**面试提问-题目版**

**python基础**

**11、列举你了解的所有Python2和Python3的区别?**

# py2  
>>> print("hello", "world")  
('hello', 'world')  
# py3  
>>> print("hello", "world")  
hello world  
py2：input\_raw()  
py3：input()  
1/2的结果  
py2：返回0  
py3：返回0.5  
py2：默认编码ascii  
py3：默认编码utf-8  
字符串  
py2：unicode类型表示字符串序列，str类型表示字节序列  
py3: str类型表示字符串序列，byte类型表示字节序列  
py2中函数用关键字global声明某个变量为全局变量，但是在嵌套函数中，想要给一个变量声明为非局部变量是没法实现的。  
py3中，新增了关键字nonlocal,使得非局部变量成为可能

**字符反转**

name="张全蛋 name[::-1]

**16、文件操作时:xreadlines和readlines的区别?**

readlines()是把文件的全部内容读到内存，并解析成一个list，当文件的体积很大的时候，需要占用很多内存  
xreadlines()则直接返回一个iter(file)迭代器，在Python 2.3之后已经不推荐这种表示方法了.直接使用for循环迭代文件对象

**is 和==的区别?**

is比较的是id ；== 比较的是值

**20、字典排序 dict={‘a’:24，‘g’:52，‘i’:12，‘k’:33}请按字典中的 value 值进行排序**

sorted(dict.items()，key = lambda x:x[1])；可选参数reverse=True

**可变不可变数据类型**

可变不可变指的是内存中的值是否可以被改变，不可变类型指的是对象所在内存块里面的值不可以改变，有数字、字符串、元组；可变类型则是可以改变，主要有列表、字典。

**简述字符串驻留机制以及小整数池的概念**

对于短字符串，将其赋值给多个不同的对象时，内存中只有一个副本，多个对象共享该副本。长字符串不遵守驻留机制。驻留适用范围: 由数字，字符和下划线（\_）组成的python标识符以及整数[-5,256]。

**超出索引的切片：下面代码的输出结果将是什么？**

list = ['a'， 'b'， 'c'， 'd'， 'e']  
print(list[10:])  
下面的代码将输出[]，不会产生IndexError错误。就像所期望的那样，尝试用超出成员的个数的index来获取某个列表的成员。例如，尝试获取list[10]和之后的成员，会导致IndexError。

**31、写一个列表生成式，产生一个公差为11的等差数列**

print([x\*11 for x in range(10)])

**32、给定两个列表，怎么找出他们相同的元素和不同的元素?**

list1 = [1，2，3]

list2 = [3，4，5]

set1 = set(list1)

set2 = set(list2)

print(set1&set2)

print(set1^set2)

**33、请写出一段Python代码实现删除一个list里面的重复元素?**

比较容易记忆的是用内置的set：

l1 = ['b'，'c'，'d'，'b'，'c'，'a'，'a']

l2 = list(set(l1))

如果想要保持他们原来的排序： 用list类的sort方法：

1. l1 = ['b'，'c'，'d'，'b'，'c'，'a'，'a']

2. l2 = list(set(l1))

3. l2.sort(key=l1.index)

4. print(l2)

也可以这样写：

1. l1 = ['b'，'c'，'d'，'b'，'c'，'a'，'a']

2. l2 = sorted(set(l1)，key=l1.index)

3. print(l2)

**字符串、列表、元组、字典每个常用方法**

字符串：repleace,strip,split,reverse,upper,lower,join.....

列表：append,pop,insert,remove,sort,count,index.....

元组：index,count,len(),dir()

字典：get,keys,values,pop,popitems,clear,update,items.....

**38、什么是反射?以及应用场景?**

在绝大多数语言当中都有反射机制的存在， 可以用字符串的方式去访问对象的属性，调用对象的方法（但是不能去访问方法），Python中一切皆对象，都可以使用反射：

1）、反射机制是很多框架的基石，如Django。

2）、getattr(), setattr()

**39、简述Python的深浅拷贝?**

copy()：浅copy，浅拷贝指仅仅拷贝数据集合的第一层数据；deepcopy():深copy,深拷贝指拷贝数据集合的所有层

**40、Python 垃圾回收机制?**

垃圾回收机制是自动帮助我们管理内存，清理垃圾的一种工具；

1）、引用计数

当一个对象的引用被创建或者复制时，对象的引用计数加1；

当一个对象的引用被销毁时，对象的引用计数减1；

当对象的引用计数减少为0时，就意味着对象已经没有被任何人使用了，可以将其所占用的内存释放了。

优点：简单、直观、实时性，只要没有了引用就释放资源。

缺点：

维护引用计数需要消耗一定的资源

循环应用时，无法回收。也正是因为这个原因，才需要通过标记-清理和分代收集机制来辅助引用计数机制。

2）、标记-清除

“标记-清除”不改动真实的引用计数，而是将集合中对象的引用计数复制一份副本，改动该对象引用的副本。对于副本做任何的改动，都不会影响到对象生命走起的维护。

3）、分代回收

将系统中的所有内存块根据其存活时间划分为不同的集合，每一个集合就成为一个“代”，垃圾收集的频率随着“代”的存活时间的增大而减小。也就是说，活得越长的对象，就越不可能是垃圾，就应该减少对它的垃圾收集频率。那么如何来衡量这个存活时间：通常是利用几次垃圾收集动作来衡量，如果一个对象经过的垃圾收集次数越多，可以得出：该对象存活时间就越长。

**题外话**

# 打乱列表

import random

random.shuffle(alist)

import sys

sys.version # python 版本

import gc

gc.collect() # 释放内存

print("\n".join("\t".join(["%s\*%s=%s" %(x,y,x\*y) for y in range(1, x+1)]) # 9\*9乘法表

enumerate函数是将一个可迭代对象中元素，按元素顺序每个增加一个索引值，将其组成一个索引序列，利用它可以同时获得索引和值，这样做的目的是为了将一个可迭代对象中元素组成一个“索引,值”对便于后续操作。

**常见的内置函数?**

数学类型

abs(a) : 求取绝对值。abs(-1)

max(list) : 求取list最大值。max([1,2,3])

min(list) : 求取list最小值。min([1,2,3])

sum(list) : 求取list元素的和。 sum([1,2,3]) >>> 6

sorted(list) : 排序，返回排序后的list。

len(list) : list长度,len([1,2,3])

divmod(a,b): 获取商和余数。 divmod(5,2) >>> (2,1)

pow(a,b) : 获取乘方数。pow(2,3) >>> 8

round(a,b) : 获取指定位数的小数。a代表浮点数，b代表要保留的位数。round(3.1415926,2) >>> 3.14

range(a[,b]) : 生成一个a到b的数组,左闭右开。 range(1,10) >>> [1,2,3,4,5,6,7,8,9]

类型转换

int(str) : 转换为int型。int('1') >>> 1

float(int/str) : 将int型或字符型转换为浮点型。float('1') >>> 1.0

str(int) : 转换为字符型。str(1) >>> '1'

bool(int) : 转换为布尔类型。 str(0) >>> False str(None) >>> False

bytes(str,code) : 接收一个字符串，与所要编码的格式，返回一个字节流类型。bytes('abc', 'utf-8') >>> b'abc' bytes(u'爬虫', 'utf-8') >>> b'\xe7\x88\xac\xe8\x99\xab'

list(iterable) : 转换为list。 list((1,2,3)) >>> [1,2,3]

iter(iterable)： 返回一个可迭代的对象。 iter([1,2,3]) >>> <list\_iterator object at 0x0000000003813B00>

dict(iterable) : 转换为dict。 dict([('a', 1), ('b', 2), ('c', 3)]) >>> {'a':1, 'b':2, 'c':3}

enumerate(iterable) : 返回一个枚举对象。

tuple(iterable) : 转换为tuple。 tuple([1,2,3]) >>>(1,2,3)

set(iterable) : 转换为set。 set([1,4,2,4,3,5]) >>> {1,2,3,4,5} set({1:'a',2:'b',3:'c'}) >>> {1,2,3}

hex(int) : 转换为16进制。hex(1024) >>> '0x400'

oct(int) : 转换为8进制。 oct(1024) >>> '0o2000'

bin(int) : 转换为2进制。 bin(1024) >>> '0b10000000000'

chr(int) : 转换数字为相应ASCI码字符。 chr(65) >>> 'A'

ord(str) : 转换ASCI字符

为相应的数字。 ord('A') >>> 65

相关操作

eval() : 执行一个表达式，或字符串作为运算。 eval('1+1') >>> 2

exec() : 执行python语句。 exec('print("Python")') >>> Python

filter(func, iterable) : 通过判断函数fun，筛选符合条件的元素。 filter(lambda x: x>3, [1,2,3,4,5,6]) >>> <filter object at 0x0000000003813828>

map(func, \*iterable) : 将func用于每个iterable对象。 map(lambda a,b: a+b, [1,2,3,4], [5,6,7]) >>> [6,8,10]

zip(\*iterable) : 将iterable分组合并。返回一个zip对象。 list(zip([1,2,3],[4,5,6])) >>> [(1, 4), (2, 5), (3, 6)]

type()：返回一个对象的类型。

id()： 返回一个对象的唯一标识值。

hash(object)：返回一个对象的hash值，具有相同值的object具有相同的hash值。 hash('python') >>> 7070808359261009780

help()：调用系统内置的帮助系统。

isinstance()：判断一个对象是否为该类的一个实例。

issubclass()：判断一个类是否为另一个类的子类。

globals() : 返回当前全局变量的字典。

next(iterator[, default]) : 接收一个迭代器，返回迭代器中的数值，如果设置了default，则当迭代器中的元素遍历后，输出default内容。

reversed(sequence) ： 生成一个反转序列的迭代器。 reversed('abc') >>> ['c','b','a']

**64、filter、map、reduce 的作用?**

filter(func, iterable) : 通过判断函数fun，筛选符合条件的元素。

filter(lambda x: x>3, [1,2,3,4,5,6])

>>> <filter object at 0x0000000003813828>

map(func, \*iterable) : 将func用于每个iterable对象。 map(lambda a,b: a+b,

map(lambda a,b: a+b, [1,2,3,4], [5,6,7])

>>> [6,8,10]

# python3中没有reduce

reduce(): 函数会对参数序列中元素进行累积。

reduce(lambda x, y: x+y, [1,2,3,4,5]) # 使用 lambda 匿名函数

>>> 15

**什么是闭包?**

闭包函数（closure function）指的是定义在一个函数内部的函数，被外层函数包裹着，其特点是可以访问到外层函数中的变量，如下inner函数就是一个闭包函数。

def outer():

num = 1

def inner():

print(num) # 内层函数中不存在num 但可以访问到外层的num

return inner # 基于函数对象的概念我们可以将内层函数返回到外界使用，从而打破函数调用的层级限制，但无论在何处调用，作用域的嵌套关系都是以定义阶段为准的，所以外界得到的不仅仅是一个函数对象（inner），在该函数外还包裹了一层作用域，这使得该函数无论在何处调用，都是访问自己外层包裹的作用域中的名字num

func = outer() # func == inner func指向的是inner的内存地址，但是func本身确实一个全局变量，可以在任意位置调用func，但无论在何处调用func，都需要按照定义阶段作用域的嵌套关系去查找名字

num=1000

func() #输出结果：1

**生成器与迭代器**

略

**标准库中functools.wraps的作用**

# 答案：Python装饰器（decorator）在实现的时候，有一些细节需要被注意。例如，被装饰后的函数其实已经是另外一个函数了（函数名等函数属性会发生改变）。这样有时候会对程序造成一些不便，例如笔者想对flask框架中的一些函数添加自定义的decorator，添加后由于函数名和函数的doc发生了改变，对测试结果有一些影响。

所以，Python的functools包中提供了一个叫wraps的decorator来消除这样的副作用。写一个decorator的时候，最好在实现之前加上functools的wrap，它能保留原有函数的名称和docstring。

**面向对象**

**4、mro 是什么?**

# 答案：mro全称Method Resolution Order，指的是方法解析顺序。方法调用时就需要对当前类和基类进行搜索以确定方法所在的位置。而搜索的顺序就是所谓的「方法解析顺序」。

**静态方法和类方法区别?**

# 答案：Python的类就是个语法糖。一个函数写在类里面和写在类外面没有区别，唯一的区别就是参数，所谓实例方法就是第一个参数是self，所谓类方法就是第一个参数是class，而静态方法不需要额外的参数，所以必须区分。

**isinstance 和 type 的作用?**

# 答案：type和isinstance都可以判断变量是否属于某个内建类型，type只接收一个参数，不但可以判断变量是否属于某个类型，而且可以得到参数变量未知的所属的类型；而isinstance只能判断是否属于某个已知类型，不能直接得到变量未知的所属的类型

**with statement(语句)，好处是什么?**

# 答案：with语句会在嵌套的代码执行之后，自动关闭文件。这种做法的还有另一个优势就是，无论嵌套的代码是以何种方式结束的，它都关闭文件。如果在嵌套的代码中发生异常，它能够在外部exception handler catch异常前关闭文件。如果嵌套代码有return/continue/break语句，它同样能够关闭文件。

如果对象实现了上下文管理协议（context manage）,即实现了\_\_enter\_\_和\_\_exit\_\_方法，那么这个对象就可以使用with。

with 进入的时候走的是\_\_enter\_\_方法，结束时走\_\_exit\_\_

with context\_expression [as target(s)]:

with-body

自定义的上下文管理器要实现上下文管理协议所需要的 \_\_enter\_\_() 和 \_\_exit\_\_() 两个方法

context\_manager.\_\_enter\_\_() ：进入上下文管理器的运行时上下文，在语句体执行前调用。with 语句将该方法的返回值赋值给 as 子句中的 target，如果指定了 as 子句的话

context\_manager.\_\_exit\_\_(exc\_type, exc\_value,exc\_traceback) ：退出与上下文管理器运行，返回一个布尔值表示是否对发生的异常进行处理。

**网络编程与并发**

**简述OSI七层协议。**

# 答案:

应用层：HTTP，FTP，NFS

表示层：Telnet，SNMP

会话层：SMTP，DNS

传输层：TCP，UDP

网络层：IP，ICMP，ARP，

数据链路层：Ethernet，PPP，PDN，SLIP，FDDI

物理层：IEEE 802.1A，IEEE 802.11

**简述TCP三次握手、四次挥手的流程。**

三次握手过程：

1首先客户端向服务端发送一个带有SYN 标志，以及随机生成的序号100(0字节)的报文

2服务端收到报文后返回一个报文(SYN200(0字节)，ACk1001(字节+1))给客户端

3客户端再次发送带有ACk标志201(字节+)序号的报文给服务端

至此三次握手过程结束，客户端开始向服务端发送数据。

1客户端向服务端发起请求：我想给你通信，你准备好了么？

2服务端收到请求后回应客户端：I'ok，你准备好了么

3客户端礼貌的再次回一下客户端：准备就绪，咱们开始通信吧！

整个过程跟打电话的过程一模一样:1喂，你在吗2在，我说的你听得到不3恩，听得到(接下来请

开始你的表演)

补充：SYN：请求询问，ACk：回复，回应。

四次挥手过程：

由于TCP连接是可以双向通信的（全双工），因此每个方向都必须单独进行关闭（这句话才是

精辟，后面四个挥手过程都是其具体实现的语言描述）

四次挥手过程，客户端和服务端都可以先开始断开连接

1客户端发送带有fin标识的报文给服务端，请求通信关闭

2服务端收到信息后，回复ACK答应关闭客户端通信(连接)请求

3服务端发送带有fin标识的报文给客户端，也请求关闭通信

4客户端回应ack给服务端，答应关闭服务端的通信(连接)请求

**5、什么是arp协议?**

# 答案:：ARP协议，全称“Address Resolution Protocol”,中文名是地址解析协议，使用ARP协议可实现通过IP地址获得对应主机的物理地址（MAC地址）。

**TCP和UDP的区别?为何基于tcp协议的通信比基于udp协议的通信更可靠?**

# 答案:

# TCP和UDP的区别?

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的，UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP首部开销20字节;UDP的首部开销小，只有8个字节

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

# 为何基于tcp协议的通信比基于udp协议的通信更可靠？

tcp:可靠 对方给了确认收到信息，才发下一个，如果没收到确认信息就重发

udp:不可靠 一直发数据，不需要对方回应

**8.什么是socket?简述基于tcp协议的套接字通信流程。**

# 答案:

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，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。

服务端：创建socket对象，绑定ip端口bind(), 设置最大链接数listen(), accept()与客户端的connect()创建双向管道， send(), recv(),close()；

客户端：创建socket对象，connect()与服务端accept()创建双向管道, send(),recv(),close()；

**什么是粘包? socket中造成粘包的原因是什么? 哪些情况会发生粘包现象?**

答案：粘包： 数据粘在一起，主要因为：接收方不知道消息之间的界限，不知道一次性提取多少字节的数据造成的数据量比较小，时间间隔比较短，就合并成了一个包，这是底层的一个优化算法（Nagle算法）。

**select、poll、epoll（Nginx使用的）模型的区别?**

答案：I/O多路复用的本质就是用select/poll/epoll，去监听多个socket对象，如果其中的socket对象有变化，只要有变化，用户进程就知道了。

select是不断轮询去监听的socket，socket个数有限制，一般为1024个；

poll还是采用轮询方式监听，只不过没有个数限制；

epoll并不是采用轮询方式去监听了，而是当socket有变化时通过回调的方式主动告知用户进程。

**13.简述进程、线程、协程的区别以及应用场景?**

答案:

1.进程是操作系统资源分配的最小单位，拥有独立的资源和地址空间

2.线程是CPU调度的单位；

3.统一进程中的线程是资源共享的。

4.协程是用户级别的，程序之间的切换由用户自行处理，节省了CPU的调度时间。

**14.什么是GIL锁?**

答案：GIL：全局解释锁，每次只能一个线程获得cpu的使用权：为了线程安全，也就是为了解决多线程之间的数据完整性和状态同步而加的锁，因为我们知道线程之间的数据是共享的。每个线程在执行的过程都需要先获取GIL，保证同一时刻只有一个线程可以执行字节码。

线程释放GIL锁的情况：

在IO操作等可能会引起阻塞的system call之前,可以暂时释放GIL,但在执行完毕后,必须重新获取GIL；

Python 3.x使用计时器（执行时间达到阈值后，当前线程释放GIL）或Python 2.x，tickets计数达到100；

Python使用多进程是可以利用多核的CPU资源的。

多线程爬取比单线程性能有提升，因为遇到IO阻塞会自动释放GIL锁

**Python中如何使用线程池和进程池?**

from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor

with ThreadPoolExecutor(4) as pool:

pool.submit(func,)

**进程之间如何进行通信?**

答案:

队列

python提供了多种进程通信的方式，主要Queue和Pipe这两种方式，Queue用于多个进程间实现通信，Pipe是两个进程的通信。

# Queue

from multiprocessing import Process, Queue

import os,time,random

#写数据进程执行的代码

def proc\_write(q,urls):

print 'Process is write....'

for url in urls:

q.put(url)

print 'put %s to queue... ' %url

time.sleep(random.random())

#读数据进程的代码

def proc\_read(q):

print('Process is reading...')

while True:

url = q.get(True)

print('Get %s from queue' %url)

if \_\_name\_\_ == '\_\_main\_\_':

#父进程创建Queue，并传给各个子进程

q = Queue()

proc\_write1 = Process(target=proc\_write,args=(q,['url\_1','url\_2','url\_3']))

proc\_write2 = Process(target=proc\_write,args=(q,['url\_4','url\_5','url\_6']))

proc\_reader = Process(target=proc\_read,args=(q,))

#启动子进程，写入

proc\_write1.start()

proc\_write2.start()

proc\_reader.start()

#等待proc\_write1结束

proc\_write1.join()

proc\_write2.join()

#proc\_raader进程是死循环，强制结束

proc\_reader.terminate()

# PIPE

import multiprocessing

import os,time,random

#写数据进程执行的代码

def proc\_send(pipe,urls):

#print 'Process is write....'

for url in urls:

print 'Process is send :%s' %url

pipe.send(url)

time.sleep(random.random())

#读数据进程的代码

def proc\_recv(pipe):

while True:

print('Process rev:%s' %pipe.recv())

time.sleep(random.random())

if \_\_name\_\_ == '\_\_main\_\_':

#父进程创建pipe，并传给各个子进程

pipe = multiprocessing.Pipe()

p1 = multiprocessing.Process(target=proc\_send,args=(pipe[0],['url\_'+str(i) for i in range(10) ]))

p2 = multiprocessing.Process(target=proc\_recv,args=(pipe[1],))

#启动子进程，写入

p1.start()

p2.start()

p1.join()

p2.terminate()

**41.简述线程死锁是如何造成的?如何避免?**

答案:

1.加锁顺序（线程按照一定的顺序加锁）

2.加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）

3.死锁检测

**什么是负载均衡?**

答案：系统的扩展可分为纵向（垂直）扩展和横向（水平）扩展。纵向扩展，是从单机的角度通过增加硬件处理能力，比如CPU处理能力，内存容量，磁盘等方面，实现服务器处理能力的提升，不能满足大型分布式系统（网站），大流量，高并发，海量数据的问题。因此需要采用横向扩展的方式，通过添加机器来满足大型网站服务的处理能力。比如：一台机器不能满足，则增加两台或者多台机器，共同承担访问压力。这就是典型的集群和负载均衡架构

**50.什么是rpc及应用场景?**

答案：RPC主要用于公司内部的服务调用，性能消耗低，传输效率高，服务治理方便。HTTP主要用于对外的异构环境，浏览器接口调用，APP接口调用，第三方接口调用等...

**谈谈你对多进程，多线程，以及协程的理解，项目是否用？**

这个问题被问的概率相当之大，其实多线程，多进程，在实际开发中用到的很少，除非是那些对项目性能要求特别高的，有的开发工作几年了，也确实没用过，你可以这么回答，给他扯扯什么是进程，线程（cpython中是伪多线程）的概念就行，实在不行你就说你之前写过下载文件时，用过多线程技术，或者业余时间用过多线程写爬虫，提升效率。

进程：一个运行的程序（代码）就是一个进程，没有运行的代码叫程序，进程是系统资源分配的最小单位，进程拥有自己独立的内存空间，所以进程间数据不共享，开销大。

线程：调度执行的最小单位，也叫执行路径，不能独立存在，依赖进程存在一个进程至少有一个线程，叫主线程，而多个线程共享内存(数据共享，共享全局变量)，从而极大地提高了程序的运行效率。

协程：是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

**59.解释一下什么是锁，有哪几种锁?**

锁(Lock)是 Python 提供的对线程控制的对象。有互斥锁、递归锁。

**60.什么是死锁呢？**

若干子线程在系统资源竞争时，都在等待对方对某部分资源解除占用状态，结果是谁也不愿先解锁，互相干等着，程序无法执行下去，这就是死锁。

GIL锁（有时候，面试官不问，你自己要主动说，增加b格，尽量别一问一答的尬聊，不然最后等到的一句话就是：你还有什么想问的么？）；

GIL锁 全局解释器锁（只在cpython里才有）；

作用：限制多线程同时执行，保证同一时间只有一个线程执行，所以cpython里的多线程其实是伪多线程!；

所以Python里常常使用协程技术来代替多线程，协程是一种更轻量级的线程，进程和线程的切换时由系统决定，而协程由我们程序员自己决定，而模块gevent下切换是遇到了耗时操作才会切换。

三者的关系：进程里有线程，线程里有协程。

**什么是线程安全，什么是互斥锁？**

每个对象都对应于一个可称为"互斥锁"的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。

同一个进程中的多线程之间是共享系统资源的，多个线程同时对一个对象进行操作，一个线程操作尚未结束，另一个线程已经对其进行操作，导致最终结果出现错误，此时需要对被操作对象添加互斥锁，保证每个线程对该对象的操作都得到正确的结果。

**什么是僵尸进程和孤儿进程？怎么避免僵尸进程?**

孤儿进程：父进程退出，子进程还在运行的这些子进程都是孤儿进程，孤儿进程将被 init 进程(进程号为 1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：进程使用fork创建子进程，如果子进程退出，而父进程并没有调用 wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中的这些进程是僵尸进程。

避免僵尸进程的方法：

-fork两次用孙子进程去完成子进程的任务；

-用wait()函数使父进程阻塞；

-使用信号量，在 signal handler 中调用waitpid，这样父进程不用阻塞。

**TCP服务器端的创建流程**

1.创建TCP服务端的socket 2.bing绑定ip地址和端口号 3.listen使套接字变为被动套接字 4.accept取出一个客户端连接，用于服务 5.recv/send接收和发送消息 6.关闭套接字

import socket

def main():

# 1、创建tcp服务端的socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 2、绑定

server\_socket.bind(('', 8888))

# 3、listen使套接字变为被动套接字

server\_socket.listen(128)

# 4、如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端服务

# client\_socket用来为这个客户端服务

# tcp\_server\_socket就可以省下来专门等待其他新客户端的链接

client\_socket, client\_addr = server\_socket.accept()

# 5、接收客户端发来的消息

recv\_data = client\_socket.recv(1024)

print("接收到客户端%s的数据：%s" % (str(client\_addr), recv\_data.decode('gbk')))

# 6、回复数据给客户端

client\_socket.send("收到消息".encode('gbk'))

# 7、关闭套接字

client\_socket.close()

server\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**描述用浏览器访问<www.baidu.com>的过程**

先要解析出baidu.com对应的ip地址

- 要先使用arp获取默认网关的mac地址

- 组织数据发送给默认网关(ip还是dns服务器的ip，但是mac地址是默认网关的mac地址)

- 默认网关拥有转发数据的能力，把数据转发给路由器

- 路由器根据自己的路由协议，来选择一个合适的较快的路径转发数据给目的网关

- 目的网关(dns服务器所在的网关)，把数据转发给dns服务器

- dns服务器查询解析出baidu.com对应的ip地址，并原路返回请求这个域名的client

得到了baidu.com对应的ip地址之后，会发送tcp的3次握手，进行连接

使用http协议发送请求数据给web服务器

- web服务器收到数据请求之后，通过查询自己的服务器得到相应的结果，原路返回给浏览器。

- 浏览器接收到数据之后通过浏览器自己的渲染功能来显示这个网页。

- 浏览器关闭tcp连接，即4次挥手结束，完成整个访问过程。

**Post和Get请求的区别?**

GET请求：

请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格式采用的是ASCII编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。

POST请求：

POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就是实际的传输数据。因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。

传输数据的大小：

- 在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。

- 对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST。

-提交数据大小进行限制，Apache、IIS都有各自的配置。

安全性：

- POST的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击。

效率：GET比POST效率高。

POST请求的过程：

1.浏览器请求tcp连接（第一次握手）

2.服务器答应进行tcp连接（第二次握手）

3.浏览器确认，并发送post请求头（第三次握手，这个报文比较小，所以http会在此时进行

第一次数据发送）

4.服务器返回100 continue响应

5.浏览器开始发送数据

6.服务器返回200 ok响应

GET请求的过程：

1.浏览器请求tcp连接（第一次握手）

2.服务器答应进行tcp连接（第二次握手）

3.浏览器确认，并发送get请求头和数据（第三次握手，这个报文比较小，所以http会在此时

进行第一次数据发送）

4.服务器返回200 OK响应

**75.cookie 和session 的区别？**

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能考虑到减轻服务器性能方面，应当使用cookie。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、建议： 将登陆信息等重要信息存放为SESSION 其他信息如果需要保留，可以放在cookie中

首先需要搞清楚的是session是存储在服务器上的，cookie是存储在客户端浏览器上的两者是相辅相成的用户首次访问服务器，服务器会为每个用户单独创建一个session对象(HttpSession)，并为每个session分配唯一一个id(sessionId)，sessionId通过cookie保存到用户端，当用户再次访问服务器时，需将对应的sessionId携带给服务器，服务器通过这个唯一sessionId就可以找到用户对应的session对象，从而达到管理用户状态

**76、HTTP协议状态码有什么用，列出你知道的 HTTP 协议的状态码，然后讲出他们都 表示什么意思？**

通过状态码告诉客户端服务器的执行状态，以判断下一步该执行什么操作。

常见的状态机器码有：

100-199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。

200-299：表示服务器成功接收请求并已完成处理过程，常用200（OK请求成功）。

300-399：为完成请求，客户需要进一步细化请求。302（所有请求页面已经临时转移到新的url）。304、307（使用缓存资源）。

400-499：客户端请求有错误，常用404（服务器无法找到被请求页面），403（服务器拒绝访问，权限不够）。

500-599：服务器端出现错误，常用500（请求未完成，服务器遇到不可预知的情况）。

**77、说说HTTP和HTTPS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。

在 OSI 网络模型中，HTTPS的加密是在传输层完成的,因为SSL是位于传输层的,TLS的前身是SSL,所以同理HTTP无需认证证书,而https需要认证证书。

**80.谈一下HTTP协议以及协议头部中表示数据类型的字段？**

HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web）服务器传输超文本到本地浏览器的传送协议。

HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件， 图片文件， 查询结果等）。

HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版，HTTP/1.1 的规范化工作正在进行之中，而且 HTTP-NG(Next Generation of HTTP)的建议已经提出。

HTTP 协议工作于客户端-服务端架构为上。浏览器作为 HTTP 客户端通过URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

表示数据类型字段： Content-Type

**81、HTTP常见请求头？**

1. Host (主机和端口号)

2. Connection (链接类型)

3. Upgrade-Insecure-Requests (升级为 HTTPS 请求)

4. User-Agent (浏览器名称)

5. Accept (传输文件类型)

6. Referer (页面跳转处)

7. Accept-Encoding（文件编解码格式）

8. Cookie （Cookie）

9. x-requested-with :XMLHttpRequest (是 Ajax 异步请求)

**模块相关**

**re的 match 和 search 区别?**

match()函数只检测字符串开头位置是否匹配，匹配成功才会返回结果，否则返回None

import re

s1 = 'abcabcabc'

print(re.match('abc', s1))

print(re.match('abc', s1).group())

print(re.match('abc', s1).span())

search()函数会在整个字符串内查找模式匹配,只到找到第一个匹配然后返回一个包含匹配信息的对象,该对象可以通过调用group()方法得到匹配的字符串,如果字符串没有匹配，则返回None。

import re

s1 = 'abcabcabc'

print(re.search('bca', s1))

print(re.search('bca', s1).group())

print(re.search('bca', s1).span())

**4.正则的贪婪匹配与非贪婪**

贪婪模式：能匹配的最大部分：“.+”中的+会一直查找到最后一个符合条件的字符；

非贪婪模式：匹配越少越好：“.+?"会匹配到第一个符合条件的字符就停止；

**json 序列化时，默认遇到中文会转换成 unicode，如果想要保留中文怎么办?**

import json

a=json.dumps({"ddf":"你好"},ensure\_ascii=False)

print(a) #{"ddf": "你好"}

**单例模式**

**1请手写一个单例**

# 至少会面向对象和装饰器

#encoding=utf8

import threading

import time

#这里使用方法\_\_new\_\_来实现单例模式

class Singleton(object):#抽象单例

def \_\_new\_\_(cls, \*args, \*\*kw):

if not hasattr(cls, '\_instance'):

orig = super(Singleton, cls)

cls.\_instance = orig.\_\_new\_\_(cls, \*args, \*\*kw)

return cls.\_instance

#总线

class Bus(Singleton):

lock = threading.RLock()

def sendData(self,data):

self.lock.acquire()

time.sleep(3)

print "Sending Signal Data...",data

self.lock.release()

#线程对象，为更加说明单例的含义，这里将Bus对象实例化写在了run里

class VisitEntity(threading.Thread):

my\_bus=""

name=""

def getName(self):

return self.name

def setName(self, name):

self.name=name

def run(self):

self.my\_bus=Bus()

self.my\_bus.sendData(self.name)

if \_\_name\_\_=="\_\_main\_\_":

for i in range(3):

print "Entity %d begin to run..."%i

my\_entity=VisitEntity()

my\_entity.setName("Entity\_"+str(i))

my\_entity.start()

**2单例模式的优点和应用？**

单例模式的优点：

1、由于单例模式要求在全局内只有一个实例，因而可以节省比较多的内存空间；

2、全局只有一个接入点，可以更好地进行数据同步控制，避免多重占用；

3、单例可长驻内存，减少系统开销。

单例模式的应用举例：

1、生成全局惟一的序列号；

2、访问全局复用的惟一资源，如磁盘、总线等；

3、单个对象占用的资源过多，如数据库等；

4、系统全局统一管理，如Windows下的Task Manager；

5、网站计数器。

6、数据库配置，数据库连接池

7、应用程序的日志应用

**3单例模式的缺点**

1、单例模式的扩展是比较困难的；

2、赋于了单例以太多的职责，某种程度上违反单一职责原则（六大原则后面会讲到）;

3、单例模式是并发协作软件模块中需要最先完成的，因而其不利于测试；

4、单例模式在某种情况下会导致“资源瓶颈”。

**前端**

**简述jsonp及其原理?**

为了保证用户访问的安全，现代浏览器使用了同源策略，即不允许访问非同源的页面，详细的概念大家可以自行百度。这里大家只要知道，在ajax中，不允许请求非同源的URL就可以了，比如www.a.com下的一个页面，其中的ajax请求是不允许访问www.b.com/c.php这样一个页面的。

JSONP就是用来解决跨域请求问题的

JSONP原理：

ajax请求受同源策略影响，不允许进行跨域请求，而script标签src属性中的链接却可以访问跨域的js脚本，利用这个特性，服务端不再返回JSON格式的数据，而是返回一段调用某个函数的js代码，在src中进行了调用，这样实现了跨域。

**20.简述cors及其原理?**

CORS：全称"跨域资源共享"（Cross-origin resource sharing）

CORS需要浏览器和服务器同时支持，才可以实现跨域请求，目前几乎所有浏览器都支持CORS，IE则不能低于IE10。CORS的整个过程都由浏览器自动完成，前端无需做任何设置，跟平时发送ajax请求并无差异。so，实现CORS的关键在于服务器，只要服务器实现CORS接口，就可以实现跨域通信。

**Django中cors的配置**

安装

pip install django-cors-headers

添加应用

在settings里面配置

INSTALLED\_APPS = (

...

