24个必须掌握的数据库.试题 寰宇Java后端 2019-11-18 点击上.Java后端，选择设为星标优质.章，及时送达

作者|寰宇来.|https://dwz.cn/MCAAbvhA上篇 |除了《颈椎康复指南》，还有这 9本书 .、为什么..增列作为主键 1、如果我们定义了主键(PRIMARYKEY)，那么InnoDB会选择主键作为聚集索引。 如果没有显式定义主键，则InnoDB会选择第.个不包含有NULL值的唯.索引作为主键索引。 如果也没有这样的唯.索引，则InnoDB会选择内置6字节.的ROWID作为隐含的聚集索引(ROWID随着.记录的写..主键递增，这个ROWID不像ORACLE的ROWID那样可引.，是隐含的)。 2、数据记录本.被存于主索引（.颗B+Tree）的叶.节点上，这就要求同.个叶.节点内（..为.个内存.或磁盘.）的各条数据记录按主键顺序存放 因此每当有.条新的记录插.时，MySQL会根据其主键将其插.适当的节点和位置，如果..达到装载因.（InnoDB默认为15/16），则开辟.个新的.（节点） 3、如果表使..增主键，那么每次插.新的记录，记录就会顺序添加到当前索引节点的后续位置，当..写满，就会.动开辟.个新的. 4、如果使...增主键（如果.份证号或学号等），由于每次插.主键的值近似于随机，因此每次新纪录都要被插到现有索引.得中间某个位置 此时MySQL不得不为了将新记录插到合适位置.移动数据，甚..标..可能已经被回写到磁盘上.从缓存中清掉，此时.要从磁盘上读回来，这增加了很多开销 同时频繁的移动、分.操作造成了.量的碎.，得到了不够紧凑的索引结构，后续不得不通过OPTIMIZETABLE来重建表并优化填充...。

.、为什么使.数据索引能提.效率 数据索引的存储是有序的在有序的情况下，通过索引查询.个数据是.需遍历索引记录的极端情况下，数据索引的查询效率为.分法查询效率，趋近于log2(N) 三、B+树索引和哈希索引的区别 B+树是.个平衡的多叉树，从根节点到每个叶.节点的.度差值不超过1，.且同层级的节点间有指针相互链接，是有序的，如 下图：

哈希索引就是采..定的哈希算法，把键值换算成新的哈希值，检索时不需要类似B+树那样从根节点到叶.节点逐级查找，只需.次哈希算法即可,是.序的，如下图所.：

图：https://imysql.com/

四、哈希索引的优势： 等值查询，哈希索引具有绝对优势（前提是：没有.量重复键值，如果.量重复键值时，哈希索引的效率很低，因为存在所谓的哈希碰撞问题。）

五、哈希索引不适.的场景： 不.持范围查询 不.持索引完成排序 不.持联合索引的最左前缀匹配规则 通常，B+树索引结构适.于绝.多数场景，像下.这种场景.哈希索引才更有优势：在HEAP表中，如果存储的数据重复度很低（也就是说基数很.），对该列数据以等值查询为主，没有范围查询、没有排序的时候，特别适合采.哈希索引，例如这种SQL： #仅等值查询 selectid,namefromtablewherename='李明'; .常.的InnoDB引擎中默认使.的是B+树索引，它会实时监控表上索引的使.情况。 如果认为建.哈希索引可以提.查询效率，则.动在内存中的"适应哈希索引缓冲区"建.哈希索引（在InnoDB中默认开启.适应哈希索引）。 通过观察搜索模式，MySQL会利.indexkey的前缀建.哈希索引，如果.个表.乎.部分都在缓冲池中，那么建..个哈希索引能够加快等值查询。 注意：在某些.作负载下，通过哈希索引查找带来的性能提升远.于额外的监控索引搜索情况和保持这个哈希表结构所带来 的开销。 但某些时候，在负载.的情况下，.适应哈希索引中添加的read/write锁也会带来竞争，.如.并发的join操作。like操作和%的通配符操作也不适.于.适应哈希索引，可能要关闭.适应哈希索引。 六、B树和B+树的区别 1、B树，每个节点都存储key和data，所有节点组成这棵树，并且叶.节点指针为nul，叶.结点不包含任何关键字信息。

2、B+树，所有的叶.结点中包含了全部关键字的信息，及指向含有这些关键字记录的指针，且叶.结点本.依关键字的.....的顺序链接 所有的.终端结点可以看成是索引部分，结点中仅含有其.树根结点中最.（或最.）关键字。(.B树的.终节点也包含需要查找的有效信息)

七、为什么说B+.B树更适合实际应.中操作系统的.件索引和数据库索引？ 1、B+的磁盘读写代价更低。 B+的内部结点并没有指向关键字具体信息的指针，因此其内部结点相对B树更.。 如果把所有同.内部结点的关键字存放在同.盘块中，那么盘块所能容纳的关键字数量也越多。.次性读.内存中的需要查找的关键字也就越多。相对来说IO读写次数也就降低了。 2、B+-tree的查询效率更加稳定。 由于.终结点并不是最终指向.件内容的结点，.只是叶.结点中关键字的索引。所以任何关键字的查找必须..条从根结点到叶.结点的路。所有关键字查询的路径.度相同，导致每.个数据的查询效率相当。 Tips：欢迎关注微信公众号：Java后端，每.技术博.推送。

.、MySQL联合索引 1、联合索引是两个或更多个列上的索引。 对于联合索引:Mysql从左到右的使.索引中的字段，.个查询可以只使.索引中的.部份，但只能是最左侧部分。 例如索引是keyindex(a,b,c).可以.持a、a,b、a,b,c3种组合进.查找，但不.持b,c进.查找.当最左侧字段是常量引.时，索引就.分有效。 2、利.索引中的附加列，您可以缩.搜索的范围，但使..个具有两列的索引不同于使.两个单独的索引。 复合索引的结构与电话簿类似，.名由姓和名构成，电话簿.先按姓.对进.

排序，然后按名字对有相同姓.的.进.排序。如果您知道姓，电话簿将.常有.；如果您知道姓和名，电话簿则更为有.，但如果您只知道名不知道姓，电话簿将没有.处。

九、什么情况下应不建或少建索引 1、表记录太少2、经常插.、删除、修改的表3、数据重复且分布平均的表字段，假如.个表有10万.记录，有.个字段A只有T和F两种值，且每个值的分布概率.约为50%， 那么对这种表A字段建索引.般不会提.数据库的查询速度。 4、经常和主字段.块查询但主字段索引值.较多的表字段

.、什么是表分区？ 表分区，是指根据.定规则，将数据库中的.张表分解成多个更.的，容易管理的部分。从逻辑上看，只有.张表，但是底层却是由多个物理分区组成。

..、表分区与分表的区别 分表：指的是通过.定规则，将.张表分解成多张不同的表。.如将..订单记录根据时间成多个表。分表与分区的区别在于：分区从逻辑上来讲只有.张表，.分表则是将.张表分解成多张表。

..、表分区有什么好处？ 1、存储更多数据。分区表的数据可以分布在不同的物理设备上，从..效地利.多个硬件设备。和单个磁盘或者.件系统相 .，可以存储更多数据2、优化查询。在where语句中包含分区条件时，可以只扫描.个或多个分区表来提.查询效率；涉及sum和count语句时，也可以在多个分区上并.处理，最后汇总结果。 3、分区表更容易维护。例如：想批量删除.量数据可以清除整个分区。4、避免某些特殊的瓶颈，例如InnoDB的单个索引的互斥访问，ext3问价你系统的inode锁竞争等。

.三、分区表的限制因素 1、.个表最多只能有1024个分区2、MySQL5.1中，分区表达式必须是整数，或者返回整数的表达式。在MySQL5.5中提供了.整数表达式分区的.持。3、如果分区字段中有主键或者唯.索引的列，那么多有主键列和唯.索引列都必须包含进来。即：分区字段要么不包含主键或 者索引列，要么包含全部主键和索引列。4、分区表中.法使.外键约束5、MySQL的分区适.于.个表的所有数据和索引，不能只对表数据分区.不对索引分区，也不能只对索引分区.不对表分区， 也不能只对表的.部分数据分区。

.四、如何判断当前MySQL是否.持分区？ 命令：showvariableslike'%partition%'运.结果:
mysql>showvariableslike'%partition%';+------------------+-------+|Variable_name|Value|+------------------+-------+|have_partitioning|YES|+------------------+-------+1rowins have_partintioning的值为YES，表..持分区。

.五、MySQL.持的分区类型有哪些？ RANGE分区：这种模式允许将数据划分不同范围。例如可以将.个表通过年份划分成若.个分区 LIST分区：这种模式允许系统通过预定义的列表的值来对数据进.分割。按照List中的值分区，与RANGE的区别是，range分区的区间范围值是连续的。HASH分区：这中模式允许通过对表的.个或多个列的HashKey进.计算，最后通过这个Hash码不同数值对应的数据区域进. 分区。例如可以建..个对表主键进.分区的表。KEY分区：上.Hash模式的.种延伸，这.的HashKey是MySQL系统产.的。 .六、四种隔离级别 Serializable(串.化)：可避免脏读、不可重复读、幻读的发.。 Repeatableread(可重复读)：可避免脏读、不可重复读的发.。 Readcommitted(读已提交)：可避免脏读的发.。Readuncommitted(读未提交)：最低级别，任何情况都.法保证。

.七、关于MVCC MySQLInnoDB存储引擎，实现的是基于多版本的并发控制协议——MVCC(Multi-VersionConcurrencyControl) 注：与MVCC相对的，是基于锁的并发控制，Lock-BasedConcurrencyControl MVCC最.的好处：读不加锁，读写不冲突。在读多写少的OLTP应.中，读写不冲突是.常重要的，极.的增加了系统的并发性能，现阶段.乎所有的RDBMS，都.持了MVCC。 LBCC：Lock-BasedConcurrencyControl，基于锁的并发控制MVCC：Multi-VersionConcurrencyControl 基于多版本的并发控制协议。纯粹基于锁的并发机制并发量低，MVCC是在基于锁的并发控制上的改进，主要是在读操作上提.了并发量。

..、在MVCC并发控制中，读操作可以分成两类： 快照读(snapshotread)：读取的是记录的可.版本(有可能是历史版本)，不.加锁（共享读锁s锁也不加，所以不会阻塞其他事务的写） 当前读(currentread)：读取的是记录的最新版本，并且，当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录

.九、.级锁定的优点： 1、当在许多线程中访问不同的.时只存在少量锁定冲突。2、回滚时只有少量的更改3、可以.时间锁定单.的.。

..、.级锁定的缺点： ..级或表级锁定占.更多的内存。当在表的.部分中使.时，..级或表级锁定速度慢，因为你必须获取更多的锁。如果你在.部分数据上经常进.GROUPBY操作或者必须经常扫描整个表，.其它锁定明显慢很多。..级别锁定，通过.持不同的类型锁定，你也可以很容易地调节应.程序，因为其锁成本.于.级锁定。

...、MySQL优化 开启查询缓存，优化查询 explain你的select查询，这可以帮你分析你的查询语句或是表结构的性能瓶颈。EXPLAIN的查询结果还会告诉你你的索引主键被如何利.的，你的数据表是如何被搜索和排序的当只要..数据时使.limit1，MySQL数据库引擎会在找到.条数据后停.搜索，.不是继续往后查少下.条符合记录的数据为搜索字段建索引使.ENUM.不是VARCHAR。如果你有.个字段，.如“性别”，“国家”，“.族”，“状态”或“部.”，你知道这些字段的取值是有限.且固定的，那么，你应该使.ENUM.不是VARCHAR Prepared StatementsPreparedStatements很像存储过程，是.种运.在后台的SQL语句集合，我们可以从使.preparedstatements获得很多好处，.论是性能问题还是安全问题。PreparedStatements可以检查.些你绑定好的变量，这样可以保护你的程序不会受到“SQL注.式”攻击垂直分表选择正确的存储引擎

...、key和index的区别 key是数据库的物理结构，它包含两层意义和作.，.是约束（偏重于约束和规范数据库的结构完整性），.是索引（辅助查询.的）。包括primarykey,uniquekey,foreignkey等 index是数据库的物理结构，它只是辅助查询的，它创建时会在另外的表空间（mysql中的innodb表空间）以.个类似.录的结构存储。索引要分类的话，分为前缀索引、全.本索引等；

..三、Mysql中MyISAM和InnoDB的区别有哪些？ 区别： 1、InnoDB.持事务，MyISAM不.持 对于InnoDB每.条SQL语.都默认封装成事务，.动提交，这样会影响速度，所以最好把多条SQL语.放在begin和commit之间，组成.个事务； 2、InnoDB.持外键，.MyISAM不.持。 对.个包含外键的InnoDB表转为MYISAM会失败； 3、InnoDB是聚集索引，数据.件是和索引绑在.起的，必须要有主键，通过主键索引效率很.。 但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此主键不应该过.，因为主键太.，其他索引也都会很.。 .MyISAM是.聚集索引，数据.件是分离的，索引保存的是数据.件的指针。主键索引和辅助索引是独.的。 4、InnoDB不保存表的具体.数，执.selectcount(*)fromtable时需要全表扫描。.MyISAM..个变量保存了整个表的.数，执.上述语句时只需要读出该变量即可，速度很快；5、Innodb不.持全.索引，.MyISAM.持全.索引，查询效率上MyISAM要.； 如何选择： 是否要.持事务，如果要请选择innodb，如果不需要可以考虑MyISAM； 如果表中绝.多数都只是读查询，可以考虑MyISAM，如果既有读写也挺频繁，请使.InnoDB系统奔溃后，MyISAM恢复起来更困难，能否接受； MySQL5.5版本开始Innodb已经成为Mysql的默认引擎(之前是MyISAM)，说明其优势是有.共睹的，如果你不知道.什么，那就.InnoDB，.少不会差。

..四、数据库表创建注意事项 1、字段名及字段配制合理性 剔除关系不密切的字段；字段命名要有规则及相对应的含义（不要.部分英.，.部分拼.，还有类似a.b.c这样不明含义的字段）；字段命名尽量不要使.缩写（.多数缩写都不能明确字段含义）；字段不要..写混.（想要具有可读性，多个英.单词可使.下划线形式连接）；字段名不要使.保留字或者关键字；保持字段名和类型的.致性；慎重选择数字类型；-给.本字段留.余量； 2、系统特殊字段处理及建成后建议 添加删除标记（例如操作.、删除时间）；建.版本机制； 3、表结构合理性配置 多型字段的处理，就是表中是否存在字段能够分解成更.独.的.部分（例如：.可以分为男.和..）；多值字段的处理，可以将表分为三张表，这样使得检索和排序更加有调理，且保证数据的完整性！ 4、其它建议 对于.数据字段，独.表进.存储，以便影响性能（例如：简介字段）；使.varchar类型代替char，因为varchar会动态分配.度，char指定.度是固定的；给表创建主键，对于没有主键的表，在查询和索引定义上有.定的影响；避免表字段运.为null，建议设置默认值（例如：int类型设置默认值为0）在索引查询上，效率.显；建.索引，最好建.在唯.和.空的字段上，建.太多的索引对后期插.、更新都存在.定的影响（考虑实际情况来创建）； -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.你们..念念的 GitHub客.端终于来了！

2. Redis实现「附近的.」这个功能 3..个秒杀系统的设计思考 4.零基础认识 Spring Boot

5.团队开发中 Git最佳实践

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

40道Java多线程.试题及答案 陈晨.Java后端 2.9.

作者|陈晨. 链接|cnblogs.com/chen-chen-chen/p/12285283.html 这篇.章主要是对多线程的问题进.总结的，因此罗列了40个多线程的问题。 这些多线程的问题，有些来源于各..站、有些来源于..的思考。可能有些问题.上有、可能有些问题对应的答案也有、也可能有些各位.友也都看过，但是本.写作的重.就是所有的问题都会按照..的理解回答.遍，不会去看.上的答案，因此可能有些问题讲的不对，能指正的希望.家不吝指教。 1、多线程有什么.？ .个可能在很多.看来很扯淡的.个问题：我会.多线程就好了，还管它有什么.？在我看来，这个回答更扯淡。所谓"知其然知其所以然"，"会."只是"知

其然"，"为什么."才是"知其所以然"，只有达到"知其然知其所以然"的程度才可以说是把.个知识点运..如。OK，下.说说我对这个问题的看法： 1 ）发挥多核CPU的优势

随着.业的进步，现在的笔记本、台式机乃.商.的应.服务器.少也都是双核的，4核、8核甚.16核的也都不少.，如果是单线程的程序，那么在双核CPU上就浪费了50%，在4核CPU上就浪费了75%。单核CPU上所谓的"多线程"那是假的多线程，同.时间处理器只会处理.段逻辑，只不过线程之间切换得.较快，看着像多个线程"同时"运.罢了。多核CPU上的多线程才是真正的多线程，它能让你的多段逻辑同时.作，多线程，可以真正发挥出多核CPU的优势来，达到充分利.CPU的.的。

2 ）防.阻塞

从程序运.效率的.度来看，单核CPU不但不会发挥出多线程的优势，反.会因为在单核CPU上运.多线程导致线程上下.的切换，.降低程序整体的效率。但是单核CPU我们还是要应.多线程，就是为了防.阻塞。试想，如果单核CPU使.单线程，那么只要这个线程阻塞了，..说远程读取某个数据吧，对端迟迟未返回.没有设置超时时间，那么你的整个程序在数据返回回来之前就停.运.了。多线程可以防.这个问题，多条线程同时运.，哪怕.条线程的代码执.读取数据阻塞，也不会影响其它任务的执.。 3）便于建模 这是另外.个没有这么明显的优点了。假设有.个.的任务A，单线程编程，那么就要考虑很多，建.整个程序模型.较.烦。但是如果把这个.的任务A分解成.个.任务，任务B、任务C、任务D，分别建.程序模型，并通过多线程分别运.这.个任务，那就简单很多了。 2、创建线程的.式 .较常.的.个问题了，.般就是两种： 1 ）继承Thread类

2 ）实现Runnable接.

.于哪个好，不.说肯定是后者好，因为实现接.的.式.继承类的.式更灵活，也能减少程序之间的耦合度，.向接.编程也是设计模式6.原则的核.。 3、start().法和run().法的区别 只有调.了start().法，才会表现出多线程的特性，不同线程的run().法..的代码交替执.。如果只是调.run().法，那么代码还是同步执.的，必须等待.个线程的run().法..的代码全部执.完毕之后，另外.个线程才可以执.其run().法..的代码。 4、Runnable接.和Callable接.的区别 有点深的问题了，也看出.个Java程序员学习知识的.度。 Runnable接.中的run().法的返回值是void，它做的事情只是纯粹地去执.run().法中的代码.已；Callable接.中的call().法是有返回值的，是.个泛型，和Future、FutureTask配合可以.来获取异步执.的结果。 这其实是很有.的.个特性，因为多线程相.单线程更难、更复杂的.个重要原因就是因为多线程充满着未知性，某条线程是否执.了？某条线程执.了多久？某条线程执.的时候我们期望的数据是否已经赋值完毕？.法得知，我们能做的只是等待这条多线程的任务执.完毕.已。.Callable+Future/FutureTask却可以获取多线程运.的结果，可以在等待时间太.没获取到需要的数据的情况下取消该线程的任务，真的是.常有.。 5、CyclicBarrier和CountDownLatch的区别 两个看上去有点像的类，都在java.util.concurrent下，都可以.来表.代码运.到某个点上，.者的区别在于： 1）Cyc licBarrier的某个线程运.到某个点上之后，该线程即停.运.，直到所有的线程都到达了这个点，所有线程才重新运.；CountDownLatch则不是，某线程运.到某个点上之后，只是给某个数值-1.已，该线程继续运.。

2）Cyc licBarrier只能唤起.个任务，CountDownLatch可以唤起多个任务。

3)CyclicBarrier可重.，CountDownLatch不可重.，计数值为0该CountDownLatch就不可再.了。

6、volatile关键字的作. .个.常重要的问题，是每个学习、应.多线程的Java程序员都必须掌握的。理解volatile关键字的作.的前提是要理解Java内存模型，这.就不讲Java内存模型了，可以参.第31点，volatile关键字的作.主要有两个： 1 ）多线程主要围绕可.性和原.性两个特性.展开，使.volatile关键字修饰的变量，保证了其在多线程之间的可.性，即每次读取到volatile变量，.定是最新的数据。

2 ）代码底层执.不像我们看到的.级语.----Java程序这么简单，它的执.是Java代码-->字节码-->根据字节码执.对应的C/C++代码-->C/C++代码被编译成汇编语.-->和硬件电路交互，现实中，为了获取更好的性能JVM可能会对指令进.重排序，多线程下可能会出现.些意想不到的问题。使.volatile则会对禁.语义重排序，当然这也.定程度上降低了代码执.效率。

从实践.度..，volatile的.个重要作.就是和CAS结合，保证了原.性，详细的可以参.java.util.concurrent.atomic包下的类，.如AtomicInteger. 7、什么是线程安全 .是.个理论的问题，各式各样的答案有很多，我给出.个.认为解释地最好的：如果你的代码在多线程下执.和在单线程下执.永远都能获得.样的结果，那么你的代码就是线程安全的。 这个问题有值得.提的地.，就是线程安全也是有.个级别的： 1）不可变 像String、Integer、Long这些，都是final类型的类，任何.个线程都改变不了它们的值，要改变除.新创建.个，因此这些不可变对象不需要任何同步.段就可以直接在多线程环境下使. 2）绝对线程安全 不管运.时环境如何，调.者都不需要额外的同步措施。要做到这.点通常需要付出许多额外的代价，Java中标注..是线程安全的类，实际上绝.多数都不是线程安全的，不过绝对线程安全的类，Java中也有，..说

CopyOnWriteArrayList、 CopyOnWriteArraySet 3）相对线程安全 相对线程安全也就是我们通常意义上所说的线程安全，像Vector这种，add、remove.法都是原.操作，不会被打断，但也仅限于此，如果有个线程在遍历某个Vector、有个线程同时在add这个Vector，99%的情况下都会出现 ConcurrentModificationException，也就是fail-fast机制。 4）线程.安全 这个就没什么好说的了，ArrayList、LinkedList、HashMap等都是线程.安全的类 8、Java中如何获取到线程dump.件 死循环、死锁、阻塞、..打开慢等问题，打线程dump是最好的解决问题的途径。所谓线程dump也就是线程堆栈，获取到线程堆栈有两步： 1）获取到线程的p id，可以通过使.jps命令，在Linux环境下还可以使.ps-ef|grepjava

2 ）打印线程堆栈，可以通过使.jstackpid命令，在Linux环境下还可以使.kill-3pid

另外提.点，Thread类提供了.个getStackTrace().法也可以.于获取线程堆栈。这是.个实例.法，因此此.法是和具体线程实例绑定的，每次获取获取到的是具体某个线程当前运.的堆栈。 9、.个线程如果出现了运.时异常会怎么样 如果这个异常没有被捕获的话，这个线程就停.执.了。另外重要的.点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被.即释放 10、如何在两个线程之间共享数据 通过在线程之间共享对象就可以了，然后通过wait/notify/notifyAll、await/signal/signalAll进.唤起和等待，..说阻塞队列 BlockingQueue就是为线程之间共享数据.设计的 11、sleep.法和wait.法有什么区别 这个问题常问，sleep.法和wait.法都可以.来放弃CPU.定的时间，不同点在于如果线程持有某个对象的监视器，sleep.法不会放弃这个对象的监视器，wait.法会放弃这个对象的监视器 12、.产者消费者模型的作.是什么 这个问题很理论，但是很重要： 1 ）通过平衡.产者的.产能.和消费者的消费能.来提升整个系统的运.效率，这是.产者消费者模型最重要的作.

2 ）解耦，这是.产者消费者模型附带的作.，解耦意味着.产者和消费者之间的联系少，联系越少越可以独.发展.不需要收到相互的制约

13、ThreadLocal有什么. 简单说ThreadLocal就是.种以空间换时间的做法，在每个Thread..维护了.个以开地址法实现的ThreadLocal.ThreadLocalMap，把数据进.隔离，数据不共享，.然就没有线程安全..的问题了 14、为什么wait().法和notify()/notifyAll().法要在同步块中被调. 这是JDK强制的，wait().法和notify()/notifyAll().法在调.前都必须先获得对象的锁 15、wait().法和notify()/notifyAll().法在放弃对象监视器时有什么区别1 wait().法和notify()/notifyAll().法在放弃对象监视器的时候的区别在于：wait().法.即释放对象监视器，notify()/notifyAll().法则会等待线程剩余代码执.完毕才会放弃对象监视器。 16、为什么要使.线程池 避免频繁地创建和销毁线程，达到线程对象的重..。另外，使.线程池还可以根据项.灵活地控制并发的数.。点击这.学习线程池详解。 17、怎么检测.个线程是否持有对象监视器 我也是在.上看到.道多线程.试题才知道有.法可以判断某个线程是否持有对象监视器：Thread类提供了.个 holdsLock(Object obj).法，当且仅当对象obj的监视器被某条线程持有的时候才会返回true，注意这是.个static.法，这意味着"某条线程"指的是当前线程。 18、synchronized和ReentrantLock的区别1 synchronized是和if、else、for、while.样的关键字，ReentrantLock是类，这是.者的本质区别。既然ReentrantLock是类，那么它就提供了.synchronized更多更灵活的特性，可以被继承、可以有.法、可以有各种各样的类变量，ReentrantLock .synchronized的扩展性体现在.点上： （1）ReentrantLock可以对获取锁的等待时间进.设置，这样就避免了死锁 （2）ReentrantLock可以获取各种锁的信息 （3）ReentrantLock可以灵活地实现多路通知 另外，.者的锁机制其实也是不.样的。ReentrantLock底层调.的是Unsafe的park.法加锁，synchronized操作的应该是对象头中markword，这点我不能确定。 19、ConcurrentHashMap的并发度是什么 ConcurrentHashMap的并发度就是segment的..，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最.优势，任何情况下，Hashtable能同时有两条线程获取 Hashtable中的数据吗? 20、ReadWriteLock是什么 .先明确.下，不是说ReentrantLock不好，只是ReentrantLock某些时候有局限。如果使.ReentrantLock，可能本.是为了防.线程A在写数据、线程B在读数据造成的数据不.致，但这样，如果线程C在读数据、线程D也在读数据，读数据是不会改变数据的，没有必要加锁，但是还是加锁了，降低了程序的性能。 因为这个，才诞.了读写锁ReadWriteLock。ReadWriteLock是.个读写锁接.，ReentrantReadWriteLock是 ReadWriteLock接.的.个具体实现，实现了读写的分离，读锁是共享的，写锁是独占的，读和读之间不会互斥，读和写、写和读、写和写之间才会互斥，提升了读写的性能。 21、FutureTask是什么 这个其实前.有提到过，FutureTask表..个异步运算的任务。FutureTask..可以传..个Callable的具体实现类，可以对这个异步运算的任务的结果进.等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接.的实现类，所以FutureTask也可以放.线程池中。 22、Linux环境下如何查找哪个线程使.CPU最. 这是.个.较偏实践的问题，这种问题我觉得挺有意义的。可以这么做： （1）获取项.的pid，jps或者ps-ef|grepjava，这个前.有讲过 （2）top-H-ppid，顺序不能改变 这样就可以打印出当前的项.，每条线程占.CPU时间的百分.。注意这.打出的是LWP，也就是操作系统原.线程的线程号，我笔记本.没有部署Linux环境下的Java.程，因此没有办法截图演.，.友朋友们如果公司是使.Linux环境部署项.的话，可以尝试.下。 使."top -H -p pid"+"jps pid"可以很容易地找到某条占.CPU.的线程的线程堆栈，从.定位占.CPU.的原因，.般是因为不当的代码操作导致了死循环。 最后提.点，"top-H-ppid"打出来的LWP是.进制的，"jpspid"打出来的本地线程号是.六进制的，转换.下，就能定位到占.CPU.的线程的当前线程堆栈了。 23、Java编程写.个会导致死锁的程序 第.次看到这个题.，觉得这是.个.常好的问

题。很多.都知道死锁是怎么.回事.：线程A和线程B相互等待对.持有的锁导致程序.限死循环下去。当然也仅限于此了，问.下怎么写.个死锁的程序就不知道了，这种情况说.了就是不懂什么是死锁，懂.个理论就完事.了，实践中碰到死锁的问题基本上是看不出来的。 真正理解什么是死锁，这个问题其实不难，.个步骤： 1 ）两个线程..分别持有两个Object对象：lock1和lock2。这两个lock作为同步代码块的锁；

2）线程1的run().法中同步代码块先获取lock1的对象锁，Thread.sleep(xxx)，时间不需要太多，50毫秒差不多了，然后接着获取lock2的对象锁。这么做主要是为了防.线程1启动.下.就连续获得了lock1和lock2两个对象的对象锁

3）线程2的run)(.法中同步代码块先获取lock2的对象锁，接着获取lock1的对象锁，当然这时lock1的对象锁已经被线程1锁持有，线程2肯定是要等待线程1释放lock1的对象锁的

这样，线程1"睡觉"睡完，线程2已经获取了lock2的对象锁了，线程1此时尝试获取lock2的对象锁，便被阻塞，此时.个死锁就形成了。代码就不写了，占的篇幅有点多，Java多线程7：死锁这篇.章..有，就是上.步骤的代码实现。 24、怎么唤醒.个阻塞的线程 如果线程是因为调.了wait()、sleep()或者join().法.导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，.能为.，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。 25、不可变对象对多线程有什么帮助 前.有提到过的.个问题，不可变对象保证了对象的内存可.性，对不可变对象的读取不需要进.额外的同步.段，提升了代码执.效率。 26、什么是多线程的上下.切换 多线程的上下.切换是指CPU控制权由.个已经正在运.的线程切换到另外.个就绪并等待获取CPU执.权的线程的过程。 27、如果你提交任务时，线程池队列已满，这时会发.什么 这.区分.下： 1 ）如果使.的是.界队列LinkedBlockingQueue，也就是.界队列的话，没关系，继续添加任务到阻塞队列中等待执.，因为 LinkedBlockingQueue可以近乎认为是.个.穷.的队列，可以.限存放任务

2 ）如果使.的是有界队列.如ArrayBlockingQueue，任务.先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，会根据maximumPoolSize的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue继续满，那么则会使.拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy

28、Java中.到的线程调度算法是什么 抢占式。.个线程.完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出.个总的优先级并分配下.个时间.给某个线程执.。 29、Thread.sleep(0)的作.是什么 这个问题和上.那个问题是相关的，我就连在.起了。由于Java采.抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级.较低的线程也能获取到CPU控制权，可以使.Thread.sleep(0).动触发.次操作系统分配时间.的操作，这也是平衡CPU控制权的.种操作。 30、什么是.旋 很多synchronized..的代码只是.些很简单的代码，执.时间.常快，此时等待的线程都加锁可能是.种不太值得的操作，因为线程阻塞涉及到..态和内核态切换的问题。既然synchronized..的代码执.得.常快，不妨让等待锁的线程不要被阻塞，.是在synchronized的边界做忙循环，这就是.旋。如果做了多次忙循环发现还没有获得锁，再阻塞，这样可能是.种更好的策略。 31、什么是Java内存模型 Java内存模型定义了.种多线程访问Java内存的规范。Java内存模型要完整讲不是这..句话能说清楚的，我简单总结.下Java内存模型的.部分内容： 1 ）Java内存模型将内存分为了主内存和.作内存。类的状态，也就是类之间共享的变量，是存储在主内存中的，每次Java线程.到这些主内存中的变量的时候，会读.次主内存中的变量，并让这些内存在..的.作内存中有.份拷.，运...线程代码的时候，.到这些变量，操作的都是...作内存中的那.份。在线程代码执.完毕之后，会将最新的值更新到主内存中去

2 ）定义了.个原.操作，.于操作主内存和.作内存中的变量

3）定义了vo latile变量的使.规则

4 ）happens-before，即先.发.原则，定义了操作A必然先.发.于操作B的.些规则，.如在同.个线程内控制流前.的代码.定先.发.于控制流后.的代码、.个释放锁unlock的动作.定先.发.于后.对于同.个锁进.锁定lock的动作等等，只要符合这些规则，则不需要额外做同步措施，如果某段代码不符合所有的happens-before规则，则这段代码.定是线程.安全的

32、什么是CAS CAS，全称为CompareandSwap，即.较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS.定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是.个不会变的值A，只要某次CAS操作失败，永远都不可能成功。 33、什么是乐观锁和悲观锁 1 ）乐观锁：就像它的名字.样，对于并发间操作产.的线程安全问题持乐观状态，乐观锁认为竞争不总是会发.，因此它不需要持有锁，将.较-替换这两个动作作为.个原.操作尝试去修改内存中的变量，如果失败则表.发.冲突，那么就应该有相应的重试逻辑。

2 ）悲观锁：还是像它的名字.样，对于并发间操作产.的线程安全问题持悲观状态，悲观锁认为竞争总是会发.，因此每次对某资源进.操作时，都会持有.个独占的锁，就像synchronized，不管三七...，直接上了锁就操作资源了。

34、什么是AQS 简单说.下AQS，AQS全称为AbstractQueuedSychronizer，翻译过来应该是抽象队列同步器。 如果说java.util.concurrent的基础是CAS的话，那么AQS就是整个Java并发包的核.了，ReentrantLock、CountDownLatch、Semaphore等等都.到了它。AQS实际上以双向队列的形式连接所有的Entry，..说ReentrantLock，所有等待的线程都被放在.个Entry中并连成双向队列，前..个线程使.ReentrantLock好了，则双向队列实际上的第.个Entry开始运.。 AQS定义了对双向队列所有的操作，.只开放了tryLock和tryRelease.法给开发者使.，开发者可以根据..的实现重写 tryLock和tryRelease.法，以实现..的并发功能。 35、单例模式的线程安全性 ..常谈的问题了，.先要说的是单例模式的线程安全意味着：某个类的实例在多线程环境下只会被创建.次出来。单例模式有很多种的写法，我总结.下： 1）饿汉式单例模式的写法：线程安全 2）懒汉式单例模式的写法：.线程安全 3）双检锁单例模式的写法：线程安全 36、Semaphore有什么作. Semaphore就是.个信号量，它的作.是限制某段代码块的并发数。Semaphore有.个构造函数，可以传..个int型整数n，表.某段代码最多只有n个线程可以访问，如果超出了n，那么请等待，等到某个线程执.完毕这段代码块，下.个线程再进.。由此可以看出如果Semaphore构造函数中传.的int型整数n=1，相当于变成了.个synchronized了。 37、Hashtable的size().法中明明只有.条语句"returncount"，为什么还要做同步？ 这是我之前的.个困惑，不知道.家有没有想过这个问题。某个.法中如果有多条语句，并且都在操作同.个类变量，那么在多线程环境下不加锁，势必会引发线程安全问题，这很好理解，但是size().法明明只有.条语句，为什么还要加锁？ 关于这个问题，在慢慢地.作、学习中，有了理解，主要原因有两点： 1 ）同.时间只能有.条线程执.固定类的同步.法，但是对于类的.同步.法，可以多条线程同时访问。所以，这样就有问题了，可能线程A在执.Hashtable的put.法添加数据，线程B则可以正常调.size().法读取Hashtable中当前元素的个数，那读取到的值可能不是最新的，可能线程A添加了完了数据，但是没有对size++，线程B就已经读取size了，那么对于线程B来说读取到的size.定是不准确的。.给size().法加了同步之后，意味着线程B调.size().法只有在线程A调.put.法完毕之后才可以调.，这样就保证了线程安全性

2 ）CPU执.代码，执.的不是Java代码，这点很关键，.定得记住。Java代码最终是被翻译成机器码执.的，机器码才是真正可以和硬件电路交互的代码。即使你看到Java代码只有..，甚.你看到Java代码编译之后.成的字节码也只有..，也不意味着对于底层来说这句语句的操作只有.个。.句"returncount"假设被翻译成了三句汇编语句执.，.句汇编语句和其机器码做对应，完全可能执.完第.句，线程就切换了。

38、线程类的构造.法、静态块是被哪个线程调.的 这是.个.常刁钻和狡猾的问题。请记住：线程类的构造.法、静态块是被new这个线程类所在的线程所调.的，.run.法..的代码才是被线程..所调.的。 如果说上.的说法让你感到困惑，那么我举个例.，假设Thread2中new了Thread1，main函数中new了Thread2，那么： 1 ）Thread2的构造.法、静态块是main线程调.的，Thread2的run().法是Thread2..调.的

2 ）Thread1的构造.法、静态块是Thread2调.的，Thread1的run().法是Thread1..调.的

39、同步.法和同步块，哪个是更好的选择 同步块，这意味着同步块之外的代码是异步执.的，这.同步整个.法更提升代码的效率。请知道.条原则：同步的范围越.越好。 借着这.条，我额外提.点，虽说同步的范围越少越好，但是在Java虚拟机中还是存在着.种叫做锁粗化的优化.法，这种.法就是把同步范围变.。这是有.的，..说StringBuffer，它是.个线程安全的类，.然最常.的append().法是.个同步.法，我们写代码的时候会反复append字符串，这意味着要进.反复的加锁->解锁，这对性能不利，因为这意味着Java虚拟机在这条线程上要反复地在内核态和..态之间进.切换，因此Java虚拟机会将多次append.法调.的代码进..个锁粗化的操作，将多次的append的操作扩展到append.法的头尾，变成.个.的同步块，这样就减少了加锁-->解锁的次数，有效地提升了代码执.的效率。 40、.并发、任务执.时间短的业务怎样使.线程池？并发不.、任务执.时间.的业务怎样使.线程池？并发.、业务执.时间.的业务怎样使.线程池？ 这是我在并发编程.上看到的.个问题，把这个问题放在最后.个，希望每个.都能看到并且思考.下，因为这个问题.常好、.常实际、.常专业。关于这个问题，个.看法是： 1 ）.并发、任务执.时间短的业务，线程池线程数可以设置为CPU核数+1，减少线程上下.的切换

2 ）并发不.、任务执.时间.的业务要区分开看：

a）假如是业务时间.集中在IO操作上，也就是IO密集型的任务，因为IO操作并不占.CPU，所以不要让所有的CPU闲下来，可以加.线程池中的线程数.，让CPU处理更多的业务 b）假如是业务时间.集中在计算操作上，也就是计算密集型任务，这个就没办法了，和（1）.样吧，线程池中的线程数设置得少.些，减少线程上下.的切换

c）并发.、业务执.时间.，解决这种类型任务的关键不在于线程池.在于整体架构的设计，看看这些业务..某些数据是否能做缓存是第.步，增加服务器是第.步，.于线程池的设置，设置参考其他有关线程池的.章。最后，业务执.时间.的问题，也可能需要分析.下，看看能不能使.中间件对任务进.拆分和解耦。

推荐阅读1 1.Java：如何更优雅的处理空值？

2.【免费】某平台16980元编程课程资料下载，仅此.次

3.安利.款IDEA中强.的代码.成利器

4.如何获取靠谱的新型冠状病毒疫情

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 41道SpringBoot.试题，帮你整理好了！ Java后端5天前 点击上.Java后端，选择设为星标优质.章，及时送达

今天跟.家分享下SpringBoot常..试题的知识。 1什么是springboot？ .来简化spring应.的初始搭建以及开发过程使.特定的.式来进.配置（properties或yml.件）创建独.的spring引.程序 main.法运.嵌.的Tomcat .需部署war.件简化maven配置.动配置spring添加对应功能starter.动化配置 答：spring boot来简化spring应.开发，约定.于配置，去繁从简，justrun就能创建.个独.的，产品级别的应 . 2 Springboot有哪些优点？ -快速创建独.运.的spring项.与主流框架集成-使.嵌.式的servlet容器，应..需打包成war包-starters.动依赖与版本控制-.量的.动配置，简化开发，也可修改默认值-准.产环境的运.应.监控-与云计算的天然集成 3如何重新加载Spring Boot上的更改，..需重新启动服务器？ 这可以使.DEV.具来实现。通过这种依赖关系，您可以节省任何更改，嵌.式tomcat将重新启动。 Spring Boot有.个开发.具（DevTools）模块，它有助于提.开发.员的.产.。Java开发.员.临的.个主要挑战是将.件更改.动部署到服务器并.动重启服务器。开发.员可以重新加载Spring Boot上的更改，..需重新启动服务器。这将消除每次.动部署更改的需要。 Spring Boot在发布它的第.个版本时没有这个功能。 这是开发.员最需要的功能。DevTools模块完全满.开发.员的需求。该模块将在.产环境中被禁.。它还提供H2数据库控制台以更好地测试应.程序。 org.springframework.boot spring-boot-devtools true 4 Spring Boot、Spring MVC和 Spring有什么区别？ 1、Spring Spring最重要的特征是依赖注.。所有 SpringModules不是依赖注.就是 IOC控制反转。当我们恰当的使. DI或者是 IOC的时候，我们可以开发松耦合应.。松耦合应.的单元测试可以很容易的进.。2、Spring MVC Spring MVC提供了.种分离式的.法来开发 Web应.。通过运.像 DispatcherServelet，MoudlAndView 和 ViewResolver等.些简单的概念，开发 Web应.将会变的.常简单。3、SpringBoot Spring和 SpringMVC的问题在于需要配置.量的参数。

Spring Boot通过.个.动配置和启动的项来.解决这个问题。为了更快的构建产品就绪应.程序，SpringBoot提供了.些.功能性特征。 5什么是.动配置？ Spring和 SpringMVC的问题在于需要配置.量的参数。

我们能否带来更多的智能？当.个 MVC JAR添加到应.程序中的时候，我们能否.动配置.些 beans？ Spring查看（CLASSPATH上可.的框架）已存在的应.程序的配置。在此基础上，Spring Boot提供了配置应.程序和框架所需要的基本配置。这就是.动配置。 6什么是 Spring Boot Stater？ 启动器是.套.便的依赖没描述符，它可以放在..的程序中。你可以.站式的获取你所需要的 Spring和相关技术，.不需要依赖描述符的通过.例代码搜索和复制黏贴的负载。例如，如果你想使. Sping和 JPA访问数据库，只需要你的项.包含 spring-boot-starter-data-jpa依赖项，你就可以完美进.。 7能否举.个例.来解释更多 Staters的内容？ 让我们来思考.个 Stater的例. -Spring Boot StaterWeb。 如果你想开发.个web应.程序或者是公开 REST服务的应.程序。Spring Boot Start Web是.选。让我们使. Spring Initializr创建.个 Spring Boot Start Web的快速项.。 Spring Boot Start Web的依赖项

下.的截图是添加进我们应.程序的不同的依赖项

依赖项可以被分为： Spring -core，beans，context，aop Web MVC-（Spring MVC） Jackson -for JSON Binding Validation -Hibernate,Validation API Enbedded Servlet Container -Tomcat Logging -logback,slf4j任何经典的 Web应.程序都会使.所有这些依赖项。Spr

i 作为.个开发者，我不需要再担.这些依赖项和它们的兼容版本。 8 Spring Boot还提供了其它的哪些 Starter Project Options？ Spring Boot也提供了其它的启动器项.包括，包括.于开发特定类型应.程序的典型依赖项。

spring-boot-starter-web-services -SOAP Web Services；

spring-boot-starter-web -Web和 RESTful应.程序；

spring-boot-starter-test -单元测试和集成测试；

spring-boot-starter-jdbc -传统的 JDBC；

spring-boot-starter-hateoas -为服务添加 HATEOAS功能；

spring-boot-starter-security -使. SpringSecurity进..份验证和授权；

spring-boot-starter-data-jpa -带有 Hibeernate的 Spring DataJPA；

spring-boot-starter-data-rest -使. Spring DataREST公布简单的 REST服务；

9创建.个 Spring Boot Project的最简单的.法是什么？ Spring Initializr是启动 Spring Boot Projects的.个很好的.具。

就像上图中所展.的.样，我们需要做.下.步：1、登录 Spring Initializr，按照以下.式进.选择：2、选择 com.in28minutes.springboot为组3、选择 studet-services为组件4、选择下.的依赖项 Web Actuator DevTools 5、点击. GenerateProject 6、将项.导. Eclipse。.件 -导. -现有的 Maven项. 10 Spring Initializr是创建 Spring Boot Projects的唯..法吗？ 不是的。 Spring Initiatlizr让创建 Spring Boot项.变的很容易，但是，你也可以通过设置.个 maven项.并添加正 确的依赖项来开始.个项.。 在我们的 Spring课程中，我们使.两种.法来创建项.。 第.种.法是 start.spring.io。 另外.种.法是在项.的标题为"Basic Web Application"处进..动设置。 .动设置.个 maven项. 这.有.个重要的步骤： 1、在 Eclipse中，使.. 件 -新建 Maven项.来创建.个新项. 2、添加依赖项。 3、添加 maven插件。 4、添加 Spring Boot应.程序类。 到这.，准备.作已经做好！ 11为什么我们需要 spring-boot-maven-plugin? spring-boot-maven-plugin提供了.些像 jar.样打包或者运.应.程序的命令。1、spring-boot:run运.你的 SpringBooty应.程序。2、spring-boot：repackage重新打包你的 jar包或者是 war包使其可执.3、spring-boot：start和 spring-boot：stop管理 Spring Boot应.程序的.命周期（也可以说是为了集成 测试）。4、spring-boot:build-info.成执.器可以使.的构造信息。 12如何使. SpringBoot .动重装我的应.程序? 使. Spring Boot开发.具。把 Spring Boot开发.具添加进.你的项.是简单的。把下.的依赖项添加.你的 Spring Boot Project pom.xml中

重启应.程序，然后就可以了。 同样的，如果你想.动装载..，有可以看看 FiveReload

在我测试的时候，发现了 LiveReload漏洞，如果你测试时也发现了，请.定要告诉我们。 13 Spring Boot中的监视器是什么？ Spring bootactuator是spring启动框架中的重要功能之.。Spring boot监视器可帮助您访问.产环境中正 在运.的应.程序的当前状态。有.个指标必须在.产环境中进.检查和监控。即使.些外部应.程序可能正在使.这些服务来向相关.员触发警报消息。监视器模块公开了.组可直接作为HTTP URL访问的REST端点来检查状态。 14什么是YAML？ YAML是.种.类可读的数据序列化语.。它通常.于配置.件。与属性.件相.，如果我们想要在配置.件中添加复杂的属性，YAML.件就更加结构化，.且更少混淆。可以看出YAML具有分层配置数据。 15 springboot.动配置的原理 在spring程序main.法中添加@SpringBootApplication或者@EnableAutoCon.guration会.动去maven中读取每个starter中的spring.factories.件该.件.配置了所有需要被创建spring容器中的bean 16 springboot读取配置.件的.式 springboot默认读取配置.件为application.properties或者是application.yml 17 springboot集成mybatis的过程 添加mybatis的startermaven依赖 org.mybatis.spring.boot mybatis-spring-boot-starter 1.3.2 在mybatis的接.中添加@Mapper注解在application.yml配置数据源信息 18什么是嵌.式服务器？我们为什么要使.嵌.式服务器呢? 思考.下在你的虚拟机上部署应.程序需要些什么。第.步：安装 Java第.步：安装 Web或者是应.程序的服务器（Tomat/Wbesphere/Weblogic等等）第三步：部署应.程序 war包如果我们想简化这些步骤，应该如何做呢？让我们来思考如何使服务器成为应.程序的.部分？你只需要.个安装了Java的虚拟机，就可以直接在上.部署应.程序了，这个想法是嵌.式服务器的起源。当我们创建.个可以部署的应.程序的时候，我们将会把服务器（例如，tomcat）嵌.到可部署的服务器中。例如，对于.个 Spring Boot应.程序来说，你可以.成.个包含 Embedded Tomcat的应.程序 jar。你就可 以想运.正常 Java应.程序.样来运.web应.程序了。嵌.式服务器就是我们的可执.单元包含服务器的.进制.件（例如，tomcat.jar）。 19如何在 Spring Boot中添加通.的 JS代码？ 在源.件夹下，创建.个名为 static的.件夹。然后，你可以把你的静态的内容放在这..。例如，myapp.js的路径是 resources\static\js\myapp.js你可以参考它在 jsp中的使..法：

错误：HAL browser gives me unauthorized error -Full authenticaition is required to access this resource.该如何来修复这个错误呢?

两种.法：.法 1：关闭安全验证 application.properties

.法.：在.志中搜索密码并传递.请求标头中 20什么是 Spring Data？ 来.：//projects.spring.io/spring-data/ Spring Data的使命是在保证底层数据存储特殊性的前提下，为数据访问提供.个熟悉的，.致性的，基于 Spring的编程模型。这使得使.数据访问技术，关系数据库和.关系数据库，map-reduce框架以及基于云的数据服务变得很容易。为了让它更简单.些，Spring Data提供了不受底层数据源限制的 Abstractions接.。下.来举.个例.:

你可以定义.简单的库，.来插.，更新，删除和检索代办事项，.不需要编写.量的代码。 21什么是 Spring Data REST? Spring DataTEST可以.来发布关于 Spring数据库的 HATEOAS RESTful资源。下.是.个使. JPA的例.:

不需要写太多代码，我们可以发布关于 Spring数据库的 RESTful API。下.展.的是.些关于 TEST服务器的例.

代码如下：

响应内容：响应包含新创建资源的 href。

22 path=”users”, collectionResourceRel=”users”如何与 Spring DataRest .起使.?

path-这个资源要导出的路径段。 collectionResourceRel-.成指向集合资源的链接时使.的 rel值。在.成 HATEOAS链接时使.。 23当 Spring Boot应.程序作为 Java应.程序运.时，后台会发.什么？ 如果你使. Eclipse IDE，Eclipse maven插件确保依赖项或者类.件的改变.经添加，就会被编译并在.标.件中准备好！在这之后，就和其它的 Java应.程序.样了。当你启动 java应.程序的时候，spring boot .动配置.件就会魔法般的启.了。当 Spring Boot应.程序检测到你正在开发.个web应.程序的时候，它就会启动 tomcat。 24我们能否在 spring-boot-starter-web中. jetty代替 tomcat？ 在 spring-boot-starter-web移除现有的依赖项，并把下.这些添加进去。

25如何使. Spring Boot .成.个 WAR .件？ 推荐阅读: https://spring.io/guides/gs/convert-jar-to-war/下.有 spring说明.档直接的链接地址：

26如何使. Spring Boot部署到不同的服务器？ 你需要做下.两个步骤：在.个项.中.成.个 war.件。将它部署到你最喜欢的服务器（websphere或者 Weblogic或者 Tomcat and soon）。第.步：这本..指南应该有所帮助：https://spring.io/guides/gs/convert-jar-to-war/第.步：取决于你的服务器。 27 RequestMapping和 GetMapping的不同之处在哪.？ RequestMapping具有类属性的，可以进. GET,POST,PUT或者其它的注释中具有的请求.法。GetMapping是GET请求.法中的.个特例。它只是 ResquestMapping的.个延伸，.的是为了提.清晰度。 28为什么我们不建议在实际的应.程序中使. Spring Data Rest? 我们认为 Spring DataRest很适合快速原型制造！在.型应.程序中使.需要谨慎。通过 Spring DataREST你可以把你的数据实体作为 RESTful服务直接发布。当你设计 RESTful服务器的时候，最佳实践表明，你的接.应该考虑到两件重要的事情：你的模型范围。你的客.。通过 With Spring DataREST，你不需要再考虑这两个..，只需要作为 TEST服务发布实体。这就是为什么我们建议使. Spring DataRest在快速原型构造上.，或者作为项.的初始解决.法。对于完整演 变项.来说，这并不是.个好的注意。 29在 Spring Initializer中，如何改变.个项.的包名字？ 好消息是你可以定制它。点击链接”转到完整版本”。你可以配置你想要修改的包名称！ 30JPA和 Hibernate有哪些区别？ 简..之JPA是.个规范或者接. Hibernate是 JPA的.个实现当我们使. JPA的时候，我们使. javax.persistence包中的注释和接.时，不需要使. hibernate的导. 包。我们建议使. JPA注释，因为哦我们没有将其绑定到 Hibernate作为实现。后来（我知道 -.于百分之.的.率），我们可以使.另.种 JPA实现。 31使. Spring Boot启动连接到内存数据库 H2的 JPA应.程序需要哪些依赖项？ 在 Spring Boot项.中，当你确保下.的依赖项都在类路..的时候，你可以加载 H2控制台。web启动器 h2 jpa数据启动器其它的依赖项在下.：

需要注意的.些地.：.个内部数据内存只在应.程序执.期间存在。这是学习框架的有效.式。这不是你希望的真是世界应.程序的.式。在问题”如何连接.个外部数据库？”中，我们解释了如何连接.个你所选择的数据库。 32如何不通过任何配置来选择 Hibernate作为 JPA的默认实现？ 因为 Spring Boot是.动配置的。下.是我们添加的依赖项:

spring-boot-stater-data-jpa对于 Hibernate和 JPA有过渡依赖性。当 Spring Boot在类路径中检测到 Hibernate中，将会.动配置它为默认的 JPA实现。 33我们如何连接.个像 MySQL或者Orcale.样的外部数据库？ 让我们以 MySQL为例来思考这个问题：第.步 -把 mysql连接器的依赖项添加. pom.xml

第.步 -从 pom.xml中移除 H2的依赖项或者.少把它作为测试的范围。

第三步 -安装你的 MySQL数据库 更多的来看看这. -https://github.com/in28minutes/jpa-with-hibernate#installing-and-setting-up-mysql第四步 -配置你的 MySQL数据库连接配置 application.properties

1 spring.jpa.hibernate.ddl-auto=nonespring.datasource.url=jdbc:mysql://localhost:3306/todo example 第五步 -重新启动，你就准备好了！就是这么简单！

34你能否举.个以 ReadOnly为事务管理的例.？ 当你从数据库读取内容的时候，你想把事物中的..描述或者是其它描述设置为只读模式，以便于 Hebernate不需要再次检查实体的变化。这是.常.效的。 35 Spring Boot的核.注解是哪个？它主要由哪.个注解组成的？ 启动类上.的注解是@SpringBootApplication，它也是 Spring Boot的核.注解，主要组合包含了以下 3个注解： @SpringBootCon.guration：组合了 @Con.guration注解，实现配置.件的功能。
@EnableAutoCon.guration：打开.动配置的功能，也可以关闭某个.动配置的选项，如关闭数据源.动配置功能：

@SpringBootApplication(exclude = { DataSourceAutoCon.guration.class })。@ComponentScan：Spring组件扫描。 36开启 Spring Boot特性有哪.种.式？ 1）继承spr ing-boot-starter-parent项.

2）导.spr ing-boot-dependencies项.依赖

37 Spring Boot需要独.的容器运.吗？ 可以不需要，内置了 Tomcat/ Jetty等容器。 38运. Spring Boot有哪.种.式？ 1 ）打包.命令或者放到容器中运.

2）. Maven/ Gradle插件运. 3）直接执. main.法运.

39你如何理解 Spring Boot中的 Starters？ Starters可以理解为启动器，它包含了.系列可以集成到应...的依赖包，你可以.站式集成 Spring及其他技术，.不需要到处找.例代码和依赖包。如你想使. Spring JPA访问数据库，只要加. spring-boot-starter-data-jpa启动器依赖就能使.了。 40 Spring Boot .持哪些.志框架？推荐和默认的.志框架是哪个？ Spring Boot .持 Java Util Logging, Log4j2, Lockback作为.志框架，如果你使. Starters启动器，Spring Boot将使. Logback作为默认.志框架. 41 SpringBoot实现热部署有哪.种.式？ 主要有两种.式： 1、SpringLoaded 2、Spring-boot-devtools 参考.献：https://blog.csdn.net/Dome_/article/details/90339363 来源：阿凯的帽.反戴链接：blog.csdn.net/Kevin_Gu6/article/details/88547424 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。同时标星（置顶）本公众号可以第.时间接受到博.推送。 推荐阅读 1..程序员做了个梦。。。

2.

Spring和SpringBoot之间到底有啥区别？

3. IDEA新特性：提前知道代码怎么. 4.ping命令还能这么玩？

6年Java.兵BAT.试.经 Java后端 2019-09-24 点击上.Java后端，选择"设为星标"优质.章，及时送达

推荐阅读：IntelliJ IDEA 2019.3这回真的要.起来了 我觉得有.个能够找.份..的offer的想法，这是很正常的，这并不是我们的饭后谈资.是每个技术.的追求。像阿.、腾讯、美团、字节跳动、京东等等的技术氛围与技术规范度还是要明显优于.些创业型公司 / .公司，如果说能够在这样的公司锻炼.年，相信对..能.的提升还是.常.的。不论是校招还是社招都避免不了各种.试、笔试，如何去准备这些东西就显得格外重要。不论是笔试还是.试都是有章可循的。 因为...试.般都有专业团队负责，某个知识点你到底是掌握了还是单纯背下来，.试官.问就可以看出来（PS：真正到.试特别是你觉得准备.试的时间不够的时候，你可以多挑.些.试常问的问题来看，注意理解，.定不要死记硬背）。可以试着参考各种.经，知道.试的.体思路，然后去提...的综合能.。 "80%的offer掌握在20%的.."中这句话也不是不.道理的。决定你.试能否成功的因素中实.固然占有很..部分.例，但是如果你的.态或者说运.不好的话，依然.法拿到满意的 o.er。运.暂且不谈，就拿.态来说，千万不要因为.试失败..馁或者说怀疑..的能.，.试失败之后多总结.下失败的原因，后.你就会发现..会越来越强.。 从..实际招聘要求来看到底.睐什么样的.？ .先要明确的.点是：985/211的学历的确会为你加分很多。 另外，再强调的.点是不要天天把..的学校是双.学校这个接.当做你.法进...的原因。只要你的能..够，..的..就会为你打开。也有着很多双.学校甚.是三本的同学就拿到像阿.、腾讯这样的.公司的 o.er。微信搜索 web_resource关注后获取更多优质.章 从阿.、腾讯等..招聘官.对于Java后端.向/后端.向的要求，我们.概可以总结看出..对招聘者的能.要求。 下.以阿.巴巴为例.，看看..的实际要求在.试Java.程师的时候，下..点也提升你的个.竞争.：

1.熟悉开源框架的底层，阅读源码； 2..型数据库系统经验； 3.熟悉分布式，缓存，消息中间件； 4.良好的表达和沟通能.,善于学习,关注前沿。 ".定要有....的特.，不管是技术还好还是其他能."。我觉得这句话真的.常有道理，.家可以仔细思考.下。在这.再强调.点：公司不需要你什么都会，但是在某...你.定要有过于常.的优点。换.之就是我们不需要去掌握每..技术（你也没精.去掌握这么多技术），.是需要去深.研究某..技术，对于其他技术我们可以简单了解.下。 我觉得.起你对每..技术都是浅尝辄.，深.吃透某..技术对你的个.竞争.提升才更有帮助。 如何获取...试机会？ 在讲如何获取...试机会之前，先来对..下两个.常常.的概念——春招和秋招。招聘.数：秋招多于春招；招聘时间：秋招.般 7 .左右开始，.概.直持续到 10 .底。但是..（如 BAT）都会早开始早结束，所以.定要把握好时间。春招 最佳时间为 3 .，次佳时间为 4 .，进. 5 .基本就不会再有春招了（.三银四）。应聘难度：秋招略.于春招；招聘公司：秋招数量多，.春招数量较少，.般为秋招的补充。综上，.般来说，秋招的含.量明显是.于春招的。下.我就说.下我..知道的.些.法，不过应该也涵盖了.部分获取.试机会的.法。关注..官.，随时投递简历（.流程的.申）；找到师兄师姐或者认识的前公司的技术..，帮忙内推（能够让你避开.申简历筛选，笔试筛选，还是挺不错的，不过也还是需要你 的简历够棒）；求职类.站投递简历（不是太推荐）。除了这些.法，我也遇到过这样的经历：有些.公司的.些部.可能暂时没招够.，然后如果你的亲戚或者朋友刚好在这个公

司，. 你正好.在寻求 o.er，那么.试机会基本上是有了，.且这种.试的难度好像.般还普遍.其他正规.试低很多。想要取得.份..满意的offer，前提是..要有过硬的实.作为资本，下.就如何提.个.硬实.给.家提.点建议！微信搜索 web_resource关注后获取更多优质.章 如何提.个.硬实.及..Java后端.试主要问些什么？ 我在这.所说的个.硬实.更多的指的是个.的专业能.，.如构建.质量.站的能.或者是对专业知识的掌握程度。 我觉得不论是对于新.还是..，想要提.个.硬实.最重要的就是不断深.学习并且将理论实践，最好可以将理论在具体项.中实践.下。想要提.个.硬实.，那么学习..新技术的.法.定是.关重要了。下.分享.下对于学习..新技术的.些要点（在这以图.的.式整理了出来，更加.便阅读）：

.定要有..的技术优势，可能你懂得不是最多的，但是别.不会的你却会，那么你就是厉害的！然.如何准备...试？我觉得最关键的.点之.就是搞清楚...试主要在问些什么。下.我将分解每.个知识点，给.家简单说.下...试主要会问些什么？ .先你要明确的是.试官所问的内容.定和你简历所写的东西是紧密联系的，.般你没有记录简历上的技能，.试管很少会去提问。 ...试.体上包括下....知识类型： Java基础、多线程、IO与NIO、虚拟机、设计模式 .试官在多线程这.部分很可能会问你有没有在项.中实际使.多线程的经历。所以，如果你在你的项.中有实际使. Java多线程的经历的话，会为你加分不少哦！

设计模式.较常.的就是让你.写.个单例模式（注意单例模式的.种不同的实现.法）或者让你说.下某个常.的设计模式在你的项.中是如何使.的，另外.试官还有可能问你“抽象..”和“...法模式的区别”、 “..模式”的思想这样的问题”。 微信搜索 web_resource关注后获取更多优质.章 建议把代理模式、观察者模式、 （抽象）..模式好好看.下，这三个设计模式很有.。

数据结构与算法（要有.写算法的能.） 数据结构.较常问的就是：.叉树、红.树（很可能让你.绘.个红.树出来哦！）、.叉查找树（BST）、平衡.叉树（Self-balancingbinary search tree）、B .树，B +树与 B *树的优缺点.较、LSM树这些知识点。数据结构很重要，.且学起来也相对要难.些。建议学习数据结构.定要循序渐进的来，.步.个脚印的.好。.定要搞懂原理，最好..能.代码实现.遍。 计算机.络（TCP三次握.和四次挥.）数据通信（RESTful、RPC、消息队列）

如果你的简历上写了你会某个RPC框架（.如：阿.的开源的dubbo）或者消息队列（.如：RabbitMQ、Kafka）的使.的话，.试官.般会以你写在简历上的技术提问，回答的时候最好能结合在项.中的实际使.。

性能优化及操作系统（常.优化.式，Linux的基本命令以及使.）

主流框架（Spring底层原理与源码问的很多） Spring.般是不可避免的，如果你的简历上注明了你会Spring Boot或者Spring Cloud的话，那么.试官也可能会同时问你这两个技术,.如他可能会问你 springboot和 spring的区别。微信搜索 web_resource关注后获取更多优质.章。所以，.定要谨慎对待写在简历上的东西，.定要对简历上的东西.常熟悉。 微信搜索 web_resource关注后获取更多优质.章 另外，AOP实现原理、动态代理和静态代理、Spring IOC的初始化过程、IOC原理、..怎么实现.个 IOC容器？这些东西都是经常会被问到的。

数据存储（最常.的是 MySQL、Redis）

分布式（分布式锁，事务等）

除了这些东西还有什么其他问题： 实际场景题 实际场景题就是对你的知识运.能.以及思维能.的考察。建议在平时养成多思考问题的习惯，这样.试的时候碰到这样的问题就不.于慌了。另外，如果..实在不会就给.试官委婉的说.下，.试官可能会给你提醒.下。切忌不懂装懂，乱答...。 .试官可能会问你类似这样的问题： 1. 假设你要做.个银. app，有可能碰到多个.同时向.个账.打钱的情况，有可能碰到什么问题，如何解决（锁）？

2.你是怎么保证你的代码质量和正确性的?

3.

下单过程中是下订单减库存还是付款减库存，分析.下两者的优劣。

4.同时给 10万个.发.资，怎么样设计并发.案，能确保在 1分钟内全部发完。

5.如果让你设计 xxx系统的话，你会如何设计。

.活 1..般到最后的 HR .的时候，.试官基本就是和你聊聊天。他可能会问你类似如下的问题：

2.

..是做什么的，具体.点

3...平时是如何学习的 4.平时的兴趣爱好是什么 性格/其他 1. 主要是看你个.的性格以及价值观是否适合他们公司，.如他会问你类似下.的问题：

2.

遇到压..的情况..是如何处理的

3.遇到很难解决的困难怎么办

4.

遇到不是很喜欢同项.组的某个成员的情况怎么办

5.如何看待加班

6.

你觉得..有什么缺点/优点

总结强调 .定要谨慎对待写在简历上的东西，.定要对简历上的东西.常熟悉。因为.般情况下，.试官都是会根据你的简历来问的；能有.个上得了台.的项.也.常重要，这很可能是.试官会.量发问的地.，所以在.试之前好好回顾.下..所做的项.；和.试官聊基础知识.如设计模式的使.、多线程的使.等等，可以结合具体的项.场景或者是..在平时是如何使.的；建议提前了解.下..想要.试的公司的价值观，判断.下..究竟是否适合这个公司。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2. IntelliJ IDEA 2019.3这回真的要.起来了 3..试官：说.说 Spring Boot .动配置原理

4.在浏览器输. URL回.之后发.了什么?

5.接私活必备的 10个开源项.

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

8种.案，帮你解决重复提交问题！ 锦成同学Java后端 2019-10-21

作者：锦成同学来源：juejin.im/post/5d31928c51882564c966a71c 1.什么是幂等

在我们编程中常.幂等 select查询天然幂等 delete删除也是幂等,删除同.个多次效果.样update直接更新某个值的,幂等update更新累加操作的,.幂等 insert.幂等操作,每次新增.条

2.产.原因 由于重复点击或者.络重发eg: 点击提交按钮两次;点击刷新按钮;使.浏览器后退按钮重复之前的操作，导致重复提交表单;使.浏览器历史记录重复提交表单;浏览器重复的HTTP请; nginx重发等情况;分布式RPC的try重发等;

3.解决.案 1、前端js提交禁.按钮可以..些js组件 2、使.Post/Redirect/Get模式 在提交后执...重定向，这就是所谓的Post-Redirect-Get(PRG)模式。简.之，当..提交了表单后，你去执..个客.端的重定向，转到提交成功信息...。这能避免..按F5导致的重复提交，.其也不会出现浏览器表单重复提交的警告，也能消除按浏览器前进和后退按导致的同样问 题。 3、在session中存放.个特殊标志 在服务器端，.成.个唯.的标识符，将它存.session，同时将它写.表单的隐藏字段中然后将表单..发给浏览器，..录.信息后点击提交在服务器端，获取表单中隐藏字段的值，与session中的唯.标识符.较，相等说明是.次提交，就处理本次请求，然后将 session中的唯.标识符移除；不相等说明是重复提交，就不再处理。 4、其他借助使.header头设置缓存控制头Cache-control等.式 .较复杂不适合移动端APP的应.这.不详解 5、借助数据库 insert使.唯.索引update使.乐观锁version版本法这种在.数据量和.并发下效率依赖数据库硬件能.,可针对.核.业务 6、借助悲观锁使.select...forupdate,这种和synchronized锁住先查再insertorupdate.样,但要避免死锁,效率也较差针对单体请求并发不.可以推荐使. 7、借助本地锁(本.重点) 原理:使.了ConcurrentHashMap并发容器putIfAbsent.法,和ScheduledThreadPoolExecutor定时任务也可以使.guavacache的机制,gauva中有配有缓存的有效时间也是可以的key的.成 Content-MD5Content-MD5是指Body的MD5值，只有当Body.Form表单时才计算MD5，计算.式直接将参数和参数

名称统.加密MD5MD5在.定范围类认为是唯.的近似唯.当然在低并发的情况下.够了本地锁只适.于单机部署的应.. ①配置注解

1 @Target(ElementType.METHOD) 2 @Retention(RetentionPolicy.RUNTIME) 3 @Documented 4 public @interface Resubmit { 5 /** 6 *延时时间在延时多久后可以再次提交 7 * 8

- @return Time unit is one second 9 */ 10 int delaySeconds() default 20; 11 } 12 ②实例化锁 1 /* 2 * @author lijing 3 重复提交锁 4 / 5 @Slf4j 6 public final class ResubmitLock { 7 8 9 private static final ConcurrentHashMap LOCK_CACHE = new ConcurrentHashMap<>(200); 10 private static final ScheduledThreadPoolExecutor EXECUTOR = new ScheduledThreadPoolExecutor(5, 11 12 13 // private Cache CACHES = CacheBuilder.newBuilder() 14 //最.缓static存100final个 15 // .maximumSize(1000) 16 //设置写缓存后 5秒钟过期 17 // .expireAfterWrite(5, TimeUnit.SECONDS) 18 // .build(); 19 20 21 private ResubmitLock() { 22 } 23 24 / 25 *静态内部类单例模式 26
- 27 * @return 28 */ 29 private static class SingletonInstance { 30 private static final ResubmitLock INSTANCE = new ResubmitLock();

31 } 32 33 public static ResubmitLock getInstance() { 34 return SingletonInstance.INSTANCE; 35 } 36 37 38 public static String handleKey(String param) { 39 return DigestUtils.md5Hex(param == null ? "" : param); 40 } 41

42 /**
43 加锁 putIfAbsent是原 .操作保证线程安全
44 *
45 * @param key 对应的 key
46 * @param value
47 * @return
48 /
49 public boolean lock(final String key, Object value) {
50 return Objects.isNull(LOCK_CACHE.putIfAbsent(key, value));
51 }
52
53 /
54 *延时释放锁 .以控制短时间内的重复提交
55 *
56 * @param lock 是否需要解锁
57 * @param key 对应的 key
58 * @param delaySeconds延时时间
59 */
60 public void unLock(final boolean lock, final String key, final int delaySeconds) {
61 if (lock) {
62 EXECUTOR.schedule(() -> {
63 LOCK_CACHE.remove(key);
64 }, delaySeconds, TimeUnit.SECONDS);
65 }
66 }
67 }

③AOP切. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 @Log4j @Aspect @Component public class ResubmitDataAspect { private final static String DATA = "data"; private final static Object PRESENT = new Object(); @Around("@annotation(com.cn.xxx.common.annotation.Resubmit)") public Object handleResubmit(ProceedingJoinPoint joinPoint) throws Throwable { = ((MethodSignature) joinPoint.getSignature()).getMethod(); //获取注Method解信息methodResubmit annotation = method.getAnnotation(Resubmit.class); int delaySeconds = annotation.delaySeconds(); Object[] pointArgs = joinPoint.getArgs(); String key = ""; //获取第.个参数 Object firstParam = pointArgs[0]; if (firstParam instanceof RequestDTO) { //解析参数 JSONObject requestDTO = JSONObject.parseObject(firstParam.toString()); JSONObject data = JSONObject.parseObject(requestDTO.getString(DATA)); if (data != null) { 24 StringBuffer sb = new StringBuffer();

```
25 data.forEach((k, v) -> {
26 sb.append(v);
27 });
28 //.成加密参数使.了content_MD5的加密.式
29 key = ResubmitLock.handleKey(sb.toString());
30 }
31 }
32 //执.锁
33 boolean lock = false;
34 try {
35 //设置解锁key
36 lock = ResubmitLock.getInstance().lock(key, PRESENT);
37 if (lock) {
38 //放.
39 return joinPoint.proceed();
40 } else {
41 //响应重复提交异常
42 return new ResponseDTO<>(ResponseCode.REPEAT_SUBMIT_OPERATION_EXCEPTION);
43 }
44 } finally {
45 //设置解锁key和解锁时间
46 ResubmitLock.getInstance().unLock(lock, key, delaySeconds);
47 }
48 }
49 }
```

④注解使.案例 1 @ApiOperation(value = "保存我的帖.接.", notes = "保存我的帖.接.") 2 @PostMapping("/posts/save") 3 @Resubmit(delaySeconds = 10) 4 public ResponseDTO saveBbsPosts(@RequestBody @Validated RequestDTO requestDto) { 5 return bbsPostsBizService.saveBbsPosts(requestDto); 6 } 以上就是本地锁的.式进.的幂等提交使.了Content-MD5进.加密只要参数不变,参数加密密值不变,key存在就阻.提交 当然也可以使..些其他签名校验在某.次提交时先.成固定签名提交到后端根据后端解析统.的签名作为每次提交的验证token去缓存中处理即可. 8、借助分布式redis锁（参考其他） 在pom.xml中添加上starter-web、starter-aop、starter-data-redis的依赖即可

py 2 org.springframework.bootgroupId> 3 spring-boot-starter-webartifactId> 4 dependency> 5 6 org.springframework.bootgroupId> 7 spring-boot-starter-aopartifactId> 8 dependency> 9 10 org.springframework.bootgroupId> 11 spring-boot-starter-data-redisartifactId> 12 dependency> 13 dependencies> 14 属性配置在application.properites资源.件中添加redis相关的配置项

主要实现.式: 熟悉Redis的朋友都知道它是线程安全的，我们利.它的特性可以很轻松的实现.个分布式锁，如opsForValue().setIfAbsent(key,value) 它的作.就是如果缓存中没有当前Key则进.缓存同时返回true反之亦然； 当缓存后给key在设置个过期时间，防.因为系统崩溃.导致锁迟迟不释放形成死锁； 那么我们是不是可以这样认为当返回true我们认为它获取到锁了，在锁未释放的时候我们进.异常的抛出... 1 @Aspect 2 @Configuration 3 public class LockMethodInterceptor { 4 5 @Autowired 6 public LockMethodInterceptor(RedisLockHelper redisLockHelper, CacheKeyGenerator cacheKeyGenera 7 this.redisLockHelper = redisLockHelper;
8 this.cacheKeyGenerator = cacheKeyGenerator;
9 }
10

11 private final RedisLockHelper redisLockHelper; 12 private final CacheKeyGenerator cacheKeyGenerator; 13 14 15 @Around("execution(public * *(..)) && @annotation(com.battcn.annotation.CacheLock)") 16 public Object interceptor(ProceedingJoinPoint pjp) { 17 MethodSignature signature = (MethodSignature) pjp.getSignature(); 18 Method method = signature.getMethod(); 19 CacheLock lock = method.getAnnotation(CacheLock.class); 20 if (StringUtils.isEmpty(lock.prefix())) { 21 throw new RuntimeException("lock key don't null..."); 22 }
23 final String lockKey = cacheKeyGenerator.getLockKey(pjp);
```
```

24 String value = UUID.randomUUID().toString();
25 try {
26 27 //假设上锁成功，但是设置过期时间失效，以后拿到的都是 false final boolean success = redisLockHelper.lock(lockKey, value, lock.expire(), lock.timeU
28 if (!success) {
29 throw new RuntimeException("重复提交 ");
30 }
31 try {
32 return pjp.proceed();
33 } catch (Throwable throwable) {
34 throw new RuntimeException("系统异常 ");
35 }
36 } finally {
37 // TODO 如果演 .的话需要注释该代码 ;实际应该放开
38 redisLockHelper.unlock(lockKey, value);
39 }
40 }
41 }

RedisLockHelper通过封装成AP I.式调.，灵活度更加. 1 @Configuration 2 @AutoConfigureAfter(RedisAutoConfiguration.class) 3 public class RedisLockHelper { 4 5 6 private static final String DELIMITER = "|"; 7 8 /** 9 *如果要求.较.可以通过注.的.式分配 10 / 11 private static final ScheduledExecutorService EXECUTOR_SERVICE = Executors.newScheduledThreadP 12 13 private final StringRedisTemplate stringRedisTemplate; 14 15 public RedisLockHelper(StringRedisTemplate stringRedisTemplate) { 16 this.stringRedisTemplate = stringRedisTemplate;
17 }
18
19 /
20 21 *获取锁（存在死锁.险） *
22 * @param lockKey lockKey
23 * @param value value
24 * @param time 超时时间
25 * @param unit 过期单位
26 * @return true or false
27 */
28 public boolean tryLock(final String lockKey, final String value, final long time, final TimeUn

return stringRedisTemplate.execute((RedisCallback) connection -> connection.set(l } } /** *获取锁 * * @param lockKey lockKey

-

@param uuid UUID

-

@param timeout超时时间

-

@param unit过期单位

-

@return true or false */

public boolean lock(String lockKey, final String uuid, long timeout, final TimeUnit unit) { final long milliseconds = Expiration.from(timeout, unit).getExpirationTimeInMilliseconds() boolean success = stringRedisTemplate.opsForValue().setIfAbsent(lockKey, (System.currentTi if (success) { stringRedisTemplate.expire(lockKey, timeout, TimeUnit.SECONDS); } else { String oldVal = stringRedisTemplate.opsForValue().getAndSet(lockKey, (System.currentTi final String[] oldValues = oldVal.split(Pattern.quote(DELIMITER)); if (Long.parseLong(oldValues[0]) + 1 <= System.currentTimeMillis()) { return true; } } return success; } /**

- @see Redis Documentation: SET */ public void unlock(String lockKey, String value) { unlock(lockKey, value, 0, TimeUnit.MILLISECONDS); } /* *延迟 unlock *

-

@param lockKey key

-

@param uuid client(最好是唯.键的)

-

@param delayTime延迟时间

-

@param unit时间单位 */ public void unlock(final String lockKey, final String uuid, long delayTime, TimeUnit unit) { if (StringUtils.isEmpty(lockKey)) {

return; } if (delayTime <= 0) { doUnlock(lockKey, uuid); } else { EXECUTOR_SERVICE.schedule(() -> doUnlock(lockKey, uuid), delayTime, unit); } } /**

- @param lockKey key @p yy 84 85 * @param uuid client(最好是唯.键的) */ 86 private void doUnlock(final String lockKey, final String uuid) { 87 String val = stringRedisTemplate.opsForValue().get(lockKey); 88 final String[] values = val.split(Pattern.quote(DELIMITER)); 89 if (values.length <= 0) { 90 return; 91 } 92 if (uuid.equals(values[1])) { 93 stringRedisTemplate.delete(lockKey); 94 } 95 } 96 97 } 98 redis的提交参照 https://blog.battcn.com/2018/06/13/springboot/v2-cache-redislock/ 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ Dubbo.试18问！这些你都会吗? DeanWangJava后端 2019-09-10 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

作者：Dean Wang dubbo是什么 dubbo是.个分布式框架，远程服务调.的分布式框架，其核.部分包含：集群容错：提供基于接..法的透明远程过程调.，包括多协议.持，以及软负载均衡，失败容错，地址路由，动态配置等集群.持。远程通讯：提供对多种基于.连接的NIO框架抽象封装，包括多种线程模型，序列化，以及"请求-响应"模式的信息交换.式。.动发现：基于注册中..录服务，使服务消费.能动态的查找服务提供.，使地址透明，使服务提供.可以平滑增加或减少机器。 dubbo能做什么 透明化的远程.法调.，就像调.本地.法.样调.远程.法，只需简单配置，没有任何API侵.。软负载均衡及容错机制，可在内.替代F5等硬件负载均衡器，降低成本，减少单点。服务.动注册与发现，不再需要写死服务提供.地址，注册中.基于接.名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。 1、默认使.的是什么通信框架，还有别的选择吗? 答：默认也推荐使.netty框架，还有mina。 2、服务调.是阻塞的吗? 答：默认是阻塞的，可以异步调.，没有返回值的可以这么做。 3、.般使.什么注册中.？还有别的选择吗? 答：推荐使.zookeeper注册中.，还有Multicast注册中.,Redis注册中.,Simple注册中.. ZooKeeper的节点是通过像树.样的结构来进.维护的，并且每.个节点通过路径来标.以及访问。除此之外，每.个节点还拥有..的.些信息，包括：数据、数据.度、创建时间、修改时间等等。 4、默认使.什么序列化框架，你知道的还有哪些? 答：默认使. Hessian序列化，还有 Duddo、FastJson、Java.带序列化。hessian是.个采..进制格式传输的服务框架，相对传统 soapwebservice，更轻量，更快速。 Hessian原理与协议简析：http的协议约定了数据传输的.式，hessian也.法改变太多： 1)hessian中client与server的交互，基于http-post.式。

2. hessian将辅助信息，封装在httpheader中，.如"授权token"等，我们可以基于http-header来封装关于"安全校验""meta数据"等。hessian提供了简单的"校验"机制。 3)对于hess ian的交互核.数据，.如"调.的.法"和参数列表信息，将通过post请求的body体直接发送，格式为字节流。

4)对于hess ian的server端响应数据，将在response中通过字节流的.式直接输出。

hessian的协议本.并不复杂，在此不再赘.；所谓协议(protocol)就是约束数据的格式，client按照协议将请求信息序列化成字节序列发送给server端，server端根据协议，将数据反序列化成"对象"，然后执.指定的.法，并将.法的返回值再次按照协议序列化成字节流，响应给client，client按照协议将字节流反序列话成"对象"。 5、服务提供者能实现失效踢出是什么原理？ 答：服务失效踢出基于zookeeper的临时节点原理。 6、服务上线怎么不影响旧版本？ 答：采.多版本开发，不影响旧版本。在配置中添加version来作为版本区分 7、如何解决服务调.链过.的问题？ 答：可以结合zipkin实现分布式服务追踪。 8、说说核.的配置有哪些？ 核.配置有： 1)dubbo:service/ 2)dubbo:reference/ 3)dubbo:protocol/ 4)dubbo:registry/ 5)dubbo:application/ 6)dubbo:provider/ 7)dubbo:consumer/ 8)dubbo:method/ 9、dubbo推荐.什么协议？ 答：默认使.dubbo协议。 10、同.个服务多个注册的情况下可以直连某.个服务吗？ 答：可以直连，修改配置即可，也可以通过telnet直接某个服务。 11、dubbo在安全机制..如何解决的？ dubbo通过token令牌防...绕过注册中.直连，然后在注册中.管理授权，dubbo提供了..名单，控制服务所允许的调...。 12、集群容错怎么做？ 答：读操作建议使.Failover失败.动切换，默认重试两次其他服务器。写操作建议使.Failfast快速失败，发.次调.失败就.即报错。 13、在使.过程中都遇到了些什么问题？如何解决的？ 1)同时配置了XML和properties.件，则properties中的配置.效只有XML没有配置时，properties才.效。

2)dubbo缺省会在启动时检查依赖是否可.，不可.就抛出异常，阻.spring初始化完成，check属性默认为true。测试时有些服务不关.或者出现了循环依赖，将check设置为false

3)为了.便开发测试，线下有.个所有服务可.的注册中.，这时，如果有.个正在开发中的服务提供者注册，可能会影响消费者不能正常运.。

解决：让服务提供者开发.，只订阅服务，.不注册正在开发的服务，通过直连测试正在开发的服务。设置dubbo:registry标签的register属性为false。 4)spring2.x初始化死锁问题。 在spring解析到dubbo:service时，就已经向外暴露了服务，.spring还在接着初始化其他bean，如果这时有请求进来，并且服务的实现类.有调.applicationContext.getBean()的.法。getBean线程和spring初始化线程的锁的顺序不.样，导致了线程死锁，不能提供服务，启动不了。 解决：不要在服务的实现类中使.applicationContext.getBean();如果不想依赖配置顺序，可以将dubbo:provider的deplay属性设置为-1，使dubbo在容器初始化完成后再暴露服务。5)服务注册不上检查dubbo的jar包有没有在classpath中，以及有没有重复的jar包 检查暴露服务的spring配置有没有加载 在服务提供者机器上测试与注册中.的.络是否通 6)出现RpcException:Noprovideravailableforremoteservice异常表.没有可.的服务提供者，

a.检查连接的注册中.是否正确 b.到注册中.查看相应的服务提供者是否存在 c.检查服务提供者是否正常运. 7)出现"消息发送失败"异常

通常是接..法的传.传出参数未实现Serializable接.。 14、dubbo和dubbox之间的区别？ 答：dubbox是当当.基于dubbo上做了.些扩展，如加了服务可restful调.，更新了开源组件等。 15、你还了解别的分布式框架吗？ 答：别的还有spring的springcloud，facebook的thrift，twitter的finagle等。 16、Dubbo.持哪些协议，每种协议的应.场景，优缺点？ dubbo：单..连接和 NIO异步通讯，适合.并发.数据量的服务调.，以及消费者远.于提供者。传输协议 TCP，异步，Hessian序列化； rmi：采. JDK标准的 rmi协议实现，传输参数和返回参数对象需要实现 Serializable接.，使. java标准序列化机制，使.阻塞式短连接，传输数据包..混合，消费者和提供者个数差不多，可传.件，传输协议 TCP。多个短连接，TCP协议传输，同步传输，适.常规的远程服务调.和rmi互操作。在依赖低版本的Common-Collections包，java序列化存在安全漏洞；关注微信公众号「web_resourc」,回复Java领取2019最新资源。 webservice:基于WebService的远程调.协议，集成CXF实现，提供和原.WebService的互操作。多个短连接，基于HTTP传输，同步传输，适.系统集成和跨语.调.；http：基于Http表单提交的远程调.协议，使.Spring的HttpInvoke实现。多个短连接，传输协议HTTP，传.参数..混合，提供者个数多于消费者，需要给应.程序和浏览器JS调.； hessian：集成Hessian服务，基于HTTP通讯，采.Servlet暴露服务，Dubbo内嵌Jetty作为服务器时默认实现，提供与Hession服务互操作。多个短连接，同步HTTP传输，Hessian序列化，传.参数较.，提供者.于消费者，提供者压.较.，可传.件； memcache：基于memcached实现的RPC协议redis：基于redis实现的RPC协议 17、Dubbo集群的负载均衡有哪些策略 Dubbo提供了常.的集群策略实现，并预扩展点予以..实现。 Random LoadBalance:随机选取提供者策略，有利于动态调整提供者权重。截.碰撞率.，调.次数越多，分布越均匀；关注微信公众号「web_resourc」,回复Java领取2019最新资源。 RoundRobinLoadBalance:轮循选取提供者策略，平均分布，但是存在请求累积的问题； LeastActiveLoadBalance:最少活跃调.策略，解决慢提供者接收更少的请求；ConstantHashLoadBalance:.致性Hash策略，使相同参数请求总是发到同.提供者，.台机器宕机，可以基于虚拟节点，分摊.其他提供者，避免引起提供者的剧烈变动； 18、服务调.超时问题怎么解决 dubbo在调.服务不成功时，默认是会重试两次的。这样在服务端的处理时间超过了设定的超时时间时，就会有重复请求，.如在发邮件时，可能就会发出多份

重复邮件，执.注册请求时，就会插.多条重复的注册数据，那么怎么解决超时问题呢？如下 对于核.的服务中.，去除dubbo超时重试机制，并重新评估设置超时时间。业务处理代码必须放在服务端，客.端只做参数验证和服务调.，不涉及业务流程处理全局配置实例

当然Dubbo的重试机制其实是.常好的QOS保证，它的路由机制，是会帮你把超时的请求路由到其他机器上，.不是本机尝试，所以 dubbo的重试机器也能.定程度的保证服务的质量。但是请.定要综合线上的访问情况，给出综合的评估。原.链接： https://deanwang1943.github.io/bugs/2018/10/05/.试/饿了么/dubbo.试题/ 如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「进群」即可进...告技术交流。 推荐阅读 1. 史上最烂的项.：苦撑12年，600多万.代码...

