数据路由

9.选择合适的表引擎，进行参数上的优化

10 进行架构级别的缓存，静态化和分布式

11.不采用全文索引

12.采用更快的存储方式，例如 Nosq 存储经常问的数据

设计数据库表的三范式：

第一范式：强调的是列的原子性，即列不可能分成其他几列

第二范式：首先是 1nf，另外包含两部分内容，一是表必须有一个主键，二是没有包含住主键中的列必须完全依赖于主键，而不能依赖主键的一部分

第三范式：首先是 2nf 的基础上，另外非主键列必须完全依赖于主键，不能存

在传递依赖，既不能存在：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。

关系型数据库和非关系性数据库的区别

答：sql 数据存在特定结构的表中，而 nosql 则更加灵活，存储方式可以是

json 文档、哈希表或其他方式

在 sql 中必须定义好表的字段和结构后才能添加数据，而 nosql 中，数据可以在任何时候任何地方添加不需要先定义表

在相同水平的系统设计下，因为 nosql 中省略了 join 的查询消耗，故理论上性

能上市优于 sql 的

数据库的负载均衡

实现原理：实现数据库的负载均衡技术，首先要有一个可以控制的连接数据的的控制端，在这里，它截断了数据库和程序的直接连接，由所有的程序来访问这个中间层，然后再有中间层来访问数据库，这样我们就可以具体控制访问某个数据库了，然后还可以根据数据库的当前负载采取有效的均衡策略来调整每次连接到哪个数据库

优点：扩展性强， 可维护性强 安全性， 易用性

缺点：不能够按 web 服务器的处理能力分配负载 负载均衡控制端故障，会导致整个数据库系统的瘫痪

MongoDB：

MongoDB 是一个面向文档的数据库系统，使用 c++编写不支持 sql，但有自己的查询语法

MongoDB 使用 BSON 作为数据存储和传输方式，BSON 是一种类似 JSON 的二进制序列化文档，支持嵌套对象和数组。

应用场景：（a）网站数据：mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性

（b）缓存：由于性能很高，mongo 也可适合作为信息基础设施的缓存层，在系统重启之后，由 mongo 搭建的持久化缓存可以避免下层数据源过载

（c）大尺寸、低价值的数据：使用传统的关系数据库存储一些数据是可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储

（d）高伸缩性的场景：mongo 非常适合数十或者数百台服务器组成的数据库

（e）由于对象及 JSON 数据的存储：mongo 的 BSON 数据格式非常适合文档格式化的存储及查询

（f）重要数据：mysql，一般数据，mongo，临时数据：mechache

优点：弱一致性（最终一致）更能保证用户的访问速度

（2）文档结构的存储方式，能够更便捷的获取数

（3）支持复制集，主备，互为主备，自动分片等特性

（4）动态查询

（6）全索引支持，扩展到内部对象和内嵌数组

缺点： 不支持事务， mongddb 占用空间过大，维护工具不够成熟

以下特点使得 MongoDB 成为优秀的 NoSQL 数据库：

1）面向文件的 2）高性能 3）高可用性 4）易扩展性 5）丰富的查询语言

Redis

redis 中的 list 底层实现有哪几种？有什么区别？

列表对象的编码可以是 ziplist 或者 linkedlist

ziplist 是一种压缩链表，他的好处是更能节省内存空间，因为他所存储的内容是在连续的内存区域当中，当列表对象元素不大，每个元素也不大的时候，就采用 ziplist 存储，但当数据量过大是 ziplist 就是不那么好用了，因为为了保证他存储的内容在内存中连续性，插入的复杂度是 o（N）。

怎样解决数据库高并发的问题：

（1）缓存式的 web 应用程序框架：在 web 层和 DB（数据库）层之间加一层 cache，主要目的就是减少数据库的读取负担，提高读取速度

（2）增加 redis 缓存数据库

（3）增加数据库索引 （4）页面静态化 （5）mysql 主从读写分离

Redis 数据库，内容是以何种结构存放在 Redis 中的：string，hash，list，set，zset

Redis 和 MongoDB 的优缺点：

Redis 优点：读写性能优异 支持数据持久化， 支持主从复制 数据结构丰富

缺点：不具备自动容错能力和恢复功能，主机从机制的宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的 IP 才能恢复

主机宕机，宕机前有部分数据未能及时同步到丛机，切换 IP 后还会引入数据不一致的问题降低了系统的可用性

redis 较难支持在线扩容，在集群容量达到上限时，在线扩容会变得很复杂，为避免这一问题，运维人员在系统上线是必须确保有足够的空间，这对资源造成了很大的浪费

MongDB：优点 弱一致性更能保证用户的访问速度 文档结构的存储方式，能够更便捷的获取数， 内置 GridFs,高效存储二进制答对象（比如照片和视频）支持复制集，主备，互为主备、自动分片等特性 动态查询 全索引支持，扩展到内部对象和内嵌数组