'corsheaders',

...

)

中间层设置

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

...

]

添加白名单

# CORS

CORS\_ORIGIN\_WHITELIST = (

'127.0.0.1:8080',

'localhost:8080', #凡是出现在白名单中的域名，都可以访问后端接口

)

CORS\_ALLOW\_CREDENTIALS = True # 指明在跨域访问中，后端是否支持对cookie的操作

**Django相关**

**1. Django ORM查询中select\_related和prefetch\_related的区别？？**

def select\_related(self, \*fields)

性能相关：表之间进行join连表操作，一次性获取关联的数据。

总结：

1. select\_related主要针一对一和多对一关系进行优化。

2. select\_related使用SQL的JOIN语句进行优化，通过减少SQL查询的次数来进行优化、提高性能。

def prefetch\_related(self, \*lookups)

性能相关：多表连表操作时速度会慢，使用其执行多次SQL查询在Python代码中实现连表操作。

总结：

1. 对于多对多字段（ManyToManyField）和一对多字段，可以使用prefetch\_related()来进行优化。

2. prefetch\_related()的优化方式是分别查询每个表，然后用Python处理他们之间的关系。

**django中间件的使用？**

面试官问你Django中间件的时候，我们不应该只是局限于面试官的问题，而应做到举一反三。 面试之前准备一些白纸，在问到一些问题的时候应该用画图的形式展示出来 比如这里问到Django的中间件，我们应该给面试官画出Django的生命周期整体流程图，把中间件作为一部分的回答内容， 这样的好处在于，即展示了你对Django从前到后的流程都很熟悉又回答了面试官的问题，还顺带秀了一把其他技能，一举两得。 中间件介绍:作为Django的门户，一切请求都会先经过中间件才会到达Django后端，所以中间件可以用来做全局方面的一些功能 详细:给我们定义了五个方法

1.def process\_request(request):

pass

2.def process\_view(request):

pass

3.def process\_template\_response(request):

pass

4.def process\_exception(request):

pass

5.def process\_response(request):

pass

这些内容应该做到快速回答，不要"慢条斯理"的，搞IT的都很忙好吧，知识点一定要掌握牢固，脱口而出 一定要记住你是需要在有限的时间内将自己的看家本领不遗余力的倾囊而出

**django开发中数据库做过什么优化?**

1.设计表时，尽量少使用外键，因为外键约束会影响插入和删除性能；

2.使用缓存，减少对数据库的访问；

3.在orm框架下设置表时，能用varchar确定字段长度时，就别用text；

4.可以给搜索频率高的字段属性，在定义时创建索引；

5.Django orm框架下的Querysets 本来就有缓存的；

6.如果一个页面需要多次连接数据库，最好一次性取出所有需要的数据，减少对数据库的查询次数；

7.若页面只需要数据库里某一个两个字段时，可以用QuerySet.values()；

8.在模板标签里使用with标签可以缓存Qset的查询结果。

**django如何提升性能（高并发）？**

对一个后端开发程序员来说，提升性能指标主要有两个一个是并发数，另一个是响应时间。

网站性能的优化一般包括web前端性能优化，应用服务器性能优化，存储服务器优化。

对前端的优化主要有： 1.减少http请求，减少数据库的访问量，比如使用雪碧图。 2.使用浏览器缓存，将一些常用的css，js，logo图标，这些静态资源缓存到本地浏览器，通过设置http头中的cache-control和expires的属性，可设定浏览器缓存，缓存时间可以自定义。 3.对html，css，javascript文件进行压缩，减少网络的通信量。

对我个人而言，我做的优化主要是以下三个方面： 1.合理的使用缓存技术，对一些常用到的动态数据，比如首页做一个缓存，或者某些常用的数据做 个缓存，设置一定得过期时间，这样减少了对数据库的压力，提升网站性能。 2.使用celery消息队列，将耗时的操作扔到队列里，让worker去监听队列里的任务，实现异步操作，比如发邮件，发短信。 3.就是代码上的一些优化，补充：nginx部署项目也是项目优化，可以配置合适的配置参数，提升效率，增加并发量。 4.如果太多考虑安全因素，服务器磁盘用固态硬盘读写，远远大于机械硬盘，这个技术现在没有普及，主要是固态硬盘技术上还不是完全成熟， 相信以后会大量普及。 5.另外还可以搭建服务器集群，将并发访问请求，分散到多台服务器上处理。 6.最后就是运维工作人员的一些性能优化技术了。

**uWSGI与uwsgi区别**

uWSGI是一个 Web 服务器，它实现了WSGI 协议、uwsgi、http 等协议。注意 uwsgi 是一种通信协议，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 管理等优点，并且搭配着 Nginx就是一个生产环境了，能够将用户访问请求与应用 app 隔离开，实现真正的部署。相比来讲，支持的并发量更高，方便管理多进程，发挥多核的优势，提升性能。

**简述 websocket 协议及实现原理。**

websocket是一种在单个TCP连接上进行全双工通讯的协议，双工（duplex）是指两台通讯设备之间，允许有双向的资料传输。全双工的是指，允许两台设备间同时进行双向资料传输。这是相对于半双工来说的，半双工不能同时进行双向传输，这期间的区别相当于手机和对讲机的区别，手机在讲话的同时也能听到对方说话，对讲机只能一个说完另一个才能说。

**django 中如何实现 websocket?**

**Python web 开发中, 跨域问题的解决思路是?**

# 方案1.安装django-cors-headers

pip install django-cors-header

配置settings.py文件

INSTALLED\_APPS = [

...

'corsheaders'，

...

]

MIDDLEWARE\_CLASSES = (

...

'corsheaders.middleware.CorsMiddleware',

'django.middleware.common.CommonMiddleware', # 注意顺序

...

)

#跨域增加忽略

CORS\_ALLOW\_CREDENTIALS = True

CORS\_ORIGIN\_ALLOW\_ALL = True

CORS\_ORIGIN\_WHITELIST = (

'\*'

)

CORS\_ALLOW\_METHODS = (

'DELETE',

'GET',

'OPTIONS',

'PATCH',

'POST',

'PUT',

'VIEW',

)

CORS\_ALLOW\_HEADERS = (

'XMLHttpRequest',

'X\_FILENAME',

'accept-encoding',

'authorization',

'content-type',

'dnt',

'origin',

'user-agent',

'x-csrftoken',

'x-requested-with',

'Pragma',

)

# 方案2.使用JSONP

使用Ajax获取json数据时，存在跨域的限制。不过，在Web页面上调用js的script脚本文件时却不受跨域的影响，JSONP就是利用这个来实现跨域的传输。因此，我们需要将Ajax调用中的dataType从JSON改为JSONP（相应的API也需要支持JSONP）格式。

JSONP只能用于GET请求。

# 方案3.直接修改Django中的views.py文件

修改views.py中对应API的实现函数，允许其他域通过Ajax请求数据：

def myview(\_request):

response = HttpResponse(json.dumps({“key”: “value”, “key2”: “value”}))

response[“Access-Control-Allow-Origin”] = “\*”

response[“Access-Control-Allow-Methods”] = “POST, GET, OPTIONS”

response[“Access-Control-Max-Age”] = “1000”

response[“Access-Control-Allow-Headers”] = “\*”

return response

**django 请求的生命周期?**

. 当用户在浏览器中输入url时,浏览器会生成请求头和请求体发给服务端

请求头和请求体中会包含浏览器的动作(action),这个动作通常为get或者post,体现在url之中.

. url经过Django中的wsgi,再经过Django的中间件,最后url到过路由映射表,在路由中一条一条进行匹配,

一旦其中一条匹配成功就执行对应的视图函数,后面的路由就不再继续匹配了.

. 视图函数根据客户端的请求查询相应的数据.返回给Django,然后Django把客户端想要的数据做为一个字符串返回给客户端.

. 客户端浏览器接收到返回的数据,经过渲染后显示给用户.

**54.django 中如何在 model 保存前做一定的固定操作,比如写一句日志?**

利用Django的Model的Signal Dispatcher,通过django.db.models.signals.pre\_save()方法，在事件发生前，发射触发信号，这一切都被调度中的receiver方法深藏功与名的保存了。 信号的处理一般都写在Model中，举个例子：

import logging

from django.db import models

from django.db.models.signals import pre\_save

from django.dispatch import receiver

class Order(models.Model):

logger = logging.getLogger(\_\_name\_\_)

@receiver(pre\_save, sender=Order)

def pre\_save\_handler(sender, \*\*kwargs):

logger.debug("{},{}".format(sender, \*\*kwargs))

**55、django的request对象是在什么时候创建的？**

`class WSGIHandler(base.BaseHandler):`

`-------request = self.request\_class(environ)`

请求走到WSGIHandler类的时候，执行\*\*cell\*\*方法，将environ封装成了request

**django orm 中你了解的所有方法?**

# QuerySet对象的所有方法

<1> all(): 查询所有结果

<2> filter(\*\*kwargs): 它包含了与所给筛选条件相匹配的对象。获取不到返回None

<3> get(\*\*kwargs): 返回与所给筛选条件相匹配的对象，返回结果有且只有一个。

如果符合筛选条件的对象超过一个或者没有都会抛出错误。

<4> exclude(\*\*kwargs): 它包含了与所给筛选条件不匹配的对象

<5> order\_by(\*field): 对查询结果排序

<6> reverse(): 对查询结果反向排序

<8> count(): 返回数据库中匹配查询(QuerySet)的对象数量。

<9> first(): 返回第一条记录

<10> last(): 返回最后一条记录

<11> exists(): 如果QuerySet包含数据，就返回True，否则返回False

<12> values(\*field): 返回一个ValueQuerySet——一个特殊的QuerySet，运行后得到的

并不是一系 model的实例化对象，而是一个可迭代的字典序列

<13> values\_list(\*field): 它与values()非常相似，它返回的是一个元组序列，values返回的是一个字典序列

<14> distinct(): 从返回结果中剔除重复纪录

**django 中如何根据数据库表生成 model 类?**

创建一个项目，修改setting文件，在setting里面设置你要连接的数据库类型和连接名称，地址之类，和创建新项目的时候一致

1.运行下面代码可以自动生成models模型文件 python manage.py inspectdb 2.把模型文件导入到app中

创建app

django-admin.py startapp app

将模型导入创建的app中去

python manage.py inspectdb > app/models.py

34,38,39，42，53,54,55信号

**支付安全？**

* 转账的话使用事务（讲一下事务的四个特性）

**解决代码冲突的整个流程？**

* 1、远程路径存在或本地新建一个分支；

2、分支库与主库代码一致；

3、本地从分支库上拉取新代码；

4、本地代码已经修改，本地与新代码之间的冲突；（git diff）

3、整合本地代码新代码；

4、整合后，提交融合后的代码到个人私库上；

5、发送合入请求。

**接口文档怎么写？**

* 接口名称
* 接口描述
* 输入参数
* 输出参数

**开发文档怎么写？**

**pep8规范？**

* xrange( ) / range( )
* ascii / utf8
* 列表 / 迭代器
* 空格无需括号也可打印 / 必须括号
* raw\_input( ) / input( )

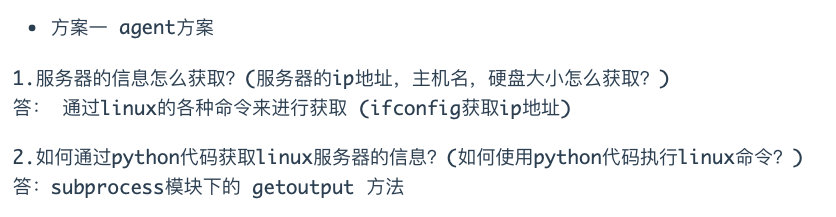
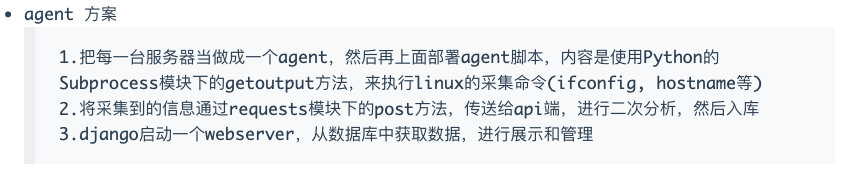
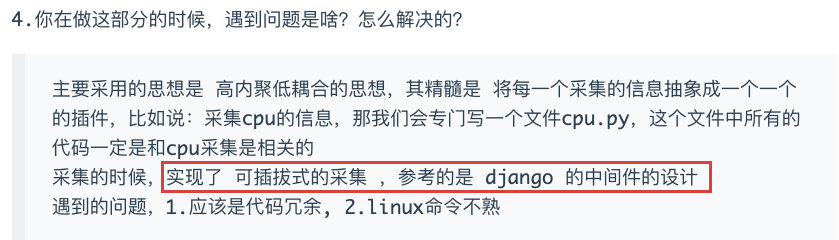
**restful规范？**

* 它的特点就是简洁嘛
* url的设计
* 请求方式
* 网络状态码
  + 100 请求
  + 200 成功
  + 300 重定向
  + 400 请求错误
  + 500 服务器错误
* 相应的结果
* 每一个url代表一种资源，根据HTTP请求方式对资源进行操作。
* get/post/put/delete
* 可以参考豆瓣的api设计规范

**字典的底层原理实现？**

* 字典是Python的一种可变、无序容器数据结构，它的元素以键值对的形式存在，键值唯一。
* Python字典的底层实现是哈希表，简单来说就是开辟一个内存地址，申请命名空间，最后指向内存空间。
* 字典是通过键值直接计算得到对应的地址空间，查找一步到位

**测试or运维工具/项目？**

* CMDB项目
* 
* 
* 
* 

**什么是包？**

* 包含多个模块的命名空间
* 就是一个模块，含有\_\_init\_\_.py，导包就是导\_\_init\_\_.py

**装饰器？**

* 不改变调用方法和源码，增加功能
* 应用：权限校验等场景

**迭代器？**

* next（）方法

**可迭代对象？**

* 含有\_iter\_方法

**迭代器对象？**

* 含有\_iter\_和\_next\_方法

**生成器？**

* yield关键字 可返回多个值

**魔法方法？**

* python的内置方法
* \_\_new\_\_和\_\_init\_\_ 前者创建类、实例对象/后者传入参数初始化对象

**多进程多线程？**

* 进程是系统资源的基本单位，线程是CPU调度的基本单位，且资源共享，还有一个协程是用户级别的，更快
* 线程里面有一个GIL锁和互斥锁要区分

**开启多线程以及多进程？**

* threading导入Thread
* 然后通过threading.Thread(target=, args=)
* multiprocessing导入Process
* 然后通过Process(target=, args=)

**互斥锁和GIL锁区别？**

* 互斥锁主要针对共享数据的修改权限
* GIL锁主要针对遇到IO操作换线程执行，可能换掉的线程还是会因为互斥锁无法修改数据（得到人却得不到心）

**python退出时，内存不会全部清空？**

* 当Python退出时，尤其是那些对其他对象具有循环引用的Python模块或者从全局名称空间引用的对象并没有被解除分配或释放。
* 退出时，由于拥有自己的高效清理机制（分代回收和标记清除），Python会尝试取消分配/销毁其他所有对象。

**Django的form和modelform？**

* form主要做数据校验，网页的标签之类的
* modelform将model和form结合，实现对数据的操作

**数据库优化？**

* 优化索引，sql语句，分析慢查询（时间太长）
  + （优）创建唯一索引保证数据的唯一性
  + （优）加速表与表之间的连接
  + （缺）创建耗时，索引占空间
  + 你的查询语句必需用到索引.每个条件都要有索引.百万数据还不算很多.索引就可以解决的.但是还回的结果集大的话,几万条,这个要视I/O和网络情况.这个问题就有点大了.要解决的问题还是比较多的,比如服务器,客户端,网络.

追问

那请问，1000万条、5000万条、1亿条这些级别跟100万条有差别么？这些的优化又该怎么做？

追答

嗯,为什么要优化,原因就是用户的体验感,用户的承受力没办法满足了.

所以,初期的表结构非常重要.

建表的时候,很多人往往只是把需要的字段,字段属性,唯一性约束,主键这几个方面着手.不会去考虑表将来的发展.有些表,系统一直用,都不会有多大的增量,但有些表,增量是与日俱增的.

所以,建表的时候,要考虑好索引,如果索引都不够用,那就要考虑分区存储(分块存储--part),分块一般是以二的次方来分的,也就是1 2 4 8 16 .....

百万条的话,不需要分块.那张表的存储空间应该不会上G..

上G的话,一定要分块.

另外还有一个:那个数据为什么那么多,不能优化吗?

我以前有做过一个项目,数据六千多万.我优化后,只有三十几万.

用了三天时间做这件事情.

查下里面有没有没用的数据,有没有冗余的数据,

然后再想想,这些数据能不能用另外一种表达方式来优化.

不要因为表结构的改变, 而引起的程序改动很大而不为.

为了用户的体验感强,系统性能健壮,一定要做(做软件是为了让客户更有效的,系统的管理某些事务,其中有一点是很重要的-->能否提高工作效率.提高生产力.)

* 设计表的时候严格按照数据库设计规范（三大范式）来设计数据库
* 使用缓存例如redis，常访问的数据放在里面。
* 还有mysql自带的缓存，默认开启但是需要自己设置set global query\_cache\_size
* 主从分离读写，采用主从复制把数据库的读操作和写操作分离开来
* 分库分表分机器，数据量特别大的时候，主要的原理是数据路由
* 表引擎要选好（innodb事务外键行锁和myIsam快）
* 不采用全文检索

**创建索引但是无法命中索引的8种情况？**

* 条件中有or；
* like查询以%开头；
* 列类型是字符串要带上引号；

**不适合创建索引？**

* 查询少的列
* 列下数据少

**数据库连表（join）？**

* left join 左连：从左表返回所有的行，如果右表中没有匹配，返回Null
* right join 右连：从右表返回所有的行，如果右表中没有匹配，返回Null
* inner join 内联：如果表中**有至少一个**匹配，则返回行
* full join 全联：**只要其中一个**表中存在匹配，则返回行

**union all 代替 union？**

* Union需要将集合合并后在进行唯一性过滤操作，这会涉及到排序，大量的cpu运算，加大资源消耗及延迟，当然，使用union all的前提条件是两个结果集没有重复数据。
* Union：对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；
* Union All：对两个结果集进行并集操作，包括重复行，不进行排序；

**mysql数据库主从配置？**

* 主库将变动的记录存入binlog日志，发送给从库；
* 从库有一个I/O线程将记录写入relay日志；
* 从库另一个Sql线程就负责读relay日志；
* 主库写，从库读；

mysql主从配置的流程大体如图：

1）master会将变动记录到二进制日志里面；

2）master有一个I/O线程将二进制日志发送到slave;

3) slave有一个I/O线程把master发送的二进制写入到relay日志里面；

4）slave有一个SQL线程，按照relay日志处理slave的数据；

**乐观锁悲观锁？**

* 乐观锁就是可以对一条记录覆盖操作（应用：库存更新，解决高并发问题）
* 悲观锁就是每次查询、修改数据操作之前就会上锁，整个数据处理过程中，将数据处于锁定状态（应用：防止库存冲突）

**常用的web后台开发框架？**

* Django/DRF/flask/tornado（它是非阻塞式服务器epoll，而且速度相当快）

**反射？**

* python中一切皆对象，通过字符串形式访问对象属性
* 四个内置函数：getattr/hasattr/delattr/setattr

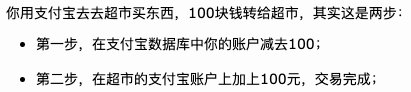
**Django实现celery异步任务？**

* 创建一个celery对象
* 设置Django的配置文件
* 对象配置
* 加载任务
* 启动celery
* 

**celery+redis出现的坑？**

* redis版本过高报错
* celery内存泄漏（设置每个worker执行的任务数）

**数据库事务？**



* 四大特性：ACID
* 原子性（要么全部执行，要么全部都不执行）
* 一致性（改变前后的状态要一致）
* 隔离性（事务之间互不影响）
* 持久性（事务提交后修改的数据永久保存）

**事务的四种隔离级别？**

* 数据库事务的隔离级别有4种，由低到高分别为Read uncommitted 、Read committed 、Repeatable read 、Serializable 。而且，在事务的并发操作中可能会出现脏读，不可重复读，幻读。

**=====下面通过事例一一阐述它们的概念与联系=====**

* 脏读（假数据，钱未到账只看到假数字）

不可重复读（钱在要用时突然被取走了）

重复读（钱在要用时不能被取走）

幻读（花了两千又有人同时花了一万）

序列化（串行化顺序执行）

**进程池线程池？**

* 进程池：一个进程内控制的进程数
* 模块导入进程池和线程池

from concurrent.futures import ProcessPoolExecutor,ThreadPoolExecutor

* 相应的 task 函数提交给线程池/进程池

submit(fn, \*args, \*\*kwargs)：将 fn 函数提交给线程池。

**跨站请求伪造？**

* 在forms组件里面加{% crsf\_token %}
* ajax的data里面加{{ csrf\_token }}或者配置请求头

**TCP和UDP区别？**

* TCP：面向字节流 ｜ UDP：面向报文
* TCP：建立连接 ｜ UDP：无连接
* TCP：数据无差错不丢失按序到达 ｜ UDP：不保证可靠交付
* TCP：每次连接点到点 ｜ UDP：支持一对一、一对多、多对一和多对多
* TCP：首部开销大 ｜ UDP：首部开销小
* TCP：全双工的可靠信道 ｜ UDP：不可靠信道

**粘包？**

* 数据粘在一起
* 原因：接收方不知道消息之间的界限，不知道一次性提取多少字节的数据造成的数据量比较小，时间间隔比较短，就合并成了一个包。

**什么是Websocket？**

* 基于TCP的一种新的协议
  + 连接时需要握手
  + 发送数据进行加密
  + 连接之后不断开
* 主动给客户端发消息

**简述同源策略？**

* 协议相同
* 端口相同
* 域名相同

**http/https区别？**

* 端口80/443
* 安全/不安全（证书）
* TSL/SSL

**反爬虫策略？**

* 添加代理
* 降低访问频率
* User-Agent
* 动态 HTML 数据加载
* 验证码处理
* Cookie

**数据分析Numpy？**

* np.arrange
* np.shape
* np.size

**docker命令？**

* docker ps / docker ps -a 查看容器
* docker pull 拉取/部署
* docker build -t='镜像名字' . 构建镜像
* docker images 查看镜像
* docker commit 镜像打包
* docker save 镜像备份
* docker load 镜像恢复
* docker tag 打标签
* docker push 上传
* docker run -id -p 8088:8088 -v /root/djangotest:/home --name=my\_django\_test python:3.6 目录挂载
* docker exec 进入容器
* docker rm 删除容器

**git常用命令？**

* git clone
* git init
* git diff
* git commit -m ""
* git add
* git remote
* git rm
* git status
* git log

**linux命令？**

* ls
* cp
* scp
* mv
* vim
* echo

**sql语句？**

* update 数据表 set字段名=字段值 where 条件表达式
* delete from 数据表 where 条件表达式
* select \* from 数据表 where字段名 = 字段值 order by 字段名[desc]
* insert into 数据表 (字段1,字段2,字段3 …) values (值1,值2,值3 …)

**中软国际面试内容**

**1、迭代器，list是迭代器吗（list是迭代器吗？）**

**迭代 生成**

**for循环遍历的原理**

for循环遍历的原理就是迭代，in后面必须是可迭代对象

**为什么要有迭代器**

对于序列类型：字符串、列表、元组，我们可以使用索引的方式迭代取出其包含的元素。但对于字典、集合、文件等类型是没有索引的，若还想取出其内部包含的元素，则必须找出一种不依赖于索引的迭代方式，这就是迭代器

**1.可迭代对象**

有\_\_iter\_\_方法的对象，都是可迭代对象，有以下6种

可迭代的对象：Python内置str、list、tuple、dict、set、file都是可迭代对象

"zx".\_\_iter\_\_()

["zx"].\_\_iter\_\_()

{"zx":"wl"}.\_\_iter\_\_()

("zx",).\_\_iter\_\_()

{"z","x"}.\_\_iter\_\_()

with open("prize.txt","r") as file:

file.\_\_iter\_\_()

**2.迭代器对象**

1.可迭代的对象执行\_\_iter\_\_方法得到的返回值是迭代器对象。2.迭代器对象指的是即内置有\_\_iter\_\_又内置有\_\_next\_\_方法的对象

list=[1,2,3,4,5,6]

zx=list.\_\_iter\_\_()

while True:

try:

print(zx.\_\_next\_\_())

except:

break

**文件类型是迭代器对象**

open('a.txt').\_\_iter\_\_()

open('a.txt').\_\_next\_\_()

**迭代器对象一定是可迭代对象，而可迭代对象不一定是迭代器对象**

**生成器**

**生成器就是一种迭代器。迭代器有一个特点就是可以被next()函数调用并不断返回下一个值的对象，**并且**只能迭代它们一次。**原因很简单,因为它们不是全部存在内存里,它们只在要调用的时候在内存里生成。

**生成器就是迭代器**（函数返回的结果就是生成器,而且它同时有\_\_iter\_\_和\_\_next\_\_）

def zx():

yield 1

yield 2

x=zx()

x.\_\_next\_\_()

x.\_\_iter\_\_()

**实验室**

每次\_\_next\_\_之后，会执行到相应的yield不会执行下面的内容。

为啥？这个实验执行了呢，因为用了for，他不知道最后执行到哪里，他就会一直\_\_next\_\_，直到最后抛出异常，然而for in里面自带捕获异常的内容，所有没有打印异常信息。

def func():

yield [1,1,23] # yield会使函数func()变成生成器对象，因此他就具有\_\_iter\_\_方法

print(789) # yield会停止函数，当运行下一次next才会继续运行下面的代码

yield 101112 # 一个yield对应一个next

print(131415)

g = func()

for i in g:

print(i)

[1, 1, 23]  
789  
101112  
131415

**关于迭代器和和生成器的区别**

生成器是一种特殊的迭代器，生成器实现了迭代器协议--iter--,--next--

生成器是可以改变迭代的值的，然而迭代器随意改值会有问题

**关于迭代器和list的区别**

list直接把所有数据加载到内存，而迭代器是一个一个取值，在需要下一个值的时候才回去计算取出这个值到内存（可以把迭代器想象成一个生成器的代码，一次next，运行下面的代码，然后返回值）。

**2、装饰器，传参**

[**python装饰器和函数传参**](https://www.cnblogs.com/mark-zh/p/11384546.html)

**装饰器[#](https://www.cnblogs.com/mark-zh/p/11384546.html" \l "3317386046)**

装饰器是一个返回函数的高阶函数。

装饰器常见用法：  
打印日志

Copy

def logger(func):

def wrapper(\*args, \*\*kw):

print 'do {}'.format(func.\_\_name\_\_)

func(\*args, \*\*kw)

print 'finish'

return wrapper

@logger

def add(x,y):

print '{} + {} = {}'.format(x,y,x+y)

add(3,5)

在函数执行前，打印一行日志do...；函数执行结束，打印一行日志finish。执行结果如下：

do add

3 + 5 = 8

finish

计算时间

import time

def timer(func):

def wrapper(\*args, \*\*kw):

t1 = time.time()

func(\*args,\*\*kw)

t2 = time.time()

cost\_time = t2 - t1

print 'cost time: {} s'.format(cost\_time)

return wrapper

@timer

def cost\_time(sleep\_time):

time.sleep(sleep\_time)

cost\_time(10)

带参数的函数装饰器

def say\_hello(country):

def wrapper(func):

def decorate(\*args,\*\*kw):

if country == 'en':

print 'hello'

elif country == 'usa':

print 'hi'

else:

return

func(\*args,\*\*kw)

return decorate

return wrapper

@say\_hello("usa")

def usa():

print 'i am from usa'

@say\_hello("en")

def en():

print 'i am from england'

usa()

print '----------------------'

en()

装饰器本身是一个函数，使用两层嵌套传参，执行结果如下：

hi

i am from usa

----------------------

hello

i am from england

不带参数的类装饰器  
基于类装饰器的实现，必须实现\_\_call\_\_和\_\_init\_\_两个内置函数。  
\_\_init\_\_ ：接收被装饰函数  
\_\_call\_\_ ：实现装饰逻辑

class logger(object):

def \_\_init\_\_(self,func):

self.func = func

def \_\_call\_\_(self,\*args,\*\*kwargs):

print 'the function {}() is running...'\

.format(self.func.\_\_name\_\_)

return self.func(\*args,\*\*kwargs)

@logger

def say(something):

print 'say {}!'.format(something)

say('hello')

运行结果如下：

the function say() is running...

say hello!

* 带参数的类装饰器  
  带参数和不带参数的类装饰器有很大的不同。  
  \_\_init\_\_ ：不再接收被装饰函数，而是接收传入参数  
  \_\_call\_\_ ：接收被装饰函数，实现装饰逻辑

class logger(object):

def \_\_init\_\_(self,level='INFO'):

self.level = level

def \_\_call\_\_(self,func):

def wrapper(\*args,\*\*kwargs):

print '{level}: the function {func} () is running...'\

.format(level=self.level, func=func.\_\_name\_\_)

func(\*args,\*\*kwargs)

return wrapper

@logger(level='WARNING')

def say(something):

print 'say {}!'.format(something)

say('hello')

运行结果如下：

WARNING: the function say () is running...

say hello!

