# Java

**1.多个线程同时读写，读线程的数量远大于写线程，你认为应该如何解决并发的问题？你会选择加什么样的锁？**

无论读或写操作直接加普通锁也是可以解决并发问题, 但是效率太低, 可以使用实现了ReadWriteLock接口的ReentrantReadWriteLock锁,

读的时候使用ReadLock, 写的时候使用WriteLock. 多个读锁之间不互斥, 提高读的效率, 但是读和写互斥, 多个写也互斥.

这样当有写操作时, 其他的所有读写线程才阻塞, 如果没有写操作时, 多个读线程不会互斥, 从而提高效率.

**2. JAVA的AQS是否了了解，它是干嘛的？**

AQS是AbstractQueuedSynchronizer的缩写, 意思就是抽象的基于队列的同步器, 它是concurrent包的基础和核心, 用于替代synchronized锁机制.

synchronized锁的实现太过简单, 只能单纯地保证一段代码同一时刻, 只能在一个线程栈中执行, 其他线程栈处于阻塞状态,

线程间的通信也比较单一, 不利于大型的多线程应用中对于线程的精细高效的控制.

AQS是一个框架, 主要是基于原子数据操作的同步机制, AQS是一个抽象类, 但是它的内部的关键方法已经实现了, 如果用户想要实现自定义锁机制, 就可以继承这个

抽象类AbstractQueuedSynchronizer, 并实现其中的方法, 从而可以自己处理锁的处理机制, 线程间的互斥和共享操作都可以自己控制, 其中的关键方法如下

这个方法会被子类的锁中的lock方法间接调用到, 并且其中的方法调用tryAcquire(arg),是会抛出异常的方法, 不能直接调用,

这个方法就是由具体子类提供实现逻辑,

public final void acquire(int arg) {

if (!tryAcquire(arg) &&

acquireQueued(addWaiter(Node.EXCLUSIVE), arg))

selfInterrupt();

}

这个方法会被子类的锁中的unlock方法间接调用到, 并且其中的方法调用tryRelease(arg),是会抛出异常的方法, 不能直接调用,

这个方法就是由具体子类提供实现逻辑

public final boolean release(int arg) {

if (tryRelease(arg)) {

Node h = head;

if (h != null && h.waitStatus != 0)

unparkSuccessor(h);

return true;

}

return false;

}

AQS的典型实现是ReentrantLock, 这个类的内部类中的实现如下:

// 默认的多线程执行方式是非公平抢占式的, 获取锁的操作就是修改锁的状态值到指定值

final boolean nonfairTryAcquire(int acquires) {

final Thread current = Thread.currentThread();

int c = getState(); // 获取锁内部状态

if (c == 0) {

if (compareAndSetState(0, acquires)) { // 原子设置锁状态为参数指定的状态, 表明获取锁成功

setExclusiveOwnerThread(current);

return true;

}

}

else if (current == getExclusiveOwnerThread()) {

int nextc = c + acquires;

if (nextc < 0) // overflow

throw new Error("Maximum lock count exceeded");

setState(nextc);

return true;

}

return false;

}

// 释放锁的操作, 和获取锁是逆操作, 把锁状态值还原.

protected final boolean tryRelease(int releases) {

int c = getState() - releases;

if (Thread.currentThread() != getExclusiveOwnerThread()) // 必须由当前线程处理

throw new IllegalMonitorStateException();

boolean free = false;

if (c == 0) {

free = true;

setExclusiveOwnerThread(null);

}

setState(c);

return free;

}

**3. 除了synchronized关键字之外，你是怎么来保障线程安全的？**

使用可重入锁ReentrantLock, 创建对象, 并调用方法lock, 锁定后面的代码, 防止其他线程执行. 然后在不需要同步的时候,再调用unlock方法解锁.

**4. 什么时候需要加volatile关键字？它能保证线程安全吗？**

在多线程程序中, 为了提高效率, 有的时候子线程会把主内存中的一些数据在子线程私有内存中复制一份副本, 这样就可以快速读写,

数据读写完成后再把私有数据同步刷新到主内存中, 但是这样带来一个问题,

如果子线程在修改了私有数据后没有来得及刷新主存, 就结束了, 就会导致数据不同步.

volatile关键字的作用是提醒子线程不要从主内存中复制副本, 子线程直接访问主内存中, 而不制作副本.

它不能保证线程安全, 它只保证没有主内存的副本, 不能控制多个线程同时访问主内存, 还是会发生线程安全问题.

**5. 线程池内的线程如果全部忙，提交一个新的任务，会发生什么？队列全部塞满了之后，还是忙，再提交会发生什么？**

/\*\*

\* Executes the given task sometime in the future. The task

\* may execute in a new thread or in an existing pooled thread.

\*

\* If the task cannot be submitted for execution, either because this

\* executor has been shutdown or because its capacity has been reached,

\* the task is handled by the current {@code RejectedExecutionHandler}.

\*

\* @param command the task to execute

\* @throws RejectedExecutionException at discretion of

\* {@code RejectedExecutionHandler}, if the task

\* cannot be accepted for execution

\* @throws NullPointerException if {@code command} is null

\*/

public void execute(Runnable command) {

if (command == null)

throw new NullPointerException();

/\*

\* Proceed in 3 steps:

\*

\* 1. If fewer than corePoolSize threads are running, try to

\* start a new thread with the given command as its first

\* task. The call to addWorker atomically checks runState and

\* workerCount, and so prevents false alarms that would add

\* threads when it shouldn't, by returning false.

