码工面试

02-13-2015

我整理了一下我准备复习面试的资料 有些是网上抄的 有些是我自己的笔记 希望有用
格式可能有些丑陋 因为都是从evernote里拷贝过来的 将就看看
我的github: https://github.com/avocadoccme 大部分是python写的 python比较像伪代码 适合看个思路

如果大家有任何问题和建议 欢迎写邮件给我 ninjacoder.california@gmail.com 本人水平有限 但是希望可以帮助到国人

Section I. Data Structures and Algorithms

总共大概800个

- cc150 cracking coding interview 第五版 150个
- leetcode 200个
- http://www.fgdsb.com/categories/ 上面有非leetcode的面经题大概150个
- epi elements of programming interview 300个 我没看
- http://n00tc0d3r.blogspot.com/?view=sidebar 写的不错 可以看看
- http://www.geeksforgeeks.org/ 上面很多解释不错 可以看看
- 买买提和careercup上面经可以看 careercup上面经比较全而且大部分有别人做的答案

Section II. OO Design

我主要就是看了看cc150上ood那章
有本书叫headfirst design pattern 据说不错 不过我没来得及看
常见的design pattern singleton和factory method可以写一写

Section III. System Design

附录是我从买买提上拷过来的别人的总结帖子 值得看一遍
以下是我自己的笔记 文档最后还有我画的框图 可以参考

————————————————
design feeds
http://highscalability.com/blog/2013/10/28/design-decisions-for-scaling-your-high-traffic-feeds.html
https://github.com/tschellenbach/Stream-Framework
http://getstream.io/
Version 1. PostgreSQL

select * from love where user_id in (…)
100M loves 1M users
Version 2. Redis
Version 3. Cassandra

Q1. denormalized [cassandra] vs normalized [redis]
Q2. use pull instead of push for high profile producers
Q3. only push to active consumers
Q4. priorities
Q5. Redis vs Cassandra

ACID
atomicity: a transaction is “all or nothing”
consistency: transaction brings one valid state to another, meets all defined rules
isolation: cocurrent transactions have same result as executed serirally
durability: non-vaolatile memory
CAP
consistency
availability
parition tolerance
BASE
basically available
soft state
eventual consistency

----
facebook
@2014: 864 million daily active users
@2010: 5 billion pieces of content shared per week
@2008: 120M+ active users
50B+ page views per month
10B+ photos
1B+ connections
50K+ platform apps
400K+ app developers
lamp: php, memcache, mysql
thrift, scribe, operational data store
newsfeed

1. newsfeed
2. chat
3. typeahed
future: user-produced content

(friend-of-friend
object-of-friend
global/memcache) -> aggregator [merging ranking returning]
option 1: trie<name, id> pointers waster space
option 2: sorted vector of name, id. binary search shipped
option 3: brute force. forward index
option 4: filtered force. tiny bloom filter
4. messge
messages
chats
emails
sms
5. POI given phone location
6. second/minute/hour/day counter
7. photo storage
8. timeline
9. memcache
10. tinyurl
11. trending topics
12. copy one file to multiple servers
13. dynamo/gfs/big table

----

tweets
6000 tweers per second

design a url shortening service, like bit.ly

1. constraints and use cases
2. abstract design
3. understanding bottlenecks
4. scaling your abstract design

Scalability:

1. First golden rule for scalability:
Every server contains exactly the same codebase and does not store any user-related data, like sessions or profile pictures, on local disc or memory.

2. Choose database:
2.1 SQL master-write slave-read

2.2 NoSQL
Column: Cassandra HBase
Document: couchDB mongoDB
Key-Value: Dynamo Redis MamcacheDB

3. Cache
3.1 Cache database queries
3.2 Cache objects

4. Asynchronism

In-memory cache (Key-Value):
Memcached
Redis

Remote-server automation tool:
Capistrano # update codebase in a cluster

Asynchronism:
RabbitMQ

distributed system introduction:http://www.aosabook.org/en/distsys.html
google:http://www.mitbbs.com/article_t/JobHunting/32289373.html
facebook:http://blog.csdn.net/sigh1988/article/details/9790337
amazon:
http://www.mitbbs.com/article_t/JobHunting/32382937.html
http://www.mitbbs.com/article_t/JobHunting/32623615.html
http://www.mitbbs.com/article_t/JobHunting/32834277.html