**函数的参数[#](https://www.cnblogs.com/mark-zh/p/11384546.html" \l "2895466662)**

* 位置参数

def power(x, n):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

power(x, n)函数有两个参数：x和n，这两个参数都是位置参数，调用函数时，传入的两个值按照位置顺序依次赋值给参数x和n。

默认参数

def power(x, n=2):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

power(x, n)函数有两个参数：x和n，如果想在不传入n值时，默认计算x的平方，此时可以将n设为默认值2。

可变参数(\*args)

def function(f\_arg, \*args):

print f\_arg, type(f\_arg)

print args, type(args)

nums = ['a','b','c']

function(1,2,\*nums)

定义可变参数时，需要在参数前面加一个\*号，可变参数的个数是可变的。在函数内部，参数\*args接收到的是一个tuple。输出结果如下：

1 <type 'int'>

(2, 'a', 'b', 'c') <type 'tuple'>

关键字参数(\*\*kwargs)

def person(name,age,\*\*kwargs):

print 'name:',name,'age:',age,'other:',kwargs,type(kwargs)

person('mark',30,city='shanghai')

\*\*kwargs允许将不定长度的键值对，作为参数传递给一个函数，关键字参数在函数内部自动组装为一个dict。输出结果如下：

name: mark age: 30 other: {'city': 'shanghai'} <type 'dict'>

将函数作为参数传递给另一个函数

def hi():

return 'hi friends'

def function(func):

print 'just test'

print func()

function(hi)

function()函数将hi函数作为参数接收，输出结果如下：

Copy

just test

hi friends

**time模块[#](https://www.cnblogs.com/mark-zh/p/11384546.html" \l "776327601)**

* 获取当前时间

>>> time.localtime()

time.struct\_time(tm\_year=2019, tm\_mon=8, tm\_mday=21, tm\_hour=14, tm\_min=31, tm\_sec=18, tm\_wday=2, tm\_yday=233, tm\_isdst=0)

获取格式化的时间

>>> time.ctime()

'Wed Aug 21 14:51:28 2019'

>>> time.asctime()

'Wed Aug 21 14:51:34 2019'

格式化日期

>>> time.strftime('%Y-%m-%d %H:%M:%S',time.localtime())

'2019-08-21 14:35:02'

>>> time.strftime('%a %b %d %H:%M:%S %Y',time.localtime())

'Wed Aug 21 14:36:09 2019'

计算运行时间

import time

start = time.time()

time.sleep(2)

end = time.time()

print end-start

**3、多继承（C继承A，B，继承顺序），答得一般，没有答深度优先，广度优先以及mro**

先传进来哪个就找哪个，如果A上层还有，继续找上层  
经典类深度优先，新式类广度优先  
如果关系比较乱，使用mro()方法查看继承图

重写和重载：重写是覆盖父类的方法

                    重载可以根据传入参数的不同而调用对应的函数

多继承的三种写法：

1. 子类（孙类）0的方法中写父类.父类方法名（\*args,\*\*kargs）    这种写法如果子类很多，当创建一个对象，定义这个对象的类继承了多个子类时，父类方法将被调用多次
2. 子类（孙类）的方法中写super().父类方法名（\*args,\*\*kargs）  这种写法会根据C3算法的结果，按顺序从子类、父类的方法中依次调用，能够保证类方法都只调用一次。C3算法的结果可以用 子类.\_\_mro\_\_ 得到
3. 子类（孙类）的方法中写super(子类名,self).父类方法名（\*args,\*\*kargs）  这种写法会根据C3算法的结果，从传入子类名开始依次往后按顺序调用子类、父类的方法

总结：

1. super().\_\_init\_\_相对于类名.\_\_init\_\_，在单继承上用法基本无差
2. 但在多继承上有区别，super方法能保证每个父类的方法只会执行一次，而使用类名的方法会导致方法被执行多次
3. 多继承时，使用super方法，对父类的传参数，应该是由于python中super的算法导致的原因，必须把参数全部传递，否则会报错
4. 单继承时，使用super方法，则不能全部传递，只能传父类方法所需的参数，否则会报错
5. 多继承时，相对于使用父类.\_\_init\_\_方法，要把每个父类全部写一遍，而使用super方法，只需写一句话便执行了全部父类的方法，这也是为何多继承需要全部传参的一个原因

**4、哪些不会被继承，析构方法（没答上）**

魔法方法，如：\_\_del\_\_()

子类继承父类，那么默认的，就是继承了父类的[成员函数](http://www.so.com/s?q=%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&ie=utf-8&src=internal_wenda_recommend_textn" \t "_blank)和[成员变量](http://www.so.com/s?q=%E6%88%90%E5%91%98%E5%8F%98%E9%87%8F&ie=utf-8&src=internal_wenda_recommend_textn)。  
初始化子类时，会先自动调用父类的[构造函数](http://www.so.com/s?q=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&ie=utf-8&src=internal_wenda_recommend_textn" \t "_blank)。然后才调用子类的构造函数，析构时，按相反[顺序](http://www.so.com/s?q=%E9%A1%BA%E5%BA%8F&ie=utf-8&src=internal_wenda_recommend_textn)进行。  
构造[从类](http://www.so.com/s?q=%E4%BB%8E%E7%B1%BB&ie=utf-8&src=internal_wenda_recommend_textn" \t "_blank)[层次](http://www.so.com/s?q=%E5%B1%82%E6%AC%A1&ie=utf-8&src=internal_wenda_recommend_textn)的最根处开始，在每一层中，首先调用[基类](http://www.so.com/s?q=%E5%9F%BA%E7%B1%BB&ie=utf-8&src=internal_wenda_recommend_textn)的构造函数，然后调用成员[对象](http://www.so.com/s?q=%E5%AF%B9%E8%B1%A1&ie=utf-8&src=internal_wenda_recommend_textn)的构造函数。析构则严格按照与构造相反的[次序](http://www.so.com/s?q=%E6%AC%A1%E5%BA%8F&ie=utf-8&src=internal_wenda_recommend_textn)执行，该次序是唯一的，否则[编译器](http://www.so.com/s?q=%E7%BC%96%E8%AF%91%E5%99%A8&ie=utf-8&src=internal_wenda_recommend_textn)将无法[自动执行](http://www.so.com/s?q=%E8%87%AA%E5%8A%A8%E6%89%A7%E8%A1%8C&ie=utf-8&src=internal_wenda_recommend_textn)析构[过程](http://www.so.com/s?q=%E8%BF%87%E7%A8%8B&ie=utf-8&src=internal_wenda_recommend_textn" \t "_blank)。

**5、map()与reduce()（reduce()没答上）**

map() 会根据提供的函数对指定序列做映射。  
map(function, iterable, ...)  
第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。  
reduce函数对参数迭代器中的元素进行类累积  
格式为：reduce(func,iter,init)  
func为函数，iter为序列，init为固定初始值，无初始值时从序列的第一个参数开始

**Python中几个内置函数是高频使用，为了让读者深入了解这几个内置函数，不多哔哔看例子。**

**1，map()函数**

描述： 会根据提供的函数对指定序列做映射。第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

语法：map(function, iterable, ...)    其中 function 表示函数 ，iterable 表示一个或多个序列

**2，filter()函数**

描述：函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。

语法：filter(function, iterable)     其中 function 表示判断函数，iterable 表示可迭代对象。

来一个过滤出列表的所有奇数的例子

**3，reduce()函数**

描述：对参数序列中元素进行累积。函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

语法：reduce(function, iterable[, initializer]) 其中function 表示函数，有两个参数，iterable 表示可迭代对象，initializer表示可选，初始参数。

**6、多线程的使用场景，GIL原理（答得台模糊，没有分析）**

使用threading.Thread()方法  
继承threading.Thread类  
由于GIL的原因， Python解释器只允许同一时间执行一个线程。多线程不能用，使用多进程  
I/O 密集型使用多线程  
CPU 密集型使用多进程

TCP三次握手，四次挥手

第一次握手：建立连接时，客户端发送syn包（syn=x）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=y+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

1）客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。

2）服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。

3）客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。

4）服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。

5）客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2∗∗MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。

6）服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

 常见面试题

【问题1】为什么连接的时候是三次握手，关闭的时候却是四次握手？

答：因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

【问题2】为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。在Client发送出最后的ACK回复，但该ACK可能丢失。Server如果没有收到ACK，将不断重复发送FIN片段。所以Client不能立即关闭，它必须确认Server接收到了该ACK。Client会在发送出ACK之后进入到TIME\_WAIT状态。Client会设置一个计时器，等待2MSL的时间。如果在该时间内再次收到FIN，那么Client会重发ACK并再次等待2MSL。所谓的2MSL是两倍的MSL(Maximum Segment Lifetime)。MSL指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

【问题3】为什么不能用两次握手进行连接？

答：3次握手完成两个重要的功能，既要双方做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

       现在把三次握手改成仅需要两次握手，死锁是可能发生的。作为例子，考虑计算机S和C之间的通信，假定C给S发送一个连接请求分组，S收到了这个分组，并发 送了确认应答分组。按照两次握手的协定，S认为连接已经成功地建立了，可以开始发送数据分组。可是，C在S的应答分组在传输中被丢失的情况下，将不知道S 是否已准备好，不知道S建立什么样的序列号，C甚至怀疑S是否收到自己的连接请求分组。在这种情况下，C认为连接还未建立成功，将忽略S发来的任何数据分 组，只等待连接确认应答分组。而S在发出的分组超时后，重复发送同样的分组。这样就形成了死锁。

【问题4】如果已经建立了连接，但是客户端突然出现故障了怎么办？

TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

**7、CSRF预防**

如何防御?

1. 请求令牌(一种简单有效的防御方法)：

2. token验证

* 在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，如果请求中没有   
  token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求。
* token需要足够随机
* 敏感的操作应该使用POST，而不是GET，以form表单的形式提交，可以避免token泄露。

关于token:

3. 提交验证码   
在表单中增加一个随机的数字或字母验证码，通过强制用户和应用进行交互，来有效地遏制CSRF攻击。

4. 在请求地址中添加 token 并验证

5. 在 HTTP 头中自定义属性并验证

**8、MySQL隔离级别，哪些产生脏读，哪些产生幻读**

1. 脏读 ：脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。

2. 不可重复读 ：是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两 次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不 可重复读。例如，一个编辑人员两次读取同一文档，但在两次读取之间，作者重写了该文档。当编辑人员第二次读取文档时，文档已更改。原始读取不可重复。如果 只有在作者全部完成编写后编辑人员才可以读取文档，则可以避免该问题。

3. 幻读 : 是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。 同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象 发生了幻觉一样。例如，一个编辑人员更改作者提交的文档，但当生产部门将其更改内容合并到该文档的主复本时，发现作者已将未编辑的新材料添加到该文档中。 如果在编辑人员和生产部门完成对原始文档的处理之前，任何人都不能将新材料添加到文档中，则可以避免该问题。

补充 : 基于元数据的 Spring 声明性事务 :

Isolation 属性一共支持五种事务设置，具体介绍如下：

l          DEFAULT 使用数据库设置的隔离级别 ( 默认 ) ，由 DBA 默认的设置来决定隔离级别 .

l          READ\_UNCOMMITTED 会出现脏读、不可重复读、幻读 ( 隔离级别最低，并发性能高 )

l          READ\_COMMITTED  会出现不可重复读、幻读问题（锁定正在读取的行）

l          REPEATABLE\_READ 会出幻读（锁定所读取的所有行）

l          SERIALIZABLE 保证所有的情况不会发生（锁表）

不可重复读的重点是修改 :

同样的条件 ,   你读取过的数据 ,   再次读取出来发现值不一样了

幻读的重点在于新增或者删除

同样的条件 ,   第 1 次和第 2 次读出来的记录数不一样

**9、MySQL什么情况下创建索引**

较频繁地作为查询条件的字段  
出现where的字段

**10、Redis，缓存穿透与缓存雪崩，解决办法（解决办法答的不够多）**

缓存穿透：缓存与数据库中都没有，恶意访问，频繁的查数据库，压力大  
解决办法：查询一次，如果没有，缓存中增加空值  
  采用布隆过滤器（将所有可能存在的数据哈希到一个足够大的bitmap中）  
缓存雪崩：同一时间，大量键过期，同时访问数据库，压力大  
解决办法：设置过期时间为随机  
缓存击穿：前台高并发访问一个已经在缓存中恰好过期的数据，全部打到后台数据库  
解决办法：互斥锁，Redis的SETNX功能，数据没有，返回一个设定值，再去查数据库  
  热点数据设置永不过期

**11、Redis的内存清理机制**

**每台redis的服务器的内存都是有限的，而且也不是所有的内存都用来存储信息。而且redis的实现并没有在内存这块做太多的优化，所以实现者为了防止内存过于饱和，采取了一些措施来管控内存。**

**文章结构：（1）内存策略；（2）内存释放机制原理；（3）项目中如何合理应用淘汰策略；（4）单机版Redis内存优化注意点。**

**一、内存策略：**[**先来吃份官方文档**](https://redis.io/topics/lru-cache)

**最大内存的设置是通过设置maxmemory来完成的，格式为maxmemory bytes ,当目前使用的内存超过了设置的最大内存，就要进行内存释放了， 当需要进行内存释放的时候，需要用某种策略对保存的的对象进行删除。Redis有六种策略（默认的策略是volatile-lru。）**

**redis中当内存超过限制时，按照配置的策略，淘汰掉相应的key-value，使得内存可以继续留有足够的空间保存新的数据。redis 确定驱逐某个键值对后，会删除这个数据并，并将这个数据变更消息发布到本地（AOF 持久化）和从机（主从连接）。**

**（1）volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰。**

**（2）volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰**

**（3）volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰**

**（4）allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰**

**（5）allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰**

**（6）no-enviction：禁止淘汰数据**

**除此之外还有一个配置项，就是maxmemory-samples，默认值是3，因为上面的策略代码实现的都是近似算法，所以不管是lru算法，还是ttl，都并不是在数据库中所有的数据为基础的算法，因为当数据库的数据很多的时候，这样效率太低，所以代码中都是基于maxmemory-samples个数据的近似算法。详情请读下文。**

**置换策略是如何工作的：**

**1）客户端执行一条新命令，导致数据库需要增加数据（比如set key value）**

**2）Redis会检查内存使用，如果内存使用超过maxmemory，就会按照置换策略删除一些key**

**3）新的命令执行成功**

**注意：**

**如果我们持续的写数据会导致内存达到或超出上限maxmemory，但是置换策略会将内存使用降低到上限以下。**

**如果一次需要使用很多的内存（比如一次写入一个很大的set），那么，Redis的内存使用可能超出最大内存限制一段时间。**

**二、内存释放机制原理：**

**（1）概述：**

**当mem\_used内存已经超过maxmemory的设定，对于所有的读写请求，都会触发redis.c/freeMemoryIfNeeded(void)函数以清理超出的内存。注意这个清理过程是阻塞的，直到清理出足够的内存空间。所以如果在达到maxmemory并且调用方还在不断写入的情况下，可能会反复触发主动清理策略，导致请求会有一定的延迟。**

**清理时会根据用户配置的maxmemory-policy来做适当的清理（一般是LRU或TTL），这里的LRU或TTL策略并不是针对redis的所有key，而是以配置文件中的maxmemory-samples个key作为样本池进行抽样清理。**

**maxmemory-samples在redis-3.0.0中的默认配置为5，如果增加，会提高LRU或TTL的精准度，redis作者测试的结果是当这个配置为10时已经非常接近全量LRU的精准度了，并且增加maxmemory-samples会导致在主动清理时消耗更多的CPU时间，有如下建议：**

**1）尽量不要触发maxmemory，最好在mem\_used内存占用达到maxmemory的一定比例后，需要考虑调大hz以加快淘汰，或者进行集群扩容。**

**2）如果能够控制住内存，则可以不用修改maxmemory-samples配置；如果Redis本身就作为LRU cache服务（这种服务一般长时间处于maxmemory状态，由Redis自动做LRU淘汰），可以适当调大maxmemory-samples。**

**（2）内存管理源码解析：[参考博文](http://blog.dolphin-game.com/post/65.html" \t "_blank)**

**Redis释放内存是由函数freeMemoryIfNeeded完成的，redis用processCommand函数处理每条命令，函数中在真正处理命令之前都会调用freeMemoryIfNeeded函数，这个函数会判断当前使用的内存是否超过了最大使用内存，如果超过，就会根据内存释放策略释放内存。**

**freeMemoryIfNeeded函数首先会计算出当前使用了多少内存，注意，这里并不会包括slaves 输出缓存以及AOF缓存，源码如下：**

**12、Restful接口，如果是做计算，群增等设计，返回的状态码**

原文地址：https://www.jianshu.com/p/8b769356ee67

**一、重要概念：REST,即Representational State Transfer的缩写。我对这个词组的翻译是"表现层状态转化"。**

**Resource（资源）** ：对象的单个实例。 例如，一只动物。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。  
　　你可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。

**集合**：对象的集合。 例如，动物。

**第三方**：使用我们接口的开发者

**表现层（Representation）**

"资源"是一种信息实体，它可以有多种外在表现形式。我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。

**状态转化（State Transfer）**

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。互联网通信协议HTTP协议，是一个无状态协议。这意味着，  
所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，  
所以就是"表现层状态转化"。

客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。  
它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

URI只代表资源的实体，不代表它的形式。严格地说，有些网址最后的".html"后缀名是不必要的，因为这个后缀名表示格式，属于"表现层"范畴，  
而URI应该只代表"资源"的位置。它的具体表现形式，应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

综合上面的解释，我们总结一下什么是RESTful架构：

（1）每一个URI代表一种资源；

（2）客户端和服务器之间，传递这种资源的某种表现层；

（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

**二、REST接口规范**

**1、动作**

GET （SELECT）：从服务器检索特定资源，或资源列表。  
POST （CREATE）：在服务器上创建一个新的资源。  
PUT （UPDATE）：更新服务器上的资源，提供整个资源。  
PATCH （UPDATE）：更新服务器上的资源，仅提供更改的属性。  
DELETE （DELETE）：从服务器删除资源。

首先是四个半种动作：  
post、delete、put/patch、get  
因为put/patch只能算作一类，所以将patch归为半个。

另外还有有两个较少知名的HTTP动词：  
HEAD - 检索有关资源的元数据，例如数据的哈希或上次更新时间。  
OPTIONS - 检索关于客户端被允许对资源做什么的信息。

**2、路径（接口命名）**

路径又称"终点"（endpoint），表示API的具体网址。

在RESTful架构中，每个网址代表一种资源（resource），所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般来说，数据库中的表都是同种记录的"集合"（collection），所以API中的名词也应该使用复数。

举例来说，有一个API提供动物园（zoo）的信息，还包括各种动物和雇员的信息，则它的路径应该设计成下面这样。

接口尽量使用名词，禁止使用动词，下面是一些例子。

GET /zoos：列出所有动物园

POST /zoos：新建一个动物园

GET /zoos/ID：获取某个指定动物园的信息

PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）

PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）

DELETE /zoos/ID：删除某个动物园

GET /zoos/ID/animals：列出某个指定动物园的所有动物

DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物

反例：

/getAllCars

/createNewCar

/deleteAllRedCars

再比如，某个URI是/posts/show/1，其中show是动词，这个URI就设计错了，正确的写法应该是/posts/1，然后用GET方法表示show。

如果某些动作是HTTP动词表示不了的，你就应该把动作做成一种资源。比如网上汇款，从账户1向账户2汇款500元，错误的URI是：

POST /accounts/1/transfer/500/to/2

正确的写法是把动词transfer改成名词transaction，资源不能是动词，但是可以是一种服务：

POST /transaction HTTP/1.1

　　Host: 127.0.0.1

　　from=1&to=2&amount=500.00

理清资源的层次结构，比如业务针对的范围是学校，那么学校会是一级资源(/school)，老师(/school/teachers)，学生(/school/students)就是二级资源。

**3、版本（Versioning）**

应该将API的版本号放入URL。如：

https://api.example.com/v1/

另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。Github采用这种做法。

**4、过滤信息（Filtering）**

如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果。  
下面是一些常见的参数。

?limit=10：指定返回记录的数量

?offset=10：指定返回记录的开始位置。

?page\_number=2&page\_size=100：指定第几页，以及每页的记录数。

?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。

?animal\_type\_id=1：指定筛选条件

参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复。比如，

GET /zoo/ID/animals 与 GET /animals?zoo\_id=ID 的含义是相同的。

**5、状态码（Status Codes）**

状态码范围

1xx 信息，请求收到，继续处理。范围保留用于底层HTTP的东西，你很可能永远也用不到。

2xx 成功，行为被成功地接受、理解和采纳

3xx 重定向，为了完成请求，必须进一步执行的动作

4xx 客户端错误，请求包含语法错误或者请求无法实现。范围保留用于响应客户端做出的错误，例如。他们提供不良数据或要求不存在的东西。这些请求应该是幂等的，而不是更改服务器的状态。

5xx 范围的状态码是保留给服务器端错误用的。这些错误常常是从底层的函数抛出来的，甚至

开发人员也通常没法处理，发送这类状态码的目的以确保客户端获得某种响应。

当收到5xx响应时，客户端不可能知道服务器的状态，所以这类状态码是要尽可能的避免。

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。

201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。

202 Accepted - [\*]：表示一个请求已经进入后台排队（异步任务）

204 NO CONTENT - [DELETE]：用户删除数据成功。

400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

401 Unauthorized - [\*]：表示用户没有权限（令牌、用户名、密码错误）。

403 Forbidden - [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。

404 NOT FOUND - [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。

406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。

410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。

500 INTERNAL SERVER ERROR - [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。

502 网关错误

503 Service Unavailable

504 网关超时

**错误处理（Error handling）**

如果状态码是4xx，就应该向用户返回出错信息。一般来说，返回的信息中将error作为键名，出错信息作为键值即可。

{

error: "Invalid API key"

}

**返回结果**

针对不同操作，服务器向用户返回的结果应该符合以下规范。

GET /collection：返回资源对象的列表（数组）

GET /collection/resource：返回单个资源对象

POST /collection：返回新生成的资源对象

PUT /collection/resource：返回完整的资源对象

PATCH /collection/resource：返回完整的资源对象

DELETE /collection/resource：返回一个空文档

**13、Django的路由层，视图层，模板层，模型层的工作流程（略）**

**14、Flask的请求流程，请求上下文（栈实现）**

上下文：有两种  
RequestContext请求上下文：内部有request和session对象  
AppContext程序上下文：内部有g和current\_app对象  
第一步：创建上下文  
Flask根据WSGI Server封装的请求等的信息(environ)新建RequestContext对象 和AppContext对象  
第二步:入栈  
将RequestContext对象push进\_request\_ctx\_stack里面。在这次请求期间，访问request对象，session对象将指向这个栈的栈顶元素  
第三步:请求分发  
response = self.full\_dispatch\_request()  
Flask将调用full\_dispatch\_request函数进行请求的分发，之所以不用给参数，是因为我们可以通过request对象获得这次请求的信息。full\_dispatch\_request将根据请求的url找到对应的蓝本里面的视图函数，并生成一个response对象。注意的是，在请求之外的时间，访问request对象是无效的，因为request对象依赖请求期间的\_request\_ctx\_stack栈。  
第四步:上下文对象出栈  
这次HTTP的响应已经生成了，就不需要两个上下文对象了。分别将两个上下文对象出栈，为下一次的HTTP请求做出准备。  
第五步：响应WSGI  
调用Response对象，向WSGI Server返回其结果作为HTTP正文。Response对象是一个可调用对象，当调用发生时，将首先执行WSGI服务器传入的start\_response()函数 发送状态码和HTTP报文头。

**15、Django ORM跨表一次性查询**

select\_related内部自动连表，连表的时候比较消耗资源，但走数据库的次数少  
prefetch\_related内部不做连表，多次查询的时候比较消耗资源，但刚给用户的感觉和连表操作一样

**16、Celery Broker使用的什么，结果要保存吗，保存到哪里**

那么什么是Celery呢？

Celery是一个用Python开发的异步的分布式任务调度模块。

Celery本身不包含消息服务，使用第三方消息服务，也就是Broker，来传递任务，目前支持的有Rebbimq，Redis，数据库以及其他的一些比如Amazon SQS,Monogdb和IronMQ 。Celery 支持多种 broker, 但主要以 RabbitMQ 和 Redis 为主。

在 RabbitMQ 和 Redis之间，该如何选择呢？

因为项目里面用的是Redis，所以这里以Redis作为Broker。

Celery 官方明确表示推荐在生产环境下使用 RabbitMQ，Redis 存在丢数据的问题。所以如果你的业务可以容忍 worker crash 或者电源故障导致的任务丢失，采用 Redis 是个不错的选择，本篇就以 Redis 为例来介绍。

Celery 对于 Redis 的支持需要安装相关的依赖，以下命令可以同时安装 Celery 和 Redis 相关的依赖，但是 redis server 还是必须单独安装的。

pip install celery

使用Redis作为Broker的话，可以两者一块安装

sudo pip install -U celery[redis] # -U 的意思是把所有指定的包都升级到最新的版本

当然如果正式生产环境中，有可能redis服务器和Celery在不同的机器上面的话，就要两者单独安装

sudo apt-get install redis-server 这个命令可以安装redis，包括了redi-cli工具

**17、uWSGI的工作流程，基于什么协议，使用的什么配置（回答的默认）**

wsgi.py：django项目携带的一个wsgi接口文件，如果项目名叫destiny的话，此文件就位于[destiny/destiny/wsgi.py]。

nginx：常用高性能代理服务器。Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。

要注意 WSGI / uwsgi / uWSGI 这三个概念的区分。一个基于自有的uwsgi协议、wsgi协议和http服务协议的web网关

WSGI看过前面小节的同学很清楚了，是一种通信协议。

为什么有了uWSGI还需要nginx？：

因为nginx具备优秀的静态内容处理能力，然后将动态内容转发给uWSGI服务器，这样可以达到很好的客户端响应；而apache等，它们自己都没有解析动态语言如php的功能，而是分派给其他模块来做，比如apache就可以说内置了php模块，让人感觉好像apache就支持php一样。

接下来，我们要看看 uWSGI 的安装配置与使用。

wsgi：一种实现python解析的通用接口标准/协议，是一种通用的接口标准或者接口协议，实现了python web程序与服务器之间交互的通用性。利用它，web.py或bottle或者django等等的python web开发框架，就可以轻松地部署在不同的web server上了；

uwsgi同WSGI一样是一种通信协议。uwsgi协议是一个uWSGI服务器自有的协议，它用于定义传输信息的类型（type of information），每一个uwsgi packet前4byte为传输信息类型描述，它与WSGI相比是两样东西。

uWSGI :一种python web server或称为Server/Gateway，uWSGI类似tornadoweb或者flup，它实现了WSGI协议、uwsgi、http等协议Web服务器。负责响应python的web请求。

**17、uWSGI/nginx/django的工作原理流程**

流程：

1、首先客户端请求服务资源。

2、nginx作为直接对外的服务接口，接收到客户端发送过来的http请求，会解包、分析。

3、如果是静态文件请求就根据nginx配置的静态文件目录，返回请求的资源。

4、如果是动态的请求，nginx就通过配置文件，将请求传递给uWSGI；uWSGI将接收的包进行处理并传给wsgi。

5、wsgi根据请求调用django工程的某个文件或函数，处理完后django将返回追交给wsgi。

6、wsgi将返回值进行打包，转发给uWSGI。

7、uWSGI接收后转发给nginx，nginx最终将返回值返回给客户端（如浏览器）。

**18、快速排序，大量重复数据重复影响效率吗，怎么处理（会影响效率，优化没回答上来）**

会影响效率

解决办法：分组

三相切割快速排序：分三组

**19、Dockerfile中ADD和COPY的区别，是用ADD还是COPY**

COPY是从本地，推荐使用，从本地拷贝，不直接解压，ADD是增强版COPY，可以使用url拷贝，直接解压。

JWT校验后，是只能一个用户登录还是多用户可以同时在线(回答只能一个，分析了一下不能，追问怎么实现只能一个设备登录，没答上来)

实现一个用户登录：

将token存入redis，每次登录就刷新redis里的token，这样就能保证token永远只有一个

logging模块，日志过大怎么办（没答上）

python的元类，Django ORM底层用到元类(没答好)

元类其实就是产生类的类，我们可以通过元类来拦截类的创建过程，这个地方我自己通过元类写过一个简易版本的ORM

首先ORM全称叫对象关系映射，能够让不会数据库操作的程序员通过面向对象的方法简单快捷的操作数据库，ORM有三层映射关系

\* 类映射数据库的表

\* 对象映射成数据库的表中的一条条记录

\* 对象获取属性映射成数据库的表中的某条记录某个字段对应的值

具体做法就是在类创建过程中通过元类拦截它的创建，在类创建出来之前给类赋上表该有的属性表名，主键字段，其他普通字段

MySQL事务+悲观锁，直接用MySQL不会崩吗

**17、[使用mysql悲观锁解决并发问题](https://www.cnblogs.com/laoyeye/p/8228467.html)**

**最近学习了一下数据库的悲观锁和乐观锁，根据自己的理解和网上参考资料总结如下：**

**悲观锁介绍（百科）：**

悲观锁，正如其名，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）。

使用场景举例：以MySQL InnoDB为例

商品goods表中有一个字段status，status为1代表商品未被下单，status为2代表商品已经被下单，那么我们对某个商品下单时必须确保该商品status为1。假设商品的id为1。

1如果不采用锁，那么操作方法如下：

//1.查询出商品信息

select status from t\_goods where id=1;

//2.根据商品信息生成订单

insert into t\_orders (id,goods\_id) values (null,1);

//3.修改商品status为2

update t\_goods set status=2;

上面这种场景在高并发访问的情况下很可能会出现问题。

前面已经提到，只有当goods status为1时才能对该商品下单，上面第一步操作中，查询出来的商品status为1。但是当我们执行第三步Update操作的时候，有可能出现其他人先一步对商品下单把goods status修改为2了，但是我们并不知道数据已经被修改了，这样就可能造成同一个商品被下单2次，使得数据不一致。所以说这种方式是不安全的。

**2使用悲观锁来实现：**

在上面的场景中，商品信息从查询出来到修改，中间有一个处理订单的过程，使用悲观锁的原理就是，当我们在查询出goods信息后就把当前的数据锁定，直到我们修改完毕后再解锁。那么在这个过程中，因为goods被锁定了，就不会出现有第三者来对其进行修改了。

注：要使用悲观锁，我们必须关闭mysql数据库的自动提交属性，因为MySQL默认使用autocommit模式，也就是说，当你执行一个更新操作后，MySQL会立刻将结果进行提交。

我们可以使用命令设置MySQL为非autocommit模式：

set autocommit=0;

设置完autocommit后，我们就可以执行我们的正常业务了。具体如下：

//0.开始事务

begin;/begin work;/start transaction; (三者选一就可以)

//1.查询出商品信息

select status from t\_goods where id=1 for update;

//2.根据商品信息生成订单

insert into t\_orders (id,goods\_id) values (null,1);

//3.修改商品status为2

update t\_goods set status=2;

//4.提交事务

commit;/commit work;

**注：**上面的begin/commit为事务的开始和结束，因为在前一步我们关闭了mysql的autocommit，所以需要手动控制事务的提交，在这里就不细表了。

上面的第一步我们执行了一次查询操作：select status from t\_goods where id=1 for update;

与普通查询不一样的是，我们使用了select…for update的方式，这样就通过数据库实现了悲观锁。此时在t\_goods表中，id为1的 那条数据就被我们锁定了，其它的事务必须等本次事务提交之后才能执行。这样我们可以保证当前的数据不会被其它事务修改。

注：需要注意的是，在事务中，只有SELECT ... FOR UPDATE 或LOCK IN SHARE MODE 同一笔数据时会等待其它事务结束后才执行，一般SELECT ... 则不受此影响。拿上面的实例来说，当我执行select status from t\_goods where id=1 for update;后。我在另外的事务中如果再次执行select status from t\_goods where id=1 for update;则第二个事务会一直等待第一个事务的提交，此时第二个查询处于阻塞的状态，但是如果我是在第二个事务中执行select status from t\_goods where id=1;则能正常查询出数据，不会受第一个事务的影响。

补充：MySQL select…for update的Row Lock与Table Lock

上面我们提到，使用select…for update会把数据给锁住，不过我们需要注意一些锁的级别，MySQL InnoDB默认Row-Level Lock，所以只有「明确」地指定主键，MySQL 才会执行Row lock (只锁住被选取的数据) ，否则MySQL 将会执行Table Lock (将整个数据表单给锁住)。

**举例说明：**

**数据库表t\_goods，包括id,status,name三个字段，id为主键，数据库中记录如下;**

Sql代码

mysql> select \* from t\_goods;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

| 2 | 1 | 装备 |

+----+--------+------+

2 rows in set

mysql>

注：为了测试数据库锁，我使用两个console来模拟不同的事务操作，分别用console1、console2来表示。

**例1: (明确指定主键，并且有此数据，row lock)**

console1：查询出结果，但是把该条数据锁定了

Sql代码

mysql> select \* from t\_goods where id=1 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

+----+--------+------+

1 row in set

mysql>

console2：查询被阻塞

Sql代码

mysql> select \* from t\_goods where id=1 for update;

console2：如果console1长时间未提交，则会报错

Sql代码

mysql> select \* from t\_goods where id=1 for update;

ERROR 1205 : Lock wait timeout exceeded; try restarting transaction

**例2: (明确指定主键，若查无此数据，无lock)**

console1：查询结果为空

Sql代码

mysql> select \* from t\_goods where id=3 for update;

Empty set

console2：查询结果为空，查询无阻塞，说明console1没有对数据执行锁定

Sql代码

mysql> select \* from t\_goods where id=3 for update;

Empty set

**例3: (无主键，table lock)**

console1：查询name=道具 的数据，查询正常

Sql代码

mysql> select \* from t\_goods where name='道具' for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

+----+--------+------+

1 row in set

mysql>

console2：查询name=装备 的数据，查询阻塞，说明console1把表给锁住了

Sql代码

mysql> select \* from t\_goods where name='装备' for update;

console2：若console1长时间未提交，则查询返回为空

Sql代码

mysql> select \* from t\_goods where name='装备' for update;

Query OK, -1 rows affected

**例4: (主键不明确，table lock)**

console1：查询正常

Sql代码

mysql> begin;

Query OK, 0 rows affected

mysql> select \* from t\_goods where id>0 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

| 2 | 1 | 装备 |

+----+--------+------+

2 rows in set

mysql>

console2：查询被阻塞，说明console1把表给锁住了

Sql代码

mysql> select \* from t\_goods where id>1 for update;

**例5: (主键不明确，table lock)**

console1：

Sql代码

mysql> begin;

Query OK, 0 rows affected

mysql> select \* from t\_goods where id<>1 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 2 | 1 | 装备 |

+----+--------+------+

1 row in set

mysql>

console2：查询被阻塞，说明console1把表给锁住了

Sql代码

mysql> select \* from t\_goods where id<>2 for update;

console1：提交事务

Sql代码

mysql> commit;

Query OK, 0 rows affected

console2：console1事务提交后，console2查询结果正常

Sql代码

mysql> select \* from t\_goods where id<>2 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

+----+--------+------+

1 row in set

mysql>

以上就是关于数据库主键对MySQL锁级别的影响实例，需要注意的是，除了主键外，使用索引也会影响数据库的锁定级别

举例：

我们修改t\_goods表，给status字段创建一个索引

修改id为2的数据的status为2，此时表中数据为：

Sql代码

mysql> select \* from t\_goods;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

| 2 | 2 | 装备 |

+----+--------+------+

2 rows in set

mysql>

**例6: (明确指定索引，并且有此数据，row lock)**

console1：

Sql代码

mysql> select \* from t\_goods where status=1 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 1 | 1 | 道具 |

+----+--------+------+

1 row in set

mysql>

console2：查询status=1的数据时阻塞，超时后返回为空，说明数据被console1锁定了

Sql代码

mysql> select \* from t\_goods where status=1 for update;

Query OK, -1 rows affected

console2：查询status=2的数据，能正常查询，说明console1只锁住了行，未锁表

Sql代码

mysql> select \* from t\_goods where status=2 for update;

+----+--------+------+

| id | status | name |

+----+--------+------+

| 2 | 2 | 装备 |

+----+--------+------+

1 row in set

mysql>

**例7: (明确指定索引，若查无此数据，无lock)**

console1：查询status=3的数据，返回空数据

Sql代码

mysql> select \* from t\_goods where status=3 for update;

Empty set

console2：查询status=3的数据，返回空数据

Sql代码

mysql> select \* from t\_goods where status=3 for update;

Empty set

以上就是关于我对数据库悲观锁的理解和总结，有不对的地方欢迎拍砖，**链接：**[**使用mysql乐观锁解决并发问题**](http://www.cnblogs.com/laoyeye/p/8097684.html)

[返回主页](https://www.cnblogs.com/angle6-liu/)

# [MongoDB面试题](https://www.cnblogs.com/angle6-liu/p/10791875.html)

**1.什么是MongoDB**

MongoDB是一个文档数据库，提供好的性能，领先的非关系型数据库。采用BSON存储文档数据。  
BSON（）是一种类json的一种二进制形式的存储格式，简称Binary JSON.  
相对于json多了date类型和二进制数组。

**2.MongoDB的优势有哪些**

* 面向文档的存储：以 JSON 格式的文档保存数据。
* 任何属性都可以建立索引。
* 复制以及高可扩展性。
* 自动分片。
* 丰富的查询功能。
* 快速的即时更新。

**3 什么是数据库**

数据库可以看成是一个电子化的文件柜,用户可以对文件中的数据运行新增、检索、更新、删除等操作。数据库是一个所有集合的容器，在文件系统中每一个数据库都有一个相关的物理文件。

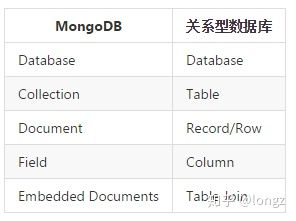
**4.什么是集合(表)**

集合就是一组 MongoDB 文档。它相当于关系型数据库（RDBMS）中的表这种概念。集合位于单独的一个数据库中。一个集合内的多个文档可以有多个不同的字段。一般来说，集合中的文档都有着相同或相关的目的。

**5 什么是文档(记录)**

　　文档由一组key value组成。文档是动态模式,这意味着同一集合里的文档不需要有相同的字段和结构。在关系型数据库中table中的每一条记录相当于MongoDB中的一个文档

**6 MongoDB和关系型数据库术语对比图**



**7.什么是非关系型数据库**

　　非关系型数据库的显著特点是不使用SQL作为查询语言，数据存储不需要特定的表格模式。

**8 为什么用MongoDB？**

* 架构简单
* 没有复杂的连接
* 深度查询能力,MongoDB支持动态查询。
* 容易调试
* 容易扩展
* 不需要转化/映射应用对象到数据库对象
* 使用内部内存作为存储工作区,以便更快的存取数据。

**9 在哪些场景使用MongoDB**

* 大数据
* 内容管理系统
* 移动端Apps
* 数据管理

**10 MongoDB中的命名空间是什么意思?**

mongodb存储bson对象在丛集(collection)中.数据库名字和丛集名字以句点连结起来叫做名字空间(namespace).