\*

\* 2. If a task can be successfully queued, then we still need

\* to double-check whether we should have added a thread

\* (because existing ones died since last checking) or that

\* the pool shut down since entry into this method. So we

\* recheck state and if necessary roll back the enqueuing if

\* stopped, or start a new thread if there are none.

\*

\* 3. If we cannot queue task, then we try to add a new

\* thread. If it fails, we know we are shut down or saturated

\* and so reject the task.

\*/

int c = ctl.get();

if (workerCountOf(c) < corePoolSize) { // 如果池中有空闲

if (addWorker(command, true)) // 直接添加并提交任务, 等待调度执行, 并立即返回

return;

c = ctl.get();

}

if (isRunning(c) && workQueue.offer(command)) { // 如果当前池正在运行中, 把新任务添加到队列中

int recheck = ctl.get();

if (! isRunning(recheck) && remove(command)) // 如果线程池不在运行了(表明正在关闭了), 删除任务, 并弹出任务

reject(command);

else if (workerCountOf(recheck) == 0)

addWorker(null, false);

}

else if (!addWorker(command, false)) // 如果添加队列失败, 尝试启动新线程执行任务, 如果又失败了, 则弹出任务

reject(command);

}

线程池内的线程如果全部忙，提交一个新的任务，会发生什么？队列全部塞满了之后，还是忙，再提交会发生什么？

线程池直接提交任务, 队列满了后, 创建新线程执行任务. 如果线程池关闭了, 则无法提交任务.

**6. Tomcat本身的参数你一般会怎么调整？**

Tomcat启动运行的JVM的参数可以通过CATALINA\_HOME/bin/catalina.sh(bat) 来设置, 通常设置如下:

JAVA\_OPTS="-server -Xms512m -Xmx1024m -XX:PermSize=64M -XX:MaxPermSize=256m"

-Xms是初始堆大小

-Xmx是最大堆大小

PermSize是永久区初始化大小

MaxPermSize是永久区最大值

Tomcat的服务器相关的设置在CATALINA\_HOME/conf/server.xml中

一些组件的资源访问权限设置在CATALINA\_HOME/conf/catalina.policy中

环境中要使用到的一些组件的设置在CATALINA\_HOME/conf/catalina.properties中

Tomcat的日志文件的处理的设置在CATALINA\_HOME/conf/logging.properties中

Tomcat的后台管理用户配置在CATALINA\_HOME/conf/tomcat-users.xml中

默认的web应用程序的一些基础配置信息在CATALINA\_HOME/conf/web.xml中

**7. synchronized关键字锁住的是什么东西, 在字节码中是怎么表示的？在内存中的对象上表现为什么？**

把锁对象内部的一个标记型数据设置为锁住状态,

在synchronized开始时, 字节码中执行 0xC2(194) monitorenter 指令, 指令目标是锁对象

在synchronized块执行完毕后 再执行 0xC3(195) monitorexit 指令 指令目标是锁对象

线程主要与内存中的对象中的头标记(Mark Word)标记数据交互. 这个数据占用2比特, 当此值为01时, 表明此对象未被加锁

当前线程就可以锁定此锁对象, 只需要把当前线程的栈桢指针赋值给MarkWord, 在赋值之前先把对象的Mark Word复制到栈中, 并且把Mark Word中的锁位设置为00

当线程释放锁的时候, 就先把栈中的原始对象数据复制到对象的Mark Word中, 并把Mark Word中的锁位设置为01, 表明锁对象释放了.

**8. wait/notify/notifyAll方法需不需要被包含在synchronized块中？这是为什么？**

必须包含在synchronized块中, 因为这几个方法的调用者对象必须是synchronized锁对象, 虽然任意对象都能调用这些方法, 但是只有当对象被作为了锁对象时才能真正调用,

否则会抛出异常.

**9. ExecutorService你一般是怎么用的？是每个service放一个还是一个项目面放一个？有什么好处？**

ExecutorService是线程池的接口定义, 一个项目使用一个线程池, 如果每个service都有自己的线程池,维护线程池的开销是不必要的.

使用项目级别的线程池可以更统一地管理和调度子线程. 并且框架自身已经提供好了线程池的自动管理, 更没有必要再去处理线程池的问题.

因为线程问题是比较复杂的, 一般情况下不要手工干预, 因为可能会出现各种问题, 除非是框架级别的设计师才需要考虑更加细化的线程池的设计.