Design distributed systems for:
availability
performance
reliability
scalability
manageability
cost
Make read faster
caches: local global distributed [memcached]
proxies: collapsed forwarding [squid varnish]
indexes
load balancers [HAProxy]
queues [RabbitMQ]

nonsql vs sql
performance
scalability
flexibility
complexity
functionality

key concepts
consistent hashing
object versioning
vertical scaling
horizontal scaling
caching
load balancing
database replication
database partitioning
avoid single point of failure
database sharding
distributed hash table
eventual consistency vs strong consistency
read heavy vs write heavy
sticky session
structured data vs unstructured data
service-oriented architecture
http://www.thoughtworks.com/insights/blog/nosql-databases-overview
http://markorodriguez.com/2011/04/30/putting-and-getting-data-from-a-database/
http://blog.andreamostosi.name/2013/03/how-it-works-facebook-part-1/
http://blog.andreamostosi.name/2013/03/how-it-works-facebook-part-2/

facebook
mysql
memcached
haystack

Section IV. Real Interview Questions

这里贴下我面flag的真题 做个参考

- **Amazon Phone**

Binary tree level order traversal

- **Amazon Onsite**

Round 1: OOD parking lot
Round 2: word ladder
Round 3: behavior
Round 4: behavior
Round 5: remove duplicates in linked list
permutation

- **Google Phone**

Find the longest increasing string in 2D matrix

ABWJW
VERTG
BEFCW

- **Google Onsite**

Round 1. find perimeter of maximum area in a matrix
find all subarray sum up to zero
Round 2. evaluate arithmetic tree
Round 3. given two array, search numbers in second one in the first
given x = [1,2,3,4] y = ax^2+bx+c return y = [?,?,?,?] sorted
Round 4. how to win the game definitely
Round 5. maximum continuous subarray having only two distinct numbers
continental divider # did not coding since I knew it
search for second array (no need to be continuous) in the first array
does sign (full vs empty) on the locker matter if some people may forget to set it as 'full' when actually using it

- **Facebook Phone**

Q1: Move all 0's to the end of array in place
Q2: Subsets
Q3: Iterator for subsets

- **Facebook Onsite**

Round 1. behavior. Print tree from left to right
Round 2. 3sum
atof
Round 3. system design. typeahead search

Round 4. given dictionary having 1 million words and a set of letters, find all the words in the dictionary that only uses letters within the set (do not need to use up all the letters).
example: given dictionary {"RAT", "AA", "BAT", "AAA"...} set is "AART"
return ["RAT", "AA"]

- **Linkedin Phone**

""" In "the 100 game," two players take turns adding, to a running
total, any integer from 1..10. The player who first causes the running
total to reach or exceed 100 wins.
What if we change the game so that players cannot re-use integers?
For example, if two players take turns drawing from a common pool of integers
1..15 without replacement until they reach a total >= 100.

This problem is to write a program that determines which player would win
with ideal play.

Write a procedure, "Boolean canIWin(int maxChoosableInteger, int desiredTotal)",
which returns true if the first player to move can force a win with optimal play.

Your priority should be programmer efficiency; don't focus on minimizing
either space or time complexity.
"""

3, 5 -> True
3, 4 -> False

I pick 1. Opponent picks either 2 or 3. In either case, total < 5. I pick remaining number, and total now excedes 5. I win.

{1,2,3}

```
def __init__(self):
    self.flag = True

# takes in two integers
# returns a boolean

# given list
# return list of equal length
# given [1, 2, 3, 4]
# return [24, 12, 8, 6]
```

Appendix I. System Design （蓝字都是来源于买买提 直接拷过来的方便大家看）

以下都是从买买提上拷过来的 我都看了一遍

发信人: flamingos (flamingos), 信区: JobHunting
标　题: 我的System Design总结
发信站: BBS 未名空间站 (Mon Sep 8 02:49:55 2014, 美东)

我的面试也结束了 因为知道FLAG这类公司都会问到System Design的问题 所以这次面试着重准备了一下 在这里分享给大家 如果有不对或者需要补充的地方 大家可以留言

这里说的System Design和OO Design不同 System Design在FLAG以及很多大公司中主要是design scalable distributed systems 这里只讨论如何准备这种题目

== 入门 ==
对于0基础的同学们 下面的资料可以按顺序开始看
1. http://www.hiredintech.com/app#system-design
这是一个专门准备面试的网站 你只用关心system design部分 有很多的link后面会重复提到 建议看完至少一遍

2. https://www.youtube.com/watch?v=-W9F__D3oY4
非常非常好的入门资料 建议看3遍以上！
这是1里面提到的资料 是Harvard web app课的最后一节 讲scalability 里面会讲到很多基础概念比如Vertical scaling, Horizontal scaling, Caching, Load balancing, Database replication, Database partitioning 还会提到很多基本思想比如avoid single point of failure
再强调一遍 非常好的资料！