　　一个集合命名空间又有多个数据域(extent)，集合命名空间里存储着集合的元数据，比如集合名称，集合的  
第一个数据域和最后一个数据域的位置等等。而一个数据域由若干条文档(document)组成，每个数据域都有一个  
头部，记录着第一条文档和最后一条文档的为知，以及该数据域的一些元数据。extent之间，document之间通过  
双向链表连接。

索引的存储数据结构是B树，索引命名空间存储着对B树的根节点的指针。

**11 monogodb 中的分片什么意思**

　　分片是将数据水平切分到不同的物理节点。当应用数据越来越大的时候，数据量也会越来越大。当数据量增长  
时，单台机器有可能无法存储数据或可接受的读取写入吞吐量。利用分片技术可以添加更多的机器来应对数据量增加  
以及读写操作的要求。

**12 为什么要在MongoDB中使用分析器**

　　mongodb中包括了一个可以显示数据库中每个操作性能特点的数据库分析器.通过这个分析器你可以找到比预期慢  
的查询(或写操作);利用这一信息,比如,可以确定是否需要添加索引.

**13 .MongoDB支持主键外键关系吗**

　　默认MongoDB不支持主键和外键关系。 用Mongodb本身的API需要硬编码才能实现外键关联，不够直观且难度  
较大

**14 MongoDB支持哪些数据类型**

* + String
  + Integer
  + Double
  + Boolean
  + Object
  + Object ID
  + Arrays
  + Min/Max Keys
  + Datetime
  + Code
  + Regular Expression等

**15 为什么要在MongoDB中用"Code"数据类型**

"Code"类型用于在文档中存储 JavaScript 代码。

**16 为什么要在MongoDB中用"Regular Expression"数据类型**

"Regular Expression"类型用于在文档中存储正则表达式

**17 为什么在MongoDB中使用"Object ID"数据类型**

"ObjectID"数据类型用于存储文档id

**18"ObjectID"有哪些部分组成**

一共有四部分组成:时间戳、客户端ID、客户进程ID、三个字节的增量计数器

**19 在MongoDb中什么是索引**

索引用于高效的执行查询,没有索引的MongoDB将扫描整个集合中的所有文档,这种扫描效率很低,需要处理大量  
的数据.

索引是一种特殊的数据结构,将一小块数据集合保存为容易遍历的形式.索引能够存储某种特殊字段或字段集的  
值,并按照索引指定的方式将字段值进行排序.

**20 如何添加索引**

使用db.collection.createIndex()在集合中创建一个索引

**21.如何查询集合中的文档**

db.collectionName.find({key:value})

**22用什么方法可以格式化输出结果**

db.collectionName.find().pretty()

**23 如何使用"AND"或"OR"条件循环查询集合中的文档**

db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

**24 更新数据**

db.collectionName.update({key:value},{$set:{newkey:newValue}})

**25 如何删除文档**

db.collectionName.remove({key:value})

**26 在MongoDB中如何排序**

并使用 1 和 -1 来指定排序方式，其中 1 表示升序，而 -1 表示降序。

db.connectionName.find({key:value}).sort({columnName:1})

**27 什么是聚合**

聚合操作能够处理数据记录并返回计算结果。聚合操作能将多个文档中的值组合起来，对成组数据执行各种操作，返回单一的结果。它相当于 SQL 中的 count(\*) 组合 group by。

对于 MongoDB 中的聚合操作，应该使用aggregate()方法。

db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

**28 在MongoDB中什么是副本集（避免单点故障）**

在MongoDB中副本集由一组MongoDB实例组成，包括一个主节点多个次节点，MongoDB客户端的所有数据都写入主节点(Primary),副节点从主节点同步写入数据，以保持所有复制集内存储相同的数据，提高数据可用性。

**29 什么是NoSQL数据库？NoSQL和RDBMS有什么区别？在哪些情况下使用和不使用NoSQL数据库？**

　NoSQL是非关系型数据库，NoSQL = Not Only SQL。  
 关系型数据库采用的结构化的数据，NoSQL采用的是键值对的方式存储数据。

在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用NoSQL数据库。

在考虑数据库的成熟度；支持；分析和商业智能；管理及专业性等问题时，应优先考虑关系型数据库。

**30 MongoDB支持存储过程吗？如果支持的话，怎么用？**

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

**31如何理解MongoDB中的GridFS机制，MongoDB为何使用GridFS来存储文件？**

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

**32 为什么MongoDB的数据文件很大？**

　　MongoDB采用的预分配空间的方式来防止文件碎片。

**33 当更新一个正在被迁移的块（Chunk）上的文档时会发生什么？**

　　更新操作会立即发生在旧的块（Chunk）上，然后更改才会在所有权转移前复制到新的分片上。

**34 MongoDB在A:{B,C}上建立索引，查询A:{B,C}和A:{C,B}都会使用索引吗？**

　　不会，只会在A:{B,C}上使用索引。

**35 mongodb成为最好nosql数据库的原因是什么?**

　　面向文件的 高性能 高可用性 易扩展性 丰富的查询语言

**36 如果用户移除对象的属性,该属性是否从存储层中删除?**

　　是的,用户移除属性然后对象会重新保存(re-save()).

**37 允许空值null吗?**

　　对于对象成员而言,是的.然而用户不能够添加空值(null)到数据库丛集(collection)因为空值不是对象.然而用户能够添加空对象{}.

**38 更新操作立刻fsync到磁盘?**

　　不会,磁盘写操作默认是延迟执行的.写操作可能在两三秒(默认在60秒内)后到达磁盘.例如,如果一秒内数据库收到一千个对一个对象递增的操作,仅刷新磁盘一次.

**39 如何执行事务/加锁?**

　　mongodb没有使用传统的锁或者复杂的带回滚的事务,因为它设计的宗旨是轻量,快速以及可预计的高性能.可以把它类比成mysql mylsam的自动提交模式.通过精简对事务的支持,性能得到了提升,特别是在一个可能会穿过多个服务器的系统里.

**40  启用备份故障恢复需要多久?**

　　从备份数据库声明主数据库宕机到选出一个备份数据库作为新的主数据库将花费10到30秒时间.这期间在主数据库上的操作将会失败–包括写入和强一致性读取(strong consistent read)操作.然而,你还能在第二数据库上执行最终一致性查询(eventually consistent query)(在slaveok模式下),即使在这段时间里.

**41  什么是master或primary?**

　　它是当前备份集群(replica set)中负责处理所有写入操作的主要节点/成员.在一个备份集群中,当失效备援(failover)事件发生时,一个另外的成员会变成primary.

**42  我应该启动一个集群分片(sharded)还是一个非集群分片的 mongodb 环境?**

　　(数据量大用集群分片,数据量小用非集群)

　　为开发便捷起见,我们建议以非集群分片(unsharded)方式开始一个 mongodb 环境,除非一台服务器不足以存放你的初始数据集.从非集群分片升级到集群分片(sharding)是无缝的,所以在你的数据集还不是很大的时候没必要考虑集群分片(sharding).

**43 分片(sharding)和复制(replication)是怎样工作的?**

　　每一个分片(shard)是一个分区数据的逻辑集合.分片可能由单一服务器或者集群组成,我们推荐为每一个分片(shard)使用集群.

**44数据在什么时候才会扩展到多个分片(shard)里?**

　　mongodb 分片是基于区域(range)的.所以一个集合(collection)中的所有的对象都被存放到一个块(chunk)中.只有当存在多余一个块的时候,才会有多个分片获取数据的选项.现在,每个默认块的大小是 64mb,所以你需要至少 64 mb 空间才可以实施一个迁移.

**45 当我试图更新一个正在被迁移的块(chunk)上的文档时会发生什么?**

　　更新操作会立即发生在旧的分片(shard)上,然后更改才会在所有权转移(ownership transfers)前复制到新的分片上.

**46 如果在一个分片(shard)停止或者很慢的时候,我发起一个查询会怎样?**

　　如果一个分片(shard)停止了,除非查询设置了“partial”选项,否则查询会返回一个错误.如果一个分片(shard)响应很慢,mongodb则会等待它的响应.

**47 可以把movechunk目录里的旧文件删除吗?**

　　没问题,这些文件是在分片(shard)进行均衡操作(balancing)的时候产生的临时文件.一旦这些操作已经完成,相关的临时文件也应该被删除掉.但目前清理工作是需要手动的,所以请小心地考虑再释放这些文件的空间.

**48 如果块移动操作(movechunk)失败了,我需要手动清除部分转移的文档吗?**

 　　不需要,移动操作是一致(consistent)并且是确定性的(deterministic);一次失败后,移动操作会不断重试;当完成后,数据只会出现在新的分片里(shard).

**49 mongodb是否支持事务**

　　MongoDB 4.0的新特性——事务（Transactions）：MongoDB 是不支持事务的，因此开发者在需要用到事务的时候，不得不借用其他工具，在业务代码层面去弥补数据库的不足。

事务和会话(Sessions)关联，一个会话同一时刻只能开启一个事务操作，当一个会话断开，这个会话中的事务也会结束。

# [Zabbix与ELK整合实现对日志数据的实时监控](https://www.cnblogs.com/flytor/p/11440799.html)

# [Zabbix与ELK整合实现对日志数据的实时监控](https://www.cnblogs.com/flytor/p/11440799.html)

# 一、 ELK与zabbix有什么关系？

ELK大家应该比较熟悉了，zabbix应该也不陌生，那么将ELK和zabbix放到一起的话，可能大家就有疑问了？这两个放到一起是什么目的呢，听我细细道来。

ELK是一套日志收集套件，它其实有由Elasticsearch、Logstash和Kibana三个软件组成，通过ELK可以收集系统日志、网站日志、应用系统日志等各种日志数据，并且还可以对日志进行过滤、清洗，然后进行集中存放并可用于实时检索、分析。这是ELK的基础功能。

但是有些时候，我们希望在收集日志的时候，能够将日志中的异常信息（警告、错误、失败等信息）及时的提取出来，因为日志中的异常信息意味着操作系统、应用程序可能存在故障，如果能将日志中的故障信息及时的告知运维人员，那么运维就可以第一时间去进行故障排查和处理，进而也就可以避免很多故障的发生。

那么如何才能做到将ELK收集的日志数据中出现的异常信息及时的告知运维人员呢，这就需要用到zabbix了，ELK（更确切的说应该是logstash）可以实时的读取日志的内容，并且还可以过滤日志信息，通过ELK的读取和过滤功能，就可以将日志中的一些异常关键字（error、failed、OutOff、Warning）过滤出来，然后通过logstash的zabbix插件将这个错误日志信息发送给zabbix，那么zabbix在接收到这个数据后，结合自身的机制，然后发起告警动作，这样就实现了日志异常zabbix实时告警的功能了。

# 二、Logstash与zabbix插件的使用

Logstash支持多种输出介质，比如syslog、HTTP、TCP、elasticsearch、kafka等，而有时候我们想将收集到的日志中一些错误信息输出，并告警时，就用到了logstash-output-zabbix这个插件，此插件可以将Logstash与zabbix进行整合，也就是将Logstash收集到的数据进行过滤，将有错误标识的日志输出到zabbix中，最后通过zabbix的告警机制进行触发、告警。

logstash-output-zabbix是一个社区维护的插件，它默认没有在Logstash中安装,但是安装起来也很容易，直接在logstash中运行如下命令即可：

/usr/local/logstash/bin/logstash-plugin install logstash-output-zabbix

其中，/usr/local/logstash是Logstash的安装目录。

此外，logstash-plugin命令还有多种用法，我们来看一下：

## 2.1、列出目前已经安装的插件

将列出所有已安装的插件

/usr/local/logstash/bin/logstash-plugin list

#将列出已安装的插件及版本信息

/usr/local/logstash/bin/logstash-plugin list --verbose

#将列出包含namefragment的所有已安装插件

/usr/local/logstash/bin/logstash-plugin list "http"

#将列出特定组的所有已安装插件（ input，filter，codec，output）

/usr/local/logstash/bin/logstash-plugin list --group input

## 2.2、安装插件

要安装某个插件，例如安装kafka插件，可执行如下命令：

/usr/local/logstash/bin/logstash-plugin install logstash-output-kafka  
要使用此命令安装插件，需要你的电脑可以访问互联网。此插件安装方法，会检索托管在公共存储库（RubyGems.org）上的插件，然后下载到本地机器并在Logstash安装之上进行自动安装。

## 2.3、更新插件

每个插件有自己的发布周期和版本更新，这些更新通常是独立于Logstash的发布周期的。因此，有时候需要单独更新插件，可以使用update子命令获得最新版本的插件。

将更新所有已安装的插件

/usr/local/logstash/bin/logstash-plugin update

将仅更新指定的插件

/usr/local/logstash/bin/logstash-plugin update logstash-output-kafka

## 2.4、删除插件

如果需要从Logstash插件中删除插件，可执行如下命令：

/usr/local/logstash/bin/logstash-plugin remove logstash-output-kafka

这样就删除了logstash-output-kafka插件。

# 三、logstash-output-zabbix插件的使用

logstash-output-zabbix安装好之后，就可以在logstash配置文件中使用了，  
下面是一个logstash-output-zabbix使用的例子：

zabbix {

zabbix\_host => "[@metadata][zabbix\_host]"

zabbix\_key => "[@metadata][zabbix\_key]"

zabbix\_server\_host => "x.x.x.x"

zabbix\_server\_port => "xxxx"

zabbix\_value => "xxxx"

}

其中：

zabbix\_host：表示Zabbix主机名字段名称， 可以是单独的一个字段， 也可以是 @metadata 字段的子字段， 是必需的设置，没有默认值。

zabbix\_key：表示Zabbix项目键的值，也就是zabbix中的item，此字段可以是单独的一个字段， 也可以是 @metadata 字段的子字段，没有默认值。

zabbix\_server\_host：表示Zabbix服务器的IP或可解析主机名，默认值是 "localhost"，需要设置为zabbix server服务器所在的地址。

zabbix\_server\_port：表示Zabbix服务器开启的监听端口，默认值是10051。

zabbix\_value：表示要发送给zabbix item监控项的值对应的字段名称，默认值是 "message"，也就是将"message"字段的内容发送给上面zabbix\_key定义的zabbix item监控项，当然也可以指定一个具体的字段内容发送给zabbix item监控项。

# 四、将logstash与zabbix进行整合

这里我们以logstash收集日志，然后对日志进行读取，最后选择关键字进行过滤并调用zabbix告警的流程，来看看如何配置logstash实现zabbix告警。

先说明一下我们的应用需求：通过对系统日志文件的监控，然后去过滤日志信息中的一些关键字，例如ERR、error、ERROR、Failed、WARNING等，将日志中这些信息过滤出来，然后发送到zabbix上，最后借助zabbix的报警功能实现对系统日志中有上述关键字的告警。

对于过滤关键字，进行告警，不同的业务系统，可能关键字不尽相同，例如对http系统，可能需要过滤500、403、503等这些错误码，对于java相关的系统，可能需要过滤OutOfMemoryError、PermGen、Java heap等关键字。在某些业务系统的日志输出中，可能还有一些自定义的错误信息，那么这些也需要作为过滤关键字来使用。

## 4.1、配置logstash事件配置文件

接下来就是创建一个logstash事件配置文件，这里将配置文件分成三个部分来介绍，首先是input部分，内容如下：

input {

file {

path => ["/var/log/secure"]

type => "system"

start\_position => "beginning"

}

}

input部分是从/var/log/secure文件中读取数据，start\_position 表示从secure文件开头读取内容。

接着是filter部分，内容如下：

filter {

grok {

match => { "message" => "%{SYSLOGTIMESTAMP:message\_timestamp} %{SYSLOGHOST:hostname} %{DATA:message\_program}(?:\[%{POSINT:messag

e\_pid}\])?: %{GREEDYDATA:message\_content}" } #这里通过grok对message字段的数据进行字段划分，这里将message字段划分了5个子字段。其中，message\_content字段会在output中用到。

}

mutate {

add\_field => [ "[zabbix\_key]", "oslogs" ] #新增的字段，字段名是zabbix\_key，值为oslogs。

add\_field => [ "[zabbix\_host]", "%{host}" ] #新增的字段，字段名是zabbix\_host，值可以在这里直接定义，也可以引用字段变量来获取。这里的%{host}获取的就是日志数据的主机名，这个主机名与zabbix web中“主机名称”需要保持一致。

}

mutate { #这里是删除不需要的字段

remove\_field => "@version"

remove\_field => "message"

}

date { #这里是对日志输出中的日期字段进行转换，其中message\_timestamp字段是默认输出的时间日期字段，将这个字段的值传给 @timestamp字段。

match => [ "message\_timestamp","MMM d HH:mm:ss", "MMM dd HH:mm:ss", "ISO8601"]

}

}

filter部分是个重点，在这个部分中，重点关注的是message\_timestamp字段、message\_content字段。

最后是output部分，内容如下：

output {

if [message\_content] =~ /(ERR|error|ERROR|Failed)/ { #定义在message\_content字段中，需要过滤的关键字信息，也就是在message\_content字段中出现给出的这些关键字，那么就将这些信息发送给zabbix。

zabbix {

zabbix\_host => "[zabbix\_host]" #这个zabbix\_host将获取上面filter部分定义的字段变量%{host}的值

zabbix\_key => "[zabbix\_key]" #这个zabbix\_key将获取上面filter部分中给出的值

zabbix\_server\_host => "172.16.213.140" #这是指定zabbix server的IP地址

zabbix\_server\_port => "10051" #这是指定zabbix server的监听端口

zabbix\_value => "message\_content" #这个很重要，指定要传给zabbix监控项item（oslogs）的值， zabbix\_value默认的值是"message"字段，因为上面我们已经删除了"message"字段，因此，这里需要重新指定，根据上面filter部分对"message"字段的内容划分，这里指定为"message\_content"字段，其实，"message\_content"字段输出的就是服务器上具体的日志内容。

}

}

#stdout { codec => rubydebug } #这里是开启调试模式，当第一次配置的时候，建议开启，这样过滤后的日志信息直接输出的屏幕，方便进行调试，调试成功后，即可关闭。

}

将上面三部分内容合并到一个文件file\_to\_zabbix.conf中，然后启动logstash服务：

[root@logstashserver ~]#cd /usr/local/logstash

[root@logstashserver logstash]#nohup bin/logstash -f config/file\_to\_zabbix.conf --path.data /data/osdata &

这里的–path.data是指定此logstash进程的数据存储目录，用于在一个服务器上启动多个logstash进程的环境中。

## 4.2、zabbix平台配置日志告警

登录zabbix web平台，选择配置—>模板—>创建模板，名称定为logstash-output-zabbix，如下图所示：



接着，在此模块下创建一个应用集，点击应用集—–>创建应用集，如下图所示：



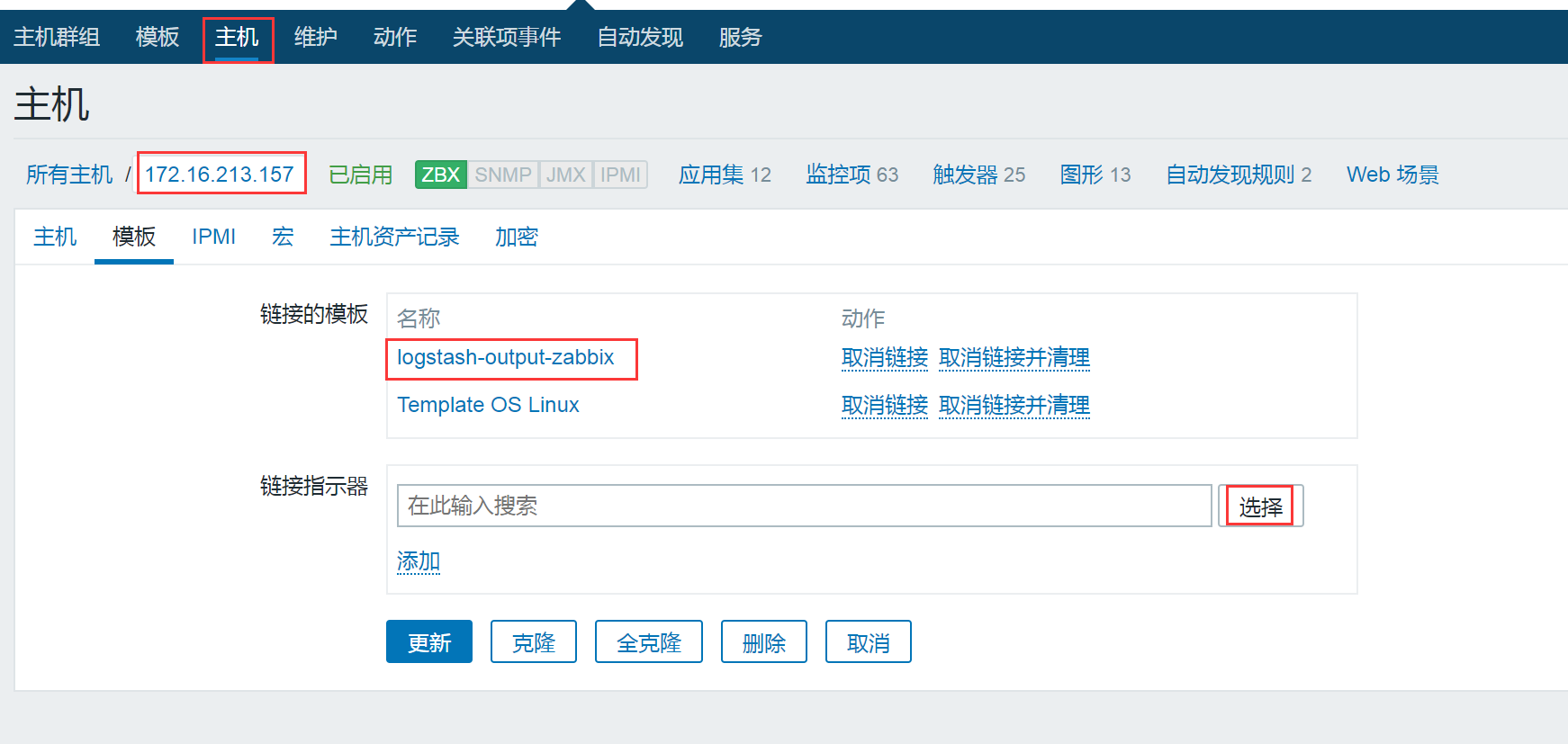
然后，在此模块下创建一个监控项，点击监控项—–>创建监控项，如下图所示：



到此为止，zabbix监控logstash的日志数据配置完成。

这里我们以客户端172.16.213.157主机为例，也就是监控172.16.213.157主机上的系统日志数据，当发现日志异常就进行告警。

在上面创建了一个模板和监控项，接着还需要将此模板链接到172.16.213.157主机上，选择“配置”—“主机”，然后选中172.16.213.157主机，选择“模板”标签，将上面创建的模板加入到172.16.213.157主机上，如下图所示：



这样，上面创建的监控项在172.16.213.157主机上就自动生效了。

下面我们模拟一个故障，在任意主机通过ssh登录172.16.213.157主机，然后输入一个错误密码，让系统的/var/log/secure文件产生错误日志，然后看看logstash是否能够过滤到，是否能够发送到zabbix中。

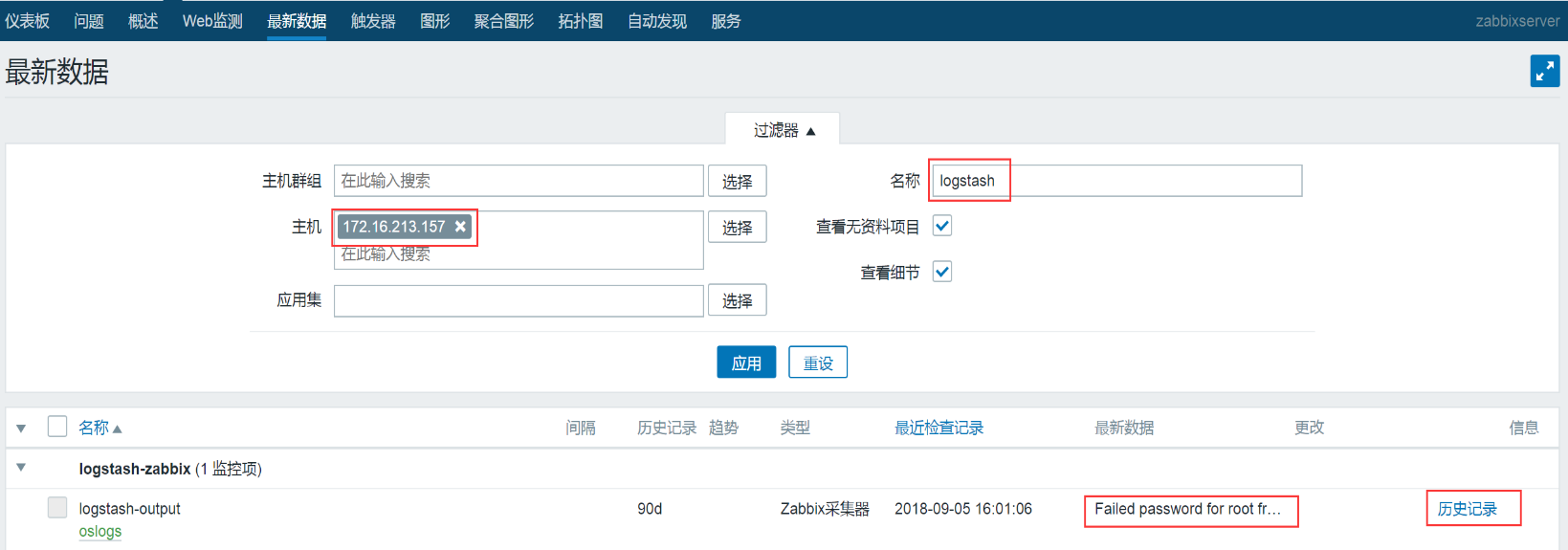
首先让系统文件/var/log/secure产生类似如下内容：

Sep 5 16:01:04 localhost sshd[27159]: pam\_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=172.16.213.127 user=root

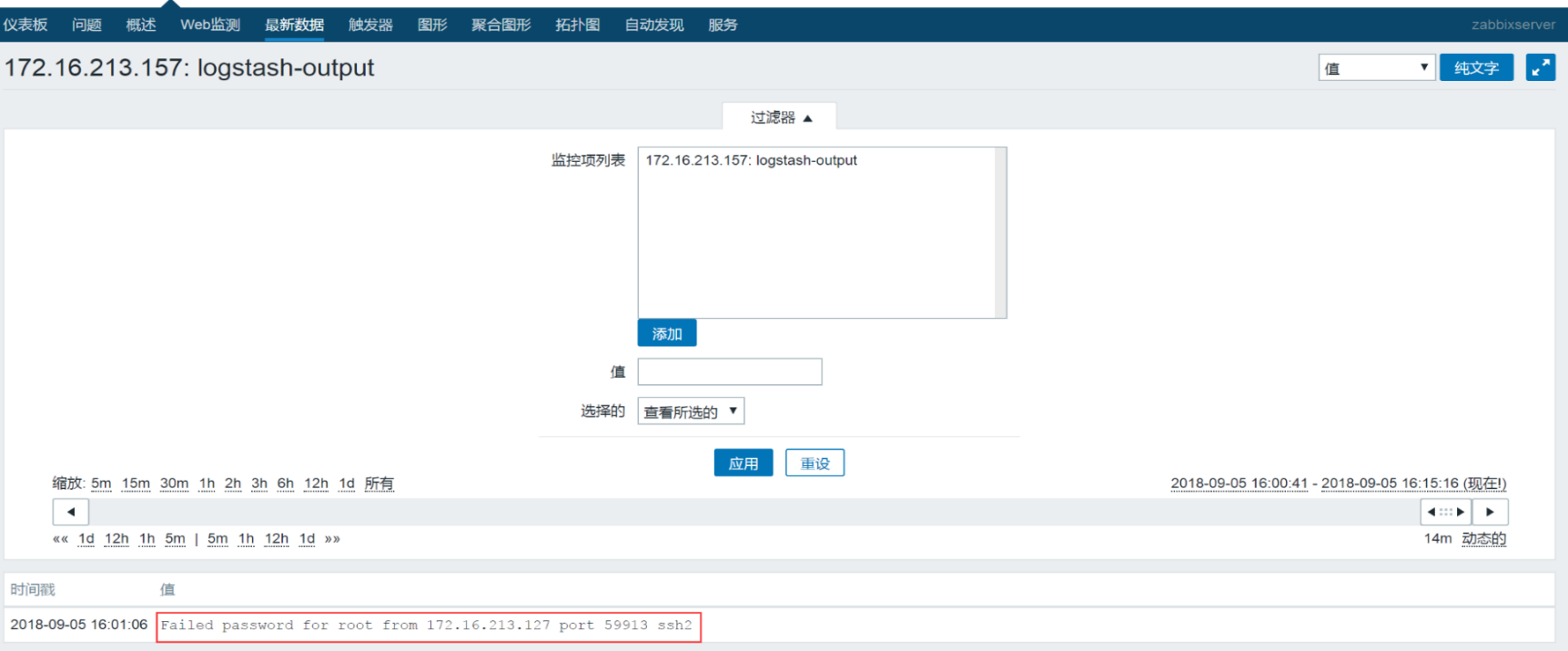
Sep 5 16:01:06 localhost sshd[27159]: Failed password for root from 172.16.213.127 port 59913 ssh2