2.请给Spr ingBoot多.些内存 3.如何从零搭建百亿流量系统？

4.

惊了！原来Web发展历史是这样的

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Dubbo.试题 Java后端 2019-11-23 点击上.Java后端，选择设为星标 优质.章，及时送达

dubbo是什么 dubbo是.个分布式框架，远程服务调.的分布式框架，其核.部分包含：集群容错：提供基于接..法的透明远程过程调.，包括多协议.持，以及软负载均衡，失败容错，地址路由，动态配置等集群.持。远程通讯：提供对多种基于.连接的NIO框架抽象封装，包括多种线程模型，序列化，以及"请求-响应"模式的信息交换.式。.动发现：基于注册中..录服务，使服务消费.能动态的查找服务提供.，使地址透明，使服务提供.可以平滑增加或减少机器。 dubbo能做什么 透明化的远程.法调.，就像调.本地.法.样调.远程.法，只需简单配置，没有任何API侵.。软负载均衡及容错机制，可在内.替代F5等硬件负载均衡器，降低成本，减少单点。服务.动注册与发现，不再需要写死服务提供.地址，注册中.基于接.名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。 1、默认使.的是什么通信框架，还有别的选择吗? 答：默认也推荐使. netty框架，还有 mina。 2、服务调.是阻塞的吗？ 答：默认是阻塞的，可以异步调.，没有返回值的可以这么做。 3、.般使.什么注册中.？还有别的选择吗？ 答：推荐使. zookeeper注册中.，还有Multicast注册中.,Redis注册中.,Simple注册中. ZooKeeper的节点是通过像树.样的结构来进.维护的，并且每.个节点通过路径来标.以及访问。除此之外，每.个节点还拥有..的.些信息，包括：数据、数据.度、创建时间、修改时间等等。 4、默认使.什么序列化框架，你知道的还有哪些？ 答：默认使. Hessian序列化，还有 Duddo、FastJson、Java .带序列化。hessian是.个采..进制格式传输的服务框架，相对传统soapwebservice，更轻量，更快速。 Hessian原理与协议简析： http的协议约定了数据传输的.式，hessian也.法改变太多： 1)hessian中client与server的交互，基于http-post.式。

2)hessian将辅助信息，封装在httpheader中，.如"授权token"等，我们可以基于http-header来封装关于"安全校验""meta数据"等。hessian提供了简单的"校验"机制。

3)对于hess ian的交互核.数据，.如"调.的.法"和参数列表信息，将通过post请求的body体直接发送，格式为字节流。

4)对于hess ian的server端响应数据，将在response中通过字节流的.式直接输出。

hessian的协议本.并不复杂，在此不再赘.；所谓协议(protocol)就是约束数据的格式，client按照协议将请求信息序列化成字节序列发送给server端，server端根据协议，将数据反序列化成"对象"，然后执.指定的.法，并将.法的返回值再次按照协议序列化成字节流，响应给client，client按照协议将字节流反序列话成"对象"。 5、服务提供者能实现失效踢出是什么原理？ 答：服务失效踢出基于 zookeeper的临时节点原理。 6、服务上线怎么不影响旧版本？ 答：采.多版本开发，不影响旧版本。在配置中添加version来作为版本区分 7、如何解决服务调.链过.的问题？ 答：可以结合 zipkin实现分布式服务追踪。 8、说说核.的配置有哪些？ 核.配置有： 1)dubbo:service/ 2)dubbo:reference/ 3)dubbo:protocol/ 4)dubbo:registry/ 5)dubbo:application/ 6)dubbo:provider/ 7)dubbo:consumer/ 8)dubbo:method/ 9、dubbo推荐.什么协议？ 答：默认使. dubbo协议。 10、同.个服务多个注册的情况下可以直连某.个服务吗？ 答：可以直连，修改配置即可，也可以通过 telnet直接某个服务。 11、dubbo在安全机制..如何解决的？ dubbo通过token令牌防...绕过注册中.直连，然后在注册中.管理授权，dubbo提供了..名单，控制服务所允许的调...。 12、集群容错怎么做？ 答：读操作建议使.Failover失败.动切换，默认重试两次其他服务器。写操作建议使.Failfast快速失败，发.次调.失败就.即报错 13、在使.过程中都遇到了些什么问题？如何解决的？ 1)同时配置了 XML和properties .件，则 properties中的配置.效

只有XML没有配置时，properties才.效。

2)dubbo缺省会在启动时检查依赖是否可.，不可.就抛出异常，阻.spring初始化完成，check属性默认为true。测试时有些服务不关.或者出现了循环依赖，将check设置为false

3)为了.便开发测试，线下有.个所有服务可.的注册中.，这时，如果有.个正在开发中的服务提供者注册，可能会影响消费

者不能正常运.。解决：让服务提供者开发.，只订阅服务，.不注册正在开发的服务，通过直连测试正在开发的服务。设置dubbo:registry标签的register属性为false。 4)spring2.x初始化死锁问题。在spring解析到dubbo:service时，就已经向外暴露了服务，.spring还在接着初始化其他bean，如果这时有请求进来，并 且服务的实现类.有调.applicationContext.getBean()的.法。getBean线程和spring初始化线程的锁的顺序不.样，导致了线程死锁，不能提供服务，启动不了。解决：不要在服务的实现类中使.applicationContext.getBean();如果不想依赖配置顺序，可以将dubbo:provider的deplay 属性设置为-1，使dubbo在容器初始化完成后再暴露服务。5)服务注册不上检查dubbo的jar包有没有在classpath中，以及有没有重复的jar包检查暴露服务的spring配置有没有加载在服务提供者机器上测试与注册中.的.络是否通 6)出现RpcException:Noprovideravailableforremoteservice异常表.没有可.的服务提供者，

a.检查连接的注册中.是否正确 b.到注册中.查看相应的服务提供者是否存在 c.检查服务提供者是否正常运. 7)出现"消息发送失败"异常通常是接..法的传.传出参数未实现Serializable接.。

14、dubbo和dubbox之间的区别？ 答：dubbox是当当.基于dubbo上做了.些扩展，如加了服务可restful调.，更新了开源组件等。 15、你还了解别的分布式框架吗？ 答：别的还有spring的springcloud，facebook的thrift，twitter的finagle等。 16、Dubbo.持哪些协议，每种协议的应.场景，优缺点？ dubbo：单..连接和NIO异步通讯，适合.并发.数据量的服务调.，以及消费者远.于提供者。传输协议TCP，异步，Hessian序列化； rmi：采.JDK标准的rmi协议实现，传输参数和返回参数对象需要实现Serializable接.，使.java标准序列化机制，使.阻塞式短连接，传输数据包..混合，消费者和提供者个数差不多，可传.件，传输协议TCP。多个短连接，TCP协议传输，同步传输，适.常规的远程服务调.和rmi互操作。在依赖低版本的Common-Collections包，java序列化存在安全漏洞；webservice:基于WebService的远程调.协议，集成CXF实现，提供和原.WebService的互操作。多个短连接，基于HTTP传输，同步传输，适.系统集成和跨语.调.；http：基于Http表单提交的远程调.协议，使.Spring的HttpInvoke实现。多个短连接，传输协议HTTP，传.参数..混合，提供者个数多于消费者，需要给应.程序和浏览器JS调.；hessian：集成Hessian服务，基于HTTP通讯，采.Servlet暴露服务，Dubbo内嵌Jetty作为服务器时默认实现，提供与Hession服务互操作。多个短连接，同步HTTP传输，Hessian序列化，传.参数较.，提供者.于消费者，提供者压.较.，可传.件； memcache：基于memcached实现的RPC协议redis：基于redis实现的RPC协 17、Dubbo集群的负载均衡有哪些策略 Dubbo提供了常.的集群策略实现，并预扩展点予以..实现。RandomLoadBalance:随机选取提供者策略，有利于动态调整提供者权重。截.碰撞率.，调.次数越多，分布越均匀；RoundRobinLoadBalance:轮循选取提供者策略，平均分布，但是存在请求累积的问题； LeastActiveLoadBalance:最少活跃调.策略，解决慢提供者接收更少的请求；ConstantHashLoadBalance:.致性Hash策略，使相同参数请求总是发到同.提供者，.台机器宕机，可以基于虚拟节点，分摊.其他提供者，避免引起提供者的剧烈变动； 18、服务调.超时问题怎么解决 dubbo在调.服务不成功时，默认是会重试两次的。这样在服务端的处理时间超过了设定的超时时间时，就会有重复请求，.如在发邮件时，可能就会发出多份重复邮件，执.注册请求时，就会插.多条重复的注册数据，那么怎么解决超时问题呢？如下对于核.的服务中.，去除dubbo超时重试机制，并重新评估设置超时时间。业务处理代码必须放在服务端，客.端只做参数验证和服务调.，不涉及业务流程处理全局配置实例

当然Dubbo的重试机制其实是.常好的QOS保证，它的路由机制，是会帮你把超时的请求路由到其他机器上，.不是本机尝试，所以dubbo的重试机器也能.定程度的保证服务的质量。但是请.定要综合线上的访问情况，给出综合的评估。【END】 推荐阅读 1.我采访了.位 Pornhub .程师，聊了这些纯纯的话题

2.

常.排序算法总结 -Java实现 3.Java：如何更优雅的处理空值？

4.

MySQL：数据库优化，可以看看这篇.章

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

GitHub获3.2W星！史上最全技术.试.册！ Java后端 2019-10-04 点击上.Java后端，选择设为星标优质.章，及时送达

安妮发.凹.寺量.位出品|公众号QbitAI 上.篇：彻底理解 Cookie，Session，Token 技术.员求职.试，单刷leetcode上的..题库可能还不够。 简历怎么写才能吸引HR的眼光，可能会被技术..问到哪些常.问题，拿到O.er之后怎样才能让..的优势最.化然后优中选优？.对这些.果果的问题，.前就职于Facebook的新加坡.哥Yangshun Tay就整理了.份.货：最全技术.员.试.册。

.试全流程需要注意的事项都在..了，包含从简历准备、.经和谈判全过程，教你如何避雷不踩坑。这份资源在GitHub上star数已经刚已破30000了，hackernews上的热度直逼700，300+.参与讨论求职雷区。这是什么神仙.货啊？ 全！太全了 先看下.致内容框架：如何准备编程.试： https://yangshun.github.io/tech-interview-handbook/coding-round-overview.试备忘录：（涉及.试前需要准备的问题，收到.试题的考虑.式、编程测试准备流程、.试总结）https://yangshun.github.io/tech-interview-handbook/cheatsheet各类算法.贴.，以及按主题分类过的最容易考到的问题： https://yangshun.github.io/tech-interview-handbook/algorithms/algorithms-introduction前端求职.试问题答案：https://github.com/yangshun/front-end-interview-handbook互联...的.试形式解析： https://yangshun.github.io/tech-interview-handbook/company-interview-formats科技巨头们的通.常.问题，以及.个..的常.通..技术问题：https://yangshun.github.io/tech-interview-handbook/behavioral-questions适合在.试结束时问考官的好问题！：https://yangshun.github.io/tech-interview-handbook/questions-to-ask做简历注意事项，让你的简历更能引起.试官注意： https://yangshun.github.io/tech-interview-handbook/resume.册在.，.试我有。具体来看，.试的前中后阶段，都有不少此前会被忽略的问题。 如何让HR 10秒内发现你简历亮点？ 如何让你的简历脱颖.出？可能要讲究.定的技巧。很多优秀.程师因为不了解HR的.作.式，..再错过.试邀请。 Yangshun认为，公司在开设职位前通常会定性职位所需具体技能，将其分为必须拥有、如果你具备我会很.兴和特殊奖励。必需技能通常包括学位、特定编程语.与编程经验。很.兴看到你具备的技能主要包括：对次要语.是否熟悉，还包括.些软技能，.如如何与团队更好交流沟通等，与主要. 作没有直接联系特殊奖励指很难得的技能/经历，不.定是.项要求，但会对.作有.。以上三者确定后，HR不寻求挑选"完美候选."，.是挑选"合适候选."就OK了。..个HR对.份简历的阅读时间，.约在10秒左右，如何让.份简历更出彩？量.位总结出.条TIPS：

简历之外附带.封求职信，告诉HR你为什么是TA要找的.简历.度最多2.，没有HR会对你的住址、.初.经历、..故事感兴趣 如果GPA过得去.定要着重处理，这是.个标准化的指标如果你.作经历丰富，那么简历上只写与所申请.作有关的就可以了 合适的联系邮箱：john.doe@gmail.com.angrybirds88@gmail.com.简历.格简单即可，标准模板也不会出错注意凸显项.经历，包括在项.中你.到了什么技术，做了什么，学习到什么等等；最好有2-3个项.符合你申请的职 位；避免使."模块代码项."之类的标题，HR看不懂啊~注意：你可能不知道HR可能会在.试你之前先在全.搜.搜你的名字，你可以先发制...排查下有没有雷区，注意保护隐私~简历注意事项，你get到了吗？Yangshun.哥哥表.，选择.公司还是.公司，其优劣都很明显。于是，这会怎对初创公司、中等规模公司和..之间.作类型、职业阶梯、迭代速度和薪酬构成等问题，Yangshun进.了对 .。他将<100.的公司定义为.规模公司，100-1000.为中等规模，>1000.为.公司。薪资待遇：..基本.资可能略.于..，但因为公司估值不明确，股权价值难以定量；中等规模公司估值清晰价值.，.. 薪资总体来说为.业最好，股票价值.。.作类型..，..以产品开发居多，..可分饰多.，前端后端Devops甚.设计；中等规模设计产品开发和.些基础设施；..为"螺丝钉"模式，内部转岗的机会.较多，基础设施的建设更常.。 .程师常..试题 由于编程语.多样，题.变化过于繁杂，题海战术只适合时间充.、有.量时间去准备...试.程师。 这份.册.有.份题..抄，作者给总结下常.题.类型，包括排列问题、.进制问题、动态规划问题(DP）、.何机构问题、图形学、哈希表（Hash table）、矩阵、堆栈、数学基础、.向对象编程等。 对于每.个可能需要注意的算法，Yangshun总结了这类问题需要注意的事项，.如在数学基础类.，提醒你如果你.的时Java和C++等类型，记得检查处理over.ow/under.ow问题。

还推荐了对应的Leetcode问题及连接，让你有的放.：

以及预测的.试问题，有效刷题，告别题海~

此外，还有特定..的常.问题，.如.歌：

Facebook：

等等。此前，量.位还推荐过.份最全leetcode中.解题攻略，带你领域..招聘原题. 别急，拿上.货再. 除了上.介绍的部分，.货中还有更多的.试常.通.问题等超多.货，需要你花费.定时间细细品读~所以请带上技术.试.册.货地址：

https://yangshun.github.io/tech-interview-handbook/resume GitHub地址： https://github.com/yangshun/tech-interview-handbook HackerNews避雷专.讨论区： https://news.ycombinator.com/item?id=20727126 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2. IDEA远程.键部署 Spring Boot到 Docker 3.这 26条，你赞同.个？ 4.7个开源的 Spring Boot前后端分离项.

5.如何设计 API接.，实现统.格式返回?

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

Javaequals和hashCode的这.个问题可以说明.吗？ Java后端 2.22.以下.章来源于.拱.兵，作者tan.拱.兵

.拱.兵 像读侦探.说.样趣读Java技术 前. 昨.留.有.个有关equals和hashcode问题。基础.试经常会碰到与之相关的问题，这不是.个复杂的问题，但很多朋友都苦于说明他们.者的关系和约束，于是写本.做单独说明，本篇.章将循序渐进 (通过举例，让记忆与理解更轻松)说明这些让你有些苦恼的问题，Let'sgo .试问题 1.Java..有了==运算符，为什么还需要equals？ ==.较的是对象地址，equals.较的是对象值 先来看.看Object类中equals.法: publicbooleanequals(Objectobj){return(this==obj);} 我们看到equals.法同样是通过==.较对象地址，并没有帮我们.较值。Java世界中 Object绝对是".祖宗"的存在，==号我们没办法改变或重写。但equals是.法，这就给了我们重写equals.法的可能，让我们实现其对值的.较: @Override publicbooleanequals(Objectobj){ }新买的电脑，每个电脑都有唯.的序列号，通常情况下，两个.模.样的电脑放在.前，你会说由于序列号不.样，这两个电脑不.样吗？ 如果我们要说两个电脑.样，通常是.较其「品牌/尺./配置」(值)，.如这样: @Override publicbooleanequals(Objectobj){ } return品牌相等&&尺.相等&&配置相等 当遇到如上场景时，我们就需要重写equals.法。这就解释了Java世界为什么有了==还有equals这个问题了. 2.equals相等和hashcode相等问题关于.者，你经常会碰到下.的两个问题:

两个对象equals相等，那他们hashCode相等吗?

两个对象hashCode相等，那他们equals相等吗?

为了说明上.两个问题的结论，这.举.个不太恰当的例.，只为.便记忆，我们将equals.作.个单词的拼写；hashCode.作.个单词的发.，在相同语境下: sea/sea「.海」，两个单词拼写.样，所以 equals相等，他们读. /si./也.样，所以 hashCode就相等，这就回答了第. 个问题: 两个对象equals相等，那他们hashCode.定也相等 sea/see「.海/看」，两个单词的读./si./.样，显然单词是不.样的，这就回答了第.个问题: 两个对象hashCode相等，那他们equals不.定相等 查看Object类的hashCode.法: publicnativeinthashCode(); 继续查看该.法的注释，明确写明关于该.法的约束

其实在这个结果的背后，还有的是关于重写equals.法的约束 3.重写equals有哪些约束？关于重写equals.法的约束，同样在该.法的注释中写的很清楚了，我在这.再说明.下:

.橙红绿.蓝紫，七彩以.列；哆来咪发唆拉西,.曲安哥拉，这些规则不是.来背诵的，只是在你需要重写 equals.法时，打开JDK查看该.法，按照准则重写就好 4.什么时候需要我们重写hashCode？为了.较值，我们重写equals.法，那什么时候.需要重写hashCode.法呢？ 通常只要我们重写equals.法就要重写hashCode.法 为什么会有这样的约束呢？按照上.讲的原则，两个对象 equals相等，那他们的 hashCode.定也相等。如果我们只重写 equals.法.不重写hashCode.法，看看会发.什么，举个例.来看:定义学.类，并通过IDE只帮我们.成equals.法: publicclassStudent{ privateStringname; privateintage; @Override publicbooleanequals(Objecto) {if(this==o)returntrue;if(o==null||getClass()!=o.getClass())returnfalse;Studentstudent=(Student)o; returnage==student.age&& Objects.equals(name,student.name);}} 编写测试代码: Studentstudent1=newStudent(); student1.setName(".拱.兵"); student1.setAge(18); Studentstudent2=newStudent(); student2.setName(".拱.兵"); student2.setAge(18); System.out.println("student1.equals(student2)的结果是："+student1.equals(student2)); Setstudents=newHashSet(); students.add(student1); students.add(student2);System.out.println("StudentSet集合.度是："+students.size()); Map<Student,java.lang.String>map=newHashMap<Student,java.lang.String>(); map.put(student1,"student1"); map.put(student2,"student2");System.out.println("StudentMap集合.度是："+map.keySet().size()); 查看运.结果: student1.equals(student2)的结果是：true StudentSet集合.度是：2 StudentMap集合.度是：2 很显然，按照集合Set和Map加.元素的标准来看，student1和student2是两个对象，因为在

调.他们的put(Setadd.法的背后也是HashMap的put).法时，会先判断hash值是否相等，这个.伙伴们打开JDK..查看吧 所以我们继续重写Student类的hashCode.法: @Override publicinthashCode(){returnObjects.hash(name,age);} 重新运.上.的测试，查看结果: student1.equals(student2)的结果是：true StudentSet集合.度是：1 StudentMap集合.度是：1 得到我们预期的结果，这也就是为什么通常我们重写equals.法为什么最好也重写hashCode.法的原因

如果你在使. Lombok，不知道你是否注意到 Lombok只有.个 @EqualsAndHashCode注解，.没有拆分成 @Equals和@HashCode两个注解，想了解更多Lombok的内容，也可以查看我之前写的.章Lomok使.详解 另外通过IDE快捷键.成重写.法时，你也会看到这两个.法放在.起，.不是像getter和setter那样分开

以上两点都是隐形的规范约束，希望.家也严格遵守这个规范，以防带来不必要的.烦，记忆的.式有多样，如果记不住这个.字约束，脑海中记住上.的图你也就懂了5.重写hashCode为什么总有31这个数字？细.的朋友可能注意到，我上.重写hashCode的.法很简答，就是.了Objects.hash.法，进去查看..的.法: publicstaticinthashCode(Objecta[]) {if(a==null)return0; intresult=1; for(Objectelement:a)result=31*result+(element==null?0:element.hashCode()); returnresult;} 这.通过31来计算对象hash值在如何妙.Spring数据绑定？.章末尾提到的在HandlerMethodArgumentResolverComposite类中有这样.个成员变量: privatefinalMap<MethodParameter,HandlerMethodArgumentResolver>argumentResolverCache= newConcurrentHashMap<MethodParameter,HandlerMethodArgumentResolver>(256) Map的 key是MethodParameter，根据我们上.的分析，这个类.定也会重写 equals和hashCode.法，进去查看发现，hashCode的计算也.到了31这个数字 @Override publicbooleanequals(Objectother){if(this==other){ returntrue;}if(!(otherinstanceofMethodParameter)){ returnfalse;}MethodParameterotherParam=(MethodParameter)other; return(this.parameterIndex==otherParam.parameterIndex&&getMember().equals(otherParam.getMember())); } @Override publicinthashCode(){return(getMember().hashCode()*31+this.parameterIndex);} 为什么计算hash值要.到31这个数字呢？我在.上看到.篇不错的.章，分享给.家，作为科普，可以简单查看.下: StringhashCode.法为什么选择数字31作为乘. 总结 如果还对equals和hashCode关系及约束含混，我们只需要按照上述步骤逐步回忆即可，更好的是直接查看JDK源码；另外拿出实际的例.来反推验证是.常好的办法。如果你还有相关疑问，也可以留.探讨。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，欢迎添加.编微信「focusoncode」，每.朋友圈更新.篇.质量技术博.（..告）。 ↓扫描.维码添加.编↓

推荐阅读 1..费送新款iPad，包邮！ 2.18个.例带你掌握Java8.期时间处理！

3.安利.款 IDEA中强.的代码.成利器 4.如何获取靠谱的新型冠状病毒疫情

阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Java并发编程73道.试题及答案 Java后端 2019-12-09 点击上.Java后端，选择设为星标 优质.章，及时送达

原.出处： https://blog.csdn.net/qq_34039315/article/details/7854931 1、在java中守护线程和本地线程区别？ java中的线程分为两种：守护线程（Daemon）和..线程（User）。 任何线程都可以设置为守护线程和..线程，通过.法Thread.setDaemon(boolon)；true则把该线程设置为守护线程，反之则 为..线程。Thread.setDaemon()必须在Thread.start()之前调.，否则运.时会抛出异常。 两者的区别：唯.的区别是判断虚拟机(JVM)何时离开，Daemon是为其他线程提供服务，如果全部的UserThread已经撤离，Daemon没有可服务的线程，JVM撤离。也可以理解为守护线程是JVM.动创建的线程（但不.定），..线程是程序创建的线程；.如JVM的垃圾回收线程是.个守护线程，当所有线程已经撤离，不再产.垃圾，守护线程.然就没事可.了，当垃圾回收线程是Java虚拟机上仅剩的线程时，Java虚拟机会.动离开。 扩展：ThreadDump打印出来的线程信息，含有daemon字样的线程即为守护进程，可能会有：服务守护进程、编译守护进程、 windows下的监听Ctrl+break的守护进程、Finalizer守护进程、引.处理守护进程、GC守护进程。 2、线程与进程的区别？ 进程是操作系统分配资源的最.单元，线程是操作系统调度的最.单元。 .个程序.少有.个进程,.个进程.少有.个线程。 3、什么是多线程中的上下.切换？ 多线程会共同使..组计算机上的CPU，.线程数.于给程序分配的CPU数量时，为了让各个线程都有执.的机会，就需要轮转使.CPU。不同的线程切换使.CPU发.的切换数据等就是上下.切换。 4、死锁与活锁的区别，死锁与饥饿的区别？ 死锁：是指两个或两个以上的进程（或线程）在执.过程中，因争夺资源.造成的.种互相等待的现象，若.外.作.，它们都将.法推进下去。 产.死锁的必要条件：

互斥条件：所谓互斥就是进程在某.时间内独占资源。 请求与保持条件：.个进程因请求资源.阻塞时，对已获得的资源保持不放。

不剥夺条件:进程已获得资源，在末使.完之前，不能强.剥夺。 循环等待条件:若.进程之间形成.种头尾相接的循环等待资源关系。活锁：任务或者执.者没有被阻塞，由于某些条件没有满.，导致.直重复尝试，失败，尝试，失败。活锁和死锁的区别在于，处于活锁的实体是在不断的改变状态，所谓的"活"，.处于死锁的实体表现为等待；活锁有可能.. 解开，死锁则不能。饥饿：.个或者多个线程因为种种原因.法获得所需要的资源，导致.直.法执.的状态。Java中导致饥饿的原因：

.优先级线程吞噬所有的低优先级线程的CPU时间。线程被永久堵塞在.个等待进.同步块的状态，因为其他线程总是能在它之前持续地对该同步块进.访问。线程在等待.个本.也处于永久等待完成的对象(.如调.这个对象的wait.法)，因为其他线程总是被持续地获得唤醒。 5、Java中.到的线程调度算法是什么？ 采.时间.轮转的.式。可以设置线程的优先级，会映射到下层的系统上.的优先级上，如.特别需要，尽量不要.，防.线程饥饿。 6、什么是线程组，为什么在Java中不推荐使.？ ThreadGroup类，可以把线程归属到某.个线程组中，线程组中可以有线程对象，也可以有线程组，组中还可以有线程，这样的组织结构有点类似于树的形式。 为什么不推荐使.？因为使.有很多的安全隐患吧，没有具体追究，如果需要使.，推荐使.线程池。 7、为什么使.Executor框架？ I.每次执.任务创建线程newThread().较消耗性能，创建.个线程是.较耗时、耗资源的。 II.调. new Thread()创建的线程缺乏管理，被称为野线程，.且可以.限制的创建，线程之间的相互竞争会导致过多占.系统资源.导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。 III.接使.newThread()启动的线程不利于扩展，.如定时执.、定期执.、定时定期执.、线程中断等都不便实现。

8、在Java中Executor和Executors的区别？ Executors.具类的不同.法按照我们的需求创建了不同的线程池，来满.业务的需求。Executor接.对象能执.我们的线程任务。 ExecutorService接.继承了Executor接.并进.了扩展，提供了更多的.法我们能获得任务执.的状态并且可以获取任务的返 回值。使.ThreadPoolExecutor可以创建.定义线程池。Future表.异步计算的结果，他提供了检查计算是否完成的.法，以等待计算的完成，并可以使.get().法获取计算的结果。 9、什么是原.操作？在JavaConcurrencyAPI中有哪些原.类(atomicclasses)？ 原.操作（atomicoperation）意为"不可被中断的.个或.系列操作"。 处理器使.基于对缓存加锁或总线加锁的.式来实现多处理器之间的原.操作。 在Java中可以通过锁和循环CAS的.式来实现原.操作。CAS操作.Compare&Set，或是Compare&Swap，现在.乎所有的CPU指令都.持CAS的原.操作。 原.操作是指.个不受其他操作影响的操作任务单元。原.操作是在多线程环境下避免数据不.致必须的.段。 int++并不是.个原.操作，所以当.个线程读取它的值并加1时，另外.个线程有可能会读到之前的值，这就会引发错误。 为了解决这个问题，必须保证增加操作是原.的，在JDK1.5之前我们可以使.同步技术来做到这.点。到JDK1.5，java.util.concurrent.atomic包提供了int和long类型的原.包装类，它们可以.动的保证对于他们的操作是原.的并且不需要使.同步。 java.util.concurrent这个包..提供了.组原.类。其基本的特性就是在多线程环境下，当有多个线程同时执.这些类的实例包含的.法时，具有排他性，即当某个线程进..法，执.其中的指令时，不会被其他线程打断，.别的线程就像.旋锁.样，.直等到该.法执.完成，才由JVM从等待队列中选择.个另.个线程进.，这只是.种逻辑上的理解。 原.类：AtomicBoolean，AtomicInteger，AtomicLong，AtomicReference原.数组：AtomicIntegerArray，AtomicLongArray，AtomicReferenceArray原.属性更新器：AtomicLongFieldUpdater，AtomicIntegerFieldUpdater，AtomicReferenceFieldUpdater解决ABA问题的原.类：AtomicMarkableReference（通过引..个boolean来反映中间有没有变过），AtomicStampedReference（通过引..个int来累加来反映中间有没有变过） 10、JavaConcurrencyAPI中的Lock接.(Lockinterface)是什么？对.同步它有什么优势？ Lock接..同步.法和同步块提供了更具扩展性的锁操作。 他们允许更灵活的结构，可以具有完全不同的性质，并且可以.持多个相关类的条件对象。 它的优势有：

可以使锁更公平可以使线程在等待锁的时候响应中断可以让线程尝试获取锁，并在.法获取锁的时候.即返回或者等待.段时间可以在不同的范围，以不同的顺序获取和释放锁整体上来说Lock是synchronized的扩展版，Lock提供了.条件的、可轮询的(tryLock.法)、定时的(tryLock带参.法)、可中断的(lockInterruptibly)、可多条件队列的(newCondition.法)锁操作。另外Lock的实现类基本都.持.公平锁(默认)和公平锁，synchronized只.持.公平锁，当然，在.部分情况下，.公平锁是.效的选择。 11、什么是Executors框架？ Executor框架是.个根据.组执.策略调.，调度，执.和控制的异步任务的框架。 .限制的创建线程会引起应.程序内存溢出。所以创建.个线程池是个更好的的解决.案，因为可以限制线程的数量并且可以回收再利.这些线程。利.Executors框架可以.常.便的创建.个线程池。 12、什么是阻塞队列？阻塞队列的实现原理是什么？如何使.阻塞队列来实现.产者-消费者模型？ 阻塞队列（BlockingQueue）是.个.持两个附加操作的队列。 这两个附加的操作是：在队列为空时，获取元素的线程会等待队列变为.空。当队列满时，存储元素的线程会等待队列可..阻塞队列常.于.产者和消费者的场景，.产者是往队列.添加元素的线程，消费者是从队列.拿元素的线程。阻塞队列就是.产者存放元素的容器，.消费者也只从容器.拿元素。 JDK7提供了7个阻塞队列。分别是： ArrayBlockingQueue：.个由数组结构组成的有界阻塞队列。 LinkedBlockingQueue：.个由链表结构组成的有界阻塞队列。 PriorityBlockingQueue：.个.持优先级排序的.界阻塞队列。 DelayQueue：.个使.优先级队列实现的.界阻塞队列。 SynchronousQueue：.个不存储元素的阻塞队列。 LinkedTransferQueue：.个由链表结构组成的.界阻塞队列。 LinkedBlockingDeque：.个由链表结构组成的双向阻塞队列。 Java5之前实现同步存取时，可以使.普通的.个集合，然后

在使.线程的协作和线程同步可以实现.产者，消费者模式，主要的技术就是.好，wait ,notify,notifyAll,sychronized这些关键字。.在java5之后，可以使.阻塞队列来实现，此.式..简少了代码量，使得多线程编程更加容易，安全..也有保障。BlockingQueue接.是Queue的.接.，它的主要.途并不是作为容器，.是作为线程同步的的.具，因此他具有.个很明显的特性，当.产者线程试图向BlockingQueue放.元素时，如果队列已满，则线程被阻塞，当消费者线程试图从中取出.个元素时，如果队列为空，则该线程会被阻塞，正是因为它所具有这个特性，所以在程序中多个线程交替向BlockingQueue中放.元素，取出元素，它可以很好的控制线程之间的通信。 阻塞队列使.最经典的场景就是socket客.端数据的读取和解析，读取数据的线程不断将数据放.队列，然后解析线程不断从队列取数据解析。 13、什么是Callable和Future? Callable接.类似于Runnable，从名字就可以看出来了，但是Runnable不会返回结果，并且.法抛出返回结果的异常，.Callable功能更强..些，被线程执.后，可以返回值，这个返回值可以被Future拿到，也就是说，Future可以拿到异步执.任务的返回值。可以认为是带有回调的Runnable。 Future接.表.异步任务，是还没有完成的任务给出的未来结果。所以说Callable.于产.结果，Future.于获取结果。 14、什么是FutureTask?使.ExecutorService启动任务。 在Java并发程序中FutureTask表..个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等.法。只有当运算完成的时候结果才能取回，如果运算尚未完成get.法将会阻塞。.个FutureTask对象可以对调.了Callable和 Runnable的对象进.包装，由于FutureTask也是调.了Runnable接.所以它可以提交给Executor来执.。 15、什么是并发容器的实现？ 何为同步容器：可以简单地理解为通过synchronized来实现同步的容器，如果有多个线程调.同步容器的.法，它们将会串.执.。.如Vector，Hashtable，以及Collections.synchronizedSet，synchronizedList等.法返回的容器。 可以通过查看Vector，Hashtable等这些同步容器的实现代码，可以看到这些容器实现线程安全的.式就是将它们的状态封装起来，并在需要同步的.法上加上关键字synchronized。 并发容器使.了与同步容器完全不同的加锁策略来提供更.的并发性和伸缩性，例如在ConcurrentHashMap中采.了.种粒度更细的加锁机制，可以称为分段锁，在这种锁机制下，允许任意数量的读线程并发地访问map，并且执.读操作的线程和写操作的线程也可以并发的访问map，同时允许.定数量的写操作线程并发地修改map，所以它可以在并发环境下实现更.的吞吐量。 16、多线程同步和互斥有.种实现.法，都是什么？ 线程同步是指线程之间所具有的.种制约关系，.个线程的执.依赖另.个线程的消息，当它没有得到另.个线程的消息时应等待，直到消息到达时才被唤醒。 线程互斥是指对于共享的进程系统资源，在各单个线程访问时的排它性。当有若.个线程都要使.某.共享资源时，任何时刻最多只允许.个线程去使.，其它要使.该资源的线程必须等待，直到占.资源者释放该资源。线程互斥可以看成是.种特殊的线程同步。 线程间的同步.法.体可分为两类：..模式和内核模式。顾名思义，内核模式就是指利.系统内核对象的单.性来进.同步，使.时需要切换内核态与..态，...模式就是不需要切换到内核态，只在..态完成操作。 ..模式下的.法有：原.操作（例如.个单.的全局变量），临界区。内核模式下的.法有：事件，信号量，互斥量。 17、什么是竞争条件？你怎样发现和解决竞争？ 当多个进程都企图对共享数据进.某种处理，.最后的结果.取决于进程运.的顺序时，则我们认为这发.了竞争条件（race condition）。 18、你将如何使.threaddump？你将如何分析Threaddump?

新建状态（New） .new语句创建的线程处于新建状态，此时它和其他Java对象.样，仅仅在堆区中被分配了内存。

就绪状态（Runnable） 当.个线程对象创建后，其他线程调.它的start().法，该线程就进.就绪状态，Java虚拟机会为它创建.法调.栈和程序计数器。处于这个状态的线程位于可运.池中，等待获得CPU的使.权。

运.状态（Running） 处于这个状态的线程占.CPU，执.程序代码。只有处于就绪状态的线程才有机会转到运.状态。

阻塞状态（Blocked）

阻塞状态是指线程因为某些原因放弃CPU，暂时停.运.。当线程处于阻塞状态时，Java虚拟机不会给线程分配CPU。直到线程重新进.就绪状态，它才有机会转到运.状态。 阻塞状态可分为以下3种： ①位于对象等待池中的阻塞状态（Blocked in object’s wait pool）：当线程处于运.状态时，如果执.了某个对象的 wait().法，Java虚拟机就会把线程放到这个对象的等待池中，这涉及到“线程通信”的内容。②位于对象锁池中的阻塞状态（Blocked inobject’slockpool）：当线程处于运.状态时，试图获得某个对象的同步锁时，如果该对象的同步锁已经被其他线程占.，Java虚拟机就会把这个线程放到这个对象的锁池中，这涉及到“线程同步”的内容。 ③其他阻塞状态（Otherwise Blocked）：当前线程执.了sleep().法，或者调.了其他线程的join().法，或者发出了 I/O请求时，就会进.这个状态。

死亡状态（Dead） 当线程退出run().法时，就进.死亡状态，该线程结束.命周期。 我们运.之前的那个死锁代码SimpleDeadLock.java，然后尝试输出信息*(/这是注释，作者..加的/)*： */时间，jvm信息*/2017-11-0117:36:28 FullthreaddumpJavaHotSpot(TM)64-BitServerVM(25.144-b01mixedmode): */线程名称：DestroyJavaVM /线程名称：DestroyJavaVM编号：#13优先级：5系统优先级：0 jvm内部线程id：0x0000000001c88800对应系统线程id（NativeThreadID）：0x1c18线程状态：waitingoncondition[0x0000000000000000]（等待某个条件）线程详细状态：java.lang.Thread.State:RUNNABLE及之后所*

有/"DestroyJavaVM"#13prio=5os_prio=0tid=0x0000000001c88800nid=0x1c18waitingoncondition[0x0000000000000000] java.lang.Thread.State:RUNNABLE "Thread-1"#12prio=5os_prio=0tid=0x0000000018d49000nid=0x17b8waitingformonitorentry[0x0000000019d7f000] */线程状态：阻塞（在对象同步上）代码位置：atcom.leo.interview.SimpleDeadLock$B.run(SimpleDeadLock.java:56)等待锁：0x00000000d629b4d8已经获得锁：0x00000000d629b4e8/*java.lang.Thread.State:BLOCKED(onobjectmonitor)atcom.leo.interview.SimpleDeadLock$B.run(SimpleDeadLock.java:56)-waitingtolock<0x00000000d629b4d8>(ajava.lang.Object)-locked<0x00000000d629b4e8>(ajava.lang.Object) "Thread-0"#11prio=5os_prio=0tid=0x0000000018d44000nid=0x1ebcwaitingformonitorentry[0x000000001907f000]java.lang.Thread.State:BLOCKED(onobjectmonitor)atcom.leo.interview.SimpleDeadLock$A.run(SimpleDeadLock.java:34)-waitingtolock<0x00000000d629b4e8>(ajava.lang.Object)-locked<0x00000000d629b4d8>(ajava.lang.Object) "ServiceThread"#10daemonprio=9os_prio=0tid=0x0000000018ca5000nid=0x1264runnable[0x0000000000000000]java.lang.Thread.State:RUNNABLE "C1CompilerThread2"#9daemonprio=9os_prio=2tid=0x0000000018c46000nid=0xb8cwaitingoncondition[0x0000000000000000]java.lang.Thread.State:RUNNABLE "C2CompilerThread1"#8daemonprio=9os_prio=2tid=0x0000000018be4800nid=0x1db4waitingoncondition[0x0000000000000000]java.lang.Thread.State:RUNNABLE "C2CompilerThread0"#7daemonprio=9os_prio=2tid=0x0000000018be3800nid=0x810waitingoncondition[0x0000000000000000]java.lang.Thread.State:RUNNABLE "MonitorCtrl-Break"#6daemonprio=5os_prio=0tid=0x0000000018bcc800nid=0x1c24runnable[0x00000000193ce000]java.lang.Thread.State:RUNNABLE atjava.net.SocketInputStream.socketRead0(NativeMethod)atjava.net.SocketInputStream.socketRead(SocketInputStream.java:116)atjava.net.SocketInputStream.read(SocketInputStream.java:171)atjava.net.SocketInputStream.read(SocketInputStream.java:141)atsun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:284)atsun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:326)atsun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)-locked<0x00000000d632b928>(ajava.io.InputStreamReader)atjava.io.InputStreamReader.read(InputStreamReader.java:184)atjava.io.BufferedReader.fill(BufferedReader.java:161)atjava.io.BufferedReader.readLine(BufferedReader.java:324)-locked<0x00000000d632b928>(ajava.io.InputStreamReader)atjava.io.BufferedReader.readLine(BufferedReader.java:389)atcom.intellij.rt.execution.application.AppMainV2$1.run(AppMainV2.java:64) "AttachListener"#5daemonprio=5os_prio=2tid=0x0000000017781800nid=0x524runnable[0x0000000000000000]java.lang.Thread.State:RUNNABLE "SignalDispatcher"#4daemonprio=9os_prio=2tid=0x000000001778f800nid=0x1b08waitingoncondition[0x0000000000000000]java.lang.Thread.State:RUNNABLE "Finalizer"#3daemonprio=8os_prio=1tid=0x000000001776a800nid=0xdacinObject.wait() [0x0000000018b6f000]java.lang.Thread.State:WAITING(onobjectmonitor)j l Obj i(N i Mhd) atjava.lang.Object.wait(NativeMethod)-waitingon<0x00000000d6108ec8>(ajava.lang.ref.ReferenceQueue$Lock)atjava.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)-locked<0x00000000d6108ec8>(ajava.lang.ref.ReferenceQueue$Lock)atjava.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:164)atjava.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:209) "ReferenceHandler"#2daemonprio=10os_prio=2tid=0x0000000017723800nid=0x1670inObject.wait() [0x00000000189ef000]java.lang.Thread.State:WAITING(onobjectmonitor)atjava.lang.Object.wait(NativeMethod)-waitingon<0x00000000d6106b68>(ajava.lang.ref.Reference$Lock)atjava.lang.Object.wait(Object.java:502)atjava.lang.ref.Reference.tryHandlePending(Reference.java:191)-locked<0x00000000d6106b68>(ajava.lang.ref.Reference$Lock)atjava.lang.ref.Reference$ReferenceHandler.run(Reference.java:153) "VMThread"os_prio=2tid=0x000000001771b800nid=0x604runnable "GCtaskthread#0(ParallelGC)"os_prio=0tid=0x0000000001c9d800nid=0x9f0runnable "GCtaskthread#1(ParallelGC)"os_prio=0tid=0x0000000001c9f000nid=0x154crunnable "GCtaskthread#2(ParallelGC)"os_prio=0tid=0x0000000001ca0800nid=0xcd0runnable "GCtaskthread#3(ParallelGC)"os_prio=0tid=0x0000000001ca2000nid=0x1e58runnable

"VMPeriodicTaskThread"os_prio=2tid=0x0000000018c5a000nid=0x1b58waitingoncondition JNIglobalreferences:33 /*此处可以看待死锁的相关信息！*/FoundoneJava-leveldeadlock: ========================== "Thread-1": waitingtolockmonitor0x0000000017729fc8(object0x00000000d629b4d8,ajava.lang.Object), whichisheldby"Thread-0" "Thread-0": waitingtolockmonitor0x0000000017727738(object0x00000000d629b4e8,ajava.lang.Object), whichisheldby"Thread-1" Javastackinformationforthethreadslistedabove: == "Thread-1": atcom.leo.interview.SimpleDeadLock$B.run(SimpleDeadLock.java:56)-waitingtolock<0x00000000d629b4d8> (ajava.lang.Object)-locked<0x00000000d629b4e8>(ajava.lang.Object) "Thread-0": atcom.leo.interview.SimpleDeadLock$A.run(SimpleDeadLock.java:34)-waitingtolock<0x00000000d629b4e8> (ajava.lang.Object)-locked<0x00000000d629b4d8>(ajava.lang.Object) Found1deadlock. /*内存使.状况，详情得看JVM..的书*/HeapPSYoungGentotal37888K,used4590K[0x00000000d6100000,0x00000000d8b00000,0x0000000100000000) edenspace32768K,14%used[0x00000000d6100000,0x00000000d657b968,0x00000000d8100000)fromspace5120K,0%used[0x00000000d8600000,0x00000000d8600000,0x00000000d8b00000)to space5120K,0%used[0x00000000d8100000,0x00000000d8100000,0x00000000d8600000)ParOldGentotal86016K,used0K[0x0000000082200000,0x0000000087600000,0x00000000d6100000)objectspace86016K,0%used[0x0000000082200000,0x0000000082200000,0x0000000087600000)Metaspaceused3474K,capacity4500K,committed4864K,reserved1056768K classspaceused382K,capacity388K,committed512K,reserved1048576K p ,py, , 19、为什么我们调.start().法时会执.run().法，为什么我们不能直接调.run().法？ 当你调.start().法时你将创建新的线程，并且执.在run().法.的代码。 但是如果你直接调.run().法，它不会创建新的线程也不会执.调.线程的代码，只会把run.法当作普通.法去执.。 20、Java中你怎样唤醒.个阻塞的线程？ 在Java发展史上曾经使.suspend()、resume().法对于线程进.阻塞唤醒，但随之出现很多问题，.较典型的还是死锁问题。 解决.案可以使.以对象为.标的阻塞，即利.Object类的wait()和notify().法实现线程阻塞。 .先，wait、notify.法是针对对象的，调.任意对象的wait().法都将导致线程阻塞，阻塞的同时也将释放该对象的锁，相应地，调.任意对象的notify().法则将随机解除该对象阻塞的线程，但它需要重新获取改对象的锁，直到获取成功才能往下执.；其次，wait、notify.法必须在synchronized块或.法中被调.，并且要保证同步块或.法的锁对象与调.wait、notify.法的对象是同.个，如此.来在调.wait之前当前线程就已经成功获取某对象的锁，执.wait阻塞后当前线程就将之前获取的对象锁释放。 21、在Java中CycliBarriar和CountdownLatch有什么区别？ CyclicBarrier可以重复使.，.CountdownLatch不能重复使.。 Java的concurrent包..的CountDownLatch其实可以把它看作.个计数器，只不过这个计数器的操作是原.操作，同时只能有.个线程去操作这个计数器，也就是同时只能有.个线程去减这个计数器..的值。 你可以向CountDownLatch对象设置.个初始的数字作为计数值，任何调.这个对象上的await().法都会阻塞，直到这个计数器的计数值被其他的线程减为0为.。 所以在当前计数到达零之前，await.法会.直受阻塞。之后，会释放所有等待的线程，await的所有后续调.都将.即返回。这种现象只出现.次——计数.法被重置。如果需要重置计数，请考虑使.CyclicBarrier。CountDownLatch的.个.常典型的应.场景是：有.个任务想要往下执.，但必须要等到其他的任务执.完毕后才可以继续往下执.。假如我们这个想要继续往下执.的任务调..个CountDownLatch对象的await().法，其他的任务执.完..的任务后调.同.个CountDownLatch对象上的countDown().法，这个调.await().法的任务将.直阻塞等待，直到这个CountDownLatch对象的计数值减到0为. CyclicBarrier.个同步辅助类，它允许.组线程互相等待，直到到达某个公共屏障点 (common barrier point)。在涉及.组固定..的线程的程序中，这些线程必须不时地互相等待，此时 CyclicBarrier很有.。因为该 barrier在释放等待线程后可以重.，所以称它为循环的barrier。 22、什么是不可变对象，它对写并发应.有什么帮助？ 不可变对象(Immutable Objects)即对象.旦被创建它的状态（对象的数据，也即对象属性值）就不能改变，反之即为可变对象(MutableObjects)。不可变对象的类即为不可变类(Immutable Class)。Java平台类库中包含许多不可变类，如String、基本类型的包装类、 BigInteger和BigDecimal等。 不可变对象天.是线程安全的。它们的常量（域）是在构造函数中创建的。既然它们的状态.法修改，这些常量永远不会变。 不可变对象永远是线程安全的。 只有满.如下状态，.个对象才是不可变的；

它的状态不能在创建后再被修改； 所有域都是final类型；并且， 它被正确创建（创建期间没有发.this引.的逸出）。 23、什么是多线程中的上下.切换？ 在上下.切换过程中，CPU会停.处理当前运.的程序，并保存当前程序运.的具体位置以便之后继续运.。从这个.度来看，上下.切换有点像我们同时阅读.本书，在来回切换书本的同时我们需要记住每本书当前读到的.码。在程序中，上下.切换过程中的".码"信息是保存在进程控制块（PCB）中的。PCB还经常被称作"切换桢"（switchframe）。".码"信息会.直保存到CPU的内存中，直到他们被再次使.。上下.切换是存储和恢复CPU状态的过程，它使得线程执.能够从中断点恢复执.。上下.切换是多任务操作系统和多线程环境的基本特征。 24、Java中.到的线程调度算法是什么？ 计算机通常只有.个CPU,在任意时刻只能执..条机器指令,每个线程只有获得CPU的使.权才能执.指令.所谓多线程的并发运.,其实是指从宏观上看,各个线程轮流获得CPU的使.权,分别执.各.的任务.在运.池中,会有多个处于就绪状态的线程在等待CPU,JAVA虚拟机的.项任务就是负责线程的调度,线程调度是指按照特定机制为多个线程分配CPU的使.权. 有两种调度模型：分时调度模型和抢占式调度模型。 分时调度模型是指让所有的线程轮流获得cpu的使.权,并且平均

分配每个线程占.的CPU的时间.这个也.较好理解。 java虚拟机采.抢占式调度模型，是指优先让可运.池中优先级.的线程占.CPU，如果可运.池中的线程优先级相同，那么就随机选择.个线程，使其占.CPU。处于运.状态的线程会.直运.，直.它不得不放弃CPU。 25、什么是线程组，为什么在Java中不推荐使.？ 线程组和线程池是两个不同的概念，他们的作.完全不同，前者是为了.便线程的管理，后者是为了管理线程的.命周期，复.线程，减少创建销毁线程的开销。 26、为什么使.Executor框架.使.应.创建和管理线程好？ 为什么要使.Executor线程池框架 I.每次执.任务创建线程newThread().较消耗性能，创建.个线程是.较耗时、耗资源的。 II.调. new Thread()创建的线程缺乏管理，被称为野线程，.且可以.限制的创建，线程之间的相互竞争会导致过多占.系

统资源.导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。 III.直接使.newThread()启动的线程不利于扩展，.如定时执.、定期执.、定时定期执.、线程中断等都不便实现。 使.Executor线程池框架的优点 I.能复.已存在并空闲的线程从.减少线程对象的创建从.减少了消亡线程的开销。 II.可有效控制最.并发线程数，提.系统资源使.率，同时避免过多资源竞争。 III.框架中已经有定时、定期、单线程、并发数控制等功能。

综上所述使.线程池框架Executor能更好的管理线程、提供系统资源使.率。 27、java中有.种.法可以实现.个线程？

继承Thread类

实现Runnable接.

实现Callable接.，需要实现的是call().法

28、如何停..个正在运.的线程？

使.共享变量的.式 在这种.式中，之所以引.共享变量，是因为该变量可以被多个执.相同任务的线程.来作为是否中断的信号，通知中断线程的执.。

使.interrupt.法终.线程如果.个线程由于等待某些事件的发..被阻塞，.该怎样停.该线程呢？这种情况经常会发.，.如当.个线程由于需要等候键盘输..被阻塞，或者调.Thread.join().法，或者Thread.sleep().法，在.络中调.ServerSocket.accept().法，或者调.了DatagramSocket.receive().法时，都有可能导致线程阻塞，使线程处于处于不可运.状态时，即使主程序中将该线程的共享变量设置为true，但该线程此时根本.法检查循环标志，当然也就.法.即中断。这.我们给出的建议是，不要使.stop().法，.是使.Thread提供的interrupt().法，因为该.法虽然不会中断.个正在运.的线程，但是它可以使.个被阻塞的线程抛出.个中断异常，从.使线程提前结束阻塞状态，退出堵塞代码。

29、 notify()和notifyAll()有什么区别？ 当.个线程进.wait之后，就必须等其他线程notify/notifyall,使.notifyall,可以唤醒所有处于wait状态的线程，使其重新进.锁的争夺队列中，.notify只能唤醒.个。 如果没把握，建议notifyAll，防.notigy因为信号丢失.造成程序异常。 30、什么是Daemon线程？它有什么意义？ 所谓后台(daemon)线程，是指在程序运.的时候在后台提供.种通.服务的线程，并且这个线程并不属于程序中不可或缺的部分。因此，当所有的.后台线程结束时，程序也就终.了，同时会杀死进程中的所有后台线程。 反过来说，只要有任何.后台线程还在运.，程序就不会终.。必须在线程启动之前调.setDaemon().法，才能把它设置为后台线程。注意：后台进程在不执.finally.句的情况下就会终.其run().法。 .如：JVM的垃圾回收线程就是Daemon线程，Finalizer也是守护线程。 31、java如何实现多线程之间的通讯和协作？ 中断和共享变量 32、什么是可重.锁（ReentrantLock）？ 举例来说明锁的可重.性 publicclassUnReentrant{Locklock=newLock(); publicvoidouter(){lock.lock();inner();lock.unlock();}publicvoidinner() {lock.lock();//dosomething lock.unlock();}} outer中调.了inner，outer先锁住了lock，这样inner就不能再获取lock。其实调.outer的线程已经获取了lock锁，但是不能 在inner中重复利.已经获取的锁资源，这种锁即称之为不可重.可重.就意味着：线程可以进.任何.个它已经拥有的锁所同 步着的代码块。 synchronized、ReentrantLock都是可重.的锁，可重.锁相对来说简化了并发编程的开发。 33、当.个线程进.某个对象的.个synchronized的实例.法后，其它线程是否可进.此对象的其它.法？ 如果其他.法没有synchronized的话，其他线程是可以进.的。 所以要开放.个线程安全的对象时，得保证每个.法都是线程安全的。 34、乐观锁和悲观锁的理解及如何实现，有哪些实现.式？ 悲观锁：总是假设最坏的情况，每次去拿数据的时候都认为别.会修改，所以每次在拿数据的时候都会上锁，这样别.想拿这个 数据就会阻塞直到它拿到锁。传统的关系型数据库.边就.到了很多这种锁机制，.如.锁，表锁等，读锁，写锁等，都是在做 操作之前先上锁。再.如Java..的同步原语synchronized关键字的实现也是悲观锁。 乐观锁：顾名思义，就是很乐观，每次去拿数据的时候都认为别.不会修改，所以不会上锁，但是在更新的时候会判断.下在此 期间别.有没有去更新这个数据，可以使.版本号等机制。乐观锁适.于多读的应.类型，这样可以提.吞吐量，像数据库提供 的类似于write_condition机制，其实都是提供的乐观锁。在Java中java.util.concurrent.atomic包下.的原.变量类就是使. 了乐观锁的.种实现.式CAS实现的。 乐观锁的实现.式：

使.版本标识来确定读到的数据与提交时的数据是否.致。提交后修改版本标识，不.致时可以采取丢弃和再次尝试的策略。 java中的Compareand Swap即CAS，当多个线程尝试使.CAS同时更新同.个变量时，只有其中.个线程能更新变量的值，.其它线程都失败，失败的线程并不会被挂起，.是被告知这次竞争中失败，并可以再次尝试。CAS操作中包含三个操作数——需要读写的内存位置（V）、进..较的预期原值（A）和拟写.的新值(B)。如果内存位置V的值与预期原值A相匹配，那么处理器会.动将该位置值更新为新值B。否则处理器不做任何操作。 CAS缺点：

ABA问题： .如说.个线程one从内存位置V中取出A，这时候另.个线程two也从内存中取出A，并且two进.了.些操作变成了B，然后two.将V位置的数据变成A，这时候线程one进.CAS操作发现内存中仍然是A，然后one操作成功。尽管线程one的CAS操作成功，但可能存在潜藏的问题。从Java1.5开始JDK的atomic包.提供了.个类AtomicStampedReference来解决ABA问题。

循环时间.开销.： 对于资源竞争严重（线程冲突严重）的情况，CAS.旋的概率会.较.，从.浪费更多的CPU资源，效率低于 synchronized。

只能保证.个共享变量的原.操作：

当对.个共享变量执.操作时，我们可以使.循环CAS的.式来保证原.操作，但是对多个共享变量操作时，循环CAS就.法保证操作的原.性，这个时候就可以.锁。 35、SynchronizedMap和ConcurrentHashMap有什么区别？ SynchronizedMap.次锁住整张表来保证线程安全，所以每次只能有.个线程来访为map。 ConcurrentHashMap使.分段锁来保证在多线程下的性能。ConcurrentHashMap中则是.次锁住.个桶。ConcurrentHashMap默认将hash表分为16个桶，诸如get,put,remove等常.操作只锁当前需要.到的桶。这样，原来只能.个线程进.，现在却能同时有16个写线程执.，并发性能的提升是显.易.的。 另外ConcurrentHashMap使.了.种不同的迭代.式。在这种迭代.式中，当iterator被创建后集合再发.改变就不再是抛出 ConcurrentModi.cationException，取.代之的是在改变时new新的数据从.不影响原有的数据，iterator完成后再将头指针替换为新的数据，这样iterator线程可以使.原来.的数据，.写线程也可以并发的完成改变。 36、CopyOnWriteArrayList可以.于什么应.场景？ CopyOnWriteArrayList(免锁容器)的好处之.是当多个迭代器同时遍历和修改这个列表时，不会抛出 ConcurrentModificationException。在CopyOnWriteArrayList中，写.将导致创建整个底层数组的副本，.源数组将保留在原地，使得复制的数组在被修改时，读取操作可以安全地执..。 I.由于写操作的时候，需要拷.数组，会消耗内存，如果原数组的内容.较多的情况下，可能导致younggc或者fullgc； II.不能.于实时读的场景，像拷.数组、新增元素都需要时间，所以调..个set操作后，读取到数据可能还是旧的,虽然 CopyOnWriteArrayList能做到最终.致性,但是还是没法满.实时性要求；

CopyOnWriteArrayList透露的思想读写分离，读和写分开最终.致性使.另外开辟空间的思路，来解决并发冲突

37、什么叫线程安全？servlet是线程安全吗? 线程安全是编程中的术语，指某个函数、函数库在多线程环境中被调.时，能够正确地处理多个线程之间的共享变量，使程序功能正确完成。 Servlet不是线程安全的，servlet是单实例多线程的，当多个线程同时访问同.个.法，是不能保证共享变量的线程安全性的。Struts2的action是多实例多线程的，是线程安全的，每个请求过来都会new.个新的action分配给这个请求，请求完成后销毁。 SpringMVC的Controller是线程安全的吗？不是的，和Servlet类似的处理流程Struts2好处是不.考虑线程安全问题；Servlet和SpringMVC需要考虑线程安全问题，但是性能可以提升不.处理太多的gc，可以使.ThreadLocal来处理多线程的问题。 38、volatile有什么.？能否..句话说明下volatile的应.场景？ volatile保证内存可.性和禁.指令重排。 volatile.于多线程环境下的单次操作(单次读或者单次写)。 39、为什么代码会重排序？ 在执.程序时，为了提供性能，处理器和编译器常常会对指令进.重排序，但是不能随意重排序，不是你想怎么排序就怎么排序，它需要满.以下两个条件：在单线程环境下不能改变程序运.的结果；存在数据依赖关系的不允许重排序需要注意的是：重排序不会影响单线程环境的执.结果，但是会破坏多线程的执.语义。

40、在java中wait和sleep.法的不同？ 最.的不同是在等待时wait会释放锁，.sleep.直持有锁。Wait通常被.于线程间交互，sleep通常被.于暂停执..。直接了解的深..点吧：在Java中线程的状态.共被分成6种：

初始态：NEW 创建.个Thread对象，但还未调.start()启动线程时，线程处于初始态。 运.态：RUNNABLE 在Java中，运.态包括就绪态和运.态。就绪态该状态下的线程已经获得执.所需的所有资源，只要CPU分配执.权就能运..。所有就绪态的线程存放在就绪队列中。运.态获得CPU执.权，正在执.的线程。由于.个CPU同.时刻只能执..条线程，因此每个CPU每个时刻只有.条运.态的 线程。 阻塞态 当.条正在执.的线程请求某.资源失败时，就会进.阻塞态。.在Java中，阻塞态专指请求锁失败时进.的状态。由.个阻塞队列存放所有阻塞态的线程。处于阻塞态的线程会不断请求资源，.旦请求成功，就会进.就绪队列，等待执..。PS：锁、IO、Socket等都资源。 等待态 当前线程中调.wait、join、park函数时，当前线程就会进.等待态。也有.个等待队列存放所有等待态的线程。线程处于等待态表.它需要等待其他线程的指.才能继续运..。进.等待态的线程会释放CPU执.权，并释放资源（如：锁） 超时等待态 当运.中的线程调.sleep(time)、wait、join、

parkNanos、parkUntil时，就会进.该状态；它和等待态.样，并不是因为请求不到资源，.是主动进.，并且进.后需要其他线程唤醒；进.该状态后释放CPU执.权和占有的资源。与等待态的区别：到了超时时间后.动进.阻塞队列，开始竞争锁。终.态 线程执.结束后的状态。 注意：

wait().法会释放CPU执.权和占有的锁。 sleep(long).法仅释放CPU使.权，锁仍然占.；线程被放.超时等待队列，与yield相.，它会使线程较.时间得不到运.。 yield().法仅释放CPU执.权，锁仍然占.，线程会被放.就绪队列，会在短时间内再次执.。 wait和notify必须配套使.，即必须使.同.把锁调.； wait和notify必须放在.个同步块中调.wait和notify的对象必须是他们所处同步块的锁对象。 41、.个线程运.时发.异常会怎样？ 如果异常没有被捕获该线程将会停.执.。Thread.UncaughtExceptionHandler是.于处理未捕获异常造成线程突然中断情况的.个内嵌接.。当.个未捕获异常将造成线程中断的时候JVM会使.Thread.getUncaughtExceptionHandler()来查询线程的 UncaughtExceptionHandler并将线程和异常作为参数传递给handler的uncaughtException().法进.处理。 42、如何在两个线程间共享数据？ 在两个线程间共享变量即可实现共享。 .般来说，共享变量要求变量本.是线程安全的，然后在线程内使.的时候，如果有对共享变量的复合操作，那么也得保证复合操作的线程安全性。 43、Java中notify和notifyAll有什么区别？ notify().法不能唤醒某个具体的线程，所以只有.个线程在等待的时候它才有.武之地。.notifyAll()唤醒所有线程并允许他们争夺锁确保了.少有.个线程能继续运.。 44、为什么wait,notify和notifyAll这些.法不在thread类..？ .个很明显的原因是JAVA提供的锁是对象级的.不是线程级的，每个对象都有锁，通过线程获得。由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象。 45、什么是ThreadLocal变量？ ThreadLocal是Java..种特殊的变量。每个线程都有.个ThreadLocal就是每个线程都拥有了..独.的.个变量，竞争条件被彻底消除了。它是为创建代价.昂的对象获取线程安全的好.法，.如你可以.ThreadLocal让SimpleDateFormat变成线程安全的，因为那个类创建代价.昂且每次调.都需要创建不同的实例所以不值得在局部范围使.它，如果为每个线程提供.个..独有的变量拷.，将..提.效率。.先，通过复.减少了代价.昂的对象的创建个数。其次，你在没有使..代价的同步或者不变性的情况下获得了线程安全。 46、Java中interrupted和isInterrupted.法的区别？ interrupt interrupt.法.于中断线程。调.该.法的线程的状态为将被置为"中断"状态。注意：线程中断仅仅是置线程的中断状态位，不会停.线程。需要....去监视线程的状态为并做处理。.持线程中断的.法（也就是线程中断后会抛出interruptedException的.法）就是在监视线程的中断状态，.旦线程的中断状态被置为"中断状态"，就会抛出中断异常。 interrupted 查询当前线程的中断状态，并且清除原状态。如果.个线程被中断了，第.次调.interrupted则返回true，第.次和后.的就返回false了。 isInterrupted 仅仅是查询当前线程的中断状态 47、为什么wait和notify.法要在同步块中调.？ Java API强制要求这样做，如果你不这么做，你的代码会抛出IllegalMonitorStateException异常。还有.个原因是为了避免 wait和notify之间产.竞态条件。 48、为什么你应该在循环中检查等待条件? 处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就会在没有满.结束条件的情况下退出。 49、Java中的同步集合与并发集合有什么区别？ 同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更.。在Java1.5之前程序员们只有同步集合来.且在多线程并发的时候会导致争.，阻碍了系统的扩展性。Java5介绍了并发集合像ConcurrentHashMap，不仅提供线程安全还.锁分离和内部分区等现代技术提.了可扩展性。 50、什么是线程池？为什么要使.它？ 创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变.，.且.个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若.线程来响应处理，它们被称为线程池，..的线程叫.作线程。从JDK1.5开始，JavaAPI提供了Executor框架让你可以创建不同的线程池。 51、怎么检测.个线程是否拥有锁？ 在java.lang.Thread中有.个.法叫holdsLock()，它返回true如果当且仅当当前线程拥有某个具体对象的锁。 52、你如何在Java中获取线程堆栈？

kill-3[javapid] 不会在当前终端输出，它会输出到代码执.的或指定的地.去。.如，kill-3tomcatpid,输出堆栈到log.录下。Jstack[javapid] 这个.较简单，在当前终端显.，也可以重定向到指定.件中。-JvisualVM：ThreadDump 不做说明，打开JvisualVM后，都是界.操作，过程还是很简单的。 53、JVM中哪个参数是.来控制线程的栈堆栈.的? -Xss每个线程的栈.. 54、Thread类中的yield.法有什么作.？ 使当前线程从执.状态（运.状态）变为可执.态（就绪状态）。 当前线程到了就绪状态，那么接下来哪个线程会从就绪状态变成执.状态呢？可能是当前线程，也可能是其他线程，看系统的分配了。 55、Java中ConcurrentHashMap的并发度是什么？ ConcurrentHashMap把实际map划分成若.部分来实现它的可扩展性和线程安全。这种划分是使.并发度获得的，它是ConcurrentHashMap类构造函数的.个可选参数，默认值为16，这样在多线程情况下就能避免争.。 在JDK8后，它摒弃了Segment（锁段）的概念，.是启.了.种全新的.式实现,利.CAS算法。同时加.了更多的辅助变量来提.并发度，具体内容还是查看源码吧。 56、Java中Semaphore是什么？ Java中的Semaphore是.种新的同步类，它是.个计数信号。从概念上讲，从概念上讲，信号量维护了.个许可集合。如有必要，在许可可.前会阻塞每.个 acquire()，然后再获取该许可。每个 release()添加.个许可，从.可能释放.个正在阻塞的获取者。但是，不使.实际的许可对象，Semaphore只对可.许可的号码进.计数，并采取相应的.动。信号量常常.于多线程的代码中，.如数据库连接池。 57、Java线程池中submit()和execute().法有什么区别？ 两个.法都可以向线程池提交任务，execute().法的返回类型是void，它定义在Executor接.中。 .submit().法可以返回持有计算结果的Future对象，它定义在ExecutorService接.中，它扩展了Executor接.，其它线程池类像ThreadPoolExecutor和ScheduledThreadPoolExecutor都有这些.法。

58、什么是阻塞式.法？ 阻塞式.法是指程序会.直等待该.法完成期间不做其他事情，ServerSocket的accept().法就是.直等待客.端连接。这.的阻塞是指调.结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和.阻塞式.法在任务完成前就返回。 59、Java中的ReadWriteLock是什么？ 读写锁是.来提升并发程序性能的锁分离技术的成果。 60、volatile变量和atomic变量有什么不同？ Volatile变量可以确保先.关系，即写操作会发.在后续的读操作之前,但它并不能保证原.性。例如.volatile修饰count变量那么count++操作就不是原.性的。 .AtomicInteger类提供的atomic.法可以让这种操作具有原.性如getAndIncrement().法会原.性的进.增量操作把当前值加.，其它数据类型和引.变量也可以进.相似操作。 61、可以直接调.Thread类的run().法么？ 当然可以。但是如果我们调.了Thread的run().法，它的.为就会和普通的.法.样，会在当前线程中执.。为了在新的线程中执.我们的代码，必须使.Thread.start().法。 62、如何让正在运.的线程暂停.段时间？ 我们可以使.Thread类的Sleep().法让线程暂停.段时间。需要注意的是，这并不会让线程终.，.旦从休眠中唤醒线程，线程的状态将会被改变为Runnable，并且根据线程调度，它将得到执.。 63、你对线程优先级的理解是什么？ 每.个线程都是有优先级的，.般来说，.优先级的线程在运.时会具有优先权，但这依赖于线程调度的实现，这个实现是和操作系统相关的(OSdependent)。我们可以定义线程的优先级，但是这并不能保证.优先级的线程会在低优先级的线程前执.。线程优先级是.个int变量(从1-10)，1代表最低优先级，10代表最.优先级。 java的线程优先级调度会委托给操作系统去处理，所以与具体的操作系统优先级有关，如.特别需要，.般.需设置线程优先级。 64、什么是线程调度器(ThreadScheduler)和时间分.(TimeSlicing)？ 线程调度器是.个操作系统服务，它负责为Runnable状态的线程分配CPU时间。.旦我们创建.个线程并启动它，它的执.便依赖于线程调度器的实现。同上.个问题，线程调度并不受到Java虚拟机控制，所以由应.程序来控制它是更好的选择（也就是说不要让你的程序依赖于线程的优先级）。 时间分.是指将可.的CPU时间分配给可.的Runnable线程的过程。分配CPU时间可以基于线程优先级或者线程等待的时间。 65、你如何确保main().法所在的线程是Java程序最后结束的线程？ 我们可以使.Thread类的join().法来确保所有程序创建的线程在main().法退出前结束。 66、线程之间是如何通信的？ 当线程间是可以共享资源时，线程间通信是协调它们的重要的.段。Object类中wait()otify()otifyAll().法可以.于线程间通信关于资源的锁的状态。 67、为什么线程通信的.法wait(),notify()和notifyAll()被定义在Object类.？ Java的每个对象中都有.个锁(monitor，也可以成为监视器)并且wait()，notify()等.法.于等待对象的锁或者通知其他线程对象的监视器可.。在Java的线程中并没有可供任何对象使.的锁和同步器。这就是为什么这些.法是Object类的.部分，这样Java的每.个类都有.于线程间通信的基本.法。 68、为什么wait(),notify()和notifyAll()必须在同步.法或者同步块中被调.？ 当.个线程需要调.对象的wait().法的时候，这个线程必须拥有该对象的锁，接着它就会释放这个对象锁并进.等待状态直到其他线程调.这个对象上的notify().法。同样的，当.个线程需要调.对象的notify().法时，它会释放这个对象的锁，以便其他在等待的线程就可以得到这个对象锁。由于所有的这些.法都需要线程持有对象的锁，这样就只能通过同步来实现，所以他们只能在同步.法或者同步块中被调.。 69、为什么Thread类的sleep()和yield().法是静态的？ Thread类的sleep()和yield().法将在当前正在执.的线程上运.。所以在其他处于等待状态的线程上调.这些.法是没有意义的。这就是为什么这些.法是静态的。它们可以在当前正在执.的线程中.作，并避免程序员错误的认为可以在其他.运.线程调.这些.法。 70、如何确保线程安全？ 在Java中可以有很多.法来保证线程安全.同步，使.原.类(atomic concurrent classes)，实现并发锁，使.volatile关键字，使.不变类和线程安全类。 71、同步.法和同步块，哪个是更好的选择？ 同步块是更好的选择，因为它不会锁住整个对象（当然你也可以让它锁住整个对象）。同步.法会锁住整个对象，哪怕这个类中有多个不相关联的同步块，这通常会导致他们停.执.并需要等待获得这个对象上的锁。 同步块更要符合开放调.的原则，只在需要锁住的代码块锁住相应的对象，这样从侧.来说也可以避免死锁。 72、如何创建守护线程？ 使.Thread类的setDaemon(true).法可以将线程设置为守护线程，需要注意的是，需要在调.start().法前调.这个.法，否则会抛出IllegalThreadStateException异常。 73、什么是JavaTimer类？如何创建.个有特定时间间隔的任务？ java.util.Timer是.个.具类，可以.于安排.个线程在未来的某个特定时间执.。Timer类可以.安排.次性任务或者周期任务。 java.util.TimerTask是.个实现了Runnable接.的抽象类，我们需要去继承这个类来创建我们..的定时任务并使.Timer去安排它的执.。 .前有开源的Qurtz可以.来创建定时任务。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.为什么你学不会递归？

2.

MySQL：Left Join避坑指南 3.AJAX请求真的不安全么？

4.

.个..不主动联系你还有机会吗？

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Java最常.的208道.试题 王磊的博客Java后端 2019-09-18 来源 |王磊的博客博客 |www.cnblogs.com/vipstone 这份.试清单是我从 2015年做 TeamLeader之后开始收集的，...是给公司招聘.，另...是想.它来挖掘我在 Java技术栈中的技术盲点，然后修复和完善它，以此来提...的技术.平。虽然我从 2009年就开始参加编程.作了，但依旧觉得还有很多东西要学，当然学习的过程也给我带来了很多成就感，这些成就感也推动我学习更多的技术知识。 聊回.试题这件事，这份.试清单原本是我们公司内部使.的，可到后来有很多朋友在微信上联系到我，让我帮他们找.些.试..的资料，.且这些关系也不太好拒绝，.呢，是因为这些找我，要.试题的.，不是我的好朋友的弟弟妹妹，就是我的弟弟妹妹们；.呢，我也不能....的对付，受.之事忠.之命，我也不能辜负这份信任。慢慢的我产.了.个想法，要不要把我整理的这 200多道.试题分享出来，来帮助更多需要的.。微信搜索 web_resource关注后获取每..道.试题推送。 说实话刚开始的时候还是.较犹豫的，.先我会觉得这么做会不会有点帮.“作弊”的嫌疑，最后我想通了，这是.件值得去做的事.。

第.：让更多的.因此.学到了更多的知识，这是.件.好事。

第.：这只是经验的.度提炼，让那些原本就掌握了技术却不知道怎么表达的.，学会如何在.试中展...。

第三：如果只是死记硬背这些.试题，只要.试官再深.问纠.下，也可对这个.有.个准确的认识，之前说的“帮.作弊”的事就存在了。

第四：学习有很多种.式，但只有好学者才会临池学书。如果是不想学的.，提供再多再好的资料放在他们的.前，他们也会视.不.。

就像之前听过的.个故事，为什么在美国有些企业只要看你是哈佛的学历就直接录取？并不是哈佛有多么厉害，当然教学质量也是其中原因之.，但更多的是在美国上.学还是挺贵的，.先你能上的起哈佛，说明你的家庭条还不错，从.应该就有很多参加更好教育的机会；第.，你能进.哈佛，也说明你脑.不笨，能考的上哈佛；最后才是哈佛确实能给你提供.个，相对不错的教育环境。综合以上特质，所以这些企业才敢直接聘请那些有哈佛学历的.。微信搜索 web_resource关注后获取每..道.试题推送。 对应到我们这份.试题其实也.样，.先你如果能记住其中.部分的答案说明你，第.，你很聪明并且记性还很好；第.，说明你有上进.，也愿意学习；第三，有了这份.试题做理论.撑之后，即使你的实践经验没有那么多，但懂得原理的你，做出来的程序也.定不会太差。 所以如果您是.试官，恰好.看到这.，如果条件允许的话，请多给这样愿意学.很聪明的年轻.多.些机会。 .试题模块介绍 说了这么多，下.进.我们本.的主题，我们这份.试题，包含的内容了.九了模块：Java基础、容器、多线程、反射、对象拷.、Java Web模块、异常、.络、设计模式、Spring/Spring MVC、Spring Boot/SpringCloud、Hibernate、Mybatis、RabbitMQ、 Kafka、Zookeeper、MySql、Redis、JVM。如下图所.：可能对于初学者不需要看后.的框架和 JVM模块的知识，读者朋友们可根据..的情况，选择对应的模块进.阅读。微信搜索web_resource关注后获取每..道.试题推送。

适宜阅读.群

需要.试的初/中/.级 java程序员想要查漏补缺的.想要不断完善和扩充.. java技术栈的.java .试官

具体.试题 下..起来看 208道.试题，具体的内容。 .、Java基础 1.JDK和 JRE有什么区别？

2.==和 equals的区别是什么？

3.两个对象的 hashCode()相同，则 equals()也.定为 true，对吗？

4..nal 在 java中有什么作.？ 5.java中的 Math.round(-1.5)等于多少？ 6.String属于基础的数据类型吗？ 7.java中操作字符串都有哪些类？它们之间有什么区别？

8.String str="i"与 String str=new String("i").样吗？9.如何将字符串反转？

10.

String类的常..法都有那些？

11.

抽象类必须要有抽象.法吗？12.普通类和抽象类有哪些区别？

13.抽象类能使. .nal修饰吗?

14.

接.和抽象类有什么区别?

15.java中 IO流分为.种?

16.BIO、N IO、AIO有什么区别?

17.

Files的常..法都有哪些?

.、容器 18.java容器都有哪些? 19. Collection和 Collections有什么区别?

20.

List、Set、Map之间的区别是什么?

21.

HashMap和 Hashtable有什么区别?

22.如何决定使. HashMap还是 TreeMap?

23.说.下 HashMap的实现原理?

24.说.下 HashSet的实现原理?

.ArrayList和 LinkedList的区别是什么?

26.

如何实现数组和 List之间的转换?

27.

ArrayList和 Vector的区别是什么?

28.Array和 ArrayList有何区别?

29.在 Queue中 poll()和 remove()有什么区别? .哪些集合类是线程安全的?

31.迭代器 Iterator是什么?

32.

Iterator怎么使.? 有什么特点?

33.

Iterator和 ListIterator有什么区别?

34.

怎么确保.个集合不能被修改?

三、多线程 .并.和并发有什么区别? 36.线程和进程的区别? 37.守护线程是什么? 38. 创建线程有哪.种.式?

39.说.下 runnable和 callable有什么区别? .线程有哪些状态?

41.

sleep()和 wait()有什么区别?

42.

notify()和 notifyAll()有什么区别?

43.线程的 run()和 start()有什么区别?

44.

创建线程池有哪.种.式?.线程池都有哪些状态?

46.线程池中 submit()和 execute().法有什么区别?

47.在 java程序中怎么保证多线程的运.安全?48.多线程锁的升级原理是什么?49.什么是死锁?

.怎么防.死锁?

51.

ThreadLocal是什么?有哪些使.场景?

52.说.下 synchronized底层实现原理?

53.

synchronized和 volatile的区别是什么?

54.

synchronized和 Lock有什么区别?

.synchronized和 ReentrantLock区别是什么?

56.说.下 atomic的原理?

四、反射 57.什么是反射? 58.什么是 java序列化?什么情况下需要序列化?

59.

动态代理是什么?有哪些应.?.怎么实现动态代理?

五、对象拷. 61.为什么要使.克隆?62.如何实现对象克隆?63.深拷.和浅拷.区别是什么? 六、JavaWeb 64.jsp和 servlet 有什么区别? .jsp有哪些内置对象?作.分别是什么?

66.说.下 jsp的 4种作.域?

67.

session和 cookie有什么区别?

68.说.下 session的.作原理?

69.

如果客.端禁. cookie能实现 session还能.吗? .spring mvc和 struts的区别是什么?

71.如何避免 sql注.?

72.什么是 XSS攻击，如何避免?

73.什么是 CSRF攻击，如何避免?

七、异常 74.throw和 throws的区别?

..nal、.nally、.nalize有什么区别? 76.try-catch-.nally中哪个部分可以省略?

77.

try-catch-.nally中，如果 catch中 return了，.nally还会执.吗？78.常.的异常类有哪些?

.、.络 79.http响应码 301和 302代表的是什么？有什么区别? .forward和 redirect的区别? 81.简述 tcp和 udp的区别?

82.

tcp为什么要三次握.，两次不.吗？为什么?

83.说.下 tcp粘包是怎么产.的? 84.OSI的七层模型都有哪些?

.get和 post请求有哪些区别？86.如何实现跨域?

87.说.下 JSONP实现原理?

九、设计模式 88. 说.下你熟悉的设计模式?

89.

简单..和抽象..有什么区别?

.、Spring/Spring MVC .为什么要使. spring? 91. 解释.下什么是 aop?

92.

解释.下什么是 ioc? 93.spring有哪些主要模块?

94.

spring常.的注..式有哪些? .spring中的 bean是线程安全的吗?

96.spring .持.种 bean的作.域?

97.spring .动装配 bean有哪些.式?

98.

spring事务实现.式有哪些?

99.说.下 spring的事务隔离?

.说.下 spring mvc运.流程? 101.spring mvc有哪些组件? 102.@RequestMapping的作.是什么? 103.@Autowired的作.是什么?

..、SpringBoot/SpringCloud 104.什么是 spring boot? .为什么要. spring boot? 106.spring boot核.配置.件是什么? 107.spring boot配置.件有哪.种类型？它们有什么区别? 108.spring boot有哪些.式可以实现热部署? 109.jpa和 hibernate有什么区别? .什么是 spring cloud? 111.spring cloud断路器的作.是什么? 112.spring cloud的核.组件有哪些？..、Hibernate 113.为什么要使. hibernate? 114.什么是 ORM框架? .hibernate中如何在控制台查看打印的 sql语句? 116.hibernate有.种查询.式? 117.hibernate实体类可以被定义为 .nal吗?

118.在 hibernate中使. Integer和 int做映射有什么区别? 119.hibernate是如何.作的? .get()和 load()的区别?

121.说.下 hibernate的缓存机制? 122.hibernate对象有哪些状态? 123.在 hibernate中 getCurrentSession和 openSession的区别是什么? 124.hibernate实体类必须要有.参构造函数吗？为什么?