# Spring

#### 你有没有用过Spring的AOP? 是用来干嘛的? 大概会怎么使用？

1.可以通过Spring的AOP实现权限验证、事务管理、日志打印等功能；创建一个切面类，在类中创建通知方法，在方法中实现具体的功能。通过注解或者配置文件的形式配置之后Spring会为我们创建代理对象并结合切入点表达式在合适的时机执行切面中的方法。

#### 2. 如果⼀个接⼝有2个不同的实现, 那么怎么来Autowire⼀个指定的实现？

通过@Qualifier注解的value属性指定实现的名字即可

#### Spring的声明式事务 @Transaction注解⼀一般写在什么位置? 抛出了异常会自动回滚吗？有没有办法控制不触发回滚?

一般写在方法上；

默认抛出了运行时异常就会回滚；

可以通过@Transaction注解中的noRollbackFor属性或者noRollbackForClassName属性设置那些异常不回滚

#### 如果想在某个Bean生成并装配完毕后执⾏自己的逻辑，可以什么方式实现？

创建Bean的后置处理器

#### SpringMVC中RequestMapping可以指定GET, POST方法么？怎么指定？

可以通过@RequestMapping注解中的method属性指定请求方式

#### SpringMVC如果希望把输出的Object(例如XXResult或者XXResponse)这 种包装为JSON输出, 应该怎么处理?

在Handler的方法上添加@ResponseBody注解，同时需要导入HttpMessageConverter接口关于处理JSON的实现类的jar包

#### 怎样拦截SpringMVC的异常，然后做⾃自定义的处理，比如打日志或者包装 成JSON

创建一个处理异常的类，在类上添加@ControllerAdvice注解，在该类中创建处理异常的方法，并在方法上添加@ExceptionHandler注解

# SpringBoot

#### SpringBoot没有放到web容器里为什么能跑HTTP服务？

使用的是嵌入式servlet容器，其实springboot的程序是放在了servlet容器中的，只不过不是我们在外部启动的方式，而是springboot程序启动的时候创建了嵌入式的servlet容器，然后加载了springmvc的DispatcherServlet

#### 2. SpringBoot中如果你想使用自定义的配置文件而不仅仅是 application.properties，应该怎么弄？

config.location属性可以指定配置文件的位置，多个可以用,隔开。springboot的所有可设置属性，包括这个属性，还可以在使用命令行参数的时候传入，来完全指定需要加载的配置文件

# MySQL

1. **如果有很多数据插入MYSQL 你会选择什么方式?**

答：（1）利用mybatis的foreach 拼接动态sql或者java中写循环拼接，将数据分组拼接成大sql，比如可以每1万行数据拼接为一个insert语句。

（2）设置Mybatis的sqlsession的ExecutorType为batch,如果用jdbc则用executeBatch。

（3）去掉表中的非主键索引。

（4）取消该表自动提交。

（5）可以利用多线程执行。

1 直接在mysql层面，可以通过写存储过程和函数构建大批量数据插入的pl/sql程序，直接在mysql数据层面自己处理。

2 mysql使用load file导入[大]批量数据

2. **如果查询很慢，你会想到的第一个方式是什么？索引是干嘛的?**

答：（1）第一个方式是建立索引

（2）索引是是帮助MySQL高效获取数据的数据结构。可以提高数据检索的效率，提高排序效率。

Mysql查询慢查询流程：

1. 如果mysql服务器变慢了，需要首先判断是CPU慢还是IO慢，需要用top命令检查出来高CPU和IO占用的情况是SQL导致还是程序导致，如果是SQL下一步。
2. 结合业务打开MySQL慢查询日志抓出执行时间超过3秒钟以上的SQL语句
3. 将抓出的语句用explain语句命令进行分析，重点看type/key/rows/extra等关键字段
4. 按照上一步的判断，可以重新新建或者调整数据与索引
5. **如果建了一个单列索引，查询的时候查出2列，会用到这个单列索引吗？**