这里面有我们要过滤的关键字Failed，因此logstash会将此内容过滤出来，发送到zabbix上。

接着，登录zabbix web平台，点击监测中—–>最新数据，如果zabbix能够接收到日志，就可以看到下图的最新数据：



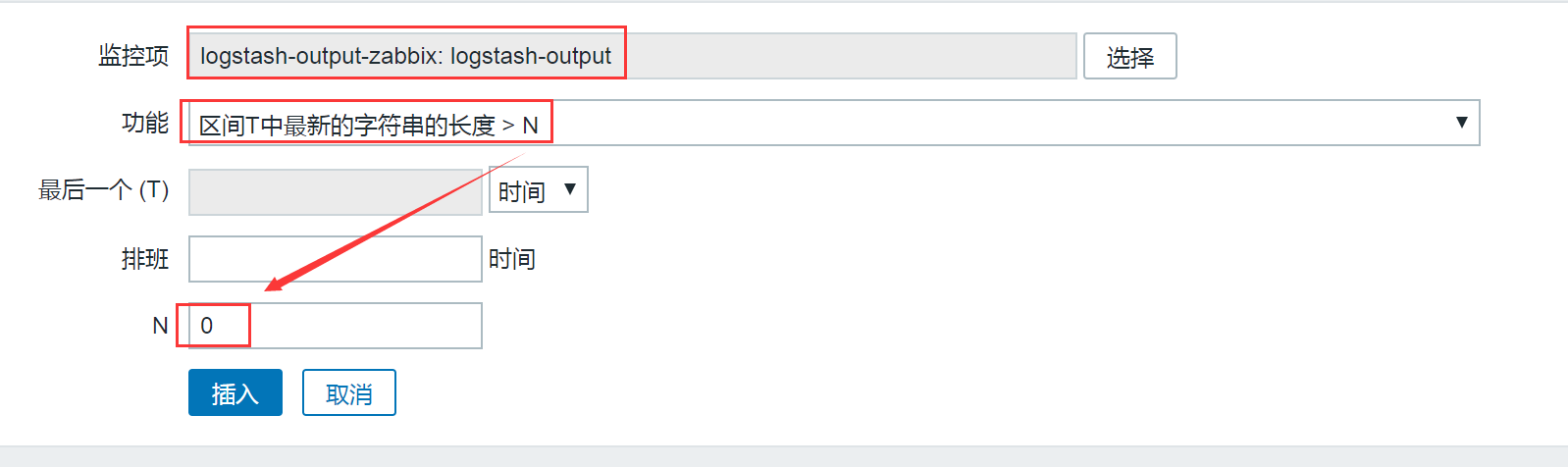
点击历史记录，可以查看详细内容，如下图所示：



可以看到，红框中的内容就是在logstash中定义的message\_content字段的内容。

到这里为止，zabbix已经可以收到logstash的发送过来的数据了，但是要实现报警，还需要在zabbix中创建一个触发器，进入配置—–>模板，选择logstash-output-zabbix这个模板，然后点击上面的触发器，继续点击右上角的创建触发器，如下图所示：





这里注意触发器创建中，表达式的写法，这里触发器的触发条件是：如果接收到logstash发送过来的数据，就进行告警，也就是说接收到的数据，如果长度大于0就告警。

触发器配置完成后，如果配置正常，就会进行告警了，告警内容如下图所示：



[**Zabbix 3.0 从入门到精通(zabbix使用详解)**](https://www.cnblogs.com/clsn/p/7885990.html)

**第1章 zabbix监控**

**1.1 为什么要监控**

   　　在需要的时刻，提前提醒我们服务器出问题了

  　　 当出问题之后，可以找到问题的根源

 　　  网站/服务器 的可用性

**1.1.1 网站可用性**

　　在软件系统的高可靠性（也称为可用性，英文描述为HA，High Available）里有个衡量其可靠性的标准——X个9，这个X是代表数字3~5。X个9表示在软件系统1年时间的使用过程中，系统可以正常使用时间与总时间（1年）之比，我们通过下面的计算来感受下X个9在不同级别的可靠性差异。

1个9：(1-90%)\*365=36.5天，表示该软件系统在连续运行1年时间里最多可能的业务中断时间是36.5天

2个9：(1-99%)\*365=3.65天 ， 表示该软件系统在连续运行1年时间里最多可能的业务中断时间是3.65天

3个9：(1-99.9%)\*365\*24=8.76小时，表示该软件系统在连续运行1年时间里最多可能的业务中断时间是8.76小时。

4个9：(1-99.99%)\*365\*24=0.876小时=52.6分钟，表示该软件系统在连续运行1年时间里最多可能的业务中断时间是52.6分钟。

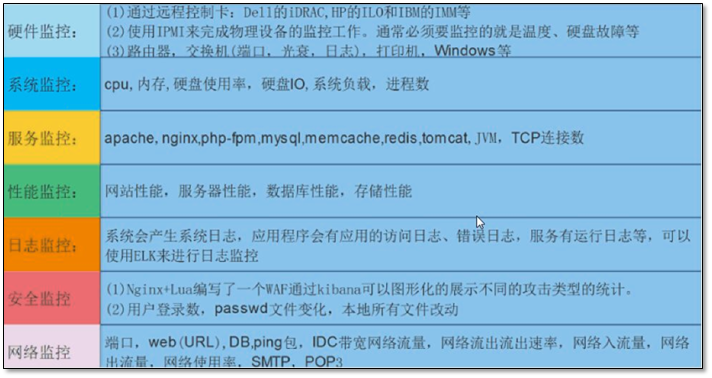
5个9：(1-99.999%)\*365\*24\*60=5.26分钟，表示该软件系统在连续运行1年时间里最多可能的业务中断时间是5.26分钟。

6个9：(1-99.9999%)\*365\*24\*60\*60=31秒， 示该软件系统在连续运行1年时间里最多可能的业务中断时间是31秒

**1.2 监控什么东西**

监控一切需要监控的东西，只要能够想到，能够用命令实现的都能用来监控

**1.2.1 监控范畴**



**1.3 怎么来监控**

**1.3.1 远程管理服务器**

如果想**远程管理服务器**就有远程管理卡，比如Dell idRAC，HP ILO，IBM IMM

**1.3.2 监控硬件**

查看硬件的温度/风扇转速，电脑有鲁大师，服务器就有ipmitool。

使用ipmitool实现对服务器的命令行远程管理

yum -y install OpenIPMI ipmitool #->IPMI在物理机可以成功，虚拟机不行

[root@KVM ~]# ipmitool sdr type Temperature

Temp | 01h | ns | 3.1 | Disabled

Temp | 02h | ns | 3.2 | Disabled

Temp | 05h | ns | 10.1 | Disabled

Temp | 06h | ns | 10.2 | Disabled

Ambient Temp | 0Eh | ok | 7.1 | 22 degrees C

Planar Temp | 0Fh | ns | 7.1 | Disabled

IOH THERMTRIP | 5Dh | ns | 7.1 | Disabled

CPU Temp Interf | 76h | ns | 7.1 | Disabled

Temp | 0Ah | ns | 8.1 | Disabled

Temp | 0Bh | ns | 8.1 | Disabled

Temp | 0Ch | ns | 8.1 | Disabled

**1.3.3 查看cpu相关**

　　lscpu、uptime、top、htop vmstat mpstat

   其中htop需要安装，安装依赖与epel源。

[znix@clsn ~]$lscpu

Architecture: x86\_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

CPU(s): 1

On-line CPU(s) list: 0

Thread(s) per core: 1

Core(s) per socket: 1

Socket(s): 1

NUMA node(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 85

Model name: Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz

Stepping: 4

CPU MHz: 2494.150

BogoMIPS: 4988.30

Hypervisor vendor: KVM

Virtualization type: full

L1d cache: 32K

L1i cache: 32K

L2 cache: 1024K

L3 cache: 33792K

NUMA node0 CPU(s): 0

**1.3.4 内存够不够可以用**

　　free

[znix@clsn ~]$free -h

total used free shared buffers cached

Mem: 996M 867M 128M 712K 145M 450M

-/+ buffers/cache: 271M 725M

Swap: 1.0G 0B 1.0G

**1.3.5 磁盘剩多少写的快不快可以用**

　　df、dd、iotop

[znix@clsn ~]$df -h

Filesystem Size Used Avail Use% Mounted on

/dev/vda1 40G 24G 15G 62% /

tmpfs 499M 20K 499M 1% /dev/shm

/dev/vdb1 20G 4.4G 15G 24% /data

**1.3.6 监控网络**

　　iftop nethogs

iftop 监控主机间流量 -i 指定监控网卡

nethogs 监控进程流量

**1.4 监控工具总览**

　　mrtg 流量监控出图

　　nagios 监控

　　cacti  流量监控出图

**zabbix 监控+出图**

**1.5 zabbix介绍**

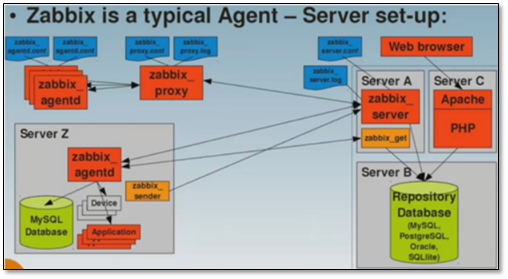
　　Zabbix 是由 Alexei Vladishev 开发的一种网络监视、管理系统，基于 Server-Client 架构。可用于监视各种网络服务、服务器和网络机器等状态。

　　使用各种 Database-end 如 MySQL, PostgreSQL, SQLite, Oracle 或 IBM DB2 储存资料。Server 端基于 C语言、Web 管理端 frontend 则是基于 PHP 所制作的。Zabbix 可以使用多种方式监视。可以只使用 Simple Check 不需要安装 Client 端，亦可基于 SMTP 或 HTTP ... 各种协定做死活监视。

　　在客户端如 UNIX, Windows 中安装 Zabbix Agent 之后，可监视 CPU Load、网络使用状况、硬盘容量等各种状态。而就算没有安装 Agent 在监视对象中，Zabbix 也可以经由 SNMP、TCP、ICMP、利用 IPMI、SSH、telnet 对目标进行监视。

另外，Zabbix 包含 XMPP 等各种 Item 警示功能。

**1.5.1 zabbix的组成**



zabbix官网: https://www.zabbix.com

zabbix 主要由2部分构成 zabbix server和 zabbix agent

zabbix proxy是用来管理其他的agent，作为代理

**1.5.2 zabbix监控范畴**

　　²  硬件监控 ：Zabbix IPMI Interface

　　²  系统监控 ：Zabbix Agent Interface

　　²  Java 监控：ZabbixJMX Interface

　　²  网络设备监抟：Zabbix SNMP Interface

　　²  应用服务监控：Zabbix Agent UserParameter

　　²  MySQL 数据库监控：percona-monitoring-pldlgins

　　²  URL监控：Zabbix Web监控

**第2章 安装zabbix**

**2.1 环境检查**

[root@m01 ~]# cat /etc/redhat-release

CentOS Linux release 7.4.1708 (Core)

[root@m01 ~]# uname -r

3.10.0-693.el7.x86\_64

[root@m01 ~]# getenforce

Disabled

[root@m01 ~]# systemctl status firewalld.service

● firewalld.service - firewalld - dynamic firewall daemon

Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)

Active: inactive (dead)

Docs: man:firewalld(1)

**2.2 安装zabbix过程**

**2.2.1 安装方式选择**

　　编译安装 （服务较多，环境复杂）

　　yum安装（干净环境）

　　使用yum 需要镜像yum源 http://www.cnblogs.com/clsn/p/7866643.html

**2.2.2 服务端快速安装脚本**

#!/bin/bash

#clsn

#设置解析 注意：网络条件较好时，可以不用自建yum源

# echo '10.0.0.1 mirrors.aliyuncs.com mirrors.aliyun.com repo.zabbix.com' >> /etc/hosts

#安装zabbix源、aliyun YUM源

curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

curl -o /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

#安装zabbix

yum install -y zabbix-server-mysql zabbix-web-mysql

#安装启动 mariadb数据库

yum install -y mariadb-server

systemctl start mariadb.service

#创建数据库

mysql -e 'create database zabbix character set utf8 collate utf8\_bin;'

mysql -e 'grant all privileges on zabbix.\* to zabbix@localhost identified by "zabbix";'

#导入数据

zcat /usr/share/doc/zabbix-server-mysql-3.0.13/create.sql.gz|mysql -uzabbix -pzabbix zabbix

#配置zabbixserver连接mysql

sed -i.ori '115a DBPassword=zabbix' /etc/zabbix/zabbix\_server.conf

#添加时区

sed -i.ori '18a php\_value date.timezone Asia/Shanghai' /etc/httpd/conf.d/zabbix.conf

#解决中文乱码

yum -y install wqy-microhei-fonts

\cp /usr/share/fonts/wqy-microhei/wqy-microhei.ttc /usr/share/fonts/dejavu/DejaVuSans.ttf

#启动服务

systemctl start zabbix-server

systemctl start httpd

#写入开机自启动

chmod +x /etc/rc.d/rc.local

cat >>/etc/rc.d/rc.local<<EOF

systemctl start mariadb.service

systemctl start httpd

systemctl start zabbix-server

EOF

#输出信息

echo "浏览器访问 http://`hostname -I|awk '{print $1}'`/zabbix"

**2.2.3 客户端快速部署脚本**

#!/bin/bash

#clsn

#设置解析

echo '10.0.0.1 mirrors.aliyuncs.com mirrors.aliyun.com repo.zabbix.com' >> /etc/hosts

#安装zabbix源、aliyu nYUM源

curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

curl -o /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

#安装zabbix客户端

yum install zabbix-agent -y

sed -i.ori 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_agentd.conf

systemctl start zabbix-agent.service

#写入开机自启动

chmod +x /etc/rc.d/rc.local

cat >>/etc/rc.d/rc.local<<EOF

systemctl start zabbix-agent.service

EOF

**2.3 检测连通性**

**2.3.1 服务端安装zabbix-get检测工具**

yum install zabbix-get

**2.3.2 在服务端进行测试**

注意：只能在服务端进行测试

zabbix\_get -s 172.16.1.61 -p 10050 -k "system.cpu.load[all,avg1]"

zabbix\_get -s 172.16.1.21 -p 10050 -k "system.cpu.load[all,avg1]"

***测试结果***

[root@m01 ~]# zabbix\_get -s 172.16.1.61 -p 10050 -k "system.cpu.load[all,avg1]"

0.000000

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "system.cpu.load[all,avg1]"

0.000000

**第3章 web界面操作**

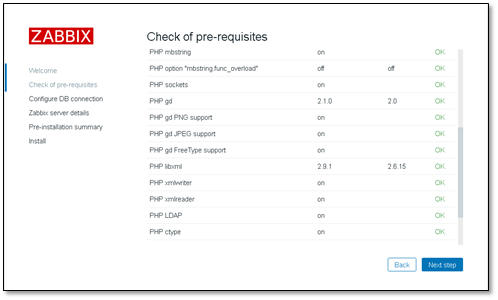
**3.1 zabbix的web安装**

**3.1.1 使用浏览器访问**

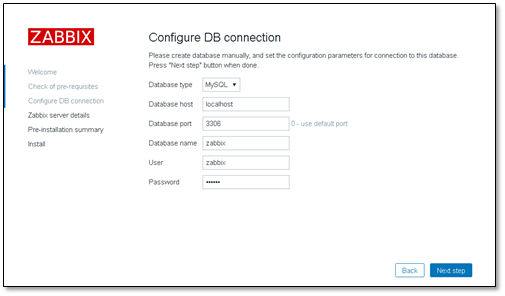
*http://10.0.0.61/zabbix/setup.php*



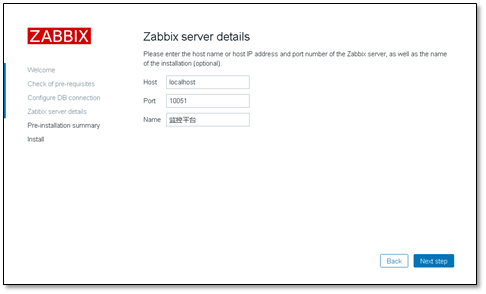
   在检测信息时，可查看具体的报错信息进行不同的解决



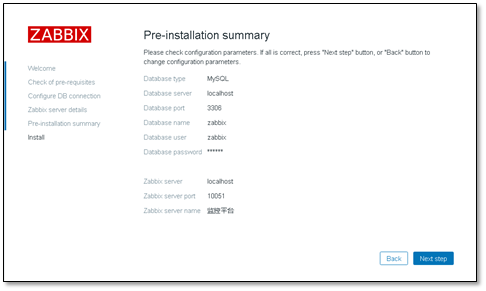
   选择mysql数据库，输入密码即可



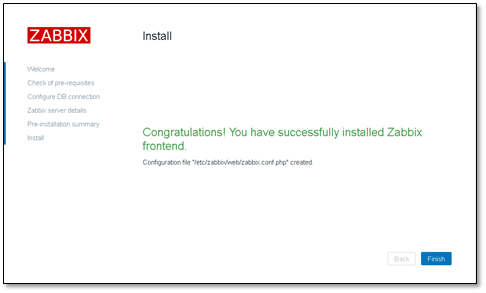
   host与port不需要修改，name自定义



确认信息,正确点击下一步



   安装完成、点击finsh



      进入登陆界面  账号**Admin**密码**zabbix   注意A大写**



**3.2 添加监控信息**

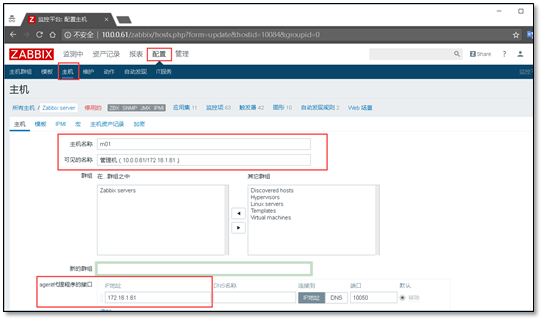
**3.2.1 修改监控管理机zabbix server**

配置 >> 主机



主机名称： 要与主机名相同，这是zabbix server程序用的

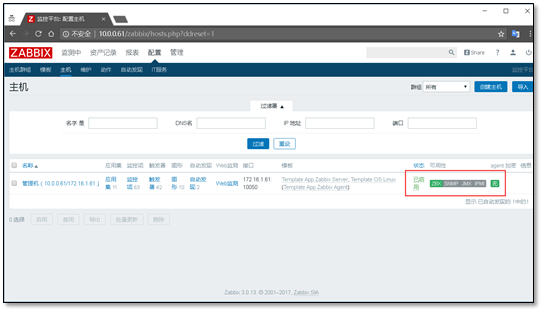
可见名称： 显示在zabbix网页上的，给我们看的



   修改后，要将下面的已启用要勾上

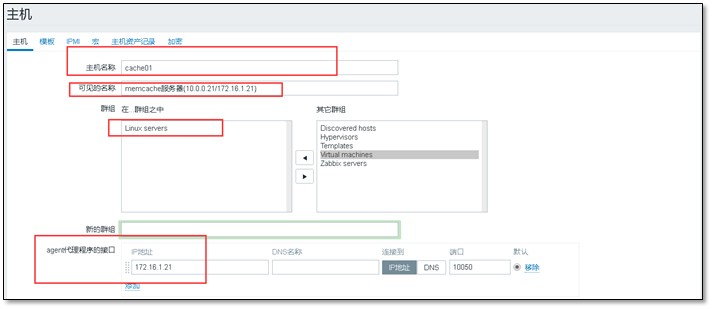


   添加完成就有了管理机的监控主机



**3.2.2 添加新的主机**

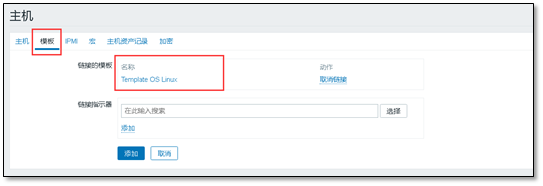
配置 >> 主机 >> 创建主机



注意勾选以启用



   然后添加模板，选择linux OS ，先点小添加，再点大添加。



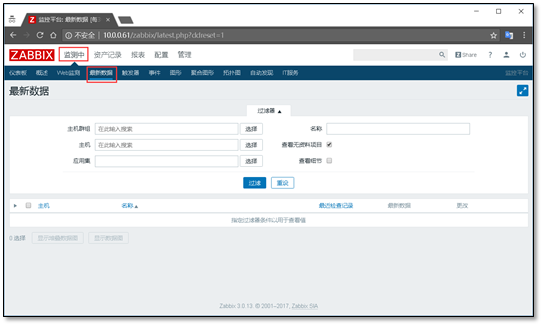
   添加完成，将会又两条监控主机信息



**3.2.3 查看监控内容**

检测中  >> 最新数据

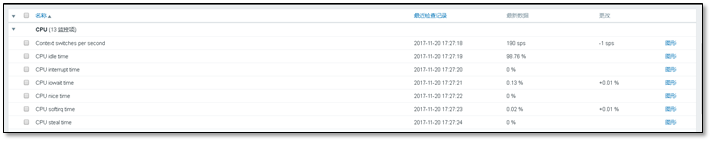
   在最新数据中需要筛选，



   输入ip或者名字都能够搜索出来



在下面就会列出所有的监控项



**3.2.4 查看图像**

检测中 >> 图形

   选择正确的主机。选择要查看的图形即可出图



**第4章 自定义监控与监控报警**

**4.1 自定义监控**

**4.1.1 说明**

zabbix自带模板Template OS Linux (Template App Zabbix Agent)提供CPU、内存、磁盘、网卡等常规监控，只要新加主机关联此模板，就可自动添加这些监控项。

**需求：**服务器登陆人数不能超过三人，超过三人报警

**4.1.2 预备知识**

自定义key能被server和agent认可

# 正确的key

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "system.uname"

Linux cache01 3.10.0-693.el7.x86\_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86\_64

# 没有登记的，自定义的key

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user"

ZBX\_NOTSUPPORTED: Unsupported item key.

# 写错的key

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "system.uname1"

ZBX\_NOTSUPPORTED: Unsupported item key.

**4.2 实现自定义监控**

**4.2.1 自定义语法**

UserParameter=<key>,<shell command>

UserParameter=login-user,who|wc -l

UserParameter=login-user,/bin/sh /server/scripts/login.sh

**4.2.2 agent注册**

[root@cache01 ~]# cd /etc/zabbix/zabbix\_agentd.d/

[root@cache01 zabbix\_agentd.d]# vim userparameter\_login.conf

UserParameter=login-user,who|wc -l

UserParameter=login-user2,who|wc -l

UserParameter=login-user3,who|wc -l

**注意：**key名字要唯一，多个key以行为分割

# 修改完成后重启服务

[root@cache01 zabbix\_agentd.d]# systemctl restart zabbix-agent.service

   在server端进行get测试

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user"

3

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user2"

3

[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user3"

3

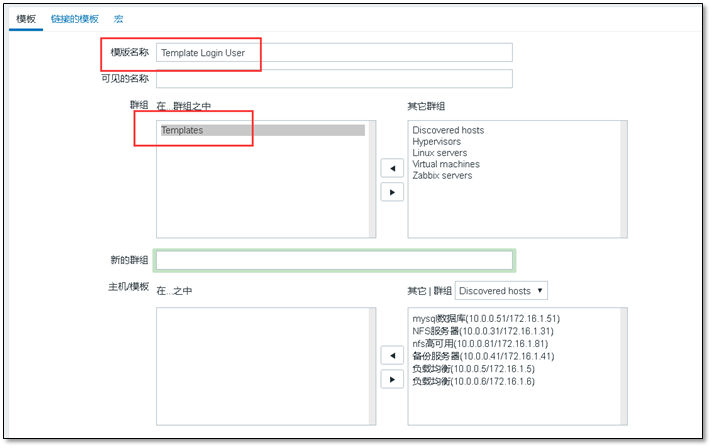
[root@m01 ~]# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user4"

ZBX\_NOTSUPPORTED: Unsupported item key.

**4.2.3 在server端注册(web操作)**

**①   创建模板**

配置 >> 模板 >> 创建模板



点击添加，即可创建出来模板



   查看创建出来的模板。↑

**②   创建应用集**

应用集类似(目录/文件夹)，其作用是给监控项分类。

点击 应用集 >> 创建应用集



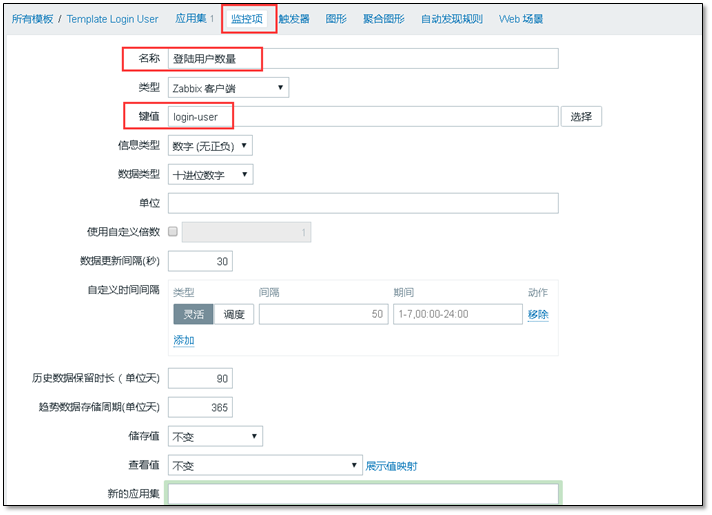
      自定义应用集的名称，然后点击添加

**③   创建监控项**

监控项 >> 创建监控项



**键值** -- key,即前面出创建的login-user。



   注意：创建监控项的时候，注意选择上应用集，即之前创建的安全。



**④   创建触发器**

触发器的作用：当监控项获取到的值达到一定条件时就触发报警

*(根据需求创建)*

触发器 >> 创建触发器

创建触发器，自定义名称，该名称是报警时显示的名称。

   表达式，点击右边的添加，选择**表达式**。

   严重性自定义。

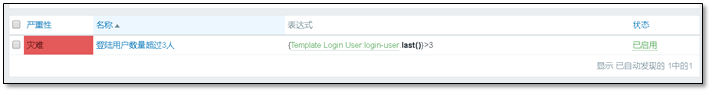


**表达式的定义 ↓ ，选择**之前创建的监控项，

最新的T值为当前获取到的值。



   添加完成，能够在触发器中看到添加的情况



**⑤   创建图形**

以图形的方式展示出来监控信息

图形 >> 创建图形

名称自定义，关联上监控项。



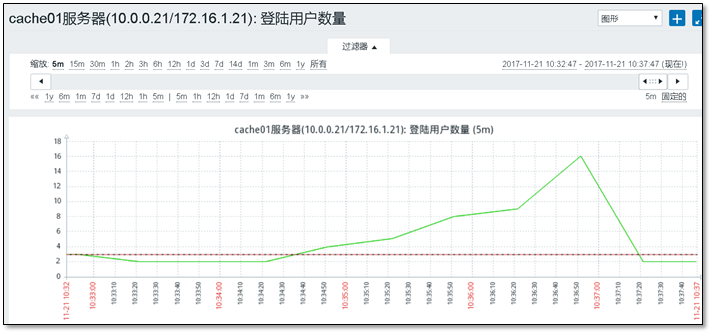
**⑥   主机关联模板**

配置 >> 主机

   一个主机可以关联多个模板



**4.2.4 查看监控的图形**



**4.3 监控报警**

**4.3.1 第三方报警平台**

http://www.**onealert**.com

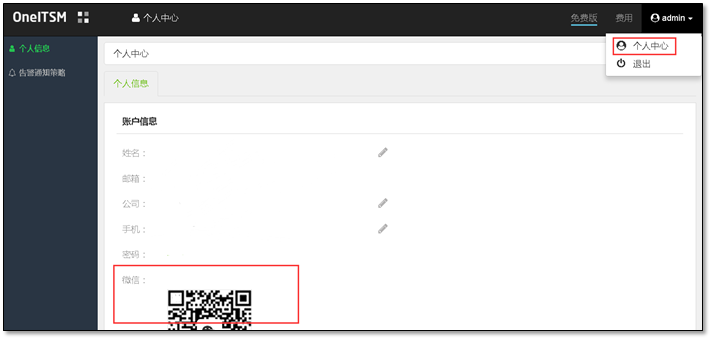
   　 通过 OneAlert 提供的通知分派与排班策略，以及全方位的短信、微信、QQ、电话提醒服务，您可以在最合适的时间，将最重要的信息推送给最合适的人员。

**4.3.2 onealert配置**

添加应用，注意添加的是zabbix



   实现微信报警需要关注微信公众号即可。



**4.3.3 安装 onealert Agent**

1.切换到zabbix脚本目录(如何查看zabbix脚本目录)：

cd /usr/local/zabbix-server/share/zabbix/alertscripts

#查看zabbix脚本目录

vi /etc/zabbix/zabbix\_server.conf

查看AlertScriptsPath

2.获取OneITSM agent包：

wget http://www.onealert.com/agent/release/oneitsm\_zabbix\_release-1.0.1.tar.gz

3.解压、安装。

tar -zxf oneitsm\_zabbix\_release-1.0.1.tar.gz

cd oneitsm/bin

bash install.sh --#个人生成的key

注：在安装过程中根据安装提示，**输入zabbix管理地址、管理员用户名、密码**。

Zabbix管理地址: http://10.0.0.61/zabbix/

Zabbix管理员账号: Admin

Zabbix管理员密码:

4.当提示"安装成功"时表示安装成功!