.三、Mybatis .mybatis中 #{}和 ${}的区别是什么？ 126.mybatis有.种分..式？127.RowBounds是.次性查询全部结果吗？为什么？ 128.mybatis逻辑分.和物理分.的区别是什么？ 129.mybatis是否.持延迟加载？延迟加载的原理是什么？ .说.下mybatis的.级缓存和.级缓存？ 131.mybatis和 hibernate的区别有哪些？ 132.mybatis有哪些执.器（Executor）？ 133.mybatis分.插件的实现原理是什么？ 134.mybatis如何编写.个.定义插件？.四、RabbitMQ .rabbitmq的使.场景有哪些？ 136.rabbitmq有哪些重要的..？ 137.rabbitmq有哪些重要的组件？ 138.rabbitmq中 vhost的作.是什么？ 139.rabbitmq的消息是怎么发送的？ .rabbitmq怎么保证消息的稳定性？ 141.rabbitmq怎么避免消息丢失？142.要保证消息持久化成功的条件有哪些？ 143.rabbitmq持久化有什么缺点？ 144.rabbitmq有.种.播类型？ .rabbitmq怎么实现延迟消息队列？ 146.rabbitmq集群有什么.？ 147.rabbitmq节点的类型有哪些？ 148.rabbitmq集群搭建需要注意哪些问题？ 149.rabbitmq每个节点是其他节点的完整拷.吗？为什么？ .rabbitmq集群中唯..个磁盘节点崩溃了会发.什么情况？ 151.rabbitmq对集群节点停.顺序有要求吗？.五、Kafka 152.kafka可以脱离 zookeeper单独使.吗？为什么？ 153.kafka有.种数据保留的策略？ 154.kafka同时设置了 7天和 10G清除数据，到第五天的时候消息达到了 10G，这个时候 kafka将如何处理？ .什么情况会导致 kafka运.变慢？ 156.使. kafka集群需要注意什么？ .六、Zookeeper 157.zookeeper是什么？158.zookeeper都有哪些功能？159.zookeeper有.种部署模式？ .zookeeper怎么保证主从节点的状态同步？161.集群中为什么要有主节点？ 162.集群中有 3台服务器，其中.个节点宕机，这个时候 zookeeper还可以使.吗？ 163.说.下zookeeper的通知机制？

.七、MySql 164.数据库的三范式是什么？ ..张.增表..总共有 7条数据，删除了最后 2条数据，重启 mysql数据库，.插.了.条数据，此时 id是.？166.如何获取当前数据库版本？ 167.说.下 ACID是什么？168.char和 varchar的区别是什么？ 169..oat和 double的区别是什么？ .mysql的内连接、左连接、右连接有什么区别？ 171.mysql索引是怎么实现的？

172.怎么验证 mysql的索引是否满.需求？ 173.说.下数据库的事务隔离？ 174.说.下 mysql常.的引擎？.说.下 mysql的.锁和表锁？ 176.说.下乐观锁和悲观锁？ 177.mysql问题排查都有哪些.段？ 178.如何做 mysql的性能优化？ 179.微信搜索web_resource关注后获取每..道.试题推送。

..、Redis 179.redis是什么？都有哪些使.场景？ .redis有哪些功能？ 181.redis和 memecache有什么区别？ 182.redis为什么是单线程的？ 183.什么是缓存穿透？怎么解决？ 184.redis .持的数据类型有哪些？ .redis .持的 java客.端都有哪些？ 186.jedis和 redisson有哪些区别？ 187.怎么保证缓存和数据库数据的.致性？ 188.redis持久化有.种.式？ 189.redis怎么实现分布式锁？ .redis分布式锁有什么缺陷？ 191.redis如何做内存优化？ 192.redis淘汰策略有哪些？ 193.redis常.的性能问题有哪些？该如何解决？ .九、JVM 194.说.下 jvm的主要组成部分？及其作.？ 195.说.下 jvm运.时数据区？ 196.说.下堆栈的区别？197.队列和栈是什么？有什么区别？198.什么是双亲委派模型？ 199.说.下类加载的执.过程？ 200.怎么判断对象是否可以被回收？ 201.java中都有哪些引.类型？ 202.说.下 jvm有哪些垃圾回收算法？ 203.说.下 jvm有哪些垃圾回收器？ 204.详细介绍.下 CMS垃圾回收器？ 205.新.代垃圾回收器和..代垃圾回收器都有哪些？有什么区别？ 206.简述分代垃圾回收器是怎么.作的？ 207.说.下 jvm调优的.具？ 208.常.的 jvm调优的参数都有哪些？

.试答案 由于.章篇幅问题，后续逐步更新答案。可以微信搜索「web_resource」关注后关注每.推送即可。此订阅号有「每..道.试题」栏.。会对.试题进.详细讲解，关注搜索关注。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进...告技术交流群。 推荐阅读 1. 把14亿中国.拉到.个微信群？

2. Spring中的18个注解，你会.个？
3.

寓教于乐，.玩游戏的.式学习Git

4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

Java后端：专注于Java技术

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Java线程池8.拒绝策略，.试必问！ Java后端 2019-10-12 点击上.Java后端，选择设为星标技术博.，及时送达 前. 谈到java的线程池最熟悉的莫过于ExecutorService接.了，jdk1.5新增的java.util.concurrent包下的这个api，..的简化了多线程代码的开发。.不论你.FixedThreadPool还是CachedThreadPool其背后实现都是ThreadPoolExecutor。

ThreadPoolExecutor是.个典型的缓存池化设计的产物，因为池.有..，当池.体积不够承载时，就涉及到拒绝策略。JDK中已经预设了4种线程池拒绝策略，下.结合场景详细聊聊这些策略的使.场景，以及我们还能扩展哪些拒绝策略。 池化设计思想 池话设计应该不是.个新名词。我们常.的如java线程池、jdbc连接池、redis连接池等就是这类设计的代表实现。这种设计会初始预设资源，解决的问题就是抵消每次获取资源的消耗，如创建线程的开销，获取远程连接的开销等。就好.你去.堂打饭，打饭的.妈会先把饭盛好.份放那.，你来了就直接拿着饭盒加菜即可，不.再临时.盛饭.打菜，效率就.了。 除了初始化资源，池化设计还包括如下这些特征：池.的初始值、池.的活跃值、池.的最.值等，这些特征可以直接映射到java线程池和数据库连接池的成员属性中。 线程池触发拒绝策略的时机 和数据源连接池不.样，线程池除了初始..和池.最.值，还多了.个阻塞队列来缓冲。 数据源连接池.般请求的连接数超过连接池的最.值的时候就会触发拒绝策略，策略.般是阻塞等待设置的时间或者直接抛异常。 .线程池的触发时机如下图：

如图，想要了解线程池什么时候触发拒绝粗略，需要明确上.三个参数的具体含义，是这三个参数总体协调的结果，.不是简单的 超过最.线程数就会触发线程拒绝粗略，当提交的任务数.于corePoolSize时，会优先放到队列缓冲区，只有填满了缓冲区后，才会判断当前运.的任务是否.于maxPoolSize，.于时会新建线程处理。.于时就触发了拒绝策略，总结就是：当前提交任务数.于（maxPoolSize+queueCapacity）时就会触发线程池的拒绝策略了。 JDK内置4种线程池拒绝策略 拒绝策略接.定义 在分析JDK.带的线程池拒绝策略前，先看下JDK定义的拒绝策略接.，如下： 1 public interface RejectedExecutionHandler {
2 void rejectedExecution(Runnable r, ThreadPoolExecutor executor)
3 ;
}

接.定义很明确，当触发拒绝策略时，线程池会调.你设置的具体的策略，将当前提交的任务以及线程池实例本.传递给你处理， 具体作何处理，不同场景会有不同的考虑，下.看JDK为我们内置了哪些实现： CallerRunsPolicy（调.者运.策略） 1 public static class CallerRunsPolicy implements RejectedExecutionHandler { 2 3 public CallerRunsPolicy() { 4 } 5 6 public void rejectedExecution(Runnable r, ThreadPoolExecutor e) 7 { 8 if (!e.isShutdown()) { 9 r.run(); 10 }

} } 功能：当触发拒绝策略时，只要线程池没有关闭，就由提交任务的当前线程处理。 使.场景：.般在不允许失败的、对性能要求不.、并发量较.的场景下使.，因为线程池.般情况下不会关闭，也就是提交的任 务.定会被运.，但是由于是调.者线程..执.的，当多次提交任务时，就会阻塞后续任务执.，性能和效率.然就慢了。 AbortPolicy（中.策略） 1 public static class AbortPolicy implements RejectedExecutionHandler { 2 3 public AbortPolicy() { 4 } 5 6 public void rejectedExecution(Runnable r, ThreadPoolExecutor e) 7 { throw new RejectedExecutionException("Task " + r.toString() +

8 9 + j p (" rejected from " g()
10 e.toString());
}
}

功能：当触发拒绝策略时，直接抛出拒绝执.的异常，中.策略的意思也就是打断当前执.流程 使.场景：这个就没有特殊的场景了，但是.点要正确处理抛出的异常。 ThreadPoolExecutor中默认的策略就是AbortPolicy，ExecutorService接.的系列ThreadPoolExecutor因为都没有显.的设置拒绝策略，所以默认的都是这个。但是请注意，ExecutorService中的线程池实例队列都是.界的，也就是说把内存撑爆了都不会触发拒绝策略。当...定义线程池实例时，使.这个策略.定要处理好触发策略时抛的异常，因为他会打断当前的执.流程。 DiscardPolicy（丢弃策略） 1 public static class DiscardPolicy implements RejectedExecutionHandler {
2
3 public DiscardPolicy() {
4 }
5
6 public void rejectedExecution(Runnable r, ThreadPoolExecutor e)
7 {
}
}

功能：直接静悄悄的丢弃这个任务，不触发任何动作 使.场景：如果你提交的任务.关紧要，你就可以使.它。因为它就是个空实现，会悄.声息的吞噬你的的任务。所以这个策略基 本上不.了 DiscardOldestPolicy（弃.策略） 1 public static

class DiscardOldestPolicy implements RejectedExecutionHandler { 2 3 public DiscardOldestPolicy() { 4 } 5 6 public void rejectedExecution(Runnable r, ThreadPoolExecutor e) 7 {
8 if (!e.isShutdown()) {
9 e.getQueue().poll();
10 e.execute(r);
11 }
}
}

功能：如果线程池未关闭，就弹出队列头部的元素，然后尝试执. 使.场景：这个策略还是会丢弃任务，丢弃时也是毫. 声息，但是特点是丢弃的是.的未执.的任务，.且是待执.优先级较.的 任务。基于这个特性，我能想到的场景就是，发布消息，和修改消息，当消息发布出去后，还未执.，此时更新的消息.来了，这 个时候未执.的消息的版本.现在提交的消息版本要低就可以被丢弃了。因为队列中还有可能存在消息版本更低的消息会排队执 .，所以在真正处理消息的时候.定要做好消息的版本.较。 第三.实现的拒绝策略 dubbo中的线程拒绝策略 1 public class AbortPolicyWithReport extends ThreadPoolExecutor.AbortPolicy { 2 3 protected static final Logger logger = LoggerFactory.getLogger(AbortPolicyWithReport.class); 4 5 private final String threadName; 6 7 private final URL url; 8 9 private static volatile long lastPrintTime = 0 10 ; 11 12 private static Semaphore guard = new Semaphore(1) 13 ; 14 15 public AbortPolicyWithReport(String threadName, URL url) 16 { 17 this.threadName = threadName; 18 this.url = url; 19 } 20 21 @Override 22 public void rejectedExecution(Runnable r, ThreadPoolExecutor e) 23 { 24 String msg = String.format("Thread pool is EXHAUSTED!" 25 + 26 " Thread Name: %s, Pool Size: %d (active: %d, core: %d, max: %d, largest: 27 + 28 " Executor status:(isShutdown:%s, isTerminated:%s, isTerminating:%s), in % 29 , 30 threadName, e.getPoolSize(), e.getActiveCount(), e.getCorePoolSize(), e.getMaximum 31 e.getTaskCount(), e.getCompletedTaskCount(), e.isShutdown(), e.isTerminated(), e.i 32 url.getProtocol(), url.getIp(), url.getPort()); 33 logger.warn(msg); 34 dumpJStack(); throw new RejectedExecutionException(msg); } private void dumpJStack() { //省略实现 }

可以看到，当dubbo的.作线程触发了线程拒绝后，主要做了三个事情，原则就是尽量让使.者清楚触发线程拒绝策略的真实原 因。 1）输出了.条警告级别的.志，.志内容为线程池的详细设置参数，以及线程池当前的状态，还有当前拒绝任务的.些详细信息。 可以说，这条.志，使.dubbo的有过.产运维经验的或多或少是.过的，这个.志简直就是.志打印的典范，其他的.志打印的 典范还有spring。得益于这么详细的.志，可以很容易定位到问题所在 2 ） 输出当前线程堆栈详情，这个太有.了，当你通过上.的.志信息还不能定位问题时，案发现场的dump线程上下.信息就是你

发现问题的救命稻草。

3 ）继续抛出拒绝执.异常，使本次任务失败，这个继承了JDK默认拒绝策略的特性

Netty中的线程池拒绝策略 1 private static final class NewThreadRunsPolicy implements RejectedExecutionHandler { 2 NewThreadRunsPolicy() { 3 super() 4 ; 5 } 6 7 public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) 8 { 9 try 10 { 11 final Thread t = new Thread(r, "Temporary task executor") 12 ; 13 t.start(); 14 } catch (Throwable e) { 15 throw new RejectedExecutionException("Failed to start a new thread", e); } } } Netty中的实现很像JDK中的CallerRunsPolicy，舍不得丢弃任务。不同的是，CallerRunsPolicy是直接在调.者线程执.的任 务。.Netty是新建了.个线程来处理的。 所以，Netty的实现相较于调.者执.策略的使..就可以扩展到.持.效率.性能的场景了。但是也要注意.点，Netty的实现 .，在创建线程时未做任何的判断约束，也就是说只要系统还有资源就会创建新的线程来处理，直到new不出新的线程了，才会抛 创建线程失败的异常 activeMq中的线程池拒绝策略

1 new RejectedExecutionHandler() { 2 @Override 3 public void rejectedExecution(final Runnable r, final ThreadPoolExecutor executor) 4 { 5 try 6 { 7 executor.getQueue().offer(r, 60, TimeUnit.SECONDS); 8 } catch (InterruptedException e) { 9 throw new RejectedExecutionException("Interrupted waiting for BrokerService.worker") 10 ; 11 } 12 throw new RejectedExecutionException("Timed Out while attempting to enqueue Task.") ; } }); activeMq中的策略属于最.努.执.任务型，当触发拒绝策略时，在尝试.分钟的时间重新将任务塞进任务队列，当.分钟超时还 没成功时，就抛出异常 pinpoint中的线程池拒绝策略 1 public class RejectedExecutionHandlerChain implements RejectedExecutionHandler { 2 private final RejectedExecutionHandler[] handlerChain; 3 4 public static RejectedExecutionHandler build(List chain) 5 { 6 Objects.requireNonNull(chain, "handlerChain must not be null") 7 ; 8 RejectedExecutionHandler[] handlerChain = chain.toArray(new RejectedExecutionHandler[0]) 9 ; 10 return new

RejectedExecutionHandlerChain(handlerChain); 11 } 12 13 private RejectedExecutionHandlerChain(RejectedExecutionHandler[] handlerChain) 14 { 15 this.handlerChain = Objects.requireNonNull(handlerChain, "handlerChain must not be null") 16 ; 17 } 18 19 @Override 20 public void rejectedExecution(Runnable r, ThreadPoolExecutor executor)

{ for (RejectedExecutionHandler rejectedExecutionHandler : handlerChain) { rejectedExecutionHandler.rejectedExecution(r, executor); } } } pinpoint的拒绝策略实现很有特点，和其他的实现都不同。他定义了.个拒绝策略链，包装了.个拒绝策略列表，当触发拒绝策略时，会将策略链中的rejectedExecution依次执..遍。 结语 前.从线程池设计思想，以及线程池触发拒绝策略的时机引出java线程池拒绝策略接.的定义。并辅以JDK内置4种以及四个第三.开源软件的拒绝策略定义描述了线程池拒绝策略实现的各种思路和使.场景。 希望阅读此.后能让你对java线程池拒绝策略有更加深刻的认识，能够根据不同的使.场景更加灵活的应.。 转.：KL博客地址：www.kailing.pub/article/index/arcid/255.html编辑：Java技术栈（id：javastack） 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ Java.试题：百度前200.都在这. 唐尤华Java后端 2019-10-14 点击上.Java后端，选择设为星标技术博.，及时送达 作者|唐尤华来源|github.com/tangyouhua 基本概念

操作系统中heap和stack的区别什么是基于注解的切.实现什么是对象/关系映射集成模块什么是Java的反射机制什么是ACIDBS与CS的联系与区别 Cookie和Session的区别 fail-fast与fail-safe机制有什么区别get和post请求的区别 Interface与abstract类的区别 IOC的优点是什么 IO和NIO的区别，NIO优点Java8/Java7为我们提供了什么新功能什么是竞态条件?举个例.说明。JRE、JDK、JVM及JIT之间有什么不同MVC的各个部分都有那些技术来实现?如何实现?RPC通信和RMI区别什么是WebService（Web服务）JSWDL开发包的介绍。JAXP、JAXM的解释。SOAP、UDDI,WSDL解释。WEB容器主要有哪些功能?并请列出.些常.的WEB容器名字。.个".java"源.件中是否可以包含多个类（不是内部类）？有什么限制简单说说你了解的类加载器。是否实现过类加载器解释.下什么叫AOP（.向切.编程）请简述Servlet的.命周期及其相关的.法请简述.下Ajax的原理及实现步骤简单描述Struts的主要功能什么是N层架构什么是CORBA？.途是什么什么是Java虚拟机？为什么Java被称作是"平台.关的编程语."什么是正则表达式？.途是什么？哪个包使.正则表达式来实现模式匹配什么是懒加载（LazyLoading）什么是尾递归，为什么需要尾递归什么是控制反转（InversionofControl）与依赖注.（DependencyInjection） 关键字 finalize

什么是finalize().法

finalize().法什么时候被调.析构函数(finalization)的.的是什么 final和finalize的区别 final

final关键字有哪些.法 final与static关键字可以.于哪.？它们的作.是什么 final,finally,finalize的区别 final、finalize和finally的不同之处？能否在运.时向staticfinal类型的赋值

使.final关键字修饰.个变量时，是引.不能变，还是引.的对象不能变.个类被声明为final类型，表.了什么意思 throws,throw,try,catch,finally分别代表什么意义 Java有.种修饰符？分别.来修饰什么 volatile

volatile修饰符的有过什么实践 volatile变量是什么？volatile变量和atomic变量有什么不同 volatile类型变量提供什么保证？能使得.个.原.操作变成原.操作吗能创建volatile数组吗？ transient变量有什么特点super什么时候使.publicstaticvoid写成staticpublicvoid会怎样说明.下publicstaticvoidmain(Stringargs[])这段声明.每个关键字的作.请说出作.域public,private,protected,以及不写时的区别 sizeof是Java的关键字吗 static

staticclass与nonstaticclass的区别 static关键字是什么意思？Java中是否可以覆盖(override).个private或者是static的.法静态类型有什么特点 main().法为什么必须是静态的？能不能声明main().法为.静态是否可以从.个静态（static）.法内部发出对.静态（non-static）.法的调.静态变量在什么时候加载？编译期还是运.期？静态代码块加载的时机呢成员.法是否可以访问静态变量？为什么静态.法不能访问成员变量 switch

switch语句中的表达式可以是什么类型数据 switch是否能作.在byte上，是否能作.在long上，是否能作.在String上 while循环和do循环有什么不同 操作符

&操作符和&&操作符有什么区别?

a=a+b与a+=b的区别？逻辑操作符(&,|,^)与条件操作符(&&,||)的区别 3*0.1==0.3将会返回什么？true还是false？floatf=3.4;是否正确？ shorts1=1;s1=s1+1;有什么错? 数据结构 基础类型(Primitives)

基础类型(Primitives)与封装类型(Wrappers)的区别在哪.简述九种基本数据类型的..，以及他们的封装类 int和Integer哪个会占.更多的内存？int和Integer有什么区别？parseInt()函数在什么时候使.到 float和double的默认值是多少如何去.数四舍五.保留.数点后两位char型变量中能不能存贮.个中.汉字，为什么 类型转换

怎样将bytes转换为long类型怎么将byte转换为String如何将数值型字符转换为数字我们能将int强制转换为byte类型的变量吗？如果该值.于byte类型的范围，将会出现什么现象能在不进.强制转换的情况下将.个double值赋值给long类型的变量吗类型向下转换是什么 数组

如何权衡是使..序的数组还是有序的数组怎么判断数组是null还是为空怎么打印数组？怎样打印数组中的重复元素Array和ArrayList有什么区别？什么时候应该使.Array.不是ArrayList数组和链表数据结构描述，各.的时间复杂度数组有没有length()这个.法?String有没有length()这个.法 队列

队列和栈是什么，列出它们的区别 BlockingQueue是什么简述ConcurrentLinkedQueueLinkedBlockingQueue的.处和不同之处。 ArrayList、Vector、LinkedList的存储性能和特性 StringStringBuffer

ByteBuffer与StringBuffer有什么区别 HashMap

HashMap的.作原理是什么内部的数据结构是什么

HashMap的table的容量如何确定？loadFactor是什么？该容量如何变化？这种变化会带来什么问题？HashMap实现的数据结构是什么？如何实现HashMap和HashTable、ConcurrentHashMap的区别HashMap的遍历.式及效率HashMap、LinkedMap、TreeMap的区别如何决定选.HashMap还是TreeMap如果HashMap的..超过了负载因.(loadfactor)定义的容量，怎么办HashMap是线程安全的吗？并发下使.的 Map是什么，它们内部原理分别是什么，.如存储.式、hashcode、扩容、默认容量等 HashSet

HashSet和TreeSet有什么区别HashSet内部是如何.作的WeakHashMap是怎么.作的？ Set

Set.的元素是不能重复的，那么.什么.法来区分重复与否呢？是.==还是equals()？它们有何区别?TreeMap：TreeMap是采.什么树实现的？TreeMap、HashMap、LindedHashMap的区别。TreeMap和TreeSet在排序时如何.较元素？Collections.具类中的sort().法如何.较元素？TreeSet：.个已经构建好的TreeSet，怎么完成倒排序。EnumSet是什么 Hash算法

Hashcode的作.简述.致性Hash算法有没有可能两个不相等的对象有相同的hashcode？当两个对象hashcode相同怎么办？如何获取值对象为什么在重写equals.法的时候需要重写hashCode.法？equals与hashCode的异同点在哪.a.hashCode()有什么.？与a.equals(b)有什么关系 hashCode()和equals().法的重要性体现在什么地. Object：Object有哪些公..法？Object类hashcode,equals设计原则？sun为什么这么设计？Object类的概述如何在.类中为.类.动完成所有的hashcode和equals实现？这么做有何优劣。可以在hashcode()中使.随机数字吗？ LinkedHashMap

LinkedHashMap和PriorityQueue的区别是什么 List

List,Set,Map三个接.，存取元素时各有什么特点 List,Set,Map是否继承.Collection接.遍历.个List有哪些不同的.式 LinkedList

1. LinkedList是单向链表还是双向链表
2. LinkedList与ArrayList有什么区别
3.

描述下Java中集合（Collections），接.（Interfaces），实现（Implementations）的概念。LinkedList与ArrayList的区别是什么？

4.

插.数据时，ArrayList,LinkedList,Vector谁速度较快？

ArrayList

1. ArrayList和HashMap的默认..是多数

2. ArrayList和LinkedList的区别，什么时候.ArrayList?
3. ArrayList和Set的区别?
4. ArrayList,LinkedList,Vector的区别
5. ArrayList是如何实现的，ArrayList和LinkedList的区别
6. ArrayList如何实现扩容
7.

Array和ArrayList有何区别？什么时候更适合.Array

8.说出ArraL ist,Vector,LinkedList的存储性能和特性

Map

Map,Set,List,Queue,StackMap接.提供了哪些不同的集合视图为什么Map接.不继承Collection接. Collections

介绍Java中的CollectionFrameWork。集合类框架的基本接.有哪些 Collections类是什么？Collection和Collections的区别？Collection、Map的实现集合类框架的最佳实践有哪些为什么Collection不从Cloneable和Serializable接.继承说出.点Java中使.Collections的最佳实践？ Collections中遗留类(HashTable、Vector)和现有类的区别什么是B+树，B-树，列出实际的使.场景。 接.

Comparator与Comparable接.是.什么的？列出它们的区别 对象 拷.(clone)

如何实现对象克隆深拷.和浅拷.区别深拷.和浅拷.如何实现激活机制写clone().法时，通常都有..代码，是什么 .较

在.较对象时，"=="运算符和equals运算有何区别如果要重写.个对象的equals.法，还要考虑什么两个对象值相同(x.equals(y)==true)，但却可有不同的hashcode，这句话对不对 构造器

构造器链是什么创建对象时构造器的调.顺序 不可变对象

什么是不可变象（immutableobject）为什么Java中的String是不可变的（Immutable）如何构建不可变的类结构？关键点在哪.能创建.个包含可变对象的不可变对象吗如何对.组对象进.排序 .法

构造器（constructor）是否可被重写（override）.法可以同时即是static.是synchronized的吗abstract的method是否可同时是static，是否可同时是native，是否可同时是synchronizedJava.持哪种参数传递类型.个对象被当作参数传递到.个.法，是值传递还是引.传递当.个对象被当作参数传递到.个.法后，此.法可改变这个对象的属性，并可返回变化后的结果，那么这.到底是值传递还是引.传递我们能否重载main().法如果main.法被声明为private会怎样 GC 概念

GC是什么？为什么要有GC什么时候会导致垃圾回收GC是怎么样运.的新.以及永久区是什么GC有.种.式？怎么配置什么时候.个对象会被GC？如何判断.个对象是否存活 System.gc()Runtime.gc()会做什么事情？能保证GC执.吗垃圾回收器可以.上回收内存吗？有什么办法主动通知虚拟机进.垃圾回收？ MinorGC、MajorGC、YoungGC与FullGC分别在什么时候发.垃圾回收算法的实现原理如果对象的引.被置为null，垃圾收集器是否会.即释放对象占.的内存？垃圾回收的最佳做法是什么 GC收集器有哪些

垃圾回收器的基本原理是什么？串.(serial)收集器和吞吐量(throughput)收集器的区别是什么 Serial与ParallelGC之间的不同之处CMS收集器与G1收集器的特点与区别CMS垃圾回收器的.作过程JVM中.次完整的GC流程是怎样的？对象如何晋升到.年代吞吐量优先和响应优先的垃圾收集器选择 GC策略

举个实际的场景，选择.个GC策略JVM的永久代中会发.垃圾回收吗 收集.法 标记清除、标记整理、复制算法的原理与特点？分别.在什么地.如果让你优化收集.法，有什么思路 JVM 参数

说说你知道的.种主要的jvm参数 -XX:+UseCompressedOops有什么作.

类加载器(ClassLoader)

Java类加载器都有哪些JVM如何加载字节码.件 内存管理

JVM内存分哪.个区，每个区的作.是什么.个对象从创建到销毁都是怎么在这些部分.存活和转移的解释内存中的栈(stack)、堆(heap)和.法区(methodarea)的.法JVM中哪个参数是.来控制线程的栈堆栈.简述内存分配与回收策略简述重排序，内存屏障，happen-before，主内存，.作内存Java中存在内存泄漏问题吗？请举例说明简述Java中软引.（SoftReferenc）、弱引.（WeakReference）和虚引.内存映射缓存区是什么 1. jstack，jstat，jmap，jconsole怎么. 2. 32位JVM和64位JVM的最.堆内存分别是多数？32位和64位的JVM，int类型变量的.度是多数？ 3.怎样通过Java程序来判断JVM是32位还是64位

4.

JVM..会维护缓存吗？是不是在堆中进.对象分配，操作系统的堆还是JVM..管理堆

5.

什么情况下会发.栈内存溢出 6.双亲委派模型是什么

多线程 基本概念

什么是线程多线程的优点多线程的.种实现.式 1. .Runnable还是Thread

什么是线程安全

1. Vector,SimpleDateFormat是线程安全类吗 2.什么Java原型不是线程安全的 3.哪些集合类是线程安全的多线程中的忙循环是什么如何创建.个线程编写多线程程序有.种实现.式什么是线程局部变量线程和进程有什么区别？进程间如何通讯，线程间如何通讯什么是多线程环境下的伪共享（falsesharing）同步和异步有何异同，在什么情况下分别使.他们？举例说明

Current

ConcurrentHashMap和Hashtable的区别 ArrayBlockingQueue,CountDownLatch的.法ConcurrentHashMap的并发度是什么 CyclicBarrier和CountDownLatch有什么不同？各.的内部原理和.法是什么Semaphore的.法 Thread

启动.个线程是调.run()还是start().法？start()和run().法有什么区别调.start().法时会执.run().法，为什么不能直接调.run().法 sleep().法和对象的wait().法都可以让线程暂停执.，它们有什么区别 yield.法有什么作.？sleep().法和yield().法有什么区别Java中如何停..个线程 stop()和suspend().法为何不推荐使.如何在两个线程间共享数据如何强制启动.个线程如何让正在运.的线程暂停.段时间什么是线程组，为什么在Java中不推荐使.你是如何调.wait（.法的）？使.if块还是循环？为什么 .命周期

有哪些不同的线程.命周期线程状态，BLOCKED和WAITING有什么区别画.个线程的.命周期状态图 ThreadLocal.途是什么，原理是什么，.的时候要注意什么 ThreadPool

线程池是什么？为什么要使.它如何创建.个Java线程池 ThreadPool.法与优势提交任务时，线程池队列已满时会发会.什么newCache和newFixed有什么区别？简述原理。构造函数的各个参数的含义是什么，.如coreSize,maxsize等线程池的实现策略线程池的关闭.式有.种，各.的区别是什么线程池中submit()和execute().法有什么区别？ 线程调度

Java中.到的线程调度算法是什么什么是多线程中的上下.切换你对线程优先级的理解是什么什么是线程调度器(ThreadScheduler)和时间分.(TimeSlicing) 线程同步

请说出你所知的线程同步的.法 synchronized的原理是什么 synchronized和ReentrantLock有什么不同什么场景下可以使.volatile替换synchronized有T1，T2，T3三个线程，怎么确保它们按顺序执.？怎样保证T2在T1执.完后执.，T3在T2执.完后执.同步块内的线程抛出异常会发.什么当.个线程进..个对象的synchronized.法A之后，其它线程是否可进.此对象的synchronized.法B使.synchronized修饰静态.法和.静态.法有什么区别如何从给定集合那.创建.个synchronized的集合 锁

JavaConcurrencyAPI中的Lock接.是什么？对.同步它有什么优势Lock与Synchronized的区别？Lock接..synchronized块的优势是什么 ReadWriteLock是什么？锁机制有什么.什么是乐观锁（OptimisticLocking）？如何实现乐观锁？如何避免ABA问题解释以下名词：重排序，.旋锁，偏向锁，轻量级锁，可重.锁，公平锁，.公平锁，乐观锁，悲观锁什么时候应该使.可重.锁简述锁的等级.法锁、对象锁、类锁Java中活锁和死锁有什么区别？什么是死锁(Deadlock)？导致线程死锁的原因？如何确保N个线程可以访问N个资源同时.不导致死锁死锁与活锁的区别，死锁与饥饿的区别怎么检测.个线程是否拥有锁如何实现分布式锁有哪些.锁数据结构，他们实现的原理是什么读写锁可以.于什么应.场景 Executors类是什

么？Executor和Executors的区别什么是Java线程转储(ThreadDump)，如何得到它如何在Java中获取线程堆栈说出3条在Java中使.线程的最佳实践在线程中你怎么处理不可捕捉异常实际项.中使.多线程举例。你在多线程环境中遇到的常.的问题是什么？你是怎么解决它的请说出与线程同步以及线程调度相关的.法程序中有3个socket，需要多少个线程来处理假如有.个第三.接.，有很多个线程去调.获取数据，现在规定每秒钟最多有10个线程同时调.它，如何做到如何在Windows和Linux上查找哪个线程使.的CPU时间最.如何确保main().法所在的线程是Java程序最后结束的线程.常多个线程（可能是不同机器），相互之间需要等待协调才能完成某种.作，问怎么设计这种协调.案你需要实现.个.效的缓存，它允许多个..读，但只允许.个..写，以此来保持它的完整性，你会怎样去实现它 异常 基本概念

Error和Exception有什么区别

1. UnsupportedOperationException是什么
2. NullPointerException和ArrayIndexOutOfBoundException之间有什么相同之处什么是受检查的异常，什么是运.时异常运.时异常与.般异常有何异同简述.个你最常.到的runtimeexception(运.时异常)

finally

finally关键词在异常处理中如何使. 1.如果执. finally代码块之前.法返回了结果，或者JVM退出了，finally块中的代码还会执.吗

2.

try.有return，.nally还执.么？那么紧跟在这个try后的.nally{}.的code会不会被执.，什么时候被执.，在return前还是后

3.在什么情况下， finally语句不会执.

throw和throws有什么区别？OOM你遇到过哪些情况？你是怎么搞定的？SOF你遇到过哪些情况？既然我们可以.RuntimeException来处理错误，那么你认为为什么Java中还存在检查型异常当..创建异常类的时候应该注意什么导致空指针异常的原因异常处理handleordeclare原则应该如何理解怎么利.JUnit来测试.个.法的异常catch块.别不写代码有什么问题你曾经.定义实现过异常吗？怎么写的什么是异常链在try块中可以抛出异常吗 JDBC

通过JDBC连接数据库有哪.种.式阐述JDBC操作数据库的基本步骤JDBC中如何进.事务处理什么是JdbcTemplate什么是DAO模块使.JDBC操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能列出5个应该遵循的JDBC最佳实践 IO File

File类型中定义了什么.法来创建.级.录 File类型中定义了什么.法来判断.个.件是否存在 流

Java中有.种类型的流JDK为每种类型的流提供了.些抽象类以供继承，分别是哪些类对.本.件操作.什么I/O流对各种基本数据类型和String类型的读写，采.什么流能指定字符编码的I/O流类型是什么 序列化

什么是序列化？如何实现Java序列化及注意事项 Serializable与Externalizable的区别 Socket

socket选项TCPNODELAY是指什么Socket.作在TCP/IP协议栈是哪.层TCP、UDP区别及Java实现.式 说.点IO的最佳实践直接缓冲区与.直接缓冲器有什么区别？怎么读写ByteBuffer？ByteBuffer中的字节序是什么当.System.in.read(buffer)从键盘输...n个字符后，存储在缓冲区buffer中的字节数是多少如何使.扫描器类（ScannerClass）令牌化 .向对象编程（OOP）

解释下多态性（polymorphism），封装性（encapsulation），内聚（cohesion）以及耦合（coupling）多态的实现原理封装、继承和多态是什么对象封装的原则是什么?类 1.获得.个类的类对象有哪些.式 2.重载（Over load）和重写（Override）的区别。重载的.法能否根据返回类型进.区分?

3.

说出.条Java中.法重载的最佳实践

抽象类 1. 抽象类和接.的区别

2.

抽象类中是否可以有静态的main.法

3.

抽象类是否可实现(implements)接.

4.

抽象类是否可继承具体类(concreteclass)

匿名类（AnonymousInnerClass） 1. 匿名内部类是否可以继承其它类？是否可以实现接.

内部类

1.

内部类分为.种

2.

内部类可以引.它的包含类（外部类）的成员吗

3.

请说.下Java中为什么要引.内部类？还有匿名内部类

继承

1.继承（ Inheritance）与聚合（Aggregation）的区别在哪. 2.继承和组合之间有什么不同

3.

为什么类只能单继承，接.可以多继承

4.

存在两个类，B继承A，C继承B，能将B转换为C么？如C=(C)B

5.

如果类a继承类b，实现接.c，.类b和接.c中定义了同名变量，请问会出现什么问题

接. 1. 接.是什么

2.

接.是否可继承接.

3.

为什么要使.接..不是直接使.具体类？接.有什么优点

泛型

泛型的存在是.来解决什么问题泛型的常.特点 List能否转为List .具类 .历

CalendarClass的.途如何在Java中获取.历类的实例解释.些.历类中的重要.法 GregorianCalendar类是什么 SimpleTimeZone类是什么 Locale类是什么如何格式化.期对象如何添加.时(hour)到.个.期对象(DateObjects)如何将字符串YYYYMMDD转换为.期 Math

Math.round()什么作.？Math.round(11.5)等于多少？Math.round(-11.5)等于多少？ XML

XML.档定义有.种形式？它们之间有何本质区别？解析XML.档有哪.种.式？DOM和SAX解析器有什么不同？Java解析XML的.式 .jdom解析xml.件时如何解决中.问题？如何解析你在项.中.到了XML技术的哪些..？如何实现 动态代理

描述动态代理的.种实现.式，分别说出相应的优缺点 设计模式

什么是设计模式（DesignPatterns）？你.过哪种设计模式？.在什么场合你知道哪些商业级设计模式？哪些设计模式可以增加系统的可扩展性单例模式 1. 除了单例模式，你在.产环境中还.过什么设计模式？

2.写S ingleton单例模式 3.单例模式的双检锁是什么

4.

如何创建线程安全的Singleton 5.什么是类的单例模式 6.写出三种单例模式实现

适配器模式

1.

适配器模式是什么？什么时候使. 2.适配器模式和代理模式之前有什么不同 3.适配器模式和装饰器模式有什么区别

什么时候使.享元模式什么时候使.组合模式什么时候使.访问者模式什么是模板.法模式请给出1个符合开闭原则的设计模式的例. 开放问题

..句话概括Web编程的特点 Google是如何在.秒内把搜索结果返回给..哪种依赖注..式你建议使.，构造器注.，还是Setter.法注.树（.叉或其他）形成许多普通数据结构的基础。请描述.些这样的数据结构以及何时可以使.它们某.项功能如何设计线上系统突然变得异常缓慢，你如何查找问题什么样的项.不适合.框架新浪微博是如何实现把微博推给订阅者简要介绍下从浏览器输.URL开始到获取到请求界.之后JavaWeb应.中发.了什么请你谈谈SSH整合.并发下，如何做到安全的修改同..数据12306.站的订票系统如何实现，如何保证不会票不被超卖.站性能优化如何优化的聊了下曾经参与设计的服务器架构请思考.个.案，实现分布式环境下的countDownLatch请思考.个.案，设计.个可以控制缓存总体..的.动适应的本地缓存在你的职业.涯中，算得上最困难的技术挑战是什么如何写.篇设计.档，.录是什么.写的O是什么？举.个例.编程中..都怎么考虑.些设计原则的，.如开闭原则，以及在.作中的应.解释.下.络应.的模式及其特点设计.个在线.档系统，.档可以被编辑，如何防.多.同时对同.份.档进.编辑更新说出数据连接池的.作机制是什么怎么获取.个.件中单词出现的最.频率描述.下你最常.的编程.格

如果有机会重新设计你们的产品，你会怎么做如何搭建.个.可.系统如何启动时不需输...名与密码如何在基于Java的Web项.中实现.件上传和下载如何实现.个秒杀系统，保证只有.位..能买到某件商品。如何实现负载均衡，有哪些算法可以实现如何设计.个购物.？想想淘宝的购物.如何实现的如何设计.套.并发.付.案，架构如何设计如何设计建.和保持100w的.连接如何避免浏览器缓存。如何防.缓存雪崩如果AB两个系统互相依赖，如何解除依如果有.恶意创建.法连接，怎么解决如果有..亿的.名单，每天.天需要.并发查询，晚上需要更新.次，如何设计这个功能如果系统要使.超.整数（超过long.度范围），请你设计.个数据结构来存储这种超.型数字以及设计.种算法来实现超.整数加法运算）如果要设计.个图形系统，请你设计基本的图形元件(Point,Line,Rectangle,Triangle)的简单实现如果让你实现.个并发安全的链表，你会怎么做应.服务器与WEB服务器的区别？应.服务器怎么监控性能，各种.式的区别？你使.过的应.服务器优化技术有哪些.型.站在架构上应当考虑哪些问题有没有处理过线上问题？出现内存泄露，CPU利.率标.，应..响应时如何处理的最近看什么书，印象最深刻的是什么描述下常.的重构技巧你使.什么版本管理.具？分.（Branch）与标签（Tag）之间的区别在哪.你有了解过存在哪些反模式（Anti-Patterns）吗你.过的.站前端优化的技术有哪些如何分析Threaddump你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织.（Weaving）、切.（Aspect）这些概念你是如何处理内存泄露或者栈溢出问题的你们线上应.的JVM参数有哪些怎么提升系统的QPS和吞吐量 知识.

解释什么是MESI协议(缓存.致性)谈谈reactor模型Java9带来了怎样的新功能Java与C++对.，C++或Java中的异常处理机制的简单原理和应.简单讲讲Tomcat结构，以及其类加载器流程虚拟内存是什么阐述下SOLID原则请简要讲.下你对测试驱动开发（TDD）的认识CDN实现原理Maven和ANT有什么区别UML中有哪些常.的图 Linux

1. Linux下IO模型有.种，各.的含义是什么。
2. Linux系统下你关注过哪些内核参数，说说你知道的
3. Linux下...命令查看.件的最后五. 4.平时.到哪些L inux命令

4.

...命令输出正在运.的Java进程

6.

使.什么命令来确定是否有Tomcat实例运.在机器上

什么是N+1难题什么是paxos算法 什么是restful，讲讲你理解的restful 什么是zab协议 什么是领域模型(domainmodel)？贫.模型(anaemicdomainmodel)和充.模型(richdomainmodel)有什么区别 什么是领域驱动开发（DomainDrivenDevelopment） 介绍.下了解的Java领域的WebService框架 WebServer、WebContainer与ApplicationServer的区别是什么 微服务（MicroServices）与巨.型应.（MonolithicApplications）之间的区别在哪. 描述Cookie和Session的作.，区别和各.的应.范围，Session.作原理 你常.的持续集成（ContinuousIntegration）、静态代码分析（StaticCodeAnalysis）.具有哪些 简述下数据库正则化（Normalizations） KISS,DRY,YAGNI等原则是什么含义 分布式事务的原理，优缺点，如何使.分布式事务？ 布式集群下如何做到唯.序列号 .络 1. HTTPS的加密.式是什么，讲讲整个加密解密流程

2.

HTTPS和HTTP的区别 3.HTTP连接池实现原理

4.

HTTP集群.案

5. Nginx、lighttpd、Apache三.主流Web服务器的区别

是否看过框架的.些代码持久层设计要考虑的问题有哪些？你.过的持久层框架有哪些 数值提升是什么 你能解释.下..替换原则吗 你是如何测试.个应.的？知道哪些测试框架 传输层常.编程协议有哪些？并说出各.的特点 编程题 计算加班费 加班10.时以下加班费是时薪的1.5倍。加班10.时或以上，按4元/时算。提.： （.个..作26天，.天正常.作8.时）

计算1000. -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.从零搭建创业公司后台技术栈

2.如何阅读 Java源码？

3.

某.公司RESTful、前后端分离的实践

4.该如何弥补 GitHub功能缺陷？

5.团队开发中 Git最佳实践

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Maven实战问题和最佳实践 静默虚空Java后端 2019-11-07 点击上.Java后端，选择设为星标优质.章，及时送达

作者 |dunwu来源 |github.com/dunwu/java-tutorial .、常.问题 1、dependencies和dependencyManagement，plugins和pluginManagement有什么区别？ dependencyManagement是表.依赖jar包的声明，即你在项.中的dependencyManagement下声明了依赖，maven不会加载该依赖，dependencyManagement声明可以被继承。dependencyManagement的.个使.案例是当有..项.的时候，.项.中可以利.dependencyManagement声明.项.中需要.到的依赖jar包，之后，当某个或者某.个.项.需要加载该插件的时候，就可以在.项.中dependencies节点只配置 groupId和artifactId就可以完成插件的引.。 dependencyManagement主要是为了统.管理插件，确保所有.项.使.的插件版本保持.致，类似的还有plugins和 pluginManagement。 2、IDEA修改JDK版本后编译报错 错误现象 修改JDK版本，指定maven-compiler-plugin的source和target为1.8。然后，在IntellijIDEA中执.maven指令，报错：

[ERROR]Failedtoexecutegoalorg.apache.maven.plugins:maven-compiler-plugin:3.0:compile(default-compile)onprojectapo 错误原因 maven的JDK源与指定的JDK编译版本不符。 排错.段 查看ProjectSettings ProjectSDK是否正确

SDK路径是否正确

查看Settings>Maven的配置 JDKforimporter是否正确

Runner是否正确

3.、重复引.依赖 在Idea中，选中Module，使.Ctrl+Alt+Shift+U，打开依赖图，检索是否存在重复引.的情况。如果存在重复引.，可以将多余的引.删除。 Tips：关注微信公众号：Java后端，每.获取技术博.推送。 4、如何打包.个可以直接运.的SpringBootjar包 可以使.spring-boot-maven-plugin插件 org.springframework.boot spring-boot-maven-plugin repackage 如果引.了第三.jar包，如何打包？ .先，要添加依赖 io.github.dunwu dunwu-common 1.0.0 system ${project.basedir}/src/main/resources/lib/dunwu-common-1.0.0.jar 接着，需要配置spring-boot-maven-plugin插件：org.springframework.boot spring-boot-maven-plugin repackage true 5、去哪.找mavendependency？ 问：刚接触maven的新.，往往会有这样的疑问，我该去哪.找jar？ 答：官.推荐的搜索mavendependency.址：https://search.maven.orghttps://repository.apache.orghttps://mvnrepository.com 6、如何指定编码？ 问：众所周知，不同编码格式常常会产.意想不到的诡异问题，那么maven构建时如何指定maven构建时的编码？ 答：在properties中指定 project.build.sourceEncoding <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> 7、如何指定JDK版本？ 问：如何指定maven构建时的JDK版本？ 答：有两种.法： （1）properties.式 ... <maven.compiler.source>1.7</maven.compiler.source> <maven.compiler.target>1.7</maven.compiler.target> ... （2）使.maven-compiler-plugin插件，并指定source和target版本 ... org.apache.maven.plugins maven-compiler-plugin 3.3

1.7 1.7 ... 8、如何避免将dependency打包到构件中？ 答：指定mavendependency的scope为 provided，这意味着：依赖关系将在运.时由其容器或JDK提供。具有此范围的依赖关系不会传递，也不会捆绑在诸如WAR之类的包中，也不会包含在运.时类路径中。 9、如何跳过单元测试 问：执.mvnpackage或mvninstall时，会.动编译所有单元测试(src/test/java.录下的代码)，如何跳过这.步？ 答：在执.命令的后.，添加命令.参数 -Dmaven.test.skip=true或者 -DskipTests=true 10、IDEA修改JDK版本后编译报错 错误现象 修改JDK版本，指定maven-compiler-plugin的source和target为1.8。然后，在IntellijIDEA中执.maven指令，报错：
[ERROR]Failedtoexecutegoalorg.apache.maven.plugins:maven-compiler-plugin:3.0:compile(default-compile)onprojectapo 错误原因 maven的JDK源与指定的JDK编译版本不符。 排错.段 查看ProjectSettings ProjectSDK是否正确

SDK路径是否正确

查看Settings>Maven的配置 JDKforimporter是否正确

Runner是否正确

11、重复引.依赖 在Idea中，选中Module，使.Ctrl+Alt+Shift+U，打开依赖图，检索是否存在重复引.的情况。 12、如何引.本地jar 问：有时候，需要引.在中央仓库找不到的jar，但.想通过maven进.管理，那么应该如何做到呢？答：可以通过设置dependency的scope为system来引.本地jar。例： 将私有jar放置在resouces/lib下，然后以如下.式添加依赖：groupId和artifactId可以按照jar包中的package设置，只要和其他jar不冲突即可。 xxx xxx 1.0.0 system ${project.basedir}/src/main/resources/lib/xxx-6.0.0.jar 13、如何排除依赖 问：如何排除依赖.个依赖关系？..项.中使.的libA依赖某个库的1.0版。libB以来某个库的2.0版，如今想统.使.2.0版，怎样去掉1.0版的依赖？ 答：通过exclusion排除指定依赖即可。 例： org.apache.zookeeper zookeeper 3.4.12 true org.slf4j slf4j-log4j12 .、最佳实践 1、通过bom统.管理版本 采.类似 spring-boot-dependencies的.式统.管理依赖版本。 spring-boot-dependencies的pom.xml形式：

-

12306的架构到底有多.逼？

5.团队开发中 Git最佳实践

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

MySQL.频.试题，都在这了 Java后端 2019-10-22

作者|呼延.链接|juejin.im/post/5d351303f265da1bd30596f9 前. 本.主要受众为开发.员,所以不涉及到MySQL的服务部署等操作,且内容较多,.家准备好耐.和..矿泉.. 前.阵系统的学习了.下MySQL,也有.些实际操作经验,偶然看到.篇和MySQL相关的.试.章,发现其中的.些问题..也回答不好,虽然知识点.部分都知道,但是.法将知识串联起来.因此决定搞.个MySQL灵魂100问,试着.回答问题的.式,让..对知识点的理解更加深..点.此.不会事.巨细的从select的.法开始讲解mysql,主要针对的是开发.员需要知道的.些MySQL的知识点,主要包括索引,事务, 优化等..,以在.试中.频的问句形式给出答案. 1.什么是索引? 索引是.种数据结构,可以帮助我们快速的进.数据的查找. 2.索引是个什么样的数据结构呢? 索引的数据结构和具体存储引擎的实现有关,在MySQL中使.较多的索引有Hash索引,B+树索引等,.我们经常使.的InnoDB存储引擎的默认索引实现为:B+树索引. 3.Hash索引和B+树所有有什么区别或者说优劣呢? .先要知道Hash索引和B+树索引的底层实现原理:hash索引底层就是hash表,进.查找时,调..次hash函数就可以获取到相应的键值,之后进.回表查询获得实际数据.B+树底层实现是多路平衡查找树.对于每.次的查询都是从根节点出发,查找到叶.节点.可以获得所查键值,然后根据查询判断是否需要回表查询数据. 那么可以看出他们有以下的不同:

hash索引进.等值查询更快(.般情况下),但是却.法进.范围查询.因为在hash索引中经过hash函数建.索引之后,索引的顺序与原顺序.法保持.致,不能.持范围查询..B+树的的所有节点皆遵 循(左节点.于.节点,右节点.于.节点,多叉树也类似),天然.持范围.

hash索引不.持使.索引进.排序,原理同上. hash索引不.持模糊查询以及多列索引的最左前缀匹配.原理也是因为hash函数的不可预测.AAAA和AAAAB的索引没有相关 性.hash索引任何时候都避免不了回表查询数据,.B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询.hash索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在.量重复的时候,发.hash碰撞,此时效率可能极差..B+树的查询效率.较稳定,对于所有的查询都是从根节点到叶.节点,且树的.度较低.因此,在.多数情况下,直接选择B+树索引可以获得稳定且较好的查询速度..不需要使.hash索引. 4.上.提到了B+树在满.聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引? 在B+树的索引中,叶.节点可能存储了当前的key值,也可能存储了当前的key值以及整.的数据,这就是聚簇索引和.聚簇索引.在 InnoDB中,只有主键索引是聚簇索引,如果没有主键,则挑选.个唯.键建.聚簇索引.如果没有唯.键,则隐式的.成.个键来建.聚簇索引. 当查询使.聚簇索引时,在对应的叶.节点,可以获取到整.数据,因此不.再次进.回表查询. 5..聚簇索引.定会回表查询吗? 不.定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进.回表查询.举个简单的例.,假设我们在员.表的年龄上建.了索引,那么当进.selectagefromemployeewhereage<20的查询时,在索引的叶.节点上,已经包含了age信息,不会再次进.回表查询. 6.在建.索引的时候,都有哪些需要考虑的因素呢? 建.索引的时候.般要考虑到字段的使.频率,经常作为条件进.查询的字段.较适合.如果需要建.联合索引的话,还需要考虑联合索引中的顺序.此外也要考虑其他..,.如防.过多的所有对表造成太.的压..这些都和实际的表结构以及查询.式有关. 7.联合索引是什么?为什么需要注意联合索引中的顺序? MySQL可以使.多个字段同时建..个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建.索引时的字段顺序挨个使.,否则.法命中索引. 具体原因为:MySQL使.索引时需要索引有序,假设现在建.了"name,age,school"的联合索引,那么索引的排序为:先按照name排序,如果name相同,则按照age排序,如果age的值也相等,则按照school进.排序. 当进.查询时,此时索引仅仅按照name严格有序,因此必须.先使.name字段进.等值查询,之后对于匹配到的列..,其按照age字段严格有序,此时可以使.age字段.做索引查找,,,以此类推.因此在建.联合索引的时候应该注意索引列的顺序,.般情况下,将查询需求频繁或者字段选择性.的列放在前..此外可以根据特例的查询或者表结构进.单独的调整. 8.创建的索引有没有被使.到?或者说怎么才可以知道这条语句运.很慢的原因? MySQL提供了explain命令来查看语句的执.计划,MySQL在执.某个语句之前,会将该语句过.遍查询优化器,之后会拿到对语句的分析,也就是执.计划,其中包含了许多信息.可以通过其中和索引有关的信息来分析是否命中了索引,例如 possilbe_key,key,key_len等字段,分别说明了此语句可能会使.的索引,实际使.的索引以及使.的索引.度. 9.那么在哪些情况下会发.针对该列创建了索引但是在查询的时候并没有使.呢?

使.不等于查询,列参与了数学运算或者函数在字符串like时左边是通配符.类似于'%aaa'.当mysql分析全表扫描.使.索引快的时候不使.索引.当使.联合索引,前..个条件为范围查询,后.的即使符合最左前缀原则,也.法使.索引.以上情况,MySQL.法使.索引.事务相关 1.什么是事务? 理解什么是事务最经典的就是转账的栗.,相信.家也都了解,这.就不再说.边了.事务是.系列的操作,他们要符合ACID特性.最常.的理解就是:事务中的操作要么全部成功,要么全部失败.但是只是这样还不够的. 2.ACID是什么?可以详细说.下吗? A=Atomicity原.性,就是上.说的,要么全部成功,要么全部失败.不可能只执..部分操作. C=Consistency系统(数据库)总是从.个.致性的状态转移到另.个.致性的状态,不会存在中间状态. I=Isolation隔离性:通常来说:.个事务在完全提交之前,对其他事务是不可.的.注意前.的通常来说加了红.,意味着有例外情况. D=Durability持久性,.旦事务提交,那么就永远是这样.了,哪怕系统崩溃也不会影响到这个事务的结果. 3.同时有多个事务在进.会怎么样呢? 多事务

的并发进..般会造成以下.个问题:脏读:A事务读取到了B事务未提交的内容,.B事务后.进.了回滚.不可重复读:当设置A事务只能读取B事务已经提交的部分,会造成在A事务内的两次查询,结果竟然不.样,因为在此期间B事务

进.了提交操作.幻读:A事务读取了.个范围的内容,.同时B事务在此期间插.了.条数据.造成"幻觉". 4.怎么解决这些问题呢? MySQL的事务隔离级别了解吗? MySQL的四种隔离级别如下:未提交读(READUNCOMMITTED)

这就是上.所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,.之后该事务进.了回滚).这个级别的性能没有.够.的优势,但是.有很多的问题,因此很少使..

已提交读(READCOMMITTED) 其他事务只能读取到本事务已经提交的部分.这个隔离级别有不可重复读的问题,在同.个事务内的两次读取,拿到的结果竟然不.样,因为另外.个事务对数据进.了修改. REPEATABLEREAD(可重复读)可重复读隔离级别解决了上.不可重复读的问题(看名字也知道),但是仍然有.个新问题,就是幻读,当你读取id>10的数据.时,对

涉及到的所有.加上了读锁,此时例外.个事务新插.了.条id=11的数据,因为是新插.的,所以不会触发上.的锁的排斥,那么进.本事务进.下.次的查询时会发现有.条id=11的数据,.上次的查询操作并没有获取到,再进.插.就会有主键冲突的问题.

SERIALIZABLE(可串.化) 这是最.的隔离级别,可以解决上.提到的所有问题,因为他强制将所以的操作串.执.,这会导致并发性能极速下降,因此也不是很常.. 5.Innodb使.的是哪种隔离级别呢? InnoDB默认使.的是可重复读隔离级别. 6.对MySQL的锁了解吗? 当数据库有并发事务的时候,可能会产.数据的不.致,这时候需要.些机制来保证访问的次序,锁机制就是这样的.个机制.就像酒店的房间,如果.家随意进出,就会出现多.抢夺同.个房间的情况,.在房间上装上锁,申请到钥匙的.才可以.住并且将房间锁起来,其他.只有等他使.完毕才可以再次使.. 7.MySQL都有哪些锁呢?像上.那样.进.锁定岂不是有点阻碍并发效率了? 从锁的类别上来讲,有共享锁和排他锁.共享锁:.叫做读锁.当..要进.数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.排他锁:.叫做写锁.当..要进.数据的写.时,对数据加上排他锁.排他锁只可以加.个,他和其他的排他锁,共享锁都相斥..上.的例.来说就是..的.为有两种,.种是来看房,多个...起看房是可以接受的..种是真正的.住.晚,在这期间,.论是 想.住的还是想看房的都不可以.锁的粒度取决于具体的存储引擎,InnoDB实现了.级锁,.级锁,表级锁.他们的加锁开销从...,并发能.也是从.到..表结构设计 1.为什么要尽量设定.个主键? 主键是数据库确保数据.在整张表唯.性的保障,即使业务上本张表没有主键,也建议添加.个.增.的ID列作为主键.设定了主键之后,在后续的删改查的时候可能更加快速以及确保操作数据范围安全. 2.主键使..增ID还是UUID? 推荐使..增ID,不要使.UUID.因为在InnoDB存储引擎中,主键索引是作为聚簇索引存在的,也就是说,主键索引的B+树叶.节点上存储了主键索引以及全部的数 据(按照顺序),如果主键索引是.增ID,那么只需要不断向后排列即可,如果是UUID,由于到来的ID与原来的..不确定,会造成.常多的数据插.,数据移动,然后导致产.很多的内存碎.,进.造成插.性能的下降.总之,在数据量..些的情况下,..增主键性能会好.些. 图.来源于《.性能MySQL》:其中默认后缀为使..增ID,_uuid为使.UUID为主键的测试,测试了插.100w.和300w.的性能.

关于主键是聚簇索引,如果没有主键,InnoDB会选择.个唯.键来作为聚簇索引,如果没有唯.键,会.成.个隐式的主键. IfyoudefineaPRIMARYKEYonyourtable,InnoDBusesitastheclusteredindex. Ifyoudonotde.neaPRIMARYKEY foryourtable,MySQLpicksthe.rstUNIQUE indexthathasonlyNOTNULL columnsastheprimarykeyandInnoDBusesitastheclusteredindex. 3.字段为什么要求定义为notnull? MySQL官.这样介绍: NULL columns require additional space in the rowto record whether their values are NULL. For MyISAM tables, eachNULLcolumntakesonebitextra,roundeduptothenearestbyte. null值会占.更多的字节,且会在程序中造成很多与预期不符的情况. 4.如果要存储..的密码散列,应该使.什么字段进.存储? 密码散列,盐,...份证号等固定.度的字符串应该使.char.不是varchar来存储,这样可以节省空间且提.检索效率.存储引擎相关 1.MySQL.持哪些存储引擎? MySQL.持多种存储引擎,.如InnoDB,MyISAM,Memory,Archive等等.在.多数的情况下,直接选择使.InnoDB引擎都是最合适的,InnoDB也是MySQL的默认存储引擎. 16. InnoDB和MyISAM有什么区别?

InnoDB.持事物，.MyISAM不.持事物

InnoDB.持.级锁，.MyISAM.持表级锁

InnoDB.持MVCC,.MyISAM不.持

InnoDB.持外键，.MyISAM不.持

InnoDB不.持全.索引，.MyISAM.持。零散问题

1.MySQL中的varchar和char有什么区别. char是.个定.字段,假如申请了char(10)的空间,那么.论实际存储多少内容.该字段都占.10个字符,.varchar是变.的,也就是 说申请的只是最..度,占.的空间为实际字符.度+1,最后.个字符存储使.了多.的空间.

在检索效率上来讲,char>varchar,因此在使.中,如果确定某个字段的值的.度,可以使.char,否则应该尽量使.varchar.例如存储..MD5加密后的密码,则应该使.char. 2.varchar(10)和int(10)代表什么含义? varchar的10代表了申请的空间.度,也是可以存储的数据的最..度,.int的10只是代表了展.的.度,不.10位以0填充.也就是说,int(1)和int(10)所能存储的数字..以及占.的空间都是相同的,只是在展.时按照.度展.. 3.MySQL的binlog有有.种录.格式?分别有什么区别? 有三种格式,statement,row和mixed.

statement模式下,记录单元为语句.即每.个sql造成的影响会记录.由于sql的执.是有上下.的,因此在保存的时候需要保存相关的信息,同时还有.些使.了函数之类的语句.法被记录复制.row级别下,记录单元为每..的改动,基本是可以全部记下来但是由于很多操作,会导致.量.的改动(.如alter table),因此 这种模式的.件保存的信息太多,.志量太.. mixed..种折中的.案,普通操作使.statement记录,当.法使.statement的时候使.row.此外,新版的MySQL中对row级别也做了.些优化,当表结构发.变化的时候,会记录语句.不是逐.记录. 4.超.分.怎么处理? 超.的分..般从两个.向上来解决.

数据库层.,这也是我们主要集中关注的(虽然收效没那么.),类似于select * from table where age > 20 limit1000000,10这种查询其实也是有可以优化的余地的.这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然.较慢.当时我们可以修改为select*fromtablewhereidin(selectidfromtablewhereage>20limit1000000,10).这样虽然也load了.百万的数据,但是由于索引覆盖.要查询的所有字段都在索引中,所以速度会很快.同时如果ID连续的好,我们还可以select*fromtablewhereid>1000000limit10,效率也是不错的,优化的可能性有许多种,但是核.思想都.样,就是减少load的数据.

从需求的.度减少这种请求....主要是不做类似的需求(直接跳转到.百万.之后的具体某...只允许逐.查看或者按照给定的路线.,这样可预测,可缓存)以及防.ID泄漏且连续被.恶意攻击. 解决超.分.,其实主要是靠缓存,可预测性的提前查到内容,缓存.redis等k-V数据库中,直接返回即可.在阿.巴巴《Java开发.册》中,对超.分.的解决办法是类似于上.提到的第.种.

5.关.过业务系统..的sql耗时吗?统计过慢查询吗?对慢查询都怎么优化过? 在业务系统中,除了使.主键进.的查询,其他的我都会在测试库上测试其耗时,慢查询的统计主要由运维在做,会定期将业务中的慢查询反馈给我们.慢查询的优化.先要搞明.慢的原因是什么?是查询条件没有命中索引?是load了不需要的数据列?还是数据量太.?所以优化也是针对这三个.向来的,

.先分析语句,看看是否load了额外的数据,可能是查询了多余的.并且抛弃掉了,可能是加载了许多结果中并不需要的列,对语句进.分析以及重写.

分析语句的执.计划,然后获得其使.索引的情况,之后修改语句或者修改索引,使得语句可以尽可能的命中索引.

如果对语句的优化已经.法进.,可以考虑表中的数据量是否太.,如果是的话可以进.横向或者纵向的分表.

6.上.提到横向分表和纵向分表,可以分别举.个适合他们的例.吗? 横向分表是按.分表.假设我们有.张..表,主键是.增ID且同时是..的ID.数据量较.,有1亿多条,那么此时放在.张表.的查询效果就不太理想.我们可以根据主键ID进.分表,.论是按尾号分,或者按ID的区间分都是可以的.假设按照尾号0-99分为100个表,那么每张表中的数据就仅有100w.这时的查询效率.疑是可以满.要求的. 纵向分表是按列分表.假设我们现在有.张.章表.包含字段id-摘要-内容..系统中的展.形式是刷新出.个列表,列表中仅包含标题和摘要,当..点击某篇.章进.详情时才需要正.内容.此时,如果数据量.,将内容这个很.且不经常使.的列放在.起会拖慢原表的查询速度.我们可以将上.的表分为两张.id-摘要,id-内容.当..点击详情,那主键再来取.次内容即可..增加的存储量只是很.的主键字段.代价很.. 当然,分表其实和业务的关联度很.,在分表之前.定要做好调研以及benchmark.不要按照..的猜想盲.操作. 7.什么是存储过程？有哪些优缺点？ 存储过程是.些预编译的SQL语句。1、更加直.的理解：存储过程可以说是.个记录集，它是由.些T-SQL语句组成的代码块，这些T-SQL语句代码像.个.法.样实现.些功能（对单表或多表的增删改查），然后再给这个代码块取.个名字，在.到这个功能的时候调.他就.了。2、存储过程是.个预编译的代码块，执.效率.较.,.个存储过程替代.量T_SQL语句，可以降低.络通信量，提.通信速率,可以.定程度上确保数据安全 但是,在互联.项.中,其实是不太推荐存储过程的,.较出名的就是阿.的《Java开发.册》中禁.使.存储过程,我个.的理解是,在互联.项.中,迭代太快,项.的.命周期也.较短,.员流动相.于传统的项.也更加频繁,在这样的情况下,存储过程的管理确实是没有那么.便,同时,复.性也没有写在服务层那么好. 8.说.说三个范式 第.范式:每个列都不可以再拆分.第.范式:.主键列完全依赖于主键,.不能是依赖于主键的.部分.第三范式:.主键列只依赖 于主键,不依赖于其他.主键.在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有.够的理由..如性能.事实上我们经常会为了性能.妥协数据库的设计. 9.MyBatis中的#和$有什么区别? 乱.了.个奇怪的问题.....我只是想单独记录.下这个问题,因为出现频率太.了. #会将传.的内容当做字符串,.$会直接将传.值拼接在sql语句中.所以#可以在.定程度上预防sql注.攻击. tips：.家可以关注微信公众号：Java后端，获取更多优秀博.推送。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.感受 Lambda之美，推荐收藏，需要时查阅

2.如何优雅的导出 Excel

3..艺互联.公司 vs .逼互联.公司

4.

Java开发中常.的 4种加密.法

5.团队开发中 Git最佳实践

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ Tomcat相关.试题，看这篇！ Java后端 2019-09-10 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

Tomcat相关的.试题出场的.率并不.，正式因为如此，很多.忽略了对Tomcat相关技能的掌握，下.这.篇.章最早发布在知识星球，整理了Tomcat相关的系统架构，介绍了Server、Service、Connector、Container之间的关系，各个模块的功能，可以说把这.个掌握住了，Tomcat相关的.试题你就不会有任何问题了！另外，在.试的时候你还要有意识.意识的往Tomcat这个地.引，就.如说常.的SpringMVC的执.流程，.个URL的完整调.链路，这些相关的题.你是可以再往Tomcat处理请求的这个过程去说的！掌握注Tomcat这些技能了，.试官.定会佩服你的！ 学了本节之后你应该明.的是：

Server、Service、Connector、Container四.组件之间的关系和联系，以及他们的主要功能点；Tomcat执.的整体架构，请求是如何被.步步处理的； Engine、Host、Context、Wrapper相关的概念关系； Container是如何处理请求的；Tomcat.到的相关设计模式； .、Tomcat顶层架构 俗话说，站在巨.的肩膀上看世界，.般学习的时候也是先总览.下整体，然后逐个部分个个击破，最后形成思路，了解具体细节，Tomcat的结构很复杂，但是Tomcat.常的模块化，找到了Tomcat最核.的模块，问题才可以游刃.解，了解了Tomcat的整体架构对以后深.了解Tomcat来说.关重要！ 先上.张Tomcat的顶层结构图（图A），如下： Tomcat中最顶层的容器是Server，代表着整个服务器，从上图中可以看出，.个Server可以包含.少.个Service，.于具体提供服务。 Service主要包含两个部分：Connector和Container。从上图中可以看出Tomcat的.脏就是这两个组件，他们的作.如下： 1、Connector.于处理连接相关的事情，并提供Socket与Request和Response相关的转化; 2、Container.于封装和管理Servlet，以及具体处理Request请求； .个Tomcat中只有.个Server，.个Server可以包含多个Service，.个Service只有.个Container，但是可以有多个Connectors，这是因为.个服务可以有多个连接，如同时提供Http和Https链接，也可以提供向相同协议不同端.的连接,.意图如下（Engine、Host、Context下边会说到）： 多个Connector和.个Container就形成了.个Service，有了Service就可以对外提供服务了，但是Service还要.个.存的环境， 必须要有.能够给她.命、掌握其.死.权，那就.Server莫属了！所以整个Tomcat的.命周期由Server控制。 另外，上述的包含关系或者说是..关系，都可以在tomcat的conf.录下的server.xml配置.件中看出，下图是删除了注释内容之后的.个完整的server.xml配置.件（Tomcat版本为8.0） 详细的配置.件.件内容可以到Tomcat官.查看：http://tomcat.apache.org/tomcat-8.0-doc/index.html 上边的配置.件，还可以通过下边的.张结构图更清楚的理解：Server标签设置的端.号为8005，shutdown="SHUTDOWN"，表.在8005端.监听"SHUTDOWN"命令，如果接收到了就会关闭Tomcat。.个Server有.个Service，当然还可以进.配置，.个Service有多个，Service左边的内容都属于Container的，Service下边是Connector。 .、Tomcat顶层架构.结： （1）Tomcat中只有.个Server，.个Server可以有多个Service，.个Service可以有多个Connector和.个Container； （2）Server掌管着整个Tomcat的.死.权； （4）Service是对外提供服务的； （5）Connector.于接受请求并将请求封装成Request和Response来具体处理； （6）Container.于封装和管理Servlet，以及具体处理request请求； 知道了整个Tomcat顶层的分层架构和各个组件之间的关系以及作.，对于绝.多数的开发.员来说Server和Service对我们来说确实很远，.我们开发中绝.部分进.配置的内容是属于Connector和Container的，所以接下来介绍.下Connector和Container。 三、Connector和Container的微妙关系 由上述内容我们.致可以知道.个请求发送到Tomcat之后，.先经过Service然后会交给我们的Connector，Connector.于接收请求并将接收的请求封装为Request和Response来具体处理，Request和Response封装完之后再交由Container进.处理，Container处理完请求之后再返回给Connector，最后在由Connector通过Socket将处理的结果返回给客.端，这样整个请求的就处理完了！Connector最底层使.的是Socket来进.连接的，Request和Response是按照HTTP协议来封装的，所以Connector同时需要实现 TCP/IP协议和HTTP协议！ Tomcat既然处理请求，那么肯定需要先接收到这个请求，接收请求这个东西我们.先就需要看.下Connector！关注微信公众号「web_resourc」,回复Java领取2019最新资源。 四、Connector架构分析 Connector.于接受请求并将请求封装成Request和Response，然后交给Container进.处理，Container处理完之后在交给Connector返回给客.端。 因此，我们可以把Connector分为四个..进.理解： （1）Connector如何接受请求的？ （2）如

何将请求封装成Request和Response的？ （3）封装完之后的Request和Response如何交给Container进.处理的？ （4）Container处理完之后如何交给Connector并返回给客.端的？ .先看.下Connector的结构图（图B），如下所.：Connector就是使.ProtocolHandler来处理请求的，不同的ProtocolHandler代表不同的连接类型，.如：Http11Protocol使.的是普通Socket来连接的，Http11NioProtocol使.的是NioSocket来连接的。 其中ProtocolHandler由包含了三个部件：Endpoint、Processor、Adapter。 （1）Endpoint.来处理底层Socket的.络连接，Processor.于将Endpoint接收到的Socket封装成Request，Adapter.于将Request交给Container进.具体的处理。 （2）Endpoint由于是处理底层的Socket.络连接，因此Endpoint是.来实现TCP/IP协议的，.Processor.来实现HTTP协议的，Adapter将请求适配到Servlet容器进.具体的处理。 （3）Endpoint的抽象实现AbstractEndpoint..定义的Acceptor和AsyncTimeout两个内部类和.个Handler接.。Acceptor.于监听请求，AsyncTimeout.于检查异步Request的超时，Handler.于处理接收到的Socket，在内部调.Processor进.处理。 .此，我们应该很轻松的回答（1） （2） （3）的问题了，但是（4）还是不知道，那么我们就来看.下Container是如何进.处理的以及处理完之后是如何将处理完的结果返回给Connector的？关注微信公众号「web_resourc」,回复Java领取2019最新资源。 五、Container架构分析 Container.于封装和管理Servlet，以及具体处理Request请求，在Connector内部包含了4个.容器，结构图如下（图C）： 4个.容器的作.分别是： （1）Engine：引擎，.来管理多个站点，.个Service最多只能有.个Engine； （2）Host：代表.个站点，也可以叫虚拟主机，通过配置Host就可以添加站点； （3）Context：代表.个应.程序，对应着平时开发的.套程序，或者.个WEB-INF.录以及下.的web.xml.件； （4）Wrapper：每.Wrapper封装着.个Servlet； 下.找.个Tomcat的.件.录对照.下，如下图所.： Context和Host的区别是Context表..个应.，我们的Tomcat中默认的配置下webapps下的每.个.件夹.录都是.个Context，其中ROOT.录中存放着主应.，其他.录存放着.应.，.整个webapps就是.个Host站点。 我们访问应.Context的时候，如果是ROOT下的则直接使.域名就可以访问，例如：www.ledouit.com,如果是Host（webapps）下的其他应.，则可以使.www.ledouit.com/docs进.访问，当然默认指定的根应.（ROOT）是可以进.设定的，只不过Host站点下默认的主营.是ROOT.录下的。 看到这.我们知道Container是什么，但是还是不知道Container是如何进.处理的以及处理完之后是如何将处理完的结果返回给Connector的？别急！下边就开始探讨.下Container是如何进.处理的！ 六、Container如何处理请求的 Container处理请求是使.Pipeline-Valve管道来处理的！（Valve是阀.之意） Pipeline-Valve是责任链模式，责任链模式是指在.个请求处理的过程中有很多处理者依次对请求进.处理，每个处理者负责做..相应的处理，处理完之后将处理后的请求返回，再让下.个处理着继续处理。 但是！Pipeline-Valve使.的责任链模式和普通的责任链模式有些不同！区别主要有以下两点： （1）每个Pipeline都有特定的Valve，.且是在管道的最后.个执.，这个Valve叫做BaseValve，BaseValve是不可删除的； （2）在上层容器的管道的BaseValve中会调.下层容器的管道。 我们知道Container包含四个.容器，.这四个.容器对应的BaseValve分别在：StandardEngineValve、StandardHostValve、 StandardContextValve、StandardWrapperValve。 Pipeline的处理流程图如下（图D）： （1）Connector在接收到请求后会.先调.最顶层容器的Pipeline来处理，这.的最顶层容器的Pipeline就是EnginePipeline（Engine的管道）； （2）在Engine的管道中依次会执.EngineValve1、EngineValve2等等，最后会执.StandardEngineValve，在StandardEngineValve中会调.Host管道，然后再依次执.Host的HostValve1、HostValve2等，最后在执.StandardHostValve，然后再依次调.Context的管道和Wrapper的管道，最后执.到StandardWrapperValve。 （3）当执.到StandardWrapperValve的时候，会在StandardWrapperValve中创建FilterChain，并调.其doFilter.法来处理请求，这个FilterChain包含着我们配置的与请求相匹配的Filter和Servlet，其doFilter.法会依次调.所有的Filter的doFilter.法和Servlet的 service.法，这样请求就得到了处理！ （4）当所有的Pipeline-Valve都执.完之后，并且处理完了具体的请求，这个时候就可以将返回的结果交给Connector了，Connector在通过Socket的.式将结果返回给客.端。 七、总结 .此，我们已经对Tomcat的整体架构有了.致的了解，从图A、B、C、D可以看出来每.个组件的基本要素和作.。我们在脑海.应该有.个.概的轮廓了！如果你.试的时候，让你简单的聊.下Tomcat，上.的内容你能脱..出吗？当你能够脱..出的时候，这位.试官.定会对你刮.相看的！ 博客链接：www.blog.csdn.net/xlgen157387 作者：xlgen 如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「进群」即可进...告技术交流。 推荐阅读 1. 史上最烂的项.：苦撑12年，600多万.代码...

2.请给Spr ingBoot多.些内存 3.如何从零搭建百亿流量系统？

4.

惊了！原来Web发展历史是这样的

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

Zookeeper.试23连问，这些你都会吗？ JeffreyLcmJava后端 2.14.

作者|JeffreyLcm segmentfault.com/a/1190000014479433 1.ZooKeeper是什么？ ZooKeeper是.个分布式的，开放源码的分布式应.程序协调服务，是Google的Chubby.个开源的实现，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进.下.步合理操作。最终，将简单易.的接.和性能.效、功能稳定的系统提供给..。 客.端的读请求可以被集群中的任意.台机器处理，如果读请求在节点上注册了监听器，这个监听器也是由所连接的zookeeper机器来处理。对于写请求，这些请求会同时发给其他zookeeper机器并且达成.致后，请求才会返回成功。因此，随着zookeeper的集群机器增多，读请求的吞吐会提.但是写请求的吞吐会下降。 有序性是zookeeper中.常重要的.个特性，所有的更新都是全局有序的，每个更新都有.个唯.的时间戳，这个时间戳称为 zxid（ZookeeperTransactionId）。.读请求只会相对于更新有序，也就是读请求的返回结果中会带有这个zookeeper最新的 zxid。 2.ZooKeeper提供了什么？ 1、.件系统2、通知机制 3.Zookeeper.件系统 Zookeeper提供.个多层级的节点命名空间（节点称为znode）。与.件系统不同的是，这些节点都可以设置关联的数据，..件系统中只有.件节点可以存放数据..录节点不.。Zookeeper为了保证.吞吐和低延迟，在内存中维护了这个树状的.录结构，这种特性使得Zookeeper不能.于存放.量的数据，每个节点的存放数据上限为1M。 4.四种类型的znode 1、PERSISTENT-持久化.录节点 客.端与zookeeper断开连接后，该节点依旧存在 2、PERSISTENT_SEQUENTIAL-持久化顺序编号.录节点 客.端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进.顺序编号 3、EPHEMERAL-临时.录节点 客.端与zookeeper断开连接后，该节点被删除 4、EPHEMERAL_SEQUENTIAL-临时顺序编号.录节点 客.端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进.顺序编号

5.Zookeeper通知机制 client端会对某个znode建..个watcher事件，当该znode发.变化时，这些client会收到zk的通知，然后client可以根据znode变化来做出业务上的改变等。 6.Zookeeper做了什么？ 命名服务配置管理集群管理分布式锁队列管理 7.zk的命名服务（.件系统） 命名服务是指通过指定的名字来获取资源或者服务的地址，利.zk创建.个全局的路径，即是唯.的路径，这个路径就可以作为.个名字，指向集群中的集群，提供的服务的地址，或者.个远程的对象等等。 8.zk的配置管理（.件系统、通知机制） 程序分布式的部署在不同的机器上，将程序的配置信息放在zk的znode下，当有配置发.改变时，也就是znode发.变化时，可以通过改变zk中某个.录节点的内容，利.watcher通知给各个客.端，从.更改配置。 9.Zookeeper集群管理（.件系统、通知机制） 所谓集群管理.在乎两点：是否有机器退出和加.、选举master。对于第.点，所有机器约定在..录下创建临时.录节点，然后监听..录节点的.节点变化消息。.旦有机器挂掉，该机器与zookeeper的连接断开，其所创建的临时.录节点被删除，所有其他机器都收到通知：某个兄弟.录被删除，于是，所有.都知道：它上船了。 新机器加.也是类似，所有机器收到通知：新兄弟.录加.，highcount.有了，对于第.点，我们稍微改变.下，所有机器创建临时顺序编号.录节点，每次选取编号最.的机器作为master就好。 10.Zookeeper分布式锁（.件系统、通知机制） 有了zookeeper的.致性.件系统，锁的问题变得容易。锁服务可以分为两类，.个是保持独占，另.个是控制时序。 对于第.类，我们将zookeeper上的.个znode看作是.把锁，通过createznode的.式来实现。所有客.端都去创建/distribute_lock节点，最终成功创建的那个客.端也即拥有了这把锁。.完删除掉..创建的distribute_lock节点就释放出锁。 对于第.类，/distribute_lock已经预先存在，所有客.端在它下.创建临时顺序编号.录节点，和选master.样，编号最.的获得锁，.完删除，依次.便。 11.获取分布式锁的流程

在获取分布式锁的时候在locker节点下创建临时顺序节点，释放锁的时候删除该临时节点。客.端调.createNode.法在locker下创建临时顺序节点，然后调.getChildren("locker")来获取locker下.的所有.节点，注意此时不.设置任何Watcher。 客.端获取到所有的.节点path之后，如果发现..创建的节点在所有创建的.节点序号最.，那么就认为该客.端获取到了锁。如果发现..创建的节点并.locker所有.节点中最.的，说明..还没有获取到锁，此时客.端需要找到....的那个节点，然后对其调.exist().法，同时对其注册事件监听器。 之后，让这个被关注的节点删除，则客.端的Watcher会收到相应通知，此时再次判断..创建的节点是否是locker.节点中序号最.的，如果是则获取到了锁，如果不是则重复以上步骤继续获取到....的.个节点并注册监听。当前这个过程中还需要许多的逻辑判断。代码的实现主要是基于互斥锁，获取分布式锁的重点逻辑在于BaseDistributedLock，实现了基于Zookeeper实现分布式锁的细节。

12.Zookeeper队列管理（.件系统、通知机制） 两种类型的队列： 同步队列，当.个队列的成员都聚.时，这个队列才可.，否则.直等待所有成员到达。队列按照FIFO.式进..队和出队操作。 第.类，在约定.录下创建临时.录节点，监听节点数.是否是我们要求的数.。 第.类，和分布式锁服务中的控制时序场景基本原理.致，.列有编号，出列按编号。在特定的.录下创建 PERSISTENT_SEQUENTIAL节点，创建成功时Watcher通知等待的队列，队列删除序列号最.的节点.以消费。此场景下Zookeeper的znode.于消息存储，znode存储的数据就是消息队列中的消息内容，SEQUENTIAL序列号就是消息的编号，按序取出即可。由于创建的节点是持久化的，所以不必担.队列消息的丢失问题。 13.Zookeeper数据复制 Zookeeper作为.个集群提供.致的数据服务，.然，它要在所有机器间做数据复制。数据复制的好处： 容错：.个节点出错，不致于让整个系统停..作，别的节点可以接管它的.作；提.系统的扩展能.：把负载分布到多个节点上，或者增加节点来提.系统的负载能.；提.性能：让客.端本地访问就近的节点，提...访问速度。 从客.端读写访问的透明度来看，数据复制集群系统分下.两种： 写主(WriteMaster)：对数据的修改提交给指定的节点。读.此限制，可以读取任何.个节点。

这种情况下客.端需要对读与写进.区别，俗称读写分离；写任意(WriteAny)：对数据的修改可提交给任意的节点，跟读.样。这种情况下，客.端对集群节点的..与变化透明。 对zookeeper来说，它采.的.式是写任意。通过增加机器，它的读吞吐能.和响应能.扩展性.常好，.写，随着机器的增多吞吐能.肯定下降（这也是它建.observer的原因），.响应能.则取决于具体实现.式，是延迟复制保持最终.致性，还是.即复制快速响应。 14.Zookeeper.作原理 Zookeeper的核.是原..播，这个机制保证了各个Server之间的同步。实现这个机制的协议叫做Zab协议。 Zab协议有两种模式，它们分别是恢复模式（选主）和.播模式（同步）。当服务启动或者在领导者崩溃后，Zab就进.了恢复模式，当领导者被选举出来，且.多数Server完成了和leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和Server具有相同的系统状态。 15.zookeeper是如何保证事务的顺序.致性的？ zookeeper采.了递增的事务Id来标识，所有的proposal（提议）都在被提出的时候加上了zxid，zxid实际上是.个64位的数字，.32位是epoch（时期;纪元;世;新时代）.来标识leader是否发.改变，如果有新的leader产.出来，epoch会.增，低32位.来递增计数。当新产.proposal的时候，会依据数据库的两阶段过程，.先会向其他的server发出事务执.请求，如果超过半数的机器都能执.并且能够成功，那么就会开始执..。 16.Zookeeper下Server.作状态 每个Server在.作过程中有三种状态： LOOKING：当前Server不知道leader是谁，正在搜寻 LEADING：当前Server即为选举出来的leader FOLLOWING：leader已经选举出来，当前Server与之同步 17.zookeeper是如何选取主leader的？ 当leader崩溃或者leader失去.多数的follower，这时zk进.恢复模式，恢复模式需要重新选举出.个新的leader，让所有的Server都恢复到.个正确的状态。Zk的选举算法有两种：.种是基于basicpaxos实现的，另外.种是基于fastpaxos算法实现的。系统默认的选举算法为fastpaxos。 1、Zookeeper选主流程(basicpaxos) （1）选举线程由当前Server发起选举的线程担任，其主要功能是对投票结果进.统计，并选出推荐的Server； （2）选举线程.先向所有Server发起.次询问(包括..)； （3）选举线程收到回复后，验证是否是..发起的询问(验证zxid是否.致)，然后获取对.的id(myid)，并存储到当前询问对象列表中，最后获取对.提议的leader相关信息(id,zxid)，并将这些信息存储到当次选举的投票记录表中； （4）收到所有Server回复以后，就计算出zxid最.的那个Server，并将这个Server相关信息设置成下.次要投票的Server； （5）线程将当前zxid最.的Server设置为当前Server要推荐的Leader，如果此时获胜的Server获得n/2+1的Server票数，设置当前推荐的leader为获胜的Server，将根据获胜的Server相关信息设置..的状态，否则，继续这个过程，直到leader被选举出来。通过流程分析我们可以得出：要使Leader获得多数Server的.持，则Server总数必须是奇数2n+1，且存活的Server的数.不得少于n+1.每个Server启动后都会重复以上流程。在恢复模式下，如果是刚从崩溃状态恢复的或者刚启动的server还会从磁盘快照中恢复数据和会话信息，zk会记录事务.志并定期进.快照，.便在恢复时进.状态恢复。

2、Zookeeper选主流程(basicpaxos) fastpaxos流程是在选举过程中，某Server.先向所有Server提议..要成为leader，当其它Server收到提议以后，解决epoch和zxid的冲突，并接受对.的提议，然后向对.发送接受提议完成的消息，重复这个流程，最后.定能选举出Leader。

18.Zookeeper同步流程 选完Leader以后，zk就进.状态同步过程。 Leader等待server连接； Follower连接leader，将最.的zxid发送给leader；Leader根据follower的zxid确定同步点；完成同步后通知follower已经成为uptodate状态；Follower收到uptodate消息后，.可以重新接受client的请求进.服务了。

19.分布式通知和协调 对于系统调度来说：操作.员发送通知实际是通过控制台改变某个节点的状态，然后zk将这些变化发送给注册了这个节点的watcher的所有客.端。 对于执.情况汇报：每个.作进程都在某个.录下创建.个临时节点。并携带.作的进度数据，这样汇总的进程可以监控.录.节点的变化获得.作进度的实时的全局情况。 20.机器中为什么会有leader？ 在分布式环境中，有些业务逻辑只需要集群中的某.台机器进.执.，其他的机器可以共享这个结果，这样可以..减少重复计算，提.性能，于是就需要进.leader选举。 21.zk节点宕机如何处理？ Zookeeper本.也是集群，推荐配置不少于3个服务器。Zookeeper..也要保证当.个节点宕机时，其他节点会继续提供服务。 如果是.个Follower宕机，还有2台服务器提供访问，因为Zookeeper上的数据是有多个副本的，数据并不会丢失； 如果是.个Leader宕机，Zookeeper会选举出新的Leader。 ZK集群的机制是只要超过半数的节点正常，集群就能正常提供服务。只有在ZK节点挂得太多，只剩.半或不到.半节点能.作，集群才失效。所以 3个节点的cluster可以挂掉1个节点(leader可以得到2票>1.5) 2个节点的cluster就不能挂掉任何1个节点了(leader可以得到1票<=1) 22.zookeeper负载均衡和nginx负载均衡区别 zk的负载均衡是可以调控，nginx只是能调权重，其他需要可控的都需要..写插件；但是nginx的吞吐量.zk.很多，应该说按业务选择.哪种.式。 23.zookeeperwatch机制 Watch机制官.声明：.个Watch事件是.个.次性的触发器，当被设置了Watch的数据发.了改变的时候，则服务器将这个改变发送给设置了Watch的客.端，以便通知它们。 Zookeeper机制的特点： 1、.次性触发数据发.改变时，.个watcherevent会被发送到client，但是client只会收到.次这样的信息。 2、watcherevent异步发送watcher的通知事件从server发送到client是异步的，这就存在.个问题，不同的客.端和服务器之间通过socket进.通信，由于.络延迟或其他因素导致客.端在不通的时刻监听到事件，由于Zookeeper本.提供了orderingguarantee，即客.端监听事件后，才会感知它所监视znode发.了变化。所以我们使.Zookeeper不能期望能够监控到节点每次的变化。Zookeeper只能保证最终的.致性，..法保证强.致性。 3、数据监视Zookeeper有数据监视和.数据监视

getdata()andexists()设置数据监视，getchildren()设置了.节点监视。 4、注册watchergetData、exists、getChildren 5、触发watchercreate、delete、setData 6、setData()会触发znode上设置的datawatch（如果set成功的话）。.个成功的create()操作会触发被创建的znode上的数据watch，以及其.节点上的childwatch。..个成功的delete()操作将会同时触发.个znode的datawatch和childwatch（因为这样就没有.节点了），同时也会触发其.节点的childwatch。 7、当.个客.端连接到.个新的服务器上时，watch将会被以任意会话事件触发。当与.个服务器失去连接的时候，是.法接收到watch的。.当client重新连接时，如果需要的话，所有先前注册过的watch，都会被重新注册。通常这是完全透明的。只有在.个特殊情况下，watch可能会丢失：对于.个未创建的znode的existwatch，如果在客.端断开连接期间被创建了，并且随后在客.端连接上之前.删除了，这种情况下，这个watch事件可能会被丢失。 8、Watch是轻量级的，其实就是本地JVM的Callback，服务器端只是存了是否有设置了Watcher的布尔类型 推荐阅读 1.如何轻松阅读GitHub上的项.源码?