缺点：不支持事务 mongoDb 占用空间过大 维护工具不够成熟

Redis 事务

开始事务 命令入队 执行事务

Redis 的使用场景有哪些？

答：1.取最新 N 个数据的操作 2.排行榜应用，取 TOP N 操作

3.需要精准设定过期时间的应用 4.计数应用 5.unip 操作，获取某段时间所有的数据排重值

6.构建队列系统 7.缓存

Redis 默认端口，默认过期时间，Value 最多可以容纳的数据 长度？

答：1.默认端口：6379 默认过期时间：可以说永不过期，一般情况下，当配置中开启了超出最大内存限制就写磁盘的话，那么没有设置过期时间的 key 可能会别写到磁盘上，假设没设置，那么 redis 将使用 lru 机制，将内存中老数据，并写入新数据。value 最最多可以容纳数据长度是 512M

MongoDB 的常见命令如下：

use yourDB：切换、创建数据库

show dbs：查询所有数据库 db.dropDatabase（）：删除当前使用的数据库

db.getName():查看当前使用的数据库

简述 Django 的运行机制

Django 的运行方式：我们在开发和调试过程中经常使用 runserver 方法，使用 Django 自己的 web server，首先 Django 会先根据 settings 中 WSGI_APPLICATION 来获取 handler，在创建 project 的时候，Django 会默

认创建 wsgi.py 文件，而 settings 中的 WSGI_APPLICATION 配置会指向这个文件，

而我们上线则使用 uwsgi 进行运行 Django 项目：uwsgi 是个 web 服务器，它实现了 WSGI 协议，uwsgi，http 等协议，注意 wsgi 是一个通信协议，而 uwsgi 是实现 uwsgi 协议和 WSGI 协议的 web 服务器，uwsgi 具有超快的性能，低内存占用和多 app 管理的特点。

运行流程：1.加载 project settings：在通过 django_admin.py 创建 project 的时候，Django 会自动默认生成 settings 文件和 manae.py 等文件，在创建，WSGIservre 之前会执行 from django。conf import settings ，同时会读取 os.environ 中的 Django_settings _module 配置文件生成 settings 对象，所以，在 manage。py 文件中你可以看到，在获取 WSGIserver 之前，会先将 project 的 settings 路径加载到 os 路径当中。

2.创建 WSGIserver：不管使用 runserver 还 uwsgi 运行 Django 项目，在启动时都会调用 django.core.severs.basehttp 中的 run（）方法，创建一个 django.core.servers.basehttp.WSGIserver 类的实例，之后调用 server_foever()方法启动 HTTP 服务，会创建 WSGIServer 实例的时候会指定 HTTP 请求的 Handler ，在使用 WSGIRequestHandler，当用户的 HTTP 请求到大服务器时，WSGIServer 会创建 WSGIRequestHandler 实例，使用器 handler 方法来处理 HTTP 请求，（其实最终是调用 wsgiref.handler.BaseHandler 中的 run 方法处理）。WSGIServer 通过 set_app 方法设置一个可调用的（callble）的对象作为 application，上面提到

的 handler 方法最终会调用设置的 application 处理 request，并返回 response

详细流程：1.用户通过浏览器请求一个页面 2.请求到达 Request Middlewares 中间件对 request 做一些预处理或者直接 response 请求

3.URLConf 通过 urls.py 文件和请求得 URL 找到相应的 View

4.view Middlewares 被访问，它同样可以对 request 做一些处理或者直接返回 response

5.调用 View 中的函数

6.View 中的方法可以选择性的通过 Models 访问底层的数据

7.所有的 Model_to-db 的交互都是通过 mananger 完成的

8.如果需要，Views 可以使用一个特殊的 Contest

9.Context 被传给 Template 用来生成页面

a.Template 使用 FIlters 和 Tages 去渲染输出

b.输出被返回 View

c。HTTPResponse 被发送到 Response Middlewares

d。任何 Response Middlewares 都可以丰富 response 或者访问一个完全不同的 response

e。Response 返回浏览器 ，呈现给用户。

Django 生命周期：前端发送请求--》Django 的 wsgi--》中间件---》路由系统---》ORM 数据库的操作---》模板----》返回数据给用户

rest framework 生命周期：前端发送请求--》Django 的 wsgi--->中间件---》路由系统执行的 CBV 的 as_view()，就是执行内部的 dispatch 烦烦烦---》在执行 dispatch 之前，有版本分析，和渲染器--》在 dispatch 内，对 request 封装---》版本---》认证---》权限----》限流---》视图---》如果视图用到缓存（request.data or request .query.params）就用到了解析器---》视图处理数据，用到了序列化（对数据进行序列化或验证）---》视图返回数据可以用到分页