验证告警集成

产生新的zabbix告警(problem),动作状态为“已送达”表示集成成功。

**4.3.1 如何删除onealert Agent**

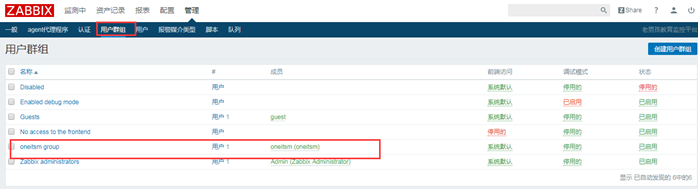
①  删除报警媒介类型中的脚本



②  删除创建的用户



③  删除用户群组



④  删除创建的动作



**4.3.2 触发器响应，发送报警信息**



   在微信和邮件中，均能收到报警信息。



**注意：当状态改变的时候才会发邮件**

   　　 好-->坏

　　    坏-->好

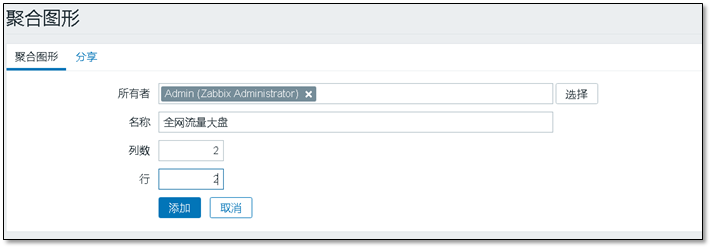
**4.4 监控可视化**

**4.4.1 聚合图形**

最新数据 >> 图形



   自定义名称



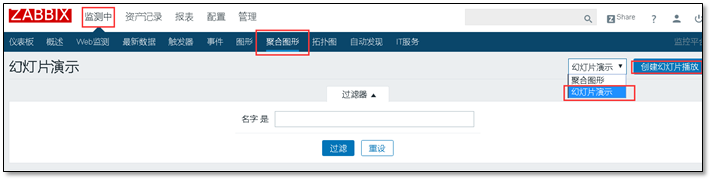
   点击聚合图形的名称，进行更改，添加要显示的图形即可。



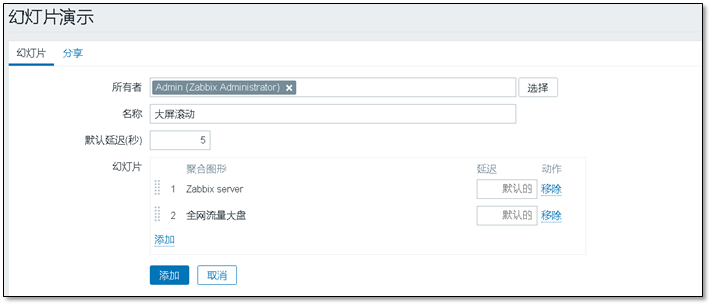
**4.4.2 幻灯片**

添加幻灯片

监测中 >> 复合图形 >> 幻灯片演示



   创建幻灯片，名称自定，选择要显示的



   幻灯片根据设定的时间自动播放

**4.5 模板的共享**

**4.5.1 主机共享**

在主机页打开，全选后点击导出



   导入



**4.5.2 模板共享**

**https://github.com/zhangyao8/zabbix-community-repos**



**第5章 监控全网服务器**

**5.1 需求说明**

实际需求：公司已经有了100台服务器，现在需要使用zabbix全部监控起来。

**5.2 规划方案**

常规监控：cpu，内存，磁盘，网卡  问题：怎样快速添加100台机器

   　　方法1：使用克隆的方式

 　　  方法2：自动注册和自动发现

 　　  方法3：调用zabbix api接口  curl 、python

        　　  开发自己的运维平台兼容zabbix的通道

**服务监控，url监控等特殊监控**：自定义监控

**5.2.1 api接口使用（*curl*）**

curl -i -X POST -H 'Content-Type:application/json' -d'{"jsonrpc": "2.0","method":"user.login","params":{"user":"Admin","password":"zabbix"},"auth": null,"id":0}' "http://10.0.0.61/zabbix/api\_jsonrpc.php"

curl -i -X POST -H 'Content-Type:application/json' -d'

{

"jsonrpc": "2.0",

"method": "host.get",

"params": {

"output": [

"hostid",

"host"

],

"selectInterfaces": [

"interfaceid",

"ip"

]

},

"id": 2,

"auth": "6a450a8fc3dce71fd310cfe338746578"

}' "http://10.0.0.61/zabbix/api\_jsonrpc.php"

**5.3 具体实施规划**

**5.3.1 硬件、系统、网络监控**

　　所有集群节点（所有虚拟机）都监控上

　　交换机，路由器监控（简单方法：换成端口对应服务器网卡流量监控；标准方法：监控交换机的网卡）

　　snmp监控

**5.3.2 应用服务监控**

1. 监控备份服务器，简单方法是监控rsync端口，如果有其他更佳方案可以说明；

方法1：监控873端口net.tcp.port[,873]

方法2：模拟推送拉取文件

2. 监控NFS服务器，使用监控NFS进程来判断NFS服务器正常，如果有其他更佳方案可以说明；

方法1：端口（通过111的rpc端口获取nfs端口） net.tcp.port[,111]

方法2：showmount -e ip|wc -l

3. 监控MySQL服务器，简单方法监控mysql的3306端口，或者使用zabbix提供的Mysql模板，如果有其他更佳方案可以说明；

方法1：端口（通过3306的mysql端口） net.tcp.port[,3306]

方法2：mysql远程登录

方法3：使用zabbix agent自带的模板及key

4. 监控2台web服务器，简单方法监控80端口，如果有其他更佳方案可以说明；

方法1：端口（通过80的web端口） net.tcp.port[,80]

方法2：看网页状态码、返回内容==zabbix 自带WEB检测

5. 监控URL地址来更精确的监控我们的网站运行正常；

使用zabbix自带的监控Web监测 进行监控

6. 监控反向代理服务器，PPTP服务器等你在期中架构部署的服务。

nginx，pptp

ntp 端口udp 123

7. 监控Nginx的7种连接状态。

自定义监控

**5.3.3 监控服务通用方法**

　　1. 监控端口 netstat ss lsof  ==》 wc -l

　　2. 监控进程 ps -ef|grep 进程|wc -l  试运行一下

　　3. 模拟客户端的使用方式监控服务端

  　　    web  ==》 curl

     　　 mysql ==》 select insert

   　　   memcache ==》 set再get

**5.4 实施全网监控**

*安装客户端脚本，for centos6*

#!/bin/bash

#设置解析

# echo '10.0.0.1 mirrors.aliyuncs.com mirrors.aliyun.com repo.zabbix.com' >> /etc/hosts

#安装zabbix源、aliyu nYUM源

curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-6.repo

curl -o /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/6/x86\_64/zabbix-release-3.0-1.el6.noarch.rpm

yum clean all

yum clean all

#安装zabbix客户端

yum install zabbix-agent -y

sed -i.ori 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_agentd.conf

/etc/init.d/zabbix-agent start

#写入开机自启动

chmod +x /etc/rc.d/rc.local

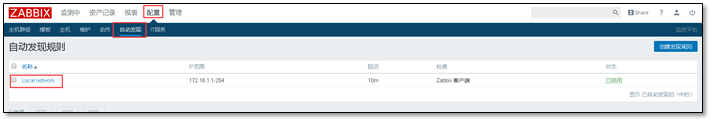
cat >>/etc/rc.d/rc.local<<EOF

/etc/init.d/zabbix-agent start

EOF

**5.4.1 使用自动发现规则**

添加自动发现规则



   创建发现动作



   查看自动发现的机器。



**5.4.2 监控备份服务器**

利用系统自带键值进行监控*net.tcp.listen[port]* 创建新的模板



在服务端进行测试

[root@m01 ~]# zabbix\_get -s 172.16.1.41 -p 10050 -k "net.tcp.listen[873]"

1

# 1为端口在监听 0为端口未监听

将模板添加到主机



**5.4.3 监控NFS服务器**

创建nfs监控模板

使用*proc.num[<name>,<user>,<state>,<cmdline>]*  键值，检测nfs进程的数量



在服务端进行测试

[root@m01 ~]# zabbix\_get -s 172.16.1.31 -p 10050 -k "proc.num[,,,rpc]"

5

[root@m01 ~]# zabbix\_get -s 172.16.1.31 -p 10050 -k "proc.num[nfsd,,,]

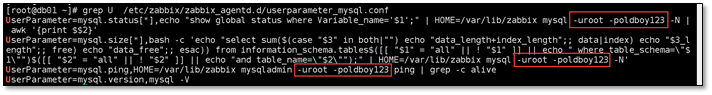
8

将模板绑定到主机



**5.4.4 监控MySQL服务器**

将自带的mysqlkey值加上mysql的账户密码，否则不能获取到数据。



使用系统自带模板  net.tcp.port[<ip>,port] 利用自带的监控端口键值进行监控



添加新的mysql监控项端口



[root@m01 ~]# zabbix\_get -s 172.16.1.51 -p 10050 -k "net.tcp.port[,3306]"

1

#检查是否能建立 TCP 连接到指定端口。返回 0 - 不能连接；1 - 可以连接

将模板关联到主机



**5.4.5 监控web服务器**

创建监控模板 监控 nginx服务与 80 端口

proc.num[<name>,<user>,<state>,<cmdline>] 进程数。返回整数

net.tcp.port[<ip>,port] 检查是否能建立 TCP 连接到指定端口。返回 0 - 不能连接；1 - 可以连接



[root@m01 ~]# zabbix\_get -s 172.16.1.8 -p 10050 -k "proc.num[,,,nginx]"

2

[root@m01 ~]# zabbix\_get -s 172.16.1.8 -p 10050 -k "net.tcp.port[,80]"

1

将模板关联到主机



**5.4.6 监控URL地址**

创建监测页面

echo ok >> /application/nginx/html/www/check.html

测试监控面页

[root@web03 ~]# for ip in 7 8 9 ;do curl 10.0.0.$ip/check.html ;done

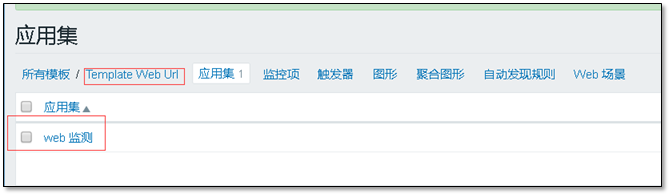
ok

ok

ok

创建web监测模板

*创建应用集*



*创建Web场景*



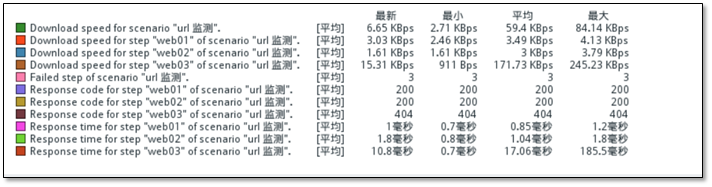
*创建图形*

[](http://10.0.0.61/zabbix/chart2.php?graphid=668&period=60&stime=20191122164643&updateProfile=1&profileIdx=web.screens&profileIdx2=668&width=1052&sid=8a1fe32c9614ae35&screenid=&curtime=1511340464305)

将模板关联到主机



监测结果



**5.4.7 监控反向代理服务器**

创建自定义key

[root@lb01 ~]# cat /etc/zabbix/zabbix\_agentd.d/userparameter\_nk.conf

UserParameter=keep-ip,ip a |grep 10.0.0.3|wc -l

在服务端测试

[root@m01 ~]# zabbix\_get -s 172.16.1.5 -p 10050 -k "keep-ip"

1

[root@m01 ~]# zabbix\_get -s 172.16.1.6 -p 10050 -k "keep-ip"

0

在web界面添加模板



将模板关联到主机



**5.4.8 监控Nginx的7种连接状态**

nginx服务器显示status

……

location /status {

stub\_status on;

access\_log off;

}

……

[root@web01 ~]# for ip in 7 8 9 ;do curl 172.16.1.$ip/status ;done

Active connections: 1

server accepts handled requests

73 73 69

Reading: 0 Writing: 1 Waiting: 0

Active connections: 1

server accepts handled requests

134 134 127

Reading: 0 Writing: 1 Waiting: 0

Active connections: 1

server accepts handled requests

7 7 7

Reading: 0 Writing: 1 Waiting: 0

在nginx服务器上添加key

cat >/etc/zabbix/zabbix\_agentd.d/userparameter\_nginx\_status.conf <<'EOF'

UserParameter=nginx\_active,curl -s 127.0.0.1/status|awk '/Active/ {print $NF}'

UserParameter=nginx\_accepts,curl -s 127.0.0.1/status|awk 'NR==3 {print $1}'

UserParameter=nginx\_handled,curl -s 127.0.0.1/status|awk 'NR==3 {print $2}'

UserParameter=nginx\_requests,curl -s 127.0.0.1/status|awk 'NR==3 {print $3}'

UserParameter=nginx\_reading,curl -s 127.0.0.1/status|awk 'NR==4 {print $2}'

UserParameter=nginx\_writing,curl -s 127.0.0.1/status|awk 'NR==4 {print $4}'

UserParameter=nginx\_waiting,curl -s 127.0.0.1/status|awk 'NR==4 {print $6}'

EOF

服务端测试

[root@m01 ~]# zabbix\_get -s 172.16.1.7 -p 10050 -k "nginx\_waiting"

0

[root@m01 ~]# zabbix\_get -s 172.16.1.8 -p 10050 -k "nginx\_waiting"

0

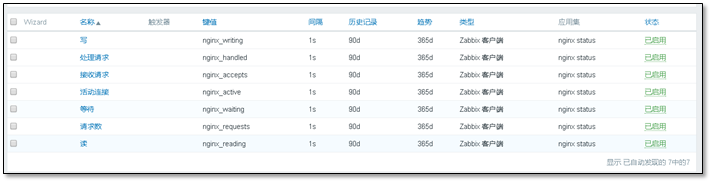
[root@m01 ~]# zabbix\_get -s 172.16.1.9 -p 10050 -k "nginx\_waiting"

0

在zabbix-web上添加



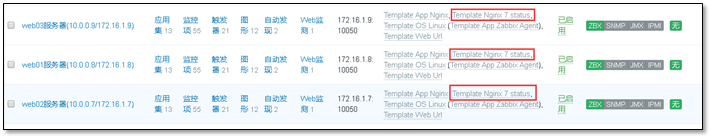
监控项



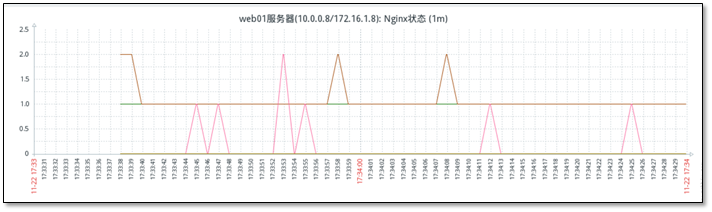
添加图形

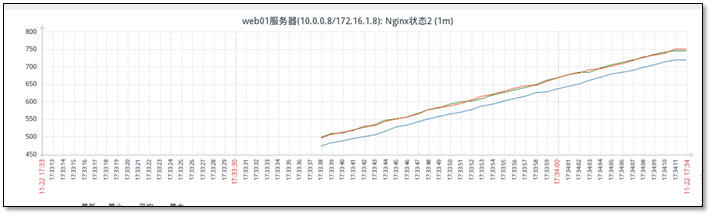


将模板关联到主机



查看添加的图形





**第6章 自动发现与自动注册**

**6.1 自动注册与自动注册**

**6.1.1 简介**

***自动发现：***

zabbix Server主动发现所有客户端，然后将客户端登记自己的小本本上，缺点zabbix server压力山大（网段大，客户端多），时间消耗多。

***自动注册：***

zabbix agent主动到zabbix Server上报到，登记；缺点agent有可能找不到Server（配置出错）

**6.1.2 两种模式**

被动模式：默认 agent被server抓取数据 （都是在agent的立场上说）

主动模式：agent主动将数据发到server端 （都是在agent的立场上说）

**注意： 两种模式都是在agent上进行配置**

**zabbix 的使用要在hosts文件中预先做好主机名的解析**

**6.2 自动发现--被动模式**

　第一个里程碑：完成之前的安装

zabbix Server安装完毕

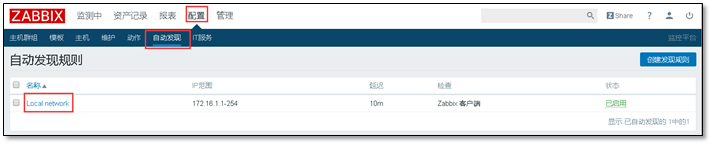
   第二个里程碑：配置agent客户端

zabbix agent安装完毕，注意配置Server=172.16.1.61

   第三个里程碑：在web界面上进行配置

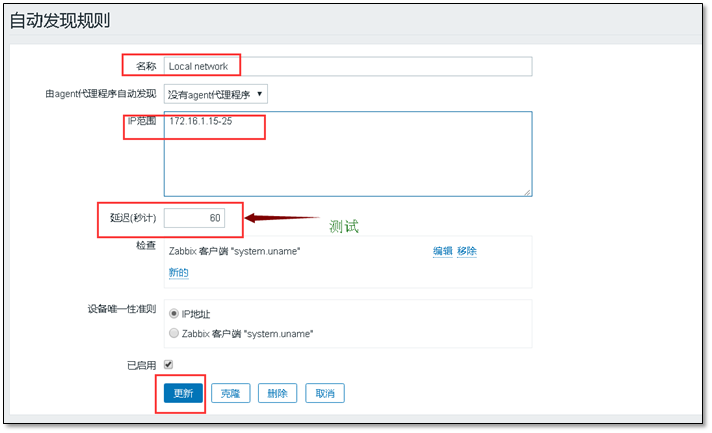
web界面：配置 >> 自动发现 >> Local network

使用自带的自动发现规则（进行修改）即可



在ip范围内输入ip，注意格式；

延迟在实际的生产环境中要大一些，实验环境可以小一些

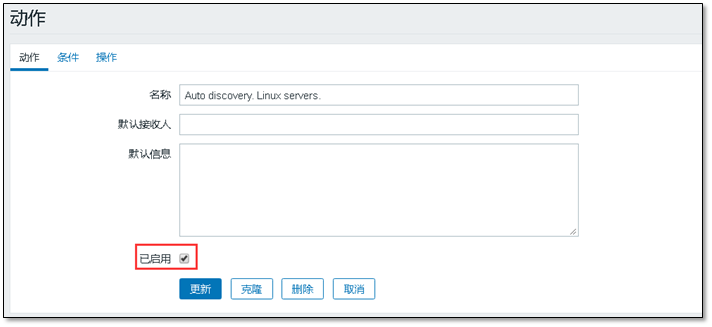


**创建发现动作**

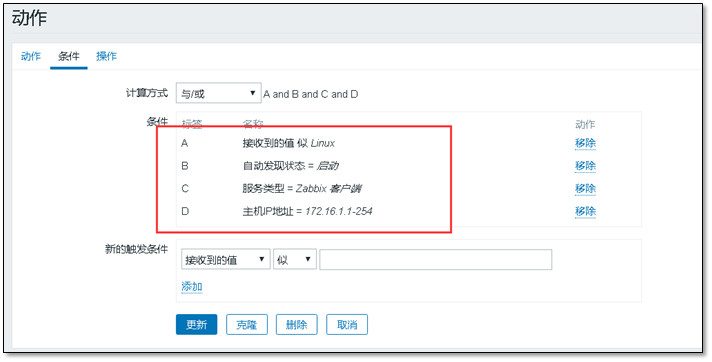
配置 >> 动作 >> Auto discovery. Linux servers.



①  配置动作

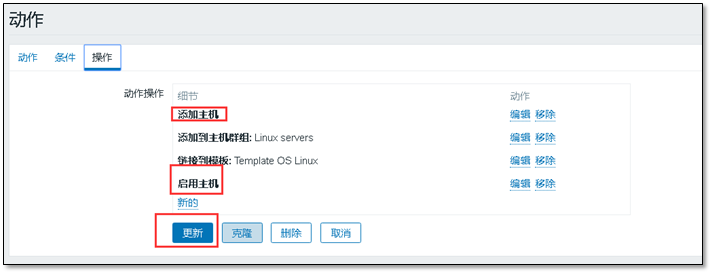


②  在条件中添加条件，让添加更准确



③  在操作中添加

a)  添加主机与启用主机



  　　  然后等待者客户端自动上门就好😏

**6.3 自动注册--主动模式**

　第一个里程碑：zabbix Server安装完毕 （完成）

zabbix Server安装完毕

   第二个里程碑：zabbix agent安装完毕，需要额外增加的配置

vim /etc/zabbix/zabbix\_agentd.conf

ServerActive=172.16.1.61

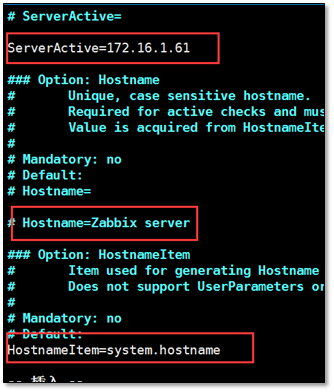
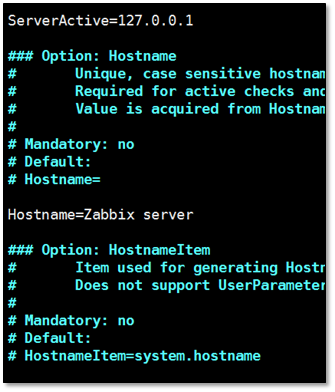
# Hostname=Zabbix server

HostnameItem=system.hostname

systemctl restart zabbix-agent.service

netstat -tunlp|grep zabbix

    源文件与修改后对比

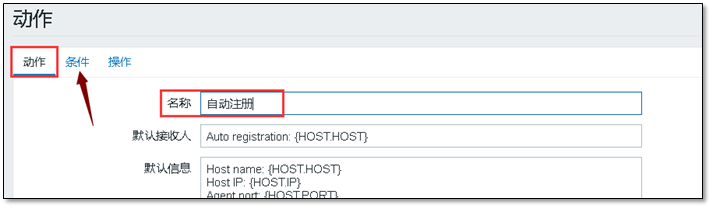


　   第三个里程碑：在web见面上进行配置

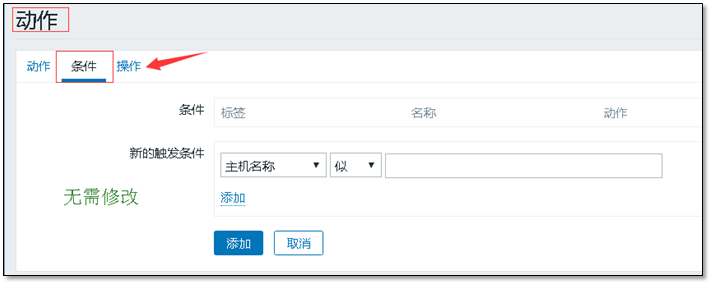
1 配置 >> 动作 >> 事件源(自动注册) >> 创建动作



    创建动作，添加名称即可

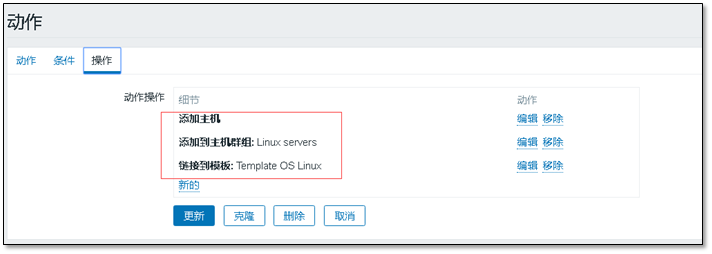


    条件中也无需修改



    在动作中添加动作

（添加主机、添加到主机群组、链接到模板）



    添加完动作后，等待就行了

    注意：重启客户端可以加速发现。但是在生产环境中勿用。



**第7章 分布式监控与SNMP监控**

**7.1 分布式监控**

**7.1.1 作用**

  　　分担压力，减轻负载

  　　多机房监控

　　zabbix Server  ===》  zabbix agent （只能同一个局域网监控）

***分担压力，降低负载***

zabbix Server ===》 zabbix proxy ===》zabbix agent1 agent2 agent3 。。。

172.16.1.61 172.16.1.21 172.16.1.0/24

===》 zabbix proxy ===》zabbix agent4 agent5 agent6 。。。

***多机房监控***

zabbix Server(北京) ==》 zabbix proxy（每个机房搭建） ==》 zabbix agent

122.71.240.233/172.16.1.61 122.71.241.11/172.16.2.21 172.16.2.0/24

**7.1.2 环境说明**

zabbix server m01

zabbix proxy cache01

zabbix agent cache01

**7.1.3 配置zabbix proxy**

**第一个里程碑**：配置zabbix yum源，并安装proxy

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

yum install zabbix-proxy-mysql -y

**第二个里程碑：安装数据库**

    zabbix  proxy也需要数据库，这个数据库不是用于存储监控数据的 只是用于存储配置信息

   #安装数据库

yum -y install mariadb-server

systemctl start mariadb.service

   #建立数据库

mysql

create database zabbix\_proxy character set utf8 collate utf8\_bin;

grant all privileges on zabbix\_proxy.\* to zabbix@'localhost' identified by 'zabbix';

exit

   #导入数据文件

zcat /usr/share/doc/zabbix-proxy-mysql-3.0.13/schema.sql.gz |mysql -uzabbix -pzabbix zabbix\_proxy

   #配置zabbix proxy 连接数据库

sed -i.ori '162a DBPassword=zabbix' /etc/zabbix/zabbix\_proxy.conf

sed -i 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_proxy.conf

sed -i 's#Hostname=Zabbix proxy#Hostname=cache01#' /etc/zabbix/zabbix\_proxy.conf

# Hostname 作为后面添加的代理程序名称，要保持一致

   #启动

systemctl restart zabbix-proxy.service

   #检查端口

[root@cache01 ~]# netstat -lntup |grep zabbix

tcp 0 0 0.0.0.0:10050 0.0.0.0:\* LISTEN 105762/zabbix\_agent

tcp 0 0 0.0.0.0:10051 0.0.0.0:\* LISTEN 85273/zabbix\_proxy

tcp6 0 0 :::10050 :::\* LISTEN 105762/zabbix\_agent

tcp6 0 0 :::10051 :::\* LISTEN 85273/zabbix\_proxy

**第三个里程碑：**修改agent配置指向 proxy

[root@cache01 ~]# grep ^Server /etc/zabbix/zabbix\_agentd.conf

Server=172.16.1.61

ServerActive=172.16.1.61

[root@cache01 ~]# sed -i 's#172.16.1.61#172.16.1.21#g' /etc/zabbix/zabbix\_agentd.conf

[root@cache01 ~]# grep ^Server /etc/zabbix/zabbix\_agentd.conf

Server=172.16.1.21

ServerActive=172.16.1.21

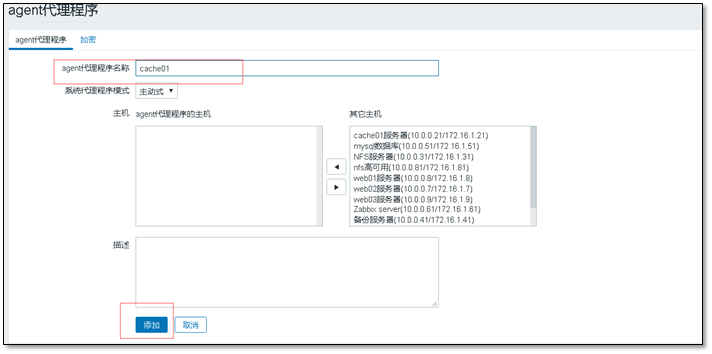
[root@cache01 ~]# systemctl restart zabbix-agent.service

**第四个里程碑：**web界面添加代理

    管理 >> agent代理程序 >> 创建代理



   代理程序名称要填写主机名



   稍等片刻就能在程序中出现代理



   在主机中能发现主机代理



**7.2 SNMP监控**

**7.2.1 使用范围**

　　无法安装agent  很多前辈的监控软件都可以监控各种设备  都是通过snmp监控

　　snmp simple network manager protocol 简单网络管理协议

  　 简单网络管理协议（SNMP），由一组网络管理的标准组成，包含一个应用层协议（application layer protocol）、数据库模型（database schema）和一组资源对象。该协议能够支持网络管理系统，用以监测连接到网络上的设备是否有任何引起管理上关注的情况。

**7.2.2 安装snmp程序**

yum -y install net-snmp net-snmp-utils

**7.2.3 配置snmp程序**

sed -i.ori '57a view systemview included .1' /etc/snmp/snmpd.conf

systemctl start snmpd.service

**7.2.4 测试snmp**

[root@m01 ~]# snmpwalk -v 2c -c public 127.0.0.1 sysname

SNMPv2-MIB::sysName.0 = STRING: m01

**说明：**

 　　   # snmpwalk 类似 zabbix\_get

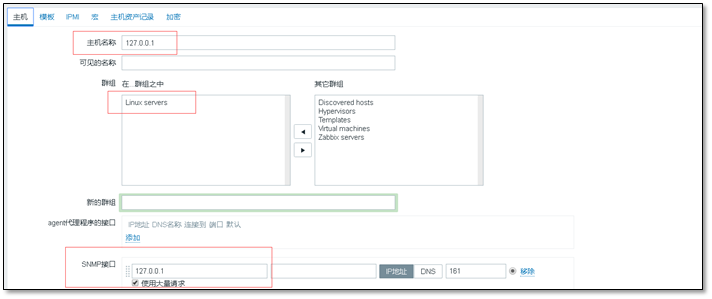
　　　# -v 2c  指定使用snmp协议的版本  snmp分为v1 v2 v3

　　　# -c public  指定暗号

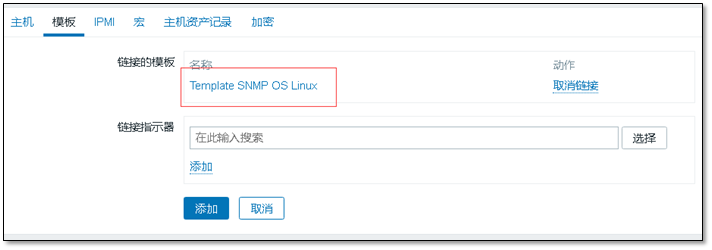
　　　# sysname  类似zabbix的key

**7.2.5 在web界面进行配置**

添加新的主机，注意使用snmp接口



选择模板，注意使用SNMP的模板



    添加完成就能够在主机中看到snmp监控对的主机



**7.2.6 附录**

##SNMP OID列表 监控需要用到的OID

http://www.ttlsa.com/monitor/snmp-oid/

cmdb 资源管理系统

# [Zabbix](https://www.cnblogs.com/bubu99/p/12275447.html)

**为什么要监控**

* 运维的职责  
  1.保障企业数据的安全可靠。  
  2.为客户提供7\*24小时服务。  
  3.不断提升用户的体验。
* 在关键时刻，提前提醒我们服务器要出问题了
* 当出问题之后，可以便于找到问题的根源

拿到公司服务器的管理权之后，

第一件事，备份

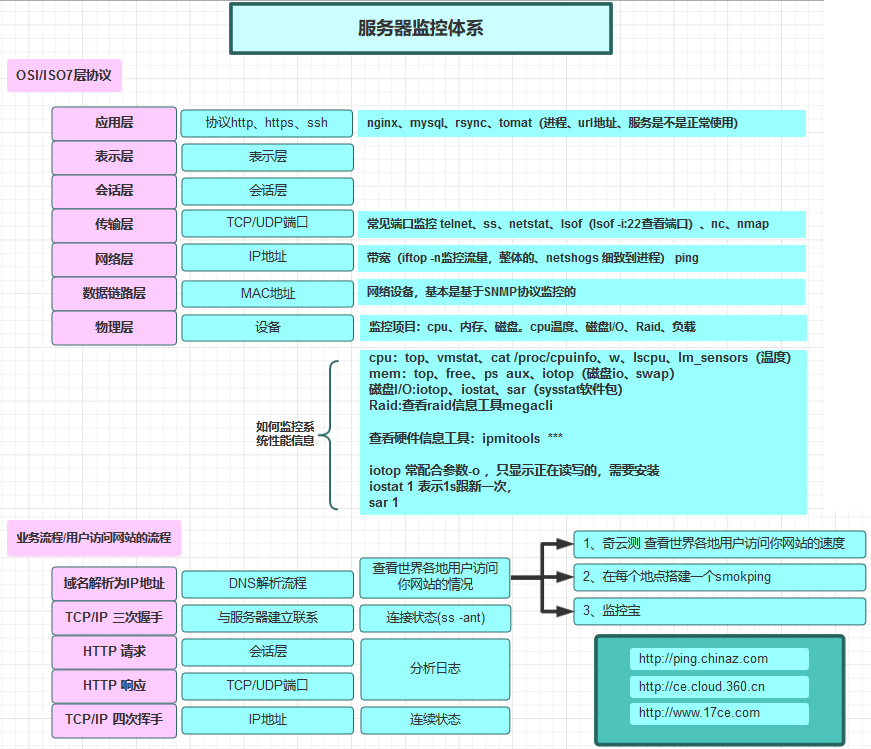
第二件事，监控

数据有故障了，服务有问题了，一些内容挂掉了，比如web服务器，或者是硬件的服务器，raid，磁盘有故障了也会报警，用户访问网站，感觉慢了，也可以获得相应的提示

这是一个非常重要的部分

**你们公司监控是怎么做的-面试题**

* OSI/ISO7层协议进行回答
* 业务流程、用户访问网站的流程进行补充



**监控工具**

mrtg    流量监控出图  
nagios  监控  
cacti   流量监控出图  
zabbix  监控+出图，所有监控范畴，都可以整合到Zabbix中。

* 硬件监控：Zabbix IPMI Interface
* 系统监控：Zabbix Agent Interface
* Java监控：Zabbix JMX Interface
* 网络设备监控：Zabbix SNMP Interface
* 应用服务监控：Zabbix Agent UserParameter
* MySQL数据库监控：percona-monitoring-plulgins
* URL监控：Zabbix Web 监控

**Zabbix简介**

zabbix是一个基于WEB界面的，提供分布式系统监视以及网络监视功能的企业级的开源解决方案

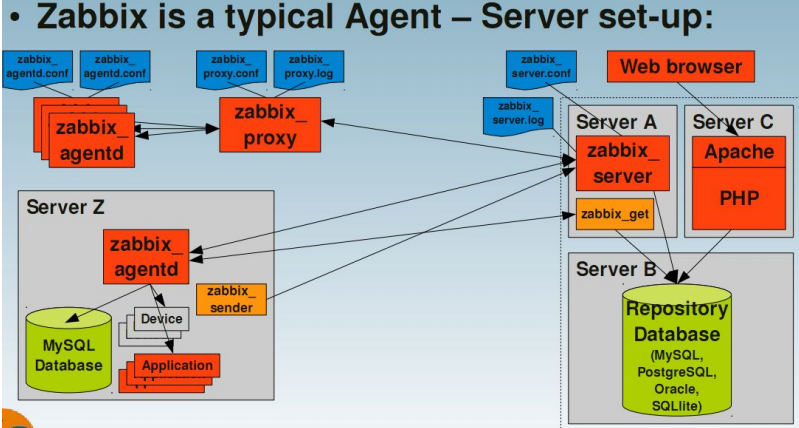
[官网：http://www.zabbix.com](http://www.zabbix.com/)

已经有20年历史，

web界面是基于PHP的，所以需要准备LNMP或者LAMP的环境

软件是c语言的

zabbix的架构图



 zabbix\_proxy 在大型架构中作为代理使用，将分部服务器的情况汇总发送到总部

**zabbix安装以及部署**

**先做环境检查**

从CentOS7模板机全新克隆一台新主机

IP规划为10.0.0.61/172.16.1.61，主机名为m01

注册账号：[http:](http://c.onealert.com/console/ucid/regist.jsp" \t "_blank)[//c.onealert.com/console//ucid/regist.jsp](http://c.onealert.com/console/ucid/regist.jsp)