2.

IntelliJIDEA新增禅模式和LightEdit模式

3.安利.款 IDEA中强.的代码.成利器 4.如何获取靠谱的新型冠状病毒疫情

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .个蚂蚁程序员，曾经的.酸.试历程 Python农夫Java后端 2019-09-12 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

作者：Python农夫出处：https://www.jianshu.com/p/7882dae7e176 本.就不分享具体的.试题了，主要讲我这.年的.试经历，以及我个.觉得可以从中吸取的经验教训，希望能给在上.学以及毕业3年以内的同学提供.点点帮助。 实习 当年我还很愚昧，根本不知道很多..有实习招聘，直到.三要结束了，学校说:"同学们，你们.四没课，.定要实习阿！"我才反应过来，喔，原来我要去找实习。 .且..也从没规划过什么职业.向。我学的是软件.程，但我当时还真不知道..未来的具体岗位。安卓？IOS？我根本就没学过。算法？学校的acm.赛上都没.过我的名字。C++？..的课程我早忘的烟消雾散了。好像就Java Web还有些印象。 所以我实习时投的.多是java开发。当时学校组织了专.的实习招聘会。我晚上找了个.觉不错的简历模版，在招聘会时投简历。我们的招聘会并没有.线..，.部分都是杭州的.些中.企业和创业公司。我当时还问负责招聘的.师:".师.师，来咱学校的这些公司，有没有特别推荐的啊？".师:".博通不错。"结果这家公司当天根本就没来，我回头.去.博通.站投简历，也是.沉.海，当时觉得很惋惜。有趣的是现在.上.博通的.碑差的.逼。也不知道死没死，得亏当时也没去。招聘会上.搜.也来了，当年.搜.刚从北京来杭州，还是刚创业阶段。我..对.也.较喜爱，家.也有汽.相关的.意。另外. .也喜欢创业公司，.觉加.创业公司，没多久说不定..就是..员.，发家致富，指.可待。所以当时特中意.搜.，于是投了简 历。 我还有好.个好朋友也投了搜.，后来我们也.多都被叫了过去.试。出乎意料的是，就我没过。主要原因是:我Java也忘的差不多了...我对实习.试根本没有任何准备。.以为..了不起，还.个劲的吹.逼。说..奖学.满满当当，绩点也是.列前茅云云。 印象深刻的.点是，.试官问我，你觉得你学习能.怎么样？ 我："我学习能.还是不错的，虽然学.会.作很忙，但依旧能保持很好的成绩。".试官:"我觉得你学习能..般。"然后我就被刷了，.我另外有三个好朋友都通过了，其中有两位同学过去了实习。不过半年多后，这两位也都离开搜.了，主要还是当时他们给的转正.资低。他们.资低也是因为当时搜.融资遇到问题，可以说是 岌岌可危。 搜.现在前端很强.的...原因就是，当时他们后端.了很多....于是前端承担了很多责任，也上了Node，在内部做了很多输出。所以搜.前端.前在业界还是.常不错的。当然最主要还是芋头领导有.啦！还好当时搜.坚持了下来，蚂蚁.服战略投资后，最近.融了钱，是现在.环境不景.中获活的不错的公司了。后来我还.了.家公司，亚信。当时好像也是相对.较.的公司。.试官正好也是浙..毕业，他.看简历，说，哟学弟啊。 我:"学.好，学.好！".试官:"你这个java怎么样呀？"我:"还可以喔。我java web跟j2ee绩点都是4点.的。".试官:"那你说说，java.的这个锁。"我:"唔.....忘记了。" 然后.氛很尴尬，然后不出意外我依旧被刷了。总之，我招聘会上投的公司都挂了，都没要我的。后来机缘巧合，.了.家也是学.创业的公司。他们看我虽然技术不.，但脑.还算灵光，才勉强收留了我。 说了这么多，总结.下，咱们实习或者说校招时要注意的点: 投.技术岗，不要太把..学习成绩当回事。别以为成绩好，.家就.定要你。那些是锦上添花，但跟专业.平并不.定挂钩。尤其是学校就不怎么样。(仅限本科，硕博我没读过，.发.权)别打没有准备的仗！要投什么岗位，就要先复习好对应的专业知识！最好是尽早能有.个专业.向，并做深.学习，别箭在弦上了还不知道射哪.。很多..都有校招跟实习的窗.，.且有固定的时间，别忘记也别错过！尽量还是去..，.创业公司.概率还是死。作为刚.职场的.，去创业公司也很难扮演关键的..，轮不到你发家致富的。 跳槽 我毕业后就留在实习的公司了，所以应届.校招的经验我也没有。 我说下我从第.家公司跳槽到丁.园过程中的.试经历。这个.试的.较多...虽然没太多.货，但是我觉得还是挺有趣的....家就当看段.吧...我离职时是裸辞的，这个也是我太.信犯得错误...我年前离职，元宵后才回杭，然后找.作.花了.个.时间。其中经历了很多挫折。 我在第.家公司是前后端都开发，但才.年多的.作经验，加之..业余不过勤奋，公司.不加班，我在前端上的积累还很薄弱。但我就想做专职做前端，便开始投前端岗位。 下.开启我的疯狂.试过程： 1 年少轻狂，先就找了阿.的同学内推，遗憾的是，..就挂了，.试评价也很不乐观。 2 投了.易，简历没过。投了微博、Segmentfault、酷家乐、红圈营销、蘑菇街等等等，简历没过。第.家公司做微信相关的开发，于是我.信满满的投了有赞、微店、.维.，.觉应该有加分，然..样简历都没过。 3 投了招银.络。笔试过了，第.轮的技术.试，.试官对我评价也不

错。然.终.挂了，终.是hr跟.位男性.试官。欢迎微信搜索Web项.聚集地获取更多博.。 那位男性.试官我不知道他是什么职位，他问了我.个问题:在银.的登录界.上，怎么保证..登录安全性。 我说，在前端上，感觉能做的不多，只知道必须https。 .试官不满意，我.觉也答的不好。 招银的笔试有.件趣事，我在招银有个同学，后来我才知道，当时我笔试的试卷是他出的，但他并不是前端...招银没啥前端，那卷.也是他.上搜了很多题.拼起来的。 4 我投了.家.外包公司，..过了，...板来谈。.板还是挺好的，我讲话那么.傲，他还是挺谦和。他觉得我想去..，想留住我，说：“其实我们也有很多.些..的外包.作，.如阿.。” 我当时由于疑问，问了他.个问题:“为什么像阿.这样的..，根本不缺.才呀，为什么还需要找我们呢？” .板:“这个嘛，有很多原因。.如说，.些..呀，他们有.些项.呢，也是处在试验阶段，不知道成不成。如果招了.去做，万.项.崩了，招的.怎么办？.没得辞退。这就需要我们外包啦。” 当然我来了阿.后，还没接触过这样的外包。我接触的更多的外包是，有些静态...较简单，但总归需要花时间，让公司前端做.有点..材..，于是就找外包的同学。 5 我还投了.家.较.的外包公司，那个业务的项.经理知道我在第.家公司写scala时，就.常的欣赏我，.常想我去。可惜最后我 还是选择了去丁.园。欢迎微信搜索 Web项.聚集地获取更多博.。不过我觉得我还是不值得他这么喜欢，毕竟我scala写的真不怎么样... 6 我还投了同花顺，也顺利拿了o.er。就是o.er.资没到我的期望，感觉跳槽出来，.资没太.变化，总是.有不.，.且同花顺周六还需要加班。 7 .个同学给我内推了曹操专.，..结束后，就跟谈.资期望。我当时说是16k，估计曹操觉得我能.不.以承担这份巨资，没有下.了。 8 我还投了九.，他们的产品是In，.款图.社交软件，在.年前还是挺.的。在17年年初，我感觉也还.。....都过了，三.是前端负责.跟CTO.。CTO很和蔼，但那位前端负责.很严格，不过.得挺帅。他问我vue，angular的.些原理，我当时答的不好，最后挂了。最近听说九.要倒闭了..... 9 我还投了.家叫脸谱中国的，办公地在杭州武林.场坤和中.。办公地极其市中.，hr个个貌美如花。我也拿了o.er，待遇也不错。说实话我还是有点..动的，毕竟hr真的挺好看。 就是我查这个脸谱的流量是真的不咋滴，应该赚不了什么钱，但办公室却这么市中....总感觉怪怪的，不靠谱，最后没接o.er。 然后前.天.个.同学告诉我...脸谱破产了....资都发不出来，还说啥.板跟.员..震、送别墅给.三云云。欢迎微信搜索 Web项.聚集地获取更多博.。 唔，难怪他们hr.姐姐都这么好看... 10 我还投了.家叫彩虹云直播的创业公司，公司.是真没.个，.试官觉得我还是靠谱的，想我来。可是我看这公司.这么少...产品我感觉我不.，是在微信上做直播购物。然后我就坐地起价了，说我想20k，估计他们也开不起，然后就没然后了。不过他们的hr.姐姐也真的很好看，很温柔。再然后，他们也倒闭了，后来我在丁.园时还.试了.个他们那过来的前端....试完后让我拒了... 11 还投了.家创业公司，是.个学妹内推的，叫.球买.。技术.完后，hr.，那个hr.的虽然不错，但说话实在太矫揉造作了...可能不喜欢我，最后也没过。刚看了下豌.荚app下载量，不到5万，估计也是靠着资本勉强活着。 还投了.家也是做医疗的公司，他们的办公地点在杭州市科技局.楼，后来我才知道，.年前丁.园刚来杭州时办公地就在这。杭州市科技局看来是真的很喜欢帮助医疗.业啊！！不过这家医疗公司我感觉不咋滴。我过去.试呢，感觉像是通过.试来解决他们的问题。说他们现在在搞前端.程化，进.到.半 了，然后打开.个.程，问我他们现在这个.程，前端.程化应该怎么做。我.看那.程，head.的script引.了jQuery跟angular...就是他们所说的进.到.半的.程化了。我说如果是我的话...我应该会先把这些代码删了...后来我表达了不满，说问题不应该这么问啦，他就说好吧，那这样吧，如果让你.css画个奥运五环，你怎么实现。我说:“呃，我应该会..张图....”最终也是不欢.散... 最后，说下丁.园的.试经历。丁.园有两.，第..是我.职后的组..试的。.般.试题.部分我也忘记了，反正经历了上述..家，我已经应答如流了。。。。唯 .有印象的是，当时他问我在.的前后端耦合的.程上怎么做前后端分离的。我说了上.家公司的.案，他哈哈哈哈哈.笑，说丁.园也是这么做的。欢迎微信搜索 Web项.聚集地获取更多博.。总之谈的挺开.，然后..是我当时的前端主管，跟我聊的主要不是技术.向了。.些职业规划、成..标云云。最后的最后，我选择了丁.园。从上.的.试经历.家也能知道当时选择丁.园的原因了。更好的平台如.易阿.我.去不了。其他要么觉得我.资要求太.，要么我觉得他们略不靠谱。幸亏我最后的选择是正确的。丁.园这家公司从.职到我离开，.乎没有什么让我难受的地.。公司发展也挺好，对社会也有价值。如果有同学想对丁.园有更多的了解，也可以来问我。 .结最后再总结.下前端跳槽注意的.个点： 尽量别裸辞，不然..很被动。能.不.，运.也不.的话，喜欢的公司进不去，不喜欢的公司不想去但.觉得...直闲着也不.，就很尴尬。注意社保，别断缴！如果实在不.，可以去淘宝买。如果真的断了再去买补缴，记得多花点钱，买会缴个税的。别问我为什么说这个..... 都是泪。最想去的公司后.再.，前.先攒.点.经。没有较.的把握谨慎来阿..试，想攒阿.的.经去看.上其他.的.试经验吧。.了结果评价很差也不是好事。..能.还不强时，平台很重要。看我曾经.的那么多公司，.部分死的死伤的伤，能去更好的公司就去更好的公司。..公司经验 对履历加分也是很多很多的。学历与公司经历很重要。虽然这不.定会成为.试的.槛，但是.家筛简历时，很可能就不愿意看你。 .职蚂蚁 后来我再从丁.园离职时，就仅.了蚂蚁，当时也是..插柳，抱着试试，没想到也就进了。如果各位也有这样的想法，不如发送个简历给我。说不定也万.就.....于.进蚂蚁的经验（针对社招P6岗），细节我也不.便说，我说两点我个.觉得.较重要的： 专业技术能.必须过关。什么学习能.强、努.刻苦之类的，在社招上，并不是决定因素。要有.定的推动.、执...什么顺利的完成.作，线上.bug之类的都是最基本的要求。这要是有bug还能写上简历吗？需要的是..去推动落实的.些地.，.如性能优化、开发效率提升之类。 有同学问，蚂蚁各轮.试的侧重点，我说.下我..的经历。不过每个bu的招聘流程可能会有些差异，我是蚂蚁保险事业群，我说的仅是当时的情况： .. ..问的js/css/html基础知识会偏多。虽然我当时主要写的还是vue,但由于蚂蚁是react的技术栈，还是问了我不少关于react..的知识。我个.感觉我答的并不是.常理想，但估计由于我个.对react使.不多，.试官还是给了我.轮.试的机会。 .. ..电话打来时，我当时正好在参加朋友婚礼，中途去接了.试电话。相对..，..基础知识问的相对偏少.丢丢，但总体来说，还是那些题.，.家.上都能搜的到，真的。但是不代表.上

能搜得到的东西，..就能答得好。你的答案是.上看的，还是..真正的理解，这些.听都能感受的到的。 所以同学们在.上搜.试题时，.定要有..的理解。对于那些.试题.定要深.研究其背后原理，.试回答时才能侃侃.谈。 三. 我的三.依旧还是那些问题。...是要招聘的部.的.的.试。三四.是各个部.交叉.。为了就是保证.试流程的公平公正，为了蚂蚁员.的保质保量。所以不要指望说..认识部.的什么.，..的.试就能容易.些。 四. 我的四..试官主要问了我个.觉得..做的最好的项.，以及..在其中发挥的最重要的作.，在其中遇到的问题，..是如何解决的。 还有因为我.过node以及egg，问了.些我对node使..得，对egg的看法。以及对阿.的其他开源项.的使.经验与个..解。 主要还是考查专业技术上的深.思考以及在技术之上的.些能.。这轮.试需要.试者有.定的项.沉淀与专业深度，不是简简单单能答上问题就.的。因为实际.作中会遇到很多千奇百怪的问题，这不是.些.试题能够覆盖的。如果你说不出..的.些问题，那说明是真的项.经验.较少，或者说..的思考与沉淀.较少。 不过.部分.遇到的问题其实都很简单，什么如何抽象组件、某某兼容性之类的，都是.些点上.的问题。阿.p6的要求是独挡..，可以负责.条业务线上的所有事情。所以需要.家将点扩散到.，.如项.整体.程架构、研发流程与效率等.些问题。 五、六. 后.就是技术部的负责.与hr的.试。考差的就不是技术相关的了，主要还是考察.本.。思想情况、价值观、对阿.的看法、对个. 发展的想法等等。这个.家如实说就好，坚持本.。 如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「进群」即可进...告技术交流。 推荐阅读 1. Java实现QQ登陆

2.请给Spr ingBoot多.些内存 3.如何从零搭建百亿流量系统?

4.

惊了！原来Web发展历史是这样的

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

不敢相信？System.currentT imeMillis()居然存在性能问题 LittleMagicJava后端 2019-09-09 System.currentTimeMillis()是极其常.的基础Java API，.泛地.来获取时间戳或测量代码执.时.等，在我们的印象中应该快如闪 电。但实际上在并发调.或者特别频繁调.它的情况下（.如.个业务繁忙的接.，或者吞吐量.的需要取得时间戳的流式程序），其性能 表现会令..跌眼镜。 直接看代码： 1 public class CurrentTimeMillisPerfDemo { 2 private static final int COUNT = 100 3 ; 4 5 public static void main(String[] args) throws Exception 6 { 7 long beginTime = System.nanoTime(); 8 for (int i = 0; i < COUNT; i++) 9 { 10 System.currentTimeMillis(); 11 } 12 13 long elapsedTime = System.nanoTime() -beginTime; 14 System.out.println("100 System.currentTimeMillis() serial calls: " + elapsedTime + " ns") 15 ; 16 17 CountDownLatch startLatch = new CountDownLatch(1) 18 ; 19 CountDownLatch endLatch = new CountDownLatch(COUNT); 20 for (int i = 0; i < COUNT; i++) 21 { 22 new Thread(() -> 23 { 24 try 25 { 26 startLatch.await() 27 ; 28 System.currentTimeMillis(); 29 } catch (InterruptedException e) { 30 e.printStackTrace(); 31 } finally 32 { 33 endLatch.countDown();

34 }

}).start(); } beginTime = System.nanoTime(); startLatch.countDown(); endLatch.await() ; elapsedTime = System.nanoTime() -beginTime; System.out.println("100 System.currentTimeMillis() parallel calls: " + elapsedTime + " ns"

执.结果如下图。

可.，并发调. System.currentTimeMillis().百次，耗费的时间是单线程调..百次的250倍。如果单线程的调.频次增加（.如达到每毫秒数次的地步），也会观察到类似的情况。实际上在极端情况下，System.currentTimeMillis()的耗时甚.会.创建.个简单的对象实例还要多，看官可以..将上.线程中的语句换成newHashMap<>之类的试试看。 为什么会这样？ 来到HotSpot源码的hotspot/src/os/linux/vm/os_linux.cpp.件中，有.个javaTimeMillis().法，这就是System.currentTimeMillis()的 native实现。 1 jlong os::javaTimeMillis() { 2 timeval time; 3 int status = gettimeofday(&time, NULL) 4 ; 5 assert(status != -1, "linux error") 6 ; return jlong(time.tv_sec) * 1000 + jlong(time.tv_usec / 1000) ; } 挖源码就到此为.，因为已经有国外.佬深.到了汇编的级别来探究，详情可以参.《TheSlowcurrentTimeMillis()》这篇.章。简单来讲 就是： 调.gettimeofday()需要从..态切换到内核态； gettimeofday()的表现受Linux系统的计时器（时钟源）影响，在HPET计时器下性能尤其差； 系统只有.个全局时钟源，.并发或频繁访问会造成严重的争.。 HPET计时器性能较差的原因是会将所有对时间戳的请求串.执.。TSC计时器性能较好，因为有专.的寄存器来保存时间戳。缺点是可能不 稳定，因为它是纯硬件的计时器，频率可变（与处理器的CLK信号有关）。关于HPET和TSC的细节可以参. https://en.wikipedia.org/wiki/HighPrecisionEventTimer与 https://en.wikipedia.org/wiki/TimeStamp_Counter。 另外，可以.以下的命令查看和修改时钟源。

如何解决这个问题？ 最常.的办法是.单个调度线程来按毫秒更新时间戳，相当于维护.个全局缓存。其他线程取时间戳时相当于从内存取，不会再造成时钟资源的争.，代价就是牺牲了.些精确度。具体代码如下。 1 public class CurrentTimeMillisClock { 2 private volatile long now; 3 4 private CurrentTimeMillisClock() 5 { 6 this.now = System.currentTimeMillis(); 7 scheduleTick(); 8 } 9 10 private void scheduleTick() 11 { 12 new ScheduledThreadPoolExecutor(1, runnable -> { 13 Thread thread = new Thread(runnable, "current-time-millis") 14 ; 15 thread.setDaemon(true) 16 ; 17 return thread 18 ; 19 }).scheduleAtFixedRate(() -> { 20 now = System.currentTimeMillis(); 21 }, 1, 1, TimeUnit.MILLISECONDS); 22 } 23 24 public long now() 25 { 26 return now 27 ;

28 } 29 30 public static CurrentTimeMillisClock getInstance()

{ return SingletonHolder.INSTANCE; } private static class SingletonHolder { private static final CurrentTimeMillisClock INSTANCE = new CurrentTimeMillisClock(); } } 使.的时候，直接 CurrentTimeMillisClock.getInstance().now()就可以了。不过，在System.currentTimeMillis()的效率没有影响程序整体的效率时，就完全没有必要做这种优化，这只是为极端情况准备的。

来源：www.jianshu.com/p/d2039190b1cb 作者：LittleMagic 如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「进群」即可进...告技术交流。 推荐阅读 1.基于SpringBoot的Restful.格实现增删改查

2.

如何使..逼的插件帮你规范代码

3.

IntelliJIDEA构建maven多模块.程项.

4.

别在Java代码.乱打.志了，这才是正确姿势 5.挑战10道超难 Java.试题

6.

什么时候进.分库分表?

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

什么是Java对象深拷.？.试必问！ 吴..Java后端 2019-11-08 点击上.Java后端，选择设为星标 技术博.，及时送达

作者|吴..编辑|Java基基链接|wudashan.com/2018/10/14/Java-Deep-Copy 介绍 在Java语..，当我们需要拷..个对象时，有两种类型的拷.：浅拷.与深拷.。浅拷.只是拷.了源对象的地址，所以源对象的值发.变化时，拷.对象的值也会发.变化。.深拷.则是拷.了源对象的所有值，所以即使源对象的值发.变化时，拷.对象的值也不会改变。如下图描述：

了解了浅拷.和深拷.的区别之后，本篇博客将教.家.种深拷.的.法。 拷.对象 .先，我们定义.下需要拷.的简单对象。 /***.. */ publicclassUser{ privateStringname;privateAddressaddress; //constructors,gettersandsetters } /***地址 / *publicclassAddress{ privateStringcity;privateStringcountry; //constructors,gettersandsetters } 如上述代码，我们定义了.个User..类，包含name姓名，和address地址，其中address并不是字符串，.是另.个Address类，包含country国家和city城市。构造.法和成员变量的get()、set().法此处我们省略不写。接下来我们将详细描述如何深拷.User对象。 .法.构造函数我们可以通过在调.构造函数进.深拷.，形参如果是基本类型和字符串则直接赋值，如果是对象则重新new.个。 测试.例 @Test publicvoidconstructorCopy(){ Addressaddress=newAddress("杭州","中国"); Useruser=newUser("..",address); //调.构造函数时进.深拷. UsercopyUser=newUser(user.getName(),newAddress(address.getCity(),address.getCountry())); //修改源对象的值 user.getAddress().setCity("深圳"); //检查两个对象的值不同 assertNotSame(user.getAddress().getCity(),copyUser.getAddress().getCity()); } .法.重载clone().法 Object.类有个clone()的拷..法，不过它是protected类型的，我们需要重写它并修改为public类型。除此之外，.类还需要实现Cloneable接.来告诉JVM这个类是可以拷.的。 Tips：关注微信公众号：Java后端，获取每.技术博.推送。 重写代码 让我们修改.下User类，Address类，实现Cloneable接.，使其.持深拷.。 /**地址 / publicclassAddressimplementsCloneable{ privateStringcity;privateStringcountry; //constructors,gettersandsetters @Override*

*publicAddressclone()throwsCloneNotSupportedException{return(Address)super.clone();} } /.. / publicclassUserimplementsCloneable{ privateStringname;privateAddressaddress; //constructors,gettersandsetters @Override publicUserclone()throwsCloneNotSupportedException{Useruser= (User)super.clone();user.setAddress(this.address.clone());returnuser; } } 需要注意的是，super.clone()其实是浅拷.，所以在重写User类的clone().法时，address对象需要调.address.clone()重新赋值。 测试.例 @Test publicvoidcloneCopy()throwsCloneNotSupportedException{ Addressaddress=newAddress("杭州","中国");Useruser=newUser("..",address); //调.clone().法进.深拷. UsercopyUser=user.clone(); //修改源对象的值 user.getAddress().setCity("深圳"); //检查两个对象的值不同 assertNotSame(user.getAddress().getCity(),copyUser.getAddress().getCity()); } .法三ApacheCommonsLang序列化 Java提供了序列化的能，我们可以先将源对象进.序列化，再反序列化.成拷.对象。但是，使.序列化的前提是拷.的类（包括其成员变量）需要实现Serializable接。。ApacheCommonsLang包对Java序列化进.了封装，我们可以直接使.它。 重写代码 让我们修改.下User类，Address类，实现Serializable接，使其.持序列化。 /**地址 / publicclassAddressimplementsSerializable{ privateStringcity;privateStringcountry; //constructors,gettersandsetters } /.. / publicclassUserimplementsSerializable{ privateStringname;privateAddressaddress; //constructors,gettersandsetters } 测试.例 @Test publicvoidserializableCopy(){ Addressaddress=newAddress("杭州","中国");Useruser=newUser("..",address); //使.ApacheCommonsLang序列化进.深拷. UsercopyUser=(User)SerializationUtils.clone(user); //修改源对象的值 user.getAddress().setCity("深圳"); //检查两个对象的值不同 assertNotSame(user.getAddress().getCity(),copyUser.getAddress().getCity()); } .法四Gson序列化 Gson可以将对象序列化成JSON，也可以将JSON反序列化成对象，所以我们可以.它进.深拷.。 测试.例 @Test publicvoidgsonCopy(){ Addressaddress=newAddress("杭州","中国");Useruser=newUser("..",address); //使.Gson序列化进.深拷. Gsongson=newGson();UsercopyUser=gson.fromJson(gson.toJson(user),User.class); //修改源对象的值 user.getAddress().setCity("深圳"); //检查两个对象的值不同 assertNotSame(user.getAddress().getCity(),copyUser.getAddress().getCity()); } .法五Jackson序列化 Jackson与Gson相似，可以将对象序列化成JSON，明显不同的地.是拷.的类（包括其成员变量）需要有默认的.参构造函数。 重写代码 让我们修改.下User类，Address类，实现默认的.参构造函数，使其.持Jackson。 /.. */* publicclassUser{ privateStringname;privateAddressaddress; //constructors,gettersandsetters publicUser(){} } /***地址 */ publicclassAddress{ privateStringcity;privateStringcountry; //constructors,gettersandsetters publicAddress(){} } 测试.例 @Test publicvoidjacksonCopy()throwsIOException{ Addressaddress=newAddress("杭州","中国");Useruser=newUser("..",address); //使.Jackson序列化进.深拷. ObjectMapperobjectMapper=newObjectMapper();UsercopyUser=objectMapper.readValue(objectMapper.writeValueAsString(user),User.class); //修改源对象的值 user.getAddress().setCity("深圳"); //检查两个对象的值不同 assertNotSame(user.getAddress().getCity(),copyUser.getAddress().getCity()); } 总结 说了这么多深拷.的实现.法，哪.种.法才是最好的呢？最简单的判断就是根据拷.的类（包括其成员变量）是否提供了深拷.的构造函数、是否实现了Cloneable接.、是否实现了Serializable接.、是否实现了默认的.参构造函数来进.选择。如果需要详细的考虑，则可以参考下.的表格：

-END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.你写的 Java代码是如何.步步输出结果的?

2. IntelliJ IDEA详细图解最常.的配置，新.收藏 3.Maven实战问题和最佳实践

3.

12306的架构到底有多.逼?

5.团队开发中 Git最佳实践

学Java，请关注公众号：Java后端 喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

什么是.致性Hash算法？.试.被问到怎么办？ Java后端 2019-12-13以下.章来源于会点代码的.叔，作者会点代码的.叔

会点代码的.叔 程序员.叔，擅编码，懂调优，会架构，能讲段.，喜欢...话讲解复杂的技术。 01 数据分. ..先让我们看.个例.吧 我们经常会.Redis做缓存，把.些数据放在上.，以减少数据的压.。当数据量少，访问压.不.的时候，通常.台Redis就能搞定，为了.可.，弄个主从也就.够了；当数据量变.，并发量也增加的时候，把全部的缓存数据放在.台机器上就有些吃.了，毕竟.台机器的资源是有限 的，通常我们会搭建集群环境，让数据尽量平均的放到每.台Redis中，.如我们的集群中有4台Redis。那么如何把数据尽量平均地放到这4台Redis中呢？最简单的就是取模算法：hash(key)%N，N为Redis的数量，在这.N=4;

看起来.常得美好，因为依靠这样的.法，我们可以让数据平均存储到 4台 Redis中，当有新的请求过来的时候，我们也可以定位数据会在哪台Redis中，这样可以精确地查询到缓存数据。

02 数据分.会遇到的问题 但是4台Redis不够了，需要再增加4台Redis；那么这个求余算法就会变成：hash(key)%8；

那么可以想象.下，当前.部分缓存的位置都会是错误的，极端情况下，就会造成缓存雪崩。

03 .致性 Hash算法 .致性Hash算法可以很好地解决这个问题，它的.概过程是这样的： 把 0作为起点，2^32-1作为终点，画.条直线，再把起点和终点重合，直线变成.个圆，.向是顺时针从.到..。0的右侧第.个点是1，然后是2，以此类推。 对三台服务器的 IP或其他关键字进. hash后对 2^32取模，这样势必能落在这个圈上的某个位置，记为 Node1、Node2、Node3。

然后对数据 key进.相同的操作，势必也会落在圈上的某个位置；然后顺时针..，可以找到某.个 Node，这就是这个key要储存的服务器。

如果增加.台服务器或者删除.台服务器，只会影响部分数据。

但如果节点太少或分布不均匀的时候，容易造成数据倾斜，也就是.部分数据会集中在某.台服务器上。

为了解决数据倾斜问题，.致性 Hash算法提出了【虚拟节点】，会对每.个服务节点计算多个哈希，然后放到圈上的不同位置。

当然我们也可以发现，.致性Hash算法，也只是解决.部分数据的问题。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.我把废旧 Android .机改造成了 Linux服务器

2.动画: .个浏览器是如何.作的？ 3.为什么你学不会递归?

4.

.个..不主动联系你还有机会吗?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

从..到熟悉HTTPS的9个问题 bestswifterJava后端 1.8.

作者|bestswifter链接|juejin.im/post/58c5268a61ff4b005d99652a Q1:什么是HTTPS？BS:HTTPS是安全的HTTP HTTP协议中的内容都是明.传输，HTTPS的.的是将这些内容加密，确保信息传输安全。最后.个字.S指的是 SSL/TLS协议，它位于HTTP协议与TCP/IP协议中间。 Q2:你说的信息传输安全是什么意思BS:信息传输的安全有四个..: 1、客.端和服务器直接的通信只2、有..能看懂，即使第三.拿到数据也看不懂这些信息的真实含义。3、第三.虽然看不懂数据，但可以XJB改，因此客.端和服务器必须有能.判断数据是否被修改过。4、客.端必须避免中间.攻击，即除了真正的服务器，任何第三.都.法冒充服务器。 很遗憾的是，.前的HTTP协议还不满.上述三条要求中的任何.条。 Q3:这么多要求，.个.个去满.是不是很累？BS:不累，第三个要求可以不.管 是的，我没开玩笑，你可以暂时别管第三个要求，因为它实际上.属于第.个需求。我们都知道加密需要密码，密码不是天下掉下来，也得需要双.经过通信才能协商出来。所以.个设计良好的加密机制必然会防.第三者的.扰和伪造。等搞明.了加密的具体原理，我们.然可以检验是否满.:"任何第三者.法冒充服务器"这.要求。 Q4:那怎么加密信息呢BS:使.对称加密技术 对称加密可以理解为对原始数据的可逆变换。.如 Hello可以变换成Ifmmp，规则就是每个字.变成它在字.表上的后.个字.，这.的秘钥就是1，另..拿到Ifmmp就可以还原成原来的信息Hello

了。引.对称加密后，HTTPS的握.流程就会多了两步，.来传递对称加密的秘钥: 1、客.端:你好，我需要发起.个HTTPS请求1、服务器:好的，你的秘钥是1。 提到了对称加密，那么.然还有.对称加密。它的思想很简单，计算两个质数的乘积很容易，但反过来分解成两个质数的乘积就很难，要经过极为复杂的运算。.对称加密有两个秘钥，.个是公钥，.个是私钥。公钥加密的内容只有私钥可以解密，私钥加密的内容只有公钥可以解密。.般我们把服务器..留着，不对外公布的密钥称为私钥，所有.都可以获取的称为公钥。点击查看图解https的认证过程详情。 使.对称加密.般要..对称加密快得多，对服务器的运算压.也.得多。 Q5:对称秘钥如何传输 服务器直接返回明.的对称加密密钥是不是不安全。如果有监听者拿到这个密钥，不就知道客.端和服务器后续的通信内容了么？ BS:利..对称加密 是这样，所以不能明.传递对称秘钥，.且也不能..个新的对称加密算法来加密原来的对称秘钥，否则新的对称秘钥同样.法传输，这就是鸡.蛋、蛋.鸡的悖论。 这.我们引..对称加密的.式，.对称加密的特性决定了服务器.私钥加密的内容并不是真正的加密，因为公钥所有.都有，所以服务器的密.能被所有.解析。但私钥只掌握在服务器.上，这就带来了两个巨.的优势: 1、服务器下发的内容不可能被伪造，因为别.都没有私钥，所以.法加密。强.加密的后果是客.端.公钥.法解开。 2、任何..公钥加密的内容都是绝对安全的，因为私钥只有服务器有，也就是只有真正的服务器可以看到被加密的原.。 所以传输对称秘钥的问题就迎刃.解了: 秘钥不是由服务器下发，.是由客.端.成并且主动告诉服务器。所以当引..对称加密后，HTTPS的握.流程依然是两步，不过细节略有变化: 客.端:你好，我需要发起.个HTTPS请求，这是我的(.公钥加密后的)秘钥。服务器:好的，我知道你的秘钥了，后续就.它传输。 Q5:那公钥怎么传输 你好像还是没有解决鸡.蛋，蛋.鸡的问题。你说客.端发送请求时要.公钥加密对称秘钥，那公钥怎么传输呢？ BS:对公钥加密就.了。。。 每.个使. HTTPS的服务器都必须去专.的证书机构注册.个证书，证书中存储了.权威机构私钥加密的公钥。这样客.端.权威机构的公钥解密就可以了。 现在HTTPS协议的握.阶段变成了四步: 1、客.端:你好，我要发起.个HTTPS请求，请给我公钥2、服务器:好的，这是我的证书，..有加密后的公钥3、客.端:解密成功!以后告诉服务器:这是我的(.公钥加密后的)对称秘钥。4、服务器:好的，我知道你的秘钥了，后续就.它传输。 Q6:你在逗我么。。。。 那权威机构的公钥.怎么传输？ BS:存在电脑. 这个公钥不.传输，会直接内置在各.操作系统(或者浏览器)的出.设置.。之所以不把每个服务器的公钥内置在电脑.，...是因为服务器太多，存不过来。另...操作系统也不信任你，凭什么你说你这个就是百度/淘宝的证书呢？ 所以各个公司要先去权威机构认证，申请证书，然后操作系统只会存储权威机构的公钥。因为权威机构数量有限，所以操作系统.商相对来说容易管理。如果这个权威机构不够权威，XJB发证书，就会取消他的资格，.如可怜的沃通。。。。 Q7:怎么知道证书有没有被篡改？ 你说服务器第.次会返回证书，也就是加密以后的公钥，那我怎么知道这个证书是可靠的？ BS:将信息hash值随着信息.起传递 我们都知道哈希算法的特点，它可以压缩数据，如果从函数.度来看，不管多复杂的数据(定义域可以.常.)经过哈希算法都会得到.个值，.且这个值处在某个特定(远.于定义域的范围)值域内。相同数据的哈希结果.定相同，不相同数据的哈希结果.般不同，不过也有.概率会重复，这叫哈希冲突。 为了确保原始证书没有被篡改，我们可以在传递证书的同时传递证书的哈希值。由于第三者.法解析数据，只能XJB改，那么修改后的数据在解密后，就不可能通过哈希。 .如说公钥就是之前的例.Hello，我们假设哈希算法是获取字符串的最后.个字符，那么Hello的哈希值就是o，所以加密字符串是 lfmmpp。虽然公钥已知，每个.都可以解密，解密完也可以篡改，但是因为没有私钥，所以.法正确的加密。所以它再返回给客.端的数据是.效数据，.公钥解析后会得到乱码。即使攻击者通过多次尝试碰巧能够解析，也.法通过哈希校验。 Q8:这样可以防.第三.冒充服务器么BS:也许可以 .先真正的服务器下发的内容，.法被别.篡改。他们有权威机构的公钥，所以可以解密，但是因为没有私钥，所以解密以后的信息.法加密。没有加密或者错误加密的信息被客.端.公钥解密以后，必然.法通过哈希校验。点击查看图解https的认证过程详情。 但是，如果你.开始请求的就不是真的服务器，.是.个攻击者，此时的他完全有机会进.中间.攻击。我们知道第.次握.的时候服务器会下发.于证明...份的证书，这个证书会.预设在设备上的公钥来解密。所以要么是经过认证的证书.权威机构的私钥加密，再.权威机构解密，要么是..权威机构的私钥加密，然后找不到公钥解密。 所以如果不..安装过.权威机构的根证书，.如.客提供的恶意证书，这时候设备上就多了.个预设的公钥，那么.恶意私钥加密的证书就能被正常解析出来。所以千万不要随便装根证书，这等于是为那些恶意证书留了.扇.。当然，凡是都有两.性。我们知道 Charles可以调试HTTPS通信，它的原理就是需要..安装Charles的根证书，然后我们的请求会被代理到Charles服务器，它下发的Charles证书才能被正确解析。另...，Charles会作为客.端，从真正的服务器哪.拿到正确的 https证书并.于后续通信。幸好 Charles不是流氓软件，或者它的私钥.旦泄露，对..都会造成很.的影响。点击查看图解https的认证过程详情。 我可以举.个例.，证书有多个种类，最贵的叫 EV (Extended Validation)，它需要公司营业执照等多个.件才能申请..审核，好处也很明显，可以在浏览器地址栏左侧准确显.公司名称，.如Bitbucket的官.:

代理模式下.法显.Q9:HTTPS握.会影响性能么 TCP有三次握.，再加上HTTPS的四次握.，会不会影响性能？ BS:影响肯定有，但是可以接受 .先，HTTPS肯定会更慢.点，时间主要花费在两组 SSL之间的耗时和证书的读取验证上，对称算法的加解密时间.乎可以忽略不计。 .且如果不是.次握.，后续的请求并不需要完整的握.过程。客.端可以把上次的加密情况直接发送给服务器从.快速恢复，具体细节可以点击查看图解https的认证过程详情。。 除此以外，SSL握.的时间并不是只能.来传递加密信息，还可以承担起客.端和服务器沟通 HTTP2兼容情况的任务。因此从HTTPS切换到HTTP2.0不会有任何性能上的开销，反倒是得益于HTTP2.0的多路复.等技术，后续可以节约.量时间。 如果把HTTPS2.0当做.标，那么HTTPS的性能损耗就更.了，远远.不上它带来的安全性提升。 结语 相信以上九个问题.够帮助新.了解HTTPS了，但这只是基本概念，关于HTTPS的使.(.如iOS上的.些具体问题)还需要不断尝试和研究。 -END - 推荐阅读 1. 实现接.幂等性校验

2.全.了解 Nginx主要应.场景

3.

Github标星10.8K！Java实战博客项.分享

4.什么是.致性 Hash算法?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

使.ThreadLocal.次解决..难问题 鲁毅Java后端 1.11.

作者|鲁毅链接|juejin.im/post/5e0d8765f265da5d332cde44参考|https://coding.imooc.com/class/409.html
1.ThreadLocal的使.场景 1.1场景1 每个线程需要.个独享对象（通常是.具类，典型需要使.的类有SimpleDateFormat和Random）每个Thread内有..的实例副本，不共享.喻：教材只有.本，.起做笔记有线程安全问题。复印后没有问题，使.ThradLocal相当于复印了教材。 1.2场景2 每个线程内需要保存全局变量（例如在拦截器中获取..信息），可以让不同.法直接使.，避免参数传递的.烦 2.对以上场景的实践 2.1实践场景1 /***两个线程打印.期 */ *publicclassThreadLocalNormalUsage00{ publicstaticvoidmain(String[]args)throwsInterruptedException{ newThread(newRunnable(){@Override publicvoidrun(){ Stringdate=newThreadLocalNormalUsage00().date(10);System.out.println(date);}}).start(); newThread(newRunnable(){@Override publicvoidrun(){ Stringdate=newThreadLocalNormalUsage00().date(104707);System.out.println(date);}}).start(); } publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000*seconds);SimpleDateFormatdateFormat=newSimpleDateFormat("yyyy-MM-ddhh:mm:ss"); returndateFormat.format(date); }} 运.结果

因为中国位于东.区，所以时间从1970年1.1.的8点开始计算的 /** *三.个线程打印.期 */ *publicclassThreadLocalNormalUsage01{ publicstaticvoidmain(String[]args)throwsInterruptedException{ for(inti=0;i<30;i++){intfinalI=i; newThread(newRunnable(){ @Override publicvoidrun(){Stringdate=newThreadLocalNormalUsage01().date(finalI);System.out.println(date); }}).start(); //线程启动后，休眠100ms Thread.sleep(100);}} publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000*seconds);SimpleDateFormatdateFormat=newSimpleDateFormat("yyyy-MM-ddhh:mm:ss"); returndateFormat.format(date);} } 运.结果

多个线程打印..的时间（如果线程超级多就会产.性能问题），所以要使.线程池。 /***1000个线程打印.期，.线程池来执. */ *publicclassThreadLocalNormalUsage02{ publicstaticExecutorServicethreadPool=Executors.newFixedThreadPool(10); publicstaticvoidmain(String[]args)throwsInterruptedException{for(inti=0;i<1000;i++){intfinalI=i; //提交任务 threadPool.submit(newRunnable(){@Override publicvoidrun(){ Stringdate=newThreadLocalNormalUsage02().date(finalI);System.out.println(date);} });}threadPool.shutdown(); } publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000*seconds);SimpleDateFormatdateFormat=newSimpleDateFormat("yyyy-MM-ddhh:mm:ss"); returndateFormat.format(date); }} 运.结果

但是使.线程池时就会发现每个线程都有.个..的SimpleDateFormat对象，没有必要，所以将SimpleDateFormat声明为静态，保证只有.个 /***1000个线程打印.期，.线程池来执.，出现线程安全问题 */ *publicclassThreadLocalNormalUsage03{ publicstaticExecutorServicethreadPool=Executors.newFixedThreadPool(10); //只创建.次SimpleDateFormat对象，避免不必要的资源消耗 staticSimpleDateFormatdateFormat=newSimpleDateFormat("yyyy-MM-ddhh:mm:ss"); publicstaticvoidmain(String[]args)throwsInterruptedException{for(inti=0;i<1000;i++){intfinalI=i; //提交任务 threadPool.submit(newRunnable(){@Override publicvoidrun(){ Stringdate=newThreadLocalNormalUsage03().date(finalI);System.out.println(date);} });}threadPool.shutdown(); } publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000*seconds); returndateFormat.format(date);}} 运.结果 出现了秒数相同的打印结果，这显然是不正确的。

出现问题的原因

多个线程的task指向了同.个SimpleDateFormat对象，SimpleDateFormat是.线程安全的。 TIps：关注公众号：Java后端，每.推送技术博.。 解决问题的.案 .案1：加锁 格式化代码是在最后.句 returndateFormat.format(date);,所以可以为最后.句代码添加synchronized锁 publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000*seconds);Strings;synchronized(ThreadLocalNormalUsage04.class){ s=dateFormat.format(date);}returns; } 运.结果运.结果中没有发现相同的时间，达到了线程安全的.的

缺点：因为添加了synchronized，所以会保证同.时间只有.条线程可以执.，这在.并发场景下肯定不是.个好的选择，所以看看其他.案吧。 .案2：使.ThreadLocal /***利.ThreadLocal给每个线程分配..的dateFormat对象 *不但保证了线程安全，还.效的利.了内存 */ publicclassThreadLocalNormalUsage05{ publicstaticExecutorServicethreadPool=Executors.newFixedThreadPool(10); publicstaticvoidmain(String[]args)throwsInterruptedException{for(inti=0;i<1000;i++){intfinalI=i; //提交任务 threadPool.submit(newRunnable(){@Override publicvoidrun(){ Stringdate=newThreadLocalNormalUsage05().date(finalI);System.out.println(date);} });}threadPool.shutdown(); } publicStringdate(intseconds){ //参数的单位是毫秒，从1970.1.100:00:00GMT开始计时 Datedate=newDate(1000seconds); //获取SimpleDateFormat对象 SimpleDateFormatdateFormat=ThreadSafeFormatter.dateFormatThreadLocal.get(); returndateFormat.format(date);}} classThreadSafeFormatter{ publicstaticThreadLocaldateFormatThreadLocal=new ThreadLocal(){ //创建.份SimpleDateFormat对象 @Override protectedSimpleDateFormatinitialValue(){returnnewSimpleDateFormat("yyyy-MM-ddhh:mm:ss");}};} 运.结果

使.了ThreadLocal后不同的线程不会有共享的SimpleDateFormat对象，所以也就不会有线程安全问题 2.2实践场景2 当前..信息需要被线程内的所有.法共享 .案1：传递参数

可以将user作为参数在每个.法中进.传递，缺点：但是这样做会产.代码冗余问题，并且可维护性差。 .案2：使.Map 对此进.改进的.案是使..个Map，在第.个.法中存储信息，后续需要使.直接get()即可，缺点：如果在单线程环境下可以保证安全，但是在多线程环境下是不可以的。如果使.加锁和ConcurrentHashMap都会产.性能问题。

.案3：使.ThreadLocal，实现不同.法间的资源共享 使.ThreadLocal可以避免加锁产.的性能问题，也可以避免层层传递参数来实现业务需求，就可以实现不同线程中存储不同信息的要求。

/***演.ThreadLocal的.法2：避免参数传递的.烦 */ publicclassThreadLocalNormalUsage06{publicstaticvoidmain(String[]args){newService1().process();}} classService1{ publicvoidprocess(){Useruser=newUser("鲁毅"); //将User对象存储到holder中 UserContextHolder.holder.set(user); newService2().process();}} classService2{ publicvoidprocess() {Useruser=UserContextHolder.holder.get();System.out.println("Service2拿到..名:"+user.name); newService3().process(); }} classService3{ publicvoidprocess(){Useruser=UserContextHolder.holder.get(); } System.out.println("Service3拿到.. 名:"+user.name); } classUserContextHolder{ publicstaticThreadLocalholder=newThreadLocal<>();} classUser{ Stringname; publicUser(Stringname){this.name=name;}} 运.结果

3.

对ThreadLocal的总结

让某个需要.到的对象实现线程之间的隔离（每个线程都有..独.的对象）可以在任何.法中轻松的获取到该对象根据共享对象.成的时机选择使.initialValue.法还是set.法对象初始化的时机由我们控制的时候使.initialValue.式如果对象.成的时机不由我们控制的时候使.set.式

4.

使.ThreadLocal的好处

达到线程安全的.的不需要加锁，执.效率.更加节省内存，节省开销免去传参的繁琐，降低代码耦合度 5.ThreadLocal原理

Thread ThreadLocal ThreadLocalMap 在Thread类内部有有ThreadLocal.ThreadLocalMapthreadLocals=null;这个变量，它.于存储ThreadLocal，因为在同.个线程当中可以有多个ThreadLocal，并且多次调.get()所以需要在内部维护.个ThreadLocalMap.来存储多个ThreadLocal 5.1ThreadLocal相关.法 TinitialValue() 该.法.于设置初始值，并且在调.get().法时才会被触发，所以是懒加载。但是如果在get()之前进.了set()操作，这样就不会调.initialValue()。 通常每个线程只能调..次本.法，但是调.了remove()后就能再次调. publicTget() {Threadt=Thread.currentThread();ThreadLocalMapmap=getMap(t); //获取到了值直接返回resule if(map!=null) {ThreadLocalMap.Entrye=map.getEntry(this);if(e!=null){ @SuppressWarnings("unchecked")Tresult=(T)e.value; returnresult; } }//没有获取到才会进.初始化 returnsetInitialValue();} privateTsetInitialValue(){ //获取initialValue.成的值，并在后续操作中进.set，最后将值返回 Tvalue=initialValue();Threadt=Thread.currentThread();ThreadLocalMapmap=getMap(t);if(map!=null) map.set(this,value); else createMap(t,value); returnvalue;} publicvoidremove() {ThreadLocalMapm=getMap(Thread.currentThread());if(m!=null) m.remove(this);} voidset(Tt) 为这个线程设置.个新值 publicvoidset(Tvalue){Threadt=Thread.currentThread();ThreadLocalMapmap=getMap(t);if(map!=null) map.set(this,value); else createMap(t,value);} Tget() 获取线程对应的value publicTget() {Threadt=Thread.currentThread();ThreadLocalMapmap=getMap(t);if(map!=null) {ThreadLocalMap.Entrye=map.getEntry(this);if(e!=null){@SuppressWarnings("unchecked")Tresult=(T)e.value; returnresult;}}returnsetInitialValue(); } voidremove() 删除对应这个线程的值 6.ThreadLocal注意点 6.1内存泄漏 内存泄露；某个对象不会再被使.，但是该对象的内存却.法被收回

staticclassThreadLocalMap{staticclassEntryextendsWeakReference<ThreadLocal>{ /**ThevalueassociatedwiththisThreadLocal.*/ Objectvalue; Entry(ThreadLocalk,Objectv){ //调..类，.类是.个弱引. super(k); //强引. value=v;}} 强引.：当内存不.时触发GC，宁愿抛出OOM也不会回收强引.的内存 弱引.：触发GC后便会回收弱引.的内存 正常情况 当Thread运.结束后，ThreadLocal中的value会被回收，因为没有任何强引.了 .正常情况 当Thread.直在运.始终不结束，强引.就不会被回收，存在以下调.链 Thread-->ThreadLocalMap-->Entry(key为null)-->value因为调.链中的value和Thread存在强引.，所以value.法被回收，就有可能出现OOM。 JDK的设计已经考虑到了这个问题，所以在set()、remove()、resize().法中会扫描到key为null的Entry，并且把对应的value设置为null，这样value对象就可以被回收。 privatevoidresize() {Entry[]oldTab=table;intoldLen=oldTab.length;intnewLen=oldLen*2;Entry[]newTab=newEntry[newLen];intcount=0; for(intj=0;j<oldLen;++j){Entrye=oldTab[j];if(e!=null){ ThreadLocal<?>k=e.get(); //当ThreadLocal为空时，将ThreadLocal对应的value也设置为null if(k==null){e.value=null;//HelptheGC }else{inth=k.threadLocalHashCode&(newLen-1); while(newTab[h]!=null) h=nextIndex(h,newLen);newTab[h]=e; count++; }}} setThreshold(newLen); size=count;table=newTab; } 但是只有在调.set()、remove()、resize()这些.法时才会进.这些操作，如果没有调.这些.法并且线程不停.，那么调.链就会.直存在，所以可能会发.内存泄漏。 6.2如何避免内存泄漏（阿.规约） 调.remove().法，就会删除对应的Entry对象，可以避免内存泄漏，所以使.完ThreadLocal后，要调.remove().法。 classService1{ publicvoidprocess(){Useruser=newUser("鲁毅"); //将User对象存储到holder中 UserContextHolder.holder.set(user); newService2().process();}} classService2{ publicvoidprocess() {Useruser=UserContextHolder.holder.get();System.out.println("Service2拿到..名:"+user.name); newService3().process();} } classService3{ publicvoidprocess() {Useruser=UserContextHolder.holder.get();System.out.println("Service3拿到..名:"+user.name); //.动释放内存，从.避免内存泄漏 UserContextHolder.holder.remove();}} 6.3ThreadLocal的空指针异常问题 /***ThreadLocal的空指针异常问题 */ publicclassThreadLocalNPE{ ThreadLocallongThreadLocal=newThreadLocal<>(); publicvoidset(){ longThreadLocal.set(Thread.currentThread().getId());} publicLongget(){returnlongThreadLocal.get();} publicstaticvoidmain(String[]args){ ThreadLocalNPEthreadLocalNPE=newThreadLocalNPE(); //如果get.法返回值为基本类型，则会报空指针异常，如果是包装类型就不会出错 System.out.println(threadLocalNPE.get()); Threadthread1=newThread(newRunnable(){@Override publicvoidrun(){ threadLocalNPE.set();System.out.println(threadLocalNPE.get()); }});thread1.start(); }} 6.4空指针异常问题的解决 如果get.法返回值为基本类型，则会报空指针异常，如果是包装类型就不会出错。这是因为基本类型和包装类型存在装箱和拆箱的关系，造成空指针问题的原因在于使.者。 6.5共享对象问题 如果在每个线程中ThreadLocal.set()进去的东西本来就是多个线程共享的同.对象，.如static对象，那么多个线程调. ThreadLocal.get()获取的内容还是同.个对象，还是会发.线程安全问题。 6.6可以不使.ThreadLocal就不要强.使. 如果在任务数很少的时候，在局部.法中创建对象就可以解决问题，这样就不需要使.ThreadLocal。 6.7优先使.框架的.持，.不是..创造 例如在Spring框架中，如果可以使.RequestContextHolder，那么就不需要..维护ThreadLocal，因为..可能会忘记调. remove().法等，造成内存泄漏。 本.

仅为..学习时记下的笔记，参考.慕课： https://coding.imooc.com/class/409.html -END - 推荐阅读 1. GitHub项.搜索技巧：让你更.效精准地搜索项.

2.

警告！你的隐私正在被上亿.友围观偷看！

3.

.场近乎完美基于 Dubbo的微服务改造实践

4.什么是.致性 Hash算法?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

分享100道Linux笔试题 Java后端 2019-10-12 点击上.Java后端，选择设为星标技术博.，及时送达 作者：PassZhang 原.：cnblogs.com/passzhang/p/9063455.html 1.cron后台常驻程序(daemon).于： A.负责.件在.络中的共享 B.管理打印.系统C.跟踪管理系统信息和错误 D.管理系统.常任务的调度

2.在.多数Linux发.版本中，以下哪个属于块设备(blockdevices)？ A.串.. B.硬盘 C.虚拟终端D.打印机 3.下.哪个Linux命令可以.次显...内容？ A.pauseB.cat C.more D.grep 4.怎样了解您在当前.录下还有多.空间？ A.Usedf B.Usedu/ C.Usedu. D.Usedf. 5.怎样更改.个.件的权限设置？ A.attrib B.chmod C.changeD.file 6.假如您需要找出/etc/my.conf.件属于哪个包(package)，您可以执.： A.rpm-q/etc/my.conf B.rpm-requires/etc/my.conf C.rpm-qf/etc/my.conf D.rpm-q|grep/etc/my.conf 7.假如当前系统是在level3运.，怎样不重启系统就可转换到level5运.？ A.Setlevel=5 B.telinit5 C.run5 D.ALT-F7-5 8.那个命令.于改变IDE硬盘的设置？ A.hdparamB.ideconfig C.hdparm D.hddparm 9.下.哪个命令可以列出定义在以后特定时间运..次的所有任务？ A.atq B.cron C.batch D.at 10.下.命令的作.是：setPS1="[\u\w\t]$";exportPS1 A.改变错误信息提. B.改变命令提.符 C.改变.些终端参数 D.改变辅助命令提.符

11.作为.个管理员，你希望在每.个新..的.录下放.个.件.bashrc，那么你应该在哪个.录下放这个.件，以便于新..创建主.录时.动将这个.件复制到..的.录下。 A./etc/skel/ B./etc/default/C./etc/defaults/D./etc/profile.d/ 12.在bash中，export命令的作.是： A.在.shell中运.命令 B.使在.shell中可以使.命令历史记录 C.为其它应.程序设置环境变量 D.提供NFS分区给.络中的其它系统使.

13.在使.了shadow.令的系统中，/etc/passwd和/etc/shadow两个.件的权限正确的是： A.-rw-r-----,-r---------B.-rw-r--r--,-r--r--r-- C.-rw-r--r--,-r--------- D.-rw-r--rw-,-r-----r-- 14．下.哪个参数可以删除.个..并同时删除..的主.录？ A.rmuser-r B.deluser-r C.userdel-r D.usermgr-r 15．有.个备份程序mybackup，需要在周..周五下午1点和晚上8点各运..次，下.哪条crontab的项可以完成这项.作？ A.013,20**1,5mybackup B.013,20**1,2,3,4,5mybackup C.*13,20**1,2,3,4,5mybackupD.013,201,5** mybackup 16．如何从当前系统中卸载.个已装载的.件系统 A.umount B.dismount C.mount-u D.从/etc/fstab中删除这个.件系统项 17．如果你的umask设置为022，缺省的你创建的.件的权限为： A.----w--w-B.-w--w----C.r-xr-x--- D.rw-r--r-- 18．在.条命令中如何查找.个.进制命令Xconfigurator的路径？ A.aproposXconfigurator B.findXconfiguratorC.whereXconfigurator D.whichXconfigurator 19．哪.条命令.来装载所有在/etc/fstab中定义的.件系统？ A.amount B.mount-a C.fmount D.mount-f 20．运..个脚本，..不需要什么样的权限？ A.read B.write C.execute D.browseonthedirectory 21．在Linux中，如何标识接在IDE0上的slave硬盘的第2个扩展分区？ A./dev/hdb2B./dev/hd1b2 C./dev/hdb6 D./dev/hd1b6 22．在应.程序起动时，如何设置进程的优先级？ A.priority B.nice C.renice D.setpri 23．在bash中,在.条命令后加."1>&2"意味着： A.标准错误输出重定向到标准输. B.标准输.重定向到标准错误输出

C.标准输出重定向到标准错误输出 D.标准输出重定向到标准输. 24．下.哪条命令可以把f1.txt复制为f2.txt? A.cpf1.txt|f2.txt B.catf1.txt|f2.txt C.catf1.txt>f2.txt D.copyf1.txt|f2.txt 25．显..个.件最后..的命令是： A.tac B.tail C.rear D.last 26.如何快速切换到..John的主.录下？ A.cd@John B.cd#John C.cd&John D.cd~John 27.把.个流中所有字符转换成.写字符，可以使.下.哪个命令？ A.tra-zA-Z B.taca-zA-Z C.sed/a-z/A-ZD.sed--toupper 28.使.什么命令可以查看Linux的启动信息？ A.mesg-d B.dmesg C.cat/etc/mesgD.cat/var/mesg 29.运.级定义在： A.inthekernel B.in/etc/inittab C.in/etc/runlevels D.usingtherlcommand 30.如何装载(mount)上在/etc/fstab.件中定义的所有.件系统？ A.mount-a

B.mount/mnt/*C.mount D.mount/etc/fstab 31.使.ln命令将.成了.个指向.件old的符号链接new，如果你将.件old删除，是否还能够访问.件中的数据？A.不可能再访问 B.仍然可以访问 C.能否访问取决于.件的所有者 D.能否访问取决于.件的权限

32.xt2fs.件系统中，缺省的为root..保留多.的空间？ A.3% B.5% C.10% D.15% 33.哪个命令.来显.系统中各个分区中inode的使.情况？ A.df-i B.df-H C.free-b D.du-a-c/ 34.多数Linux发.版本中，图形.式的运.级定义为？ A.1 B.2 C.3 D.5 35.在系统.档中找到关于print这个单词的所有说明？ A.manprint B.whichprint C.locateprint D.aproposprint 36.man5passwd含义是？ A.显.passwd命令的使..法 B.显.passwd.件的结构 C.显.passwd命令的说明的前五. D.显.关于passwd的前五处说明.档。

37.如何在.件中查找显.所有以""*打头的.？ A.find*file B.wc-l*<file C.grep-n*file D.grep'^*'file 38.在ps命令中什么参数是.来显.所有..的进程的？ A.a B.b C.u D.x 39.显..进制.件的命令是？ A.od B.vil C.view D.binview 40.如何显.Linux系统中注册的..数（包含系统..）？ A.account-l B.nl/etc/passwd|head C.wc--users/etc/passwd D.wc--lines/etc/passwd 41.在..结束位置加上什么符号，表.未结束，下..继续？ A./ B.\ C.;D.| 42.命令kill9的含义是： A.killstheprocesswhosePIDis9. B.killsallprocessesbelongingtoUID9. C.sendsSIGKILLtotheprocesswhosePIDis9. D.sendsSIGTERMtotheprocesswhosePIDIS9. 43.如何删除.个.空..录/tmp？ A.del/tmp/ B.rm-rf/tmp C.rm-Ra/tmp/* D.rm-rf/tmp/* 44.使.什么命令可以在今天午夜运.命令cmd1？ A.atmidnightcmd1 B.cron-at"00:00"cmd1 C.batch-t"00:00"<cmd1 D.echo"cmd1"|atmidnight 45.你的系统使.增量备份策略，当需要恢复系统时，你需要按什么顺序恢复备份数据？ A.最后.次全备份，然后从最早到最近的增量备份 B.最后.次全备份，然后从最近到最早的增量备份 C.最早到最近的增量备份，然后最后.次全备份 D.最近到最早的增量备份，然后最后.次全备份

46.对所有..的变量设置，应当放在哪个.件下？ A./etc/bashrc B./etc/profile C.~/.bash_profile D./etc/skel/.bashrc 47.Linux系统中，.般把命令ls定义为ls--color的别名，以便以不同颜.来标识不同类型的.件。但是，如何能够使.原先的 ls命令？ A.\ls B.;ls C.ls$$ D.ls--noalias 48.在Linux系统中的脚本.件.般以什么开头？ A.$/bin/sh B.#!/bin/sh C.use/bin/shD.setshell=/bin/sh 49.下.哪种写法表.如果cmd1成功执.，则执.cmd2命令？ A.cmd1&&cmd2 B.cmd1|cmd2 C.cmd1;cmd2D.cmd1||cmd2 50.在哪个.件中定义.卡的I/O地址？ A.cat/proc/modules B.cat/proc/devices C.cat/proc/ioports D.cat/io/dma 51.Linux中，提供TCP/IP包过滤功能的软件叫什么？ A.rarpB.route C.iptables D.filter 52.如何暂停.个打印队列？ A.lpr B.lpq C.lpc D.lpd 53.在vi中退出不保存的命令是？ A.:qB.:w C.:wq D.:q! 54.在XFree863.x中,缺省的字体服务器为： A.xfs B.xfserv C.fonts D.xfstt 55.使.什么命令检测基本.络连接？ A.ping B.route C.netstat D.ifconfig 56.下.哪个协议使.了.个以上的端.？ A.telnet B.FTP C.rsh D.HTTP 57.在PPP协议中，哪个认证协议不以明.传递密码？ A.PAM B.PAP C.PGP D.CHAP 58.下.哪个.件系统应该分配最.的空间？ A./usr B./lib C./rootD./bin 59.如何在Debian系统中安装rpm包？ A.alienpkgname.rpmB.dpkg--rpmpkgname.rpm C.dpkg--alienpkgname.rpm D.alienpkganme.rpm;dpkg-ipkganme.deb 60.在安装软件时下.哪.步需要root权限？ A.make B.makedepsC.makeconfig D.makeinstall 61.什么命令.来只更新已经安装过的rpm软件包？ A.rpm-U*.rpm B.rpm-F*.rpm C.rpm-e*.rpm D.rpm-q*.rpm 62.在windows与Linux双起动的系统中，如果要让LILO管理引导，则LILO应该放在： A.MBR B./C.root分区的.扇区 D./LILO 63.ldconfig的配置.件是 A./lib/ld.so B./etc/ld.so.conf C./etc/ld.so.cache D./etc/modules.conf 64.下.哪个命令可以压缩部分.件： A.tar-dzvffilename.tgz* B.tar-tzvffilename.tgz* C.tar-czvffilename.tgz* D.tar-xzvffilename.tgz* 65..络服务的daemon是： A.lpdB.netd C.httpd D.inetd 66.Linux与windows的.上领居互联，需要提供什么daemon? A.bind B.smbd C.nmbd D.shard 67.对于Apache服务器，提供的.进程的缺省的..是： A.root B.apachedC.httpd D.nobody 68.sendmail中缺省的未发出信件的存放位置是： A./var/mail/B./var/spool/mail/ C./var/spool/mqueue/ D./var/mail/deliver/ 69.apache的主配置.件是： A.httpd.conf B.httpd.cfgC.access.cfgD.apache.conf 70.关于可装载的模块，装载时的参数，如I/O地址等的存放位置是： A./etc/conf.modules B./etc/lilo.conf C./boot/System.mapD./etc/sysconfig 71.在Linux中，如何关闭邮件提.？ A.biffn B.mesgnC.notifyoff D.setnotify=off 72.在bashshell环境下，当.命令正在执.时，按下control-Z会： A.中.前台任务 B.给当前.件加上EOF. C.将前台任务转.后台 D.注销当前..

73.定义bash环境的...件是： A.bash&.bashrc B.bashrc&.bash_conf C.bashrc&bash_profile D..bashrc&.bash_profile 74.下.哪条命令.来显..个程序所使.的库.件？ A.ldd B.ldso C.modprobeD.ldconfig 75.如何查看.个RPM软件的配置.件的存放位置？ A.rpm-qcrpm1 B.rpm-Vcrpm1 C.rpm--configrpm1 D.rpm-qa--configrpm1 76.如何查看.个RPM软件的修改记录？ A.rpm-Vcpostfix B.rpm-qpilpostfix C.rpm--changelogpostfix D.rpm-q--changelogpostfix 77.通过Makefile来安装已编译过的代码的命令是： A.make B.install C.makedepend D.makeinstall 78.什么命令解压缩tar.件？ A.tar-czvffilename.tgz B.tar-xzvffilename.tgz C.tar-tzvffilename.tgz D.tar-dzvffilename.tgz 79.在XF86Config配置.件中，哪个段.来设置字体.件？ A.TheFontssection. B.TheFilessection. C.ThexfsCodessection. D.TheGraphicssection. 80.8bitcolor指的是： A.64Kcolors B.16Kcolors C.256colors D.16Mcolors 81.下.哪个.件.来设置Xwindow的显.分辨率？ A.xinit B.xinitrc

C.XF86Setup D.XF86Config 82.哪个变量.来指定.个远程X应.程序将输出放到哪个Xserver上？ A.DISPLAY B.TERM C.ECHO D.OUTPUT 83.在xdm的配置.录中，哪个.件.来设置在..通过xdm登录后.动起动的应.程序？ A.TheXsessionfile B.TheXsetup_0file C.TheXstart_upfile D.TheGiveConsolefile 84.命令netstat-a停了很.时间没有响应，这可能是哪.的问题？ A.NFS. B.DNS. C.NIS. D.routing. 85.ping使.的协议是： A.TCP B.UDP C.SMB D.ICMP 86.下.哪个命令不是.来查看.络故障的？ A.ping B.init C.telnet D.netstat 87.拨号上.使.的协议通常是： A.PPP B.UUCP C.SLIP D.Ethernet 88.TCP/IP中，哪个协议是.来进.IP.动分配的？ A.ARP B.NFS C.DHCP D.DNS 89.下.哪个.件定义了.络服务的端.？ A./etc/netport B./etc/services C./etc/serverD./etc/netconf 90.下.哪个功能.来.成.个.件的校验码？ A.md5 B.tar C.cryptD.md5sum 91.缺省的，..邮件放在： A.~/mail/B./var/mail/C./var/mail/spool/ D./var/spool/mail/ 92.下.哪个.件包含了供NFSdaemon使.的.录列表？ A./etc/nfs B./etc/nfs.conf C./etc/exports D./etc/netdir 93.如何停..台机器的telnet服务？ A.PutNONEin/etc/telnet.allow B.Putaline'ALL:ALL'in/etc/hosts.denyC.Commentthetelnetentryinⁱ/etc/inittab D.Commentthetelnetentryinⁱ/etc/xinetd.conf 94.在哪个.件中保存了sendmail的别名？ A./etc/aliases B./etc/mailaliases C./etc/sendmail.aliases D./etc/sendmail/aliases 95.smbdandnmbddaemons的配置.件是： A./etc/exports B./etc/smb.conf C./etc/samba/config D./usr/local/samba.cfg 96.下.哪个命令.来卸载.个内核模块？ A.rmmod B.unmod C.delmod D.modprobe 97.什么情况下必须运.lilo A.onceadayfromcron B.onceaweekfromcron C.afterinstallinganewkernel D.afterinstallinganewmodule 98.什么命令显.所有装载的模块？ A.lsmod B.dirmod C.modules D.modlist 99.下.哪个命令刷新打印机队列？ A.lpflush B.lprm- C.lpclear D.lprmall 100.下.哪个命令可以查看.卡的中断？ A.cat/proc/ioports B.cat/proc/interrupts C.cat/proc/memoryinfo D.whichinterrupts 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 去阿..试Java都是问什么? huashiouJava后端 2019-10-31 点击上.Java后端，选择设为星标 优质.章，及时送达

作者 |不穿格.衫的Java程序猿来源 |jianshu.com/p/1c8271f03aa5 上篇 |终于有.把Docker讲清楚了 每.个互联...中都有.个..梦，百度、阿.巴巴、腾讯是很多互联..梦寐以求的地.，.我也不例外。但是，BAT等.线互联...并不是想进就能够进的，它对.才的技术能.和学历都是有.定要求的，所以除了学历以外，我们的技术和能.都要过硬才.。 今年前前后后我参加了阿.巴巴两次.试，.次是社招，.次是内推，第.次社招3.过后就被挂了，内推历经5.拿到的o ffer，进.的是阿..碑部.，分享.下这次的.经，希望能帮助到.家。 社招阿.巴巴（新零售部.），三.被挂 阿.巴巴..（55分钟） 先介绍.下..吧 说.下..的优缺点 具体讲.下之前做过的项. 你觉得项..给.最.的挑战是什么？ Hashmap为什么不.平衡树？ AQS知道吗？知道哪.些呢？讲.讲。CLH同步队列是怎么实现.公平和公平的？ ReetrantLock和synchron ized的区别 讲.下JVM的内存结构 JVM.new对象时，堆会发.抢占吗？你是怎么去设计JVM的堆的线程安全的？ 讲.下red is的数据结构 redis缓存同步问题 讲.讲MySQL的索引结构 你有什么问题要问我吗？ 直接.头通知我：答得不错，准备..吧 阿.巴巴..（45分钟） 根据项.问了.些细节问题 说.下HashMap的数据结构 红.树和AVL树有什么区别？ 如何才能得到.个线程安全的HashMap？ 讲.下JVM常.垃圾回收期 redis分布式锁再描述.下你之前的项.吧你觉得这个项.的亮点在哪.呢？你设计的数据库遵循的范式？你有没有问题？ 阿.巴巴三.（50分钟） .聊项.在项.中，并发量.的情况下，如何才能够保证数据的.致性？ elasticsearch为什么检索快，它的底层数据结构是怎么样的？JVM内存模型netty应.在哪些中间件和框架中呢？线程池的参数讲.下B树和B+树的区别为什么要.red is做缓存？了解Spr ingboot吗？那讲.下Spr ingboot的启动流程吧如何解决bean的循环依赖问题？Java有哪些队列？讲.讲Spr ing和Spr ingboot的区别最近看了什么书？为什么？你平时是怎么学习Java的呢？ 内推阿.巴巴（阿..碑） 5.拿o ffer（3轮技术.+总监.+HR.） 阿.巴巴..（38分钟）-.我介绍 介绍项.,具体.点讲.下Red is分布式锁的实现 HashMap了解么吗？说.下put.法过程HashMap是不是线程安全？ConcurrentHashMap如何保证线程安全？数据库索引了解吗？讲.下常.排序算法TCP三次握.，四次挥.。深.问了乐观锁，悲观锁及其实现。 阿.巴巴..（45分钟） .我介绍+项.介绍 你在项.中担任什么样的..？那你觉得你.别.的优势在哪.？你.了哪些别.没有的东西吗？Java怎么加载类？ linux常.命令有哪些？ Spring的 IOC,AOP。讲.下ORM框架H ibernate设计模式了解吗？讲.下..实现.个.阶段提交，如何设计？你还有什么想问的？ 阿.巴巴三.（30分钟） 说.下..做的项.问了.些项.相关的问题 wait()和s leep()的区别原.变量的实现原理 CAS的问题，讲.下解决.案。有没有更好的计数器解决策略讲.讲N IO和B IO的区别 Nginx负载均衡时是如何判断某个节点挂掉了？讲.下red is的数据类型和使.场景k8s的储存.式是怎样的？ SpringAOP原理是什么？怎么使.？什么是切点，什么是切.？最好是举个例.算法题：给.堆硬币的array，返回所有的组合 阿.巴巴总监.（34分钟） 算法：给.个set打印出所有.集；多线程从多个.件中读.数据，写到同.个.件中；判断 ip是否在给定范围内；打乱.副扑克牌，不能.额外空间，证明为什么是随机的。Tcp和udp区别线程池的原理以及各种线程池的应.场景线程池中使.有限的阻塞队列和.限的阻塞队列的区别如果你发现你的sq l语句始终.另.个索引，但是你希望它.你想要的索引，怎么办？ mysql执.计划数据库索引为什么.b+树？你在做sq l优化主要从哪.个..做，.到哪些.法.具？有没有想问的？ 阿.巴巴HR.（23分钟） .我介绍 平时怎么学习的？ 有什么兴趣爱好吗？ 怎么看待996？怎么平衡.作和学习？有没有什么想问的 总结 社招时.试新零售部.，主要因为准备不充分，.试..较紧张，所以发挥不是很好，三.之后没有了后续。之后意识到学习的重要性，平时多拿出时间来学习，后来幸运地拿到内推资格，为了把握住这次机会，做了很多准备，好在已经拿到offer。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.淘宝为什么能抗住双 11?

2.

为什么？阿.规定超过 3张表禁. join

3.理解 IntelliJ IDEA的项.配置和 Web部署

4.

Java开发中常.的 4种加密.法

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

发布没有答案的.试题，都是耍流氓 江湖.称..哥Java后端 2019-12-23 点击上.Java后端，选择设为星标 优质.章，及时送达

作者 |江湖.称..哥 链接|blog.csdn.net/dd864140130/article/details/55833087 今天要谈的主题是关于求职，求职是在每个技术.员的.涯中都要经历多次。对于我们.部分...，在进....仪的公司之前少不了准备.作，有.份全.细致.试题将帮助我们减少许多.烦。这个系列的.章，...帮助..巩固下基础，另...也希望帮助想要换.作的朋友。 相关概念 .向对象的三个特征 封装，继承，多态，这个应该是..皆知，有时候也会加上抽象。 多态的好处 允许不同类对象对同.消息做出响应，即同.消息可以根据发送对象的不同.采.多种不同的.为.式(发送消息就是函数调 .)。主要有以下优点：

可替换性：多态对已存在代码具有可替换性可扩充性：增加新的.类不影响已经存在的类结构接.性：多态是超类通过.法签名,向.类提供.个公共接.,由.类来完善或者重写它来实现的。灵活性简化性 代码中如何实现多态 实现多态主要有以下三种.式： 1. 接.实现

2.

继承.类重写.法

3.

同.类中进..法重载

虚拟机是如何实现多态的 动态绑定技术(dynamicbinding)，执.期间判断所引.对象的实际类型，根据实际类型调.对应的.法。 接.的意义 接.的意义.三个词就可以概括：规范，扩展，回调。 抽象类的意义 抽象类的意义可以.三句话来概括：

为其他.类提供.个公共的类型 封装.类中重复定义的内容 定义抽象.法,.类虽然有不同的实现，但是定义时.致的 接.和抽象类的区别 .较抽象类接. java8之前,接.中不存在.法的默认.法抽象类可以有默认的.法实现实现. .类使.extends关键字来继承抽象类.如果.类.类使.implements来实现接实现.式不是抽象类,.类需要提供抽象类中所声明.法.,需要提供接.中所有声明的实 的实现.现.构造器抽象类中可以有构造器,接.中不能和正常类区别抽象类不能被实例化接.则是完全不同的类型访问修饰符抽象.法可以有public,protected和default等接.默认是public,不能使.其他 修饰修饰符 多继承.个.类只能存在.个.类.个.类可以存在多个接.添加新.法想抽象类中添加新.法,可以提供默认的实现,因如果往接.中添加新.法,则.类此可以不修改.类现有的代码中需要实现该.法. .类的静态.法能否被.类重写 不能。重写只适.于实例.法,不能.于静态.法，..类当中含有和.类相同签名的静态.法，我们.般称之为隐藏。 什么是不可变对象 不可变对象指对象.旦被创建，状态就不能再改变。任何修改都会创建.个新的对象，如String、Integer及其它包装类。 静态变量和实例变量的区别? 静态变量存储在.法区，属于类所有。实例变量存储在堆当中，其引.存在当前线程栈。 Tips：欢迎关注Java后端，每天技术.章推送。 能否创建.个包含可变对象的不可变对象? 当然可以创建.个包含可变对象的不可变对象的，你只需要谨慎.点，不要共享可变对象的引.就可以了，如果需要变化时，就返回原对象的.个拷.。最常.的例.就是对象中包含.个.期对象的引.。 java创建对象的.种.式

采.new通过反射 采.clone 通过序列化机制 前2者都需要显式地调.构造.法。造成耦合性最.的恰好是第.种，因此你发现.论什么框架，只要涉及到解耦必先减少new的使.。 switch中能否使.string做参数 在idk1.7之前，switch只能.持byte, short, char, int或者其对应的封装类以及Enum类型。从idk1.7之后switch开始.持 String。 switch能否作.在byte,long上?

可以.在byte上，但是不能.在long上。 Strings1=”ab”,Strings2=”a”+”b”,Strings3=”a”,Strings4=”b”,s5=s3+s4请问s5==s2返回什么？ 返回false。在编译过程中，编译器会将s2直接优化为”ab”，会将其放置在常量池当中，s5则是被创建在堆区，相当于s5=new String(“ab”); 你对String对象的intern()熟悉么? intern().法会.先从常量池中查找是否存在该常量值，如果常量池中不存在则现在常量池中创建，如果已经存在则直接返回。.如 Strings1=”aa”;Strings2=s1.intern();System.out.print(s1==s2);//返回true Object中有哪些公共.法?

equals()clone()getClass()notify(),notifyAll(),wait()toString java当中的四种引. 强引.，软引.，弱引.，虚引.。不同的引.类型主要体现在GC上:

强引.：如果.个对象具有强引.，它就不会被垃圾回收器回收。即使当前内存空间不.，JVM也不会回收它，.是抛出OutOfMemoryError错误，使程序异常终.。如果想中断强引.和某个对象之间的关联，可以显式地将引.赋值为 null，这样.来的话，JVM在合适的时间就会回收该对象。软引.：在使.软引.时，如果内存的空间.够，软引.就能继续被使.，.不会被垃圾回收器回收，只有在内存不.时，软引.才会被垃圾回收器回收。弱引.：具有弱引.的对象拥有的.命周期更短暂。因为当 JVM进.垃圾回收，.旦发现弱引.对象，.论当前内存空间是否充.，都会将弱引.回收。不过由于垃圾回收器是.个优先级较低的线程，所以并不.定能迅速发现弱引.对象。虚引.：顾名思义，就是形同虚设，如果.个对象仅持有虚引.，那么它相当于没有引.，在任何时候都可能被垃圾回收器回收。更多了解参.深.对象引.：http://blog.csdn.net/dd864140130/article/details/49885811 WeakReference与SoftReference的区别? 这点在四种引.类型中已经做了解释,这.简单说明.下即可:虽然WeakReference与SoftReference都有利于提.GC和内存的效率，但是WeakReference，.旦失去最后.个强引.，就会被GC回收，.软引.虽然不能阻.被回收，但是可以延迟到JVM内存不.的时候。 为什么要有不同的引.类型 不像C语.，我们可以控制内存的申请和释放，在Java中有时候我们需要适当的控制对象被回收的时机，因此就诞.了不同的引.类型，可以说不同的引.类型实则是对GC回收时机不可控的妥协。有以下.个使.场景可以充分的说明：

利.软引.和弱引.解决OOM问题：..个HashMap来保存图.的路径和相应图.对象关联的软引.之间的映射关系，在内存不.时，JVM会.动回收这些缓存图.对象所占.的空间，从.有效地避免了OOM的问题.

通过软引.实现Java对象的.速缓存:.如我们创建了.Person的类，如果每次需要查询.个.的信息,哪怕是.秒中之前刚刚查询过的，都要重新构建.个实例，这将引起.量Person对象的消耗，并且由于这些对象的.命周期相对较短，会引起多次GC影响性能。此时，通过软引.和HashMap的结合可以构建.速缓存，提供性能。

java中==和eqauls()的区别,equals()和`hashcode的区别 ==是运算符，.于.较两个变量是否相等，.equals是Object类的.法，.于.较两个对象是否相等。默认Object类的 equals.法是.较两个对象的地址，此时和==的结果.样。换句话说：基本类型.较.==，.较的是他们的值。默认下，对象.==.较时，.较的是内存地址，如果需要.较对象内容，需要重写equal.法。 equals()和hashcode()的联系 hashCode()是Object类的.个.法，返回.个哈希值。如果两个对象根据equal().法.较相等，那么调.这两个对象中任意.个对象的hashCode().法必须产.相同的哈希值。 如果两个对象根据eqaul().法.较不相等，那么产.的哈希值不.定相等(碰撞的情况下还是会相等的。) a.hashCode()有什么.?与a.equals(b)有什么关系 hashCode().法是相应对象整型的 hash值。它常.于基于 hash的集合类，如 Hashtable、HashMap、 LinkedHashMap等等。它与equals().法关系特别紧密。根据 Java规范，使. equal().法来判断两个相等的对象，必须具有相同的 hashcode。 将对象放.到集合中时，.先判断要放.对象的hashcode是否已经在集合中存在，不存在则直接放.集合。如果hashcode相等，然后通过equal().法判断要放.对象与集合中的任意对象是否相等：如果equal()判断不相等，直接将该元素放.集合中，否则不放.。 有没有可能两个不相等的对象有相同的hashcode 有可能，两个不相等的对象可能会有相同的 hashcode值，这就是为什么在 hashmap中会有冲突。如果两个对象相等，必须有相同的hashcode值，反之不成.。 可以在hashcode中使.随机数字吗? 不.，因为同.对象的hashcode值必须是相同的 a==b与a.equals(b)有什么区别 如果a和b都是对象，则a==b是.较两个对象的引.，只有当 a和 b指向的是堆中的同.个对象才会返回true，. a.equals(b)是进.逻辑.较，所以通常需要重写该.法来提供逻辑.致性的.较。例如，String类重写 equals().法，所以可以.于两个不同对象，但是包含的字.相同的.较。 3*0.1==0.3返回值是什么 false，因为有些浮点数不能完全精确的表.出来。 a=a+b与a+=b有什么区别吗? +=操作符会进.隐式.动类型转换，此处a+=b隐式的将加操作的结果类型强制转换为持有结果的类型，.a=a+b则不会.动进.类型转换。如： bytea=127;byteb=127;b=a+b;//error:cannotconvertfrominttobyte b+=a;//ok （译者注：这个地.应该表述的有误，其实.论 a+b的值为多少，编译器都会报错，因为 a+b操作会将 a、b提升为 int类型，所以将 int类型赋值给byte就会编译出错） shorts1=1;s1=s1+1;该段代码是否有错,有的话怎么改？ 有错误，short类型在进.运算时会.动提升为int类型，也就是说s1+1的运算结果是int类型。 shorts1=1;s1+=1;该段代码是否有错，有的话怎么改？ +=操作符会.动对右边的表达式结果强转匹配左边的数据类型，所以没错。 &和&&的区别 .先记住&是位操作，.&&是逻辑运算符。另外需要记住逻辑运算符具有短路特性，.&不具备短路特性。 publicclassTest{ staticStringname; publicstaticvoidmain(String[]args){if(name!=null&userName.equals(""))

{System.out.println("ok");}else{System.out.println("erro"); }}} 以上代码将会抛出空指针异常。 .个java.件内部可以有类? (.内部类) 只能有.个public公共类，但是可以有多个default修饰的类。 如何正确的退出多层嵌套循环?