答：会用到。

 4. **如果建了一个包含多个列的索引，查询的时候只用了第一列，能不能用上 这个索引？查三列呢？**

答：（1）查询的时候只用了第一列，能用上这个索引

（2）查三列，不能用上这个索引

查三列要看具体情况，如果这个索引刚好就是三个，那就是全部对应，自然可以全部用上。其它情况看分析。

5. **接上题，如果where条件后面带有⼀一个 i + 5 < 100 会使用到这个索引吗？**

答：不会使用到这个索引

6. **怎么看是否用到了某个索引？**

答：使用Explain分析sql，key字段是实际使用的索引

7. **like %aaa%会使用索引吗? like aaa%呢?**

答：like %aaa%不会使用索引，like aaa%能使用上索引

8. **drop、truncate、delete的区别？**

答：（1）drop 语句将删除表的结构被依赖的约束，和表数据

（2）truncate只删除表数据，不能回滚

（3）delete只删除表数据，能回滚

9. **平时你们是怎么监控数据库的? 慢SQL是怎么排查的？**

答：（1）可以通过mysql日志监控数据库

（2）慢SQL可以通过配置慢查询日志排查

10. **你们数据库是否支持emoji表情，如果不支持，如何操作?**

答：（1）mysql的utf8编码的一个字符最多3个字节，但是一个emoji表情为4个字节，所以utf8不支持存储emoji表情。

（2）可以修改mysql数据库的编码格式变为utf8mb4，utf8mb4一个字符最多能有4字节，能支持emoji表情的存储。

11. **你们的数据库单表数据量量是多少？一般多大的时候开始出现查询性能急 剧下降？**

答：（1）最大100-200万。

（2）一般500万出现查询性能急剧下降

12. **查询死掉了，想要找出执行的查询进程用什么命令？找出来之后一般你会干嘛？**

答：（1）使用SHOW PROCESSLIST查看进程

（2）找出后使用kill 【id】杀掉进程

13. **读写分离是怎么做的？你认为中间件会怎么来操作？这样操作跟事务有 什么关系？**

答：（1）读写分离可以通过写库和读库做主从备份，再加上数据库中间件（如mycat）来实现。

（2）中间件会对sql语句进行拦截，之后判断是读操作还是写操作，分别分配到读库或写库

（3）事务内部的一切操作都会走写节点，所以读操作不要加事务。

14. **分库分表有没有做过？线上的迁移过程是怎么样的？如何确定数据是正确的？**

答：（1）分库分表表可以通过mycat来实现

（2）备份所有节点数据，如迁移失败后数据恢复

进行扩容、缩容操作：表迁移信息迁移计划、执行数据迁移、清理旧节点冗余数据、校验迁移是否成功等。

（3）验证数据总量是否一致

随机抽样数据验证是否一致

迁移前后统计报表是否一致

 15. **什么是幂等？什么情况下需要考虑幂等？你怎么解决幂等的问题？**

答：（1）一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。

（2）在电商或者其他的项目中，处理重复的订单需要考虑幂等

（3）可通过插入数据的唯一索引、分布式锁等方式解决幂等的问题

# JVM

1. **你知道哪些或者你们线上使用什么GC策略略? 它有什么优势，适用于什么场景？**

1). New Generation的GC策略

Serial GC。采用单线程方式，用Copying算法。New Generation会再次被划分成Eden Space和S0、S1，S0和S1又称为From Space和To Space

Copying算法就是开辟另外一个内存空间，把扫描到可达的对象复制过去，然后把原内存空间全部清除即可。适用于存活对象较少的情况。

Parallel Scavenge。将内存空间分段来使用多线程，也是用Copying算法。

Parallel New 比Parallel Scavenge多做了与Old Generation使用CMS GC一起发生时的特殊处理。

2). Old Generation的GC策略

Serial GC。也是单线程方式，但是实现是将Mark-Sweep和Mark-Compact结合了下，做了点改进。

Mark-Sweep算法是把扫描到可达的对象都标记下来，然后把所有未标记的对象清除。但这个方法会引起内存碎片。适用于存活对象较多的情况。

Mark-Compact算法是在Mark-Sweep基础上，在把内存空间整理一下，让存储连续以消除内存碎片。

Parallel Mark-Sweep、Parallel Mark-Compact。同样也是把Old Generation空间进行划分成regions，只是粒度更细了。

CMS（Concurrent Mark-Sweep） GC。主要是为实现并发，就是算法使用的还是Mark-Sweep，对于内存碎片的问题，CMS提供了一个额外的内存碎片的整理功能，会在执行几次Full GC以后执行一次。

2. **JAVA类加载器包括几种？它们之间的父子关系是怎么样的？双亲委派机制是什么意思？有什么好处？**

有4种

1). 引导类加载器(BootstrapClassLoader) 是扩展类加载器的父加载器(不是父类), 是最底层的核心类加载器, 由C++代码写成, 属于JVM一部分.

主要加载JAVA\_HOME/lib/rt.jar中的最核心的类.

2). 扩展类加载器(ExtClassLoader) 是应用程序类加载器的父加载器(不是父类), 扩展类加载器是由java代码写的, 主要加载JAVA\_HOME/lib/ext下的jar

3). 应用程序类加载器(AppClassLoader) 是自定义类加载器的父加载器(不是父类),也是由java代码写的, 主要加载classpath中指示的目录中的类文件.

4）. 自定义类加载器, 用户可以选择自行处理class文件如何获取, 并读取其中的数据, 再调用父类的方法完成类的初始化.

双亲委派机制是指 除引导类加载器外的所有类加载器在加载类时, 都必须先把加载类的任务交给父类加载器完成, 如果父类加载器追朔到引导类加载器,

就不再向上追朔, 直接使用引导类加载器加载类, 所以,可以认为所有类的加载, 都必须要先经过引导类加载器.

如果父类加载器加载类成功,则加载完成, 并返回. 如果父类加载器加载失败, 才会把加载类的任务向下传递.

这样做的好处是核心类库的加载都是由引导类加载器完成的, 防止用户加载恶意的核心类, 因为引导类加载器只加载rt.jar中的类. 如果是其它地方的类则拒绝加载.

3. **如何自定义一个类加载器？你使用过哪些或者你在什么场景下需要一个自定义的类加载器吗？**

通常自定义类加载器需要继承URLClassLoader, 重写方法loadClass(String name), 在方法中实现如何读取类文件的内容, 并最终需要调用父类的defineClass来加载并初始化类

在一些需要热布署的应用中使用自定义类加载器, 当有某个模块需要动态启动时, 创建一个自定义类加载器, 然后加载所有相关的类, 并执行相关的代码

模块执行完毕后, 如果需要卸载, 则把类加载器所加载的所有类清空, 然后把类加器本身也清空. 这样模块的所有使用到的类就会清除.

类似于Tomcat的工作模式, 布署一个应用就是创建一个WebAppClassLoader对象, 然后由这个类加载器加载所有资源.