账号用于接受报警

**安装zabbix服务端**

本次使用yum安装，编译安装留为作业

官网---download

官方安装说明中可以找到下面这些信息和安装步骤

版本号中的LTS指的是长时间维护版本

或者找zabbix官方源repo.zabbix.com----找到有release的，复制链接地址

# 安装Zabbix官方源

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

yum makecahe fast # 快点缓存

# 安装zabbix，httpd，php

yum install zabbix-server-mysql zabbix-web-mysql httpd php -y

**yum安装数据库**

数据库因为用的是centos7，MySQL是正常装不上的，需要配置MySQL源或者二进制安装

所以，偷懒安装mariadb（本质上基本和mysql是一样的）

# 安装mariadb

yum -y install mariadb-server

systemctl start mariadb.service

# 创建数据库，权限

mysql

create database zabbix character set utf8 collate utf8\_bin;

grant all privileges on zabbix.\* to zabbix@localhost identified by 'zabbix';  
创建用户的时候，不用flush跟新信息，跟新只在修改用户信息的时候

exit

# 导入数据库sql文件，官方安装说明中可以找到这些信息和安装步骤

zcat /usr/share/doc/zabbix-server-mysql-3.0.25/create.sql.gz|mysql -uzabbix -pzabbix zabbix

zcat 命令是显示gz压缩包内容的软件

请根据自己的zabbix软件版本 修改zcat的文件目录

**配置zabbix server**

# 配置zabbix Server连接mysql

sed -i.ori '115a DBPassword=zabbix' /etc/zabbix/zabbix\_server.conf

# 修改apache-php配置，因为我们是自动安装，他会自动创建一个虚拟主机  
# 我们修改下配置文件，添加时区，否则后面打开web页面安装zabbix的时候会报错

sed -i.ori '18a php\_value date.timezone Asia/Shanghai' /etc/httpd/conf.d/zabbix.conf

**启动zabbix server**

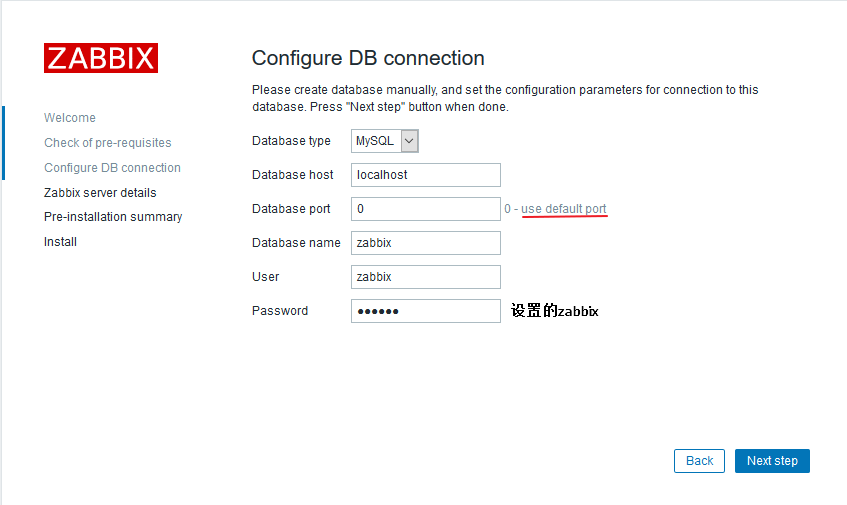
systemctl start zabbix-server # 启动zabbix

systemctl start httpd # 启动apache

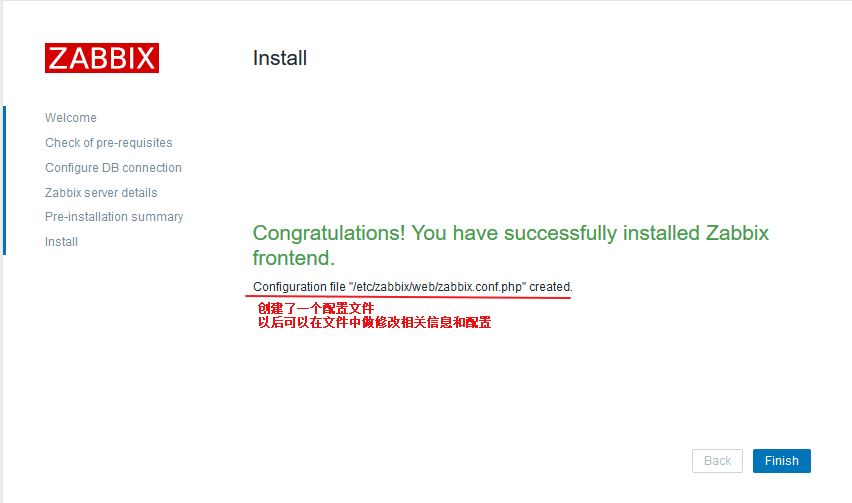
**网页配置**

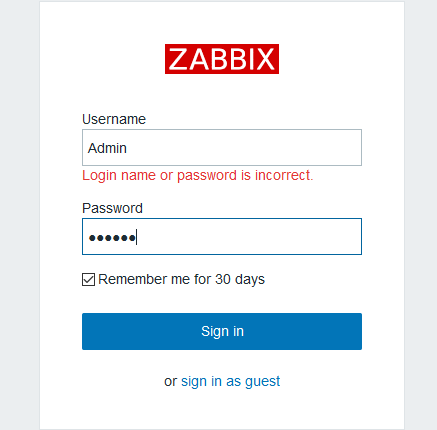
浏览器中访问10.0.0.61/zabbix 。记得用自己的IP

进入页面按照指令操作。









默认的账号Admin密码zabbix

进入主页，单击右上角的小人，修改显示语言和密码

**zabbix开始监控状态-安装配置zabbix-agent**

zabbix客户端就是zabbix agent

国内源：搜索

选择yum源的时候注意版本号，有的是centos6有的是7

可以直接安装agent，但是有一些依赖可能装不上

选择release最为稳妥

所有需要被监控的服务器都要执行下面步骤，下面以在m01和cache01安装客户端为例：

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

yum install zabbix-agent

sed -i.ori 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_agentd.conf

systemctl start zabbix-agent.service

**客户端测试**

此时，还不能在web页面张查看监控状态

需要先在客户端测试确认一下数据，需要用到zabbix-get

# 只能在服务端m01操作，需要使用-s执行相应客户端IP

yum install zabbix-get -y

# agent.ping 表示是否在线 1表示在线  
[root@m01 doc]# zabbix\_get -s 172.16.1.61 -p 10050 -k 'agent.ping'  
1

zabbix\_get -s 172.16.1.61 -p 10050 -k "system.cpu.load[all,avg1]"

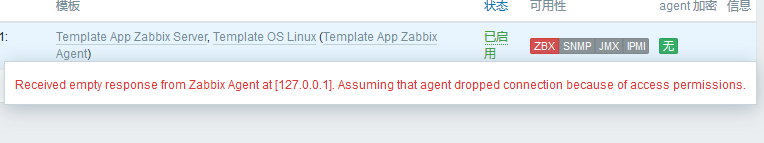
zabbix\_get -s 172.16.1.21 -p 10050 -k "system.cpu.load[all,avg1]"

web页面上怎么看？

配置----主机----将 zabbix server 启用--- 当可用性 都变成绿色，就表名监控上线

等待......如何加快（ping 主机名 可以解析）

等了很久，看一下日志吧...（/var/log/zabbix/zabbix\_server.log）



我们等来了一个错误，意思是，客户端无法访问服务端，是监控IP设置的问题

点击Zabbix server 将 agent代理程序的接口 这一项下面的 ip地址改为172.16.1.61，跟新保存

过一会就变成绿色了，就可以查看监控中---最新数据、图形等

因为中文字体的原因，有些地方会乱码，改成英文就行了

解决乱码问题，单独安装字体就可以了，不断地尝试，发现某个字体可用

yum -y install wqy-microhei-fonts  
将可用字体，复制到zabbix下

\cp /usr/share/fonts/wqy-microhei/wqy-microhei.ttc /usr/share/fonts/dejavu/DejaVuSans.ttf

**添加一台被监控服务器主机**

现在web01服务器上安装zabbix agent 客户端

清华或者阿里软件镜像站------找到zabbix----找到版本-----选择rhel----选择6（web01是centos6的系统）----选择64位----找到release----复制链接地址

在Linux中加入yum源

[root@web01 ~]# rpm -ivh https://mirrors.tuna.tsinghua.edu.cn/zabbix/zabbix/3.4/rhel/6/x86\_64/zabbix-release-3.4-1.el6.noarch.rpm

[root@web01 ~]# yum install zabbix-agent -y

以后这些操作要交给kickstart和cobbler，装完系统就装上zabbix-agent

#修改一下服务端的IP地址

第97行 Server = 172.16.1.61

或者使用sed -i.ori ''

sed -i.ori 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_agentd.conf

#将zabbix 设置为 开机自启动,并重启

[root@web01 ~]# chkconfig zabbix-agent on

[root@web01 ~]# /etc/init.d/zabbix-agent restart

Shutting down Zabbix agent: [FAILED]

Starting Zabbix agent: [ OK ]

在m01 zabbix的服务端上检查，这些服务都是要启动的，而且要开机自启动

[root@m01 doc]# systemctl is-active httpd.service mariadb.service zabbix-server.service zabbix-agent.service

active

active

active

active

[root@m01 doc]# systemctl is-enable httpd.service mariadb.service zabbix-server.service zabbix-agent.service

Unknown operation 'is-enable'.

[root@m01 doc]# systemctl is-enabled httpd.service mariadb.service zabbix-server.service zabbix-agent.service

disabled

disabled

disabled

enabled

[root@m01 doc]# systemctl enable httpd.service mariadb.service zabbix-server.service zabbix-agent.service   
[root@m01 doc]# systemctl is-enabled httpd.service mariadb.service zabbix-server.service zabbix-agent.service   
enabled  
enabled  
enabled  
enabled

在服务端检查一下是不是在线，可以连接

[root@m01 doc]# zabbix\_get -s 172.16.1.7 -p 10050 -k 'agent.ping'  
1

在web界面中  配置----主机----创建主机

主机名称： 主机名

可见名称： 随意设置，知道意思就行，

群组： 最好设置一个群组，方便批量管理

agent代理程序的接口： 172.16.1.7 端口号不用改

下面的SNMP、JMX、IPMI暂时不用管

接下来 点击 模板选项卡

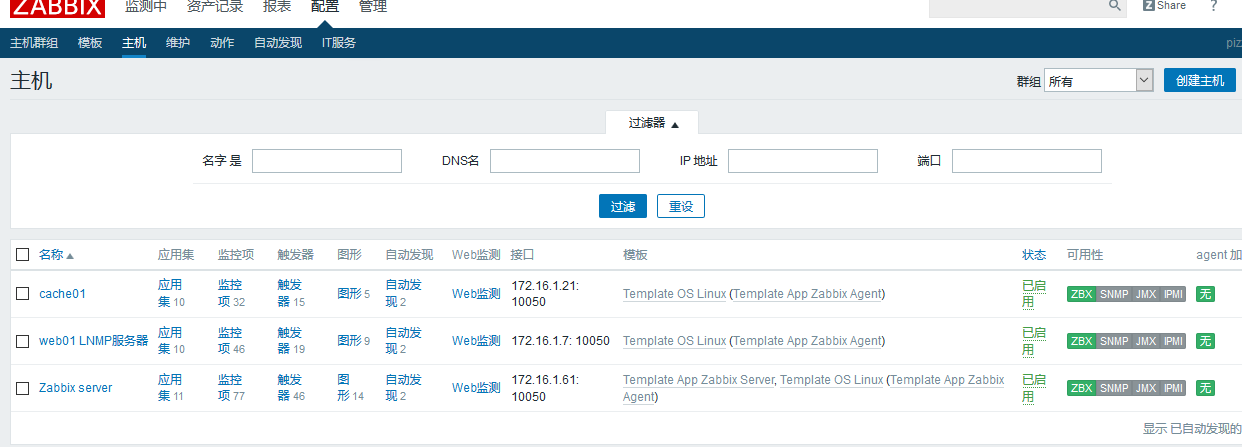
不用点击选择，在搜索框中输入linux ，选择Template OS linux-----点击小的添加----点击添加

保证在m01上能够解析web01 的名字   -----   172.16.1.7 web01

注：实际做的时候，我没有在host中解析web01 也连上了....后面有问题再说吧，

继续添加一台服务器cache01

都连接上了



**如何快速添加主机及键值**

随便选择一个，然后最下面有一个克隆，修改一下主机名和IP

模板中有什么呢？

可以查看监控项中的键值，就是zabbix\_get 命令中的-k参数 中的k值，是获取数据的本质

**自定义创建键值**

也叫作自定义监控，在模板提供的键值中没有我们需要的数据时，我们要自己做键值

zabbix自带模板Template OS Linux (Template App Zabbix Agent)  
提供CPU、内存、磁盘、网卡等常规监控，只要新加主机关联此模板，就可自动添加这些监控项。

需求：服务器登录人数不超过3人，超过3人报警

1、获取登陆服务器人数  
[root@web01 ~]# who  
root     pts/0        2019-03-16 15:49 (10.0.0.253)  
2、统计  
[root@web01 ~]# who|wc -l  
1  
3、将命令放在/etc/zabbix/zabbix\_agentd.d/userparameter\_mysql.conf 中  
这是用来放置用户自定义监控项的，用户可以在这个目录中写自己的.conf文件  
4、在agent注册key

# 语法：key名字要唯一，多个key以行为分割

UserParameter=<key>,<shell command>

UserParameter=login-user,who|wc -l

UserParameter=login-user,/bin/sh /server/scripts/login.sh # 执行脚本

# cd /etc/zabbix/zabbix\_agentd.d/

# vim userparameter\_login.conf

UserParameter=login-user,who|wc -l

# 修改配置文件要重启

systemctl restart zabbix-agent.service   
centos6中重启  
/etc/init.d/zabbix-agent restart

# 在server检测

# zabbix\_get -s 172.16.1.21 -p 10050 -k "login-user"

自定义key能被server和agent认可

**WEB页面中操作**

1. 创建模板：填好名称、群组，就可以保存了
2. 创建应用集：给监控项（item）分类
   1. 添加一个user，表示用户相关的
3. 创建监控项：监控的内容
   1. 点击监控项---创建监控项---填写内容（历史时长短一点，趋势时长可以长到365d）
4. 创建触发器：当监控项获取到的值 达到一定条件时 就触发报警
   1. 名称要表达清楚报警内容　　login.user bigger than 3
   2. 表达式要 选择好 {fist\_model\_web:login.user.last()}>3 这是选择后自动生成的
   3. 点击添加
5. 创建图形
   1. 在选中监控项后，的功能下拉框选最大值
   2. 绘图风格选先
   3. 其他随意
6. 主机关联模板
   1. 配置---主机---web01---模板---添加之前做好的模板

处理报警：在设置的触发器中有一项允许手动关闭，在收到报警后，我们可以在确认事件中关闭报警

**添加报警**

报警也可以在web中设置，就在主页小人处

我们还可以通过网站设置报警，然后等着报警信息，就不用一直盯着页面了

登陆网站：http://www.onealert.com/index.html 注册账号

报警的本质：

什么时候发邮件：当状态改变的时候才会发邮件

好--坏

坏--好

如何查找安装报警脚本的路径？

[root@m01 doc]# grep -i alertscript /etc/zabbix/zabbix\_server.conf

### Option: AlertScriptsPath

# AlertScriptsPath=${datadir}/zabbix/alertscripts

AlertScriptsPath=/usr/lib/zabbix/alertscript

安装报警插件，下载地址和安装命令都是报警网站提供的

[root@m01 alertscripts]# wget http://www.onealert.com/agent/release/oneitsm\_zabbix\_release-1.2.0.tar.gz

...下载过程略...

[root@m01 alertscripts]# tar -zxf oneitsm\_zabbix\_release-1.2.0.tar.gz

[root@m01 alertscripts]# cd oneitsm/bin

[root@m01 bin]# bash install.sh 42dd9ef9-cc1c-8def-e95a-d20526a5341f

./log.sh: line 6: /usr/lib/zabbix/alertscripts/oneitsm/bin/oneitsm.conf: No such file or directory

start to create config file...

Zabbix管理地址: 172.16.1.61

Zabbix访问失败,请重新输入.

Zabbix管理地址: 10.0.0.61

Zabbix访问失败,请重新输入.

Zabbix管理地址: 10.0.0.61/zabbix # 地址 zabbix的WEB页面的地址

Zabbix管理员账号: Admin

Zabbix管理员密码:

start to auth by zabbix admin user and password...

...安装过程略...

create action success!

安装成功.

登陆zabbix的WEB界面配置

管理---报警媒介类型---已经配置好了！！

在onealert的配置下中有一个分配策略---就是报警了分配给谁

**聚合图形**

同时看几个数据的图形

聚合图形中还有幻灯片演示

现有聚合图形，才能创建幻灯片

https://share.zabbix.com/中有许多别人创建好的模板

https://github.com/zhangyao8/zabbix-community-repos 模板共享

**全网监控项目说明**

考试目标：使用zabbix将期中架构所有集群监控

具体内容：

一、硬件、系统、网络监控

所有集群节点（所有虚拟机）都监控上

二、应用服务监控

1. 监控备份服务器，简单方法是监控rsync端口，如果有其他更佳方案可以说明；

2. 监控NFS服务器，使用监控NFS进程来判断NFS服务器正常，如果有其他更佳方案可以说明；

3. 监控MySQL服务器，简单方法监控mysql的3306端口，或者使用zabbix提供的Mysql模板，  
　　如果有其他更佳方案可以说明；

4. 监控2台web服务器，简单方法监控80端口，如果有其他更佳方案可以说明；

5. 监控URL地址来更精确的监控我们的网站运行正常；

6. 监控反向代理服务器，PPTP服务器等你在期中架构部署的服务。

7. 监控Nginx的7种连接状态。

项目一的模板都有

项目二中的端口、进程。可以进入任意模板查看，都有

项目二中的第5个检测URL，自定义web检测

　　创建web场景

项目二中的第6个检测端口+进程

项目二中的第7个Nginx的7中链接状态，用状态模块进行配置，

　　1、写7行（简单粗暴的方法）

　　2、写脚本

**自动发现与自动注册**

自动发现：zabbix Server主动发现所有客户端，然后将客户端登记自己的小本本上，  
　　缺点zabbix server压力山大（网段大，客户端多），时间消耗多。

自动注册：zabbix agent主动到zabbix Server上报到，登记；  
　　缺点agent有可能找不到Server（配置出错）

被动模式：默认，都是站在agent的立场上说话，agent被server抓取数据

主动模式：都是站在agent的立场上说话，agent主动的将数据发送给Server

两种模式都是在agent上的配置文件配置的

**自动发现（被动模式）-服务器少的时候使用**

第一步：zabbix Server安装完毕 （完成）

第二步：zabbix agent安装完毕，Server=172.16.1.61 （完毕）

第三步：网页上配置自动发现规则

配置---自动发现---新建或者使用已经有的

修改IP范围---172.16.1.1-10,172.16.1.45多个用，号分割

默认本质上使用的是system.uname，也可以添加新的

设置唯一性

更新或者保存，启用

------------------------------------------------------------------------------

配置---动作---事件源选择‘自动发现’---创建或者修改原有的，也可以克隆

其中，新的触发条件可以不写，因为在自动发现中已经设置了

在操作选项卡里面，添加新的动作操作（添加主机和启用主机）

完成后，跟新启用

什么时候算是完成自动发现了呢，在自动发现中显示主机名就算成功了，然后会出现在主机列表中

**自动注册（主动模式）**

把之前的自动发现关闭

第一步：zabbix Server安装完毕 （完成）

第二步：zabbix agent安装完毕，需要额外增加的配置

vim /etc/zabbix/zabbix\_agentd.conf

ServerActive=172.16.1.61

# Hostname=Zabbix server 和下面的HostnameItem配置一个就行

HostnameItem=system.hostname # 定义获取服务器主机名的方式  
　 HostMetadataItem=system.uname # 主机的源数据

systemctl restart zabbix-agent.service  
　　 重启centos6的方式/etc/init.d/zabbix-agent restart

netstat -tunlp|grep zabbix

第三步：Server上做hosts解析，解析是在这里发挥作用的

echo "172.16.1.21 cache01" >> /etc/hosts

第四步：网页上配置自动注册

　　配置----动作---自动注册（事件源）  
　　动作-触发条件 设置 成 主机元数据 似 Linux（手打的）  
　　操作-添加主机---添加到群组---链接到模板  
　　添加

主机没有添加上，等候时间

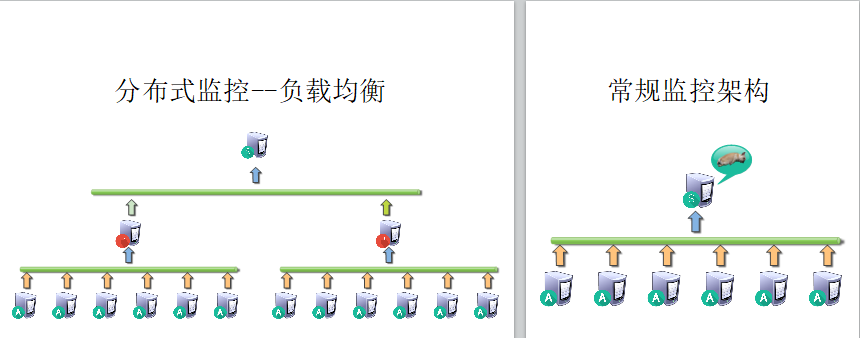
**分布式监控**

Zabbix Server -------->Zabbix agent （只能同一个局域网监控）

分布式监控：

a. 分担压力，降低负载

b. 多机房监控



等于是给各个分区的服务器安排了一个领导，层层汇总

环境准备

做好快照

机器准备：

Zabbix Server m01 监控中心

Zabbix Proxy web01 分中心领导

Zabbix Agent web01 web02 客户端

开始操作，将之前的自动发现和自动注册都关闭

第一步：在代理机web01安装zabbix proxy

rpm -ivh http://repo.zabbix.com/zabbix/3.0/rhel/7/x86\_64/zabbix-release-3.0-1.el7.noarch.rpm

yum -y install zabbix-proxy-mysql

第二步：安装数据库

zabbix proxy也需要数据库，这个数据库不是用于存储监控数据的 只是用于存储配置信息

# 安装数据库

yum -y install mariadb-server

systemctl start mariadb.service

#建库 创建用户授权

mysql

create database zabbix\_proxy character set utf8 collate utf8\_bin;

grant all privileges on zabbix\_proxy.\* to zabbix@'localhost' identified by 'zabbix';

# 将多余的空的用户删除  
drop user “‘@’localhost”；  
drop user “‘@’web01”；  
# 跟新  
flush privileges；  
exit

# 导入数据库文件

zcat /usr/share/doc/zabbix-proxy-mysql-3.0.13/schema.sql.gz |mysql -uzabbix -pzabbix zabbix\_proxy

第三步： 配置zabbix proxy，并在zabbix网页登记

# 配置zabbix proxy连接数据库

sed -i.ori '162a DBPassword=zabbix' /etc/zabbix/zabbix\_proxy.conf

systemctl start zabbix-proxy.service

# 修改Server

sed -i 's#Server=127.0.0.1#Server=172.16.1.61#' /etc/zabbix/zabbix\_proxy.conf

# 修改hostname，确保这个名字是唯一的，不用web01是因为，web01要给下面的客户端使用  
# 这个hostname，一定要能在m01中解析，即最顶端的老大  
sed -i 's#Hostname=Zabbix proxy#Hostname=proxy-web01#' /etc/zabbix/zabbix\_proxy.conf  
将172.16.1.7      web01  proxy-web01 添加在m01的 host文件中

systemctl start zabbix-proxy.service

Centos6中的重启方法/etc/init.d/zabbix-proxy start(第一次启动所以使用的start  
### 问题 ###  
# 查看zabbix-proxy日志发现，一只连接不上数据库，它连接的的是/var/lib/mysql/mysql.sock  
　　# 在/etc/zabbix/zabbix\_proxy.conf中的默认的是/tmp/mysql.sock  
　　# 我们需要将它启动,把配置文件中DBSocket前面的# 删掉  
继续重启  
第四步：web页面设置  
管理---agent代理程序---创建代理  
agent代理程序名称：要和 Hostname=proxy-web01 一样  
把客户机加上  
添加

第五步：客户端agent 指向 proxy

[root@web01 ~]# grep ^Server /etc/zabbix/zabbix\_agentd.conf

Server=172.16.1.61

ServerActive=172.16.1.61

[root@web01 ~]# sed -i 's#172.16.1.61#172.16.1.7#g' /etc/zabbix/zabbix\_agentd.conf

[root@web01 ~]# grep ^Server /etc/zabbix/zabbix\_agentd.conf

Server=172.16.1.7

ServerActive=172.16.1.7  
# 文件中的Hostname 要和 页面上的一致，即本服务器的hostname

[root@web01 ~]# systemctl restart zabbix-agent.service

把客户端服务器hosts中的解析本机的去掉？？

主机的名字只需要在proxy和m01上解析就可以了

**SNMP监控**

主要用来监控网络设备，也可以用来监控服务器

只需要在网络设备中开启SNMP监控即可

小结：前面所讲内容全部属于zabbix agent方式监控

当前场景：无法安装agent，通过snmp监控

snmp ：simple network manager protocol 简单网络管理协议

# 在m01下，安装软件

yum -y install net-snmp net-snmp-utils

配置snmp

sed -i.ori '57a view systemview included .1' /etc/snmp/snmpd.conf  
意思是开启被读取的权限

systemctl start snmpd.service

默认的端口是161

使用SNMP

[root@db01 ~]# snmpwalk -v 2c -c public 127.0.0.1 sysname

# snmpwalk 类似 zabbix\_get

# -v 2c 指定使用snmp协议的版本 snmp分为v1 v2 v3

# -c public 指定暗号

# sysname 类似zabbix的key 键值中的键，还有很多其他的 OID

怎么通过snmp协议来监控呢？

在web页面中选择主机。在设置中的SNMP接口处添加

它和agent的区别是不支持自定义监控

模板也要改，把原来的linux os 取消链接。添加Template SNMP OS Linux

还是不行的话，点击原来的先添加再取消并清理，然后保存

补充：web监测的时候，要把SNMP的模板删除，换成之前用的普通的

**最核心的部分是监控的体系部分，一定要作为自己的必杀技，搞定**

分类: [Operations](https://www.cnblogs.com/bubu99/category/1620808.html)

# [Zabbix快速入门](https://www.cnblogs.com/yanshicheng/p/10436954.html)

**模板的管理与使用**

**一 模板**

　　模板是zabbix的核心,因为模板继承了所有要监控的内容,以及所要展示的图形等等,zabbix的安装部署完成之后,自带了很多模板(网络设备模板,操作系统模板,常用应用软件模板),这些模板足够满足日常百分之八十左右的应用需求.

　　模板的组成: 应用集, 监控项, 触发器, 图形, 聚合图形,自动发现, Web检测,

　　模板的重要组成: 监控项, 触发器, 自动发现, 图形

1.1.1 模板的分类

　　1) 监控系统状态的模板

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # 对 Linux 系统的监控模板  Template OS Linux  # 对 Windows 系统的监控模板  Template OS Windows  # 对 Mac OS X 系统的监控模板  Template OS Mac OS X  # 对 VM VMware 系统的监控模板  Template VM VMware |

　　2) 监控网络和网络设备的模板

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #开启SNMPv1监控的模板  Template Module Generic SNMPv1    #开启SNMPv2监控的模板  Template Module Generic SNMPv2    Template Module Interfaces Simple SNMPv2  emplate Net Cisco IOS SNMPv2  Template Net Juniper SNMPv2  Template Net Huawei VRP SNMPv2</p> |

　　3) 监控应用软件和服务的模板

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #对HTTPD服务的监控模板  Template App HTTP Service    #对MySQL服务的监控模板  Template DB MySQL    #对SSH服务的监控模板  Template App SSH Service    #对主机Ping的监控模板  Template Module ICMP Ping    #对Java服务的监控模板  Template App Generic Java JMX    #对Zabbix Agent状态的监控模板  Template App Zabbix Agent    #对Zabbix Server状态的监控模板  Template App Zabbix Server |

**二 创建应用集**

**1) 操作步骤**

　　配置 --->> 模板 ---> (新建模板/打开现有模板) --->> 应用集 --->> 创建应用集 --->> 输入要创建的应用集名称

　　创建应用集可以理解为一个群组,部门,列如要对CPU属性监控,那么就可以创建一个CPU的应用集,在这个应用集下面可以创建多个监控CPU的监控项.

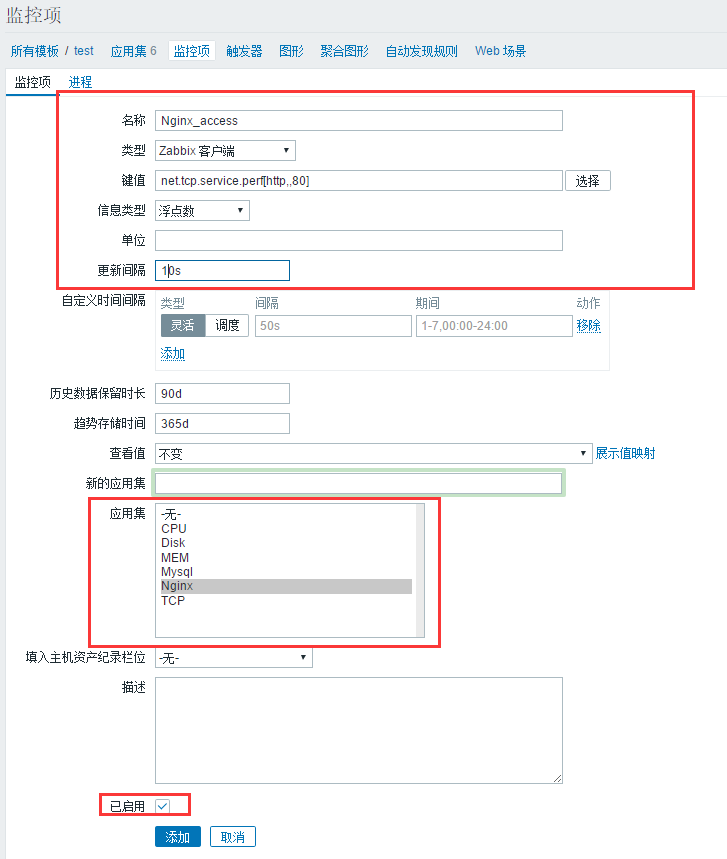
　　应用集的主要作用: 便于对监控项管理,对监控项进行分类



**三 创建监控项**

**1) 操作步骤**

　　　　配置 --->> 模板 ---> (新建模板/打开现有模板) --->> 监控项 --->> 创建监控项 --->> .....



**2) 参数说明**

　　1. 名称

　　　　　名称是创建的监控项的名称,自定义即可,

　　2. 类型

　　　　类型是设置此监控项通过什么方式进行监控,zabbix可选的监控类型有很多,常用的有 zabbix 客户端,zabbix客户端(主动模式),简单检查, SNMP客户端, zabbix采集器等,zabbix 客户端监控,也称为zabbix客户端(被动式)监控,这个需要根据需求进行选择.

　　2. 键值

　　　　键值可以自己定义,也可以使用zabbix 自带的键值,简单来说,键值就是获取数据的方式.

　　　　可以使用: [zabbix\_get -s 192.168.10.53 -k "agent.ping"] 工具进行测试

　　3. 信息类型

　　　　信息类型看具体返回的值,此信息类型为 浮点数,所以选择浮点数.

　　4. 单位

　　　　范围就是数据的返回单位,百分比还是什么

　　5. 更新间隔时间

　　　　数据获取更新的事件

　　6. 自定义时间间隔

　　　　灵活的在指定的时间段进行更新数据

　　7. 历史数据保留时长

　　　　历史数据的保留时间,自动删除

　　8. 趋势储存时间

　　　　趋势图的保留时间

　　9. 应用集

　　　　在那个应用集,

**四 创建触发器**

　　　触发器是用于故障告警的一个设置,将一个监控项,添加触发器后,此监控项如果出现问题,就会激活触发器,然后触发器将自动连接告警动作,最后触发告警.