3. http://www.lecloud.net/post/7295452622/scalability-for-dummies-part-1-clones
1里面提到的 Scalability for Dummies 还算不错 可以看一遍 知道基本思想

结束语：当你结束这一部分的学习的时候 你已经比50%的candidate知道的多了(因为很多人都不准备 或者不知道怎么准备system design) 恭喜:)

== 进阶 ==
这一部分的资料更加零散 每个看的可能不一样 但是你每多看一篇文章或者一个视频 你就比别人强一点
这部分你会遇到很多新名词 我的建议是每当你遇到一个不懂的概念时 多google一下 看看这个概念或者技术是什么意思 优点和缺点各是什么 什么时候用 这些你都知道以后 你就可以把他运用到面试中 让面试官刮目相看了

4.
http://highscalability.com/blog/2009/8/6/an-unorthodox-approach-to-database-design-the-coming-of-the.html
Database Sharding是一个很重要的概念 建议看一看

5. http://highscalability.com/all-time-favorites/
这个里面会讲到很多非常流行的网站架构是如何实现的 比如Twitter, Youtube, Pinterest, Google等等 我的建议是看5-6个 然后你应该已经建立起了一些基本的意识 还有知道了某些技术和产品的作用和mapping 比如说到cache你会想到memcached和Redis 说到
load balancer你会想到 Amazon ELB, F5一类的

6. http://www.infoq.com/
5里面很多的文章都会有链接 其中有很多会指向这个网站 这里面有很多的tech talk 很不错 可以看看

7. https://www.facebook.com/Engineering/notes
Facebook非常好的技术日志 会讲很多facebook的feature怎么实现的 比如facebook message:https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919 建议看看 尤其是准备面facebook的同学
这有一个facebook talk讲storage的https://www.youtube.com/watch?v=5RfFhMwRAic

8. 一些国内网站上的资料
http://blog.csdn.net/sigh1988/article/details/9790337
http://blog.csdn.net/v_july_v/article/details/6279498

9. 最后一些概念很有用 都是我再看这些资料的时候发现的 如果你没有遇到或者查过 建议查查
Distributed Hash Table
Eventual Consistency vs Strong Consistency
Read Heavy vs Write Heavy
Consistent Hashing
Sticky Sessions
Structured Data(uses DynamoDB) vs Unstructured Data(uses S3)http://smartdatacollective.com/michelenemschoff/206391/quick-guide-structured-and-unstructured-data http://stackoverflow.com/questions/18678315/amazon-s3-or-dynamodb

10 给有兴趣深入研究的人看的
Mining Massive Datasets --讲很多big data和data mining的东西
Big Data: Principles and best practices of scalable realtime data systems --twitter的前员工讲述如何处理实时数据

10 凌乱的资料 随便看看吧

http://highscalability.com/blog/2013/10/28/design-decisions-for
== 小结＝＝
看多了以后 你的最终目标应该是心里有了一个大框架 一个基本的distributed system 是怎么搭起来的 然后心里有很多if condition 如果要是满足这个条件 我应该用什么技术 比如如果read heavy那么用cache会提升performance之类的 同时知道应该避免什么东西 比如避免single point of failure 再比如时间和空间的tradeoff在read heavy的时候应该倾向于时间 Write heavy的时候倾向于空间等等

你总结出来的和我总结出来的大框架和if conditions肯定不完全一样 但因为system design本来就是一个open ended question 所以不用害怕 能够自圆其说 就不会有问题

最后 本文纯属抛砖引玉 如果有大牛发现有错误或者有补充 欢迎留言 大家一起讨论

== FAQ ==
1. New Grad需要看System Design么？

答案是it depends. 有的公司会考system design 有的公司只考到OO design 有的公司压根不考 当然 考到的公司对new grad的期望值会稍微低一点 但是 你有这么一个机会能让你gain leverage over other candidates why not? 为什么要让自己在面试前害怕面试官出system design的题目呢?