使.标号和break;通过在外层循环中添加标识符 内部类的作. 内部类可以有多个实例，每个实例都有..的状态信息，并且与其他外围对象的信息相互独..在单个外围类当中，可以让多个内部类以不同的.式实现同.接.，或者继承同.个类.创建内部类对象的时刻不依赖于外部类对象的创建。内部类并没有令.疑惑的"is-a"管系，它就像是.个独.的实体。 内部类提供了更好的封装，除了该外围类，其他类都不能访问。 final,finalize和finally的不同之处 .nal是.个修饰符，可以修饰变量、.法和类。如果 .nal修饰变量，意味着该变量的值在初始化后不能被改变。.nalize.法是在对象被回收之前调.的.法，给对象..最后.个复活的机会，但是什么时候调. .nalize没有保证。.nally是.个关键字，与 try和catch.起.于异常的处理。.nally块.定会被执.，.论在 try块中是否有发.异常。 clone()是哪个类的.法? java.lang.Cloneable是.个标.性接.，不包含任何.法，clone.法在 object类中定义。并且需要知道clone().法是.个本地.法，这意味着它是由c或c++或其他本地语.实现的。 深拷.和浅拷.的区别是什么? 浅拷.：被复制对象的所有变量都含有与原来的对象相同的值，.所有的对其他对象的引.仍然指向原来的对象。换.之，浅拷.仅仅复制所考虑的对象，.不复制它所引.的对象。

深拷.：被复制对象的所有变量都含有与原来的对象相同的值，.那些引.其他对象的变量将指向被复制过的新对象，.不再是原有的那些被引.的对象。换.之，深拷.把要复制的对象所引.的对象都复制了.遍。 static都有哪些.法? .乎所有的.都知道static关键字这两个基本的.法：静态变量和静态.法。也就是被static所修饰的变量/.法都属于类的静态资源，类实例所共享。 除了静态变量和静态.法之外，static也.于静态块，多.于初始化操作： publiccalssPreCache{static{ //执.相关操作 } } 此外static也多.于修饰内部类，此时称之为静态内部类。 最后.种.法就是静态导包，即importstatic.importstatic是在JDK1.5之后引.的新特性，可以.来指定导.某个类中的静态 资源，并且不需要使.类名。资源名，可以直接使.资源名，.如： importstaticjava.lang.Math.*; publicclassTest{publicstaticvoidmain(String[]args){ //System.out.println(Math.sin(20)); 传统做法System.out.println(sin(20)); } } final有哪些.法 final也是很多.试喜欢问的地.，能回答下以下三点就不错了： 1.被 final修饰的类不可以被继承

2.被 final修饰的.法不可以被重写

3.被 final修饰的变量不可以被改变。如果修饰引.，那么表.引.不可变，引.指向的内容可变。

4.被 final修饰的.法，JVM会尝试将其内联，以提.运.效率

5.被 final修饰的常量，在编译阶段会存.常量池中。

回答出编译器对final域要遵守的两个重排序规则更好： 1. 在构造函数内对.个final域的写.，与随后把这个被构造对象的引.赋值给.个引.变量,这两个操作之间不能重排序。

2.初次读.个包含 final域的对象的引.，与随后初次读这个final域,这两个操作之间不能重排序。

数据类型相关 java中intchar,long各占多少字节? 类型 位数 字节数
short 2 16
int 4 32
longfloat 8 4 64 32
double 8 64
char 2 16

64位的JVM当中,int的.度是多少? Java中，int类型变量的.度是.个固定值，与平台.关，都是32位。意思就是说，在32位和64位的Java虚拟机中，int类型的.度是相同的。 int和Integer的区别 Integer是int的包装类型，在拆箱和装箱中，.者.动转换。int是基本类型，直接存数值，.integer是对象，..个引.指向这个对象。 int和Integer谁占.的内存更多? Integer对象会占.更多的内存。Integer是.个对象，需要存储对象的元数据。但是 int是.个原始类型的数据，所以占.的空间更少。String,StringBuffer和StringBuilder区别 String是字符串常量，final修饰：StringBuffer字符串变量(线程安全)；StringBuilder字符串变量(线程不安全)。 String和StringBuffer String和StringBuffer主要区别是性能：String是不可变对象，每次对String类型进.操作都等同于产.了.个新的String对象，然后指向新的String对象。所以尽量不在对String进..量的拼接操作，否则会产.很多临时对象，导致GC开始.作，影响系统性能。 StringBuffer是对对象本.操作，.不是产.新的对象，因此在有.量拼接的情况下，我们建议使.StringBuffer。 但是需要注意现在JVM会对String拼接做.定的优化：Strings="Thisisonly"+"simple"+"test"会被虚拟机直接优化成Strings="Thisisonlysimpletest"，此时就不存在拼接过程。StringBuffer和StringBuilder StringBu.er是线程安全的可变字符串，其内部实现是可变数组。StringBuilder是jdk 1.5新增的，其功能和StringBu.er类似，但是.线程安全。因此，在没有多线程问题的前提下，使.StringBuilder会取得更好的性

能。 什么是编译器常量？使.它有什么.险？ 公共静态不可变（public static .nal）变量也就是我们所说的编译期常量，这.的 public可选的。实际上这些变量在编译时会被替换掉，因为编译器知道这些变量的值，并且知道这些变量在运.时不能改变。这种.式存在的.个问题是你使.了.个内部的或第三.库中的公有编译时常量，但是这个值后.被其他.改变了，但是你的客.端仍然在使..的值，甚.你已经部署了.个新的jar。为了避免这种情况，当你在更新依赖JAR.件时，确保重新编译你的程序。 java当中使.什么类型表.价格.较好？如果不是特别关.内存和性能的话，使.BigDecimal，否则使.预定义精度的double类型。 如何将byte转为String 可以使. String接收 byte[]参数的构造器来进.转换，需要注意的点是要使.的正确的编码，否则会使.平台默认编码，这个编码可能跟原来的编码相同，也可能不同。 可以将int强转为byte类型么?会产.什么问题？我们可以做强制转换，但是Java中int是32位的.byte是8位的，所以,如果强制转化int类型的.24位将会被丢弃，byte类型的范围是从-128到128 关于垃圾回收 你知道哪些垃圾回收算法？垃圾回收从理论上.常容易理解,具体的.法有以下.种: 1. 标记-清除

2.

标记-复制

3.

标记-整理4.分代回收

更详细的内容参.深.理解垃圾回收算法： http://blog.csdn.net/dd864140130/article/details/50084471 如何判断.个对象是否应该被回收 这就是所谓的对象存活性判断，常.的.法有两种：1.引.计数法; 2.对象可达性分析。由于引.计数法存在互相引.导致.法进.GC的问题，所以.前JVM虚拟机多使.对象可达性分析算法。 简单的解释.下垃圾回收 Java垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。.般的实现是划分成3个世代：年轻、年.和永久。内存的分配是发.在年轻世代中的。当.个对象存活时间.够.的时候，它就会被复制到年.世代中。对于不同的世代可以使.不同的垃圾回收算法。进.世代划分的出发点是对应.中对象存活时间进.研究之后得出的统计规律。.般来说，.个应.中的.部分对象的存活时间都很短。.如局部变量的存活时间就只在.法的执.过程中。基于这.点，对于年轻世代的垃圾回收算法就可以很有针对性。 调.System.gc()会发.什么？通知GC开始.作，但是GC真正开始的时间不确定。 进程,线程相关 说说进程，线程，协程之间的区别 简..之，进程是程序运.和资源分配的基本单位，.个程序.少有.个进程，.个进程.少有.个线程。进程在执.过程中拥有独.的内存单元，.多个线程共享内存资源，减少切换次数，从.效率更.。线程是进程的.个实体，是cpu调度和分派的基本单位，是.程序更.的能独.运.的基本单位。同.进程中的多个线程之间可以并发执.。 你了解守护线程吗？它和.守护线程有什么区别 程序运.完毕，jvm会等待.守护线程完成后关闭，但是jvm不会等待守护线程。守护线程最典型的例.就是GC线程。 什么是多线程上下.切换 多线程的上下.切换是指CPU控制权由.个已经正在运.的线程切换到另外.个就绪并等待获取CPU执.权的线程的过程。 创建两种线程的.式?他们有什么区别？通过实现java.lang.Runnable或者通过扩展java.lang.Thread类。相.扩展Thread，实现Runnable接.可能更优.原因有.:

Java不.持多继承。因此扩展Thread类就代表这个.类不能扩展其他类。.实现Runnable接.的类还可能扩展另.个类。 类可能只要求可执.即可，因此继承整个Thread类的开销过.。 Thread类中的start()和run().法有什么区别？start().法被.来启动新创建的线程，.且start()内部调.了run().法，这和直接调.run().法的效果不.样。当你调.run().法的时候，只会是在原来的线程中调.，没有新的线程启动，start().法才会启动新线程。 怎么检测.个线程是否持有对象监视器 Thread类提供了.个holdsLock(Objectobj).法，当且仅当对象obj的监视器被某条线程持有的时候才会返回true，注意这是.个static.法，这意味着"某条线程"指的是当前线程。 Runnable和Callable的区别 Runnable接.中的run().法的返回值是void，它做的事情只是纯粹地去执.run().法中的代码.已；Callable接.中的call().法是有返回值的，是.个泛型，和Future、FutureTask配合可以.来获取异步执.的结果。 这其实是很有.的.个特性，因为多线程相.单线程更难、更复杂的.个重要原因就是因为多线程充满着未知性，某条线程是否执.了？某条线程执.了多久？某条线程执.的时候我们期望的数据是否已经赋值完毕？.法得知，我们能做的只是等待这条多线程的任务执.完毕.已。.Callable+Future/FutureTask却可以.便获取多线程运.的结果，可以在等待时间太.没获取到需要的数据的情况下取消该线程的任务。 什么导致线程阻塞 阻塞指的是暂停.个线程的执.以等待某个条件发.（如某资源就绪），学过操作系统的同学对它.定已经很熟悉了。Java提供了.量.法来.持阻塞，下.让我们逐.分析。 .法说明 sleep()允许指定以毫秒为单位的.段时间作为参数，它使得线程在指定的 sleep()时间内进.阻塞状态，不能得到CPU时间，指定的时间.过，线程重新进.可执.状态。典型地，sleep()被.在等待某个资源就绪的情形：测试发现条件不满.后，让线程阻塞.段时间后重新测试，直到条件满.为.两个.法配套使.，suspend()使得线程进.阻塞状态，并且不会.动恢复 suspend()，必须其对应的resume()被调.，才能使得线程重新进.可执.状态。典 和resume()型地，suspend()和resume()被.在等待另.个线程产.的结果的情形：测试发现结果还没有产.后，让线程阻塞，另.个线程产.了结果后，调 .resume()使其恢复。 yield()使当前线程放弃当前已经分得的CPU时间，但不使当前线程阻塞， yield()即线程

仍处于可执.状态，随时可能再次分得 CPU时间。调. yield()的效果等价于调度程序认为该线程已执.了.够的时间从.转到另.个线程两个.法配套使.，wait()使得线程进.阻塞状态，它有两种形式，.种 wait()和no允许指定以毫秒为单位的.段时间作为参数，另.种没有参数，前者当对 tify()应的 notify()被调.或者超出指定时间时线程重新进.可执.状态，后者则必须对应的notify()被调.。 wait(),notify()和suspend(),resume()之间的区别 初看起来它们与 suspend()和 resume() .法对没有什么分别，但是事实上它们是截然不同的。区别的核.在于，前.叙述的所有.法，阻塞时都不会释放占.的锁（如果占.了的话），.这.对.法则相反。上述的核.区别导致了.系列的细节上的区别。 .先，前.叙述的所有.法都.属于Thread类，但是这.对却直接.属于Object类，也就是说，所有对象都拥有这.对.法。初看起来这.分不可思议，但是实际上却是很.然的，因为这.对.法阻塞时要释放占.的锁，.锁是任何对象都具有的，调.任意对象的 wait().法导致线程阻塞，并且该对象上的锁被释放。.调.任意对象的notify().法则导致从调.该对象的wait().法.阻塞的线程中随机选择的.个解除阻塞（但要等到获得锁后才真正可执.）。 其次，前.叙述的所有.法都可在任何位置调.，但是这.对.法却必须在 synchronized .法或块中调.，理由也很简单，只有在synchronized.法或块中当前线程才占有锁，才有锁可以释放。同样的道理，调.这.对.法的对象上的锁必须为当前线程所拥有，这样才有锁可以释放。因此，这.对.法调.必须放置在这样的 synchronized .法或块中，该.法或块的上锁对象就是调.这.对.法的对象。若不满.这.条件，则程序虽然仍能编译，但在运.时会出现IllegalMonitorStateException异常。 wait()和notify().法的上述特性决定了它们经常和synchronized关键字.起使.，将它们和操作系统进程间通信机制作.个.较就会发现它们的相似性：synchronized.法或块提供了类似于操作系统原语的功能，它们的执.不会受到多线程机制的.扰，.这.对.法则相当于 block和wakeup原语（这.对.法均声明为 synchronized）。它们的结合使得我们可以实现操作系统上.系列精妙的进程间通信的算法（如信号量算法），并.于解决各种复杂的线程间通信问题。 关于wait()和notify().法最后再说明两点： 第.：调.notify().法导致解除阻塞的线程是从因调.该对象的 wait().法.阻塞的线程中随机选取的，我们.法预料哪.个线程将会被选择，所以编程时要特别..，避免因这种不确定性.产.问题。 第.：除了notify()，还有.个.法notifyAll()也可起到类似作.，唯.的区别在于，调.notifyAll().法将把因调.该对象的 wait().法.阻塞的所有线程.次性全部解除阻塞。当然，只有获得锁的那.个线程才能进.可执.状态。谈到阻塞，就不能不谈.谈死锁，略.分析就能发现，suspend().法和不指定超时期限的wait().法的调.都可能产.死锁。遗憾的是，Java并不在语.级别上.持死锁的避免，我们在编程中必须..地避免死锁。 以上我们对Java中实现线程阻塞的各种.法作了.番分析，我们重点分析了wait()和notify().法，因为它们的功能最强.，使.也最灵活，但是这也导致了它们的效率较低，较容易出错。实际使.中我们应该灵活使.各种.法，以便更好地达到我们的.的。 产.死锁的条件 1. 互斥条件：.个资源每次只能被.个进程使.。

2.

请求与保持条件：.个进程因请求资源.阻塞时，对已获得的资源保持不放。

3.

不剥夺条件:进程已获得的资源，在未使.完之前，不能强.剥夺。

4.

循环等待条件:若.进程之间形成.种头尾相接的循环等待资源关系。

为什么wait().法和notify()/notifyAll().法要在同步块中被调. 这是JDK强制的，wait().法和notify()/notifyAll().法在调.前都必须先获得对象的锁 wait().法和notify()/notifyAll().法在放弃对象监视器时有什么区别 wait().法和notify()/notifyAll().法在放弃对象监视器的时候的区别在于：wait().法.即释放对象监视器，notify()/notifyAll().法则会等待线程剩余代码执.完毕才会放弃对象监视器。 wait()与sleep()的区别 关于这两者已经在上.进.详细的说明,这.就做个概括好了:

sleep()来.Thread类，和wait()来.Object类。调.sleep().法的过程中，线程不会释放对象锁。.调. wait .法线程会释放对象锁

sleep()睡眠后不出让系统资源，wait让其他线程可以占.CPU

sleep(milliseconds)需要指定.个睡眠时间，时间.到会.动唤醒..wait()需要配合notify()或者notifyAll()使.

为什么wait,nofity和nofityAll这些.法不放在Thread类当中 .个很明显的原因是JAVA提供的锁是对象级的.不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调.对象中的wait().法就有意义了。如果wait().法定义在Thread类中，线程正在等待的是哪个锁就不明显了。简单的说，由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象。 怎么唤醒.个阻塞的线程 如果线程是因为调.了wait()、sleep()或者join().法.导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，.能为.，因为IO是

操作系统实现的，Java代码并没有办法直接接触到操作系统。 什么是多线程的上下.切换 多线程的上下.切换是指CPU控制权由.个已经正在运.的线程切换到另外.个就绪并等待获取CPU执.权的线程的过程。 synchronized和ReentrantLock的区别 synchronized是和if、else、for、while.样的关键字，ReentrantLock是类，这是.者的本质区别。既然ReentrantLock是类，那么它就提供了.synchronized更多更灵活的特性，可以被继承、可以有.法、可以有各种各样的类变量，ReentrantLock .synchronized的扩展性体现在.点上： （1）ReentrantLock可以对获取锁的等待时间进.设置，这样就避免了死锁（2）ReentrantLock可以获取各种锁的信息（3）ReentrantLock可以灵活地实现多路通知 另外，.者的锁机制其实也是不.样的:ReentrantLock底层调.的是Unsafe的park.法加锁，synchronized操作的应该是对象头中markword。 FutureTask是什么 这个其实前.有提到过，FutureTask表..个异步运算的任务。FutureTask..可以传..个Callable的具体实现类，可以对这个异步运算的任务的结果进.等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接.的实现类，所以FutureTask也可以放.线程池中。 .个线程如果出现了运.时异常怎么办？如果这个异常没有被捕获的话，这个线程就停.执.了。另外重要的.点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被.即释放。 Java当中有哪.种锁

.旋锁: .旋锁在JDK1.6之后就默认开启了。基于之前的观察，共享数据的锁定状态只会持续很短的时间，为了这..段时间.去挂起和恢复线程有点浪费，所以这.就做了.个处理，让后.请求锁的那个线程在稍等.会，但是不放弃处理器的执.时间，看看持有锁的线程能否快速释放。为了让线程等待，所以需要让线程执..个忙循环也就是.旋操作。在jdk6之后，引.了.适应的.旋锁，也就是等待的时间不再固定了，.是由上.次在同.个锁上的.旋时间及锁的拥有者状态来决定。

偏向锁:在JDK1.之后引.的.项锁优化，.的是消除数据在.竞争情况下的同步原语。进.步提升程序的运.性能。偏向锁就是偏.的偏，意思是这个锁会偏向第.个获得他的线程，如果接下来的执.过程中，改锁没有被其他线程获取，则持有偏向锁的线程将永远不需要再进.同步。偏向锁可以提.带有同步但.竞争的程序性能，也就是说他并不.定总是对程序运.有利，如果程序中.多数的锁都是被多个不同的线程访问，那偏向模式就是多余的，在具体问题具体分析的前提下，可以考虑是否使.偏向锁。

轻量级锁:为了减少获得锁和释放锁所带来的性能消耗，引.了"偏向锁"和"轻量级锁"，所以在Java SE1.6.锁.共有四种状态，.锁状态，偏向锁状态，轻量级锁状态和重量级锁状态，它会随着竞争情况逐渐升级。锁可以升级但不能降级，意味着偏向锁升级成轻量级锁后不能降级成偏向锁。

如何在两个线程间共享数据 通过在线程之间共享对象就可以了，然后通过wait/notify/notifyAll、await/signal/signalAll进.唤起和等待，..说阻塞队列 BlockingQueue就是为线程之间共享数据.设计的。 如何正确的使.wait()？使.if还是while？wait().法应该在循环调.，因为当线程获取到CPU开始执.的时候，其他条件可能还没有满.，所以在处理前，循环检测条件是否满.会更好。下.是.段标准的使.wait和notify.法的代码： synchronized(obj) {while(conditiondoesnothold)obj.wait();//(Releaseslock,andreacquiresonwakeup)... //Performactionappropriatetocondition } 什么是线程局部变量ThreadLocal 线程局部变量是局限于线程内部的变量，属于线程..所有，不在多个线程间共享。Java提供ThreadLocal类来.持线程局部变量，是.种实现线程安全的.式。但是在管理环境下（如web服务器）使.线程局部变量的时候要特别..，在这种情况下，.作线程的.命周期.任何应.变量的.命周期都要.。任何线程局部变量.旦在.作完成后没有释放，Java应.就存在内存泄露的.险。 ThreadLoal的作.是什么? 简单说ThreadLocal就是.种以空间换时间的做法在每个Thread..维护了.个ThreadLocal.ThreadLocalMap把数据进.隔离，数据不共享，.然就没有线程安全..的问题了。 .产者消费者模型的作.是什么？（1）通过平衡.产者的.产能.和消费者的消费能.来提升整个系统的运.效率，这是.产者消费者模型最重要的作.。（2）解耦，这是.产者消费者模型附带的作.，解耦意味着.产者和消费者之间的联系少，联系越少越可以独.发展.不需要收到相互的制约。 写.个.产者-消费者队列 可以通过阻塞队列实现，也可以通过wait-notify来实现。使.阻塞队列来实现 //消费者 publicclassProducerimplementsRunnable{privatefinalBlockingQueuequeue; publicProducer(BlockingQueueq) {this.queue=q;} @Override publicvoidrun(){ try{while(true){Thread.sleep(1000); //模拟耗时 queue.put(produce());}}catch(InterruptedExceptione){}} privateintproduce() {intn=newRandom().nextInt(10000);System.out.println("Thread:"+Thread.currentThread().getId()+"produce:"+n); returnn;}} //消费者 publicclassConsumerimplementsRunnable{privatefinalBlockingQueuequeue; publicConsumer(BlockingQueueq){this.queue=q;} @Override publicvoidrun(){while(true){try{Thread.sleep(2000) //模拟耗时 consume(queue.take());}catch(InterruptedExceptione){}}} privatevoidconsume(Integern) {System.out.println("Thread:"+Thread.currentThread().getId()+"consume:"+n);}} //测试 publicclassMain{ publicstaticvoidmain(String[]args) {BlockingQueuequeue=newArrayBlockingQueue(100);Producerp=newProducer(queue);Consumerc1=newConsumer(queue);Consumerc2=newConsumer(queue); newThread(p).start(); newThread(c1).start(); newThread(c2).start();}} 使.wait-notify来实现 该种.式应该最经典，这.就不做说明了。 如果你提交任务时，线程池队列已满，这时会发.什么 如

果你使.的LinkedBlockingQueue，也就是.界队列的话，没关系，继续添加任务到阻塞队列中等待执.，因为LinkedBlockingQueue可以近乎认为是.个.穷.的队列，可以.限存放任务；如果你使.的是有界队列..说ArrayBlockingQueue的话，任务.先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，则会使.拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy。 为什么要使.线程池 避免频繁地创建和销毁线程，达到线程对象的重.。另外，使.线程池还可以根据项.灵活地控制并发的数.。 java中.到的线程调度算法是什么 抢占式。.个线程.完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出.个总的优先级并分配下.个时间.给某个线程执.。 Thread.sleep(0)的作.是什么 由于Java采.抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级.较低的线程也能获取到CPU控制权，可以使.Thread.sleep(0).动触发.次操作系统分配时间.的操作，这也是平衡CPU控制权的.种操作。 什么是CAS CAS，全称为CompareandSwap，即.较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS.定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是.个不会变的值A，只要某次CAS操作失败，永远都不可能成功。 什么是乐观锁和悲观锁 乐观锁：乐观锁认为竞争不总是会发.，因此它不需要持有锁，将.较-替换这两个动作作为.个原.操作尝试去修改内存中的变量，如果失败则表.发.冲突，那么就应该有相应的重试逻辑。 悲观锁：悲观锁认为竞争总是会发.，因此每次对某资源进.操作时，都会持有.个独占的锁，就像synchronized，不管三七...，直接上了锁就操作资源了。 ConcurrentHashMap的并发度是什么? ConcurrentHashMap的并发度就是segment的..，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最.优势，任何情况下，Hashtable能同时有两条线程获取 Hashtable中的数据吗？ ConcurrentHashMap的.作原理 ConcurrentHashMap在jdk1.6和jdk1.8实现原理是不同的。 jdk1.6: ConcurrentHashMap是线程安全的，但是与Hashtablea相.，实现线程安全的.式不同。Hashtable是通过对hash表结构进.锁定，是阻塞式的，当.个线程占有这个锁时，其他线程必须阻塞等待其释放锁。ConcurrentHashMap是采.分离锁的.式，它并没有对整个hash表进.锁定，.是局部锁定，也就是说当.个线程占有这个局部锁时，不影响其他线程对hash表其他地.的访问。具体实现:ConcurrentHashMap内部有.个Segment. jdk1.8 在jdk8中，ConcurrentHashMap不再使.Segment分离锁，.是采..种乐观锁CAS算法来实现同步问题，但其底层还是"数组+链表->红.树"的实现。 CyclicBarrier和CountDownLatch区别 这两个类.常类似，都在java.util.concurrent下，都可以.来表.代码运.到某个点上，.者的区别在于：

CyclicBarrier的某个线程运.到某个点上之后，该线程即停.运.，直到所有的线程都到达了这个点，所有线程才重新运.；CountDownLatch则不是，某线程运.到某个点上之后，只是给某个数值-1.已，该线程继续运.。 CyclicBarrier只能唤起.个任务，CountDownLatch可以唤起多个任务 CyclicBarrier可重.，CountDownLatch不可重.，计数值为0该CountDownLatch就不可再.了。 java中的++操作符线程安全么? 不是线程安全的操作。它涉及到多个指令，如读取变量值，增加，然后存储回内存，这个过程可能会出现多个线程交差。 你有哪些多线程开发良好的实践?

给线程命名最.化同步范围优先使.volatile尽可能使.更.层次的并发.具..wait和notify()来实现线程通信,如BlockingQueue,Semeaphore优先使.并发容器..同步容器.考虑使.线程池 关于volatile关键字 可以创建Volatile数组吗? Java中可以创建 volatile类型数组，不过只是.个指向数组的引.，.不是整个数组。如果改变引.指向的数组，将会受到volatile的保护，但是如果多个线程同时改变数组的元素，volatile标.符就不能起到之前的保护作.了。 volatile能使得.个.原.操作变成原.操作吗? .个典型的例.是在类中有.个long类型的成员变量。如果你知道该成员变量会被多个线程访问，如计数器、价格等，你最好是将其设置为volatile。为什么？因为Java中读取long类型变量不是原.的，需要分成两步，如果.个线程正在修改该long变量的值，另.个线程可能只能看到该值的.半（前32位）。但是对.个volatile型的long或double变量的读写是原..。 .种实践是.volatile修饰long和double变量，使其能按原.类型来读写。double和long都是64位宽，因此对这两种类型的读是分为两部分的，第.次读取第.个32位，然后再读剩下的32位，这个过程不是原.的，但Java中volatile型的 long或 double变量的读写是原.的。volatile修复符的另.个作.是提供内存屏障（memory barrier），例如在分布式框架中的应.。简单的说，就是当你写.个volatile变量之前，Java内存模型会插..个写屏障（writebarrier），读.个volatile变量之前，会插..个读屏障（read barrier）。意思就是说，在你写.个 volatile域时，能保证任何线程都能看到你写的值，同时，在写之前，也能保证任何数值的更新对所有线程是可.的，因为内存屏障会将其他所有写的值更新到缓存。 volatile类型变量提供什么保证? volatile主要有两..的作.:1.避免指令重排2.可.性保证.例如，JVM或者 JIT为了获得更好的性能会对语句重排序，但是 volatile类型变量即使在没有同步块的情况下赋值也不会与其他语句重排序。volatile提供 happens-before的保证，确保.个线程的修改能对其他线程是可.的。某些情况下，volatile还能提供原.性，如读64位数据类型，像 long和double都不是原.的(低32位和.32位)，但volatile类型的double和long就是原.的。 关于集合 Java中的集合及其继承关系 关于集合的体系是每个.都应该烂熟于.的,尤其是对我们经常使.的List,Map的原理更该如此.这.我们看这张图即可: 更多内容可.集合类总结： http://write.blog.csdn.net/postedit/40826423 poll().法和remove().法区别? poll()和remove()都是从队列中取出.个元素，但是poll()在获取元素失败的时候会返回空，但是remove()失败的时候会抛出异常。 LinkedHashMap和PriorityQueue的区别 PriorityQueue是.个优先级队列,保证最.或者最低优先级的的元素总是

在队列头部，但是LinkedHashMap维持的顺序是元素插.的顺序。当遍历.个PriorityQueue时，没有任何顺序保证，但是LinkedHashMap课保证遍历顺序是元素插.的顺序。 WeakHashMap与HashMap的区别是什么? WeakHashMap的.作与正常的HashMap类似，但是使.弱引.作为key，意思就是当key对象没有任何引.时，key/value将会被回收。 ArrayList和LinkedList的区别? 最明显的区别是ArrrayList底层的数据结构是数组，.持随机访问，.LinkedList的底层数据结构是双向循环链表，不.持随机访问。使.下标访问.个元素，ArrayList的时间复杂度是O(1)，.LinkedList是O(n)。 ArrayList和Array有什么区别?

Array可以容纳基本类型和对象，.ArrayList只能容纳对象。

Array是指定..的，.ArrayList..是固定的

ArrayList和HashMap默认..? 在 Java7中，ArrayList的默认..是 10个元素，HashMap的默认..是16个元素（必须是2的幂）。这就是 Java7中 ArrayList和HashMap类的代码.段。
privatestaticfinalintDEFAULT_CAPACITY=10;//fromHashMap.java
JDK7staticfinalintDEFAULT_INITIAL_CAPACITY=1<<4;//aka16 Comparator和Comparable的区别? Comparable接..于定义对象的.然顺序，.comparator通常.于定义..定制的顺序。Comparable总是只有.个，但是可以有多个comparator来定义对象的顺序。 如何实现集合排序? 你可以使.有序集合，如TreeSet或TreeMap，你也可以使.有顺序的的集合，如list，然后通过Collections.sort()来排序。 如何打印数组内容 你可以使.Arrays.toString()和Arrays.deepToString().法来打印数组。由于数组没有实现toString().法，所以如果将数组传递给System.out.println().法，将.法打印出数组的内容，但是Arrays.toString()可以打印每个元素。 LinkedList的是单向链表还是双向? 双向循环列表，具体实现..查阅源码。 TreeMap是实现原理 采.红.树实现，具体实现..查阅源码。 遍历ArrayList时如何正确移除.个元素 该问题的关键在于.试者使.的是ArrayList的remove()还是Iterator的remove().法。这有.段.例代码，是使.正确的.式来实现在遍历的过程中移除元素，.不会出现ConcurrentModificationException异常的.例代码。 什么是ArrayMap?它和HashMap有什么区别? ArrayMap是AndroidSDK中提供的，.Android开发者可以略过。ArrayMap是.两个数组来模拟map，更少的内存占.空间,更.的效率。具体参考这篇.章：ArrayMapVSHashMap：http://lvable.com/?p=217] HashMap的实现原理 1. HashMap概述：HashMap是基于哈希表的Map接.的.同步实现。此实现提供所有可选的映射操作，并允许使.null值和null键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

2.

HashMap的数据结构：在java编程语.中，最基本的结构就是两种，.个是数组，另外.个是模拟指针（引.），所有的数据结构都可以.这两个基本结构来构造的，HashMap也不例外。HashMap实际上是.个“链表散列”的数据结构，即数组和链表的结合体。

当我们往Hashmap中put元素时,.先根据key的hashcode重新计算hash值,根绝hash值得到这个元素在数组中的位置 (下标),如果该数组在该位置上已经存放了其他元素,那么在这个位置上的元素将以链表的形式存放,新加.的放在链头,最先加.的放.链尾.如果数组中该位置没有元素,就直接将该元素放到数组的该位置上. 需要注意Jdk1.8中对HashMap的实现做了优化,当链表中的节点数据超过.个之后,该链表会转为红.树来提.查询效率,从原来的O(n)到O(logn) 你了解Fail-Fast机制吗？ Fail-Fast即我们常说的快速失败，更多内容参看fail-fast机制：
http://blog.csdn.net/chenssy/article/details/38151189 Fail-fast和Fail-safe有什么区别 Iterator的fail-fast属性与当前的集合共同起作.，因此它不会受到集合中任何改动的影响。Java.util包中的所有集合类都被设计为fail->fast的，.java.util.concurrent中的集合类都为fail-safe的。当检测到正在遍历的集合的结构被改变时，Fail-fast迭代器抛出ConcurrentModi.cationException，.fail-safe迭代器从不抛出 ConcurrentModi.cationException。 关于.期SimpleDateFormat是线程安全的吗? .常不幸，DateFormat的所有实现，包括SimpleDateFormat都不是线程安全的，因此你不应该在多线程序中使.，除.是在对外线程安全的环境中使.，如将SimpleDateFormat限制在ThreadLocal中。如果你不这么做，在解析或者格式化.期的时候，可能会获取到.个不正确的结果。因此，从.期、时间处理的所有实践来说，我强.推荐joda-time库。 如何格式化.期? Java中，可以使.SimpleDateFormat类或者joda-time库来格式.期。DateFormat类允许你使.多种流.的格式来格式化.期。参.答案中的.例代码，代码中演.了将.期格式化成不同的格式，如dd-MM-yyyy或ddMMyyyy。 关于异常 简单描述java异常体系 相.没有.不了解异常体系,关于异常体系的更多信息可以. .话异常机制：
http://blog.csdn.net/dd864140130/article/details/42504189 什么是异常链 详情直接参.上.的.话异常机制，不做解释了。 throw和throws的区别 throw.于主动抛出java.lang.Throwable类的.个实例化对象，意思是说你可以通过关键字throw抛出.个 Error或者.个Exception，如：thrownew IllegalArgumentException(izemustbemultipleof2") “s ,.throws的作.是作为.法声明和签名的.部分，.法被抛出相应的异常以便调.者能处理。Java中，任何未处理的受检查异常强制在throws.句中声明。 关于序列化 Java中，Serializable与Externalizable的区别 Serializable接.是.个序列化Java类的接.，以

便于它们可以在.络上传输或者可以将它们的状态保存在磁盘上，是JVM内嵌的默认序列化.式，成本.、脆弱.且不安全。Externalizable允许你控制整个序列化过程，指定特定的.进制格式，增加安全机制。 关于JVM JVM特性 平台.关性.Java语.的.个.常重要的特点就是与平台的.关性。.使.Java虚拟机是实现这.特点的关键。.般的.级语.如果要在不同的平台上运.，.少需要编译成不同的.标代码。.引.Java语.虚拟机后，Java语.在不同平台上运.时不需要重新编译。Java语.使.模式Java虚拟机屏蔽了与具体平台相关的信息，使得Java语.编译程序只需.成在Java虚拟机上运.的.标代码（字节码），就可以在多种平台上不加修改地运.。Java虚拟机在执.字节码时，把字节码解释成具体平台上的机器指令执.。 简单解释.下类加载器 有关类加载器.般会问你四种类加载器的应.场景以及双亲委派模型, 更多的内容参看深.理解JVM加载器：http://blog.csdn.net/dd864140130/article/details/49817357 简述堆和栈的区别 VM中堆和栈属于不同的内存区域，使..的也不同。栈常.于保存.法帧和局部变量，.对象总是在堆上分配。栈通常都.堆.，也不会在多个线程之间共享，.堆被整个JVM的所有线程共享。 简述JVM内存分配

基本数据类型.变量和对象的引.都是在栈分配的。堆内存.来存放由new创建的对象和数组。类变量（static修饰的变量），程序在.加载的时候就在堆中为类变量分配内存，堆中的内存地址存放在栈中。实例变量：当你使.java关键字new的时候，系统在堆中开辟并不.定是连续的空间分配给变量，是根据零散的堆内存地址，通过哈希算法换算为..串数字以表征这个变量在堆中的"物理位置",实例变量的.命周期.当实例变量的引.丢失后，将被GC（垃圾回收器）列.可回收"名单"中，但并不是.上就释放堆中内存。局部变量:由声明在某.法，或某代码段.（.如for循环），执.到它的时候在栈中开辟内存，当局部变量.但脱离作.域，内存.即释放。 其他 java当中采.的是.端还是.端? XML解析的.种.式和特点DOM,SAX,PULL三种解析.式：

DOM:消耗内存：先把xml.档都读到内存中，然后再.DOM API来访问树形结构，并获取数据。这个写起来很简单，但是很消耗内存。要是数据过.，.机不够.逼，可能.机直接死机 SAX:解析效率.，占.内存少，基于事件驱动的：更加简单地说就是对.档进.顺序扫描，当扫描到.档(document)开始与结束、元素(element)开始与结束、.档(document)结束等地.时通知事件处理函数，由事件处理函数做相应动作，然后继续同样的扫描，直..档结束。 PULL:与 SAX类似，也是基于事件驱动，我们可以调.它的next（）.法，来获取下.个解析事件（就是开始.档，结束.档，开始标签，结束标签），当处于某个元素时可以调.XmlPullParser的getAttributte().法来获取属性的值，也可调.它的nextText()获取本节点的值。 JDK1.7特性 然JDK1.7不像JDK5和8.样的.版本，但是，还是有很多新的特性，如try-with-resource语句，这样你在使.流或者资源的时候，就不需要.动关闭，Java会.动关闭。Fork-Join池某种程度上实现 Java版的 Map-reduce。允许 Switch中有String变量和.本。菱形操作符(<>).于类型推断，不再需要在变量声明的右边申明泛型，因此可以写出可读写更强、更简洁的代码。 JDK1.8特性 java8在Java历史上是.个开创新的版本，下.JDK8中5个主要的特性：Lambda表达式，允许像对象.样传递匿名函数 StreamAPI，充分利.现代多核CPU，可以写出很简洁的代码Date与TimeAPI，最终，有.个稳定、简单的.期和时间库可供你使.扩展.法，现在，接.中可以有静态、默认.法。重复注解，现在你可以将相同的注解在同.类型上使.多次。 Maven和ANT有什么区别? 虽然两者都是构建.具，都.于创建 Java应.，但是 Maven做的事情更多，在基于"约定优于配置"的概念下，提供标准的Java项.结构，同时能为应..动管理依赖（应.中所依赖的JAR.件。 JDBC最佳实践

优先使.批量操作来插.和更新数据使.PreparedStatement来避免SQL漏洞使.数据连接池通过列名来获取结果集 IO操作最佳实践

使.有缓冲的IO类,不要单独读取字节或字符使.NIO和NIO2或者AIO,..BIO在.nally中关闭流使.内存映射.件获取更快的IO来源于： https://blog.csdn.net/dd864140130/article/details/55833087 -END- 推荐阅读 1.955不加班的公司名单：955.WLB

2.

8岁上海.学.B站教编程惊动苹果 3.我在华为做外包的真实经历！

4.什么是.致性 Hash算法?

5.团队开发中 Git最佳实践

喜欢.章，点个在看

阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 在阿..了5年，.试个.公司挂了... Java后端 2019-09-12 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

来源：公众号51CTO官微 .名阿.员.在互联.社区吐槽，称...作经验丰富，在阿..作了五年，去.试.个.互联.创业公司竟然挂了，真是..吐槽： 我就郁闷了，在阿..作五年，去.试某公司，上来啥都不问，就两道算法题我没有第.时间给出最优解，

想了.会.才做出来，结果就把我挂了，.作那么多年了，还这样.试也是令.醉了。

很快，这样的吐槽就获得了不少同.的关注。 有不少程序员表.这种公司不去也罢，仗着..是外资，以为有多..上，在其本国就是个low货，.跑到这边来就变得..上，这样的企业不稀罕去，还不如国内的互联.公司：这种公司不去也罢，能在阿.呆5年，这就不..试，直接进，你要了解这种公司都是这个.格；外企主要就是.算法.，跟国内企业完全两种.试套路。 不过也有同.表.：美企都是这种.试.格，即使你刷过了.遍leetcode，你也不.定能现场解出.道变形题，我觉得.起问项.经验、问那些搜索.下就能搜到答案的吹..题.，问算法题要靠谱得多。

.部分同.都对这样的企业嗤之以.：你还认真去做，.且做出来了，要我看到出这种题就.了，最烦上来拿卷.做的公司了，招.不多，待遇.般，要求还很.，不去也罢，不知道是什么给他们的企业，现在国内的互联.不.国外差，没必要去外资企业。

各位朋友.是如何看待此事的呢？说说你们的看法，欢迎底部留.！ 推荐阅读 1. Java实现QQ登陆

2.请给Spr ingBoot多.些内存 3.如何从零搭建百亿流量系统?

4.

惊了！原来Web发展历史是这样的

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

...试官最喜欢问的.试难点 Java后端 2019-10-22

链接|toutiao.com/a6749111386116194830 .平常经常使.外键和外键和级联吗，可以说说你对它们的理解吗？ 对于外键和级联，阿.巴巴开发.册这样说到： 【强制】不得使.外键与级联，.切外键概念必须在应.层解决。说明:以学.和成绩的关系为例，学.表中的student_id是主键，那么成绩表中的student_id则为外键。如果更新学.表中的 student_id，同时触发成绩表中的student_id更新，即为级联更新。外键与级联更新适.于单机低并发，不适合分布式、.并发集群;级联更新是强阻塞，存在数据库更新.暴的.险;外键影响数据库的插.速度 为什么不要.外键呢？.部分.可能会这样回答： 增加了复杂性：a.每次做DELETE或者UPDATE都必须考虑外键约束，会导致开发的时候很痛苦,测试数据极为不.便;b.外键的主从关系是定的，假如那天需求有变化，数据库中的这个字段根本不需要和其他表有关联的话就会增加很多.烦。增加了额外.作：数据库需要增加维护外键的.作，.如当我们做.些涉及外键字段的增，删，更新操作之后，需要触发相关操作去检查，保证数据的的.致性和正确性，这样会不得不消耗资源；（个.觉得这个不是不.外键的原因，因为即使你不使.外键，你在应.层.也还是要保证的。所以，我觉得这个影响可以忽略不计。）外键还会因为需要请求对其他表内部加锁.容易出现死锁情况；对分不分表不友好：因为分库分表下外键是.法.效的。 ..... 我个.觉得上.这种回答不是特别的全.，只是说了外键存在的.个常.的问题。实际上，我们知道外键也是有很多好处的，.如： 1. 保证了数据库数据的.致性和完整性；

2.

级联操作.便，减轻了程序代码量；

3. ..... 所以说，不要.股脑的就抛弃了外键这个概念，既然它存在就有它存在的道理，如果系统不涉及分不分表，并发量不是很.的情况还是可以考虑使.外键的。 我个.是不太喜欢外键约束，.较喜欢在应.层去进.相关操作。 .解释.下什么是池化设计思想。什么是数据库连接池?为什么需要数据库连接池? 池话设计应该不是.个新名词。我们常.的如java线程池、jdbc连接池、redis连接池等就是这类设计的代表实现。这种设计会初始预设资源，解决的问题就是抵消每次获取资源的消耗，如创建线程的开销，获取远程连接的开销等。就好.你去.堂打饭，打饭的.妈会先把饭盛好.份放那.，你来了就直接拿着饭盒加菜即可，不.再临时.盛饭.打菜，效率就.了。除了初始化资源，池化设计还包括如下这些特征：池.的初始值、池.的活跃值、池.的最.值等，这些特征可以直接映射到java线程池和数据库连接池的成员属性中。 数据库连接本质就是.个socket的连接。数据库服务端还要维护.些缓存和..权限信息之类的所以占.了.些内存。我们可以把数据库连接池是看做是维护的数据库连接的缓存，以便将来需要对数据库的请求时可以重.这些连接。为每个..打开和维护数据库连接，尤其是对动态数据库驱动的.站应.程序的请求，既昂贵.浪费资源。在连接池中，创建连接后，将其放置在池中，并再次使.它，因此不必建.新的连接。如果使.了所有连接，则会建..个新连接并将其添加到池中。连接池还减少了..必须等待建.与数据库的连接的时间。 三分库分表之后,id主键如何处理？ 因为要是分成多个表之后，每个表都是从1开始累加，这样是不对的，我们需要.个全局唯.的id来.持。 .成全局id有下.这.种.式： UUID：不适合作为主键，因为太.了，并且.序不可读，查询效率低。.较适合.于.成唯.的名字的标..

如.件的名字。数据库.增 id:两台数据库分别设置不同步.，.成不重复ID的策略来实现.可.。这种.式.成的 id有序，但是需要独.部署数据库实例，成本.，还会有性能瓶颈。利.redis.成id:性能.较好，灵活.便，不依赖于数据库。但是，引.了新的组件造成系统更加复杂，可.性降低，编码更加复杂，增加了系统成本。 Twitter的snowflake算法：Github地址：https://github.com/twitter-archive/snowflake。美团的Leaf分布式ID.成系统：Leaf是美团开源的分布式ID.成器，能保证全局唯.性、趋势递增、单调递增、信息安全，..也提到了.种分布式.案的对.，但也需要依赖关系数据库、Zookeeper等中间件。感觉还不错。美团技术团队的.篇.章：https://tech.meituan.com/2017/04/21/mt-leaf.html。 ..... 四.过BigDecimal吗？为啥要.它？ 4.1BigDecimal的.处 《阿.巴巴Java开发.册》中提到：浮点数之间的等值判断，基本数据类型不能.==来.较，包装数据类型不能.equals来判断。具体原理和浮点数的编码.式有关，这.就不多提了，我们下.直接上实例： floata=1.0f-0.9f; floatb=0.9f-0.8f; System.out.println(a);//0.100000024 System.out.println(b);//0.099999964 System.out.println(a==b);//false 具有基本数学知识的我们很清楚的知道输出并不是我们想要的结果（精度丢失），我们如何解决这个问题呢？.种很常.的.法是：使.使.BigDecimal来定义浮点数的值，再进.浮点数的运算操作。 BigDecimala=newBigDecimal("1.0"); BigDecimalb=newBigDecimal("0.9"); BigDecimalc=newBigDecimal("0.8"); BigDecimalx=a.subtract(b);//0.1 BigDecimaly=b.subtract(c);//0.1 System.out.println(x.equals(y));//true

4.

2BigDecimal的...较

a.compareTo(b):返回-1表..于，0表.等于，1表..于。 BigDecimala=newBigDecimal("1.0"); BigDecimalb=newBigDecimal("0.9"); System.out.println(a.compareTo(b));//1

4.

3BigDecimal保留.位.数

通过setScale.法设置保留.位.数以及保留规则。保留规则有挺多种，不需要记，IDEA会提.。 BigDecimalm=newBigDecimal("1.255433"); BigDecimaln=m.setScale(3,BigDecimal.ROUND_HALF_DOWN); System.out.println(n);//1.255 4.4BigDecimal的使.注意事项 注意：我们在使.BigDecimal时，为了防.精度丢失，推荐使.它的BigDecimal(String)构造.法来创建对象。《阿.巴巴Java开发.册》对这部分内容也有提到如下图所.。

《阿.巴巴Java开发.册》对这部分BigDecimal的描述 4.5总结 BigDecimal主要.来操作（.）浮点数，BigInteger主要.来操作.整数（超过long类型）。 BigDecimal的实现利.到了BigInteger,所不同的是BigDecimal加.了.数位的概念 五Java中IO流分为.种? 按照流的流向分，可以分为输.流和输出流； 按照操作单元划分，可以划分为字节流和字符流；按照流的..划分为节点流和处理流。 Javalo流共涉及40多个类，这些类看上去很杂乱，但实际上很有规则，.且彼此之间存在.常紧密的联系，Java I0流的40多个类都是从如下4个抽象类基类中派.出来的。 InputStream/Reader:所有的输.流的基类，前者是字节输.流，后者是字符输.流。 OutputStream/Writer:所有输出流的基类，前者是字节输出流，后者是字符输出流。按操作.式分类结构图：

IO-操作.式分类 按操作对象分类结构图：

IO-操作对象分类 六既然有了字节流,为什么还要有字符流? 问题本质想问：不管是.件读写还是.络发送接收，信息的最.存储单元都是字节，那为什么I/O流操作要分为字节流操作和字符流操作呢？ 回答：字符流是由Java虚拟机将字节转换得到的，问题就出在这个过程还算是.常耗时，并且，如果我们不知道编码类型就很容易出现乱码问题。所以，I/O流就.脆提供了.个直接操作字符的接.，.便我们平时对字符进.流操作。如果.频.件、图.等媒体.件.字节流.较好，如果涉及到字符的话使.字符流.较好。 tips：.家可以关注微信公众号：Java后端，获取更多优秀博.推送。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.感受 Lambda之美，推荐收藏，需要时查阅

2.如何优雅的导出 Excel

3..艺互联.公司 vs .逼互联.公司

4.

Java开发中常.的 4种加密.法

5.团队开发中 Git最佳实践

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

如果我是.试官，我会问你Spring那些问题？ 静默虚空Java后端 2019-11-29 点击上.Java后端，选择设为星标 优质.章，及时送达

作者|静默虚空来源|https://urlify.cn/R3ymqq 1..般问题 1. 1.不同版本的SpringFramework有哪些主要功能？1.2.什么是SpringFramework?

1.3.列举Spr ingFramework的优点。 1.4.SpringFramework有哪些不同的功能？ 1.5.SpringFramework中有多少个模块，它们分别是什么?

1.

6.什么是Spring配置.件?

1.

7.Spring应.程序有哪些不同组件?

1.8.使.Spr ing有哪些.式?

2.依赖注.（ Ioc）

2.

1.什么是SpringIOC容器?

2.

2.什么是依赖注.?

2.

3.可以通过多少种.式完成依赖注.?

2.

4.区分构造函数注.和setter注.。

2.

5.spring中有多少种IOC容器?

2.

6.区分BeanFactory和ApplicationContext。

2.7.列举 IoC的.些好处。 2.8.SpringIoC的实现机制。

3.Beans

3.

1.什么是springbean?

3.

2.spring提供了哪些配置.式?

3.

3.spring.持集中beanscope?

3.

4.springbean容器的.命周期是什么样的?

3.

5.什么是spring的内部bean?

3.

6.什么是spring装配

3.

7..动装配有哪些.式? 3.8..动装配有什么局限?

4.注解

4.

1.你.过哪些重要的Spring注解?

4.

2.如何在spring中启动注解装配? 4.3.@Component,@Controller,@Repository,@Service有何区别?

4.

4.@Required注解有什么.?

4.

5.@Autowired注解有什么.?

4.

6.@Qualifier注解有什么.?

4.

7.@RequestMapping注解有什么.?

5.数据访问

5.

1.springDAO有什么.?

5.2.列举Spr ingDAO抛出的异常。 5.3.springJDBCAPI中存在哪些类?

5.4.使.Spr ing访问Hibernate的.法有哪些?

5.5.列举spr ing.持的事务管理类型

5.

6.spring.持哪些ORM框架

6.AOP

6.

1.什么是AOP?

6.

2.AOP中的Aspect、Advice、Pointcut、JointPoint和Advice参数分别是什么?

6.

3.什么是通知（Advice）?

6.

4.有哪些类型的通知（Advice）?

6.

5.指出在springaop中concern和cross-cuttingconcern的不同之处。

6.

6.AOP有哪些实现.式? 6.7.SpringAOPandAspectJAOP有什么区别?

6.

8.如何理解Spring中的代理?

6.

9.什么是编织（Weaving）?

7.MVC

7.

1.SpringMVC框架有什么.?

7.2.描述.下D ispatcherServlet的.作流程

7.

3.介绍.下WebApplicationContext

8.资料 #.般问题 1.1.不同版本的SpringFramework有哪些主要功能? Version Feature Spring2.5发布于2007年。这是第.个.持注解的版本。 发布于2009年。它完全利.了Java5中的改进，并为Spring3.0JEE6提供了.持。 Spring4.0发布于2013年。这是第.个完全.持JAVA8的版本。 1.2.什么是SpringFramework? Spring是.个开源应.框架，旨在降低应.程序开发的复杂度。 它是轻量级、松散耦合的。 它具有分层体系结构，允许..选择组件，同时还为J2EE应.程序开发提供了.个有凝聚.的框架。 它可以集成其他框架，如Structs、Hibernate、EJB等，所以.称为框架的框架。 1.3.列举SpringFramework的优点。 由于SpringFrameworks的分层架构，..可以.由选择..需要的组件。 SpringFramework.持POJO(PlainOldJavaObject)编程，从.具备持续集成和可测试性。由于依赖注.和控制反转，JDBC得以简化。它是开源免费的。 1.4.SpringFramework有哪些不同的功能? 轻量级-Spring在代码量和透明度..都很轻便。 IOC-控制反转 AOP-.向切.编程可以将应.业务逻辑和系统服务分离，以实现.内聚。容器-Spring负责创建和管理对象（Bean）的.命周期和配置。 MVC-对web应.提供了.度可配置性，其他框架的集成也.分.便。事务管理-提供了.于事务管理的通.抽象层。Spring的事务.持也可.于容器较少的环境。JDBC异常-Spring的JDBC抽象层提供了.个异常层次结构，简化了错误处理策略。 1.5.SpringFramework中有多少个模块，它们分别是什么?

Spring核.容器.该层基本上是SpringFramework的核.。它包含以下模块： SpringCore SpringBeanSpEL(SpringExpressionLanguage)SpringContext 数据访问/集成.该层提供与数据库交互的.持。它包含以下模块：
JDBC(JavaDataBaseConnectivity)ORM(ObjectRelationalMapping)OXM(ObjectXMLMappers)JMS(JavaMessagingService)Transaction Web.该层提供了创建Web应.程序的.持。它包含以下模块： Web Web.Servlet Web.Socket Web.Portlet AOP.该层.持.向切.编程 Instrumentation.该层为类检测和类加载器实现提供.持。 Test.该层为使.JUnit和TestNG进.测试提供.持。 .个杂项模块: Messaging.该模块为STOMP提供.持。它还.持注解编程模型，该模型.于从WebSocket客.端路由和处理STOMP消息 Aspects.该模块为与AspectJ的集成提供.持。 1. 6.什么是Spring配置.件?

Spring配置.件是XML.件。该.件主要包含类信息。它描述了这些类是如何配置以及相互引.的。但是，XML配置.件冗.且更加.净。如果没有正确规划和编写，那么在.项.中管理变得.常困难。

1.

7.Spring应.程序有哪些不同组件?

Spring应..般有以下组件： 接.-定义功能。Bean类-它包含属性，setter和getter.法，函数等。 Spring.向切.编程（AOP）-提供.向切.编程的功能。Bean配置.件-包含类的信息以及如何配置它们。..程序-它使.接.。 1.8.使.Spring有哪些.式? 使.Spring有以下.式： 作为.个成熟的SpringWeb应.程序。作为第三.Web框架，使.SpringFrameworks中间层。.于远程使.。作为企业级JavaBean，它可以包装现有的POJO（PlainOldJavaObjects）。 #依赖注.（Ioc） 1.什么是SpringIOC容器? Spring框架的核.是Spring容器。容器创建对象，将它们装配在.起，配置它们并管理它们的完整.命周期。Spring容器使.依赖注.来管理组成应.程序的组件。 容器通过读取提供的配置元数据来接收对象进.实例化，配置和组装的指令。该元数据可以通过XML，Java注解或Java代码提供。

2.

什么是依赖注.?

在依赖注.中，您不必创建对象，但必须描述如何创建它们。您不是直接在代码中将组件和服务连接在.起，.是描述配置.件中哪些组件需要哪些服务。由IoC容器将它们装配在.起。

3.

可以通过多少种.式完成依赖注.?

通常，依赖注.可以通过三种.式完成，即： * 构造函数注.

-

setter注.

-

接.注.

在SpringFramework中，仅使.构造函数和setter注.。 4. 区分构造函数注.和setter注.。

构造函数注.setter注.没有部分注.有部分注.不会覆盖setter属性会覆盖setter属性任意修改都会创建.个新实例任意修改不会创建.个新实例适.于设置很多属性适.于设置少量属性

5.

spring中有多少种IOC容器?

-

BeanFactory-BeanFactory就像.个包含bean集合的..类。它会在客.端要求时实例化bean。

-

ApplicationContext-ApplicationContext接.扩展了BeanFactory接.。它在BeanFactory基础上提供了.些额外的功能。

6.

区分BeanFactory和ApplicationContext。

BeanFactory ApplicationContext它使.懒加载它使.即时加载它使.语法显式提供资源对象它..创建和管理资源对象不.持国际化.持国际化不.持基于依赖的注解.持基于依赖的注解

7.列举 IoC的.些好处。

IoC的.些好处是： * 它将最.化应.程序中的代码量。

-

它将使您的应.程序易于测试，因为它不需要单元测试.例中的任何单例或JNDI查找机制。

-

它以最.的影响和最少的侵.机制促进松耦合。

-

它.持即时的实例化和延迟加载服务。

8.SpringIoC的实现机制。 Spring中的IoC的实现原理就是..模式加反射机制。.例： interfaceFruit{ publicabstractvoideat();}classAppleimplementsFruit{ publicvoideat(){System.out.println("Apple"); }}classOrangeimplementsFruit{ publicvoideat(){System.out.println("Orange"); }}classFactory{ publicstaticFruitgetInstance(StringClassName){Fruitf=null;try{ f= (Fruit)Class.forName(ClassName).newInstance();}catch(Exceptione){ e.printStackTrace();}returnf; }}classClient{ publicstaticvoidmain(String[]a){Fruitf=Factory.getInstance("io.github.dunwu.spring.Apple");if(f!=null){ f.eat();}}}

Beans

1.什么是springbean？ * 它们是构成..应.程序主.的对象。

*Bean由Spr ingIoC容器管理。

*它们由Spr ingIoC容器实例化，配置，装配和管理。

-

Bean是基于..提供给容器的配置元数据创建。

2.spring提供了哪些配置.式？ *基于xml配置 bean所需的依赖项和服务在XML格式的配置.件中指定。这些配置.件通常包含许多bean定义和特定于应.程序的配置选项。它们通常以bean标签开头。例如： <beanid="studentbean"class="org.edureka.firstSpring.StudentBean"> <propertyname="name"value="Edureka"> *基于注解配置 您可以通过在相关的类，.法或字段声明上使.注解，将bean配置为组件类本.，.不是使.XML来描述bean装配。默认情况 下，Spring容器中未打开注解装配。因此，您需要在使.它之前在Spring配置.件中启.它。例如： context:annotation-config/
*基于JavaAPI配置 Spring的Java配置是通过使.@Bean和@Configuration来实现。 1. @Bean注解扮演与元素相同的..。 2. @Configuration类允许通过简单地调.同.个类中的其他@Bean.法来定义bean间依赖关系。例如：

@Configuration publicclassStudentConfig{ @Bean publicStudentBeanmyStudent(){returnnewStudentBean();}} 3.spring.持集中beanscope？ Springbean.持5种scope： * Singleton-每个SpringIoC容器仅有.个单实例。

-

Prototype-每次请求都会产..个新的实例。

-

Request-每.次HTTP请求都会产..个新的实例，并且该bean仅在当前HTTP请求内有效。

-

Session-每.次HTTP请求都会产..个新的bean，同时该bean仅在当前HTTPsession内有效。

-

Global-session-类似于标准的HTTPSession作.域，不过它仅仅在基于portlet的web应.中才有意义。Portlet规范定义了

全局Session的概念，它被所有构成某个portletweb应.的各种不同的portlet所共享。在globalsession作.域中定义的bean被限定于全局portletSession的.命周期范围内。如果你在web中使.globalsession作.域来标识bean，那么web会.动当 成session类型来使.。 仅当..使..持Web的ApplicationContext时，最后三个才可..。更多spring内容 4.springbean容器的.命周期是什么样的？ springbean容器的.命周期流程如下： 1. Spring容器根据配置中的bean定义中实例化bean

2.

Spring使.依赖注.填充所有属性，如bean中所定义的配置。

3.

如果bean实现BeanNameAware接.，则..通过传递bean的ID来调.setBeanName()。

4.

如果bean实现BeanFactoryAware接.，..通过传递..的实例来调.setBeanFactory()。

5.

如果存在与bean关联的任何BeanPostProcessors，则调.preProcessBeforeInitialization().法。

6.

如果为bean指定了init.法（的init-method属性），那么将调.它。

7.

最后，如果存在与bean关联的任何BeanPostProcessors，则将调.postProcessAfterInitialization().法。

8.

如果bean实现DisposableBean接.，当spring容器关闭时，会调.destory()。

9.

如果为bean指定了destroy.法（的destroy-method属性），那么将调.它。

5.什么是spr ing的内部bean?

只有将bean.作另.个bean的属性时，才能将bean声明为内部bean。为了定义bean，Spring的基于XML的配置元数据 在或中提供了元素的使.。内部bean总是匿名的，它们总是作为原型。 例如，假设我们有.个Student类，其中引.了Person类。这.我们将只创建.个Person类实例并在Student中使.它。 Student.java
publicclassStudent{privatePersonperson;//SettersandGetters }
publicclassPerson{privateStringname;privateStringaddress;//SettersandGetters
}bean.xml<beanid=“StudentBean"class="com.edureka.Student"> <propertyname="person">

<beanclass="com.edureka.Person"> <propertyname="name"value=“Scott"> <propertyname="address"value=“Bangalore"> 6.什么是spring装配 当bean在Spring容器中组合在.起时，它被称为装配或bean装配。Spring容器需要知道需要什么bean以及容器应该如何使.依赖注.来将bean绑定在.起，同时装配bean。 7..动装配有哪些.式？ Spring容器能够.动装配bean。也就是说，可以通过检查BeanFactory的内容让Spring.动解析bean的协作者。 .动装配的不同模式： * no-这是默认设置，表.没有.动装配。应使.显式bean引.进.装配。

-

byName-它根据bean的名称注.对象依赖项。它匹配并装配其属性与XML.件中由相同名称定义的bean。

-

byType-它根据类型注.对象依赖项。如果属性的类型与XML.件中的.个bean名称匹配，则匹配并装配属性。

*构造函数 -它通过调.类的构造函数来注.依赖项。它有.量的参数。

-

autodetect-.先容器尝试通过构造函数使.autowire装配，如果不能，则尝试通过byType.动装配。

8..动装配有什么局限？

*覆盖的可能性 -您始终可以使.和设置指定依赖项，这将覆盖.动装配。

*基本元数据类型 -简单属性（如原数据类型，字符串和类）.法.动装配。

*令.困惑的性质 -总是喜欢使.明确的装配，因为.动装配不太精确。

#注解 1.你.过哪些重要的Spring注解？ * @Controller-.于SpringMVC项.中的控制器类。 *@Service-.于服务类。

-

@RequestMapping-.于在控制器处理程序.法中配置URI映射。

-

@ResponseBody-.于发送Object作为响应，通常.于发送XML或JSON数据作为响应。

-

@PathVariable-.于将动态值从URI映射到处理程序.法参数。

-

@Autowired-.于在springbean中.动装配依赖项。

-

@Qualifier-使.@Autowired注解，以避免在存在多个bean类型实例时出现混淆。

-

@Scope-.于配置springbean的范围。

-

@Configuration，@ComponentScan和@Bean-.于基于java的配置。

-

@Aspect，@Before，@After，@Around，@Pointcut-.于切.编程（AOP）。

2.如何在spring中启动注解装配？ 默认情况下，Spring容器中未打开注解装配。因此，要使.基于注解装配，我们必须通过配置<context：annotation-config/>元素在Spring配置.件中启.它。
3.@Component,@Controller,@Repository,@Service有何区别？ * @Component：这将java类标记为bean。它是任何Spring管理组件的通.构造型。spring的组件扫描机制现在可以将其拾取并将其拉.应.程序环境中。

-

@Controller：这将.个类标记为SpringWebMVC控制器。标有它的Bean会.动导.到IoC容器中。

-

@Service：此注解是组件注解的特化。它不会对@Component注解提供任何其他.为。您可以在服务层类中使.@Service.不是@Component，因为它以更好的.式指定了意图。

-

@Repository：这个注解是具有类似.途和功能的@Component注解的特化。它为DAO提供了额外的好处。它将DAO导.IoC容器，并使未经检查的异常有资格转换为SpringDataAccessException。

4.@Required注解有什么.？ @Required应.于bean属性setter.法。此注解仅指.必须在配置时使.bean定义中的显式属性值或使..动装配填充受影响的bean属性。如果尚未填充受影响的bean属性，则容器将抛出BeanInitializationException。 .例： publicclassEmployee{privateStringname;@Required publicvoidsetName(Stringname){ this.name=name;}publicstringgetName(){ returnname;}} 5.@Autowired注解有什么.？ @Autowired可以更准确地控制应该在何处以及如何进..动装配。此注解.于在setter.法，构造函数，具有任意名称或多个参 数的属性或.法上.动装配bean。默认情况下，它是类型驱动的注.。 publicclassEmployee{privateStringname;@Autowired publicvoidsetName(Stringname){ this.name=name;}publicstringgetName(){ returnname;}} 6.@Qualifier注解有什么.？ 当您创建多个相同类型的bean并希望仅使.属性装配其中.个bean时，您可以使.@Qualifier注解和@Autowired通过指定应该装配哪个确切的bean来消除歧义。 例如，这.我们分别有两个类，Employee和EmpAccount。在EmpAccount中，使.@Qualifier指定了必须装配id为emp1的bean。 Employee.javapublicclassEmployee{privateStringname;@Autowired publicvoidsetName(Stringname){ this.name=name;}publicstringgetName(){ returnname;}} EmpAccount.java publicclassEmpAccount{privateEmployeeemp; @Autowired @Qualifier(emp1)publicvoidshowName(){ System.out.println(“Employeename:”+emp.getName);}} 7.@RequestMapping注解有什么.？ @RequestMapping注解.于将特定HTTP请求.法映射到将处理相应请求的控制器中的特定类/.法。此注解可应.于两个级别： * 类级别：映射请求的URL

-

.法级别：映射URL以及HTTP请求.法

#数据访问 1.springDAO有什么.？ SpringDAO使得JDBC，Hibernate或JDO这样的数据访问技术更容易以.种统.的.式.作。这使得..容易在持久性技术之 间切换。它还允许您在编写代码时，.需考虑捕获每种技术不同的异常。 2.列举SpringDAO抛出的异常。3.springJDBCAPI中存在哪些类？

JdbcTemplate SimpleJdbcTemplate NamedParameterJdbcTemplate SimpleJdbcInsert SimpleJdbcCall 4.使.Spring访问Hibernate的.法有哪些？ 我们可以通过两种.式使.Spring访问Hibernate： 1.使.H ibernate模板和回调进.控制反转

2.扩展H ibernateDAOSupport并应.AOP拦截器节点

5.列举spr ing.持的事务管理类型

Spring.持两种类型的事务管理：

1.

程序化事务管理：在此过程中，在编程的帮助下管理事务。它为您提供极.的灵活性，但维护起来.常困难。

2.

声明式事务管理：在此，事务管理与业务代码分离。仅使.注解或基于XML的配置来管理事务。

6.

spring.持哪些ORM框架

Hibernate iBatis JPA JDO OJB #AOP 1.什么是AOP？ AOP(Aspect-OrientedProgramming),即.向切.编程,它与OOP(Object-OrientedProgramming,.向对象编程)相辅相成,提供了与OOP不同的抽象软件结构的视.. 在OOP中,我们以类(class)作为我们的基本单元,.AOP中的基本单元是Aspect(切.) 2.AOP中的Aspect、Advice、Pointcut、JointPoint和Advice参数分别是什么?

1.

Aspect-Aspect是.个实现交叉问题的类，例如事务管理。..可以是配置的普通类，然后在SpringBean配置.件中配置，或者我们可以使.SpringAspectJ.持使.@Aspect注解将类声明为Aspect。

2.

Advice-Advice是针对特定JoinPoint采取的操作。在编程..，它们是在应.程序中达到具有匹配切.点的特定JoinPoint时执.的.法。您可以将Advice视为Spring拦截器（Interceptor）或Servlet过滤器（filter）。

3.

AdviceArguments-我们可以在advice.法中传递参数。我们可以在切.点中使.args()表达式来应.于与参数模式匹配的任何.法。如果我们使.它，那么我们需要在确定参数类型的advice.法中使.相同的名称。

4.

Pointcut-Pointcut是与JoinPoint匹配的正则表达式，.于确定是否需要执.Advice。Pointcut使.与JoinPoint匹配的不同类型的表达式。Spring框架使.AspectJPointcut表达式语.来确定将应.通知.法的JoinPoint。

5.

JoinPoint-JoinPoint是应.程序中的特定点，例如.法执.，异常处理，更改对象变量值等。在SpringAOP中，JoinPoint始终是.法的执.器。

3.

什么是通知（Advice）？

特定JoinPoint处的Aspect所采取的动作称为Advice。SpringAOP使..个Advice作为拦截器，在JoinPoint"周围"维护.系列的拦截器。

4.

有哪些类型的通知（Advice）？

-

Before-这些类型的Advice在joinpoint.法之前执.，并使.@Before注解标记进.配置。

-

AfterReturning-这些类型的Advice在连接点.法正常执.后执.，并使.@AfterReturning注解标记进.配置。

-

AfterThrowing-这些类型的Advice仅在joinpoint.法通过抛出异常退出并使.@AfterThrowing注解标记配置时执.。

-

After(finally)-这些类型的Advice在连接点.法之后执.，.论.法退出是正常还是异常返回，并使.@After注解标记进.配置。

-

Around-这些类型的Advice在连接点之前和之后执.，并使.@Around注解标记进.配置。

5.指出在springaop中concern和cross-cuttingconcern的不同之处。 concern是我们想要在应.程序的特定模块中定义的.为。它可以定义为我们想要实现的功能。 cross-cuttingconcern是.个适.于整个应.的.为，这会影响整个应.程序。例如，.志记录，安全性和数据传输是应.程序.乎每个模块都需要关注的问题，因此它们是跨领域的问题。 6.AOP有哪些实现.式？ 实现AOP的技术，主要分为两.类： *静态代理 -指使.AOP框架提供的命令进.编译，从.在编译阶段就可.成AOP代理类，因此也称为编译时增强；*编译时编织（特殊编译器实现）*类加载时编织（特殊的类加载器实现）。

*动态代理 -在运.时在内存中"临时".成AOP动态代理类，因此也被称为运.时增强。*JDK动态代理 *CGLIB

7.SpringAOPandAspectJAOP有什么区别？ SpringAOP基于动态代理.式实现；AspectJ基于静态代理.式实现。SpringAOP仅.持.法级别的PointCut；提供了完全的AOP.持，它还.持属性级别的PointCut。 8.如何理解Spr ing中的代理？

将Advice应.于.标对象后创建的对象称为代理。在客.端对象的情况下，.标对象和代理对象是相同的。Advice+TargetObject=Proxy

9.

什么是编织（Weaving）？

为了创建.个advice对象.链接.个aspect和其它应.类型或对象，称为编织（Weaving）。在SpringAOP中，编织在运.时执.。请参考下图：

#MVC 1. SpringMVC框架有什么.？

SpringWebMVC框架提供模型-视图-控制器架构和随时可.的组件，.于开发灵活且松散耦合的Web应.程序。MVC模式有助于分离应.程序的不同..，如输.逻辑，业务逻辑和UI逻辑，同时在所有这些元素之间提供松散耦合。

2.描述.下D ispatcherServlet的.作流程

DispatcherServlet的.作流程可以..幅图来说明：

1.

向服务器发送HTTP请求，请求被前端控制器DispatcherServlet捕获。

2.

DispatcherServlet根据-servlet.xml中的配置对请求的URL进.解析，得到请求资源标识符（URI）。然后根据该URI，调.HandlerMapping获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以HandlerExecutionChain对象的形式返回。

3.

DispatcherServlet根据获得的Handler，选择.个合适的HandlerAdapter。（附注：如果成功获得HandlerAdapter后，此时将开始执.拦截器的preHandler(…).法）。

4.

提取Request中的模型数据，填充Handler.参，开始执.Handler（Controller)。在填充Handler的.参过程中，根据你的配置，Spring将帮你做.些额外的.作：

-

HttpMessageConveter：将请求消息（如Json、xml等数据）转换成.个对象，将对象转换为指定的响应信息。

-

数据转换：对请求消息进.数据转换。如 `String` 转换成 `Integer` 、 `Double` 等。

-

数据根式化：对请求消息进.数据格式化。如将字符串转换成格式化数字或格式化.期等。

-

数据验证：验证数据的有效性（.度、格式等），验证结果存储到 `BindingResult` 或 `Error` 中。

5.

Handler(Controller)执.完成后，向DispatcherServlet返回.个ModelAndView对象；

6.

根据返回的ModelAndView，选择.个适合的ViewResolver（必须是已经注册到Spring容器中的ViewResolver)返回给DispatcherServlet。

7.

ViewResolver结合Model和View，来渲染视图。

8.

视图负责将渲染结果返回给客.端。

3.介绍.下WebApplicationContext WebApplicationContext是ApplicationContext的扩展。它具有Web应.程序所需的.些额外功能。它与普通的 ApplicationContext在解析主题和决定与哪个servlet关联的能...有所不同。 #资料 https://www.edureka.co/blog/interview-questions/spring-interview-questions/https://www.journaldev.com/2696/spring-interview-questions-and-answers 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。↓扫描.维码进群↓

推荐阅读 1.Java 9 ← 2017，2019 → Java13

2.Spring Boot整合 Spring-cache

3.我们再来聊.聊 Java的单例吧

4.我采访了.位 Pornhub .程师

5.团队开发中 Git最佳实践

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 如果.试官问String，就把这篇.章丢给他！ 原创 NYfor2020Java后端1周前

Java中String的应..处不在，.论是算法题还是.试题，String都独占..，甚.是.数.试者.中难以名状的痛。本.着重对 String（若.特地说明，默认是JDK1.8版本）常.的问题来进.介绍：

字符串的不可变性JDK1.6和JDK1.7中substring的原理及区别 replaceFirst、replaceAll、replace区别 String对"+"的"重载"字符串拼接的.种.式和区别 Integer.toString()和String.valueOf()的区别 switch对String的.持（JDK1.7及其后版本）字符串常量池、Class常量池、运.时常量池 String.intern().法 1.字符串的不可变性 我们先来看看下.这段代码: /*公众号：*Java后端*/ publicclassTest{ publicstaticvoidmain(String[]args) {Stringstr1=newString("abc");Stringstr2=newString("abc");System.out.println("str1==str2:"+str1==str2); }} .般都能看出来，这运.结果肯定是false啊，可是为什么呢？在解释之前，先介绍.下System.identityHashCode()：

System.identityHashCode()的作.是.来判断两个对象是否是内存中同.个对象，跟.==判断内存地址是否.样的效果.样。 System.out.println("str1:"+System.identityHashCode(str1));System.out.println("str2:"+System.identityHashCode(str2)); 从关键词new就可以看出，这两个String变量在堆上不可能是同.块内存。其表现（本图是基于JDK1.7，.于字符串常量池后.会介绍）：那么如果加.以下代码，其输出结果会是怎么样的呢?

Stringstr3=str1;System.out.println("str1==str3:"+str1==str3); str3+="ny";System.out.println("str1==str3:"+str1==str3); 第.个结果为true，.第.个结果为false。显.易.，第.个结果出现不同是因为str3赋值为"ny"，那么这整个过程是怎么表现 的呢? 当str3赋值为str1的时候，实际上是str3与str1指向同.块内存地址：

.str3赋值为str3+"ny"时，实际上是在常量池重新创建了.个新的常量"abcny"，并且赋予了不同的内存地址，即：

总结.下：字符串.旦创建，虚拟机就会在常量池..为此字符串分配.块内存，所以它不能被改变。所有的字符串.法都是不能改变..的，.是返回.个新的字符串。 如果需要改变字符串的话，可以考虑使.StringBuffer或StringBuilder来，否则每次改变都会创建.个新的字符串，很浪费内存。 2.JDK1.6和JDK1.7中substring的原理及区别 JDK1.6和JDK1.7中的substring(intbeginIndex, intendIndex).法的实现是不同的，为简单起.，后.中.substring()代表(intbeginIndex,intendIndex).法。.先我们先连接.下substring().法的作.： Stringstr="我不是你最爱的.甜甜了吗？"; str=str.substring(1,3);System.out.println(str);运.结果为：不是我们可以看到，substring().法的作.是截取字段并返回其[beginIndex,endIndex-1]的内容。接下来我们来看看JDK1.6和JDK1.7在实现substring()时原理上的不同。 JDK1.6的substring String是通过字符数组来实现的，我们先来看下源码： publicString(intoffset,intcount,charvalue[]){ this.value=value; this.offset=offset; this.count=count;} publicStringsubstring(intbeginIndex,intendIndex){ return newString(offset+beginIndex,endIndex-beginIndex,value);} 可以看到，在JDK1.6中，String类包含3个重要的成员变量： charvalue[]（存储真正的字符串数组）、into.set（数组的第.个位置索引）、intcount（字符串中包含的字符个数）。 .在虚拟机中，当调.substring.法的时候，堆上会创建.个新的string对象，但是这个string与原先的string.样，指向同.个字符数组，它们之间只是offset和count不相同.已。

这种结构看上去挺好的，只需要创建.个字符数组，然后可以通过调整offset和count就可以返回不同的字符串了。但事实证明，这种情况还是.较少.的，更常.的是从.个很.很.的字符串中切割出需要.到的..段字符序列，这种结构会导致很.的字符数组.直在被使.，.法回收，可能导致内存泄露。所以.般都是这么解决的，原理就是.成.个新的字符并引.它。 str=str.substring(1,3)+""; JDK1.7的substring 所以在JDK1.7提出了.个新的substring()截取字符串的实现： publicString(charvalue[],intoffset,intcount){ this.value=Arrays.copyOfRange(value,offset,offset+count);} publicStringsubstring(intbeginIndex,intendIndex){ intsubLen=endIndex-beginIndex; returnnewString(value,beginIndex,subLen);} 我们可以看到，String构造函数的实现已经换成了Arrays.copyOfRange().法了，这个.法最后会.成.个新的字符数组。也就是说，使.substring().法截取字段，str不会使.之前的字符数组，.是引.新.成的字符数组。

总结.下：JDK1.6与JDK1.7在实现substring().法时最.的不同在于，前者沿.了原来的字符数组，.后者引.了新创建的字 符数组。 3.replaceFirst、replaceAll、replace区别 从字.上看，这三者的区别在于名称：replace（替换）、replaceAll（替换全部）、replaceFirst（替换第.个符合条件）。在从功能、源码上对这三者进.介绍之前，我们先来看看这道题： publicstaticvoidmain(String[]args){ Stringstr="I.am.fine.";System.out.println(str.replace(".","\"));System.out.println(str.replaceAll(".","\\"));System.out.println(str.replaceFirst(".","\\")); } 运.结果为： I\am\fine\ \\\\\ .am.fine. 做对了吗？下.来分别对这三者进.介绍。 replace 结合题.中的执.replace().法后的输出结果，我们来看看在Java中的replace()的源码： publicStringreplace(CharSequencetarget,CharSequencereplacement) {returnPattern.compile(target.toString(),Pattern.LITERAL). matcher(this).replaceAll(Matcher.quoteReplacement(replacement.toString()));} 可以看到replace()只.持.参为字符序列，.且实现的是完全替换，只要符合target的字段都进.替换。 replaceAll 在进.介绍之前我们先看看源码： publicStringreplaceAll(Stringregex,Stringreplacement) {returnPattern.compile(regex).matcher(this).replaceAll(replacement);} 我们可以看到，replaceAll().持.参为正则表达式，.且此.法也是实现字段的完全替换。从运.结果中我们能看到所有的字符都被替换了，其实是因为"."在正则表达式中表."所有字符"，如果想要只替换"."..全部字段，则可以这么写： System.out.println(str.replaceAll("\.","\\")); replaceFirst其实从上.的运.结果来看，也知道replaceFirst也是.持.参为正则表达式，但是此.法实现的是对第.个符合条件的字段进.替换。 publicStringreplaceFirst(Stringregex,Stringreplacement) {returnPattern.compile(regex).matcher(this).replaceFirst(replacement);} 总结.下，replace不.持.参为正则表达式但能实现完全替换；replaceAll.持.参为正则表达式且能实现完全替换； replaceFirst.持.参为正则表达式，但替换动作只发..

次。 4.String对"+"的"重载" 当我们查看String的源码时，我们可以看到： privatefinalcharvalue[];.且在上.我们已经提到String具有不可变性，可当我们在使."+"对字符串进.拼接时，却可以成功。它的原理是什么呢？举 个栗.： publicstaticvoidmain(String[]args){Stringstr="abc"; str+="123"; } 然后我们查看反编译后的结果：

可以看到，虽然我们没有.到java.lang.StringBuilder类，但编译器为了执.上述代码时会引.StringBuilder类，对字符串进.拼接。 其实很多.认为使."+"拼接字符串的功能可以理解为运算符重载，但Java是不.持运算符重载的（但C++.持）。 运算符重载：在计算机程序设计中，运算符重载（operator overloading）是多态的.种。运算符重载就是对已有的运算符进.定义，赋予其另.种功能，以适应不同的数据类型。 从反编译的代码来看，其实这只是.种Java语法糖。 总结.下，String使."+"进.拼接的原理是编译器使.了StringBuilder.append().法进.拼接，且这是.种语法糖。 5.字符串拼接的.种.式和区别 字符串拼接是字符串处理中常.的操作之.，即将多个字符串拼接到.起，但从上.我们已经知道了String具有不可变性，那么字符串拼接.是怎么做到的呢？ String.concat()拼接 在介绍concat原理之前，我们先看看concat是怎么使.的： publicstaticvoidmain(String[]args){Stringstr="我不是你最爱的.甜甜了吗？"; str=str.concat("你是个好姑娘");System.out.println(str); } 运.结果为：我不是你最爱的.甜甜了吗？你是个好姑娘 我们可以看到，concat().法是String类的，且是将原本的字符串与参数中的字符串进.拼接。现在我们来看看它的源码： publicStringconcat(Stringstr){intotherLen=str.length();if(otherLen==0){ returnthis;}intlen=value.length; charbuf[]=Arrays.copyOf(value,len+otherLen); str.getChars(buf,len); returnnewString(buf,true); } 可以看到，concat()的拼接实际上是，创建.个.度为已有字符串和待拼接字符串的.度之和的字符数组，然后将两个字符串的值赋值到新的字符数组中，最后利.这个字符数组创建.个新的String对象。 StringBuilder.append()拼接 上.在介绍String的"+"拼接时，StringBuilder已经出来混个脸熟了，现在我们看个例.： publicstaticvoidmain(String[]args){StringBuildersb=newStringBuilder("我不是你最爱的.甜甜了吗？"); sb.append("你是个好姑娘");System.out.println(sb.toString()); } 运.结果同上，接下来我们来看看StringBuilder的实现原理。StringBuilder内部同String类似，也封装了.个字符数组： char[]value; 与String相.，StringBuilder的字符数组并不是final修饰的，即可修改。.且字符数组中不.定所有位置都已经被使.了，StringBuilder有.个专.记录使.字符个数的实例变量： intcount; .StringBuilder.append()的源码如下： publicStringBuilderappend(Stringstr){super.append(str);returnthis; } 可以看到StringBuilder.append().法是采..类AbstractStringBuilder的append().法： publicAbstractStringBuilderappend(Stringstr){if(str==null)returnpublicAbstractStringBuilderappend(Stringstr){ if(str==null)returnappendNull();intlen=str.length();ensureCapacityInternal(count+len); str.getChars(0,len,value,count); count+=len; returnthis; }nappendNull();intlen=str.length();ensureCapacityInternal(count+len); str.getChars(0,len,value,count); count+=len; returnthis; } ensureCapacityInternal().法.于扩展字符数组.度（有兴趣的读者可以查看其扩展的.法），所以这.的append.法会直接拷.字符到内部的字符数组中，如果字符数组.度不够，则进.扩展。 StringBuffer.append()拼接 StringBuffer和StringBuilder结构类似，且.类都是AbstractStringBuilder，.者最.的区别在于StringBuffer是线程安全的，我们来看下StringBuffer.append()的源码： publicsynchronizedStringBufferappend(Stringstr){toStringCache=null; super.append(str);returnthis; } 可以看到，StringBuffer.append().法是使.synchronized进.声明，说明这是.个线程安全的.法，.上. StringBuilder.append()则不是线程安全的.法。 StringUtils.join()拼接 这个拼接.式适.于字符串集合的拼接，举个栗.： publicstaticvoidmain(String[]args){Listlist=newArrayList<>();list.add("我不是你最爱的.甜甜了吗？");list.add("你是个好姑娘");Strings=newString(); s=StringUtils.join(list,s);System.out.println(s); } 运.结果同上，接下来我们来看.下原理： publicstaticStringjoin(Collectionvar0,Stringvar1){StringBuffervar2=newStringBuffer(); for(Iteratorvar3=var0.iterator();var3.hasNext();var2.append((String)var3.next())){if(var2.length()!=0){ var2.append(var1); }}returnvar2.toString(); } StringUtils.join().法中依然是使.StringBuffer和Iterator迭代器来实现，.且如果集合类中的数据不是String类型，在遍历集合的过程中还会强制转换成String。 总结.下，加上上.介绍的使."+"进.字符串拼接的.式，此..共介绍了五种字符串拼接的.式，分别是：使."+"、使. String.concat()、使.StringBuilder.append()、使.StringBuffer.append()、使.StringUtils.join()。需要强调的是： 1.使. StringBuilder.append()的.式是效率最.的；

2.

如果不是在循环体中进.字符串拼接，."+".式就.了；

3.

如果在并发场景中进.字符串拼接的话，要使.StringBuffer代替StringBuilder。

6.