　　触发器函数(diff,avg,last,nodata)

　　1) diff

|  |  |
| --- | --- |
| 1  2  3 | # 参数：不需要参数  # 支持值类型：float,int,str,text,log  # 作用：返回值为1表示最近的值与之前的值不同，即值发生变化，0表示无变化。 |

　　2) last

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 参数：#num  支持值类型：float,int,str,text,log  作用：获取最近的值，“#num”表示最近第N个值，请注意当前的#num和其他一些函数的#num的意思是不同的，  例如：last(0)或last()等价于last(#1)，表示获取最新的值，last(#3)表示最近第3个值（并不是最近的三个值），  注意，last函数使用不同的参数将会得到不同的值，#2表示倒数第二新的数据。例入从老到最新值为1,2,3,4,5,6,7,8,9,10，last(#2)得到的值为9，last(#9)得到的值为2。  另外，last函数必须包含参数。 |

　　3) avg

|  |  |
| --- | --- |
| 1  2  3  4  5 | 参数：秒或#num  支持类型：float,int  作用：返回一段时间的平均值  例如，avg(5)表示最后5秒的平均值，avg(#5）表示最近5次得到值的平均值，avg(3600,86400）表示一天前的一个小时的平均值。  如果仅有一个参数，表示指定时间的平均值，从现在开始算起，如果有第二个参数，表示漂移，从第二个参数前开始算时间，#n表示最近n次的值。 |

　　4) change

|  |  |
| --- | --- |
| 1  2  3  4 | 参数：无需参数  支持类型：float,int,str,text,log  作用：返回最近获得值与之前获得值的差值，返回字符串0表示相等，1表示不同。  例如，change(0)>n表示最近得到的值与上一个值的差值大于n，其中，0表示忽略参数。 |

　　5) nodata

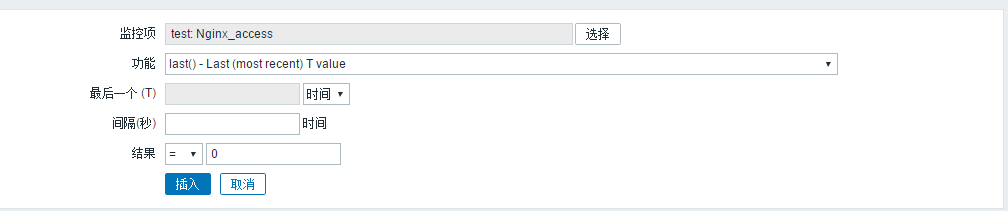
|  |  |
| --- | --- |
| 1  2  3 | 参数：秒  支持值类型：any  作业：探测是否能接收到数据，当返回值为1表示指定的间隔(间隔不应小于30秒)没有接收到数据，0表示其正常接收数据 |

　　6) count

|  |  |
| --- | --- |
| 1  2  3  4  5 | 参数：秒或#num  支持类型：float,int,str,text,log  作用：返回指定时间间隔内数值的统计。  例如：count(600)：表示最近10分钟得到值的个数  count(600,12)：表示最近10分钟得到值的个数等于12其中，第一个参数是指定时间段，第二个参数是样本数据。 |

　　7.) sum

|  |  |
| --- | --- |
| 1  2  3  4 | 参数：秒或#num  支持值类型：float,int  作用：返回指定时间间隔中收集到的值的总和，时间间隔作为第一个参数支持秒或收集值的数目(以#开始)。  例如：sum(600)：表示在600秒之内接收到所有值的和。sum(#5)：表示最后5个值的和。 |



作者：闫世成

出处：[http://cnblogs.com/yanshicheng](http://www.cnblogs.com/yanshicheng/)

联系：[yans121@sina.com](https://www.cnblogs.com/yanshicheng/p/yans121@sina.com)

本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接，如有问题或建议，请多多赐教，非常感谢。

# ELK学习总结——我们为什么要用ELK

# **一. ELK是什么？**

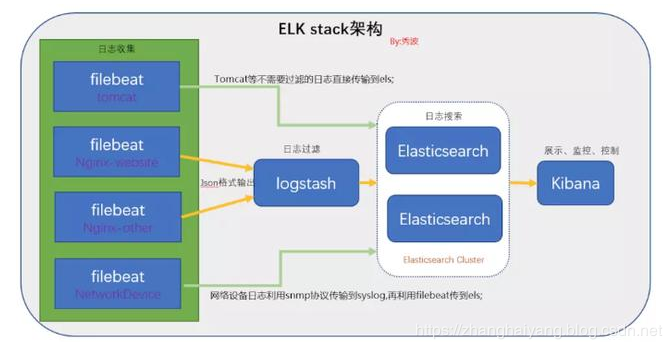
ELK 是elastic公司提供的**一套完整的日志收集以及展示的解决方案**，是三个产品的首字母缩写，分别是ElasticSearch、Logstash 和 Kibana。



**ElasticSearch简称ES**，它是一个实时的分布式搜索和分析引擎，它可以用于全文搜索，结构化搜索以及分析。它是一个建立在全文搜索引擎 Apache Lucene 基础上的搜索引擎，使用 Java 语言编写。  
**Logstash**是一个具有实时传输能力的数据收集引擎，用来进行数据收集（如：读取文本文件）、解析，并将数据发送给ES。  
**Kibana**为 Elasticsearch 提供了分析和可视化的 Web 平台。它可以在 Elasticsearch 的索引中查找，交互数据，并生成各种维度表格、图形。  
**二：ELK的用途**

传统意义上，**ELK是作为替代Splunk的一个开源解决方案**。Splunk 是日志分析领域的领导者。日志分析并不仅仅包括系统产生的错误日志，异常，也包括业务逻辑，或者任何文本类的分析。而基于日志的分析，能够在其上产生非常多的解决方案，譬如：

1.**问题排查**。我们常说，运维和开发这一辈子无非就是和问题在战斗，所以这个说起来很朴实的四个字，其实是沉甸甸的。很多公司其实不缺钱，就要稳定，而要稳定，就要运维和开发能够快速的定位问题，甚至防微杜渐，把问题杀死在摇篮里。日志分析技术显然问题排查的基石。基于日志做问题排查，还有一个很帅的技术，叫全链路追踪，比如阿里的eagleeye 或者Google的dapper，也算是日志分析技术里的一种。  
2.**监控和预警**。 日志，监控，预警是相辅相成的。基于日志的监控，预警使得运维有自己的机械战队，大大节省人力以及延长运维的寿命。  
3.**关联事件**。多个数据源产生的日志进行联动分析，通过某种分析算法，就能够解决生活中各个问题。比如金融里的风险欺诈等。这个可以可以应用到无数领域了，取决于你的想象力。  
4.**数据分析**。 这个对于数据分析师，还有算法工程师都是有所裨益的。



# **三：ELK火热的原因？**

**ELK之前，有没有类似解决方案呢？**

某大神说是有的，当时应该是基于 Sphinx + Google char的。 Sphinx 对应ES, Google char 对应 Kibana。

**那为啥当时它没有火而现在的ELK火了呢？一种比较玄幻的解释是**：

事实上开源界永远有多种选择，比如基于java的lucene的es，也有基于c的lucy的dezi。但是谁火谁不火，真的是一个很玄妙的事情。

我觉得原因有很多方面，一个简单而较为核心的因数是时机，所谓时势造英雄是也。当然，任何一件事情不可能是一个因子引起的，或者我们说时机是一个较为宽泛抽象的因子。

**下面我从多个因素去阐述为什么ELK突然蓬勃发展？**

**1，数据(日志)的日益增多：**

早年能够产生足够数据的就那么一些站点，**而现在一个初创的企业可能都需要面临海量用户/海量请求/海量分析的压力**，其中产生的日志自然也是非常可观，而随着业务越来越复杂，微服务重新得到重视，无论系统日志，还是业务日志都更进一步了。运维或者开发们发现，我要从这么大规模的系统中(几百个上千个服务)产生的这么多日志(千亿规模)，去排查问题，简直是没有可能了。以前有这么大数据量的公司，都是有实力的公司，他们可能有内部专用的系统去处理。然而**现在突然成为了一个普遍需求，这个时候ELK顺势而上，也就水到渠成**。

**2，开源：**

开源现在已经融入到IT社区的血液里。虽然我们说商业,自研，还有开源三者之间是相辅相成，相濡以沫或者偶尔会相爱相杀，但是如果有开源可以选择，显然大部分开发或者运维还是首选开源的。有位大牛说的好：  
**ELK其开源属性，显然是比Splunk 略胜一筹的。**

**3，行业属性：**

有些行业对日志的依赖是非常大的，比如 CDN 日志除了能排查错误，对其分析还能对CDN调度等很多方面产生影响，这些都是实打实的经济效益。

**4，运维自身发展：**

运维本身也在发展，不可能一直在刀耕火种的年代。而日志对于运维来说，应该算是命根子了。对一个成体系的，标准化的日志分析方案的需求，也是历史发展的必然。**ELK在恰当的时候产生，运维接受他就是自然而然的了。**

**5，ELK自身属性问题：**

引用一位大神的说法：

ELK能解决的核心问题，覆盖面也广，标准化，易扩展集成，开发和运维都对其感冒

**ELK 本身非常易用，现在也有一个非常好的社区，加上需求如此之大，不火都不行。**

**6，大数据导致机器开始变得廉价：**

大数据的一个很好的副作用是让机器在某种意义上变得廉价了。少则几十上百台，多则上万甚至几十万台。服务器数量的急速攀升促进了很多技术的发展，典型的比如现在火的不要不要的深度学习。这就意味着，拿出几十台，上百台服务器做日志分析，一点问题也没有，集中式的日志分析慢慢成为主流。而ELK也是一个典型的集中式日志分析方案。

**7，写入时计算->查询时计算：**

所谓写入时计算是指将数据经过较为复杂的处理，聚合，得到的结果直接面向查询。 写入时计算规则由查询需求决定。随着存储格式的不断进步，譬如列式存储等的普及，以及强大的计算资源(一个ES集群动则上百台)，使得直接存储原汁原味的数据，然后查询的时候做各种计算变得可能。而ELK已经提供较为强大的查询功能。总体而言，写入时计算的大方向是往查询时计算转化。查询时计算最大的优势是支持任意查询，不丢失信息。

# [ELK日志系统介绍](https://www.cnblogs.com/Epir/p/10694694.html)

**ELK介绍**

**需求背景：**

* 业务发展越来越庞大，服务器越来越多
* 各种访问日志、应用日志、错误日志量越来越多，导致运维人员无法很好的去管理日志
* 开发人员排查问题，需要到服务器上查日志，不方便
* 运营人员需要一些数据，需要我们运维到服务器上分析日志

**为什么要用到ELK：**

一般我们需要进行日志分析场景：直接在日志文件中 grep、awk 就可以获得自己想要的信息。但在规模较大也就是日志量多而复杂的场景中，此方法效率低下，面临问题包括日志量太大如何归档、文本搜索太慢怎么办、如何多维度查询。需要集中化的日志管理，所有服务器上的日志收集汇总。常见解决思路是建立集中式日志收集系统，将所有节点上的日志统一收集，管理，访问。

大型系统通常都是一个分布式部署的架构，不同的服务模块部署在不同的服务器上，问题出现时，大部分情况需要根据问题暴露的关键信息，定位到具体的服务器和服务模块，构建一套集中式日志系统，可以提高定位问题的效率。

一个完整的集中式日志系统，需要包含以下几个主要特点：

* 收集－能够采集多种来源的日志数据
* 传输－能够稳定的把日志数据传输到中央系统
* 存储－如何存储日志数据
* 分析－可以支持 UI 分析
* 警告－能够提供错误报告，监控机制

而ELK则提供了一整套解决方案，并且都是开源软件，之间互相配合使用，完美衔接，高效的满足了很多场合的应用。是目前主流的一种日志系统。

**ELK简介：**

ELK是三个开源软件的缩写，分别为：Elasticsearch 、 Logstash以及Kibana , 它们都是开源软件。不过现在还新增了一个Beats，它是一个轻量级的日志收集处理工具(Agent)，Beats占用资源少，适合于在各个服务器上搜集日志后传输给Logstash，官方也推荐此工具，目前由于原本的ELK Stack成员中加入了 Beats 工具所以已改名为Elastic Stack。

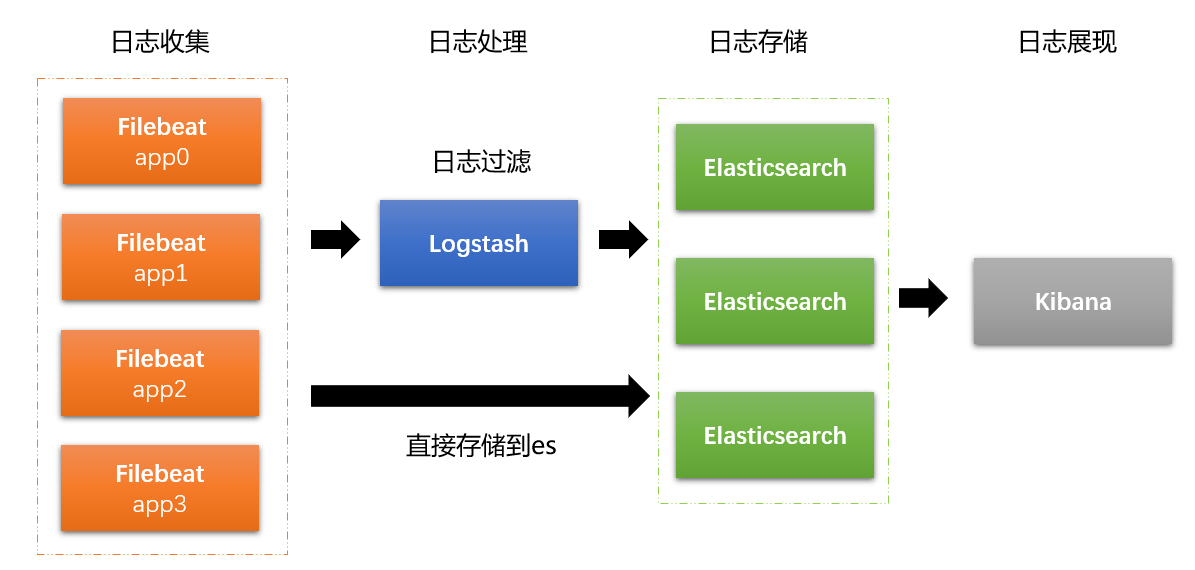
Elastic Stack包含：

* Elasticsearch是个开源分布式搜索引擎，提供搜集、分析、存储数据三大功能。它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。详细可参考[Elasticsearch权威指南](https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html)
* Logstash 主要是用来日志的搜集、分析、过滤日志的工具，支持大量的数据获取方式。一般工作方式为c/s架构，client端安装在需要收集日志的主机上，server端负责将收到的各节点日志进行过滤、修改等操作在一并发往elasticsearch上去。
* Kibana 也是一个开源和免费的工具，Kibana可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助汇总、分析和搜索重要数据日志。
* Beats在这里是一个轻量级日志采集器，其实Beats家族有6个成员，早期的ELK架构中使用Logstash收集、解析日志，但是Logstash对内存、cpu、io等资源消耗比较高。相比 Logstash，Beats所占系统的CPU和内存几乎可以忽略不计

ELK Stack （5.0版本之后）--> Elastic Stack == （ELK Stack + Beats）。目前Beats包含六种工具：

* Packetbeat： 网络数据（收集网络流量数据）
* Metricbeat： 指标 （收集系统、进程和文件系统级别的 CPU 和内存使用情况等数据）
* Filebeat： 日志文件（收集文件数据）
* Winlogbeat： windows事件日志（收集 Windows 事件日志数据）
* Auditbeat：审计数据 （收集审计日志）
* Heartbeat：运行时间监控 （收集系统运行时的数据）

上面是一些官方说明，比较清晰明了，下面是一般的ELK架构：



 filebeat是作为客户端的日志收集器，部署在若干个产出日志的应用系统上，将数据传输到es集群和logstash服务，传递给logstash的数据会被进一步处理最终流入es集群，kibana从es中获取索引日志信息进行展现。

之后会依次详细介绍logstash、beats、kibana组件，由于在工作中elasticsearch是作为存储组件使用的，无需特别调研，就不详细介绍了，只介绍些在elk中使用到的操作。

**标签:**[ELK](https://www.cnblogs.com/Epir/tag/ELK/)**,**[日志系统](https://www.cnblogs.com/Epir/tag/%E6%97%A5%E5%BF%97%E7%B3%BB%E7%BB%9F/)

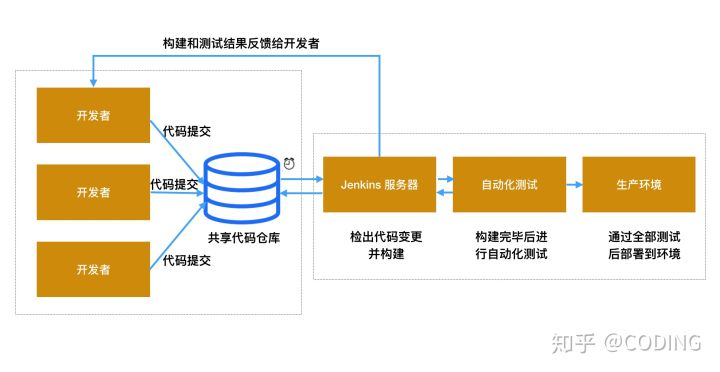
**持续集成（CI）**是每次团队成员提交版本控制更改时自动构建和测试代码的过程。这鼓励开发人员通过在每个小任务完成后将更改合并到共享版本控制存储库来共享代码和单元测试。

**2019 DevOps 必备面试题——持续集成篇**

**Q1：什么是持续集成？**

**我会建议你以持续集成的最小定义作为开始来回答这个问题。这是一种研发实践，需要开发人员每天多次将代码集成到共享代码库中。然后通过自动构建来验证每次代码的修改，以便团队尽早发现问题。**

**我建议你解释一下在以前的工作中是如何实施持续集成的，可以参考以下示例：**



**在上图中： 1、开发人员将代码 clone 至私有工作区。 2、完成编码后，他们将更改提交至共享代码库中（版本控制仓库）。 3、CI 服务器监视代码仓库并在发生更改时检出更改。 4、紧接着 CI 服务器提取这些变更进行构建、运行单元以及集成测试。 5、CI 服务器会立即告知团队构建成功与否。 6、如果构建失败，CI 服务器会向团队发送告警。 7、研发团队将尽快解决问题。 8、这个过程会不断重复。**

**Q2：为什么研发团队需要开发与测试的持续集成？**

**对于这个答案，你应该关注持续集成的需求。我建议你在回答中提到以下解释： 开发和测试的持续集成通过在完成所有开发之后替换传统的测试实践，来提高软件质量并减少交付耗时。它允许开发团队尽早检测和定位问题，因为开发人员需要每天多次（或更频繁地）将代码集成到代码仓库中，然后自动验证每次集成。**

**Q3：持续集成的成功因素有哪些？**

在这里，你必须提到持续集成的要求，可以在回答中包含以下几点：

* 维护代码仓库
* 自动化构建
* 让构建自我检测
* 每个人每天都确保已将修改提交至基线
* 保持快速构建
* 在生产环境的克隆环境中进行测试
* 研发团队可以轻松获得最新的可交付成果
* 每个人都可以看到最新构建的结果
* 自动部署

**Q4：如何将 Jenkins 从一台服务器迁移或者复制到另一台服务器？**

我会通过将 jobs 目录从旧服务器复制到新服务器的方式来完成这个事情。有很多种方法可以做到这一点：

* 只需复制相应的 job 目录，即可将 job 从一个 Jenkins 服务器移动到另一个。
* 通过使用其它名称克隆 job 目录来制作现有 job 的副本。
* 通过重命名目录来重命名现有 job。请注意，如果你更改了 job 名称，则需要更改尝试调用该重命名 job 的所有 job 。

**Q5：如何在 Jenkins 中创建备份和复制文件？**

可以很直接地回答这个问题：要创建备份。你需要做的就是定期备份 JENKINS\_HOME 目录。这包含所有构建 job 配置，从属节点配置和构建历史记录。要创建 Jenkins 的备份，只需复制此目录即可，你还可以复制 job 目录或重命名目录。

**Q6：如何配置 Jenkins 的 job？**

关于这个答案的解决方法是首先提一下如何创建 job：转到 Jenkins 首页，选择“New Job”，然后选择“Build a free-style software project”。然后你可以设置这个自由式 job 的元素：

* 可选的 SCM，例如源代码所在的 CVS 或 Subversion。
* 用于控制 Jenkins 何时执行构建的触发器。
* 某种构建脚本，用于执行实际工作的构建（ant，maven，shell 脚本，批处理文件等）。
* 从构建中收集信息的可选步骤，例如归档制品、记录 javadoc 和测试结果。
* 配置构建结果通知其他人/系统的步骤，例如发送电子邮件、即时消息、更新问题跟踪器等。

**Q7：列举 Jenkins 中一些有用的插件**

下面我将提到一些重要插件：

* Maven 2 project
* Amazon EC2
* HTML publisher
* Copy artifact
* Join
* Green Balls

我觉得这些是最有用的插件，你也可以添加你认为有用的插件。但是请确保首先提到上述插件，然后添加你自己的插件。

**Q8：如何保证 Jenkins 的安全？**

* 确保 global security 配置项已经打开。
* 确保用适当的插件将 Jenkins 与企业员工目录进行集成。
* 确保启用项目矩阵的权限访问设置。
* 通过自定义版本控制的脚本来自动化 Jenkins 中设置权限/特权的过程。
* 限制对 Jenkins 数据/文件夹的物理访问。
* 定期对其进行安全审核。

[**Jenkins+Github持续集成**](https://www.cnblogs.com/dancicici/p/9205602.html)

由于最近团队代码库从coding迁移到github，在CI工具的选型上尝试了[travis-ci](http://travis-ci.com/" \t "_blank)和[circle-ci](https://circleci.com/" \t "_blank)，最后决定自己搭建CI服务器，而我也有幸认领了这个任务的调研，因此有了这篇文章。

之前写过一篇文章[浅谈Jenkins+Node.js持续集成](https://segmentfault.com/a/1190000004639923)，那真的是浅谈，Jenkins包含的东西实在太多了，作为从hudson分支出来的开源免费的版本，插件与hudson通用，有更快的迭代速度和稳定性。

**为什么选择Jenkins**

答案简单：因为免费，学习资料多。

**开始吧**

安装配置这里就不赘述了，移步[浅谈Jenkins+Node.js持续集成](http://www.jianshu.com/p/64b498304d07" \t "_blank)

**0. 准备**

因为要与Github通信，所以需要准备一台服务器，该服务器能访问到Github，Github能访问到它。  
为了这个测试，我特地在[搬瓦工VPS](https://bandwagonhost.com/aff.php?aff=5443)买了服务器，顺便介绍一下这个高性价比的vps供应商，$2.99约合人民币18元每个月，可一键搭建shadowsocks。但是记得有个坑就是购买的时候一定要选好机房，之前买过洛杉矶的卡的要死，打条命令之后要等好久才显示，对它失去信心不想用它了，后来听一个朋友说*亚利桑那州*（*Arizona*）的机房挺稳定的，再给它一次机会，这次买了Arizona机房的果然速度挺快的😀

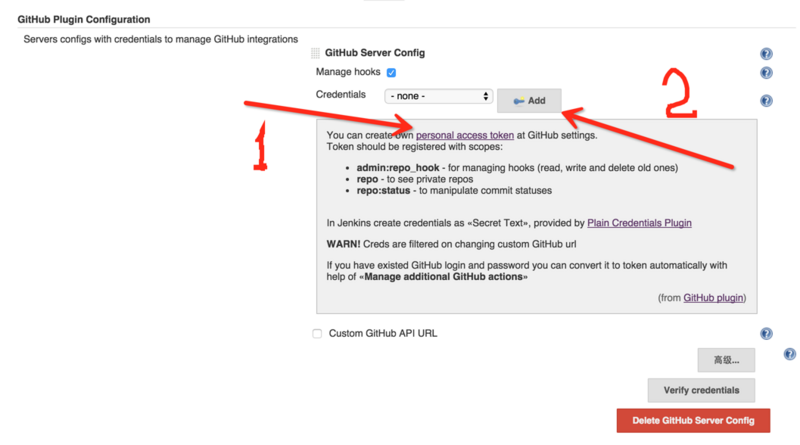
**1. 安装[GitHub Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Github+Plugin" \t "_blank)**

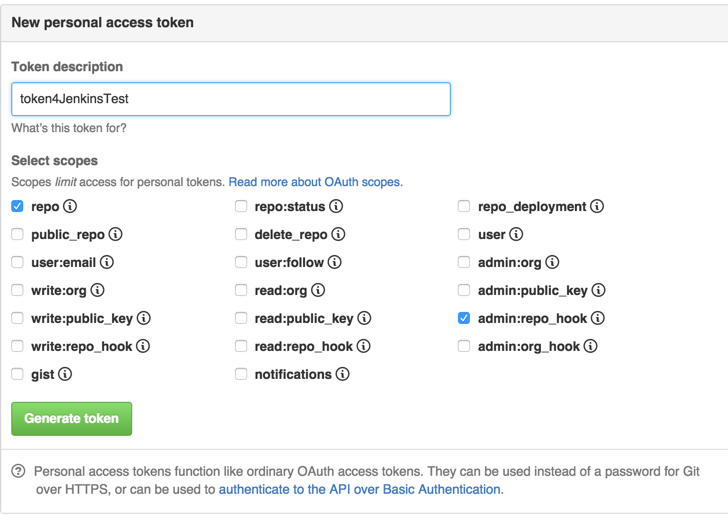
直接安装Github Plugin, jenkins会自动帮你解决其他插件的依赖，直接安装该插件Jenkins会自动帮你安装[plain-credentials](https://wiki.jenkins-ci.org/display/JENKINS/Plain+Credentials+Plugin" \t "_blank) 、[git](https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin" \t "_blank) 、 [credentials](https://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin) 、 [github-api](https://wiki.jenkins-ci.org/display/JENKINS/GitHub+API+Plugin" \t "_blank)

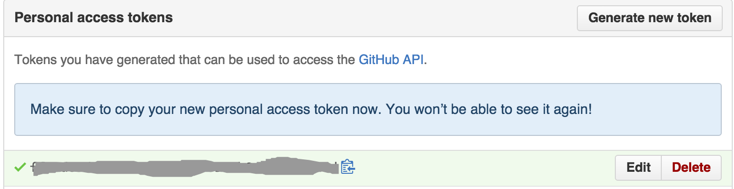


**2. 配置Github插件**

系统管理 >> 系统设置 >> GitHub Plugin Configuration

  
首先点击[personal access token](https://github.com/settings/tokens/new" \t "_blank)到github上  
也就是github上用户Settings >> personal access tokens

  
勾选给Jenkins的访问权限，Github plugin的帮助信息里说要admin:repo\_hook、repo和repo:status权限，其实repo:status是包含在repo里的，详见[这里](https://developer.github.com/v3/oauth/" \l "scopes" \t "_blank)。 点击Generate token创建一个token

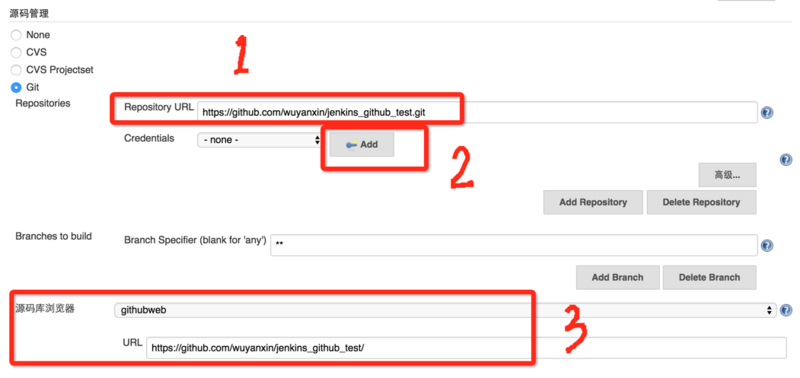
  
复制这个token，回到Jenkins点击Add按钮

  
选择Secret text，粘贴token，添加描述，点击添加。  
点击Verify credentials测试token，显示Credentials verified for user xxx, rate limit: xxxx，说明配置完成了，这样你的Jenkins就具有访问你的github的权限了。

**3. 创建一个freestyle任务**

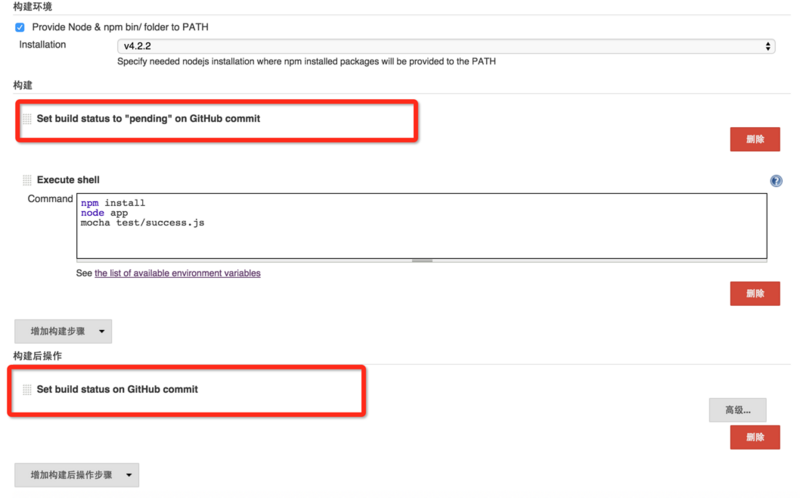
* 填写GitHub project URL￼￼￼, 也就是你的项目主页  
  [https://github.com/your\_name/your\_repo\_name](https://github.com/%08your_name/your_repo_name/)



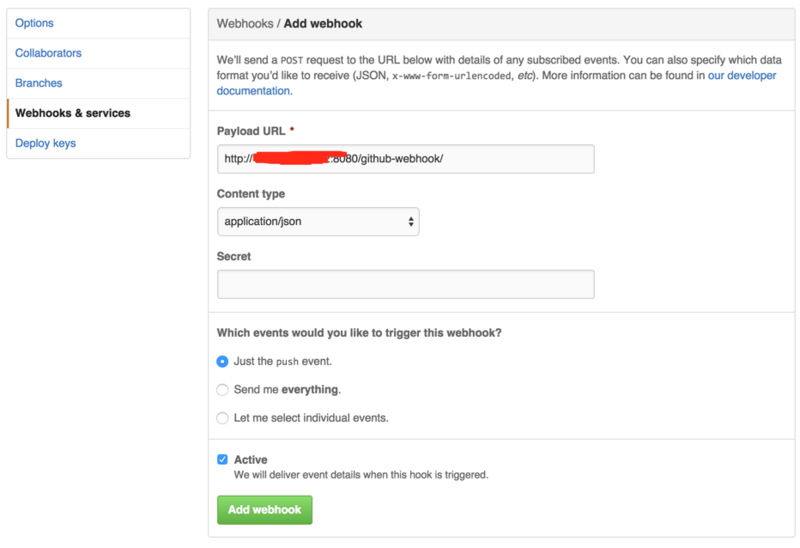
* 配置源码管理  
  
  1. 填写项目的git地址, eg. <https://github.com/>your\_name/your\_repo\_name.git
  2. 添加github用户和密码
  3. 选择githubweb源码库浏览器，并填上你的项目URL，这样每次构建都会生成对应的changes，可直接链到github上看变更详情
* 构建触发器勾选Build when a change is pushed to GitHub，这样该仓库的每一次push或者pull request都会触发build



* 配置构建步骤随后配置构建环境、构建步骤和构建后步骤

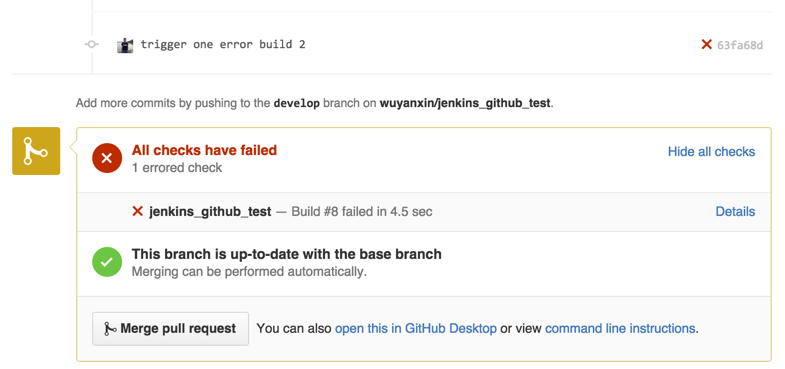
安装了Github Plugin之后在构建步骤和构建后操作会多两个设置，用于在构建时和构建后同步构建状态到Github的，后面有效果图  


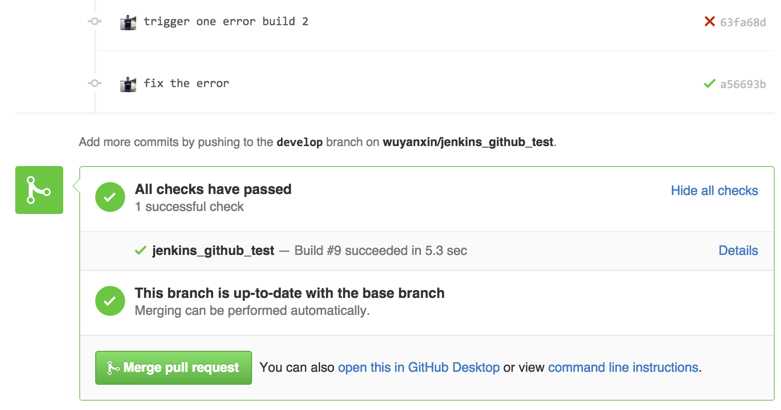
**4. 配置Github仓库的Webhook**

仓库的创建人在仓库的Settings >> Webhooos & services添加  
我们只需要push事件触发就可以了，选中Just the push event  
点击Add webhook  
yes，与github集成的Jenkins CI环境就配置好了  


**5. 效果**

每次push都会触发一次build，pull request的话还会在该界面直接显示build结果





**最后**

整个环境终于搭好了，中间遇到了蛮多大坑小坑的，有些记录了下来，后续整理好再发上来。  
这段时间学习Jenkins收获蛮多的，只是到现在也只学了些皮毛，写出来的东西也颇有些晦涩。  
接下来要做的实验是通过Jenkins实现自动远程部署。