4. **堆内存设置的参数是什么？**

-Xms字节 初始化堆大小

-Xmx字节 最大堆大小

5. **Perm Space中保存什么数据? 会引起OutOfMemory吗？**

永久代中保存的是方法区数据, 在JVM规范中这是不一定的. 方法区中包含所有类的信息和常量池, 在JDK1.7之前, 如果由于大量的调用字符串方法intern

就会导致大量的对象创建在常量池中, 有可能导致OutOfMemory,但是在JDK1.7以后, 这个问题消除了.

6. **做gc时，一个对象在内存各个Space中被移动的顺序是什么？**

最先在Eden代中, 如果GC后仍存活, 则会将对象移动到Yong代(青年代), 如果GC后仍存活, 则会将对象移动到Old代(老年代).

7. **你有没有遇到过OutOfMemory问题？你是怎么来处理这个问题的？处理理过程中有哪些收获？**

有, 在eclipse中编译大型项目时, 如果eclipse配置的内存不够大, 就会抛出这个问题, 解决方法就是修改eclipse/eclipse.ini文件,

修改或添加参数-Xmx设置堆内存的最大空间

8. **1.8之后Perm Space有哪些变动? MetaSpace大小默认是无限的么? 还是你们会通过什么?方式来指定大小?**

1.8之后永久代取消了, 原来在其中保存的类信息数据被移动到元数据区, 元数据区是在本地内存中保存的, 不受普通GC的管理.

MetaSpace默认大小是21m, 但是它的容量是受元空间虚拟机管理的, 理论上是可以很大的, 甚至可以扩展到交换区.

也可以通过设置参数来改变默认大小 -XX:MetaSpaceSize.

也可以通过设置参数来改变最大空间 -XX:MaxMetaSpaceSize

9. **Jstack是干什么的? Jstat呢? 如果线上程序周期性地出现卡顿，你怀疑可能是gc导致的，你会怎么来排查这个问题？线程日志一般你会看其中的什么部分？**

Jstack是一个命令, 可以查看运行的java进程的内部栈情况, 每个线程都拥有各自的栈.

Jstack命令执行时, 需要加上一些选项和java进程ID

执行这个命令时, 会自动侦测进程中有无死锁, 还会显示每个线程的当前状态(运行, 阻塞)和方法调用踪迹, 还有当前线程持有的锁.

jstat可以查看虚拟机的内存管理情况, 比如查看各个代的GC工作情况, 以及GC的统计信息等

可以通过 jstat -options 查看详细的选项.

-class // 查看加载了多少类

-compiler // 查看编译了多少类

-gc // 查看GC情况

-gccapacity // 内存容量

-gccause

-gcmetacapacity // 元空间容量

-gcnew // 年轻代

-gcnewcapacity

-gcold // 老年代

-gcoldcapacity

-gcutil

-printcompilation

10. **StackOverFlow异常有没有遇到过？一般你猜测会在什么情况下被触发？如何指定一个线程的堆栈大小？一般你们写多少？**

StackOverFlow是栈溢出错误, 通常会在方法无限递归时会发生, 或者有可能方法调用层数太深(一般不太可能)

线程的堆栈大小设置可以使用选项 -Xss字节数, 一般设置50M左右就可以.

# Linux命令

1. **日志特别大只想看最后100行怎么弄? 如果想一直看日志的持续输出，用 么命令?**

答：（1）运行命令：tail -100 文件名

（2）运行命令：tail -f 文件名

1. **如果日志一边输出，一边想实时看到有没有某个关键字应该怎么弄？**

答：tail -f xxx.log | perl -pe 's/(ERROR)/\e[1;31m$1\e[0m/g'

1. **grep如果忽略略大小写应该怎么弄? 正则表达式呢？**

答：（1）grep -i

（2）(?i)、(?-i)、(?i:X)、(?-i:X)

4. **vim往下一行是什么键？往下30行呢? 跳到文件末尾一行是什么? 跳回来是什么? 向后搜索是什么?**

答：（1）往下一行是：j

（2）往下30行：30j

（3）跳到文件末尾一行：shift+g

（4）跳回来是：``

（5）向后搜索：/ 要查找的词 +n查找下一个

5. **如果有个文本，按空格作为列的分隔符，如果想统计第三列里⾯的每个单词的出现次数应该怎么弄？**

答：awk '{sum[$3]+=1}END{for(i in sum)print i"\t"sum[i]}'  test.txt

6. **如果把上面的出现次数排个序应该怎么弄? 想按照数字本身的顺序而不不是字符串的顺序排列怎么弄？**

答：（1）awk '{sum[$3]+=1}END{for(i in sum)print i"\t"sum[i]}' abc.txt | sort -k1

（2）awk '{sum[$3]+=1}END{for(i in sum)print i"\t"sum[i]}' abc.txt | sort -k2

7. **Linux环境变量是以什么作为分隔符的？环境变量通过什么命令设置？**

答：Linux环境变量是 : 作为分隔符的，环境变量通过export设置

8. **给某个文件设置权限比如设置为644 是用什么命令？这个6是什么意思？**

答：（1）设置权限的命令为：chmod 644

（2）文件所有者有读权限（r），写权限（w）

9. **Linux下面如果想看某个进程的资源占用情况是怎么看的？系统load大概 指的什么意思？你们线上系统load一般多少？如果一个4核机器，你认为多少load是比较正常的？top命令里面按一下1会发生什么?**

