常见的面试点

（红色必问，绿色很重要）

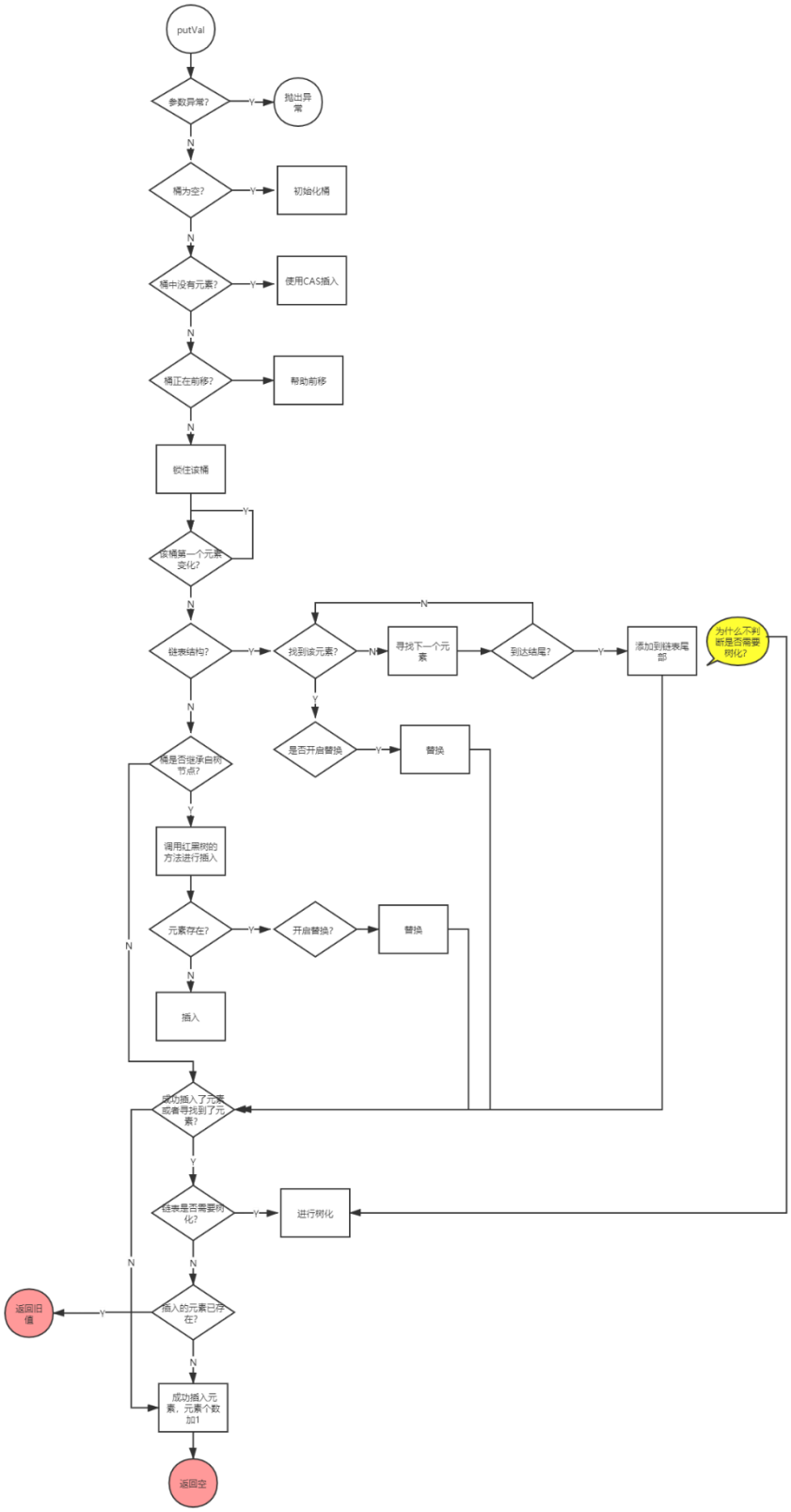
源码（http://cmsblogs.com/）这个网站讲的很不错

1. HashMap（面试官一般会叫你直接将 HashMap，最好把下面的都讲了，给面试官留下好印象）
   a) putVal
   b) resize
   c) rehash
   d) 树化为什么是 8（泊松定理）
      https://www.jianshu.com/p/9b913eabd585
      这是我的总结。流程图有些位置画的不对，最好自己去画一下，增强记忆
   e) 对链表进行插入的时候，1.7 使用的是尾插法（会造成死循环），1.8 是使用的尾插法；但是在并发的情况下还是不要去使用 HashMap，最好是使用 HashTable（加锁版的 HashMap）或者 ConcurrentHashMap（锁的粒度校小）