Integer.toString()和String.valueOf()的区别

Integer.toString().法和String.valueOf().法来进.int类型转String，举个栗.： publicstaticvoidmain(String[]args) {inti=1;StringintegerTest=Integer.toString(i);StringstringTest=String.valueOf(i); } 平常我们在使.这两个.法来进.int类型转String时，并没有对其加以区分，这次就来深究.下它们之间有何区别，以及使.哪个.法.较好。 Integer.toString().法 以下为Integer.toString()的实现源码，其中的stringSize().法会返回整型数值i的.度，getChars().法是将整型数值填充字符数组buf： publicstaticStringtoString(inti){if(i==Integer.MIN_VALUE) return"-2147483648";intsize=(i<0)?stringSize(-i)+1:stringSize(i); char[]buf=newchar[size]; getChars(i,size,buf); returnnewString(buf,true); } 可以看到，Integer.toString()先是通过判断整型数值的正负性来给出字符数组buf的..，然后再将整型数值填充到字符数组中，最后返回创建.个新的字符串并返回。在包装类中不仅是Integer，同理Double、Long、Float等也有对应的toString().法。 String.valueOf().法 String.valueOf()相对于Integer.toString().法来说，有.量的重载.法，在此列举出.个典型的.法。 publicstaticStringvalueOf(Objectobj) 这个.法的.参是Object类型，所以只需要调.Object的toString().法即可（在编写类的时候，最好重写其toString().法）。 publicstaticStringvalueOf(Objectobj){return(obj==null)?"null":obj.toString();} publicstaticStringvalueOf(chardata[]) 当.参为字符数组时，看过上.的String.concat().法的原理，我们.乎可以下意识地反应：这.的字符数组，应该是.于创建.个新的字符串对象来并返回该字符串了。 publicstaticStringvalueOf(chardata[]) {returnnewString(data);} 除了字符数组外，字符也是通过转换成字符数组后，创建.个新的字符串对象来返回字符串。 publicstaticStringvalueOf(booleanb) 其实布尔型数值的返回结果只有两种：true或false，所以只要对这两个数值进.字符处理即可。 publicstaticStringvalueOf(booleanb){returnb?"true":"false";} publicstaticStringvalueOf(inti) 上.我们介绍了Integer.toString().法，这.法String.valueOf()就.到了。.且重载的.参类型不.int，还有long、float、 double等。 publicstaticStringvalueOf(inti){returnInteger.toString(i);} 总结.下，我们看到String.valueOf()有许多重载.法，且关乎于包装类如Integer等的.法内部还是调.了包装类..的.法如 Integer.toString()。因其内部重载了不同类型转换成String的处理，所以推荐使.String.valueOf().法。 7.switch对String的.持（JDK1.7及其后版本） 在JDK 1.7之前，switch只.持对int、short、byte、char这四类数据做判断，.在JVM内部实际上只.持对int类型的处理。因为虚拟机在处理之前，会将如short等类型数据转换成int型，再进.条件判断。 在JDK1.7的中switch增加了对String的.持，照常，先举个栗.： publicstaticvoidmain(String[]args){Stringstr="abc"; switch(str){ case"ab": System.out.println("ab");break; case"abc": System.out.println("abc");break; default: break;}} 运.结果为： ab 因为switch关键词不像是类和.法，可以直接查看源码，所以这.采.查看编译后的Class.件和查看反编译的.式。.先我们查看编译后的Class.件： publicstaticvoidmain(String[]var0){Stringvar1="abc";bytevar3=-1; switch(var1.hashCode()){case3105: if(var1.equals("ab")){var3=0;}break; case96354: if(var1.equals("abc")){var3=1; }} ..... } 可以看到，switch的.参为字符串"abc"的hashCode，switch进.判断的依然还是整数，.且进.判断的字符串也被转换成整型数值，在case中还使.了equals().法对字符串进.判断，以确认是否进.case内代码的下.步操作。接下来我们看看反编译之后的情况：

看到..框的代码，我们可以知道String需要转换成int类型的整型数据之后才能进.在switch中进.判断。.红.框中的代码中我们可以看到，这个过程不.使.了hashCode().法，还使.了equals().法对字符串进.判断。 但也因switch判断字符串的实现原理是求出String的hashCode，所以String不能赋值为null，否则会报 NullPointerException。 总结.下，switch.持String本质上还是switch在对int类型数值进.判断。 8.字符串常量池、Class常量池、运.时常量池 在Java的内存分配中经常听到关于常量池的描述，但名声最.的还是运.时常量池，对于字符串常量池和Class常量池近乎没有印象，甚.是混在.起，在此将这.个概念进.区分。 字符串常量池 在不知道这个名词之前，笔者以为字符串会跟类的其他信息.样存储在.法区（或永久代）中，但遇到它之后，笔者发觉这事情没那么简单。 我们来看看它和永久代的搬家史：

JDK1.7之前，字符串常量池在永久代中JDK1.7，将字符串常量池移出了永久代，搬到了 DataSegument中，.个在堆中.个相对特殊的位置（失去唯.引.也不

会被回收）JDK1.8，永久代被元空间取代了

字符串常量池中的内容是在字符串对象实例的引.值（字符串常量池中存储的是引.值，具体的字符串对象实例存放在堆的另.块空间），.且在HotSpotVM中实现的字符串常量池是.个由C++实现的 StringTable，结构跟Hashtable类似，但区别在于不能.动扩容。这个StringTable在每个HotSpotVM中是被所有的类共享的。 这么说可能有点抽象，不如使.HSDB来亲眼看看吧。举个栗.： classNY{ Stringstr="nyfor2020";}publicclassTest{ publicstaticvoidmain(String[]args) {NYny1=newNY();NYny2=newNY();try{ System.in.read();}catch(IOExceptione){e.printStackTrace();}}} 在命令提.符中输.“jps”查看进程号后，在命令提.符中输.： java-classpath"%JAVA_HOME%/lib/sa-jdi.jar"sun.jvm.hotspot.HSDB 打开HSDB，输.进程号后使.ObjectHistogram找到相应类之后，可以找到两个NY对象引.的字符串的地址是同.个。

Class常量池 在《深.理解Java虚拟机》中对Class常量池的介绍是从这.引.： Class.件中除了有类的版本、字段、.法、接.等描述信息外，还有.项信息是常量池（Constant Pool Table），.于存放编译期.成的各种字.量和符号引.，这部分内容将在类加载后进..法去的运.时常量池中存放。 字.量即常量概念，如.本字符串、被声明为final的常量值等。.符号引.即.组

符号来描述所引.的.标，符号可以是任何形式的字.量，只要使.的时候能直接定位到.标即可。 .般所说的类常量有以下三类：

类和接.的全限定名字段的名称和描述符.法的名称和描述符

关于常量池中的每.个常量表.什么含义在此就不赘述，想了解的朋友可以参考《深.理解Java虚拟机》的第六章。举个栗.： publicclassTest{publicstaticvoidmain(String[]args){Strings1="nyfor2020";}} 当我们使.以下命令进.反编译： javap-verboseTest.class

在反编译之后我们可以直接看到Class常量池中的内容，有类的全限定名、.法的描述符和字段的描述符。也就是说，当Java.件被编译成Class.件的过程之后，就会.成Class常量池。那么运.时常量池.是什么时候产.的呢？运.时常量池运.时常量池是.法区的.部分，.于存放Class.件编译后.成的Class常量池等信息。接下来我们结合类加载过程来认识这.个常量池之间的关系：在JVM进.类加载过程中必须经过加载、连接、初始化这三个阶段（在《Java的继承（深.版）》中有介绍），.连接过程.包

括了验证、准备和解析这三个阶段。 当类加载到内存后，JVM就会将Class常量中的内容存放到运.时常量池中。.在Class常量池中存储的是字.量和符号引.，..真正的对象实例，所以在经过解析之后，会将符号引.替换为直接引.，.在解析过程中会去查询字符串常量池，以保证运.时常量池所应.的字符串与字符串常量池中的信息.致。 9.String.intern().法 在了解三个常量池之间的区别之后，我们来看看与字符串常量池有关的intern().法.
/***Returnsacanonicalrepresentationforthestringobject. *

*Apoolofstrings,initiallyempty,ismaintainedprivatelybythe *class{@codeString}. *

*Whentheinternmethodisinvoked,ifthepoolalreadycontainsa *stringequaltothis{@codeString}objectasdeterminedbythe{@link#equals(Object)}method,thenthestringfromthepoolis *returned.Otherwise,this{@codeString}objectisaddedtothe *poolandareferencetothis{@codeString}objectisreturned. *

Itfollowsthatforanytwostrings{@codes}and{@codet}, {@codes.intern()==t.intern()}is{@codetrue}*ifandonlyif{@codes.equals(t)}is{@codetrue}. *

*Allliteralstringsandstring-valuedconstantexpressionsare *interned.Stringliteralsaredefinedinsection3.10.5ofthe *_TheJava™LanguageSpecification_. * *@return astringthathasthesamecontentsasthisstring,butis *guaranteedtobefromapoolofuniquestrings. */ publicnativeStringintern(); 我们可以看到，intern().法是.个本地.法，注释描述的.致意思是： “当intern().法被调.时，如果常量池中存在当前字符串，就会直接返回当前字符串；如果常量池中没有此字符串，会将此字符串放.常量池中后，再返回”。该.法的作.就是把字符串加载到常量池中。 刚刚在介绍字符串常量池时提到它在JDK 1.6和JDK 1.7的内存位置发.了变化，所以在不同版本的JDK中intern().法的表现也有所差别。举个栗.： publicstaticvoidmain(String[]args){Stringstr1=newString("1")+newString("1");
str1.intern();Stringstr2="11";System.out.println(str1==str2); } 在JDK 1.6中的运.结果为false，在JDK 1.7中的运.结果为true。为什么会出现这种情况呢？主要是字符串常量池的内存位置变了，导致intern()的内部实现也发.了变化。 在JDK1.6中的intern() intern().法将字符串复制到字符串常量池，然后返回.个该字符串在常量池的引.，但是str1并没接收到这个应.，所以str1指向的还是堆，但是str2指向的是常量区，所以这两个地址不.样。

在JDK1.7中的intern() 在JDK 1.7中的intern().法，（在字符串常量池找不到该字符串时）将该字符串对象在堆.的引.注册到常量池，以后在使.相同字.量声明的字符串对象则都指向该地址，也就是该字符串在堆中的地址。

等等，如果把intern()的位置下移..之后呢？ （基于JDK1.7） publicstaticvoidmain(String[]args) {Stringstr1=newString("1")+newString("1")Stringstr2="11";
str1.intern();System.out.println(str1==str2);System.out.println(System.identityHashCode(str1));System.out.println(System.identityHashCode(str2)); } 运.结果为： false 22307196 10568834 可以看到intern()的执.顺序改变之后，字符串常量池已经存储了"1"和"11"引.了，所以str2依然指向的是常量池中的引.，.str1指向的是new出来的字符串对象地址。 结语 在.常使.的时候，我们对于String的态度就像是对待空.，只有在出问题了才会发现之前没对它加以了解。此.以String问题为契机，对String相关原理进.回顾。 如果本.对你的学习有帮助，请给.个赞吧，这会是我最.的动. 参考资料 Java中String对象的不可变性 https://www.cnblogs.com/qingergege/p/5701011.html 在Java虚拟机中，字符串常量到底存放在哪https://blog.csdn.net/weixin_34413802/article/details/88009934了解JDK6和JDK7中substring的原理及区别 https://www.cnblogs.com/dsitn/p/7151624.html java的replaceFirst和(反斜杠)[replace、replaceAll和replaceFirst的区别] https://blog.csdn.net/lonfee88/article/details/7333883 String重载”+”原理分析

https://blog.csdn.net/codejas/article/details/78662146字符串拼接的.种.式和区别
https://www.cnblogs.com/lujiahua/p/11408689.html Java—String.valueof()和Integer.toString()的不同
https://blog.csdn.net/whp1473/article/details/79935082 javaswitch是如何.持String类型的?
https://blog.csdn.net/guohengcook/article/details/81267768 JVM|运.时常量池和字符串常量池及intern()
https://hacpai.com/article/1581300080734Java中.种常量池的区分 http://tangxman.github.io/2015/07/27/the-difference-of-java-string-pool/《深.理解Java虚拟机》 -END- 推荐阅读 1.GitHub移动端来了！

2.

..读懂HTTPS

3.

知乎.友：这是哪个傻逼写的代码?

4.

Git.级.法

阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 建议收藏！写给程序员的MySQL.试.频100问 Java后端 2019-12-09 点击上.Java后端，选择设为星标优质.章，及时送达

来源|juejin.im/post/5d351303f265da1bd30596f9 前. 本.主要受众为开发.员,所以不涉及到MySQL的服务部署等操作,且内容较多,.家准备好耐.和..矿泉.. 前.阵系统的学习了.下MySQL,也有.些实际操作经验,偶然看到.篇和MySQL相关的.试.章,发现其中的.些问题..也回答不好,虽然知识点.部分都知道,但是.法将知识串联起来.因此决定搞.个MySQL灵魂100问,试着.回答问题的.式,让..对知识点的理解更加深..点.此.不会事.巨细的从select的.法开始讲解mysql,主要针对的是开发.员需要知道的.些MySQL的知识点主要包括索引,事务,优化等..,以在.试中.频的问句形式给出答案. 索引相关 关于MySQL的索引,曾经进.过.次总结,.章链接在这.Mysql索引原理及其优化. 1.什么是索引? 索引是.种数据结构,可以帮助我们快速的进.数据的查找. 2.索引是个什么样的数据结构呢? 索引的数据结构和具体存储引擎的实现有关,在MySQL中使.较多的索引有Hash索引,B+树索引等,.我们经常使.的InnoDB存储引擎的默认索引实现为:B+树索引. 3.Hash索引和B+树所有有什么区别或者说优劣呢? .先要知道Hash索引和B+树索引的底层实现原理: hash索引底层就是hash表,进.查找时,调..次hash函数就可以获取到相应的键值,之后进.回表查询获得实际数据.B+树底层实现是多路平衡查找树.对于每.次的查询都是从根节点出发,查找到叶.节点.可以获得所查键值,然后根据查询判断是否需要回表查询数据.那么可以看出他们有以下的不同:

hash索引进.等值查询更快(.般情况下),但是却.法进.范围查询. 因为在hash索引中经过hash函数建.索引之后,索引的顺序与原顺序.法保持.致,不能.持范围查询..B+树的的所有节点皆遵循(左节点.于.节点,右节点.于.节点,多叉树也类似),天然.持范围.

hash索引不.持使.索引进.排序,原理同上. hash索引不.持模糊查询以及多列索引的最左前缀匹配.原理也是因为hash函数的不可预测.AAAA和AAAAB的索引没有相关性. hash索引任何时候都避免不了回表查询数据,.B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询. hash索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在.量重复的时候,发.hash碰撞,此时效率可能极差..B+树的查询效率.较稳定,对于所有的查询都是从根节点到叶.节点,且树的.度较低. 因此,在.多数情况下,直接选择B+树索引可以获得稳定且较好的查询速度..不需要使.hash索引. 4.上.提到了B+树在满.聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引? 在B+树的索引中,叶.节点可能存储了当前的key值,也可能存储了当前的key值以及整.的数据,这就是聚簇索引和.聚簇索引. 在InnoDB中,只有主键索引是聚簇索引,如果没有主键,则挑选.个唯.键建.聚簇索引.如果没有唯.键,则隐式的.成.个键来建.聚簇索引.当查询使.聚簇索引时,在对应的叶.节点,可以获取到整.数据,因此不.再次进.回表查询. 5..聚簇索引.定会回表查询吗? 不.定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进.回表查询. 举个简单的例.,假设我们在员.表的年龄上建.了索引,那么当进. selectagefromemployeewhereage<20的查询时,在索引的叶.节点上,已经包含了age信息,不会再次进.回表查询. 6.在建.索引的时候,都有哪些需要考虑的因素呢? 建.索引的时候.般要考虑到字段的使.频率,经常作为条件进.查询的字段.较适合.如果需要建.联合索引的话,还需要考虑联合索引中的顺序. 此外也要考虑其他..,.如防.过多的所有对表造成太.的压..这些都和实际的表结构以及查询.式有关. 7.联合索引是什么?为什么需要注意联合索引中的顺序? MySQL可以使.多个字段同时建..个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建.索引时的字段顺序挨个使.,否则.法命中索引.具体原因为:MySQL使.索引时需要索引有序,假设现在建.了"name,age,schoo l"的联合索引 那么索引的排序为:先按照name排序,如果name相同,则按照age排序,如果age的值也相等,则按照school进.排序. 当进.查询时,此时索引仅仅按照name严格有序,因此必须.先使.name字段进.等值查询,之后对于匹配到的列..,其按照age字段严格有序,此时可以

使.age字段.做索引查找,以此类推. 因此在建.联合索引的时候应该注意索引列的顺序,.般情况下,将查询需求频繁或者字段选择性.的列放在前..此外可以根据特例的查询或者表结构进.单独的调整. 8.创建的索引有没有被使.到?或者说怎么才可以知道这条语句运.很慢的原因? MySQL提供了explain命令来查看语句的执.计划,MySQL在执.某个语句之前,会将该语句过.遍查询优化器,之后会拿到对语句的分析,也就是执.计划,其中包含了许多信息. 可以通过其中和索引有关的信息来分析是否命中了索引,例如possilbe_key,key,key_len等字段,分别说明了此语句可能会使.的索引,实际使.的索引以及使.的索引.度. 9.那么在哪些情况下会发.针对该列创建了索引但是在查询的时候并没有使.呢?

使.不等于查询列参与了数学运算或者函数 在字符串like时左边是通配符.类似于'%aaa'. 当mysql分析全表扫描.使.索引快的时候不使.索引. 当使.联合索引,前..个条件为范围查询,后.的即使符合最左前缀原则,也.法使.索引. 以上情况,MySQL.法使.索引. 事务相关 1.什么是事务? 理解什么是事务最经典的就是转账的栗.,相信.家也都了解,这.就不再说.边了.事务是.系列的操作,他们要符合ACID特性.最常.的理解就是:事务中的操作要么全部成功,要么全部失败.但是只是这样还不够的. 2.ACID是什么?可以详细说.下吗? A=Atomicity 原.性,就是上.说的,要么全部成功,要么全部失败.不可能只执..部分操作. C=Consistency 系统(数据库)总是从.个.致性的状态转移到另.个.致性的状态,不会存在中间状态. I=Isolation 隔离性:通常来说:.个事务在完全提交之前,对其他事务是不可.的.注意前.的通常来说加了红.,意味着有例外情况. D=Durability 持久性,.旦事务提交,那么就永远是这样.了,哪怕系统崩溃也不会影响到这个事务的结果. 3.同时有多个事务在进.会怎么样呢? 多事务的并发进..般会造成以下.个问题:

脏读:A事务读取到了B事务未提交的内容,.B事务后.进.了回滚. 不可重复读:当设置A事务只能读取B事务已经提交的部分,会造成在A事务内的两次查询,结果竟然不.样,因为在此期间B事务进.了提交操作. 幻读:A事务读取了.个范围的内容,.同时B事务在此期间插.了.条数据.造成"幻觉". 4.怎么解决这些问题呢?MySQL的事务隔离级别了解吗? MySQL的四种隔离级别如下:

未提交读(READUNCOMMITTED) 这就是上.所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,.之后该事务进.了回滚). 这个级别的性能没有.够.的优势,但是.有很多的问题,因此很少使..

已提交读(READCOMMITTED)

REPEATABLEREAD(可重复读)

其他事务只能读取到本事务已经提交的部分.这个隔离级别有不可重复读的问题,在同.个事务内的两次读取,拿到的结果竟然不.样,因为另外.个事务对数据进.了修改. 可重复读隔离级别解决了上.不可重复读的问题(看名字也知道),但是仍然有.个新问题,就是幻读 当你读取id>10的数据.时,对涉及到的所有.加上了读锁,此时例外.个事务新插.了.条id=11的数据,因为是新插.的,所以不会触发上.的锁的排斥那么进.本事务进.下.次的查询时会发现有.条id=11的数据,.上次的查询操作并没有获取到,再进.插.就会有主键冲突的问题.

SERIALIZABLE(可串.化) 这是最.的隔离级别,可以解决上.提到的所有问题,因为他强制将所以的操作串.执.,这会导致并发性能极速下降,因此也不是很常.. 5.Innodb使.的是哪种隔离级别呢? InnoDB默认使.的是可重复读隔离级别. 6.对MySQL的锁了解吗? 当数据库有并发事务的时候,可能会产.数据的不.致,这时候需要.些机制来保证访问的次序,锁机制就是这样的.个机制. 就像酒店的房间,如果.家随意进出,就会出现多.抢夺同.个房间的情况,.在房间上装上锁,申请到钥匙的.才可以.住并且将房间锁起来,其他.只有等他使.完毕才可以再次使.. 7.MySQL都有哪些锁呢?像上.那样.进.锁定岂不是有点阻碍并发效率了? 从锁的类别上来讲,有共享锁和排他锁.共享锁:.叫做读锁.当..要进.数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.排他锁:.叫做写锁.当..要进.数据的写.时,对数据加上排他锁.排他锁只可以加.个,他和其他的排他锁,共享锁都相斥..上.的例.来说就是..的.为有两种,.种是来看房,多个...起看房是可以接受的..种是真正的.住.晚,在这期间,.论是 想.住的还是想看房的都不可以.锁的粒度取决于具体的存储引擎,InnoDB实现了.级锁,.级锁,表级锁.他们的加锁开销从...,并发能.也是从.到.. 表结构设计 1.为什么要尽量设定.个主键? 主键是数据库确保数据.在整张表唯.性的保障,即使业务上本张表没有主键,也建议添加.个.增.的ID列作为主键.设定了主键之后,在后续的删改查的时候可能更加快速以及确保操作数据范围安全. 2.主键使..增ID还是UUID? 推荐使..增ID,不要使.UUID.因为在InnoDB存储引擎中,主键索引是作为聚簇索引存在的 也就是说,主键索引的B+树叶.节点上存储了主键索引以及全部的数据(按照顺序) 如果主键索引是.增ID,那么只需要不断向后排列即可,如果是UUID,由于到来的ID与原来的..不确定,会造成.常多的数据插.,数据移动,然后导致产.很多的内存碎.,进.造成插.性能的下降.总之,在数据量..些的情况下,..增主键性能会好.些. 图.来源于《.性能MySQL》:其中默认后缀为使..增ID,_uuid为使.UUID为主键的测试,测试了插.100w.和300w.的性能.

关于主键是聚簇索引,如果没有主键,InnoDB会选择.个唯.键来作为聚簇索引,如果没有唯.键,会.成.个隐式的主键. IfyoudefineaPRIMARYKEYonyourtable,InnoDBusesitastheclusteredindex.

IfyoudonotdefineaPRIMARYKEYforyourtable,MySQLpicksthefirstUNIQUEindexthathasonlyNOTNULLcolumnsas theprimarykeyandInnoDBusesitastheclusteredindex. 3.字段为什么要求定义为notnull? MySQL官.这样介绍: NULLcolumnsrequireadditionalspaceintherowtorecordwhethertheirvaluesareNULL.ForMyISAMtables,eachNULL columntakesonebitextra,roundeduptothenearestbyte. null值会占.更多的字节,且会在程序中造成很多与预期不符的情况. 4.如果要存储..的密码散列,应该使.什么字段进.存储? 密码散列,盐,...份证号等固定.度的字符串应该使.char.不是varchar来存储,这样可以节省空间且提.检索效率. 存储引擎相关 1.MySQL.持哪些存储引擎? MySQL.持多种存储引擎,.如InnoDB,MyISAM,Memory,Archive等等.在.多数的情况下,直接选择使.InnoDB引擎都是最合适的,InnoDB也是MySQL的默认存储引擎.

1. InnoDB和MyISAM有什么区别?

InnoDB.持事物，.MyISAM不.持事物 InnoDB.持.级锁，.MyISAM.持表级锁 InnoDB.持MVCC,.MyISAM不.持 InnoDB.持外键，.MyISAM不.持 InnoDB不.持全.索引，.MyISAM.持。 零散问题 1.MySQL中的varchar和char有什么区别. char是.个定.字段,假如申请了 char(10)的空间,那么.论实际存储多少内容.该字段都占.10个字符,.varchar是变.的也就是说申请的只是最..度,占.的空间为实际字符.度+1,最后.个字符存储使.了多.的空间.在检索效率上来讲,char>varchar,因此在使.中,如果确定某个字段的值的.度,可以使.char,否则应该尽量使.varchar.例如存 储..MD5加密后的密码,则应该使.char. 2.varchar(10)和int(10)代表什么含义? varchar的10代表了申请的空间.度,也是可以存储的数据的最..度,.int的10只是代表了展.的.度,不.10位以0填充.也就是说,int(1)和int(10)所能存储的数字..以及占.的空间都是相同的,只是在展.时按照.度展.. 3.MySQL的binlog有有.种录.格式?分别有什么区别? 有三种格式,statement,row和mixed.

statement模式下,记录单元为语句.即每.个sql造成的影响会记录.由于sql的执.是有上下.的,因此在保存的时候需要保存相关的信息,同时还有.些使.了函数之类的语句.法被记录复制. row级别下,记录单元为每..的改动,基本是可以全部记下来但是由于很多操作,会导致.量.的改动(.如alter table),因此这种模式的.件保存的信息太多,.志量太.. mixed..种折中的.案,普通操作使.statement记录,当.法使.statement的时候使.row. 此外,新版的MySQL中对row级别也做了.些优化,当表结构发.变化的时候,会记录语句.不是逐.记录. 4.超.分.怎么处理? 超.的分..般从两个.向上来解决.

数据库层.,这也是我们主要集中关注的(虽然收效没那么.) 类似于 select*fromtablewhereage>20limit1000000,10这种查询其实也是有可以优化的余地的. 这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然.较慢. 我们可以修改为 select*fromtablewhereidin(selectidfromtablewhereage>20limit1000000,10) 这样虽然也load了.百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快. 同时如果ID连续的好,我们还可以select*fromtablewhereid>1000000limit10,效率也是不错的 优化的可能性有许多种,但是核.思想都.样,就是减少load的数据.

从需求的.度减少这种请求....主要是不做类似的需求(直接跳转到.百万.之后的具体某...只允许逐.查看或者按照给定的路线.,这样可预测,可缓存)以及防.ID泄漏且连续被.恶意攻击. 解决超.分.,其实主要是靠缓存,可预测性的提前查到内容,缓存.redis等k-V数据库中,直接返回即可.在阿.巴巴《Java开发.册》中,对超.分.的解决办法是类似于上.提到的第.种.

5.关.过业务系统..的sql耗时吗?统计过慢查询吗?对慢查询都怎么优化过? 在业务系统中,除了使.主键进.的查询,其他的我都会在测试库上测试其耗时,慢查询的统计主要由运维在做,会定期将业务中的慢查询反馈给我们.慢查询的优化.先要搞明.慢的原因是什么?是查询条件没有命中索引?是load了不需要的数据列?还是数据量太.?所以优化也是针对这三个.向来的,

.先分析语句,看看是否load了额外的数据,可能是查询了多余的.并且抛弃掉了,可能是加载了许多结果中并不需要的列,对语句进.分析以及重写.

分析语句的执.计划,然后获得其使.索引的情况,之后修改语句或者修改索引,使得语句可以尽可能的命中索引.

如果对语句的优化已经.法进.,可以考虑表中的数据量是否太.,如果是的话可以进.横向或者纵向的分表.

6.上.提到横向分表和纵向分表,可以分别举.个适合他们的例.吗? 横向分表是按.分表.假设我们有.张..表,主键是.增ID且同时是..的ID.数据量较.,有1亿多条,那么此时放在.张表.的查询效果就不太理想.我们可以根据主键ID进.分表,.论是按尾号分,或者按ID的区间分都是可以的. 假设按照尾号0-99分为100个表,那么每张表中的数据就仅有100w.这时的查询效率.疑是可以满.要求的.纵向分表是按列分表.假设我们现在有.张.章表.包含字段 id-摘要-内容..系统中的展.形式是刷新出.个列表,列表中仅包含标题和摘要 当..点击某篇.章进.详情时才需要正.内容.此时,如果数据量.,将内容这个很.且不经常使.的列放在.起会拖慢原表的查 询速度.我们可以将上.的表分为两张. id-摘要 , id-内容.当..点击详情,那主键再来取.次内容即可..增加的存储量只是很.的主键字段.代价很.. 当然,分表其实和业务的关联度很.,在分表之前.定要做好调研以及benchmark.不要按

照..的猜想盲.操作. 7.什么是存储过程？有哪些优缺点？ 存储过程是.些预编译的SQL语句。1、更加直.的理解：存储过程可以说是.个记录集，它是由.些T-SQL语句组成的代码块这些T-SQL语句代码像.个.法.样实现.些功能（对单表或多表的增删改查），然后再给这个代码块取.个名字，在.到这个 功能的时候调.他就.了。 2、存储过程是.个预编译的代码块，执.效率.较.,.个存储过程替代.量T_SQL语句，可以降低.络通信量，提.通信速率,可以.定程度上确保数据安全但是,在互联.项.中,其实是不太推荐存储过程的,.较出名的就是阿.的《Java开发.册》中禁.使.存储过程我个.的理解是,在互联.项.中,迭代太快,项.的.命周期也.较短,.员流动相.于传统的项.也更加频繁在这样的情况下,存储过程的管理确实是没有那么.便,同时,复.性也没有写在服务层那么好. 8.说.说三个范式 第.范式:每个列都不可以再拆分. 第.范式:.主键列完全依赖于主键,.不能是依赖于主键的.部分.第三范式:.主键列只依赖于主键,不依赖于其他.主键.在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有.够的理由..如性能.事实上我们经常会为了性能.妥协数据库的 设计. 9.MyBatis中的# 乱.了.个奇怪的问题..... 我只是想单独记录.下这个问题,因为出现频率太.了.#会将传.的内容当做字符串,.$会直接将传.值拼接在sql语句中.所以#可以在.定程度上预防sql注.攻击. 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.为什么你学不会递归?

2.

MySQL：Left Join避坑指南

3.AJAX请求真的不安全么?

4.

.个..不主动联系你还有机会吗?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

我和.试官的博弈：Redis篇 坚持就是胜利 Java后端3天前 点击上.Java后端，选择设为星标 优质.章，及时送达

今天，我.试了某..的java开发岗位，迎..来.位.尘仆仆的中年男.，..拿着屏幕还亮着的mac，他冲着我礼貌的笑了笑，然后说了句"不好意思，让你久等了"，然后.意我坐下，说："我们开始吧。看了你的简历，觉得你对redis应该掌握的不错，我们今天就来讨论下redis....."。我想："来就来，兵来将挡.来.掩"。 Redis是什么

.试官：你先来说下redis是什么吧我：（这不就是总结下redis的定义和特点嘛）Redis是C语.开发的.个开源的（遵从BSD协议）.性能键值对（key-value）的内存数据库，可以.作数据库、缓存、消息中间件等。它是.种NoSQL（not-only sql，泛指.关系型数据库）的数据库。我顿了.下，接着说：Redis作为.个内存数据库。1、性能优秀，数据在内存中，读写速度.常快，.持并发10WQPS；2、单进程单线程，是线程安全的，采.IO多路复.机制；3、丰富的数据类型，.持字符串（strings）、散列（hashes）、列表（lists）、集合（sets）、有序集合（sorted sets）等；4、.持数据持久化。可以将内存中数据保存在磁盘中，重启时加载；5、主从复制，哨兵，.可.；6、可以.作分布式锁；7、可以作为消息中间件使.，.持发布订阅 五种数据类型

.试官：总结的不错，看来是早有准备啊。刚来听你提到redis.持五种数据类型，那你能简单说下这五种数据类型吗?
我：当然可以，但是在说之前，我觉得有必要先来了解下Redis内部内存管理是如何描述这5种数据类型的。说着，我拿着笔给.试官画了.张图：

我：.先redis内部使..个redisObject对象来表.所有的key和value，redisObject最主要的信息如上图所.：type表..个value对象具体是何种数据类型，encoding是不同数据类型在redis内部的存储.式。.如：type=string表.value存储的是.个普通字符串，那么encoding可以是raw或者int。 我顿了.下，接着说：下.我简单说下5种数据类型： 1、string是redis最基本的类型，可以理解成与memcached.模.样的类型，.个key对应.个value。value不仅是 string，也可以是数字。string类型是.进制安全的，意思是redis的string类型可以包含任何数据，.如jpg图.或者序列化的对象。string类型的值最.能存储512M。2、Hash是.个键值（key-value）的集合。redis的hash是.个string的key和value的映射表，Hash特别适合存储对象。常.命令：hget,hset,hgetall等。 3、list列表是简单的字符串列表，按照插.顺序排序。可以添加.个元素到列表的头部（左边）或者尾部（右边）常.命令：lpush、rpush、lpop、rpop、lrange(获取列表.段)等。应.场景：list应.场景.常多，也是Redis最重要的数据结构之.，.如twitter的关注列表，粉丝列表都可以.list结构来实现。数据结构：list就是链

表，可以.来当消息队列.。redis提供了List的push和pop操作，还提供了操作某.段的api，可以直接查询或者删除某.段的元素。实现.式：redislist的是实现是.个双向链表，既可以.持反向查找和遍历，更.便操作，不过带来了额外的内存开销。 4、set是string类型的.序集合。集合是通过hashtable实现的。set中的元素是没有顺序的，.且是没有重复的。常.命令：sdd、spop、smembers、sunion等。应.场景：redisset对外提供的功能和list.样是.个列表，特殊之处在于set是.动去重的，.且set提供了判断某个成员是否在.个set集合中。 5、zset和set.样是string类型元素的集合，且不允许重复的元素。常.命令：zadd、zrange、zrem、zcard等。使.场景：sortedset可以通过..额外提供.个优先级（score）的参数来为成员排序，并且是插.有序的，即.动排序。当你需要.个有序的并且不重复的集合列表，那么可以选择sortedset结构。和set相.，sortedset关联了.个double类型权重的参数score，使得集合中的元素能够按照score进.有序排列，redis正是通过分数来为集合中的成员进.从.到.的排序。实现.式：Redis sortedset的内部使.HashMap和跳跃表(skipList)来保证数据的存储和有序，HashMap.放的是成员到score的映射，.跳跃表.存放的是所有的成员，排序依据是HashMap.存的score，使.跳跃表的结构可以获得.较.的查找效率，并且在实现上.较简单。

我：我之前总结了.张图，关于数据类型的应.场景，如果您感兴趣，可以去我的掘.看。。 数据类型应.场景总结

.试官：想不到你平时也下了不少.夫，那redis缓存你.定.过的吧我：.过的。。.试官：那你跟我说下你是怎么.的？我是结合springboot使.的。.般有两种.式，.种是直接通过RedisTemplate来使.，另.种是使.springcache集成 Redis（也就是注解的.式）。具体的代码我就不说了，在我的掘.中有.个demo（.下）。 Redis缓存

直接通过RedisTemplate来使.使.springcache集成Redispom.xml中加.以下依赖： org.springframework.boot spring-boot-starter-data-redis org.apache.commons commons-pool2 org.springframework.boot spring-boot-starter-web org.springframework.session spring-session-data-redis org.projectlombok lombok true org.springframework.boot spring-boot-starter-test test

spring-boot-starter-data-redis:在spr ingboot2.x以后底层不再使.Jed is，.是换成了Lettuce。

commons-pool2：.作red is连接池，如不引.启动会报错

spring-session-data-redis：spr ingsession引.，.作共享sess ion。配置.件app lication.yml的配置：

server: port:8082 servlet: session: timeout:30ms spring: cache: type:redis redis: host:127.0.0.1 port:6379 password: #redis默认情况下有 16个分.，这.配置具体使.的分.，默认为 0 database:0 lettuce:

pool:#连接池最.连接数(使.负数表.没有限制),默认 8 max-active:100 创建实体类User. java publicclassUserimplementsSerializable{ privatestaticfinallongserialVersionUID=662692455422902539L; privateIntegerid; privateStringname; privateIntegerage; publicUser(){ } publicUser(Integerid,Stringname,Integerage) {this.id=id;this.name=name;this.age=age; } publicIntegergetId(){returnid;} publicvoidsetId(Integerid){this.id=id;} publicStringgetName(){returnname;} publicvoidsetName(Stringname){this.name=name;} publicIntegergetAge() {returnage;} publicvoidsetAge(Integerage){this.age=age;} @Override publicStringtoString(){ return"User{"+ "id="+id+ ",name='"+name+'\''+ ",age="+age+ '}';} } RedisTemplate的使..式 默认情况下的模板只能.持RedisTemplate<String, String>，也就是只能存.字符串，所以.定义模板很有必要。添加配置类 RedisCacheConfig.java @Configuration @AutoConfigureAfter(RedisAutoConfiguration.class)publicclassRedisCacheConfig{ @BeanpublicRedisTemplate<String,Serializable>redisCacheTemplate(LettuceConnectionFactoryconnectionFactory){ RedisTemplate<String,Serializable>template=newRedisTemplate<> ();template.setKeySerializer(newStringRedisSerializer());template.setValueSerializer(newGenericJackson2JsonRedisSerializer());template.setConnectionFactory(connectionFactory); returntemplate; }} 测试类 @RestController @RequestMapping("/user")publicclassUserController{ publicstaticLoggerlogger=LogManager.getLogger(UserController.class); @Autowired privateStringRedisTemplatestringRedisTemplate; @Autowired privateRedisTemplate<String,Serializable>redisCacheTemplate; @RequestMapping("/test") publicvoidtest() {redisCacheTemplate.opsForValue().set("userkey",newUser(1,"张三",25));Useruser= (User)redisCacheTemplate.opsForValue().get("userkey"); } logger.info("当前获取对象：{}",user.toString()); 然后在浏览器访问，观察后台.志http://localhost:8082/user/test

使.springcache集成redis springcache具备很好的灵活性，不仅能够使.SPEL(springexpressionlanguage)来定义缓存的key和各种condition，还提供了开箱即.的缓存临时存储.案，也.持和主流的专业缓存如EhCache、Redis、Guava的集

成。定义接.UserService.java publicinterfaceUserService{ Usersave(Useruser); voiddelete(intid); Userget(Integerid);}接.实现类UserServiceImpl.java @Service publicclassUserServiceImplimplementsUserService{ publicstaticLoggerlogger=LogManager.getLogger(UserServiceImpl.class); privatestaticMap<Integer,User>userMap=newHashMap<>(); static{userMap.put(1,newUser(1,"肖战",25)); userMap.put(2,newUser(2,"王.博",26)); } userMap.put(3,newUser(3,"杨紫",24)); @CachePut(value="user",key="#user.id")@Override publicUsersave(Useruser) {userMap.put(user.getId(),user);logger.info("进.save.法，当前存储对象：{}",user.toString()); returnuser; } @CacheEvict(value="user",key="#id")@Override publicvoiddelete(intid){userMap.remove(id); } logger.info("进.delete.法，删除成功"); @Cacheable(value="user",key="#id")@Override publicUserget(Integerid){logger.info("进.get.法，当前获取对象：{}",userMap.get(id)==null?null:userMap.get(id).toString()); returnuserMap.get(id); }} 为了.便演.数据库的操作，这.直接定义了.个Map<Integer,User>userMap，这.的核.是三个注解@Cachable、@CachePut和@CacheEvict。测试类：UserController @RestController @RequestMapping("/user")publicclassUserController{ publicstaticLoggerlogger=LogManager.getLogger(UserController.class); @Autowired privateStringRedisTemplatestringRedisTemplate; @Autowired privateRedisTemplate<String,Serializable>redisCacheTemplate; @Autowired privateUserServiceuserService; @RequestMapping("/test") publicvoidtest(){redisCacheTemplate.opsForValue().set("userkey",newUser(1,"张三",25));Useruser=(User)redisCacheTemplate.opsForValue().get("userkey"); } logger.info("当前获取对象：{}",user.toString()); @RequestMapping("/add") publicvoidadd(){Useruser=userService.save(newUser(4,"李现",30)); } logger.info("添加的..信息：{}",user.toString()); @RequestMapping("/delete")publicvoiddelete(){userService.delete(4);} @RequestMapping("/get/{id}") publicvoidget(@PathVariable("id")StringidStr)throwsException{if(StringUtils.isBlank(idStr)){ } thrownewException("id为空"); Integerid=Integer.parseInt(idStr);Useruser=userService.get(id); } logger.info("获取的..信息：{}",user.toString()); } .缓存要注意，启动类要加上.个注解开启缓存 @SpringBootApplication(exclude=DataSourceAutoConfiguration.class)@EnableCaching publicclassApplication{ publicstaticvoidmain(String[]args){SpringApplication.run(Application.class,args);} } 1、先调.添加接.：http://localhost:8082/user/add

2、再调.查询接.，查询id=4的..信息：

可以看出，这.已经从缓存中获取数据了，因为上.步add.法已经把id=4的..数据放.了redis缓存3、调.删除.法，删除 id=4的..信息，同时清除缓存

4、再次调.查询接.，查询id=4的..信息：

没有了缓存，所以进.了get.法，从userMap中获取。 缓存注解 1、@Cacheable根据.法的请求参数对其结果进.缓存

key：缓存的key，可以为空，如果指定要按照SPEL表达式编写，如果不指定，则按照.法的所有参数进.组合。 value：缓存的名称，必须指定.少.个（如@Cacheable(value='user')或者@Cacheable(value={'user1','user2'})）condition：缓存的条件，可以为空，使.SPEL编写，返回true或者false，只有为true才进.缓存。2、@CachePut根据.法的请求参数对其结果进.缓存，和@Cacheable不同的是，它每次都会触发真实.法的调.。参数描述.上。 3、@CacheEvict根据条件对缓存进.清空

key：同上 value：同上 condition：同上 allEntries：是否清空所有缓存内容，缺省为false，如果指定为true，则.法调.后将.即清空所有缓存 beforeInvocation：是否在.法执.前就清空，缺省为false，如果指定为true，则在.法还没有执.的时候就清空缓存。缺省情况下，如果.法执.抛出异常，则不会清空缓存。 缓存问题

.试官：看了.下你的demo，简单易懂。那你在实际项.中使.缓存有遇到什么问题或者会遇到什么问题你知道吗？我：缓存和数据库数据.致性问题：分布式环境下.常容易出现缓存和数据库间数据.致性问题，针对这.点，如果项.对缓存的要求是强.致性的，那么就不要使.缓存。我们只能采取合适的策略来降低缓存和数据库间数据不.致的概率，..法保证两者间的强.致性。合适的策略包括合适的缓存更新策略，更新数据库后及时更新缓存、缓存失败时增加重试机制。.试官：Redis雪崩了解吗？我：我了解的，.前电商..以及热点数据都会去做缓存，.般缓存都是定时任务去刷新，或者查不到之后去更新缓存的，定时任务刷新就有.个问题。举个栗.：如果..所有Key的失效时间都是12.时，中午12点刷新的，我零点有个.促活动.量..涌.，假设每秒6000个请求，本来缓存可以抗住每秒5000个请求，但是缓存中所有Key都失效了。此时6000个/秒的请求全部落在了数据库上，数据库必然扛不住，真实情况可能DBA都没反应过来直接挂了，此时，如果没什么特别的.案来处理，DBA很着急，重启数据库，但是数据库...被新流量给打死了。这就是我理解的缓存雪崩。我.

想：同.时间..积失效，瞬间Redis跟没有.样，那这个数量级别的请求直接打到数据库.乎是灾难性的，你想想如果挂的是.个..服务的库，那其他依赖他的库所有接..乎都会报错，如果没做熔断等策略基本上就是瞬间挂..的节奏，你怎么重启..都会把你打挂，等你重启好的时候，..早睡觉去了，临睡之前，骂骂咧咧“什么垃圾产品”。.试官摸摸了..的头发：嗯，还不错，那这种情况你都是怎么应对的？我：处理缓存雪崩简单，在批量往Redis存数据的时候，把每个Key的失效时间都加个随机值就好了，这样可以保证数据不会再同.时间..积失效。

setRedis（key,value,time+Math.random()*10000）；如果Redis是集群部署，将热点数据均匀分布在不同的Redis库中也能避免全部失效。或者设置热点数据永不过期，有更新操作就更新缓存就好了（.如运维更新了..商品，那你刷下缓存就好了，不要设置过期时间），电商..的数据也可以.这个操作，保险。.试官：那你了解缓存穿透和击穿么，可以说说他们跟雪崩的区别吗？我：嗯，了解，先说下缓存穿透吧，缓存穿透是指缓存和数据库中都没有的数据，...（.客）不断发起请求，举个栗.：我们数据库的id都是从1.增的，如果发起id=-1的数据或者id特别.不存在的数据，这样的不断攻击导致数据库压.很.，严重会击垮数据库。我.接着说：.于缓存击穿嘛，这个跟缓存雪崩有点像，但是.有.点不.样，缓存雪崩是因为..积的缓存失效，打崩了DB，.缓存击穿不同的是缓存击穿是指.个Key.常热点，在不停地扛着.量的请求，.并发集中对这.个点进.访问，当这个Key在失效的瞬间，持续的.并发直接落到了数据库上，就在这个Key的点上击穿了缓存。.试官露出欣慰的眼光：那他们分别怎么解决？我：缓存穿透我会在接.层增加校验，.如..鉴权，参数做校验，不合法的校验直接return，.如id做基础校验，id<=0直接拦截。.试官：那你还有别的.法吗？我：我记得Redis.还有.个.级.法**布隆过滤器（Bloom Filter）**这个也能很好的预防缓存穿透的发.，他的原理也很简单，就是利..效的数据结构和算法快速判断出你这个Key是否在数据库中存在，不存在你return就好了，存在你就去查DB刷新KV再return。缓存击穿的话，设置热点数据永不过期，或者加上互斥锁就搞定了。作为暖男，代码给你准备好了，拿.不谢。

```
publicstaticStringgetData(Stringkey)throwsInterruptedException{ //从Redis查询数据 Stringresult=getDataByKV(key); //参数校验 if(StringUtils.isBlank(result)){try{ //获得锁 if(reenLock.tryLock()){ //去数据库查询 result=getDataByDB(key); //校验 if(StringUtils.isNotBlank(result)){ //插进缓存 setDataToKV(key,result);}}else{ //睡.会再拿 Thread.sleep(100L); result=getData(key);}}finally{ //释放锁 reenLock.unlock(); }}returnresult; }
```

.试官：嗯嗯，还不错。 Redis为何这么快

.试官：redis作为缓存.家都在.，那redis.定很快咯？我：当然了，官.提供的数据可以达到100000+的QPS（每秒内的查询次数），这个数据不.Memcached差！.试官：redis这么快，它的“多线程模型”你了解吗？（露出邪魅.笑）我：您是想问Redis这么快，为什么还是单线程的吧。Redis确实是单进程单线程的模型，因为Redis完全是基于内存的操作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的..或者.络带宽。既然单线程容易实现，.且CPU不会成为瓶颈，那就顺理成章的采.单线程的.案了（毕竟采.多线程会有很多.烦）。

.试官：嗯，是的。那你能说说Redis是单线程的，为什么还能这么快吗？ 我：可以这么说吧。第.：Redis完全基于内存，绝.部分请求是纯粹的内存操作，.常迅速，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是O(1)。第.：数据结构简单，对数据操作也简单。第三：采.单线程，避免了不必要的上下.切换和竞争条件，不存在多线程导致的CPU切换，不.去考虑各种锁的问题，不存在加锁释放锁操作，没有死锁问题导致的性能消耗。第四：使.多路复.IO模型，.阻塞IO。

Redis和Memcached的区别

.试官：嗯嗯，说的很详细。那你为什么选择Redis的缓存.案.不.memcached呢 我：1、存储.式上：memcache会把数据全部存在内存之中，断电后会挂掉，数据不能超过内存...。redis有部分数据存在硬盘上，这样能保证数据的持久性。2、数据.持类型上：memcache对数据类型的.持简单，只.持简单的key-value，，.redis.持五种数据类型。

3、使.底层模型不同：它们之间底层实现.式以及与客.端之间通信的应.协议不.样。redis直接..构建了VM机制，因为.般的系统调.系统函数的话，会浪费.定的时间去移动和请求。4、value的..：redis可以达到1GB，.memcache只有1MB。
淘汰策略

.试官：那你说说你知道的redis的淘汰策略有哪些？我：Redis有六种淘汰策略

补充.下：Redis4.0加.了LFU(least frequency use)淘汰策略，包括volatile-lfu和allkeys-lfu，通过统计访问频率，将访问频率最少，即最不经常使.的KV淘汰。 持久化

.试官：你对redis的持久化机制了解吗？能讲.下吗？我：redis为了保证效率，数据缓存在了内存中，但是会周期性的把更新的数据写.磁盘或者把修改操作写.追加的记录.件中，以保证数据的持久化。Redis的持久化策略有两种：1、RDB：

快照形式是直接把内存中的数据保存到.个dump的.件中，定时保存，保存策略。2、AOF：把所有的对Redis的服务器进.修改的命令都存到.个.件.，命令的集合。Redis默认是快照RDB的持久化.式。当Redis重启的时候，它会优先使.AOF.件来还原数据集，因为AOF.件保存的数据集通常.RDB.件所保存的数据集更完整。你甚.可以关闭持久化功能，让数据只在服务器运.时存。.试官：那你再说下RDB是怎么.作的？我：默认Redis是会以快照"RDB"的形式将数据持久化到磁盘的.个.进制.件dump.rdb。.作原理简单说.下：当Redis需要做持久化时，Redis会fork.个.进程，.进程将数据写到磁盘上.个临时RDB.件中。当.进程完成写临时.件后，将原来的RDB替换掉，这样的好处是可以copy-on-write。我：RDB的优点是：这种.件.常适合.于备份：.如，你可以在最近的24.时内，每.时备份.次，并且在每个.的每.天也备份.个RDB.件。这样的话，即使遇上问题，也可以随时将数据集还原到不同的版本。RDB.常适合灾难恢复。RDB

的缺点是：如果你需要尽量避免在服务器故障时丢失数据，那么RDB不合适你。.试官：那你要不再说下AOF？？

我：（说就.起说下吧）使.AOF做持久化，每.个写命令都通过write函数追加到appendonly.aof中，配置.式如下 appendfsyncyes appendfsyncalways#每次有数据修改发.时都会写.AOF.件。 appendfsynceverysec#每秒钟同步.次，该策略为AOF的缺省策略。 AOF可以做到全程持久化，只需要在配置中开启 appendonly yes。这样redis每执..个修改数据的命令，都会把它添加到AOF.件中，当redis重启时，将会读取AOF.件进.重放，恢复到redis关闭前的最后时刻。

我顿了.下，继续说：使.AOF的优点是会让redis变得.常耐久。可以设置不同的fsync策略，aof的默认策略是每秒钟fsync.次，在这种配置下，就算发.故障停机，也最多丢失.秒钟的数据。缺点是对于相同的数据集来说，AOF的.件体积通常要.于RDB.件的体积。根据所使.的fsync策略，AOF的速度可能会慢于RDB。.试官.问：你说了这么多，那我该.哪.个呢？我：如果你.常关.你的数据，但仍然可以承受数分钟内的数据丢失，那么可以额只使.RDB持久。AOF将Redis执.的每.条命令追加到磁盘中，处理巨.的写.会降低Redis的性能，不知道你是否可以接受。数据库备份和灾难恢复：定时.成RDB快照.常便于进.数据库备份，并且RDB恢复数据集的速度也要.AOF恢复的速度快。当然了，redis.持同时开启RDB和AOF，系统重启后，redis会优先使.AOF来恢复数据，这样丢失的数据会最少。 主从复制

.试官：redis单节点存在单点故障问题，为了解决单点问题，.般都需要对redis配置从节点，然后使.哨兵来监听主节点的存活状态，如果主节点挂掉，从节点能继续提供缓存功能，你能说说redis主从复制的过程和原理吗？我有点懵，这个说来就话.了。但幸好提前准备了：主从配置结合哨兵模式能解决单点故障问题，提.redis可.性。从节点仅提供读操作，主节点提供写操作。对于读多写少的状况，可给主节点配置多个从节点，从.提.响应效率。我顿了.下，接着说：关于复制过程，是这样的：1、从节点执.slaveof[masterIP][masterPort]，保存主节点信息2、从节点中的定时任务发现主节点信息，建.和主节点的socket连接 3、从节点发送Ping信号，主节点返回Pong，两边能互相通信4、连接建.后，主节点将所有数据发送给从节点（数据同步）5、主节点把当前的数据同步给从节点后，便完成了复制的建.过程。接下来，主节点就会持续的把写命令发送给从节点，保证主从数据.致性。.试官：那你能详细说下数据同步的过程吗？（我.想：这也问的太细了吧）我：可以。redis2.8之前使.sync[runId][offset]同步命令，redis2.8之后使.psync[runId][offset]命令。两者不同在于，sync命令仅.持全量复制过程，psync.持全量和部分复制。介绍同步之前，先介绍.个概念：runId：每个redis节点启动都会.成唯.的uuid，每次redis重启后，runId都会发.变化。offset：主节点和从节点都各.维护..的主从复制偏移量offset，当主节点有写.命令时，offset=offset+命令的字节.度。从节点在收到主节点发送的命令后，也会增加..的offset，并把..的offset发送给主节点。这样，主节点同时保存..的offset和从节点的offset，通过对.offset来判断主从节点数据是否.致。repl_backlog_size：保存在主节点上的.个固定.度的先进先出队列，默认..是1MB。（1）主节点发送数据给从节点过程中，主节点还会进..些写操作，这时候的数据存储在复制缓冲区中。从节点同步主节点数据完成后，主节点将缓冲区的数据继续发送给从节点，.于部分复制。（2）主节点响应写命令时，不但会把命名发送给从节点，还会写.复制积压缓冲区，.于复制命令丢失的数据补救。

上.是psync的执.流程：从节点发送psync[runId][offset]命令，主节点有三种响应：（1）FULLRESYNC：第.次连接，进.全量复制（2）CONTINUE：进.部分复制（3）ERR：不.持psync命令，进.全量复制

.试官：很好，那你能具体说下全量复制和部分复制的过程吗？我：可以

上.是全量复制的流程。主要有以下.步： 1、从节点发送psync?-1命令（因为第.次发送，不知道主节点的runId，所以为?，因为是第.次复制，所以offset=-1）。 2、主节点发现从节点是第.次复制，返回FULLRESYNC {runId} {o.set}，runId是主节点的runId，o.set是主节点.前的 offset。 3、从节点接收主节点信息后，保存到info中。 4、主节点在发送FULLRESYNC后，启动bgsave命令，.成RDB.件（数据持久化）。 5、主节点发送RDB.件给从节点。到从节点加载数据完成这段期间主节点的写命令放.缓冲区。 6、从节点清理..的数据库数据。 7、从节点加载RDB.件，将数据保存到..的数据库中。 8、如果从节点开启了AOF，从节点会异步重写AOF.件。 关于部分复制有以下.点说明：1、部分复制主要是Redis针对全量复制的过.开销做出的.种优化措施，使.psync[runId][offset]命令实现。当从节点正在复制主节点时，如果出现.络闪断或者命令丢失等异常情况时，从节点会向主节点要求补发丢失的命令数据，主节点的复制积压缓冲区将这

部分数据直接发送给从节点，这样就可以保持主从节点复制的.致性。补发的这部分数据.般远远.于全量数据。2、主从连接中断期间主节点依然响应命令，但因复制连接中断命令.法发送给从节点，不过主节点内的复制积压缓冲区依然可以保存最近.段时间的写命令数据。3、当主从连接恢复后，由于从节点之前保存了..已复制的偏移量和主节点的运.ID。因此会把它们当做psync参数发送给主节点，要求进.部分复制。4、主节点接收到psync命令后.先核对参数runId是否与...致，如果.致，说明之前复制的是当前主节点；之后根据参数offset在复制积压缓冲区中查找，如果 offset之后的数据存在，则对从节点发送+COUTINUE命令，表.可以进.部分复制。因为缓冲区..固定，若发.缓冲溢出，则进.全量复制。5、主节点根据偏移量把复制积压缓冲区.的数据发送给从节点，保证主从复制进.正常状态。 哨兵

.试官：那主从复制会存在哪些问题呢？我：主从复制会存在以下问题：1、.旦主节点宕机，从节点晋升为主节点，同时需要修改应..的主节点地址，还需要命令

所有从节点去复制新的主节点，整个过程需要...预。2、主节点的写能.受到单机的限制。3、主节点的存储能.受到单机的限制。4、原.复制的弊端在早期的版本中也会.较突出，.如：red is复制中断后，从节点会发起psync。此时如果同步不成功，则会进.全量同步，主库执.全量备份的同时，可能会造成毫秒或秒级的卡顿。.试官：那.较主流的解决.案是什么呢？我：当然是哨兵啊。.试官：那么问题.来了。那你说下哨兵有哪些功能？

我：如图，是Red is Sentinel is Sentinel （哨兵）的架构图。Red（哨兵）主要功能包括主节点存活检测、主从运.情况检测、.动故障转移、主从切换。Red is Sentinel最.配置是.主.从。Red is的Sent inel系统可以.来管理多个Red is服务器，该系统可以执.以下四个任务：1、监控：不断检查主服务器和从服务器是否正常运.。2、通知：当被监控的某个red is服务器出现问题，Sent inel通过AP I脚本向管理员或者其他应.程序发出通知。3、.动故障转移：当主节点不能正常.作时，Sent inel会开始.次.动的故障转移操作，它会将与失效主节点是主从关系的其中.个从节点升级为新的主节点，并且将其他的从节点指向新的主节点，这样...预就可以免了。4、配置提供者：在Red is Sentinel模式下，客.端应.在初始化时连接的是Sent inel节点集合，从中获取主节点的信息。.试官：那你能说下哨兵的.作原理吗？我：话不多说，直接上图：

1、每个Sent inel节点都需要定期执.以下任务：每个Sent inel以每秒.次的频率，向它所知的主服务器、从服务器以及其他的 Sentinel实例发送.个P ING命令。（如上图）

2、如果.个实例距离最后.次有效回复P ING命令的时间超过down-a fter-milliseconds所指定的值，那么这个实例会被Sentinel标记为主观下线。（如上图）

3、如果.个主服务器被标记为主观下线，那么正在监视这个服务器的所有Sent inel节点，要以每秒.次的频率确认主服务器的确进.了主观下线状态。

4、如果.个主服务器被标记为主观下线，并且有.够数量的Sent inel（.少要达到配置.件指定的数量）在指定的时间范围内同意这.判断，那么这个主服务器被标记为客观下线。

5、.般情况下，每个Sent inel会以每10秒.次的频率向它已知的所有主服务器和从服务器发送 INFO命令，当.个主服务器被标记为客观下线时，Sent inel向下线主服务器的所有从服务器发送 INFO命令的频率，会从10秒.次改为每秒.次。

6、Sent inel和其他Sent inel协商客观下线的主节点的状态，如果处于SDOWN状态，则投票.动选出新的主节点，将剩余从节点指向新的主节点进.数据复制。

7、当没有.够数量的Sentinel同意主服务器下线时，主服务器的客观下线状态就会被移除。当主服务器重新向Sentinel的PING命令返回有效回复时，主服务器的主观下线状态就会被移除。

.试官：不错，.试前没少下.夫啊，今天Redis这关你过了，明天找个时间我们再聊聊其他的。（露出欣慰的微笑）我：没问题。 总结 本.在.次.试的过程中讲述了Redis是什么，Redis的特点和功能，Redis缓存的使.，Redis为什么能这么快，Redis缓存的淘汰策略，持久化的两种.式，Redis.可.部分的主从复制和哨兵的基本原理。只要功夫深，铁杵磨成针，平时准备好，.试不.慌。虽然.试不.定是这样问的，但万变不离其"宗"。（笔者觉得这种问答形式的博客很不错，可读性强.且读后记的.较深刻） 来源：坚持就是胜利 链接：juejin.im/post/5dccf260f265da0bf66b626d -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。同时标星（置顶）本公众号可以第.时间接受到博.推送。 推荐阅读 1. 三个统.

2.

今天聊聊Java序列化

3.41道SpringBoot.试题，帮你整理好了！

4.

Zookeeper...章

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 技术群群友的.次阿..试经历 Java后端 2019-09-16 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

最近，收到技术交流群..个.伙.的报喜：群主，按照你之前说的，经过3个.的坚持，终于斩获了阿.的o.er。 虽然只是P6，但也满.了，毕竟经验有限，终于可以摆脱之前的那些CRUD重复劳动了，可以去..学习.逼技术了。

看完之后，我眼前飘过..字，"年纪轻轻福报."不过还是回复："恭喜，恭喜!!!.试都问了些什么？".伙.："就是你之前说的那些东西，数据结构、锁实现、cas原理，volatile使.场景，还有我在项.中不是.到redis嘛，也被重点问 了下，.路问到了很底层的实现细节，没能给出答案。" .伙.："对了，我发现阿..试官会对.个知识点.直问问问问，问到你不会为.，然后再问别的，不过还好我看过.些源码，虽然没能扛到最后，但也回答了差不多""嗯，之前就和你说过，对于现阶段的你来讲，多看看底层实现，才是最重要的，性价.最."其实，在互联.公司，很多.会被公司的技术和业务牵着.，进..些重复的crud.作。 那些底层的框架由专.的中间件团队负责，你只需简单的引.直接.就.，强.的封装性让你不.感知内部的复杂实现，即可完成.系列.并发操作和稳定性保障，从.忽略了底层技术，在.试的时候经常被扣上"技术深度不够"的帽.。

所以，写业务需求的同学要实现技术提升，除了寻找更.规模的业务外，要时刻保持跳出当前层级和环境来思考的习惯。 .般最快捷的.式，就是阅读源码和框架，但是阅读源码和框架，不是.件容易的事情，需要.期的坚持。

为此，我在这.给.家准备了源码和框架的直播和视频资源和资料包，.家可以.上，查漏补缺。

添加.姐姐领取资料包和免费直播

微信号：neteasejvm 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 挑战10道超难Java.试题 Java后端 2019-09-04 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

译者：Yujiaao 来源：segmentfault.com/a/1190000019962661 原.：http://t.cn/AiH7NCW1 这是我收集的10个最棘.的Java.试问题列表。这些问题主要来. Java核.部分,不涉及JavaEE相关问题。你可能知道这些棘.的 Java问题的答案，或者觉得这些不.以挑战你的Java知识，但这些问题都是容易在各种 Java.试中被问到的，.且包括我的朋友和同事在内的许多程序员都觉得很难回答。 1.为什么等待和通知是在 Object类.不是Thread中声明的？ .个棘.的 Java问题，如果 Java编程语.不是你设计的，你怎么能回答这个问题呢。Java编程的常识和深.了解有助于回答这种棘.的 Java核...的.试问题。 为什么 wait，notify和notifyAll是在 Object类中定义的.不是在Thread类中定义 这是有名的 Java.试问题，招2~4年经验的到.级Java开发.员.试都可能碰到。 这个问题的好在它能反映了.试者对等待通知机制的了解,以及他对此主题的理解是否明确。就像为什么 Java中不.持多继承或者为什么 String在 Java中是.nal的问题.样，这个问题也可能有多个答案。 为什么在 Object类中定义wait和 notify.法，每个.都能说出.些理由。从我的.试经验来看,wait和no.ty仍然是.多数Java程序员最困惑的，特别是2到3年的开发.员，如果他们要求使. wait和notify,他们会很困惑。因此，如果你去参加Java.试，请确保对 wait和 notify机制有充分的了解，并且可以轻松地使. wait来编写代码，并通过.产者-消费者问题或实现阻塞队列等了解通知的机制。 为什么等待和通知需要从同步块或.法中调.,以及Java中的wait，sleep和 yield.法之间的差异，如果你还没有读过，你会觉得有趣。为何 wait，notify和 notifyAll属于Object类?为什么它们不应该在Thread类中?以下是我认为有意义的.些想法： 1)wait和 notify不仅仅是普通.法或同步.具，更重要的是它们是Java中两个线程之间的通信机制。对语.设计者..,如果不能通过 Java关键字(例如 synchronized)实现通信此机制，同时.要确保这个机制对每个对象可.,那么Object类则是的正确声明位置。记住同步和等待通知是两个不同的领域，不要把它们看成是相同的或相关的。同步是提供互斥并确保 Java类的线程安全，. wait和notify是两个线程之间的通信机制。

2)每个对象都可上锁，这是在 Object类.不是 Thread类中声明 wait和 notify的另.个原因。

3)在Java中为了进.代码的临界区，线程需要锁定并等待锁定，他们不知道哪些线程持有锁，.只是知道锁被某个线程持有，并且他们应该等待取得锁,.不是去了解哪个线程在同步块内，并请求它们释放锁定。

4)Java是基于 Hoare的监视器的思想。在Java中，所有对象都有.个监视器。

线程在监视器上等待，为执.等待，我们需要2个参数：.个线程 .个监视器(任何对象) 在Java设计中，线程不能被指定，它总是运.当前代码的线程。但是，我们可以指定监视器(这是我们称之为等待的对象)。这是.个很好的设计，因为如果

我们可以让任何其他线程在所需的监视器上等待，这将导致".侵"，导致在设计并发程序时会遇到困难。请记住，在Java中，所有在另.个线程的执.中侵.的操作都被弃.了(例如 stop.法)。 2.为什么Java中不.持多重继承？ 我发现这个 Java核.问题很难回答，因为你的答案可能不会让.试官满意，在.多数情况下，.试官正在寻找答案中的关键点，如果你提到这些关键点，.试官会很.兴。在 Java中回答这种棘.问题的关键是准备好相关主题,以应对后续的各种可能的问题。 这是.常经典的问题，与为什么 String在Java中是不可变的很类似;这两个问题之间的相似之处在于它们主要是由Java创作者的设计决策使然。 为什么Java不.持多重继承,可以考虑以下两点: 1)第.个原因是围绕钻.形继承问题产.的歧义，考虑.个类A有foo().法,然后 B和C派.. A,并且有..的 foo()实现，现在 D类使.多个继承派.. B和C，如果我们只引. foo(),编译器将.法决定它应该调.哪个 foo()。这也称为 Diamond问题，因为这个继承.案的结构类似于菱形，.下图: 1 A foo() 2 3 八 4 5 八 6 7 foo() B C foo() / / D foo( )

即使我们删除钻.的顶部 A类并允许多重继承，我们也将看到这个问题含糊性的..。如果你把这个理由告诉.试官，他会问为什么 C++可以.持多重继承.Java不.。嗯，在这种情况下，我会试着向他解释我下.给出的第.个原因，它不是因为技术难度,.是更多的可维护和更清晰的设计是驱动因素,虽然这只能由 Java.语设计师确认，我们只是推测。维基百科链接有.些很好的解释，说明在使.多重继承时，由于钻.问题，不同的语.地址问题是如何产.的。 2)对我来说第.个也是更有说服.的理由是，多重继承确实使设计复杂化并在转换、构造函数链接等过程中产.问题。假设你需要多重继承的情况并不多，简单起.，明智的决定是省略它。此外，Java可以通过使.接..持单继承来避免这种歧义。由于接.只有.法声明.且没有提供任何实现，因此只有.个特定.法的实现，因此不会有任何歧义。 3.为什么Java不.持运算符重载？ 另.个类似棘.的Java问题。为什么C++.持运算符重载.Java不.持?有.可能会说+运算符在 Java中已被重载.于字符串连接，不要被这些论据所欺骗。与C++不同，Java不.持运算符重载。Java不能为程序员提供.由的标准算术运算符重载，例如+，-，*和/等。如果你以前.过C++，那么Java与 C++相.少了很多功能，例如 Java不.持多重继承，Java中没有指针，Java中没有引.传递。另.个类似的问题是关于 Java通过引.传递，这主要表现为Java是通过值还是引.传参。虽然我不知道背后的真正原因，但我认为以下说法有些道理，为什么 Java不.持运算符重载。 1)简单性和清晰性。清晰性是Java设计者的.标之.。设计者不是只想复制语.，.是希望拥有.种清晰，真正.向对象的语..。添加运算符重载.没有它肯定会使设计更复杂，并且它可能导致更复杂的编译器,或减慢 JVM，因为它需要做额外的.作来识别运算符的实际含义，并减少优化的机会,以保证Java中运算符的.为。

2)避免编程错误。Java不允许..定义的运算符重载，因为如果允许程序员进.运算符重载，将为同.运算符赋予多种含义，这将使任何开发.员的学习曲线变得陡峭，事情变得更加混乱。据观察，当语..持运算符重载时，编程错误会增加，从.增加了开发和交付时间。由于 Java和 JVM已经承担了.多数开发.员的责任，如在通过提供垃圾收集器进.内存管理时，因为这个功能增加污染代码的机会,成为编程错误之源,因此没有多.意义。

3)JVM复杂性。从JVM的.度来看，.持运算符重载使问题变得更加困难。通过更直观，更.净的.式使..法重载也能实现同样的事情，因此不.持 Java中的运算符重载是有意义的。与相对简单的JVM相.，复杂的JVM可能导致JVM更慢，并为保证在 Java中运算符.为的确定性从.减少了优化代码的机会。

4)让开发.具处理更容易。这是在 Java中不.持运算符重载的另.个好处。省略运算符重载使语.更容易处理，这反过来.更容易开发处理语.的.具，例如 IDE或重构.具。Java中的重构.具远胜于 C++。

4.为什么 String在Java中是不可变的？ 我最喜欢的 Java.试问题，很棘.，但同时也.常有..些.试者也常问这个问题，为什么String在 Java中是 .nal的。 字符串在 Java中是不可变的，因为 String对象缓存在String池中。由于缓存的字符串在多个客.之间共享，因此始终存在.险，其中.个客.的操作会影响所有其他客..。例如，如果.段代码将String"Test"的值更改为"TEST"，则所有其他客.也将看到该值。由于 String对象的缓存性能是很重要的...，因此通过使String类不可变来避免这种.险。 同时，String是 .nal的，因此没有.可以通过扩展和覆盖.为来破坏String类的不变性、缓存、散列值的计算等。String类不可变的另.个原因可能是由于 HashMap。 由于把字符串作为 HashMap键很受欢迎。对于键值来说，重要的是它们是不可变的，以便.它们检索存储在 HashMap中的值对象。由于 HashMap的.作原理是散列，因此需要具有相同的值才能正常运..。如果在插.后修改了 String的内容，可变的String将在插.和检索时.成两个不同的哈希码，可能会丢失Map中的值对象。 如果你是印度板球迷，你可能能够与我的下.句话联系起来。字符串是Java的 VVSLaxman，即.常特殊的类。我还没有看到.个没有使. String编写的 Java程序。这就是为什么对String的充分理解对于 Java开发.员来说.常重要。 String作为数据类型，传输对象和中间...的重要性和流.性也使这个问题在Java.试中很常..。 为什么String在Java中是不可变的是Java中最常被问到的字符串访问问题之.，它.先讨论了什么是 String，Java中的String如何与 C和C++中的 String不同，然后转向在Java中什么是不可变对象，不可变对象有什么好处，为什么要使.它们以及应该使.哪些场景。 这个问题有时也会问："为什么String在 Java中是.nal的"。在类似的说明中，如果你正在准备Java.试，我建议你看看《Java程序员.试宝典(第4版)》，这是.级和中级Java程序员的优秀资源。它包含来.所有重要 Java主题的问题，包括多线程，集合，GC，JVM内部以及 Spring和 Hibernate框架等。 正如我所说，这个问题可能有很多可能的答案，.

String类的唯.设计者可以放.地回答它。我在 JoshuaBloch的E.ective Java书中期待.些线索，但他也没有提到它。我认为以下.点解释了为什么 String类在 Java中是不可变的或 .nal的： 1)想象字符串池没有使字符串不可变，它根本不可能，因为在字符串池的情况下，.个字符串对象/.字，例如"Test"已被许多参考变量引.，因此如果其中任何.个更改了值，其他参数将.动受到影响，即假设

现在字符串 B调. "Test".toUpperCase(),将同.个对象改为"TEST"，所以 A也是"TEST"，这不是期望的结果。下图显.了如何在堆内存和字符串池中创建字符串。

2)字符串已被.泛.作许多 Java类的参数，例如，为了打开.络连接，你可以将主机名和端.号作为字符串传递，你可以将数据库 URL作为字符串传递,以打开数据库连接，你可以通过将.件名作为参数传递给 FileI/O类来打开Java中的任何.件。如果 String不是不可变的，这将导致严重的安全威胁，我的意思是有.可以访问他有权授权的任何.件，然后可以故意或意外地更改.件名并获得对该.件的访问权限。由于不变性，你.需担.这种威胁。这个原因也说明了，为什么 String在 Java中是最终的，通过使 java.lang.String .nal，Java设计者确保没有.覆盖String类的任何.为。

3)由于Str ing是不可变的，它可以安全地共享许多线程，这对于多线程编程.常重要.并且避免了Java中的同步问题，不变性也使得String实例在 Java中是线程安全的，这意味着你不需要从外部同步String操作。关于String的另.个要点是由截取字符串 SubString引起的内存泄漏，这不是与线程相关的问题，但也是需要注意的。

4)为什么 String在 Java中是不可变的另.个原因是允许 String缓存其哈希码，Java中的不可变 String缓存其哈希码，并且不会在每次调.String的 hashcode.法时重新计算，这使得它在 Java中的 HashMap中使.的 HashMap键.常快。简..之，因为 String是不可变的，所以没有.可以在创建后更改其内容，这保证了String的 hashCode在多次调.时是相同的。

5)String不可变的绝对最重要的原因是它被类加载机制使.，因此具有深刻和基本的安全考虑。如果String是可变的，加载"java.io.Writer"的请求可能已被更改为加载"mil.vogoon.DiskErasingWriter"安全性和字符串池是使字符串不可变的主要原因。顺便说.句，上.的理由很好回答另.个Java.试问题:"为什么Strig在Java中是最终的"。要想是不 n.

可变的，你必须是最终的，这样你的.类不会破坏不变性。你怎么看？ 5.为什么 char数组. Java中的String更适合存储密码？ 另.个基于 String的棘. Java问题，相信我只有很少的Java程序员可以正确回答这个问题。这是.个真正艰难的核.Java.试问题，并且需要对String的扎实知识才能回答这个问题。 这是最近在 Java.试中向我的.位朋友询问的问题。他正在接受技术主管职位的.试，并且有超过6年的经验。如果你还没有遇到过这种情况，那么字符数组和字符串可以.来存储.本数据，但是选择.个.不是另.个很难。但正如我的朋友所说，任何与 String相关的问题都必须对字符串的特殊属性有.些线索，.如不变性，他.它来说服访提问的.。在这.，我们将探讨为什么你应该使.char[]存储密码.不是String的.些原因。 字符串： 1)由于字符串在 Java中是不可变的，如果你将密码存储为纯.本，它将在内存中可.，直到垃圾收集器清除它.并且为了可重.性，会存在 String在字符串池中,它很可能会保留在内存中持续很.时间，从.构成安全威胁。 由于任何有权访问内存转储的.都可以以明.形式找到密码，这是另.个原因，你应该始终使.加密密码.不是纯.本。由于字符串是不可变的，所以不能更改字符串的内容，因为任何更改都会产.新的字符串，.如果你使.char[]，你就可以将所有元素设置为空.或零。因此，在字符数组中存储密码可以明显降低窃取密码的安全.险。 2)Java本.建议使. JPasswordField的getPassword().法，该.法返回.个char[]和不推荐使.的getTex().法，该.法以明.形式返回密码，由于安全原因。应遵循 Java团队的建议,坚持标准.不是反对它。

3)使.Str ing时，总是存在在.志.件或控制台中打印纯.本的.险，但如果使.Array，则不会打印数组的内容.是打印其内存位置。虽然不是.个真正的原因，但仍然有道理。

1 String strPassword ="Unknown"; 2 char [] charPassword = new char [] {'U'，'n'，'k'，'w'，'o'，'n'}; 3 4 System.out.println("字符密码："+ strPassword); System.out.println("字符密码："+ charPassword); 输出

我还建议使.散列或加密的密码.不是纯.本，并在验证完成后.即从内存中清除它。因此,在Java中,.字符数组.存储密码.字符串是更好的选择。虽然仅使.char[]还不够，还你需要擦除内容才能更安全。 6.如何使.双重检查锁定在 Java中创建线程安全的单例？ 这个Java问题也常被问:什么是线程安全的单例，你怎么创建它。好吧，在Java 5之前的版本,使.双重检查锁定创建单例 Singleton时，如果多个线程试图同时创建Singleton实例，则可能有多个Singleton实例被创建。从 Java5开始，使.Enum创建线程安全的Singleton很容易。但如果.试官坚持双重检查锁定，那么你必须为他们编写代码。记得使. volatile变量。 为什么枚举单例在Java中更好 枚举单例是使..个实例在 Java中实现单例模式的新.法。虽然Java中的单例模式存在很.时间,但枚举单例是相对较新的概念,在引.Enum作为关键字和功能之后,从Java5开始在实践中。本.与之前关于 Singleton的内容有些相关,其中讨论了有关 Singleton模式的.试中的常.问题,以及10个Java枚举.例,其中我们看到了如何通.枚举可以。这篇.章是关于为什么我们应该使.Eeame作为Java中的单例,它.传统的单例.法相.有什么好处等等。

Java枚举和单例模式 Java中的枚举单例模式是使.枚举在Java中实现单例模式。单例模式在Java中早有应.,但使.枚举类型创建单例模式时间却不..如果感兴趣,你可以了解下构建者设计模式和装饰器设计模式。 1)枚举单例易于书写 这是迄今为.最.的优势,如果你在Java 5之前.直在编写单例,你知道,即使双检查锁定,你仍可以有多个实例。虽然这个问题通过 Java内存模型的改进已经解决了,从Java5开始的volatile类型变量提供了保证,但是对于许多初学者来说,编写起来仍然很棘.。与同步双检查锁定相.,枚举单例实在是太简单了。如果你不相信,那就.较.下下.的传统双检查锁定单例和枚举单例的代码:在 Java中使.枚举的单例 这是我们通常声明枚举的单例的.式,它可能包含实例变量和实例.法,但为了简单起.,我没有使.任何实例.法,只是要注意,如果你使.的实例.法且该.法能改变对象的状态的话,则需要确保该.法的线程安全。默认情况下,创建枚举实例是线程安全的,但 Enum上的任何其他.法是否线程安全都是程序员的责任。 1 /** 2 *使. Java枚举的单例模式. 例 3 / 4 public enum EasySingleton{ 5 INSTANCE 6 ; } 你可以通过EasySingleton.INSTANCE来处理它,这.在单例上调.getInstance().法容易得多。 具有双检查锁定的单例.例 下.的代码是单例模式中双重检查锁定的.例,此处的getInstance().法检查两次,以查看 INSTANCE是否为空,这就是为什么它被称为双检查锁定模式,请记住,双检查锁定是代理之前Java5,但Java5内存模型中易失变量的.扰,它应该.作完美。 1 / 2 *单例模式.例,双重锁定检查 3 */ 4 public class DoubleCheckedLockingSingleton{ 5 private volatile DoubleCheckedLockingSingleton INSTANCE; 6 7 private DoubleCheckedLockingSingleton(){ 8 } 9 10 public DoubleCheckedLockingSingleton getInstance() 11 { if(INSTANCE == null)

12 13 14 15 16 17 18 19 20 { { if(INSTANCE null) synchronized(DoubleCheckedLockingSingleton.class){ //double checking Singleton instance if(INSTANCE == null) INSTANCE = new DoubleCheckedLockingSingleton(); } } }
return INSTANCE;
}
}

你可以调.DoubleCheckedLockingSingleton.getInstance()来获取此单例类的访问权限。 现在,只需查看创建延迟加载的线程安全的Singleton所需的代码量。使.枚举单例模式,你可以在..中具有该模式,因为创建枚举实例是线程安全的,并且由JVM进.。 .们可能会争辩说,有更好的.法来编写Singleton.不是双检查锁定.法,但每种.法都有..的优点和缺点,就像我最喜欢在类加载时创建的静态字段 Singleton,如下.所.,但请记住,这不是.个延迟加载单例: 单例模式.静态...法 这是我最喜欢的在Java中影响 Singleton模式的.法之.,因为 Singleton实例是静态的,并且最后.个变量在类.次 加载到内存时初始化,因此实例的创建本质上是线程安全的。 1 /** 2 *单例模式.例与静态...法 3 */ 4 public class Singleton{ 5 //initailzed during class loading 6 private static final Singleton INSTANCE = 7 8 //to prevent creating another instance of 9 private Singleton(){ 10 } 11 12 public static Singleton getSingleton() 13 {
14 return INSTANCE;
}
}

new Singleton(); Singleton 你可以调. Singleton.getSingleton()来获取此类的访问权限。 2)枚举单例..处理序列化 传统单例的另.个问题是,.旦实现可序列化接.,它们就不再是 Singleton,因为readObject().法总是返回.个新实例,就像 Java中的构造函数.样。通过使.readResolve().法,通过在以下.例中替换 Singeton来避免这种情况:

//readResolve to prevent another instance of Singleton

如果Singleton类保持内部状态,这将变得更加复杂,因为你需要标记为 transient(不被序列化),但使.枚举单例,序列化由JVM进.。 3)创建枚举实例是线程安全的 如第1点所述,因为Enum实例的创建在默认情况下是线程安全的,你.需担.是否要做双重检查锁定。 总之,在保证序列化和线程安全的情况下,使.两.代码枚举单例模式是在 Java5以后的世界中创建Singleton的最佳.式。你仍然可以使.其他流.的.法,如你觉得更好,欢迎讨论。 7.编写 Java程序时,如何在Java中创建死锁并修复它? 经典但核.Java.试问题之.。如果你没有参与过多线程并发 Java应.程序的编码，你可能会失败。 如何避免Java线程死锁? 如何避免 Java中的死锁? 是 Java.试的热.问题之.,也是多线程的编程中的重.味之.,主要在招.级程序员时容易被问到,且有很多后续问题。尽管问题看起来.常基本,但.多数Java开发.员.旦你开始深.,就会陷.困境。 .试问题总是以"什么是死锁? "开始 当两个或多个线程在等待彼此释放所需的资源(锁定)并陷..限等待即是死锁。它仅在多任务或多线程的情况下发.。 如何检测Java中的死锁? 虽然这可以有很多答案,但我的版本是.先我会看看代码,如果我看到.个嵌套的同步块，或从.个同步的.法调.其他同 步.法,或试图在不同的对象上获取锁,如果开发.员不是.常..，就很容易造成死锁。另.种.法是在运.应.程序时实际锁定时找到它,尝试采取线程转储,在 Linux中,你可以通过kill-3命令执.此操作,这将打印应.程序.志.件中所有线程的状态,并且你可以看到哪个线程被锁定在哪个线程对象上。 你可以使. fastthread.io .站等.具分析该线程转储,这些.具允许你上载线程转储并对其进.分析。另.种.法是使. jConsole或 VisualVM,它将显.哪些线程被锁定以及哪些对象被锁定。如果你有兴趣了解故障排除.具和分析线程转储的过程,我建议你看看UriahLevy在多元视觉(Pluralsight)上

《分析 Java线程转储》课程。旨在详细了解Java线程转储,并熟悉其他流.的.级故障排除.具。 编写.个将导致死锁的Java程序? .旦你回答了前.的问题,他们可能会要求你编写代码,这将导致Java死锁。这是我的版本之.

```
1 /** 2 * Java程序通过强制循环等待来创建死锁。 3 *
4 *
5 */
6 public class DeadLockDemo {
7
8 /
9 *
10 *此 .法请求两个锁 ,第 .个字符串 ,然后整数
11 */
12 public void method1()
13 {
14 synchronized (String.class) {
15 System.out.println("Aquired lock on String.class object")
16 ;
17
18 synchronized (Integer.class) {
19 System.out.println("Aquired lock on Integer.class object")
20 ;
21 }
22 }
23 }
24
25
26 /
27 *
28 *此 .法也请求相同的两个锁 ,但完全
29 *相反的顺序 ,即 .先整数 ,然后字符串。
30 *如果 .个线程持有字符串锁 ,则这会产 .潜在的死锁
31 *和其他持有整数锁 ,他们等待对 . ,永远。
32 */
33 public void method2()
34 {
35 synchronized (Integer.class) {
36 System.out.println("Aquired lock on Integer.class object")
37 ;
```

synchronized (String.class) { System.out.println("Aquired lock on String.class object") ; } } } } 如果method1 ()和method2 ()都由两个或多个线程调.,则存在死锁的可能性,因为如果线程 1在执. method1 ()时在St ing对象上获取锁,线程2在执.method2( )时在 Integer对象上获取锁,等待彼此释放 Integer和 Str ing上的锁以继续进..步,但这永远不会发.。

此图精确演.了我们的程序,其中.个线程在.个对象上持有锁,并等待其他线程持有的其他对象锁。 你可以看到,Thread1需要Thread2持有的Object2上的锁,.Thread2希望获得Thread1持有的 Object1上的锁。由于没有线程愿意放弃,因此存在死锁,Java程序被卡住。 其理念是,你应该知道使.常.并发模式的正确.法,如果你不熟悉这些模式,那么JosePaumard《应.于并发和多线程的常.Java模式》是学习的好起点。 如何避免Java中的死锁? 现在.试官来到最后.部分,在我看来,最重要的部分之.;如何修复代码中的死锁?或如何避免Java中的死锁? 如果你仔细查看了上.的代码,那么你可能已经发现死锁的真正原因不是多个线程,.是它们请求锁的.式,如果你提供有序访问,则问题将得到解决。下.是我的修复版本,它通过避免循环等待, .避免死锁,.不需要抢占,这是需要死锁的四个条件之.。 1 public class DeadLockFixed { 2 3 /* 4 * 5 *两种.法现在都以相同的顺序请求锁,.先采.整数,然后是 String。 6 *你也可以做反向,例如,第.个字符串,然后整数,
7 *只要两种.法都请求锁定,两者都能解决问题
8 *顺序.致。

```
9 */
10 public void method1()
11 {
12 synchronized (Integer.class) {
13 System.out.println("Aquired lock on Integer.class object")
14 ;
15
16 synchronized (String.class) {
17 System.out.println("Aquired lock on String.class object")
18 ;
19 }
20 }
21 }
22
```

现在没有任何死锁,因为两种.法都按相同的顺序访问 Integer和String类.本上的锁。因此,如果线程A在 Integer对象上获取锁,则线程 B不会继续,直到线程A释放 Integer锁,即使线程 B持有 String锁,线程A也不会被阻.,因为现在线程B不会期望线程A释放 Integer锁以继续。 8.如果你的Serializable类包含.个不可序列化的成员，会发.什么？你是如何解决的？ 任何序列化该类的尝试都会因NotSerializableException.失败，但这可以通过在Java中为static设置瞬态 (trancient)变量来轻松解决。 Java序列化相关的常.问题 Java序列化是.个重要概念,但它很少.作持久性解决.案,开发.员.多忽略了 Java序列化API。根据我的经验,Java序列化在任何 Java核.内容.试中都是.个相当重要的话题,在.乎所有的..试中,我都遇到过.两个Java序列化问题,我看过.次.试,在问.个关于序列化的问题之后候选.开始感到不.在,因为缺乏这..的经验。 他们不知道如何在Java中序列化对象,或者他们不熟悉任何Java .例来解释序列化,忘记了诸如序列化在Java中如何.作,什么是标记接.,标记接.的.的是什么,瞬态变量和可变变量之间的差异,可序列化接.具有多少种.法,在 Java中,Serializable和Externalizable有什么区别,或者在引.注解之后,为什么不. @Serializable注解或替换 Serializalbe接.。