这里原帖地址: http://www.mitbbs.com/article_t/JobHunting/32492515.html
以下为转载内容
==========我是分割线==========
稍微总结一下

1. 入门级的news feed
http://www.quora.com/What-are-best-practices-for-building-somet
http://www.infoq.com/presentations/Scale-at-Facebook
http://www.infoq.com/presentations/Facebook-Software-Stack
一般的followup question是估算需要多少server
另外这个帖子有讨论
http://www.mitbbs.ca/article_t/JobHunting/32463885.html
这篇文章稍微提到要怎么approach这种题，可以稍微看看
http://book.douban.com/reading/23757677/

2. facebook chat,这个也算是挺常问的
http://www.erlang-factory.com/upload/presentations/31/EugeneLet
https://www.facebook.com/note.php?note_id=14218138919
http://www.cnblogs.com/piaoger/archive/2012/08/19/2646530.html
http://essay.utwente.nl/59204/1/scriptie_J_Schipers.pdf

3. typeahead search/search suggestion，这个也常见
https://www.facebook.com/video/video.php?v=432864835468
问题在这个帖子里被讨论到，基本上每个问题，在视频里都有回答
http://www.mitbbs.com/article_t/JobHunting/32438927.html

4. Facebook Messaging System(有提到inbox search, which has been asked before）
messaging system就是一个把所有chat/sms/email之类的都结合起来的一个系统
http://www.infoq.com/presentations/HBase-at-Facebook
http://sites.computer.org/debull/A12june/facebook.pdf
http://www.slideshare.net/brizzzdotcom/facebook-messages-hbase/
https://www.youtube.com/watch?v=UaGINWPK068

5. 任给一个手机的位置信号(经纬度)，需要返回附近5mile 的POI
这个这里有讨论，这题貌似nyc很爱考...
http://www.mitbbs.ca/article0/JobHunting/32476139_0.html

6. Implement second/minute/hour/day counters
这题真不觉得是system design，但万一问道，还是要有准备，貌似在总部面试会被问道....
这个帖子有讨论
http://www.mitbbs.com/article_t/JobHunting/32458451.html

7. facebook photo storage，这个不太会被问起，但是知道也不错
https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Beaver.pdf
https://www.facebook.com/note.php?note_id=76191543919

8. facebook timeline,这个也不太是个考题，看看就行了
https://www.facebook.com/note.php?note_id=10150468255628920
http://highscalability.com/blog/2012/1/23/facebook-timeline-bro

除了这些，准备一下这些题目
implement memcache
http://www.adayinthelifeof.nl/2011/02/06/memcache-internals/

implement tinyurl（以及distribute across multiple servers)
http://stackoverflow.com/questions/742013/how-to-code-a-url-sho

determine trending topics(twitter)
http://www.americanscientist.org/issues/pub/the-britney-spears-
http://www.michael-noll.com/blog/2013/01/18/implementing-real-t

copy one file to multiple servers
http://vimeo.com/11280885

稍微知道一下dynamo key value store，以及google的gfs和big table

另外推荐一些网站
http://highscalability.com/blog/category/facebook
这个high scalability上有很多讲system design的东西，不光是facebook的，没空的话，就光看你要面试的那家就好了..
facebook engineering blog
http://www.quora.com/Facebook-Engineering/What-is-Facebooks-arc
http://stackoverflow.com/questions/3533948/facebook-architectur

其他家的
http://www.quora.com/What-are-the-top-startup-engineering-blogs

==========
在说说怎么准备这样的面试
首先如果你连availability/scalability/consistency/partition之类的都不是太有概念的话，我建议先去wikipedia或者找一个某个大学讲这门课的网站稍微看一下，别一点都不知道
这个链接也不错
http://www.aosabook.org/en/distsys.html

如果你这些基本的东西都还知道，那么我觉得你就和大部分毫无实际经验的人差不多一个水平...
能做的就是一点一点去准备，如果你还有充足的时间的话，建议从你面试的那家公司的engineering blog看起，把人家用的technology stack/product都搞清楚，然后在把能找到的面试题都做一遍呗....我们做coding题说白了不也是题海战术...而且你如果坚持看下去，真的会看出心得，你会发现很多地方都有相同之处，看多了就也能照葫芦画瓢了...

再有就是面试的时候应该怎么去approach这种题，我说说我的做法
1. product spec/usage scenario 和面试者confirm这个东西到底是做什么的
可以先列出来几个major functionality，然后有时间的话，再补充一些不重要的
把你想的都写下来

2. define some major components
就是画几个圈圈框框的，每个发表一番您的高见....然后讲他们之间怎么interact

以上是question specific的东西，
这个讲完了，我们可以讲一些每道题都是用的，比如说
怎么scale/怎么partition/怎么实现consistency，这些东西，可以套用到任何题上

当然了，我们遇到的题和解题的方法可能都有些出入，不见得每道题有一个路数下来，最重要的是，讲题的时候要有条理，画图要清楚，保持和面试官的交流，随时问一下人家的意见。

我能想到的就这么多，欢迎大家交流，希望大家都能找到理想的工作.