   这段高亮的话可以直接说出来，之后面试官很有可能问你 ConcurrentHashMap

2. ConcurrentHashMap
   a) 主要是 putVal 的时候如何加锁

   还是推荐自己去看，去画流程图


putVal
参数异常?
Y
抛出异常
N
桶为空?
Y
初始化桶
N
桶中没有元素?
Y
使用CAS插入
N
桶正在前移?
帮助前移
N
锁住该桶
该桶第一个元素变化?
Y
N
链表结构?
Y
找到该元素?
N
寻找下一个元素
到达结尾?
Y
添加到链表尾部
为什么不判断是否需要树化?
N
Y
是否开启替换
Y
替换
N
桶是否继承自树节点?
Y
调用红黑树的方法进行插入
元素存在?
Y
开启替换?
替换
N
插入
N
成功插入了元素或者寻找到了元素?
Y
链表是否需要树化?
Y
进行树化
N
N
返回旧值
Y
插入的元素已存在?
N
成功插入元素，元素个数加1
返回空

MySQL

这个不好详细说，注意下面的问题和关键词，Mysql 范围很广

1. 为什么使用 MySQL（免费，技术成熟，二维表），以及和其他数据库的一个区别 MongoDB,PostPreSQL– **这个问题不是经常问，但是就所有的为什么使用这个技术栈都必须知道**。
2. 为什么使用索引
   a) 主要一点加快查询速度
   b) 小表加索引不一定快（10w 数据量有明显差别，我自己测试的）
      i. 为什么不一定快，MySQL 查询很怕随机 IO（因为基于磁盘查询），不使用索引（不是不加索引，没有索引的话 InNoDB 会为唯一键加索引）的话，直接全表查询，使用 B+树作为索引，在遍历非根节点的时候都是随机 IO。
   c) 可以去说道使用索引的坏处，尝试扩展，但是别给自己挖坑，说一下自己不会的
3. B 树和 B+区别（可以自己再上网查一下）
   a) B+树所有的数据都存在叶子节点（为了存储更多的子节点信息）
   b) B 树离根节点越近，查询速度越快，B+树都一样
   c) B+树的叶子节点是条链表，天然排序，所以直接返回所有的数据比 B 树快
   d) B 树，可以了解，作用不大
4. InNoDB 和 MYISAM 的区别，关于事务的展开说

比较常用的是 MyISAM 和 InnoBD

	MyISAM	
构成上的区别：	每个 MyISAM 在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。 .frm 文件存储表定义。 数据文件的扩展名为.MYD (MYData)。 索引文件的扩展名是.MYI (MYIndex)。	基于磁盘的资源是 InnoD InnoDB 表的大小只受限 2GB
事务处理上方面	MyISAM 类型的表强调的是性能，其执行数度比 InnoDB 类型更快，但是不提供事务支持	InnoDB 提供事务支持事务 库功能
SELECT UPDATE,INSERT ，Delete 操作	如果执行大量的 SELECT，MyISAM 是更好的选择，不是绝对	**1.**如果你的数据执行大量 **面的考虑，应该使用 In** **InnoDB 不会重新建立表** TABLE FROM MASTER 操 法是首先把 InnoDB 表改 InnoDB 表，但是对于使用 的表不适用
AUTO_INCREMENT	每表一个 AUTO_INCREMEN 列的内部处理。 **MyISAM** 为	如果你为一个表指定 AU

	MyISAM	
的操作	INSERT 和 UPDATE 操作自动更新这一列。这使得 AUTO_INCREMENT 列更快（至少 10%）。在序列顶的值被删除之后就不能再利用。(当 AUTO_INCREMENT 列被定义为多列索引的最后一列，可以出现重使用从序列顶部删除的值的情况）。 AUTO_INCREMENT 值可用 ALTER TABLE 或 myisamch 来重置 对于 AUTO_INCREMENT 类型的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM 表中，可以和其他字段一起建立联合索引 更好和更快的 auto_increment 处理	InnoDB 表句柄包含一个 用在为该列赋新值。 自 而不是存在磁盘上 关 **AUTO_INCREMENT** 列在
表的具体行数	select count() *from table,MyISAM 只要简单的读出保存好的行数，注意的是，当 count()*语句包含 where 条件时，两种表的操作是一样的	InnoDB 中不保存表的具 count() from table 时，In 少行
锁	表锁	提供行锁(locking on row l 加锁读取(non-locking rea 锁也不是绝对的，如果在 定要扫描的范围，InnoDB set num=1 where name li

5. 关键词部分
   a) 回表（减少回表），聚簇索引，MVCC（redo log—innodb 实现事务的原理和 undo log），最左匹配原则，覆盖索引，锁，行级锁，表加锁,悲观锁，乐观锁，意向锁
      i. InNoDB 的锁实现
         1. recordLock,GapLocks，next_key_lock
6. 事务隔离级别