答：（1）top命令

（2）系统load：系统CPU繁忙程度的度量，即有多少进程在等待被CPU调度

（3）单核机器<=1

（4）4核机器<=4比较正常

（5）top命令里按1，会展开各个CPU的使用率情况

10. **top命令里⾯，有时候所有进程的CPU使用率加起来超过100%是怎么回事？**

答：这里显示的所有的cpu加起来的使用率，说明你的CPU是多核，你运行top后按1看看，可以显示每个cpu的使用率，top里显示的是把所有使用率加起来

11. **还有哪些查看系统性能或者供你发现问题的命令？你一般是看哪个参数？**

答：mpstat -P ALL、vmstat、ps -aux

一般看CUP的使用情况，内存使用情况

12. **想看某个进程打开了哪些⽹络连接是什么命令？里面连接的状态你比较 关心哪几种？**

答：netstat命令

连接状态：LISTEN（监听）、ESTABLISHED（连接已经建立）、

## 偏题

#### 1. 有没有做过Linux系统参数方面的优化，大概优化过什么？

|  |
| --- |
| **Linux安装后，可以调整的一些参数(大概)：**  **1、优化开机启动项**  A：执行ntsysv命令，然后弹出窗口进行设置；  B：执行setup --> system service,弹出窗口进行设置；  **2、linux中文支持及显示**  # echo 'LANG="ZH\_cn.UTF-8"' >/etc/sysconfig/i18n  # source /etc/sysconfig/i18n  **3、设置Linux服务器时间同步**  # /usr/sbin/ntpdate time.nist.gov  **4、Linux内核参数的优化**  linux优化没有统一的标准，下面的方法仅供参考。  优化方法是执行 vi /etc/sysctl.conf 命令到文件结尾，然后添加如下内容：  这个优化适合Apache、Nginx、Squid等多种WEB应用，特殊的业务可能需要略作调整。  net.ipv4.tcp\_fin\_timeout = 2  net.ipv4.tcp\_tw\_reuse = 1  net.ipv4.tcp\_tw\_recycle = 1  net.ipv4.tcp\_syncookies = 1  net.ipv4.tcp\_keepalive\_time = 600  net.ipv4.ip\_local\_port\_range = 4000 65000  net.ipv4.tcp\_max\_syn\_backlog = 16384  net.ipv4.tcp\_max\_tw\_buckets = 36000  net.ipv4.route.gc\_timeout = 100  net.ipv4.tcp\_syn\_retries = 1  net.ipv4.tcp\_synack\_retries =  1  net.core.somaxconn = 16384  net.core.netdev\_max\_backlog = 16384  net.ipv4.tcp\_max\_orphans = 16384  一下参数是对iptables防火墙的优化，防火墙不开启会提示，可以忽略不计  net.nf\_conntrack\_max = 25000000  net.netfi1ter.nf\_conntrack~max = 25000000  net.netfilter.nf\_conntrack~tcp-t1'meout\_estab1ished = 180  net.netfilter.nf\_conntrack\_tcp\_t1'meout\_t1'me\_wait = 120  net.netfilter.nf\_conntrack\*tcp~t1'meout,close\_wait = 60  net.netfilter.nf\_conntrack\*tcp~t1'meout\*fin\_\_wait = 120 |

#### 2. 系统参数里面有个叫做backlog的可以用来干什么？

|  |
| --- |
| Backlog是linux的listen函数的一个参数。  linux的listen，是系统可以使一台主机上的一个tcp socket在某个端口号被动侦听，等待来自其它主机的tcp socket的连接请求  而listen的backlog参数指定的是已经三次握手完成，达到了established状态但是等待accept的队列的容量。  当这个容量超过上限的时候服务器端便不处理客户端的三次握手了。  backlog参数是在linux内核2.2版本以后，控制已经握手成功的还在accept queue的大小。  int类型的backlog参数，意义为已经完成三次握手、已经成功建立连接的套接字将要进入队列的长度。 |