在本.中,我们将从初学者和.级别进.提问,这对新.和具有多年 Java开发经验的.级开发.员同样有益。 关于Java序列化的10个.试问题 .多数商业项.使.数据库或内存映射.件或只是普通.件,来满.持久性要求,只有很少的项.依赖于 Java中的序列化过程。.论如何,这篇.章不是Java序列化教程或如何序列化在Java的对象,但有关序列化机制和序列化API的.试问题,这是值得去任何 Java.试前先看看以免让.些未知的内容惊到..。 对于那些不熟悉 Java序列化的.,Java序列化是.来通过将对象的状态存储到带有.ser扩展名的.件来序列化Java中的对象的过程,并且可以通过这个.件恢复重建Java对象状态,这个逆过程称为 deserialization。 什么是 Java序列化 序列化是把对象改成可以存到磁盘或通过.络发送到其他运.中的Java虚拟机的.进制格式的过程,并可以通过反序列化恢复对象状态.Java序列化API给开发.员提供了.个标准机制,通过 java.io.Serializable和java.io.Externalizable接.,ObjectInputStream及ObjectOutputStream处理对象序列化.Java程序员可.由选择基于类结构的标准序列化或是他们.定义的.进制格式,通常认为后者才是最佳实践,因为序列化的.进制.件格式成为类输出 API的.部分,可能破坏Java中私有和包可.的属性的封装. 如何序列化 让Java中的类可以序列化很简单.你的Java类只需要实现java.io.Serializable接.,JVM就会把Object对象按默认格式序列化.让.个类是可序列化的需要有意为之.类可序列会可能为是.个.期代价,可能会因此.限制你修改或改变其实现.当你通过实现添加接.来更改类的结构时,添加或删除任何字段可能会破坏默认序列化,这可以通过.定义.进制格式使不兼容的可能性最.化,但仍需要.量的努.来确保向后兼容性。序列化如何限制你更改类的能.的.个.例是 SerialVersionUID。 如果不显式声明 SerialVersionUID,则 JVM会根据类结构.成其结构,该结构依赖于类实现接.和可能更改的其他.个因素。假设你新版本的类.件实现的另.个接.,JVM将.成.个不同的SerialVersionUID的,当你尝试加载旧版本的程序序列化的旧对象时,你将获得.效类异常 InvalidClassException。 问题 1)Java中的可序列化接.和可外部接.之间的区别是什么？ 这是Java序列化访谈中最常问的问题。下.是我的版本Externalizable给我们提供writeExternal()和 readExternal().法,这让我们灵活地控制 Java序列化机制,.不是依赖于Java的默认序列化。正确实现Externalizable接.可以显著提.应.程序的性能。 问题 2)可序列化的.法有多少？如果没有.法,那么可序列化接.的.途是什么？ 可序列化 Serializalbe接.存在于java.io包中,构成了Java序列化机制的核..。它没有任何.法,在 Java中也称为标记接.。当类实现 java.io.Serializable接.时,它将在Java中变得可序列化,并指.编译器使.Java序列化机制序列化此对象。 问题 3)什么是serialVersionUID？如果你不定义这个,会发.什么？ 我最喜欢的关于Java序列化的问题.试问题之.。serialVersionUID是.个privatestatic.nallong型 ID,当它被印在对象上时,它通常是对象的哈希码,你可以使.serialver这个JDK.具来查看序列化对象的serialVersionUID。 SerialVerionUID.于对象的版本控制。也可以在类.件中指定 serialVersionUID。不指定serialVersionUID的后果是,当你添加或修改类中的任何字段时,则已序列化类将.法恢复,因为为新类和旧序列化对象.成的serialVersionUID将有所不同。Java序列化过程依赖于正确的序列化对象恢复状态的，并在序列化对象序列版本不匹配的情况下引发 java.io.InvalidClassException.效类异常。 问题 4)序列化时,你希望某些成员不要序列化？你如何实现它？

另.个经常被问到的序列化.试问题。这也是.些时候也问,如什么是瞬态trasient变量,瞬态和静态变量会不会得到序列化等,所以,如果你不希望任何字段是对象的状态的.部分,然后声明它静态或瞬态根据你的需要,这样就不会是在 Java序列化过程中被包含在内。 问题 5)如果类中的.个成员未实现可序列化接.,会发.什么情况? 关于Java序列化过程的.个简单问题。如果尝试序列化实现可序列化的类的对象,但该对象包含对不可序列化类的引.,则在运.时将引发不可序列化异常 NotSerializableException,这就是为什么我始终将.个可序列化警报(在我的代码注释部分中),代码注释最佳实践之.,指.开发.员记住这.事实,在可序列化类中添加新字段时要注意。 问题 6)如果类是可序列化的,但其超类不是,则反序列化后从超级类继承的实例变量的状态如何? Java序列化过程仅在对象层次都是可序列化结构中继续,即实现Java中的可序列化接.,并且从超级类继承的实例变量的值将通过调.构造函数初始化,在反序列化过程中不可序列化的超级类。.旦构造函数链接将启动,就不可能停.,因此,即使层次结构中较.的类实现可序列化接.,也将执.构造函数。正如你从陈述中看到的,这个序列化.试问题看起来.常棘.和有难度,但如果你熟悉关键概念,则并不难。 问题 7)是否可以.定义序列化过程,或者是否可以覆盖Java中的默认序列化过程? 答案是肯定的,你可以。我们都知道,对于序列化.个对象需调. ObjectOutputStream.writeObject(saveThisObject),并. ObjectInputStream.readObject()读取对象,但Java虚拟机为你提供的还有.件事,是定义这两个.法。如果在类中定义这两种.法,则 JVM将调.这两种.法,.不是应.默认序列化机制。你可以在此处通过执.任何类型的预处理或后处理任务来.定义对象序列化和反序列化的.为。 需要注意的重要.点是要声明这些.法为私有.法,以避免被继承、重写或重载。由于只有Java虚拟机可以调.类的私有.法,你的类的完整性会得到保留,并且Java序列化将正常.作。在我看来,这是在任何Java序列化.试中可以问的最好问题之.,.个很好的后续问题是,为什么要为你的对象提供.定义序列化表单? 问题 8)假设新类的超级类实现可序列化接.,如何避免新类被序列化? 在Java序列化中.个棘.的.试问题。如果类的Super类已经在Java中实现了可序列化接.,那么它在 Java中已经可以序列化,因为你不能取消接.,它不可能真正使它.法序列化类,但是有.种.法可以避免新类序列化。为了避免Java序列化,你需要在类中实现writeObject()和 readObject().法,并且需要从该.法引发不序列化异常 NotSerializableException。这是.定义 Java序列化过程的另.个好处,如上述序列化.试问题中所述,并且通常随着.试进度,它作为后续问题提出。 问题 9)在Java中的序列化和反序列化过程中使.哪些.法? 这是很常.的.试问题,在序列化基本上.试官试图知道:你是否熟悉 readObject()的.法、writeObject()、readExternal()和writeExternal()。Java序列化由java.io.ObjectOutputStream类完成。该类是.个筛选器流,它封装在较低级别的字节流中,以处理序列化机制。要通过序列化机制存储任何对象,我们调. ObjectOutputStream.writeObject(savethisobject),并反序列化该对象,我们称之为 ObjectInputStream.readObject().法。调.以 writeObject().法在 java中触发序列化过程。关于 readObject().法,需要注意的.点很重要.点是,它.于从持久性读取字节,并从这些字节创建对象,并返回.个对象,该对象需要类型强制转换为正确的类型。 问题 10)假设你有.个类,它序列化并存储在持久性中,然后修改了该类以添加新字段。如果对已序列化的对象进.反序列化,会发.什么情况? 这取决于类是否具有其..的 serialVersionUID。正如我们从上.的问题知道,如果我们不提供 serialVersionUID,则Java编译器将.成它,通常它等于对象的哈希代码。通过添加任何新字段,有可能为该类新版本.成的新 serialVersionUID与已序列化的对象不同,在这种情况下,Java序列化 API将引发 java.io.InvalidClassException,因此建议在代码中拥有..的 serialVersionUID,并确保在单个类中始终保持不变。 11) Java序列化机制中的兼容更改和不兼容更改是什么? 真正的挑战在于通过添加任何字段、.法或删除任何字段或.法来更改类结构, .法是使.已序列化的对象。根据 Java序列化规范,添加任何字段或.法都.临兼容的更改和更改类层次结构或取消实现的可序列化接.,有些接.在.兼容更改下。对于兼容和.兼容更改的完整列表,我建议阅读 Java序列化规范。 12)我们可以通过.络传输.个序列化的对象吗? 是的,你可以通过.络传输序列化对象,因为Java序列化对象仍以字节的形式保留,字节可以通过.络发送。你还可以将序列化对象存储在磁盘或数据库中作为 Blob。 13)在 Java序列化期间,哪些变量未序列化? 这个问题问得不同,但.的还是.样的,Java开发.员是否知道静态和瞬态变量的细节。由于静态变量属于类,.不是对象,因此它们不是对象状态的.部分,因此在Java序列化过程中不会保存它们。由于 Java序列化仅保留对象的状态,.不是对象本.。瞬态变量也不包含在 Java序列化过程中,并且不是对象的序列化状态的.部分。在提出这个问题之后,.试官会询问后续内容,如果你不存储这些变量的值,那么.旦对这些对象进.反序列化并重新创建这些变量,这些变量的价值是多少?这是你们要考虑的。 9.为什么Java中 wait.法需要在synchronized的.法中调.? 另.个棘.的核. Java问题，wait和notify。它们是在有 synchronized标记的.法或 synchronized块中调.的，因为wait和 modify需要监视对其上调. wait或 notify-get的Object。 .多数Java开发.员都知道对象类的wait()，notify()和notifyAll().法必须在Java中的 synchronized.法或 synchronized块中调.,但是我们想过多少次,为什么在Java中wait,notify和 notifyAll来.synchronized块或.法? 最近这个问题在Java.试中被问到我的.位朋友，他思索了.下，并回答说:如果我们不从同步上下.中调.wait()或 notify().法，我们将在Java中收到 IllegalMonitorStateException。 他的回答从实际效果上年是正确的，但.试官对这样的答案不会完全满意，并希望向他解释这个问题。.试结束后他和我讨论了同样的问题，我认为他应该告诉.试官关于 Java中wait()和notify()之间的竞态条件，如果我们不在同步.法或块中调.它们就可能存在。 让我们看看竞态条件如何在Java程序中发.。它也是流.的线程.试问题之.，并经常在电话和.对.的Java开发.员.试中出现。因此，如果你正在准备Java.试，那么你应该准备这样的问题，并且可以真正帮助你的.本书是《Java程序员.试公式书》的。这是.本罕.的书，涵盖了Java访谈的.乎所有重要主题，例如核.Java，多线程，IO和 NIO以及Spring和Hibernate等框架。你可以在这.查看。为什么要等待来. Java中的 synchronized.法的 wait.法为什么必须从Java中的 synchronized块或.法调.? 我们主要使. wait()，

notify()或 notifyAll().法.于 Java中的线程间通信。.个线程在检查条件后正在等待，例如，在经典的.产者 -消费者问题中，如果缓冲区已满，则.产者线程等待，并且消费者线程通过使.元素在缓冲区中创建空间后通知.产者线程。调.notify()或notifyAll().法向单个或多个线程发出.个条件已更改的通知，并且.旦通知线程离开synchronized块，正在等待的所有线程开始获取正在等待的对象锁定，幸运的线程在重新获取锁之后从wait().法返回并继续进.。

让我们将整个操作分成.步，以查看Java中wait()和notify().法之间的竞争条件的可能性，我们将使.ProduceConsumer线程.例更好地理解.案： Producer线程测试条件(缓冲区是是否完整)并确认必须等待(找到缓冲区已满)。 Consumer线程在使.缓冲区中的元素后设置条件。 Consumer线程调. notify().法;这是不会被听到的，因为 Producer线程还没有等待。 Producer线程调.wait().法并进.等待状态。 因此，由于竞态条件，我们可能会丢失通知，如果我们使.缓冲区或只使..个元素，.产线程将永远等待，你的程序将挂起。"在java同步中等待notify和 notifyall现在让我们考虑如何解决这个潜在的竞态条件？ 这个竞态条件通过使. Java提供的 synchronized关键字和锁定来解决。为了调.wait()，notify()或notifyAll(),在Java中，我们必须获得对我们调..法的对象的锁定。由于Java中的 wait().法在等待之前释放锁定并在从wait()返回之前重新获取锁定.法，我们必须使.这个锁来确保检查条件(缓冲区是否已满)和设置条件(从缓冲区获取元素)是原.的，这可以通过在 Java中使. synchronized.法或块来实现。 我不确定这是否是.试官实际期待的，但这个我认为.少有意义，请纠正我如果我错了，请告诉我们是否还有其他令.信服的理由调. wait()，notify()或 Java中的 notifyAll().法。 总结.下，我们.Java中的synchronized .法或 synchronized块调. Java中的wait()，notify()或 notifyAll().法来避免： 1)Java会抛出 IllegalMonitorStateException，如果我们不调.来.同步上下.的wait()，notify()或者notifyAll().法。

2)Javac中 wait和 notify.法之间的任何潜在竞争条件。

10.你能.Java覆盖静态.法吗？如果我在.类中创建相同的.法是编译时错误？ 不，你不能在Java中覆盖静态.法，但在.类中声明.个完全相同的.法不是编译时错误，这称为隐藏在Java中的.法。 你不能覆盖Java中的静态.法，因为.法覆盖基于运.时的动态绑定，静态.法在编译时使.静态绑定进.绑定。虽然可以在.类中声明.个具有相同名称和.法签名的.法，看起来可以在Java中覆盖静态.法，但实际上这是.法隐藏。Java不会在运.时解析.法调.，并且根据.于调.静态.法的Object类型，将调.相应的.法。这意味着如果你使..类的类型来调.静态.法，那么原始静态将从.类中调.，另...如果你使..类的类型来调.静态.法，则会调.来..类的.法。简..之，你.法在Java中覆盖静态.法。如果你使.像Eclipse或Netbeans这样的Java IDE，它们将显.警告静态.法应该使.类名.不是使.对象来调.，因为静态.法不能在Java中重写。 1 /** 2

- 3 * Java program which demonstrate that we can not override static method in Java. 4 * Had Static method can be overridden, with Super class type and sub class object 5 * static method from sub class would be called in our example, which is not the case. 6 */ 7 public class CanWeOverrideStaticMethod { 8 9 public static void main(String args[]) 10 { 11 12 Screen scrn = new ColorScreen(); 13 14 //if we can override static , this should call method from Child class 15 scrn.show(); //IDE will show warning, static method should be called from classname 16 17 } 18 19 } 20 21 class Screen{ 22 / 23
-

24 * public static method which can not be overridden in Java 25 */ 26 public static void show() 27 { 28 System.out.printf("Static method from parent class") 29 ; 30 } 31 } 32 33 class ColorScreen extends Screen{ 34 / 35

-

36 * static method of same name and method signature as existed in super 37 * class, this is not method overriding instead this is called 38 * method hiding in Java */ public static void show()

public static void show() { System.err.println("Overridden static method in Child Class in Java") ; } } 输出: Staticmethod fromparentclass 此输出确认你.法覆盖Java中的静态.法，并且静态.法基于类型信息.不是基于Object进.绑定。如果要覆盖静态mehtod，则会调..类或ColorScreen中的.法。这.切都在讨论中我们可以覆盖Java中的静态.法。我们已经确认没有，我们不能覆盖静态.法，我们只能在Java中隐藏静态.法。创建具有相同名称和mehtod签名的静态.法称为Java隐藏.法。IDE将显.警告："静态.法应该使.类名.不是使.对象来调.",因为静态.法不能在Java中重写。 这些是我的核.Java.试问题和答案的清单。对于有经验的程序员来说，.些Java问题看起来并不那么难，但对于Java中的中级和初学者来说，它们真的很难回答。顺便说.句，如果你在.试中遇到任何棘.的Java问题，请与我们分享。 如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「进群」即进.咱..的技术交流群。 推荐阅读 1. 什么时候进.分库分表？

2.

从Java程序员的.度理解加密

3.

IDEA中使.Git图.教程

4...梳理Redis基础

5.优化你的SpringBoot 6.数据库不使.外键的9个理由

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 最近.试Java后端开发的感受 hsm_computerJava后端 2019-11-28 点击上.Java后端，选择设为星标优质.章，及时送达

作者|hsm_computer链接| cnblogs.com/JavaArchitect/p/10011253.html 在上周，我密集.试了若.位Java后端的候选.，.作经验在3到5年间。我的标准其实不复杂：第.能.活，第.Java基础要好，第三最好熟悉些分布式框架，我相信其它公司招初级开发时，应该也照着这个标准来.的。 我也知道，不少候选.能.其实不差，但.试时没准备或不会说，这样的.可能在进团队.活后确实能达到期望，但可能就.法通过.试，但.试官总是只根据.试情况来判断。 但现实情况是，.多数.可能.试前没准备，或准备.法不得当。要知道，我们平时.活更偏重于业务，不可能.量接触到算法，数据结构，底层代码这类.试必问的问题点，换句话说，.试准备点和平时.作要点匹配度很.。 作为.试官，我只能根据候选.的回答来决定.试结果。不过，与..便...便，所以我在本..，将通过.些常.的问题来介绍.试的准备技巧。.家在看后.定会感叹：只要.法得当，准备.试第.不难，第..的时间也不会太多。 1、框架是重点，但别让.感觉你只会.寨别.的代码 在.试前，我会阅读简历以查看候选.在框架..的项.经验，在候选.的项.介绍的环节，我也会着重关注候选.最近的框架经验，.前.较热.的是SSM。 不过，.般.作在5年内的候选.，.多仅仅是能".寨"别.的代码，也就是说能在现有框架的基础上，照着别.写的流程，扩展出新的功能模块。.如要写个股票挂单的功能模块，是会模仿现有的下单流程，然后从前端到后端再到数据库，依样画葫芦写.遍，最多把功能相关的代码点改掉。 其实我们每个.都这样过来的，但在.试时，如果你仅仅表现出这样的能.，就和.多数.的.平差不多了，在这点就没法体现出你的优势了。 我们知道，如果单纯使.SSM框架，.多数项.都会有痛点。.如数据库性能差，或者业务模块.较复杂，并发量.较.，. SpringMVC.的Controller.法满.跳转的需求。所以我.般还会主动问：你除了依照现有框架写业务代码时，还做了哪些改动？ 我听到的回答有：增加了Redis缓存，以避免频繁调..些不变的数据。或者，在MyBitas的xml.，select语句where条件有 isnull，即这个值有就增加.个where条件，对此，会对任何.个where增加.个不带isnull的查询条件，以免该语句当传.参数都是null时，做全表扫描。或者，.脆说，后端异步返回的数据量很.，时间很.，我在项..就调.了异步返回的最.时间，或者对返回信息做了压缩处理，以增加.络传输性能。对于这个问题，我不在乎听到什么回答，我只关.回答符不符逻辑。.般只要答对，我就会给出"在框架层.有..的体会，有.定的了解"，否则，我就只会给出"只能在项.经理带领下编写框架代码，对框架本.了解不多"。 其实，在准备.试时，归纳框架.的要点并不难，我就不信所有.在做项.时.点积累也没，只要你说出来，可以说，这..你就碾压了将近7成的竞争者。 2、别单纯看单机版的框架，适当了解些分布式 此外，在描述项..框架技术时，最好你再带些分布式的技术。下.我列些.家可以准备的分布式技术。 1、反向代理..，nginx的基本配置，.如如何通过lua语.设置规则，如何设置session粘滞。如果可以，再看些nginx的底层，.如协议，集群设置，失效转移等。 2、远程调.dubbo..，可以看下dubbo和zookeeper整合的知识点，再深.步，了解下dubbo底层的传输协议和序列化.式。 3、消息队列..，可以看下kafka或任意.种组件的使..式，简单点可以看下配置，.作组的设置，再深.点，可以看下 Kafka集群，持久化的.式，以及发送消息是..连接还是短拦截。 以上仅仅是.3个组件举例，.家还可以看下Redis缓存，.志框架，MyCAT分库分表等。准备的.式有两.类，第.是要会说怎么.，这.较简单，能通过配置.件搭建成.个功能模块即可，第.是可以适当读些底层代码，以此了解下协议，集群和失效转移之类的.级知识点。 如果能在.试中侃侃.谈分布式组件的底层，那么得到的评价就会.较好了，.如"深.了解框架底层"，或"框架经验丰富"，这样就算去.试架构师也.了，更何况是.级开发。 3、数据库..，别就知道增删改查，得了解性能优化 在实际项..，.多数程序员.到的可能仅仅是增删改查，当我们.Mybatis时，这个情况更普遍。不过如果你.试时也这样表现，估计你的能.就和其它竞争者差不多了。 这..，你可以准备如下的技能。 1、SQL.级..，.如groupby,having，左连接，.查询（带in），.转列等.级.法。 2、建表..，你可以考虑下，你项.是.三范式还是反范式，理由是什么？ 3、尤其是优化，你可以准备下如何通过执.计划查看SQL语句改进点的.式，或者其它能改善SQL性能的.式（.如建索引等）。4、如果你感觉有能.，还可以准备些MySQL集群，MyCAT分库分表的技能。.如通过LVS+Keepalived实现MySQL负载均衡，MyCAT的配置.式。同样，如果可以，也看些相关的底层代码。 哪怕你在前三点表现.般，那么.少也能超越将近.般的候选.，尤其当你在SQL优化..表现.常好，那么你在.试.级开发时，数据库层..定是达标的，如果你连第四点也回答.常好，那么恭喜你，你在数据库..的能.甚.达到了初级架构的级别。 4、Java核...，围绕数据结构和性能优化准备.试题 Java核.这块，.上的.试题很多，不过在此之外，.家还应当着重关注集合（即数据结构）和多线程并发这两块，在此基础上，.家可以准备些设计模式和虚拟机的说辞。 下.列些我.般会问的部分问题： 1、Stringa="123";Stringb="123";a==b

的结果是什么？这包含了内存，String存储.式等诸多知识点。 2、HashMap.的hashcode.法和equal.法什么时候需要重写？如果不重写会有什么后果？对此.家可以进.步了解HashMap（甚.ConcurrentHashMap）的底层实现。 3、ArrayList和LinkedList底层实现有什么差别？它们各.适.于哪些场合？对此.家也可以了解下相关底层代码。 4、volatile关键字有什么作.？由此展开，.家可以了解下线程内存和堆内存的差别。 5、CompletableFuture，这个是JDK1.8.的新特性，通过它怎么实现多线程并发控制？ 6、JVM.，new出来的对象是在哪个区？再深..下，问下如何查看和优化JVM虚拟机内存。 7、Java的静态代理和动态代理有什么差别？最好结合底层代码来说。 通过上述的问题点，我其实不仅仅停留在“会.”级别，.如我不会问如何在ArrayList.放元素。.家可以看到，上述问题包含了“多线程并发”，“JVM优化”，“数据结构对象底层代码”等细节，.家也可以举.反三，通过看.些.级知识，多准备些其它类似.试题。 我们知道，.前Java开发是以Web框架为主，那么为什么还要问Java核.知识点呢？我这个是有切.体会的。 之前在我团队.，我.过两个.，.个是就会.活，具体表现是会.Java核.基本的API，.且也没有深.了解的意愿（估计不知道该怎么深.了解），另.位平时专.会看些Java并发，虚拟机等的.级知识。过了半年以后，后者的能.快速升级到.级开发，由于对JAVA核.知识点了解很透彻，所以看.些分布式组件的底层实现没什么.问题。.前者，.直在重复劳动，能.也只.直停留在“会.活”的层.。 .在现实的.试中，如果不熟悉Java核.知识点，估计升.级开发都难，更别说是.试架构师级别的岗位了。 5、Linux..，.少了解如何看.志排查问题如果候选.能证明..有“排查问题”和“解决问题”的能.，这绝对是个加分项，但怎么证明？ .前.多数的互联.项.，都是部署在Linux上，也就是说，.志都是在Linux，下.归纳些实际的Linux操作。 1、能通过less命令打开.件，通过Shift+G到达.件底部，再通过?+关键字的.式来根据关键来搜索信息。 2、能通过grep的.式查关键字，具体.法是,grep关键字.件名，如果要两次在结果.查找的话，就.grep关键字1.件名|关键字2--color。最后--color是.亮关键字。 3、能通过vi来编辑.件。 4、能通过chmod来设置.件的权限。 当然，还有更多更实.的Linux命令，但在实际.试过程中，不少候选.连.条linux命令也不知道。还是这句话，你哪怕知道些很基本的，也..般.强了。 6、通读.段底层代码，作为加分项 如何证明..对.个知识点.常了解?莫过于能通过底层代码来说明。我在和不少.作经验在5年之内的程序员沟通时，不少.认为这很难？确实，如果要通过阅读底层代码了解分布式组件，那难度不.，但如果如下部分的底层代码，并不难懂。 1、ArrayList,LinkedList的底层代码.，包含着基于数组和链表的实现.式，如果.家能以此讲清楚扩容，“通过枚举器遍历”等.式，绝对能证明..。 2、HashMap直接对应着Hash表这个数据结构，在HashMap的底层代码.，包含着hashcode的put，get等的操作，甚.在ConcurrentHashMap.，还包含着Lock的逻辑。我相信，如果.家在.试中，看看..ConcurrentHashMap，再结合在纸上边说边画，那.定能征服.试官。 3、可以看下静态代理和动态代理的实现.式，再深..下，可以看下SpringAOP.的实现代码。 4、或许SpirngIOC和MVC的底层实现代码.较难看懂，但.家可以说些关键的类，根据关键流程说下它们的实现.式。 其实准备的底层代码未必要多，.且也不限于在哪个..，.如集合.基于红.树的TreeSet，基于NIO的开源框架，甚.分布式组件的Dubbo，都可以准备。.且准备时未必要背出所有的底层（事实上很难做到），你只要能结合.些重要的类和.法，讲清楚思路即可（.如讲清楚HashMap如何通过hashCode快速定位）。 那么在.试时，如何找到个好机会说出你准备好的上述底层代码？在.试时，总会被问到集合，SpringMVC框架等相关知识点，你在回答时，顺便说.句，“我还了解这块的底层实现”，那么.试官.定会追问，那么你就可以说出来了。 不要.看这个对候选.的帮助，.旦你讲了，只要意思到位，那么最少能得到个“肯积极专业”的评价，如果描述很清楚，那么评价就会升级到“熟悉Java核.技能（或SpringMVC），且基本功扎实”。要知道，.试中，很少有.能讲清楚底层代码，所以你抛出了这个话题，哪怕最后没达到预期效果，.试官也不会由此对你降低评价。所以说，准备这块绝对是“有百利...害”的挣钱买卖。 7、.切的.切，把上述技能嵌.到你做过的项.. 在.试过程中，我经常会听到.些.较遗憾的回答，.如候选.对SQL优化技能讲得头头是道，但最后得知，这是他平时.学时掌握的，并没.在实际项..。 当然这总.不说要好，所以我会写下“在平时.学过SQL优化技能”，但如果在项..实践过，那么我就会写下“有实际数据库SQL优化的技能”。.家可以对.下两者的差别，.个是偏重理论，.个是直接能.活了。其实，很多场景.，我就不信在实际项...定没有实践过SQL优化技能。 从这个案例中，我想告诉.家的是，你之前费了千.万苦（其实.法.向得到，也不.费太.精.）准备的很多技能和说辞，最后应该落实到你的实际项..。 .如你有过在Linux.志.查询关键字排查问题的经验，在描述时你可以带.句，在之前的项..我就这样.的。.如，你通过看底层代码，了解了TreeSet和HashSet的差别以及它们的适.范围，那么你就可以回想下你之前做的项.，是否有个场景仅仅适.于TreeSet?如果有，那么你就可以适当描述下项.的需求，然后说，通过读底层代码，我了解了两者的差别，.且在这个实际需求.，我就.了TreeSet，.且我还专.做了对.性试验，发现.TreeSet.HashSet要.xx个百分点。 请记得，“实践经验”.定.“理论经验”值钱，.且.多数你知道的理论上的经验，.定在你的项...过。所以，如果你仅仅让.试官感觉你只有“理论经验”，那就太亏了。 8、.结：本.更多讲述的准备.试的.法 本.给出的.试题并不多，但本.并没有打算给出太多的.试题。从本..，.家更多看到的是.试官发现的诸多候选.的痛点。 本.的.意是让.家别再重蹈别.的覆辙，这还不算，本.还给出了不少准备.试的.法。你的能.或许.别.出众，但如果你准备.试的.式和别.差不多，或者就拿你在项...的活来说事，.没有归纳出你在项.中的亮点，那么.试官还真的会看扁你。 本.欢迎转载，不过请注明.章来源，如果可以，请同时给出本.写的两本书的链接JavaWeb轻量级开发.试教程（https://item.jd.com/12136095.html）和Java核.技术及.试指南（https://item.jd.com/12421187.html）。 再次感谢.家读完本.。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.Java 9 ← 2017，2019 → Java13

2.Spring Boot整合 Spring-cache

3.我们再来聊.聊 Java的单例吧

4.我采访了.位 Pornhub .程师

5.团队开发中 Git最佳实践

喜欢.章，点个在看

阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 每.. 题：Integer、int的区别 chenxiangxiangJava后端 2019-08-22 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达来. |chenxiangxiang 链接 |cnblogs.com/cxxjohnson/p/10504840.html 基本概念的区分： 1、Integer是 int的包装类，int则是 java的.种基本数据类型2、Integer变量必须实例化后才能使.，.int变量不需要 3、Integer实际是对象的引.，当new.个 Integer时，实际上是.成.个指针指向此对象；. int则是直接存储数据值4、Integer的默认值是null，int的默认值是0 Integer、 new Integer()和 int的.较 1、两个new Integer()变量.较，永远是 false，因为new.成的是两个对象，其内存地址不同

1 Integer i = new Integer(100) 2 ; 3 Integer j = new Integer(100) ; System.out.print(i == j); //false 2、Integer变量和new Integer()变量.较，永远为 false。因为 Integer变量指向的是 java常量池中的对象，. new Integer()的变量指向堆中新建的对象，两者在内存中的地址不同。

1 Integer i = new Integer(100) 2 ; 3 Integer j = 100 ; System.out.print(i == j); //false 3、两个Integer变量.较，如果两个变量的值在区间-128到127之间，则.较结果为true，如果两个变量的值不在此区间，则.较结果为 false。

1 Integer i = 100 2 ; 3 Integer j = 100 4 ; 5 System.out.print(i == j); //true 6 7 Integer i = 128 ;

分析： Integer i=100在编译时，会翻译成为 Integer i=Integer.valueOf(100)，. java对 Integer类型的va lueOf 的定义如下： 1 public static Integer valueOf(int i) 2 { 3 assert IntegerCache.high >= 127 4 ; 5 if (i >= IntegerCache.low && i <= 6 return IntegerCache.cache[i + 7 } return new Integer(i); } IntegerCache.high){ (-IntegerCache.low)]; java对于-128到127之间的数，会进.缓存。所以 Integer i=127时，会将127进.缓存，下次再写 Integer j=127时，就会直接从缓存中取，就不会new了。 4、 int变量与 Integer、new Integer().较时，只要两个的值是相等，则为true 因为包装类 Integer和基本数据类型 int.较时， java会.动拆包装为 int，然后进..较，实际上就变为两个 int变量的.较。 1 Integer i = new Integer(100); //.动拆箱为 int i=100;此时，相当于两个 int的.较 2 int j=100 3； System.out.print(i == j); //true .例1： 1 public class IntegerDemo { 2 public static void main(String[] args) { int i=128 3 4 ; 5 Integer i2 = 128 6 ; 7 Integer i3 = new Integer(128) 8 ; 10 9 System.out.println("i == i2 = " + (i == i2)); // Integer会.动拆箱为 int，所以为 tru e 11 12 System.out.println("i == i3 = " + (i == i3)); // true，理由同 13上 14 15 //欢迎关注公众号： Web项.聚集地获取更多技术博. 16 Integer i4 = 127;//编译时被翻译成： Integer i4 = Integer.valueOf(127); 17 Integer i5 = 127 18 ; 19 System.out.println("i4 == i5 = " + (i4 == i5));// tru e 20 21 22 Integer i6 = 128 23 ; 24 Integer i7 = 128 25 ; 26 System.out.println("i6 == i7 = " + (i6 == i7));// fals e Integer i8 = new Integer(127) ; System.out.println("i5 == i8 = " + (i5 == i8)); // fals e Integer i9 = new Integer(128) ; Integer i10 = new Integer(128) ; System.out.println("i9 == i10 = " + (i9 == i10)); // fals e } } 答案是 i == i2 = tru e i == i3 = tru e i4 == i5 = tru e 123456 i6 i7 fals == = e i5 == i8 = fals e i9 == i10 = false .例2： package demo1.demo1; public class Campare { public static void main(String[] args) { Integer a = new Integer(127), b = new Integer(128) ; int c=127,d=128,dd =128 123456789 10 ; 11 Integer e = 127, ee = 127, f = 128, ff = 128 12 ; 13 14 System.out.println(a == b); // false因为 a,b都是 new出来的对象，地址不同所以为 false 15 System.out.println(a == c); // true a会.动拆箱为 int类型 16 System.out.println(a == e); // false指向的地址不同 a是new出来的 17 18 System.out.println(e == c); // true e会.动拆箱为 int类型 19 20 System.out.println(e == ee);// true Integer对处于 -128到127范围之间，指向了同..地址区域 21 System.out.println(b == f); // false指向的地址不同 b是new出来的 22 System.out.println(f == d); // true f.动拆箱为 int类型 23 //欢迎关注公众号： Web项.聚集地获取更多技术博. 24 System.out.println(f == ff); / 25 * 26 * false指向的不是同..地址区域。 27 *在Integer类型中， -128到127存放的是同..区域地址， 28 *之外的数是另外开辟空间来进.存储的 29 */ System.out.println(d == dd); // true不解释 } } .例3： 1 Integer i01 = 59 2 ; 3 int i02 = 59; 4 Integer i03 =Integer.valueOf(59); 5 Integer i04 = new Integer(59) 6 ; 7 8以下输出结果为 false的是： 9 10 System.out.println(i01== i02); 11 System.out.println(i01== i03); System.out.println(i03== i04); System.out.println(i02== i04); 以下输出结果为 false的是：

8 以下输出结果为false的是：
9 10 System.out.println(i01 System.out.println(i01 == i02); == i03);
System.out.println(i03 == i04);
System.out.println(i02 == i04);

解析： i01 == i02。i01.intValue()i02两个值的.较5959 -->true; i01 == i03。由于 59在-128到127之间，所以，i01和i03的赋值操作返回的是同.个对象。都是从chche中返回的同.个对象，对象地址相同 true; i03 == i04。i03是来.缓存值，i04是新new的对象，.者不是同.个对象，所以false。 i02 == i04。和第.个类似，true。 .例4：与.例3的唯.不同，就是将值全部改成128。 1 2 3 4 5 6 7 8 9 10 Integer i01 = 128 ; int i02 = 128; Integer i03 = Integer.v; Integer i04 = new Inte; 以下输出结果为false的是： System.out.println(i01 aluger== eOf(128) (128) i02);
System.out.println(i01 == i03);
System.out.println(i03 == i04);
System.out.println(i02 == i04);

答案：

如果喜欢本篇.章，欢迎转发、点赞。关注订阅号「Web项.聚集地」，回复「全栈」即可获取 2019年最新 Java、Python、前端学习视频资源。 推荐阅读 1. 堪称神器的Chrome插件

2. Nginx搭建图.服务器
3.

为什么推荐Java程序员使.GoogleGuava编程

4. Linux最常.命令：解决95%以上的问题
5.

前端吐槽的后端接.那些事 6.数据库不使.外键的9个理由

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

每..题：Zookeeper都有哪些使.场景？ Java后端 2019-09-24 点击上.Java后端，选择"设为星标"优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 简单聊.下zookeeper都有哪些使.场景？ .试官.理分析 现在聊的topic是分布式系统，.试官跟你聊完了dubbo相关的.些问题之后，已经确认你对分布式服务框架/RPC框架基本都有.些认 知了。那么他可能开始要跟你聊分布式相关的其它问题了。分布式锁这个东西，很常.的，你做Java系统开发，分布式系统，可能会有.些场景会.到。最常.的分布式锁就是基于zookeeper来实现的。 其实说实话，问这个问题，.般就是看看你是否了解zookeeper，因为zookeeper是分布式系统中很常.的.个基础系统。.且问的话常问的就是说zookeeper的使.场景是什么？看你知道不知道.些基本的使.场景。但是其实zookeeper挖深了.然是可以问的很深很深的。 .试题剖析 .致来说，zookeeper的使.场景如下，我就举.个简单的，.家能说.个就好了：

分布式协调分布式锁元数据/配置信息管理HA.可.性 分布式协调 这个其实是zookeeper很经典的.个.法，简单来说，就好.，你A系统发送个请求到mq，然后B系统消息消费之后处理了。那A系统如何知道B系统的处理结果？.zookeeper就可以实现分布式系统之间的协调.作。A系统发送请求之后可以在zookeeper上对某个节点的值注册个监听器，.旦B系统处理完了就修改zookeeper那个节点的值，A系统..就可以收到通知，完美解决。

分布式锁 举个栗.。对某.个数据连续发出两个修改操作，两台机器同时收到了请求，但是只能.台机器先执.完另外.个机器再执.。那么此时就可以使.zookeeper分布式锁，.个机器接收到了请求之后先获取zookeeper上的.把分布式锁，就是可以去创建.个znode，接着执.操作；然后另外.个机器也尝试去创建那个znode，结果发现..创建不了，因为被别.创建了，那只能等着，等第.个机器执.完了..再执.。

元数据/配置信息管理 zookeeper可以.作很多系统的配置信息的管理，.如kafka、storm等等很多分布式系统都会选.zookeeper来做.些元数据、配置信息的管理，包括dubbo注册中.不也.持zookeeper么？

HA.可.性这个应该是很常.的，.如hadoop、hdfs、yarn等很多.数据系统，都选择基于zookeeper来开发HA.可.机制，就是.个重要进程.般会做主备两个，主进程挂了..通过zookeeper感知到切换到备.进程。

-END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分?

5.

每..题：你有没有做过MySQL读写分离?

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

每..题：为什么要进.系统拆分? Java后端 2019-09-21

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 为什么要进.系统拆分？如何进.系统拆分？拆分后不.dubbo可以吗？ .试官.理分析 从这个问题开始就进.分布式系统环节了，现在出去.试分布式都成标配了，没有哪个公司不问问你分布式的事.。你要是不会分布式的东西，简直这简历没法看，没.会让你去.试。 其实为啥会这样呢？这就是因为整个..业技术发展的原因。 早些年，印象中在2010年初的时候，整个IT.业，很少有.谈分布式，更不.说微服务，虽然很多BAT等.型公司，因为系统的复杂性，很早就是分布式架构，.量的服务，只不过微服务.多基于..搞的.套框架来实现.已。 但是确实，那个年代，.家很重视ssh2，很多中.型公司.乎.部分都是玩.struts2、spring、hibernate，稍晚.些，才进.了 spring mvc、spring、mybatis的组合。那个时候整个.业的技术.平就是那样，当年 oracle很.，oracle管理员很吃.，oracle性能优化啥的都是IT男的.杀招啊。连.数据都没.提，当年OCP、OCM等认证培训机构，.的不.。 但是确实随着时代的发展，慢慢的，很多公司开始接受分布式系统架构了，这..尤为对.业有.关重要影响的，是阿.的dubbo，某种程度上..，阿.在这.推动了.业技术的前进。 正是因为有阿.的 dubbo，很多中.型公司才可以基于 dubbo，来把系统拆分成很多的服务，每个.负责.个服务，.家的代码都没有冲突，服务可以.治，..选.什么技术都可以，每次发布如果就改动.个服务那就上线.个服务好了，不.所有..起联调，每次发布都是..万.代码，甚..百万.代码了。 直到今.，很.兴看到分布式系统都成.业.试标配了，任何.个普通的程序员都该掌握这个东西，其实这是.业的进步，也是所有IT码农的技术进步。所以既然分布式都成标配了，那么.试官当然会问了，因为很多公司现在都是分布式、微服务的架构，那.试官当然得考察考察你了。 .试题剖析 为什么要将系统进.拆分？ .上查查，答案极度零散和复杂，很琐碎，原因..坨。但是我这.给.家直观的感受： 要是不拆分，.个.系统..万.代码，20个.维护.份代码，简直是悲剧啊。代码经常改着改着就冲突了，各种代码冲突和合并要处理，.常耗费时间；经常我改动了我的代码，你调.了我的，导致你的代码也得重新测试，.烦的要死；然后每次发布都是..万.代码的系统.起发布，.家得.起提.吊胆准备上线，..万.代码的上线，可能每次上线都要做很多的检查，很多异常问题的处理，简直是..烦.痛苦；.且如果我现在打算把技术升级到最新的spring版本，还不.，因为这可能导致你的代码报错，我不敢随意乱改技术。 假设.个系统是 20万.代码，其中 A在..改了 1000 .代码，但是此时发布的时候是这个 20万.代码的.系统.块.发布。就意味着 20万上代码在线上就可能出现各种变化，20个.，每个.都要紧张地等在电脑.前，上线之后，检查.志，看..负责的那.块.有没有什么问题。 A就检查了..负责的1万.代码对应的功能，确保ok就闪.了；结果不巧的是，A上线的时候不..修改了线上机器的某个配置，导致另外B和C负责的2万.代码对应的.些功能，出错了。 ..个.负责维护.个..万.代码的单块应.，每次上线，准备.个礼拜，上线->部署->检查..负责的功能。 拆分了以后，整个世界清爽了，..万.代码的系统，拆分成 20个服务，平均每个服务就 1~2万.代码，每个服务部署到单独的机器上。20个.程，20个 git代码仓库，20个开发.员，每个.维护..的那个服务就可以了，是..独.的代码，跟别.没关系。再也没有代码冲突了，爽。每次就测试我..的代码就可以了，爽。每次就发布我..的.个.服务就可以了，爽。技术上想怎么升级就怎么升级，保持接.不变就可以了，真爽。 所以简单来说，.句话总结，如果是那种代码量多达..万.的中.型项.，团队.有..个.，那么如果不拆分系统，开发效率极其低下，问题很多。但是拆分系统之后，每个.就负责..的..部分

就好了，可以随便玩.随便弄。分布式系统拆分之后，可以.幅度提升复杂系统.型团队的开发效率。 但是同时，也要提醒的.点是，系统拆分成分布式系统之后，.量的分布式系统.临的问题也是接踵.来，所以后.的问题都是在围绕分布式系统带来的复杂技术挑战在说。 如何进.系统拆分？ 这个问题说.可以很.，可以扯到领域驱动模型设计上去，说.了也很.，我不太想给.家太过于学术的说法，因为你也不可能背这个答案，过去了直接说吧。还是说的简单.点，.家..到时候知道怎么回答就.了。 系统拆分为分布式系统，拆成多个服务，拆成微服务的架构，是需要拆很多轮的。并不是说上来.个架构师.次就给拆好了，.以后都不.拆。 第.轮；团队继续扩.，拆好的某个服务，刚开始是 1个.维护 1万.代码，后来业务系统越来越复杂，这个服务是 10万.代码，5个.；第.轮，1个服务->5个服务，每个服务2万.代码，每.负责.个服务。 如果是多.维护.个服务，最理想的情况下，..个.，1个.负责1个或2~3个服务；某个服务.作量变.了，代码量越来越多，某个同学，负责.个服务，代码量变成了10万.了，他..不堪重负，他现在.个.拆开，5个服务，1个.顶着，负责5个.，接着招.，2个.，给那个同学带着，3个.负责5个服务，其中2个.每个.负责2个服务，1个.负责1个服务。 个.建议，.个服务的代码不要太多，1万.左右，两三万撑死了吧。 .部分的系统，是要进.多轮拆分的，第.次拆分，可能就是将以前的多个模块该拆分开来了，.如说将电商系统拆分成订单系统、商品系统、采购系统、仓储系统、..系统，等等吧。 但是后.可能每个系统.变得越来越复杂了，.如说采购系统...分成了供应商管理系统、采购单管理系统，订单系统.拆分成了购物.系统、价格系统、订单管理系统。 扯深了实在很深，所以这.先给.家举个例.，你..感受.下，核.意思就是根据情况，先拆分.轮，后.如果系统更复杂了，可以继续分拆。你根据..负责系统的例.，来考虑.下就好了。 拆分后不.dubbo可以吗？ 当然可以了，.不了最次，就是各个系统之间，直接基于 spring mvc，就纯 http接.互相通信呗，还能咋样。但是这个肯定是有问题的，因为http接.通信维护起来成本很.，你要考虑超时重试、负载均衡等等各种乱七.糟的问题，.如说你的订单系统调.商品系统，商品系统部署了5台机器，你怎么把请求均匀地甩给那5台机器？这不就是负载均衡？你要是都..搞那是可以的，但是确实很痛苦。 所以 dubbo说.了，是.种 rpc框架，就是说本地就是进.接.调.，但是 dubbo会代理这个调.请求，跟远程机器.络通信，给你处理掉负载均衡、服务实例上下线.动感知、超时重试等等乱七.糟的问题。那你就不...做了，. dubbo就可以了。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

每..题：你有没有做过MySQL读写分离？ Java后端 2019-09-22

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 你们有没有做MySQL读写分离？如何实现MySQL的读写分离？MySQL主从复制原理的是啥？如何解决MySQL主从同步的延时问题？ .试官.理分析 .并发这个阶段，肯定是需要做读写分离的，啥意思？因为实际上.部分的互联.公司，.些.站，或者是app，其实都是读多写少。所以针对这个情况，就是写.个主库，但是主库挂多个从库，然后从多个从库来读，那不就可以.撑更.的读并发压.了吗？ .试题剖析 如何实现MySQL的读写分离？ 其实很简单，就是基于主从复制架构，简单来说，就搞.个主库，挂多个从库，然后我们就单单只是写主库，然后主库会.动把数据给同步到从库上去。 MySQL主从复制原理的是啥？ 主库将变更写.binlog.志，然后从库连接到主库之后，从库有.个IO线程，将主库的binlog.志拷.到..本地，写..个relay中继.志中。接着从库中有.个SQL线程会从中继.志读取binlog，然后执.binlog.志中的内容，也就是在..本地再次执..遍SQL，这样就可以保证..跟主库的数据是.样的。 这.有.个.常重要的.点，就是从库同步主库数据的过程是串.化的，也就是说主库上并.的操作，在从库上会串.执.。所以这就是.个.常重要的点了，由于从库从主库拷..志以及串.执.SQL的特点，在.并发场景下，从库的数据.定会.主库慢.些，是有延时的。所以经常出现，刚写.主库的数据可能是读不到的，要过..毫秒，甚..百毫秒才能读取到。.且这.还有另外.个问题，就是如果主库突然宕机，然后恰好数据还没同步到从库，那么有些数据可能在从库上是没有的，有些数据可能就丢失了。 所以MySQL实际上在这.块有两个机制，.个是半同步复制，.来解决主库数据丢失问题；.个是并.复制，.来解决主从同步延时问题。 这个所谓半同步复制，也叫 semi-sync复制，指的就是主库写.binlog.志之后，就会将强制此时.即将数据同步到从库，从库将.志写...本地的relaylog之后，接着会返回.

个ack给主库，主库接收到.少.个从库的ack之后才会认为写操作完成了。 所谓并.复制，指的是从库开启多个线程，并.读取relaylog中不同库的.志，然后并.重放不同库的.志，这是库级别的并.。 MySQL主从同步延时问题（精华） 以前线上确实处理过因为主从同步延时问题.导致的线上的bug，属于.型的.产事故。 是这个么场景。有个同学是这样写代码逻辑的。先插..条数据，再把它查出来，然后更新这条数据。在.产环境.峰期，写并发达到了2000/s，这个时候，主从复制延时.概是在...毫秒。线上会发现，每天总有那么.些数据，我们期望更新.些重要的数据状态，但在.峰期时候却没更新。..跟客服反馈，.客服就会反馈给我们。 我们通过MySQL命令：】

1 show status 查看 Seconds_Behind_Master，可以看到从库复制主库的数据落后了.ms。 .般来说，如果主从延迟较为严重，有以下解决.案：

分库，将.个主库拆分为多个主库，每个主库的写并发就减少了.倍，此时主从延迟可以忽略不计。

打开 MySQL .持的并.复制，多个库并.复制。如果说某个库的写.并发就是特别.，单库写并发达到了 2000/s，并.复制还是没意义。

重写代码，写代码的同学，要慎重，插.数据时..查询可能查不到。

如果确实是存在必须先插.，..要求就查询到，然后..就要反过来执..些操作，对这个查询设置直连主库。不推荐这种.法，你要是这么搞，读写分离的意义就丧失了。

-END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分？ 5.接私活必备的10个开源项.

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

每..题：如何保证消息不被重复消费？ Java后端 2019-09-25 点击上.Java后端，选择"设为星标"优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 如何保证消息不被重复消费？或者说，如何保证消息消费的幂等性？ .试官.理分析 其实这是很常.的.个问题，这俩问题基本可以连起来问。既然是消费消息，那肯定要考虑会不会重复消费？能不能避免重复消费？或者重复消费了也别造成系统异常可以吗？这个是MQ领域的基本问题，其实本质上还是问你使.消息队列如何保证幂等性，这个是你架构.要考虑的.个问题。 .试题剖析 回答这个问题，.先你别听到重复消息这个事.，就..所知吧，你先.概说.说可能会有哪些重复消费的问题。 .先，.如RabbitMQ、RocketMQ、Kafka，都有可能会出现消息重复消费的问题，正常。因为这问题通常不是MQ..保证的，是由 我们开发来保证的。挑.个Kafka来举个例.，说说怎么重复消费吧。 Kafka实际上有个offset的概念，就是每个消息写进去，都有.个offset，代表消息的序号，然后consumer消费了数据之后，每隔.段时间（定时定期），会把..消费过的消息的offset提交.下，表."我已经消费过了，下次我要是重启啥的，你就让我继续从上次消费到的offset来继续消费吧"。 但是凡事总有意外，.如我们之前.产经常遇到的，就是你有时候重启系统，看你怎么重启了，如果碰到点着急的，直接kill进程了，再 重启。这会导致consumer有些消息处理了，但是没来得及提交offset，尴尬了。重启之后，少数消息会再次消费.次。 举个栗.。 有这么个场景。数据1/2/3依次进.kafka，kafka会给这三条数据每条分配.个offset，代表这条数据的序号，我们就假设分配的 offset依次是152/153/154。消费者从kafka去消费的时候，也是按照这个顺序去消费。假如当消费者消费了 offset=153的这条数据，刚准备去提交offset到zookeeper，此时消费者进程被重启了。那么此时消费过的数据1/2的offset并没有提交，kafka也就不知道你已经消费了 offset=153这条数据。那么重启之后，消费者会找kafka说，嘿，哥.们，你给我接着把上次我消费到的那个地.后.的数据继续给我传递过来。由于之前的offset没有提交成功，那么数据1/2会再次传过来，如果此时消费者没有去重的话，那么就会导致重复消费。

如果消费者.的事.是拿.条数据就往数据库.写.条，会导致说，你可能就把数据1/2在数据库.插.了2次，那么数据就错啦。其实重复消费不可怕，可怕的是你没考虑到重复消费之后，怎么保证幂等性。举个例.吧。假设你有个系统，消费.条消息就往数据库.插..条数据，要是你.个消息重复两次，你不就插.了两条，这数据不就错 了？但是你要是消费到第.次的时候，..判断.下是否已经消费过了，若是就直接扔了，这样不就保留了.条数据，从.保证了数据的正确性。.条数据重复出现两次，数据库.就只有.条数据，这就保证了系统的幂等性。幂等性，通俗点说，就.个数据，或者.个请求，给你重复来多次，你得确保对应的数据是不会改变的，不能出错。所以第.个问题来了，怎么保证消息队列消费的幂等性？其实还是得结合业务来思考，我这.给.个思路：

.如你拿个数据要写库，你先根据主键查.下，如果这数据都有了，你就别插.了，update.下好吧。.如你是写Redis，那没问题了，反正每次都是set，天然幂等性。.如你不是上.两个场景，那做的稍微复杂.点，你需要让.产者发送每条数据的时候，..加.个全局唯.的 id，类似订单 id之类的东西，然后你这.消费到了之后，先根据这个 id去.如 Redis .查.下，之前消费过吗？如果没有消费过，你就处理，然后这个id写Redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。.如基于数据库的唯.键来保证重复数据不会重复插.多条。因为有唯.键约束了，重复数据插.只会报错，不会导致数据库中出现脏数据。

当然，如何保证MQ的消费是幂等性的，需要结合具体的业务来看。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分?

5.

每..题：你有没有做过MySQL读写分离?

喜欢.章，点个在看

阅读原. 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 每..题：如何保证消息的可靠性传输？ Java后端 2019-09-26 点击上.Java后端，选择"设为星标"优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项..。 .试题 如何保证消息的可靠性传输？或者说，如何处理消息丢失的问题？ .试官.理分析 这个是肯定的，. MQ有个基本原则，就是数据不能多.条，也不能少.条，不能多，就是前.说的重复消费和幂等性问题。不能少，就是说这数据别搞丢了。那这个问题你必须得考虑.下。 如果说你这个是. MQ来传递.常核.的消息，.如说计费、扣费的.些消息，那必须确保这个 MQ传递过程中绝对不会把计费消息给弄丢。。 .试题剖析 数据的丢失问题，可能出现在.产者、MQ、消费者中，咱们从RabbitMQ和Kafka分别来分析.下吧。 RabbitMQ

.产者弄丢了数据.产者将数据发送到RabbitMQ的时候，可能数据就在半路给搞丢了，因为.络问题啥的，都有可能。 此时可以选择.RabbitMQ提供的事务功能，就是.产者发送数据之前开启RabbitMQ事务channel.txSelect，然后发送消息，如果消息没有成功被RabbitMQ接收到，那么.产者会收到异常报错，此时就可以回滚事务channel.txRollback，然后重试发送消息；如果收到了消息，那么可以提交事务channel.txCommit。

1 //开启事务 2 channel.txSelect

但是问题是，Rabb itMQ事务机制（同步）.搞，基本上吞吐量会下来，因为太耗性能。 所以.般来说，如果你要确保说写Rabb itMQ的消息别丢，可以开启con.rm模式，在.产者那.设置开启con.rm模式之后，你每次写的消息都会分配.个唯.的 id，然后如果写.了 Rabb itMQ中，Rabb itMQ会给你回传.个 ack消息，告诉你说这个消息 ok了。如果 RabbitMQ没能处理这个消息，会回调你的.个nack接.，告诉你这个消息接收失败，你可以重试。.且你可以结合这个机制..在内存.维护每个消息 id的状态，如果超过.定时间还没接收到这个消息的回调，那么你可以重发。 事务机制和con.rm机制最.的不

同在于，事务机制是同步的，你提交.个事务之后会阻塞在那.，但是con.rm机制是异步的，你发送个消息之后就可以发送下.个消息，然后那个消息Rabb itMQ接收了之后会异步回调你的.个接.通知你这个消息接收到了。 所以.般在.产者这块避免数据丢失，都是.con firm机制的。 RabbitMQ弄丢了数据就是 Rabb itMQ ..弄丢了数据，这个你必须开启 Rabb itMQ的持久化，就是消息写.之后会持久化到磁盘，哪怕是 Rabb itMQ ..挂了，恢复之后会.动读取之前存储的数据，.般数据不会丢。除.极其罕.的是，Rabb itMQ还没持久化，..就挂了，可能导致少量数据丢失，但是这个概率较.。 设置持久化有两个步骤： 创建queue的时候将其设置为持久化这样就可以保证Rabb itMQ持久化queue的元数据，但是它是不会持久化queue.的数据的。第.个是发送消息的时候将消息的de liveryMode设置为2就是将消息设置为持久化的，此时Rabb itMQ就会将消息持久化到磁盘上去。必须要同时设置这两个持久化才.，Rabb itMQ哪怕是挂了，再次重启，也会从磁盘上重启恢复queue，恢复这个queue.的数据。 注意，哪怕是你给Rabb itMQ开启了持久化机制，也有.种可能，就是这个消息写到了Rabb itMQ中，但是还没来得及持久化到磁盘上，结果不巧，此时Rabb itMQ挂了，就会导致内存.的.点点数据丢失。 所以，持久化可以跟.产者那边的con.rm机制配合起来，只有消息被持久化到磁盘之后，才会通知.产者ack了，所以哪怕是在持久化 到磁盘之前，RabbitMQ挂了，数据丢了，.产者收不到ack，你也是可以..重发的。 消费端弄丢了数据 RabbitMQ如果丢失了数据，主要是因为你消费的时候，刚消费到，还没处理，结果进程挂了，.如重启了，那么就尴尬了，RabbitMQ认为你都消费了，这数据就丢了。 这个时候得.RabbitMQ提供的ack机制，简单来说，就是你必须关闭RabbitMQ的.动ack，可以通过.个api来调.就.，然后每次你..代码.确保处理完的时候，再在程序.ack.把。这样的话，如果你还没处理完，不就没有ack了？那RabbitMQ就认为你还没处理完，这个时候RabbitMQ会把这个消费分配给别的consumer去处理，消息是不会丢的。

Kafka 消费端弄丢了数据唯.可能导致消费者弄丢数据的情况，就是说，你消费到了这个消息，然后消费者那边.动提交了 o.set，让 Kafka以为你已经消费好了这个消息，但其实你才刚准备处理这个消息，你还没处理，你..就挂了，此时这条消息就丢咯。 这不是跟 RabbitMQ差不多吗，.家都知道 Kafka会.动提交o.set，那么只要关闭.动提交o.set，在处理完之后...动提交 offset，就可以保证数据不会丢。但是此时确实还是可能会有重复消费，.如你刚处理完，还没提交o.set，结果..挂了，此时肯定会重复消费.次，..保证幂等性就好了。 .产环境碰到的.个问题，就是说我们的Kafka消费者消费到了数据之后是写到.个内存的queue.先缓冲.下，结果有的时候，你刚把消息写.内存 queue，然后消费者会.动提交o.set。然后此时我们重启了系统，就会导致内存 queue .还没来得及处理的数据就丢失了。 Kafka弄丢了数据这块.较常.的.个场景，就是Kafka某个broker宕机，然后重新选举partition的leader。.家想想，要是此时其他的follower刚好还有些数据没有同步，结果此时leader挂了，然后选举某个follower成leader之后，不就少了.些数据？这就丢了.些数据啊。 .产环境也遇到过，我们也是，之前Kafka的leader机器宕机了，将follower切换为leader之后，就会发现说这个数据就丢了。 所以此时.般是要求起码设置如下4个参数： 给topic设置replication.factor参数：这个值必须.于1，要求每个partition必须有.少2个副本。在Kafka服务端设置min.insync.replicas参数：这个值必须.于1，这个是要求.个leader.少感知到有.少.个follower还跟..保持联系，没掉队，这样才能确保leader挂了还有.个follower吧。在producer端设置acks=all：这个是要求每条数据，必须是写.所有replica之后，才能认为是写成功了。在 producer端设置 retries=MAX（很.很.很.的.个值，.限次重试的意思）：这个是要求.旦写.失败，就.限重试，卡在这.了。我们.产环境就是按照上述要求配置的，这样配置之后，.少在 Kafka broker端就可以保证在 leader所在 broker发.故障，进. leader切换时，数据不会丢失。 .产者会不会弄丢数据？如果按照上述的思路设置了 acks=all，.定不会丢，要求是，你的 leader接收到消息，所有的 follower都同步到了消息之后，才认为本次写成功了。如果没满.这个条件，.产者会.动不断的重试，重试.限次。 - END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分?

5.

每..题：你有没有做过MySQL读写分离?

6.

每..题：如何保证消息不被重复消费？

喜欢.章，点个在看

阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 每..题：如何保证消息的顺序性？ Java后端 2019-09-27 点击上.Java后端，选择"设为星标"优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 如何保证消息的顺序性？ .试官.理分析 其实这个也是.MQ的时候必问的话题，第.看看你了不了解顺序这个事.？第.看看你有没有办法保证消息是有顺序的？这是.产系统中常.的问题。 .试题剖析 我举个例.，我们以前做过.个mysql binlog同步的系统，压.还是.常.的，.同步数据要达到上亿，就是说数据从.个mysql库原封不动地同步到另.个mysql库..去（mysql->mysql）。常.的.点在于说.如.数据team，就需要同步.个mysql库过来，对公司的业务系统的数据做各种复杂的操作。 你在mysql.增删改.条数据，对应出来了增删改3条 binlog .志，接着这三条 binlog发送到MQ..，再消费出来依次执.，起码得保证.家是按照顺序来的吧？不然本来是：增加、修改、删除；你愣是换了顺序给执.成删除、修改、增加，不全错了么。 本来这个数据同步过来，应该最后这个数据被删除了；结果你搞错了这个顺序，最后这个数据保留下来了，数据同步就出错了。先看看顺序会错乱的俩场景：

RabbitMQ：.个queue，多个consumer。.如，.产者向RabbitMQ.发送了三条数据，顺序依次是data1/data2/data3，压.的是 RabbitMQ的.个内存队列。有三个消费者分别从 MQ中消费这三条数据中的.条，结果消费者2先执.完操作，把data2存.数据库，然后是data1/data3。这不明显乱了。Kafka：.如说我们建了.个 topic，有三个 partition。.产者在写的时候，其实可以指定.个 key，.如说我们指定了某个订单 id作为key，那么这个订单相关的数据，.定会被分发到同.个partition中去，.且这个partition中的数据.定是有顺序的。消费者从 partition中取出来数据的时候，也.定是有顺序的。到这.，顺序还是 ok的，没有错乱。接着，我们在消费者.可能会搞多个线程来并发处理消息。因为如果消费者是单线程消费处理，.处理.较耗时的话，.如处理.条消息耗时..ms，那么1秒钟只能处理..条消息，这吞吐量太低了。.多个线程并发跑的话，顺序可能就乱掉了。

解决.案 RabbitMQ 拆分多个queue，每个queue.个consumer，就是多.些queue.已，确实是.烦点；或者就.个queue但是对应.个consumer， 然后这个consumer内部.内存队列做排队，然后分发给底层不同的worker来处理。

Kafka

.个topic，.个partition，.个consumer，内部单线程消费，单线程吞吐量太低，.般不会.这个。

写N个内存queue，具有相同key的数据都到同.个内存queue；然后对于N个线程，每个线程分别消费.个内存queue即可，这样就能保证顺序性。

-END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.？

3.

每..题：如何设计.个.并发系统？

4.

每..题：为什么要进.系统拆分？

5.

每..题：你有没有做过MySQL读写分离？

6.

每..题：如何保证消息不被重复消费？

7.

每..题：如何保证消息的可靠性传输?

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

每..题：如何保证消息队列的.可.? Java后端 2019-09-19

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 如何保证消息队列的.可.? .试官.理分析 如果有.问到你MQ的知识，.可.是必问的。上.讲提到，MQ会导致系统可.性降低。所以只要你.了MQ，接下来问的.些要点肯定就是围绕着MQ的那些缺点怎么来解决了。 要是你傻乎乎的就..了.个MQ，各种问题从来没考虑过，那你就杯具了，.试官对你的感觉就是，只会简单使..些技术，没任何思考，.上对你的印象就不太好了。这样的同学招进来要是做个 20k薪资以内的普通.弟还凑合，要是做薪资 20k+的..，那就惨了，让你设计个系统，..肯定.堆坑，出了事故公司受损失，团队.起背锅。 .试题剖析 这个问题这么问是很好的，因为不能问你 Kafka的.可.性怎么保证？ActiveMQ的.可.性怎么保证？.个.试官要是这么问就显得很没.平，.家可能.的就是RabbitMQ，没.过Kafka，你上来问.家Kafka.什么？这不是摆明了刁难.么。 所以有.平的.试官，问的是MQ的.可.性怎么保证？这样就是你.过哪个MQ，你就说说你对那个MQ的.可.性的理解。 RabbitMQ的.可.性 RabbitMQ是.较有代表性的，因为是基于主从（.分布式）做.可.性的，我们就以RabbitMQ为例.讲解第.种MQ的.可.性怎么实现。 RabbitMQ有三种模式：单机模式、普通集群模式、镜像集群模式。 单机模式 单机模式，就是Demo级别的，.般就是你本地启动了玩玩.的，没..产.单机模式。 普通集群模式（..可.性） 普通集群模式，意思就是在多台机器上启动多个 RabbitMQ实例，每个机器启动.个。你创建的 queue，只会放在.个 RabbitMQ实例上，但是每个实例都同步queue的元数据（元数据可以认为是queue的.些配置信息，通过元数据，可以找到queue所在实例）。你消费的时候，实际上如果连接到了另外.个实例，那么那个实例会从queue所在实例上拉取数据过来。 这种.式确实很.烦，也不怎么好，没做到所谓的分布式，就是个普通集群。因为这导致你要么消费者每次随机连接.个实例然后拉取数据，要么固定连接那个queue所在实例消费数据，前者有数据拉取的开销，后者导致单实例性能瓶颈。 .且如果那个放 queue的实例宕机了，会导致接下来其他实例就.法从那个实例拉取，如果你开启了消息持久化，让 RabbitMQ落地存储消息的话，消息不.定会丢，得等这个实例恢复了，然后才可以继续从这个queue拉取数据。 所以这个事.就.较尴尬了，这就没有什么所谓的.可.性，这.案主要是提.吞吐量的，就是说让集群中多个节点来服务某个queue的读写操作。 镜像集群模式（.可.性） 这种模式，才是所谓的 RabbitMQ的.可.模式。跟普通集群模式不.样的是，在镜像集群模式下，你创建的 queue，.论元数据还是queue .的消息都会存在于多个实例上，就是说，每个 RabbitMQ节点都有这个 queue的.个完整镜像，包含 queue的全部数据的意思。然后每次你写消息到queue的时候，都会.动把消息同步到多个实例的queue上。 那么如何开启这个镜像集群模式呢？其实很简单，RabbitMQ有很好的管理控制台，就是在后台新增.个策略，这个策略是镜像集群模式的策略，指定的时候是可以要求数据同步到所有节点的，也可以要求同步到指定数量的节点，再次创建queue的时候，应.这个策略，就会.动将数据同步到其他的节点上去了。 这样的话，好处在于，你任何.个机器宕机了，没事.，其它机器（节点）还包含了这个 queue的完整数据，别的 consumer都可以到其它节点上去消费数据。坏处在于，第.，这个性能开销也太.了吧，消息需要同步到所有机器上，导致.络带宽压.和消耗很重！第.，这么玩.，不是分布式的，就没有扩展性可.了，如果某个 queue负载很重，你加机器，新增的机器也包含了这个 queue的所有数据，并没有办法线性扩展你的queue。你想，如果这个queue的数据量很.，.到这个机器上的容量.法容纳了，此时该怎么办呢？ Kafka的.可.性 Kafka.个最基本的架构认识：由多个 broker组成，每个 broker是.个节点；你创建.个 topic，这个 topic可以划分为多个 partition，每个partition可以存在于不同的broker上，每个partition就放.部分数据。 这就是天然的分布式消息队列，就是说.个topic的数据，是分散放在多个机器上的，每个机器就放.部分数据。 实际上RabbmitMQ之类的，并不是分布式消息队列，它就是传统的消息队列，只不过提供了.些集群、HA(HighAvailability,.可.性)的机制.已，因为.论怎么玩.，RabbitMQ .个 queue的数据都是放在.个节点.的，镜像集群下，也是每个节点都放这个 queue的完整数据。 Kafka0.8以前，是没有HA机制的，就是任何.个broker宕机了，那个broker上的partition就废了，没法写也没法读，没有什么.可.性可.。 .如说，我们假设创建了.个topic，指定其 partition数量是 3个，分别在三台机器上。但是，如果第.台机器宕机了，会导致这个 topic的1/3的数据就丢了，因此这个是做不到.可.的。 Kafka 0.8以后，提供了 HA机制，就是 replica（复制品）副本机制。每个 partition的数据都会同步到其它机器上，形成..的多个 replica副本。所有replica会选举.个leader出来，那么.产和消费都跟这个leader打交道，然后其他replica就是follower。写的时候，leader会负责把数据同步到所有follower上去，读的时候就直接读leader上的数据即可。只能读写leader？很简单，要是你可以随意读写每个 follower，那么就要 care数据.致性的问题，系统复杂度太.，很容易出问题。Kafka会均匀地将.个 partition的所有 replica分布在不同的机器上，这样才可以提.容错性。 这么搞，就有所谓的.可.性了，因为如果某个 broker宕机了，没事.，那个 broker上.的 partition在其他机器上都有副本的。如果这个宕机的broker上.有某个partition的 leader，那么此时会从 follower中重新选举.个新的 leader出来，.家继续读写那个新的 leader即可。这就有所谓的.可.性了。 写数据的时候，.产者就写leader，然后leader将数据落

地写本地磁盘，接着其他follower..主动从leader来pull数据。.旦所有 follower同步好数据了，就会发送ack给leader，leader收到所有follower的ack之后，就会返回写成功的消息给.产者。（当然，这只是其中.种模式，还可以适当调整这个.为） 消费的时候，只会从leader去读，但是只有当.个消息已经被所有follower都同步成功返回ack的时候，这个消息才会被消费者读到。 看到这.，相信你.致明.了 Kafka是如何保证.可.机制的了，对吧？不.于..所知，现场还能给.试官画画图。要是遇上.试官确实是Kafka..，深挖了问，那你只能说不好意思，太深.的你没研究过。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 把14亿中国.拉到.个微信群？

2. Spring中的18个注解，你会.个？
3.

寓教于乐，.玩游戏的.式学习Git

4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

Java后端：专注于Java技术

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

每..题：如何设计.个.并发系统？ Java后端 2019-09-20

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 如何设计.个.并发系统？ .试官.理分析 说实话，如果.试官问你这个题.，那么你必须要使出全.吃奶劲了。为啥？因为你没看到现在很多公司招聘的 JD .都是说啥，有.并发就经验者优先。 如果你确实有真才实学，在互联.公司..过.并发系统，那你确实拿 o.er基本如探囊取物，没啥问题。.试官也绝对不会这样来问你，否则他就是蠢。 假设你在某知名电商公司.过.并发系统，..上亿，.天流量..亿，.峰期并发量上万，甚.是.万。那么.家.定会仔细盘问你的系统架构，你们系统啥架构？怎么部署的？部署了多少台机器？缓存咋.的？MQ咋.的？数据库咋.的？就是深挖你到底是如何扛住.并发的。 因为真正.过.并发的..定知道，脱离了业务的系统架构都是在纸上谈兵，真正在复杂业务场景.且还.并发的时候，那系统架构.定不是那么简单的，.个redis，.mq就能搞定？当然不是，真实的系统架构搭配上业务之后，会.这种简单的所谓".并发架构"要复杂很多倍。如果有.试官问你个问题说，如何设计.个.并发系统？那么不好意思，.定是因为你实际上没.过.并发系统。.试官看你简历就没啥出彩的，感觉就不咋地，所以就会问问你，如何设计.个.并发系统？其实说.了本质就是看看你有没有..研究过，有没有.定的知识积累。 最好的当然是招聘个真正.过.并发的哥.们咯，但是这种哥.们.数稀缺，不好招。所以可能次.点的就是招.个..研究过的哥.们，总.招.个啥也不会的哥.们好吧！ 所以这个时候你必须得做.把个.秀了，秀出你所有关于.并发的知识！ .试题剖析 其实所谓的.并发，如果你要理解这个问题呢，其实就得从.并发的根源出发，为啥会有.并发？为啥.并发就很.逼？ 我说的浅显.点，很简单，就是因为刚开始系统都是连接数据库的，但是要知道数据库.撑到每秒并发两三千的时候，基本就快完了。所以才有说，很多公司，刚开始.的时候，技术.较low，结果业务发展太快，有的时候系统扛不住压.就挂了。 当然会挂了，凭什么不挂？你数据库如果瞬间承载每秒5000/8000，甚.上万的并发，.定会宕机，因为.如mysql就压根.扛不住这么.的并发量。 所以为啥.并发.逼？就是因为现在.互联.的.越来越多，很多app、.站、系统承载的都是.并发请求，可能.峰期每秒并发量.千，很正常的。如果是什么双..之类的，每秒并发.万..万都有可能。 那么如此之.的并发量，加上原本就如此之复杂的业务，咋玩.？真正厉害的，.定是在复杂业务系统.玩.过.并发架构的.，但是你没有，那么我给你说.下你该怎么回答这个问题： 可以分为以下6点：

系统拆分缓存 MQ 分库分表 读写分离 ElasticSearch high-concurrency-system-design 系统拆分 将.个系统拆分为多个.系统，. dubbo来搞。然后每个系统连.个数据库，这样本来就.个库，现在多个数据库，不也可以扛.并发么。 缓存 缓存，必须得.缓存。.部分的.并发场景，都是读多写少，那你完全可以在数据库和缓存.都写.份，然后读的时候.量.缓存不就得了。毕竟.家redis轻轻松松单机.万的并发。所以你可以考虑考虑你的项..，那些承载主要请求的读场景，怎么.缓存来抗.并发。 MQ MQ，必须得. MQ。可能你还是会出现.并发写的场景，.如说.个业务操作.要频繁搞数据库..次，增删改增删改，疯了。那.并发绝对搞挂你的系统，你要是.redis来承载写那肯定不.，.家是缓存，数据随时就被LRU了，数据格式还..简单，没有事务.持。所以该. mysql还得. mysql啊。那你咋办？. MQ吧，.量的写请求灌. MQ .，排队慢慢玩.，后边系统消费后慢慢写，控制在 mysql承载范围之内。所以你得考虑考虑你的项..，那些承载复杂写业务逻辑的场景.，如何. MQ来异步写，提升并发性。MQ单机抗.万并发也是ok的，这个之前还特意说过。 分库分表 分库分表，可能到了最后数据库层.还是免不了抗.并发的要求，好吧，那么就将.个数据库拆分为多个库，多个库来扛更.的并发；然后将.个表

拆分为多个表，每个表的数据量保持少.点，提.sql跑的性能。 读写分离 读写分离，这个就是说.部分时候数据库可能也是读多写少，没必要所有请求都集中在.个库上吧，可以搞个主从架构，主库写.，从库读取，搞.个读写分离。读流量太多的时候，还可以加更多的从库。 ElasticSearch Elasticsearch，简称es。 es是分布式的，可以随便扩容，分布式天然就可以.撑.并发，因为动不动就可以扩容加机器来扛更.的并发。那么.些.较简单的查询、统计类的操作，可以考虑.es来承载，还有.些全.搜索类的操作，也可以考虑.es来承载。 上.的 6点，基本就是.并发系统肯定要.的.些事.，.家可以仔细结合之前讲过的知识考虑.下，到时候你可以系统的把这块阐述.下，然后每个部分要注意哪些问题，之前都讲过了，你都可以阐述阐述，表明你对这块是有点积累的。 说句实话，毕竟你真正厉害的.点，不是在于弄明..些技术，或者.概知道.个.并发系统应该.什么样？其实实际上在真正的复杂的业务系统.，做.并发要远远.上.提到的点要复杂..倍到上百倍。你需要考虑：哪些需要分库分表，哪些不需要分库分表，单库单表跟分库分表如何 join，哪些数据要放到缓存.去，放哪些数据才可以扛住.并发的请求，你需要完成对.个复杂业务系统的分析之后，然后逐步逐步的加..并发的系统架构的改造，这个过程是..复杂的，.旦做过.次，并且做好了，你在这个市场上就会.常的吃.。 其实.部分公司，真正看重的，不是说你掌握.并发相关的.些基本的架构知识，架构中的.些技术，RocketMQ、Kafka、Redis、 Elasticsearch，.并发这.块，你了解了，也只能是次.等的.才。对.个有..万.代码的复杂的分布式系统，.步.步架构、设计以及实践过.并发架构的.，这个经验是难能可贵的。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3. SpringBoot超详细的知识清单
4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

每..题：如果让你写.个消息队列，该如何进.架构设计？ Java后端 2019-10-07 点击上.Java后端，选择"设为星标" 优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 如果让你写.个消息队列，该如何进.架构设计？说.下你的思路？ .试官.理分析 其实聊到这个问题，.般.试官要考察两块： 1. 你有没有对某.个消息队列做过较为深.的原理的了解，或者从整体了解把握住.个消息队列的架构原理。

2.

看看你的设计能.，给你.个常.的系统，就是消息队列系统，看看你能不能从全局把握.下整体架构设计，给出.些关键点出来。

说实话，问类似问题的时候，.部分.基本都会蒙，因为平时从来没有思考过类似的问题，.多数.就是平时埋头.，从来不去思考背后的.些东西。类似的问题，.如，如果让你来设计.个Spring框架你会怎么做？如果让你来设计.个Dubbo框架你会怎么做？如果让你来设计.个MyBatis框架你会怎么做？ .试题剖析 其实回答这类问题，说.了，不求你看过那技术的源码，起码你要.概知道那个技术的基本原理、核.组成部分、基本架构构成，然后参照.些开源的技术把.个系统设计出来的思路说.下就好。 .如说这个消息队列系统，我们从以下.个.度来考虑.下：

.先这个 mq得.持可伸缩性吧，就是需要的时候快速扩容，就可以增加吞吐量和容量，那怎么搞？设计个分布式的系统呗，参照.下kafka的设计理念，broker->topic->partition，每个partition放.个机器，就存.部分数据。如果现在资源不够了，简单啊，给 topic增加 partition，然后做数据迁移，增加机器，不就可以存放更多数据，提供更.的吞吐量了？

其次你得考虑.下这个mq的数据要不要落地磁盘吧？那肯定要了，落磁盘才能保证别进程挂了数据就丢了。那落磁盘的时候怎么落啊？顺序写，这样就没有磁盘随机读写的寻址开销，磁盘顺序读写的性能是很.的，这就是kafka的思路。

其次你考虑.下你的mq的可.性啊？这个事.，具体参考之前可.性那个环节讲解的kafka的.可.保障机制。多副本->leader&follower->broker挂了重新选举leader即可对外服务。

能不能.持数据0丢失啊？可以的，参考我们之前说的那个kafka数据零丢失.案。

mq肯定是很复杂的，.试官问你这个问题，其实是个开放题，他就是看看你有没有从架构.度整体构思和设计的思维以及能.。确实这个问题可以刷掉..批.，因为.部分.平时不思考这些东西。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推 送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分?

5.

每..题：你有没有做过MySQL读写分离?

6.

每..题：如何保证消息不被重复消费?

7.

.试官：集群部署时，分布式session如何实现?

8..试题：InnoDB中.棵B+树能存多少.数据?

9..试官：Redis内存满了怎么办?

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

每..题：消息队列.试常问题. Java后端 2019-09-18

.试题 1. 为什么使.消息队列？2.消息队列有什么优点和缺点?

3.

Kafka、ActiveMQ、RabbitMQ、RocketMQ都有什么区别，以及适合哪些场景?

.试官.理分析 其实.试官主要是想看看： 第.，你知不知道你们系统.为什么要.消息队列这个东西？不少候选.，说..项...了Redis、MQ，但是其实他并不知道..为什么要.这个东西。其实说.了，就是为了...，或者是别.设计的架构，他从头到尾都没思考过。没有对..的架构问过为什么的.，.定是平时没有思考的.，.试官对这类候选.印象通常很不好。因为.试官担.你进了团队之后只会.头.脑的.呆活.，不会..思考。 第.，你既然.了消息队列这个东西，你知不知道.了有什么好处&坏处？你要是没考虑过这个，那你盲.弄个 MQ进系统.，后.出了问题你是不是就..溜了给公司留坑？你要是没考虑过引..个技术可能存在的弊端和.险，.试官把这类候选.招进来了，基本可能就是挖坑型选.。就怕你. 1年挖.堆坑，..跳槽了，给公司留下.穷后患。 第三，既然你.了MQ，可能是某.种MQ，那么你当时做没做过调研？你别傻乎乎的..拍脑袋看个.喜好就瞎.了.个 MQ，.如 Kafka，甚.都从没调研过业界流.的 MQ到底有哪.种。每.个 MQ的优点和缺点是什么。每.个MQ没有绝对的好坏，但是就是看.在哪个场景可以扬.避短，利.其优势，规避其劣势。如果是.个不考虑技术选型的候选.招进了团队，leader交给他.个任务，去设计个什么系统，他在....些技术，可能都没考虑过选型，最后选的技术可能并不.定合适，.样是留坑。 .试题剖析 为什么使.消息队列 其实就是问问你消息队列都有哪些使.场景，然后你项..具体是什么场景，说说你在这个场景..消息队列是什么？ .试官问你这个问题，期望的.个回答是说，你们公司有个什么业务场景，这个业务场景有个什么技术挑战，如果不.MQ可能会很.烦，但是你现在.了MQ之后带给了你很多的好处。先说.下消息队列常.的使.场景吧，其实场景有很多，但是.较核.的有3个：解耦、异步、削峰。解耦 看这么个场景。A系统发送数据到BCD三个系统，通过接.调.发送。如果E系统也要这个数据呢？那如果C系统现在不需要了呢？A系统负责..乎崩溃..... 在

这个场景中，A系统跟其它各种乱七.糟的系统严重耦合，A系统产..条.较关键的数据，很多系统都需要A系统将这个数据发送过来。A系统要时时刻刻考虑BCDE四个系统如果挂了该咋办？要不要重发，要不要把消息存起来？头发都.了啊！如果使.MQ，A系统产..条数据，发送到MQ..去，哪个系统需要数据..去MQ..消费。如果新系统需要数据，直接从MQ.消费即可；如果某个系统不需要这条数据了，就取消对MQ消息的消费即可。这样下来，A系统压根.不需要去考虑要给谁发送数据，不需要维护这个代码，也不需要考虑.家是否调.成功、失败超时等情况。 总结：通过.个MQ，Pub/Sub发布订阅消息这么.个模型，A系统就跟其它系统彻底解耦了。 .试技巧：你需要去考虑.下你负责的系统中是否有类似的场景，就是.个系统或者.个模块，调.了多个系统或者模块，互相之间的调.很复杂，维护起来很.烦。但是其实这个调.是不需要直接同步调.接.的，如果.MQ给它异步化解耦，也是可以的，你就需要去考虑在你的项..，是不是可以运.这个MQ去进.系统的解耦。在简历中体现出来这块东西，.MQ作解耦。 异步 再来看.个场景，A系统接收.个请求，需要在..本地写库，还需要在BCD三个系统写库，..本地写库要3ms，BCD三个系统分别写库要300ms、450ms、200ms。最终请求总延时是3+300+450+200=953ms，接近1s，..感觉搞个什么东西，慢死了慢死了。..通过浏览器发起请求，等待个1s，这.乎是不可接受的。 .般互联.类的企业，对于..直接的操作，.般要求是每个请求都必须在200ms以内完成，对...乎是.感知的。 如果使.MQ，那么A系统连续发送3条消息到MQ队列中，假如耗时5ms，A系统从接受.个请求到返回响应给..，总时.是3+5=8ms，对于....，其实感觉上就是点个按钮，8ms以后就直接返回了，爽！.站做得真好，真快！ 削峰 每天0:00到12:00，A系统.平浪静，每秒并发请求数量就50个。结果每次.到12:00~13:00，每秒并发请求数量突然会暴增到5k+条。但是系统是直接基于MySQL的，.量的请求涌.MySQL，每秒钟对MySQL执.约5k条SQL。 .般的MySQL，扛到每秒2k个请求就差不多了，如果每秒请求到5k的话，可能就直接把MySQL给打死了，导致系统崩溃，..也就没法再使.系统了。 但是.峰期.过，到了下午的时候，就成了低峰期，可能也就1w的..同时在.站上操作，每秒中的请求数量可能也就50个请求，对整个系统.乎没有任何的压.。 如果使.MQ，每秒5k个请求写.MQ，A系统每秒钟最多处理2k个请求，因为MySQL每秒钟最多处理2k个。A系统从MQ中慢慢拉取请求，每秒钟就拉取2k个请求，不要超过..每秒能处理的最.请求数量就ok，这样下来，哪怕是.峰期的时候，A系统也绝对不会挂掉。.MQ每秒钟5k个请求进来，就2k个请求出去，结果就导致在中午.峰期（1个.时），可能有..万甚..百万的请求积压在MQ中。 这个短暂的.峰期积压是ok的，因为.峰期过了之后，每秒钟就50个请求进MQ，但是A系统依然会按照每秒2k个请求的速度在处理。所以说，只要.峰期.过，A系统就会快速将积压的消息给解决掉。 消息队列有什么优缺点 优点上.已经说了，就是在特殊场景下有其对应的好处，解耦、异步、削峰。缺点有以下.个：

系统可.性降低系统引.的外部依赖越多，越容易挂掉。本来你就是 A系统调. BCD三个系统的接.就好了，. ABCD四个系统好好的，没啥问题，你偏加个MQ进来，万.MQ挂了咋整，MQ.挂，整套系统崩溃的，你不就完了？如何保证消息队列的.可.，可以点击这.查看。

系统复杂度提.硬..加个 MQ进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头.头.，问题..堆，痛苦不已。

.致性问题 A系统处理完了直接返回成功了，.都以为你这个请求就成功了；但是问题是，要是BCD三个系统那.，BD两个系统写库成功了，

结果C系统写库失败了，咋整？你这数据就不.致了。 所以消息队列实际是.种.常复杂的架构，你引.它有很多好处，但是也得针对它带来的坏处做各种额外的技术.案和架构来规避掉，做好之后，你会发现，妈呀，系统复杂度提升了.个数量级，也许是复杂了10倍。但是关键时刻，.，还是得.的。 Kafka、ActiveMQ、RabbitMQ、RocketMQ有什么优缺点?