JVM

1. GC 算法

复制算法（适用于新生代）

标记清除算法 - （适用于老年代）

会有碎片，只有 CMS 的老年代使用了这个算法，所以 CMS 在 JDK9 倍淘汰了

标记整理算法 - （适用于老年代）

分代收集（上面只是说算法并没说到底哪个代使用，这个算法就是说组合两种不同年代的算法）

3. 确定可回收对象
   a) 引用计数法
   b) 可达性分析，GCroots(OOP MAP)
4. CMS 和 G1 的区别
   a) 主要的步骤不同

b) 关键词 Rememberd Set 减少 stop the world

3.类加载机制

加载，验证，准备，解析，初始化

4.双亲委派

c) 为什么使用双亲委派

i. 沙箱机制，防止核心类被修改

ii. 避免重复加载

d) 为什么要打破双亲委派，怎么打破

i. 为什么？因为例如多个 war 包，相同的全限定名，功能不同

ii. 怎么打破，使用不同的类加载器对其进行加载（在 JVM 当中确定唯一一个类的方式是通过 全限定名+类加载器）

5.JVM 调优

6.JVM 的内存布局

7.引用的四种类型

Redis

为什么使用 Redis（可以加上和 MemCached 的对比）

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1)；

2、数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的；（说出至少五中数据结构，基本的加一下扩展的 --https://blog.csdn.net/weixin_33961829/article/details/91923093?depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1&utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1 ）

> 位图最好了解一下，可以用于布隆过滤器，解决缓存穿透问题（cache null 也可以，常见的缓存问题要去看，缓存穿透，雪崩，以及服务降级）

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

> 使用单线程，但是为什么可以同时接收多个请求，就是使用到了多路复用，详细的去看 IO 多路复用，‘

>

> select,poll,epoll,以及两种触发方式 LT 和 ET.

4、使用多路 I/O 复用模型，非阻塞 IO；（建议看一下 BIO，NIO，AIO 原理，用不了多久，大致流程必须清楚）

> NIO 关键词 selector,buffer,channel

5、可持久化策略，AOF，RDB，AOF 重写原理。

AOF -- 日志

RDB -- 快照

以及 SAVE 和 BGSAVE

积压缓冲区

6、集群（不怎么问，反正我没怎么问）

Spring

为什么使用 SpringBoot

1. SpringBoot 内嵌 Tomcat,可以更换成 Jetty`、 `Undertow`容器（无需部署 war 文件）
2. 提供的`starters` 简化构建配置
3. 尽可能自动配置`spring`应用。
4. 0XML
5. 提供生产就绪型功能，如指标，健康检查和外部配置(SpringBootTest,Acutor)
6. 1. `Spring`使用`web.xml` 或`SpringServletContainerInitializer`作为其引导入口点。
   2. `Spring Boot`仅使用`Servlet 3`功能来引导应用程序
7. 打包部署，Spring Boot 更加简化

关键性注解 -- >SpringBoot 为什么能自动装配的原因，涉及到源码

@SpringBootApplication -> @EnableAutoConfiguration(@Import(AutoConfigurationImportSelector.class)) -> ImportSelector ->spring.factories

Spring 的传播机制

事务的传播性一般在事务嵌套时候使用，比如在事务 A 里面调用了另外一个使用事务的方法，那么这俩个事务是各自作为独立的事务执行提交，还是内层的事务合并到外层的事务一块提交那，这就是事务传播性要确定的问题。下面一一介绍比较常用的事务传播性。

所谓事务传播机制，也就是在事务在多个方法的调用中是如何传递的，是重新创建事务还是使用父方法的事务？父方法的回滚对子方法的事务是否有影响？这些都是可以通过事务传播机制来决定的。

1.2 PROPAGATION_REQUIRED（自己可能会开启）

> 默认机制，调用方法有事务则加入，没有则自己开启。

1.3 PROPAGATION_REQUIRES_NEW（自己一定会开启）

> 不管调用方法是不是有事务，被调用方法都会开启一个事务

1.4 PROPAGATION_SUPPORTS（自己不会开启，但会加入）

> 有则加入，没有就没有

1.5 PROPAGATION_NOT_SUPPORTED（都不起作用）

> 有没有，都没有

1.6 PROPAGATION_NEVER（不允许）

> 不允许调用方法有事务，有的话就报错

1.7 PROPAGATION_MANDATORY 强制要事务，没有报错

> 调用事务中有方法，加入；没有，报错

1.8 PROPAGATION_NESTED（鸟巢）保护调用方法

>调用方法的事务失败时候，回滚调用方法的事务，但是不回滚被调用方法的事务。

因为 Spring 源码基本没问我，不代表不考，自己是情况而定

OS

线程和进程的区别
进程的通信方式
https://www.jianshu.com/p/c1015f5ffa74

计网

Cookie 和 Session 的区别，可以去说一下 Token
HTTPS，三次握手四次挥手（可以扩展说，为什么用三次和四次，人打招呼和说拜拜。有例子最好），OSI7 层协议，网络安全（SQL 注入【可以去看的我简书，写的还不错】，Cookie 和 Session 的安全问题【Cookie 的两个关键字段 – secure 和 HttpOnly】，DNS 攻击 – 有多种，同源策略，csrf）

Linux

常见的排查命令
Top,netstat,sed,awk,grep,ps

总结

时间少的行情况下多刷题，面大厂要多问自己为什么