#### 3. 查看网络连接发现好多TIME\_WAIT 可能是什么原因？对你的应用会有什么影响？你会选择什么样的方式来减少这些TIME\_WAIT

|  |
| --- |
| **1 首先什么是TIME\_WAIT：**  在tcp四步挥手的过程中，主动关闭的一方（一般是指后台服务）会先发出close，然后等待对方的ack和fin，在接收到对方fin后，自身会进入TIME\_WAIT状态。这样做的好处有两个，一是防止最后的ack N+1丢失，二是确保请求包都在网络上消失，不影响新建的连接。  在使用tcp短连接的时候，我们经常会发现服务器存在着大量的TIME\_WAIT状态。  根据TCP协议定义的3次握手断开连接规定,发起socket主动关闭的一方 socket将进入TIME\_WAIT状态,TIME\_WAIT状态将持续2个MSL(Max Segment Lifetime),在Windows下默认为4分钟,即240秒,TIME\_WAIT状态下的socket不能被回收使用。  **2 具体现象如下：**  如果一个处理大量短连接的服务器,是由服务器主动关闭客户端的连接,将导致服务器端存在大量的处于TIME\_WAIT状态的socket, 甚至比处于Established状态下的socket多的多,严重影响服务器的处理能力  **3 如何解决：**  发现系统存在大量TIME\_WAIT状态的连接，通过调整内核参数解决。  编辑文件/etc/sysctl.conf  net.ipv4.tcp\_syncookies = 1  net.ipv4.tcp\_tw\_reuse = 1  net.ipv4.tcp\_tw\_recycle = 1  net.ipv4.tcp\_fin\_timeout = 30  参数含义：  net.ipv4.tcp\_syncookies = 1 表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；  net.ipv4.tcp\_tw\_reuse = 1 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；  net.ipv4.tcp\_tw\_recycle = 1 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。  net.ipv4.tcp\_fin\_timeout 修改系統默认的 TIMEOUT 时间  然后执行 /sbin/sysctl -p 让参数生效 |

#### 4. 可否介绍一下TCP三次握手的过程，如果现在有个网络程序，你用第三方的library来发送数据，你怀疑这个library发送的数据有问题，那么怎么来验证？tcpdump导出的文件你一般是怎么分析的？

|  |
| --- |
| 第一次  第一次握手：建立连接时，客户端发送syn包（syn=j）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  第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；  第三次  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=k+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。  tcpdump是对网络上的数据包进行截获的包分析工具，它支持针对网络层、协议、主机、网络或端口的过滤，并提供and、or、not等逻辑语句来去掉无用的信息。  下面是几个比较常见的tcpdump参数：  -w 保持到指定的文件  -i 指定监听的网卡，缺省显示第一块网卡  -nn 以IP方式显示host  -v 显示详细信息  -s 指定数据包大小，缺省是65535  -t 不显示时间 ，缺省是显示时间戳  -c 获取数据包数量，缺省不限制，需要用Ctrl+c来终止 |

#### 5. KeepAlive是用来干什么的？这样的好处是什么？

|  |
| --- |
| **1 什么是KeepAlive**  TCP链接建立之后，如果应用程序或者上层协议一直不发送数据，或者隔很长时间才发送一次数据，当链接很久没有数据报文传输时如何去确定对方还在线，到底是掉线了还是确实没有数据传输，链接还需不需要保持，这种情况在TCP协议设计中是需要考虑到的。  TCP协议通过一种巧妙的方式去解决这个问题，当超过一段时间之后，TCP自动发送一个数据为空的报文给对方，如果对方回应了这个报文，说明对方还在线，链接可以继续保持，如果对方没有报文返回，并且重试了多次之后则认为链接丢失，没有必要保持链接。  注意：TCP的KeepAlive和HTTP的Keep-Alive是不同的概念  **2 Nginx配置**  从TCP层面Nginx不仅要和Client关心KeepAlive,而且还要和Upstream关心KeepAlive, 同时从HTTP协议层面，Nginx需要和Client关心Keep-Alive,如果Upstream使用的HTTP协议，还要关心和Upstream的Keep-Alive。 |

# Redis

#### 1. 缓存穿透可以介绍⼀一下么？你认为应该如何解决这个问题

答：缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

1、对所有可能查询的参数以hash形式存储，在控制层先进行校验，不符合则丢弃。还有最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。

2、也可以采用一个更为简单粗暴的方法，如果一个查询返回的数据为空（不管是数 据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

#### 2. 你是怎么触发缓存更更新的？(比如设置超时间(被动方式), 比如更更新的时候主动update)？如果是被动的方式如何控制多个入⼝同时触发某个缓存更更新？

答：　[主动]需要操作人员去操作，或者定时调度、[被动]由用户触发更新、[预加载]提前加载好数据

被动更新出现并发问题，可以通过程序锁，锁住更新操作只能有一个进入DB查询，可以避免问题。

#### 3. 你们用Redis来做什么？为什么不用其他的KV存储例例如Memcached, Cassandra等?

答：用redis做缓存数据库。Redis与Memcached相比，几乎覆盖后者绝大部分功能，还有其独有功能：支持持久化、支持多数据类型

#### 4. 你们用什么Redis客户端? Redis高性能的原因大概可以讲一些?

答：使用Java的Redis客户端Jedis，Redis的高性能在于单线程和多路IO复用

#### 5. 你熟悉哪些Redis的数据结构? zset是干什么的? 和set有什么区别?

答：Redis五大数据结构：string、set、list、hash、zset。

zset是有序集合，最常用来实现排行榜。

和set的区别，set是无序集合，zset是有序集合

#### 6. Redis的hash, 存储和获取的具体命令叫什么名字?

答：存储命令：hset <key> <field> <value>

获取命令：hget <key1> <field>

#### 7. LPOP和BLPOP的区别?

答：BLPOP 是阻塞式列表的弹出命令。 它是命令 LPOP 的阻塞版本，这是因为当给定列表内没有任何元素可供弹出的时候， 连接将被 BLPOP 命令阻塞。 当给定多个 key 参数时，按参数 key 的先后顺序依次检查各个列表，弹出第一个非空列表的头元素。

#### 8. Redis的有一些包含SCAN关键字的命令是干嘛的? SCAN返回的数据量是固定的吗?

答：SCAN 命令及其相关的 SSCAN 命令、 HSCAN 命令和 ZSCAN 命令都用于增量地迭代，它们每次执行都只会返回少量元素。

SCAN返回的数据量不固定。

#### 9. Redis中的Lua有没有使用过? 可以用来做什么? 为什么可以这么用?

答：使用过，可以用来解决秒杀业务的高并发请求。

1. Lua可以将复杂或多步的redis操作，写成脚本一次性提交给redis执行，提升性能
2. Lua脚本类似于redis事务有一定原子性，不会被其他命令插队，可以完成一些redis事务操作

#### 10. Redis的Pipeline是用来干什么的?

答：pipeline（管道）可以一次性发送多条命令并在执行完后一次性将结果返回，pipeline通过减少客户端与redis的通信次数来实现降低往返延时时间。

#### 11. Redis持久化大概有几种方式? aof和rdb的区别是什么? AOF有什么优缺 点吗?

答：redis持久化分两种：aof、rdb。

rdb是以快照的方式全量备份redis数据，aof是以日志方式增量备份redis写操作。

aof优点：备份机制更稳健、丢失数据概率更低；可读的日志文本，通过操作aof文件可处理误操作。

aof缺点：比rdb占用更多磁盘空间；恢复备份速度慢；如每次写都同步，有性能压力；有个别bug，造成恢复不能。

#### 12. Redis Replication的大致流程是什么? bgsave这个命令的执行过程?

答：redis Replication 流程：（1）从机向主机发起复制请求（2）主机立刻进行存盘操作，发送rdb文件给从机（3）从机收到rdb文件后进行全盘加载（4）之后每次主机写操作都会立刻发给从机，从机执行相同命令。

Bgsave执行过程：BGSAVE 命令执行之后立即返回 OK ，然后 Redis fork 出一个新子进程，原来的 Redis 进程(父进程)继续处理客户端请求，而子进程则负责将数据保存到磁盘，然后退出。

#### 13. 如果有很多 KV数据要存储到Redis, 但是内存不⾜, 通过什么方式可以缩 减内存? 为什么这样可以缩小内存?

答：可以配置redis内存淘汰机制，通过内存淘汰机制淘汰已存储数据，存入新数据。

#### 14. Redis中List, HashTable都⽤用到了ZipList, 为什么会选择它?

答：redis初始创建hash表,有序集合，链表时, 存储结构采用一种ziplist的存储结构, 这种结构内存排列更紧密, 能提高访存性能。

## 监控、稳定性

#### 业务日志是通过什么方式来收集的？

|  |
| --- |
| 对于java 平台来说，主流的日志工具可以使用log4j、slf4j、logback，多种选择。  但是对于大型互联网公司来说，每天产生的业务日志数量过多，所以有一套专门的日志收集系统。  如京东或者淘宝会让日志走kafka消息队列，然后将日志存储在hdfs和ElasticSearch上并生成索引。  ElasticSearch集中保存日志和索引，后期用大数据对hdfs上的日志进行分析。 |

#### 线上机器如何监控？采用什么开源产品或者自研的产品？它是分钟级的还是秒级的？

|  |
| --- |
| 1 zabbix监控/分钟级的。  2 58用的是内部自研的内部称守望者，线上机器的检测分两部分：运维监测服务是否还存活，架构平台根据框架的日志上报进行报警，框架层捕获异常进行相应的处理，报警是秒级的。  3 美团用的是falcon和cat,都是分钟级的。  4 京东自研用UMP(unified monitoring platform),这个平台上可以监控服务器的内存, cpu, 以及通过配置监控某些方法, 某些SQL语句的执行时间等等，秒级的。 |

#### 如果让你来想办法收集一个JAVA后端应用的性能数据，你会在意哪些方面? 你会选择什么样的⼯具、思路来收集?

|  |
| --- |
| **首先，**可以通过日志监控并收集Java应用程序性能数据。  1 Logback、Log4j是常用于Java平台的日志记录API.目前大部分产品只是将系统重要参数、状态的变化及异常信息通过日志输出。  2 Perf4j是一款专门用于Java服务器端代码计时、记录日志和监控结果的开源工具包。Perf4j对常用日志工具包进行了扩展。Java 本地代码调用(JNI)、分布式系统、集群部署、面向服务体系结构(SOA)、远程方法调用(RMI)都可以使用Perf4j。  **另外，**还有一些应用服务器提供性能参数接口，我们可以自己根据接口建立一个性能监控的项目。  如IBM的付费应用服务器websphere，可以提供一些列的接口，可以检测到后台服务的运行情况。  **另外，**还可以通过对linux系统的运行状况进行监控，来收集系统运行的数据，如设置一个定时任何 执行getperf.sh，采集性能数据。 |

#### 一般你调用第三方的时候会不会监控调用情况？

|  |
| --- |
| 可以使用java平台的日志工具log4j、slf4j、logback等，在第三方系统被调用时，收集第三方系统的调用情况。但是这样做可能会增加本系统服务的负担。  另外，还可以使用消息队列，调用独立的日志系统服务，对第三方系统的调用情况日志进行记录。这样需要开发独立日志系统，但是同时消息队列的使用不会增加本系统的服务器负担。 |