综上，各种对.之后，有如下建议： .般的业务系统要引.MQ，最早.家都.ActiveMQ，但是现在确实.家.的不多了，没经过.规模吞吐量场景的验证，社区也不是很活跃，所以.家还是算了吧，我个.不推荐.这个了； 后来.家开始.RabbitMQ，但是确实erlang语.阻.了.量的Java.程师去深.研究和掌控它，对公司..，.乎处于不可控的状态，但是确实.家是开源的，.较稳定的.持，活跃度也.； 不过现在确实越来越多的公司会去.RocketMQ，确实很不错，毕竟是阿.出品，但社区可能有突然.掉的.险（.前RocketMQ已捐给Apache，但GitHub上的活跃度其实不算.）对..公司技术实.有绝对.信的，推荐.RocketMQ，否则回去..实实. RabbitMQ吧，.家有活跃的开源社区，绝对不会.。 所以中.型公司，技术实.较为.般，技术挑战不是特别.，.RabbitMQ是不错的选择；.型公司，基础架构研发实.较强，.RocketMQ是很好的选择。 如果是.数据领域的实时计算、.志采集等场景，.Kafka是业内标准的，绝对没问题，社区活跃度很.，绝对不会.，何况.乎是全世界这个领域的事实性规范。 原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进...告技术 交流群。 推荐阅读 1. 把14亿中国.拉到.个微信群?

2. Spring中的18个注解，你会.个?
3.

寓教于乐，.玩游戏的.式学习Git

4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

Java后端：专注于Java技术

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

每..题：讲.讲你理解的微服务架构？ Java后端 2019-10-27 点击上.Java后端，选择设为星标 优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 讲.讲你理解的微服务架构？ .试题剖析 翻译.MartinFowler.站Microservices ..。.章篇幅较.，阅读需要.点耐.。 本..平有限，若有不妥之处，还请各位帮忙指正，谢谢。 过去.年中出现了“微服务架构”这.术语，它描述了将软件应.程序设计为若.个可独.部署的服务套件的特定.法。尽管这种架构.格尚未有精确的定义，但围绕业务能.、.动部署、端点智能以及语.和数据的分散控制等组织来说，它们还是存在着某些共同特征。 “微服务”——在拥挤的软件架构街道上..个新名词。虽然我们的.然倾向是对它轻蔑.瞥，但这.术语描述了.种越来越具有吸引.的软件系统.格。在过去.年中，我们已经看到许多项.使.了这种.格，到.前为.其结果都是正向的，以.于它变成了我们ThoughtWorks许多同事构建企业应.程序的默认.格。然.遗憾的是，并没有太多信息可以概述微服务的.格以及如何实现。 简..之，微服务架构.格[1]是.种将单个应.程序开发为.套.型服务的.法，每个.型服务都在..的进程中运.，并以轻量级机制（通常是 HTTP资源 API）进.通信。这些服务围绕业务功能构建，可通过全.动部署机制来独.部署。这些服务共..个最.型的集中式管理，它们可以使.不同的编程语.编写，并使.不同的数据存储技术。 在开始解释微服务.格之前，将它与单体（monolithic）.格进..较是有.的：单体应.程序被构建为单.单元。企业应.程序通常由三个部分构成：客.端..界.（由..机器上的浏览器中运.的HTML..和Javascript组成）、数据库（由许多表组成，通常是在关系型数据库中管理）系统、服务器端应.程序。服务器端应.程序处理 HTTP请求，执..些逻辑处理，从数据库检索和更新数据，选择数据并填充到要发送到浏览器的 HTML视图中。这个服务器端应.程序是.个整体..个逻辑可执..件[2]。对系统的任何更改都涉及构建和部署新版本的服务器端应.程序。 这种单体服务器是构建这种系统的.然.式。处理.个请求的所有逻辑都在.个进程中运.，允许你使.语.的基本功能将应.程序划分为类、函数和命名空间。需要注意的是，你可以在开发.员的笔记本电脑上运.和测试应.程序，并使.部署管道确保对程序做出的改动被适当测试并部署到.产环境中。你可以通过在负载均衡器后.运.许多实例来.平扩展整体块。 单体应.程序可以取得成功，但越来越多的.对它们感到不满——尤其是在将更多应.程序部署到云的时候。变更周期被捆绑在.起——即使只是对应.程序的..部分进.了更改，也需要重建和部署整个单体应.。随着时间的推移，通常很难保持良好的模块化结构，也更难以保持应该只影响该模块中的.个模块的更改。对系统进.扩展时，不得不扩展整个应.系统，.不能仅扩展该系统中需要更多资源的那些部分。这些不满催.了微服务架构.格：将应.程序构建为服务套件。除了服务可独.部署、独.扩展的事实之外，每个服务还提供了.个牢固的模块边界，甚.允许以不同的编程语.编写不同的服务。他们也可以由不同的团队管理。

我们并不认为微服务.格是新颖的或创新的，其根源.少可以追溯到 Un ix的设计原则。但我们认为没有.够多的.考虑微服务架构，如果使.它，许多软件的开发会变得更好。 微服务架构的特征 虽然不能说微服务架构.格有正式的定义，但我们可以尝试描述.下我们认为的在符合这个标签的架构中，它们所具有的.些共同特征。与概述共同特征的任何定义.样，并.所有微服务架构都具有所有特征，但我们确实期望.多数微服务架构都具有.多数特征。虽然我们的作者.直是这个相当宽松的社区的活跃成员，但我们的本意还是尝试描述我们两.在..和..所了解的团队的.作中所看到的情况。特别要说明的是，我们没有制定.些相关的定义。 通过服务进.组件化 只要我们参与软件.业，就.直希望通过将组件集成在.起来构建系统，就像我们在物理世界中看到的事物的构建.式.样。在过去的..年中，我们已经看到了.多数语.平台的公共软件库都取得了极.的进展。 在谈论组件时，就会碰到.个有关定义的难题，即什么是组件？我们的定义是，组件是可独.更换和升级的软件单元。 微服务架构也会使.软件库，但组件化软件的主要.式是拆分为多个服务。我们把库定义为链接到程序并使.内存函数调.来调.的组件，.服务是.种进程外组件，通过Web服务请求或远程过程调.等机制进.通信。（这与许多.向对象程序中的服务对象的概念是不同的 [3]。） 将服务作为组件（.不是库）的.个主要原因是服务可以独.部署。如果你有.个应.程序[4]是由单.进程.的多个库组成，任何.个组件的更改都会导致整个应.程序的重新部署。但如果应.程序可拆分为多个服务，那么单个服务的变更只需要重新部署该服务即可。当然这也不是绝对的，.些服务接.的修改可能会导致多个服务之间的协同修改，但.个好的微服务架构的.的是通过内聚服务边界和服务协议的演进机制来最.化这些协同修改。 将服务.作组件的另.个结果是更明确的组件接.。.多数语.没有.个良好的机制来定义显式发布的接.。通常，它只是.档和规则来阻.客.端破坏组件的封装，这会导致组件之间过于紧耦合。通过使.显式远程调.机制，服务可以更轻松地避免这种情况。 像这样使.服务确实存在.些不好的地.。远程调..进程内调.更昂贵，远程 API需要设计成较粗的粒度，这通常更难以使.。如果你需要更改组件之间的职责分配，那么当你跨越进程边界时，这种组件.为的改动会更加难以实现。 近似地，

我们可以把.个个服务映射为.个个运.时进程，但这仅仅是.个近似.已。.个服务可能包括多个始终.起开发和部署的进程，.如.个应.系统的进程和仅由该服务使.的数据库。 围绕业务能.进.组织 在将.型应.程序拆分为多个部分时，管理层往往侧重于技术层.，从.导致U I团队、服务器端逻辑团队、数据库团队的划分。当团队按照这些.式分开时，即便是简单的更改也可能导致跨团队项.的时间和预算批准。.个聪明的团队将围绕这个进.优化，“两害相权取其轻”——只需将逻辑强制应.到他们可以访问的任何应.程序中。换句话说，逻辑.处不在。这是康威定律 [5]的.个例.。 任何设计系统（.义上的）的组织都会产..种设计，其结构是组织通信结构的副本。——梅尔..康威，1967年

微服务采.不同的划分.式，它是围绕业务功能将系统拆分为多个服务。这些服务为该业务领域采..泛的软件实现，包括..界.、持久化存储和任何外部协作。因此，团队是跨职能的，包括开发所需的全部技能：..体验、数据库和项.管理。

以这种.式组建的.家公司是www.comparethemarket.com。跨职能团队负责构建和运营每个产品，每个产品拆分为多个独.的服务，彼此通过消息总线来通信。 .型单体应.程序也可以围绕业务功能进.模块化，尽管这不是常.的情况。当然，我们会敦促构建单体应.系统的.型团队根据业务线来将..分解为若..团队。我们在这.看到的主要问题是，它们往往围绕太多的上下.进.组织。如果单体跨越了模块边界，对团队的个体成员来说，很难将它们装.短期的记忆中。此外，我们看到模块化.产线需要.量的规则来执.。服务组件所要求的更加明确的分离，使得它更容易保持团队边界清晰。 是产品不是项. 我们看到的.多数应.程序开发.作都使.这样.个项.模式：.标是交付.些软件，然后就完.了。.旦完成后，软件将移交给维护组织，然后构建它的项.团队也随之解散了。 微服务.持者倾向于避免这种模式，.是认为团队应该负责产品的整个.命周期。对此.个共同的启.是亚.逊的”you build, you run it”的概念，开发团队对.产中的软件负全部责任。这使开发者经常接触他们的软件在.产环境如何.作，并增加与他们的..联系，因为他们必须承担.少部分的.持.作。 产品.态与业务能.的联系紧密相连。要持续关注软件如何帮助..提升业务能.，.不是把软件看成是将要完成的.组功能。 没有理由说为什么这种.法不能.在单.应.程序上，但较.的服务粒度，使得它更容易在服务开发者和..之间建.个.关系。 智能端点和哑管 在不同进程之间建.通信时，我们已经看到许多产品和.法，都强调将.量的智能特性放.通信机制本..。.个很好的例.是企业服务总线（ESB），其中ESB产品通常包括.于消息路由、编排、转换和应.业务规则的复杂.具。 微服务社区倾向于采.另.种.法：智能端点和哑管。基于微服务构建的应.程序的.标是尽可能的解耦和尽可能的内聚——他们拥有..的领域逻辑，他们的.为更像经典 UNIX理念中的过滤器.接收请求，应.适当的逻辑并产.响应。使.简单的REST.格的协议来编排它们，.不是使.像WS-Choreography或者BPEL或者通过中..具编制(orchestrat ion)等复杂的协议。 最常.的两种协议是带有资源AP I的HTTP请求-响应和轻量级消息传递 [8]。对第.种协议最好的表述是本.就是web，.不是隐藏在web的后.。——IanRobinson 微服务团队使.的规则和协议，正是构建万维.的规则和协议(在更.程度上是UN IX的)。从开发者和运营.员的.度讲，通常使.的资源可以很容易的缓存。 第.种常..法是在轻量级消息总线上传递消息。选择的基础设施是典型的哑的(哑在这.只充当消息路由器).像 RabbitMQ或ZeroMQ这样简单的实现仅仅提供.个可靠的异步交换结构.在服务.，智能特性仍旧存在于那些.产和消费诸多消息的各个端点中，即存在于各个服务中。 单体应.中，组件都在同.进程内执.，它们之间通过.法调.或函数调.通信。把单体变成微服务最.的问题在于通信模式的改变。.种幼稚的转换是从内存.法调.转变成RPC，这导致频繁通信且性能不好。相反，你需要.粗粒度通信代替细粒度通信。 去中.化的治理 集中治理的.个后果是单.技术平台的标准化发展趋势。经验表明，这种.法正在收缩.不是每个问题都是钉.，不是每个问题都是锤.。我们更喜欢使.正确的.具来完成.作，.单体应.程序在.定程度上可以利.语.的优势，这是不常.的。 把单体的组件分裂成服务，在构建这些服务时可以有..的选择。你想使.Node. js开发.个简单的报告..？去吧。.C++实现.个特别粗糙的近乎实时的组件？好极了。你想换..个更适合组件读操作数据的不同.格的数据库？我们有技术来重建它。 当然，仅仅因为你可以做些什么，.不意味着你应该这样做——但.这种.式划分系统意味着你可以选择。 团队在构建微服务时也更喜欢.不同的.法来达标。他们更喜欢.产有.的.具这种想法，.不是写在纸上的标准，这样其他开发者可以.这些.具解决他们所.临的相似的问题。有时，这些.具通常在实施中收获并与更.泛的群体共享，但不完全使..个内部开源模型。现在g it和g ithub已经成为事实上的版本控制系统的选择，在内部开放源代码的实践也正变得越来越常.。 Net.ix是遵循这.理念的.个很好的例.。尤其是，以库的形式分享有.的且经过市场检验的代码，这激励其他开发者.类似的.式解决相似的问题，同时还为采.不同.法敞开了...。共享库倾向于聚焦在数据存储、进程间通信和我们接下来要深.讨论的基础设施.动化的共性问题。 对于微服务社区来说，开销特别缺乏吸引.。这并不是说社区不重视服务合约。恰恰相反，因为他们有更多的合约。只是他们正在寻找不同的.式来管理这些合约。像To lerantReader和Consumer-Dr ivenContracts这样的模式通常被.于微服务。这些援助服务合约在独.进化。执.消费者驱动的合约作为构建的.部分，增加了信.并对服务是否在运作提供了更快的反馈。事实上，我们知道澳.利亚的.个团队.消费者驱动的合约这种模式来驱动新业务的构建。他们使.简单的.具定义服务的合约。这已变成.动构建的.部分，即使新服务的代码还没写。服务仅在满.合约的时候才被创建出来 -这是在构建新软件时避免 "YAGNI"[9]困境的.个优雅的.法。围绕这些成.起来的技术和.具，通过减少服务间的临时耦合，限制了中.合约管理的需要。 也许去中.化治理的最.境界就是亚.逊.为流传的bu ildit/runit理念。团队要对他们构建的软件的各..负责，包括7*24.时的运营。这.级别的责任下放绝对是不规范的，但我们看到越来越多的公司让开发团队负起更多责任。Net.ix是采.这.理念的另.家公司 [11]。每天凌晨 3点被传呼机叫醒.疑是.个强有.的激励，使你在写代码时关注质量。这是关于尽可能远离传统的集中治理模式的.些想法。 分散数据管理 数据管理的去中.化有许多不同的呈现.式。在最抽象的层.

上，这意味着使系统间存在差异的世界概念模型。在整合.个.型企业时，客.的销售视图将不同于.持视图，这是.个常.的问题。客.的销售视图中的.些事情可能不会出现在.持视图中。 它们确实可能有不同的属性和 (更坏的)共同属性，这些共同属性在语义上有微妙的不同。 这个问题常.于应.程序之间，但也可能发.在应.程序内部，尤其当应.程序被划分成分离的组件时。.个有.的思维.式是有界上下. (BoundedContext)内的领域驱动设计 (Domain-DrivenDesign,DDD)理念。DDD 把.个复杂域划分成多个有界的上下.，并且映射出它们之间的关系。这个过程对单体架构和微服务架构都是有.的，但在服务和上下.边界间有天然的相关性，边界有助于澄清和加强分离，就像业务能.部分描述的那样。 和概念模型的去中.化决策.样，微服务也去中.化数据存储决策。虽然单体应.程序更喜欢单.的逻辑数据库做持久化存储，但企业往往倾向于.系列应.程序共..个单.的数据库——这些决定是供应商授权许可的商业模式驱动的。微服务更倾向于让每个服务管理..的数据库，或者同.数据库技术的不同实例，或完全不同的数据库系统 -这就是所谓的混合持久化 (Polyglot Persistence)。你可以在单体应.程序中使.混合持久化，但它更常出现在为服务.。

对跨微服务的数据来说，去中.化责任对管理升级有影响。处理更新的常..法是在更新多个资源时使.事务来保证.致性。这个.法通常.在单体中。 像这样使.事务有助于.致性，但会产.显著地临时耦合，这在横跨多个服务时是有问题的。分布式事务是出了名的难以实现，因此微服务架构强调服务间的.事务协作，对.致性可能只是最后.致性和通过补偿操作处理问题有明确的认知。 对很多开发团队来说，选择.这样的.式管理不.致性是.个新的挑战，但这通常与业务实践相匹配。通常业务处理.定程度的不.致，以快速响应需求，同时有某些类型的逆转过程来处理错误。这种权衡是值得的，只要修复错误的代价.于更..致性下损失业务的代价。 基建.动化 基础设施.动化技术在过去.年中发.了巨.变化——特别是云和AWS的发展降低了构建、部署和运.微服务的操作复杂性。 许多使.微服务构建的产品或系统都是由具有丰富的持续交付和持续集成经验的团队构建的。以这种.式构建软件的团队.泛使.基础设施.动化技术。如下.显.的构建管道所.。

由于这并不是.篇关于持续交付的.章，我们在这.只关注持续交付的.个关键特性。我们希望有尽可能多的信.确保我们的软件正常运.，因此我们进.了.量的.动化测试。想让软件达到“晋级”(Promotion)状态从.“推上”流.线，就意味着要在每.个新的环境中，对软件进..动化部署。 .个单体应.程序可以.常愉快地通过这些环境构建、测试和推动。事实证明，.旦你为单体投.了.动化整体.产，那么部署更多的应.程序似乎不再那么可怕了。请记住，持续交付的.标之.就是让“部署”.作变得“枯燥”，所以.论是.个还是三个应.程序，只要部署.作依旧很“枯燥”，那么就没什么可担.的了[12]。 我们看到团队.量的基础设施.动化的另.个领域是在管理.产环境中的微服务。与我们上.的断.（只要部署很.聊）相.，单体和微服务之间没有太.的区别，但是每个部署的运.环境可能会截然不同。

设计时为故障做好准备 使.服务作为组件的结果是，需要设计应.程序以便它们能够容忍服务的失败。如果服务提供者商不可.，任何服务呼叫都可能失败，客.必须尽可能优雅地对此做出响应。与单体设计相.，这是.个缺点，因为它这会引.额外的复杂性来处理它。结果是微服务团队不断反思服务失败是如何影响..体验的。Net. ix的 Simian Army能够引发服务甚.数据中.的故障在.作.发.故障，从.来测试应.程序的弹性和监控能.。 .产中的这种.动化测试.以让.多数运维团队兴奋得浑.颤栗，就像在.周的.假即将到来前.样。这并不是说单体架构.格不能构建先进的监控系统——只是根据我们的经验，这在单体系统中并不常.罢了。 由于服务可能随时发.故障，因此能够快速检测故障并在可能的情况下.动恢复服务就显得.关重要。微服务应.程序.常重视应.程序的实时监控，.如检查架构元素（数据库每秒获得多少请求）和业务相关度量（例如每分钟收到多少订单）。语义监控可以提供出现问题的早期预警系统，从.触发开发团队跟进和调查。 这对于微服务架构来说尤为重要，因为微服务偏好编排和事件写作，这会导致.些紧急状况。虽然许多权威..对于偶然事件的价值持积极态度，但事实是，“突发.为”有时可能是.件坏事。监控.关重要，它能够快速发现不良紧急.为并进.修复。单体系统也可以像微服务.样实现透明的监控——事实上，它们也应该如此。不同之处在于你必须能够知道在不同进程中运.的服务在何时断开了连接。对于同.过程中的库，这种透明性.处并不.。 微服务团队希望看到针对每个服务的复杂监控和.志记录，例如显.“运./宕机”状态的仪表盘以及各种运维和业务相关的指标。有关断路器状态，当前吞吐量和延迟的详细信息也是我们在.作中经常遇到的其他例.。 演化设计 微服务从业者通常有进化设计的背景，并把服务分解视为进.步的.具，使应.程序开发.员能够控制应.程序中的更改，.不会降低变更速度。变更控制并不.定意味着变更的减少——在正确的态度和.具的帮助下，你可以对软件进.频繁，快速且有良好控制的更改。 每当要试图将软件系统分解为组件时，你就会.临这样的决策，即如何进.拆分——我们决定拆分应.程序的原则是什么？组件的关键属性具有独.替换和可升级性的特点 [13]——这意味着我们寻找这些点，想象如何在不影响其协作者的情况下重写组件。实际上，许多微服务组通过明确地期望许多服务被废弃.不是.期演变来进.步考虑这.点。 Guardian.站是设计和构建成单体应.程序的.个很好的例.，但是它也在微服务.向上不断发展演化。原先的单体系统仍然是.站的核.，但他们更喜欢通过构建.些微服务 AP I的.式来添加新的功能。这种.法对于本质上是临时的功能尤其.便，例如处理体育赛事的专...。.站的这.部分可以使.快速开发语.快速组合在.起，在赛事结束后.即删除。我们在.融机构看到过类似的.法，为市场机会增加新服务，并在.个.甚..周后丢弃。这种强调可替换性的特点，是模块化设计.般性原则的.个特例，即通过变化模式来驱动模块化的实现[14]。.家都愿意将那些同时发.变化的东西放在同.个模块，很少变化的系统模块应该与.前正在经历.量变动的系统处于不同的服务中。如果你发现..反复更改两项服务，那就表明它们应该合并了。 将组件放.服务中可以为更细粒度的发布计划添加机会。对于

单体来说，任何更改都需要完整构建和部署整个应.程序。但是，使.微服务，你只需要重新部署你修改的服务。这可以简化并加快发布过程。缺点是你必须担..项服务的变化会打破其消费者。传统的集成.法是尝试使.版本控制来解决这个问题，但微服务世界中的偏好是仅仅把使.版本控制作为最后的.段。我们可以通过设计服务尽可能容忍服务提供者的变化来避免.量的版本控制。 微服务是未来吗？ 我们写这篇.章的主要.的是解释微服务的主要思想和原则。通过花时间来做到这.点，我们清楚地认为微服务架构.格是.个重要的想法——在研发企业系统时，值得对它进.认真考虑。我们最近使.这种.式构建了.个系统，并且了解到其它团队也赞同这种.格。 我们了解到那些在某种程度上开创这种架构.格的先驱，包括亚.逊、Net flix、英国卫报、英国政府数字化服务中.、 realestate.com.au、Forward和comparethemarket.com。2013年的技术会议上充满了.些公司的例.，这些公司正在转向可以归类为微服务的公司，包括Trav isCI。此外，有很多组织.期以来.直在做我们称之为微服务的东西，但没有使.过这个名字。（通常这被标记为SOA——尽管如我们所说，SOA有许多相互.盾的形式。[15]） 然.，尽管有这些积极的经验，但并不是说我们确信微服务是软件架构的未来发展.向。虽然到.前为.我们的经验与整体应.相.是积极的，但我们意识到没有.够的时间让我们做出充分完整的判断。 通常，架构决策所产.的真正效果，只有在该决策做出若.年后才能真正显现。我们已经看到由带着强烈的模块化愿望的优秀团队所做的.些项.，最终却构建出.个单体架构，并在.年之内不断腐化。许多.认为，如果使.微服务就不.可能出现这种腐化，因为服务的边界是明确的，.且难以随意搞乱。然.，对于那些开发时间.够.的各种系统，除.我们已经.识得.够多，否则我们.法真正评价微服务架构是如何成熟的。有.觉得微服务或许很难成熟起来，这当然是有原因的。在组件化上所做的任何.作的成功与否，取决于软件与组件的匹配程度。准确地搞清楚某个组件的边界的位置应该出现在哪.，是.项困难的.作。进化设计承认难以对边界进.正确定位，所以它将.作的重点放到了易于对边界进.重构之上。但是当各个组件成为各个进.远程通信的服务后，.起在单.进程内进.各个软件库之间的调.，此时的重构就变得更加困难。跨越服务边界的代码移动就变得困难起来。接.的任何变化，都需要在其各个参与者之间进.协调。向后兼容的层次也需要被添加进来。测试也会变得更加复杂。 另.个问题是，如果这些组件不能.净利落地组合成.个系统，那么所做的.切.作，仅仅是将组件内的复杂性转移到组件之间的连接之上。这样做的后果，不仅仅是将复杂性搬了家，它还将复杂性转移到那些不再明确且难以控制的边界之上。当在观察.个.型且简单的组件内部时，.们很容易觉得事情已经变得更好了，然.他们却忽视了服务之间杂乱的连接。 最后，还有.个团队技能的因素。新技术往往会被技术更加过硬的团队所采.。对于技术更加过硬的团队.更有效的.项技术，不.定适.于.个技术略逊.筹的团队。我们已经看到.量这样的案例，那些技术略逊.筹的团队构建出了杂乱的单体架构。当这种杂乱发.到微服务.上时，会出现什么情况？这需要花时间来观察。.个糟糕的团队，总会构建.个糟糕的系统——在这种情况下，很难讲微服务究竟是减少了杂乱，还是让事情变得更糟。 我们听到的.个合理的论点是，你不应该从微服务架构开始，.是从整体开始，保持模块化，并在整体出现问题时将其拆分为微服务。（这个建议并不理想，因为好的进程内接.通常不是.个好的服务接.。） 所以我们谨慎乐观地写下这个。到.前为.，我们已经看到了.够多的微服务.格，觉得它可能是.条值得.的路。我们.法确定最终会在哪.结束，但软件开发的挑战之.是你只能根据你当前必须拥有的不完善信息做出决策。 tips：欢迎关注微信公众号：Java后端，获取每.技术博.推送。 脚注 1:Theterm "microservice" wasdiscussedataworkshopofsoftwarearchitectsnearVeniceinMay,2011todescribe whatthe participantssawas a common architectural stylethatmany ofthem had beenrecentlyexploring. In May 2012,thesamegroupdecidedon"microservices"asthemostappropriatename.Jamespresentedsomeoftheseideas asacasestudyinMarch2012at33rdDegreeinKrakowinMicroservices-Java,theUnixWayasdidFredGeorgeabout thesametime.AdrianCockcroftatNet.ix,describingthisapproachas".negrainedSOA"waspioneeringthestyleat webscaleasweremanyoftheothersmentioned inthisarticle-JoeWalnes,DanNorth,EvanBotcherandGraham Tackley. 2:ThetermmonolithhasbeeninusebytheUnixcommunityforsometime.ItappearsinTheArtofUnixProgramming todescribesystemsthatgettoobig. 3:Manyobject-orienteddesigners,includingourselves,usethetermserviceobjectintheDomain-DrivenDesignsense foranobjectthatcarriesoutasigni.cantprocessthatisn'ttiedtoanentity.Thisisadi.erentconcepttohowwe're using"service"inthisarticle.Sadlythetermservicehasbothmeaningsandwehavetolivewiththepolyseme. 4:Weconsideranapplicationtobeasocialconstructionthatbindstogetheracodebase,groupoffunctionality,and bodyoffunding. 5:TheoriginalpapercanbefoundonMelvynConway'swebsitehere. 6:Wecan'tresistmentioningJimWebber'sstatementthatESBstandsfor"EgregiousSpaghettiBox". 7:Netflixmakesthelinkexplicit-untilrecentlyreferringtotheirarchitecturalstyleasfine-grainedSOA. 8:Atextremesofscale,organisationsoftenmovetobinaryprotocols-protobufsforexample.Systemsusingthesestill exhibitthecharacteristicofsmartendpoints,dumbpipes-andtradeo.transparencyforscale.Mostwebproperties andcertainlythevastmajorityofenterprisesdon'tneedtomakethistradeoff-transparencycanbeabigwin. 9:"YAGNI"or"YouAren'tGoingToNeedIt"isanXPprincipleandexhortationtonotaddfeaturesuntilyouknowyou needthem. 10: It'sa littledisengenuousofustoclaimthatmonolithsaresinglelanguage -inordertobuildsystemsontodays web,youprobablyneedtoknowJavaScriptandXHTML,CSS,yourserverside languageofchoice,SQLandanORM dialect.Hardlysinglelanguage,butyouknowwhatwemean. 11:AdrianCockcroftspeci.callymentions"developerself-

service"and"Developersrunwhattheywrote"(sic)inthis excellentpresentationdeliveredatFlowconinNovember,2013.
12:Wearebeingalittledisengenuoushere.Obviouslydeployingmoreservices,inmorecomplextopologiesismore di.cultthandeployingasinglemonolith.Fortunately,patternsreducethiscomplexity-investmentintoolingisstilla mustthough.
13:Infact,DanNorthreferstothisstyleasReplaceableComponentArchitectureratherthanmicroservices.Sincethis seemstotalktoasubsetofthecharacteristicswepreferthelatter.
14:KentBeckhighlightsthisasonehisdesignprinciplesinImplementationPatterns.
15:AndSOAishardlytherootofthishistory.Irememberpeoplesaying"we'vebeendoingthisforyears"whenthe SOAtermappearedatthebeginningofthecentury.Oneargumentwasthatthisstyleseesitsrootsasthewaycobol programscommunicatedviadata.les intheearliestdaysofenterprisecomputing. Inanotherdirection,onecould argue that microservices are the same thing as the Erlang programming model, but applied to an enterprise applicationcontext.
-END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后即可获取每..题的推送。 推荐阅读 1. 每..题：消息队列.试常问题.

2.

每..题：如何保证消息队列的.可.?

3.

每..题：如何设计.个.并发系统?

4.

每..题：为什么要进.系统拆分?

5.

每..题：你有没有做过MySQL读写分离?

6.

每..题：Java线程池8.拒绝策略

7.

每..题：分布式事务

8.

让你写.个消息队列，该如何设计?

9....试官最喜欢问的.试难点

10.

分享100道Linux笔试题

11. InnoDB中.棵B+树能存多少.数据?
12.

每..题：如何保证消息的顺序性?

13.

DubboRPC常问.试题整理好了，来拿!

14.

每..题：如何保证消息不被重复消费?

15.

每..题：Zookeeper都有哪些使.场景？

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

看完这篇HTTP，.试官就难不倒你了 Java后端 2.26. 以下.章来源于Java建设者，作者cxuan

Java建设者 cxuan的个.公众号，只分享原创精品技术.章，不定期更新。

我是.名程序员，我的主要编程语.是 Java，我更是.名Web开发.员，所以我必须要了解 HTTP，所以本篇.章就来带你从HTTP..到进阶，看完让你有.种恍然.悟、醍醐灌顶的感觉。 最初在有.络之前，我们的电脑都是单机的，单机系统是孤.的，我还记得 05年前那会.家.有个电脑，想打电脑游戏还得两个.在.个电脑上玩.，及其不.便。我就想为什么家..不让上.，我的同学 xxx家.有.，每次.提这个就落.通批评：xxx上xxx什xxxx么xxxx.xxxx看xxxx你xxxx考xxxx的xxxx那xxxx点xxxx分。虽然我家.没有上.，但是此时互联.已经在.速发展了，HTTP就是.速发展的.个产物。 认识HTTP .先你听的最多的应该就是HTTP是.种超.本传输协议(HypertextTransferProtocol)，这你.定能说出来，但是这样还不够，假如你是...试官，这不可能是他想要的最终结果，我们在.试的时候往往把..知道的尽可能多的说出来，才有和.试官谈价钱的资本。那么什么是超.本传输协议？ 超.本传输协议可以进..字分割：超.本（Hypertext）、传输（Transfer）、协议（Protocol），它们之间的关系如下

按照范围的..协议>传输>超.本。下.就分别对这三个名次做.个解释。 什么是超.本 在互联.早期的时候，我们输.的信息只能保存在本地，.法和其他电脑进.交互。我们保存的信息通常都以.本即简单字符的形式存在，.本是.种能够被计算机解析的有意义的.进制数据包。.随着互联.的.速发展，两台电脑之间能够进.数据的传输后，.们不满.只能在两台电脑之间传输.字，还想要传输图.、.频、视频，甚.点击.字或图.能够进.超链接的跳转，那么.本的语义就被扩.了，这种语义扩.后的.本就被称为超.本 (Hypertext)。 什么是传输 那么我们上.说到，两台计算机之间会形成互联关系进.通信，我们存储的超.本会被解析成为.进制数据包，由传输载体（例如同轴电缆，电话线，光缆）负责把.进制数据包由计算机终端传输到另.个终端的过程（对终端的详细解释可以参考你说你懂互联.，那这些你知道么？这篇.章）称为传输(transfer)。 通常我们把传输数据包的..称为请求.，把接到.进制数据包的..称为应答.。请求.和应答.可以进.互换，请求.也可以作为应答.接受数据，应答.也可以作为请求.请求数据，它们之间的关系如下

如图所.，A和 B是两个不同的端系统，它们之间可以作为信息交换的载体存在，刚开始的时候是 A作为请求.请求与B交换信息，B作为响应的..提供信息；随着时间的推移，B也可以作为请求.请求A交换信息，那么A也可以作为响应.响应B请求的信息。 什么是协议 协议这个名词不仅局限于互联.范畴，也体现在.常.活中，.如情侣双.约定好在哪个地点吃饭，这个约定也是.种协议，.如你应聘成功了，企业会和你签订劳动合同，这种双.的雇佣关系也是.种协议。注意...个.对..的约定不能成为协议，协议的前提条件必须是多.约定。 那么.络协议是什么呢？ .络协议就是.络中(包括互联.)传递、管理信息的.些规范。如同.与.之间相互交流是需要遵循.定的规矩.样，计算机之间的相互通信需要共同遵守.定的规则，这些规则就称为.络协议。 没有.络协议的互联.是混乱的，就和.类社会.样，.不能想怎么样就怎么样，你的.为约束是受到法律的约束的；那么互联.中的端系统也不能..想发什么发什么，也是需要受到通信协议约束的。 那么我们就可以总结.下，什么是HTTP？可以.下.这个经典的总结回答.下：HTTP是.个在计算机世界.专.在两点之间传输.字、图.、.频、视频等超.本数据的约定和规范 与HTTP有关的组件 随着.络世界演进，HTTP协议已经.乎成为不可替代的.种协议，在了解了 HTTP的基本组成后，下.再来带你进.步认识.下HTTP协议。 .络模型 .络是.个复杂的系统，不仅包括.量的应.程序、端系统、通信链路、分组交换机等，还有各种各样的协议组成，那么现在我们就来聊.下.络中的协议层次。 为了给.络协议的设计提供.个结构，.络设计者以分层(layer)的.式组织协议，每个协议属于层次模型之.。每.层都是向它的上.层提供服务(service)，即所谓的服务模型(servicemodel)。每个分层中所有的协议称为协议栈 (protocolstack)。因特.的协议栈由五个部分组成：物理层、链路层、.络层、运输层和应.层。我们采..上.下的.法研究其原理，也就是应.层->物理层的.式。应.层 应.层是.络应.程序和.络协议存放的分层，因特.的应.层包括许多协议，例如我们学web离不开的HTTP，电.邮件传送协议SMTP、端系统.件上传协议FTP、还有为我们进.域名解析的DNS协议。应.层协议分布在多个端系统上，.个端系统应.程序与另外.个端系统应.程序交换信息分组，我们把位于应.层的信息分组称为报.(message)。 运输层 因特.的运输层在应.程序断点之间传送应.程序报.，在这.层主要有两种传输协议TCP和 UDP，利.这两者中的任何.个都能够传输报.，不过这两种协议有巨.的不同。 TCP向它的应.程序提供了.向连接的服务，它能够控制并确认报.是否到达，并提供了拥塞机制来控制.络传输，因此当.络拥塞时，会抑制其传输速率。 UDP协议向它的应.程序提供了.连接服务。它不具备可靠性的特征，没有流量控制，也没有拥塞控制。我们把运输层的分组称为报.段(segment) .络层 因特.的.络层负责将称为数据报(datagram)的.络分层从.台主机移动到另.台主机。.络层.个.常重要的协议是IP协议，所有具有.络层的因特.组件都必须运.IP协议，IP协议是.种.际协议，除了IP协议外，.络层还包括.些其他.际协议和路由选择协议，.般把.络层就称为IP

层，由此可知IP协议的重要性。 链路层 现在我们有应.程序通信的协议，有了给应.程序提供运输的协议，还有了.于约定发送位置的 IP协议，那么如何才能真正的发送数据呢？为了将分组从.个节点（主机或路由器）运输到另.个节点，.络层必须依靠链路层提供服务。链路层的例.包括以太.、WiFi和电缆接.的 DOCSIS协议，因为数据从源.的地传送通常需要经过.条链路，.个数据包可能被沿途不同的链路层协议处理，我们把链路层的分组称为帧(frame) 物理层 虽然链路层的作.是将帧从.个端系统运输到另.个端系统，.物理层的作.是将帧中的.个个.特从.个节点运输到另.个节点，物理层的协议仍然使.链路层协议，这些协议与实际的物理传输介质有关，例如，以太.有很多物理层协议：关于双绞铜线、关于同轴电缆、关于光纤等等。 五层.络协议的.意图如下

OSI模型 我们上.讨论的计算.络协议模型不是唯.的协议栈，ISO（国际标准化组织）提出来计算机.络应该按照7层来组织，那么7层.络协议栈与5层的区别在哪.？ 从图中可以.眼看出，OSI要.上.的.络模型多了表.层和会话层，其他层基本.致。表.层主要包括数据压缩和数据加密以及数据描述，数据描述使得应.程序不必担.计算机内部存储格式的问题，.会话层提供了数据交换的定界和同步功能，包括建.检查点和恢复.案。

浏览器 就如同各.邮箱使.电.邮件传送协议SMTP .样，浏览器是使. HTTP协议的主要载体，说到浏览器，你能想起来.种？是的，随着.景.战结束后，浏览器迅速发展，.今已经出现过的浏览器主要有

浏览器正式的名字叫做Web Broser，顾名思义，就是检索、查看互联.上..资源的应.程序，名字.的 Web，实际上指的就是WorldWideWeb，也就是万维.。 我们在地址栏输.URL（即.址），浏览器会向DNS（域名服务器，后.会说）提供.址，由它来完成URL到IP地址的映射。然后将请求你的请求提交给具体的服务器，在由服务器返回我们要的结果（以HTML编码格式返回给浏览器），浏览器执.HTML编码，将结果显.在浏览器的正.。这就是.个浏览器发起请求和接受响应的过程。 Web服务器 Web服务器的正式名称叫做WebServer，Web服务器.般指的是.站服务器，上.说到浏览器是HTTP请求的发起.，那么Web服务器就是HTTP请求的应答.，Web服务器可以向浏览器等Web客.端提供.档，也可以放置.站.件，让全世界浏览；可以放置数据.件，让全世界下载。.前最主流的三个Web服务器是Apache、Nginx、 IIS。 CDN CDN的全称是ContentDeliveryNetwork，即内容分发.络，它应.了HTTP协议.的缓存和代理技术，代替源站响应客.端的请求。CDN是构建在现有.络基础之上的.络，它依靠部署在各地的边缘服务器，通过中.平台的负载均衡、内容分发、调度等功能模块，使..就近获取所需内容，降低.络拥塞，提...访问响应速度和命中率。CDN的关键技术主要有内容存储和分发技术。 打..说你要去亚.逊上买书，之前你只能通过购物.站购买后从美国发货过海关等重重关卡送到你的家.，现在在中国建..个亚.逊分基地，你就不.通过美国进.邮寄，从中国就能把书尽快给你送到。 WAF WAF是.种 Web应.程序防护系统（Web Application Firewall，简称 WAF），它是.种通过执..系列针对HTTP/HTTPS的安全策略来专.为Web应.提供保护的.款产品，它是应.层.的防.墙，专.检测HTTP流量，是防护Web应.的安全技术。 WAF通常位于 Web服务器之前，可以阻.如 SQL注.、跨站脚本等攻击，.前应.较多的.个开源项.是 ModSecurity，它能够完全集成进Apache或Nginx。 WebService WebService是.种Web应.程序，WebService是.种跨编程语.和跨操作系统平台的远程调.技术。 WebService是.种由W3C定义的应.服务开发规范，使.client-server主从架构，通常使.WSDL定义服务接.，使.HTTP协议传输XML或SOAP消息，它是.个基于Web（HTTP）的服务架构技术，既可以运.在内.，也可以在适当保护后运.在外.。 HTML HTML称为超.本标记语.，是.种标识性的语.。它包括.系列标签．通过这些标签可以将.络上的.档格式统.，使分散的Internet资源连接为.个逻辑整体。HTML.本是由HTML命令组成的描述性.本，HTML命令可以说明.字，图形、动画、声.、表格、链接等。 Web..构成 Web..（Web page）也叫做.档，是由.个个对象组成的。.个对象(Objecy)只是.个.件，.如.个 HTML.件、.个JPEG图形、.个Java.程序或.个视频.段，它们在.络中可以通过URL地址寻址。多数的Web..含有.个HTML基本.件以及.个引.对象。 举个例.，如果.个Web..包含HTML.件和5个JPEG图形，那么这个Web..就有6个对象：.个HTML.件和5个JPEG图形。HTML基本.件通过URL地址引...中的其他对象。 与HTTP有关的协议 在互联.中，任何协议都不会单独的完成信息交换，HTTP也.样。虽然 HTTP属于应.层的协议，但是它仍然需要其他层次协议的配合完成信息的交换，那么在完成.次 HTTP请求和响应的过程中，需要哪些协议的配合呢？.起来看.下 TCP/IP TCP/IP协议你.定听过，TCP/IP我们.般称之为协议簇，什么意思呢？就是 TCP/IP协议簇中不仅仅只有 TCP协议和 IP协议，它是.系列.络通信协议的统称。.其中最核.的两个协议就是 TCP / IP协议，其他的还有 UDP、 ICMP、ARP等等，共同构成了.个复杂但有层次的协议栈。 的缩写，意思是传输控制协议，HTTP使. TCP作为通信协TCP协议的全称是TransmissionControlProtocol 议，这是因为TCP是.种可靠的协议，.可靠能保证数据不丢失。 IP协议的全称是InternetProtocol的缩写，它主要解决的是通信双.寻址的问题。IP协议使. IP地址来标识互联.上的每.台计算机，可以把 IP地址想象成为你.机的电话号码，你要与他.通话必须先要知道他.的.机号码，计算机.络中信息交换必须先要知道对.的IP地址。（关于TCP和IP更多的讨论我们会在后.详解） DNS 你有没有想过为什么你可以通过键.www.google.com就能够获取你想要的.站？我们上.说到，计算机.络中的每个端系统都有.个IP地址存在，.把IP地址转换为便于.类记忆的协议就是DNS协议。 DNS的全称是域名系统（DomainNameSystem，缩写：DNS），它作为将域名和 IP地址相互映射的.个分布式数据库，能够使.更.便地访问互联.。 URI/URL 我们上.提到，你可以通过输. www.google.com地址来访问.歌的官.，那么这个地址有什么规定吗？我怎么输都可以？AAA.BBB.CCC是不是也.？当然不是的，你输.的地址格式必须要满.URI的规范。 URI的全称是（Uniform

Resource Identi.er），中.名称是统.资源标识符，使.它就能够唯.地标记互联.上资源。 URL的全称是（Uniform Resource Locator），中.名称是统.资源定位符，也就是我们俗称的.址，它实际上是 URI的.个.集。 URI不仅包括URL，还包括URN（统.资源名称），它们之间的关系如下

HTTPS HTTP.般是明.传输，很容易被攻击者窃取重要信息，鉴于此，HTTPS应运..。HTTPS的全称为（Hyper Text Transfer Protocolover SecureSocket Layer），全称有点.，HTTPS和 HTTP有很.的不同在于 HTTPS是以安全为.标的HTTP通道，在 HTTP的基础上通过传输加密和.份认证保证了传输过程的安全性。HTTPS在HTTP的基础上增加了SSL层，也就是说HTTPS=HTTP+SSL。（这块我们后.也会详谈HTTPS） HTTP请求响应过程 你是不是很好奇，当你在浏览器中输..址后，到底发.了什么事情？你想要的内容是如何展现出来的？让我们通过.个例.来探讨.下，我们假设访问的URL地址为 http://www.someSchool.edu/someDepartment/home.index，当我们输..址并点击回.时，浏览器内部会进.如下操作

DNS服务器会.先进.域名的映射，找到访问www.someSchool.edu所在的地址，然后HTTP客.端进程在80端.发起.个到服务器www.someSchool.edu的TCP连接（80端.是HTTP的默认端.）。在客.和服务器进程中都会有.个套接字与其相连。HTTP客.端通过它的套接字向服务器发送.个 HTTP请求报.。该报.中包含了路径 someDepartment/home.index的资源，我们后.会详细讨论HTTP请求报.。 HTTP服务器通过它的套接字接受该报.，进.请求的解析.作，并从其存储器(RAM或磁盘)中检索出对象 www.someSchool.edu/someDepartment/home.index，然后把检索出来的对象进.封装，封装到HTTP响应报.中，并通过套接字向客.进.发送。HTTP服务器随即通知TCP断开TCP连接，实际上是需要等到客.接受完响应报.后才会断开TCP连接。HTTP客.端接受完响应报.后，TCP连接会关闭。HTTP客.端从响应中提取出报.中是.个 HTML响应.件，并检查该HTML.件，然后循环检查报.中其他内部对象。检查完成后，HTTP客.端会把对应的资源通过显.器呈现给..。.此，键..址再按下回.的全过程就结束了。上述过程描述的是.种简单的请求-响应全过程，真实的请求-响应情况可能要.上.描述的过程复杂很多。 HTTP请求特征 从上.整个过程中我们可以总结出HTTP进.分组传输是具有以下特征

.持客.-服务器模式简单快速：客.向服务器请求服务时，只需传送请求.法和路径。请求.法常.的有 GET、HEAD、POST。每种.法规定了客.与服务器联系的类型不同。由于 HTTP协议简单，使得 HTTP服务器的程序规模.，因.通 信速度很快。灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。.连接：.连接的含义是限制每次连接只处理.个请求。服务器处理完客.的请求，并收到客.的应答后，即 断开连接。采.这种.式可以节省传输时间。.状态：HTTP协议是.状态协议。.状态是指协议对于事务处理没有记忆能.。缺少状态意味着如果后续处理需要前.的信息，则它必须重传，这样可能导致每次连接传送的数据量增.。另...，在服务器不需要先 前信息时它的应答就较快。 详解HTTP报. 我们上.描述了.下 HTTP的请求响应过程，流程.较简单，但是凡事就怕认真，你这.认真，就能拓展出很多东西，.如HTTP报.是什么样的，它的组成格式是什么？下.就来探讨.下 HTTP协议主要由三.部分组成：

起始.（startline）：描述请求或响应的基本信息；头部字段（header）：使.key-value形式更详细地说明报.；消息正.（entity）：实际传输的数据，它不.定是纯.本，可以是图.、视频等.进制数据。 其中起始.和头部字段并成为请求头或者响应头，统称为Header；消息正.也叫做实体，称为body。HTTP协议规定每次发送的报.必须要有 Header，但是可以没有 body，也就是说头信息是必须的，实体信息可以没有。.且在header和body之间必须要有.个空.（CRLF），如果..幅图来表..下的话，我觉得应该是下.这样

我们使.上.的那个例.来看.下http的请求报.

如图，这是 http://www.someSchool.edu/someDepartment/home.index请求的请求头，通过观察这个 HTTP报.我们就能够学到很多东西，.先，我们看到报.是.普通 ASCII.本书写的，这样保证.能够可以看懂。然后，我们可以看到每..和下..之间都会有换.，.且最后..（请求头部后）再加上.个回.换.符。 每个报.的起始.都是由三个字段组成：.法、URL字段和HTTP版本字段。

HTTP请求.法 HTTP请求.法.般分为8种，它们分别是

GET获取资源，GET.法.来请求访问已被URI识别的资源。指定的资源经服务器端解析后返回响应内容。也就是说，如果请求的资源是.本，那就保持原样返回； POST传输实体，虽然GET.法也可以传输主体信息，但是便于区分，我们.般不.GET传输实体信息，反. 使.POST传输实体信息， PUT传输.件，PUT .法.来传输.件。就像 FTP协议的.件上传.样，要求在请求报.的主体中包含.件 内容，然后保存到请求URI指定的位置。 但是，鉴于 HTTP的 PUT .法..不带验证机制，任何.都可以上传.件 ,存在安全性问题，因此.般的 Web .站不使.该.法。若配合 W eb应.程序的验证机制，或架构设计采.REST（REpresentational State Transfer，表征状态转移）标准的同类Web.站，就可能会开放使.PUT.法。

HEAD获得响应.部，HEAD.法和GET.法.样，只是不返回报.主体部分。.于确认URI的有效性及资源更新的.期时间等。DELETE删除.件，DELETE.法.来删除.件，是与PUT相反的.法。DELETE.法按请求URI删除指定的 资源。 OPTIONS询问.持的.法，OPTIONS.法.来查询针对请求URI指定的资源.持的.法。 TRACE追踪路径，TRACE.法是让Web服务器端将之前的请求通信环回给客.端的.法。 CONNECT要求.隧道协议连接代理，CONNECT .法要求在与代理服务器通信时建.隧道，实现.隧道协议进.TCP通信。主要使.SSL（SecureSocketsLayer，安全套接层）和TLS（TransportLayerSecurity，传输层安全）协议把通信内容加密后经.络隧道传输。 我们.般最常.的.法也就是GET.法和POST.法，其他.法暂时了解即可。下.是HTTP1.0和HTTP1.1.持的.法清单

HTTP请求URL HTTP协议使. URI定位互联.上的资源。正是因为 URI的特定功能，在互联.上任意位置的资源都能访问到。URL带有请求对象的标识符。在上.的例.中，浏览器正在请求对象/somedir/page.html的资源。 我们再通过.个完整的域名解析.下URL .如http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument这个URL.较繁琐了吧，你把这个URL搞懂了其他的URL也就不成问题了。 .先出场的是http

http://告诉浏览器使.何种协议。对于.部分 Web资源，通常使. HTTP协议或其安全版本，HTTPS协议。另外，浏览器也知道如何处理其他协议。例如， mailto:协议指.浏览器打开邮件客.端；ftp:协议指.浏览器处理.件传输。 第.个出场的是主机

www.example.com既是.个域名，也代表管理该域名的机构。它指.了需要向.络上的哪.台主机发起请求。当然，也可以直接向主机的IPaddress地址发起请求。但直接使.IP地址的场景并不常.。 第三个出场的是端.

我们前.说到，两个主机之间要发起 TCP连接需要两个条件，主机 +端.。它表..于访问 Web服务器上资源的 ...。如果访问的该 Web服务器使.HTTP协议的标准端.（HTTP为80，HTTPS为443）授予对其资源的访问权限，则通常省略此部分。否则端.就是UR I必须的部分。上.是请求URL所必须包含的部分，下.就是URL具体请求资源路径第四个出场的是路径

/path/to/myfile.html是Web服务器上资源的路径。以端.后.的第.个/开始，到?号之前结束，中间的每.个/都代表了层级（上下级）关系。这个URL的请求资源是.个htm l...。 紧跟着路径后.的是查询参数

?key1=value1&key2=value2是提供给Web服务器的额外参数。如果是GET请求，.般带有请求URL参数，如果是 POST请求，则不会在路径后.直接加参数。这些参数是. &符号分隔的键/值对列表。key1 = value1是第.对，key2 =value2是第.对参数 紧跟着参数的是锚点

#SomewhereInTheDocument是资源本.的某.部分的.个锚点。锚点代表资源内的.种"书签"，它给予浏览器显.位于该"加书签"点的内容的指.。例如，在HTML.档上，浏览器将滚动到定义锚点的那个点上；在视频或.频.档上，浏览器将转到锚点代表的那个时间。值得注意的是 #号后.的部分，也称为.段标识符，永远不会与请求.起发送到服务器。 HTTP版本 表.报.使.的HTTP协议版本。 请求头部 这部分内容只是.致介绍.下，内容较多，后.会再以.篇.章详述 在表述完了起始.之后我们再来看.下请求头部，现在我们向上找，找到http://www.someSchool.edu/someDepartment/home.index，来看.下它的请求头部 Host:www.someschool.edu Connection:close User-agent:Mozilla/5.0 Accept-language:fr 这个请求头信息.较少，.先Host表.的是对象所在的主机。你也许认为这个Host是不需要的，因为URL不是已经指明了请求对象的路径了吗？这个.部.提供的信息是 Web代理.速缓存所需要的。Connection:close表.的是浏览器需要告诉服务器使.的是.持久连接。它要求服务器在发送完响应的对象后就关闭连接。User-agent:这是请求头.来告诉Web服务器，浏览器使.的类型是 Mozilla/5.0，即Firefox浏览器。Accept-language告诉Web服务器，浏览器想要得到对象的法语版本，前提是服务器需要.持法语类型，否则将会发送服务器的默认版本。下.我们针对主要的实体字段进.介绍（具体的可以参考https://developer.mozilla.org/zh-CN/docs/Web/HTTP/HeadersMDN官.学习） HTTP的请求标头分为四种：通.标头、请求标头、响应标头和实体标头，依次来进.详解。 通.标头 通.标头主要有三个，分别是Date、Cache-Control和Connection Date Date是.个通.标头，它可以出现在请求标头和响应标头中，它的基本表.如下Date:Wed,21Oct201507:28:00GMT 表.的是格林威治标准时间，这个时间要.北京时间慢.个.时

Cache-Control Cache-Control是.个通.标头，他可以出现在请求标头和响应标头中，Cache-Control的种类.较多，虽然说这是.个通.标头，但是有.些特性是请求标头具有的，有.些是响应标头才有的。主要.类有可缓存性、阈值性、重新验证并重新加载和其他特性 可缓存性是唯.响应标头才具有的特性，我们会在响应标头中详述。 阈值性，这个我翻译可能不准确，它的原英.是 Expiration，我是根据它的值来翻译的，你看到这些值可能会觉得我翻译的有点道理

max-age:资源被认为仍然有效的最.时间，与 Expires不同，这个请求是相对于request标头的时间，. Expires是相对于响应标头。（请求标头）s-maxage:重写了max-age和Expires请求头，仅仅适.于共享缓存，被私有缓存所忽略（这块不理

解，看完响应头的Cache-Control再进.理解）（请求标头） max-stale：表.客.端将接受的最.响应时间，以秒为单位。（响应标头） min-fresh:表.客.端希望响应在指定的最.时间内有效。（响应标头） Connection Connection决定当前事务（.次三次握.和四次挥.）完成后，是否会关闭.络连接。Connection有两种，.种是持久性连接，即.次事务完成后不关闭.络连接 Connection:keep-alive 另.种是.持久性连接，即.次事务完成后关闭.络连接 Connection:close HTTP1.1其他通.标头如下

实体标头 实体标头是描述消息正.内容的 HTTP标头。实体标头.于 HTTP请求和响应中。头部Content-Length、Content-Language、Content-Encoding是实体头。

Content-Length实体报头指.实体主体的..，以字节为单位，发送到接收.。 Content-Language实体报头描述了客.端或者服务端能够接受的语.，例如

Content-Language:de-DEContent-Language:en-USContent-Language:de-DE,en-CA

Content-Encoding这.是.个.较.烦的属性，这个实体报头.来压缩媒体类型。Content-Encoding指.对实体应.了何种编码。常.的内容编码有这.种：gzip、compress、deflate、identity，这个属性可以应.在请求报.和响应报 .中 Accept-Encoding:gzip,deflate//请求头 Content-Encoding:gzip //响应头 下.是.些实体标头字段

请求标头 上.给出的例.请求报.的属性.较少，下.给出.个MDN官.的例. GET/home.htmlHTTP/1.1 Host:developer.mozilla.orgUser-Agent:Mozilla/5.0(Macintosh;IntelMacOSX10.9;rv:50.0)Gecko/20100101Firefox/50.0 Accept:text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8 Accept-Language:en-US,en;q=0.5Accept-Encoding:gzip,deflate,br Referer:https://developer.mozilla.org/testpage.html Connection:keep-alive Upgrade-Insecure-Requests:1 If-Modified-Since:Mon,18Jul201602:36:04GMT If-None-Match:"c561c68d0ba92bbeb8b0fff2a9199f722e3a621a" Cache-Control:max-age=0 Host Host请求头指明了服务器的域名（对于虚拟主机来说），以及（可选的）服务器监听的TCP端.号。如果没有给定端.号，会.动使.被请求服务的默认端.（.如请求.个HTTP的URL会.动使.80作为端.）。 Host:developer.mozilla.org 上.的Accpet、Accept-Language、Accept-Encoding都是属于内容协商的请求标头，我们会在下.说明 Referer HTTPReferer属性是请求标头的.部分，当浏览器向web服务器发送请求的时候，.般会带上Referer，告诉服务器该..是从哪个..链接过来的，服务器因此可以获得.些信息.于处理。Referer:https://developer.mozilla.org/testpage.html Upgrade-Insecure-Requests Upgrade-Insecure-Requests是.个请求标头，.来向服务器端发送信号，表.客.端优先选择加密及带有.份验证的响应。 Upgrade-Insecure-Requests:1 If-Modified-Since HTTP的If-Modified-Since使其成为条件请求：

返回200，只有在给定.期的最后.次修改资源后，服务器才会以200状态发送回请求的资源。

如果请求从开始以来没有被修改过，响应会返回304并且没有任何响应体 If-Modi.ed-Since通常会与 If-None-Match搭配使.，If-Modi.ed-Since .于确认代理或客.端拥有的本地资源的有效性。获取资源的更新.期时间，可通过确认.部字段 Last-Modified来确定。 ..话说就是如果在Last-Modified之后更新了服务器资源，那么服务器会响应200，如果在 Last-Modified之后没有更新过资源，则返回304。 If-Modified-Since:Mon,18Jul201602:36:04GMT If-None-Match If-None-Match HTTP请求标头使请求成为条件请求。对于 GET和 HEAD .法，仅当服务器没有与给定资源匹配的ETag时，服务器才会以200状态发送回请求的资源。对于其他.法，仅当最终现有资源的ETag与列出的任何值都不匹配时，才会处理请求。 If-None-Match:"c561c68d0ba92bbeb8b0fff2a9199f722e3a621a" ETag属于响应标头，后.进.介绍。 内容协商 内容协商机制是指客.端和服务器端就响应的资源内容进.交涉，然后提供给客.端最为适合的资源。内容协商会以响应资源的语.、字符集、编码.式等作为判断的标准。

内容协商主要有以下3种类型：

服务器驱动协商（Server-drivenNegotiation） 这种协商.式是由服务器端进.内容协商。服务器端会根据请求.部字段进..动处理

客.端驱动协商（Agent-drivenNegotiation） 这种协商.式是由客.端来进.内容协商。

透明协商（TransparentNegotiation） 是服务器驱动和客.端驱动的结合体，是由服务器端和客.端各.进.内容协商的.种.法。 内容协商的分类有很多种，主要的.种类型是Accept、Accept-Charset、Accept-Encoding、Accept-Language、Content-Language。 Accept 接受请求HTTP标头会通告客.端其能够理解的MIME类型 那么什么是MIME类型呢？在回答这个问题前你应该先了解.下什么是MIME

MIME: MIME (Multipurpose Internet Mail Extensions)是描述消息内容类型的因特.标准。MIME消息能包含.本、图 像、.频、视频以及其他应.程序专.的数据。 也就是说，MIME类型其实就是.系列消息内容类型的集合。那么MIME类型都有哪些呢？ .本.件：text/html、text/plain、text/css、application/xhtml+xml、application/xml 图..件：image/jpeg、image/gif、image/png 视频.件：video/mpeg、video/quicktime 应.程序.进制.件：application/octet-stream、application/zip .如，如果浏览器不.持 PNG图.的显.，那 Accept就不指定image/png，.指定可处理的 image/gif和image/jpeg等图.类型。 .般MIME类型也会和q这个属性.起使.，q是什么？q表.的是权重，来看.个例.Accept:text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8 这是什么意思呢？若想要给显.的媒体类型增加优先级，则使.q=来额外表.权重值，没有显.权重的时候默认值是1.0，我给你列个表格你就明.了

也就是说，这是.个放置顺序，权重.的在前，低的在后，application/xml;q=0.9是不可分割的整体。 Accept-Charset accept-charset属性规定服务器处理表单数据所接受的字符集。accept-charset属性允许您指定.系列字符集，服务器必须.持这些字符集，从.得以正确解释表单中的数据。该属性的值是.引号包含字符集名称列表。如果可接受字符集与..所使.的字符即不相匹配的话，浏览器可以选 择忽略表单或是将该表单区别对待。此属性的默认值是unknown，表.表单的字符集与包含表单的.档的字符集相同。常.的字符集有：UTF-8-Unicode字符编码；ISO-8859-1-拉丁字.表的字符编码 Accept-Language .部字段Accept-Language.来告知服务器..代理能够处理的.然语.集（指中.或英.等），以及.然语.集的相对优先级。可.次指定多种.然语.集。和Accept.部字段.样，按权重值q来表.相对优先级。 Accept-Language:en-US,en;q=0.5 请求标头我们.概就介绍这.种，后.会有.篇.章详细深挖所有的响应头的，下.是.个响应头的汇总，基于HTTP1.1

响应标头 响应标头是可以在 HTTP响应种使.的 HTTP标头，这听起来是像.句废话，不过确实是这样解释。并不是所有出现在响应中的标头都是响应标头。还有.些特殊的我们上.说过，有通.标头和实体标头也会出现在响应标头中，.如Content-Length就是.个实体标头，但是，在这种情况下，这些实体请求通常称为响应头。下.以.个例.为例和你探讨.下响应头 200OK Access-Control-Allow-Origin:* Connection:Keep-Alive Content-Encoding:gzipContent-Type:text/html;charset=utf-8 Date:Mon,18Jul201616:06:00GMT Etag:"c561c68d0ba92bbeb8b0f612a9199f722e3a621a" Keep-Alive:timeout=5,max=997 Last-Modified:Mon,18Jul201602:36:04GMT Server:ApacheSet-Cookie:mykey=myvalue;expires=Mon,17-Jul-201716:06:00GMT;Max-Age=31449600;Path=/;secure Transfer-Encoding:chunked Vary:Cookie,Accept-Encodingx-frame-options:DENY 响应状态码 .先出现的应该就是200OK，这是HTTP响应标头的状态码，它表.着响应成功完成。HTTP响应标头的状态码有很多，并做了如下规定 以2xx为开头的都表.请求成功响应。

以3xx为开头的都表.需要进.附加操作以完成请求

以4xx的响应结果表明客.端是发.错误的原因所在。以5xx为开头的响应标头都表.服务器本.发.错误

Access-Control-Allow-Origin .个返回的HTTP标头可能会具有Access-Control-Allow-Origin，Access-Control-Allow-Origin指定.个来源，它告诉浏览器允许该来源进.资源访问。否则-对于没有凭据的请求*通配符，告诉浏览器允许任何源访问资源。例如，要允许源https://mozilla.org的代码访问资源，可以指定： Access-Control-Allow-Origin:https://mozilla.orgVary:Origin 如果服务器指定单个来源.不是*通配符的话，则服务器还应在Vary响应标头中包含Origin，以向客.端指.服务器响应将根据原始请求标头的值.有所不同。 Keep-Alive 上.我们提到，HTTP报.标头会分为四种，这其实是按着上下.来分类的还有.种分类是根据代理进.分类，根据代理会分为端到端头和逐跳标头 .Keep-Alive表.的是Connection.持续连接的存活时间，如下 Connection:Keep-Alive Keep-Alive:timeout=5,max=997 Keep-Alive有两个参数，它们是以逗号分隔的参数列表，每个参数由.个标识符和.个由等号=分隔的值组成。 timeout：指.空闲连接必须保持打开状态的最短时间（以秒为单位）。max：指.在关闭连接之前可以在此连接上发送的最.请求数。上述HTTP代码的意思就是限制最.的超时时间是5s和最.的连接请求是997个。 Server 服务器标头包含有关原始服务器.来处理请求的软件的信息。 应该避免使.过于冗.和详细的 Server值，因为它们可能会泄露内部实施细节，这可能会使攻击者容易地发现并利.已知的安全漏洞。例如下.这种写法 Server:Apache/2.4.1(Unix) Set-Cookie Cookie.是另外.个领域的内容了，我们后..章会说道 Cookie，这.需要记住 Cookie、Set-Cookie和 Content-Disposition等在其他 RFC中定义的.部字段，它们不是属于 HTTP 1.1的.部字段，但是使.率仍然很.。 Transfer-Encoding .部字段Transfer-Encoding规定了传输报.主体时采.的编码.式。 Transfer-Encoding:chunked HTTP/1.1的传输编码.式仅对分块传输编码有效。 X-Frame-Options HTTP.部字段是可以..扩展的。所以在Web服务器和浏览器的应.上，会出现各种.标准的.部字段。 .部字段X-Frame-Options属于HTTP响应.部，.于控制.站内容在其他 Web .站的 Frame标签内的显.问题。其主要.的是为了防.点击劫持（clickjacking）攻击。下.是.个响应头的汇总，基于HTTP1.1

.HTTP/1.1.部字段 在 HTTP协议通信交互中使.到的.部字段，不限于 RFC2616中定义的 47种.部字段。还有 Cookie、Set-Cookie和 Content-Disposition等在其他 RFC中定义的.部字段，它们的使.频率也很.。这些.正式的.部字段统.归纳在 RFC4229HTTPHeaderFieldRegistrations中。 End-to-end.部和Hop-by-hop.部 HTTP.部字段将定义成缓存代理和.缓存代理的.为，分成2种类型。.种是End-to-end.部和Hop-by-hop.部 End-to-end（端到端）.部 这些标头必须发送给消息的最终接收者:请求的服务器，或响应的客.端。中间代理必须重新传输未经修改的标头，并且缓存必须存储这些信息 Hop-by-hop（逐跳）.部 分在此类别中的.部只对单次转发有效，会因通过缓存或代理.不再转发。 下.列举了HTTP/1.1中的逐跳.部字段。除这8个.部字段之外，其他所有字段都属于端到端.部。 Connection、Keep-Alive、Proxy-Authenticate、Proxy-Authorization、Trailer、TE、Transfer-Encoding、Upgrade HTTP的优点和缺点 HTTP的优点 简单灵活易扩展 HTTP最重要也是最突出的优点是简单、灵活、易于扩展。 HTTP的协议.较简单，它的主要组成就是header + body，头部信息也是简单的.本格式，.且 HTTP的请求报.根据英.也能猜出来个.概的意思，降低学习.槛，能够让更多的.研究和开发HTTP应.。 所以，在简单的基础上，HTTP协议.多了灵活和易扩展的优点。 HTTP协议.的请求.法、URI、状态码、原因短语、头字段等每.个核.组成要素都没有被制定死，允许开发者任意定制、扩充或解释，给予了浏览器和服务器最.程度的信任和.由。 应..泛、环境成熟 因为过于简单，普及，因此应.很.泛。因为 HTTP协议本.不属于.种语.，它并不限定某种编程语.或者操作系统，所以天然具有跨语.、跨平台的优越性。.且，因为本.的简单特性很容易实现，所以.乎所有的编程语.都有HTTP调.库和外围的开发测试.具。 随着移动互联.的发展，HTTP的触.已经延伸到了世界的每.个.落，从简单的Web..到复杂的JSON、XML数据，从台式机上的浏览器到.机上的各种 APP、新闻、论坛、购物、.机游戏，你很难找到.个没有使. HTTP的地.。 .状态 .状态其实既是优点.是缺点。因为服务器没有记忆能.，所以就不需要额外的资源来记录状态信息，不仅实现上会简单.些，.且还能减轻服务器的负担，能够把更多的CPU和内存.来对外提供服务。 HTTP的缺点 .状态 既然服务器没有记忆能.，它就.法.持需要连续多个步骤的事务操作。每次都得问.遍.份信息，不仅.烦，.且还增加了不必要的数据传输量。由此出现了Cookie技术。 明. HTTP协议.还有.把优缺点.体的双刃剑，就是明.传输。明.意思就是协议.的报.（准确地说是 header部分）不使..进制数据，.是.简单可阅读的.本形式。 对. TCP、UDP这样的.进制协议，它的优点显.易.，不需要借助任何外部.具，.浏览器、Wireshark或者tcpdump抓包后，直接..眼就可以很容易地查看或者修改，为我们的开发调试.作带来极.的便利。 当然缺点也是显.易.的，就是不安全，可以被监听和被窥探。因为.法判断通信双.的.份，不能判断报.是否被更改过。 性能 HTTP的性能不算差，但不完全适应现在的互联.，还有很.的提升空间。 来源 作者：cxuan，公众号：Java建设者 参考资料：
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Technical_overview 《极客时间》-透视HTTP协议
https://developer.mozilla.org/en-US/docs/Web/HTTP https://baike.baidu.com/item/WEB服务器/8390210?fr=aladdin
https://baike.baidu.com/item/内容分发.络/4034265 https://baike.baidu.com/item/HTML/97049?fr=aladdin
https://www.jianshu.com/p/3dd8f1879acb 《计算机.络-.顶向下.法》 《图解HTTP》 HTTP协议的内容协商
https://www.w3school.com.cn/tags/att_form_accept_charset.asp 有偿投稿：欢迎投稿原创技术博.，.旦采.将给予50元-200元不等稿费，要求个.原创，图.并茂。可以是职场经验、.试经 历，也可是技术教程，学习笔记。投稿请联系微信「web527zsd」，备注投稿。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，欢迎添加.编微信「focusoncode」，每.朋友圈更新.篇.质量技术博.（..告）。 ↓扫描.维码添加.编↓

推荐阅读 1.快速建站利器！ 2.厉害！SpringCloud20000字总结！

3.

终于可以告别单调的Swagger了

4.

答应我，别再写上千.的类了好吗

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 硬核！16000字Redis.试知识点总结，建议收藏！ 坚持就是胜利Java后端 3.2.

作者|坚持就是胜利链接|juejin.im/post/5dccf260f265da0bf66b626d 今天，我不.量.的.试了某..的 Java开发岗位，迎..来.位.尘仆仆的中年男.，..拿着屏幕还亮着的 Mac。他冲着我礼貌的笑了笑，然后说了句"不好意思，让你久等了"，然后.意我坐下，说："我们开始吧，看了你的简历，觉得你对 Redis应该掌握的不错，我们今天就来讨论下 Redis……"。我想："来就来，兵来将挡.来.掩"。 Redis是什么 .试官：你先来说下 Redis是什么吧！ 我：（这不就是总结下 Redis的定义和特点嘛）Redis是 C语.开发的.个开源的（遵从 BSD协议）.性能键值对（key-value）的内存数据库，可以.作数据库、缓存、消息中间件等。 它是.种 NoSQL（not-onlysql，泛指.关系型数据库）的数据库。 我顿了.下，接着说，Redis作为.个内存数据库：

性能优秀，数据在内存中，读写速度.常快，.持并发 10WQPS。单进程单线程，是线程安全的，采. IO多路复.机制。 丰富的数据类型，.持字符串（strings）、散列（hashes）、列表（lists）、集合（sets）、有序集合（sortedsets）等。 .持数据持久化。 可以将内存中数据保存在磁盘中，重启时加载。 主从复制，哨兵，.可.。 可以.作分布式锁。 可以作为消息中间件使.，.持发布订阅。 五种数据类型 .试官：总结的不错，看来是早有准备啊。刚来听你提到 Redis.持五种数据类型，那你能简单说下这五种数据类型吗？ 我：当然可以，但是在说之前，我觉得有必要先来了解下 Redis内部内存管理是如何描述这5种数据类型的。 说着，我拿着笔给.试官画了.张图：我：.先Redis内部使..个redisObject对象来表.所有的key和value。

redisObject最主要的信息如上图所.：type表..个 value对象具体是何种数据类型，encoding是不同数据类型在 Redis内部的存储.式。.如：type=string表.value存储的是.个普通字符串，那么encoding可以是 raw或者 int。 我顿了.下，接着说，下.我简单说下 5种数据类型： ①String是 Redis最基本的类型，可以理解成与 Memcached.模.样的类型，.个 Key对应.个 Value。Value不仅 是String，也可以是数字。 String类型是.进制安全的，意思是 Redis的 String类型可以包含任何数据，.如 jpg图.或者序列化的对象。String类型的值最.能存储 512M。 ②Hash是.个键值（key-value）的集合。Redis的 Hash是.个 String的 Key和 Value的映射表，Hash特别适合存储 对象。常.命令：hget，hset，hgetall等。 ③List列表是简单的字符串列表，按照插.顺序排序。可以添加.个元素到列表的头部（左边）或者尾部（右边）常.命令： lpush、rpush、lpop、rpop、lrange（获取列表.段）等。 应.场景：List应.场景.常多，也是 Redis最重要的数据结构之.，.如 Twitter的关注列表，粉丝列表都可以. List 结构来实现。数据结构：List就是链表，可以.来当消息队列.。Redis提供了 List的 Push和 Pop操作，还提供了操作某.段的 API，可以直接查询或者删除某.段的元素。 实现.式：Redis List的是实现是.个双向链表，既可以.持反向查找和遍历，更.便操作，不过带来了额外的内存开销。 ④Set是 String类型的.序集合。集合是通过 hashtable实现的。Set中的元素是没有顺序的，.且是没有重复的。常.命令：sdd、spop、smembers、sunion等。应.场景：Redis Set对外提供的功能和 List .样是.个列表，特殊之处在于 Set是.动去重的，.且 Set提供了判断某 个成员是否在.个 Set集合中。⑤Zset和 Set.样是String类型元素的集合，且不允许重复的元素。常.命令：zadd、zrange、zrem、zcard等。使.场景：SortedSet可以通过..额外提供.个优先级（score）的参数来为成员排序，并且是插.有序的，即.动排序。当你需要.个有序的并且不重复的集合列表，那么可以选择SortedSet结构。 和 Set相.，Sorted Set关联了.个 Double类型权重的参数 Score，使得集合中的元素能够按照 Score进.有序排列，Redis正是通过分数来为集合中的成员进.从.到.的排序。 实现.式：Redis Sorted Set的内部使. HashMap和跳跃表（skipList）来保证数据的存储和有序，HashMap .放的是成员到Score的映射。 .跳跃表.存放的是所有的成员，排序依据是 HashMap.存的Score，使.跳跃表的结构可以获得.较.的查找效率，并且在实现上.较简单。 数据类型应.场景总结：

.试官：想不到你平时也下了不少.夫，那 Redis缓存你.定.过的吧？我：.过的。.试官：那你跟我说下你是怎么.的？我是结合 Spring Boot使.的。.般有两种.式，.种是直接通过 RedisTemplate来使.，另.种是使. Spring Cache集成 Redis（也就是注解的.式）。 Redis缓存直接通过 RedisTemplate来使.，使. SpringCache集成 Redispom.xml中加.以下依赖： org.springframework.boot spring-boot-starter-data-redis org.apache.commons commons-pool2 org.springframework.boot spring-boot-starter-web org.springframework.session spring-session-data-redis org.projectlombok lombok true org.springframework.boot spring-boot-starter-test test spring-boot-starter-data-redis：在 SpringBoot2.x以后底层不再使. Jedis，.是换成了Lettuce。 commons-pool2：.作 Redis连接池，如不引.启动会报错。 spring-session-data-redis：SpringSession引.，.作共享 Session。 配置.件 application.yml的配置： server:port:8082servlet: session: timeout:30ms spring:cache: type:redis redis:host:127.0.0.1 port:6379password: #redis默认情况下有16个分.，这.配置具体使.的分.，默认为0 database:0lettuce: pool: #连接池最.连接数(使.负数表.没有限制),默认8 max-active:100 创建实体类 User.java：

publicclassUserimplementsSerializable{ privatestaticfinallongserialVersionUID=662692455422902539L; privateIntegerid; privateStringname; privateIntegerage; publicUser(){ } publicUser(Integerid,Stringname,Integerage) {this.id=id;this.name=name;this.age=age; } publicIntegergetId(){returnid;} publicvoidsetId(Integerid){this.id=id;} publicStringgetName(){returnname;} publicvoidsetName(Stringname){this.name=name;} publicIntegergetAge() {returnage;} publicvoidsetAge(Integerage){this.age=age;} @Override publicStringtoString(){ return"User{"+"id="+id+",name='"+name+'''+",age="+age+'}';} } RedisTemplate的使..式 默认情况下的模板只能.持RedisTemplate<String,String>，也就是只能存.字符串，所以.定义模板很有必要。 添加配置类 RedisCacheCon.g.java：@Configuration@AutoConfigureAfter(RedisAutoConfiguration.class)publicclassRedisCacheConfig{ @BeanpublicRedisTemplate<String,Serializable>redisCacheTemplate(LettuceConnectionFactoryconnectionFactory){ RedisTemplate<String,Serializable>template=newRedisTemplate<> ();template.setKeySerializer(newStringRedisSerializer());template.setValueSerializer(newGenericJackson2JsonRedisSerializer());template.setConnectionFactory(connectionFactory);returntemplate; }} 测试类：

@RestController@RequestMapping("/user") publicclassUserController{ publicstaticLoggerlogger=LogManager.getLogger(UserController.class); @Autowired privateStringRedisTemplatestringRedisTemplate; @Autowired privateRedisTemplate<String,Serializable>redisCacheTemplate; @RequestMapping("/test") publicvoidtest() {redisCacheTemplate.opsForValue().set("userkey",newUser(1,"张三",25));Useruser= (User)redisCacheTemplate.opsForValue().get("userkey"); } logger.info("当前获取对象：{}",user.toString()); 然后在浏览器访问，观察后台.志 http:// localhost:8082/user/test 使.SpringCache集成Redis Spring Cache具备很好的灵活性，不仅能够使.SPEL（spring expression language）来定义缓存的 Key和各种 Condition，还提供了开箱即.的缓存临时存储.案，也.持和主流的专业缓存如EhCache、Redis、Guava的集成。 定义接. UserService.java： publicinterfaceUserService{ Usersave(Useruser); voiddelete(intid); Userget(Integerid);}

接.实现类 UserServiceImpl.java： @Service publicclassUserServiceImplimplementsUserService{ publicstaticLoggerlogger=LogManager.getLogger(UserServiceImpl.class); privatestaticMap<Integer,User>userMap=newHashMap<>(); static{userMap.put(1,newUser(1,"肖战",25));userMap.put(2,newUser(2,"王.博",26)); } userMap.put(3,newUser(3,"杨紫",24)); @CachePut(value="user",key="#user.id") @Override publicUsersave(Useruser){ userMap.put(user.getId(),user); ogger.("进.get.法，当前获取对象：{}",userMap.get( ) :userMap.get(d).toStrng());returnuserMap.get(id); logger.info("进.save.法，当前存储对象： returnuser;} {}",user.toStr ing()); @CacheEvict(value="user",key="#id")@Override publicvoiddelete(intid){userMap.remove(id); } logger.info("进.delete.法，删除成功"); @Cacheable(value="user",key="#id")@Override publicUserget(Integerid){l info id ==null?null i i

}} 为了.便演.数据库的操作，这.直接定义了.个 Map<Integer,User>userMap。 这.的核.是三个注解： @Cachable @CachePut @CacheEvict 测试类：UserController @RestController@RequestMapping("/user")publicclassUserController{ publicstaticLoggerlogger=LogManager.getLogger(UserController.class); @Autowired privateStringRedisTemplatestringRedisTemplate; @Autowired privateRedisTemplate<String,Serializable>redisCacheTemplate; @Autowired privateUserServiceuserService; @RequestMapping("/test") publicvoidtest(){redisCacheTemplate.opsForValue().set("userkey",newUser(1,"张三" ,25)); Useruser=(User)redisCacheTemplate.opsForValue().get("userkey"); } logger.info("当前获取对象： {}",user.toString()); @RequestMapping("/add") publicvoidadd(){Useruser=userService.save(newUser(4,"李现" ,30)); } logger.info("添加的..信息：{}" ,user.toString()); @RequestMapping("/delete") publicvoiddelete(){ userService.delete(4);} @RequestMapping("/get/{id}") publicvoidget(@PathVariable("id")StringidStr)throwsException{if(StringUtils.isBlank(idStr)){ } thrownewException("id为空"); Integerid=Integer.parseInt(idStr);Useruser=userService.get(id); } logger.info("获取的..信息：{}" ,user.toString()); } .缓存要注意，启动类要加上.个注解开启缓存： @SpringBootApplication(exclude=DataSourceAutoConfiguration.class) @EnableCaching publicclassApplication{ publicstaticvoidmain(String[]args){ SpringApplication.run(Application.class,args);} } ①先调.添加接.：http:// localhost:8082/user/add ②再调.查询接.，查询 id=4的..信息： 可以看出，这.已经从缓存中获取数据了，因为上.步 add.法已经把 id=4的..数据放.了Red is缓存3、调.删除. 法，删除 id=4的..信息，同时清除缓存：

④再次调.查询接.，查询 id=4的..信息：没有了缓存，所以进.了 get.法，从 userMap中获取。

缓存注解 ①@Cacheable 根据.法的请求参数对其结果进.缓存：Key：缓存的 Key，可以为空，如果指定要按照 SPEL表达式编写，如果不指定，则按照.法的所有参数进.组合。 Value：缓存的名称，必须指定.少.个（如 @Cacheable (value='user')或者 @Cacheable(value= {'user1','user2'})）Condition：缓存的条件，可以为空，使. SPEL编写，返回 true或者 false，只有为true才进.缓存。

②@CachePut 根据.法的请求参数对其结果进.缓存，和 @Cacheable不同的是，它每次都会触发真实.法的调.。参数描述.上。 ③@CacheEvict 根据条件对缓存进.清空：

Key：同上。 Value：同上。 Condition：同上。 allEntries：是否清空所有缓存内容，缺省为 false，如果指定为true，则.法调.后将.即清空所有缓存。 beforeInvocation：是否在.法执.前就清空，缺省为 false，如果指定为 true，则在.法还没有执.的时候就清空缓存。缺省情况下，如果.法执.抛出异常，则不会清空缓存。 缓存问题.试官：看了.下你的

Demo，简单易懂。那你在实际项.中使.缓存有遇到什么问题或者会遇到什么问题你知道吗？我：缓存和数据库数据.致性问题：分布式环境下.常容易出现缓存和数据库间数据.致性问题，针对这.点，如果项.对缓存的要求是强.致性的，那么就不要使.缓存。我们只能采取合适的策略来降低缓存和数据库间数据不.致的概率，..法保证两者间的强.致性。合适的策略包括合适的缓存更新策略，更新数据库后及时更新缓存、缓存失败时增加重试机制。.试官：Redis雪崩了解吗？我：我了解的，.前电商..以及热点数据都会去做缓存，.般缓存都是定时任务去刷新，或者查不到之后去更新缓存的，定时任务刷新就有.个问题。举个栗.：如果..所有 Key的失效时间都是 12 .时，中午 12点刷新的，我零点有个.促活动.量..涌.，假设每秒6000个请求，本来缓存可以抗住每秒 5000个请求，但是缓存中所有 Key都失效了。此时6000个/秒的请求全部落在了数据库上，数据库必然扛不住，真实情况可能DBA都没反应过来直接挂了。此时，如果没什么特别的.案来处理，DBA很着急，重启数据库，但是数据库...被新流量给打死了。这就是我理解的缓存雪崩。 我.想：同.时间..积失效，瞬间 Redis跟没有.样，那这个数量级别的请求直接打到数据库.乎是灾难性的。你想想如果挂的是.个..服务的库，那其他依赖他的库所有接..乎都会报错。如果没做熔断等策略基本上就是瞬间挂..的节奏，你怎么重启..都会把你打挂，等你重启好的时候，..早睡觉去了， 临睡之前，骂骂咧咧“什么垃圾产品”。.试官摸摸了..的头发：嗯，还不错，那这种情况你都是怎么应对的？我：处理缓存雪崩简单，在批量往 Redis存数据的时候，把每个 Key的失效时间都加个随机值就好了，这样可以保证数据不 会再同.时间..积失效。 setRedis（key,value,time+Math.random()*10000）; 如果Redis是集群部署，将热点数据均匀分布在不同的 Redis库中也能避免全部失效。 或者设置热点数据永不过期，有更新操作就更新缓存就好了（.如运维更新了..商品，那你刷下缓存就好了，不要设置过期时间），电商..的数据也可以.这个操作，保险。.试官：那你了解缓存穿透和击穿么，可以说说他们跟雪崩的区别吗？我：嗯，了解，先说下缓存穿透吧，缓存穿透是指缓存和数据库中都没有的数据，...（.客）不断发起请求。举个栗.：我们数据库的 id都是从1.增的，如果发起 id=-1的数据或者 id特别.不存在的数据，这样的不断攻击导致数 据库压.很.，严重会击垮数据库。 我.接着说：.于缓存击穿嘛，这个跟缓存雪崩有点像，但是.有.点不.样，缓存雪崩是因为..积的缓存失效，打崩了DB。.缓存击穿不同的是缓存击穿是指.个 Key.常热点，在不停地扛着.量的请求，.并发集中对这.个点进.访问，当这 个Key在失效的瞬间，持续的.并发直接落到了数据库上，就在这个 Key的点上击穿了缓存。.试官露出欣慰的眼光：那他们分别怎么解决？我：缓存穿透我会在接.层增加校验，.如..鉴权，参数做校验，不合法的校验直接return，.如 id做基础校验，id<=0 直接拦截。.试官：那你还有别的.法吗？我：我记得 Redis.还有.个.级.法布隆过滤器（Bloom Filter）这个也能很好的预防缓存穿透的发.。它的原理也很简单，就是利..效的数据结构和算法快速判断出你这个 Key是否在数据库中存在，不存在你 return就好 了，存在你就去查 DB刷新 KV再 return。缓存击穿的话，设置热点数据永不过期，或者加上互斥锁就搞定了。作为暖男，代码给你准备好了，拿.不谢。 publicstaticStringgetData(Stringkey)throwsInterruptedException{ //从Redis查询数据 Stringresult=getDataByKV(key); //参数校验 if(StringUtils.isBlank(result)){try{ //获得锁 if(reenLock.tryLock()){ //去数据库查询 result=getDataByDB(key); //校验 if(StringUtils.isNotBlank(result)){ //插进缓存 setDataToKV(key,result);}}else{ //睡.会再拿 Thread.sleep(100L);result=getData(key);}}finally{ //释放锁 reenLock.unlock(); }}returnresult; } .试官：嗯嗯，还不错。 Redis为何这么快 .试官：Redis作为缓存.家都在.，那 Redis.定很快咯？ 我：当然了，官.提供的数据可以达到 100000+的 QPS（每秒内的查询次数），这个数据不. Memcached差！ .试官：Redis这么快，它的“多线程模型”你了解吗？（露出邪魅.笑） 我：您是想问 Redis这么快，为什么还是单线程的吧。Redis确实是单进程单线程的模型，因为 Redis完全是基于内存的操 作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的..或者.络带宽。 既然单线程容易实现，.且 CPU不会成为瓶颈，那就顺理成章的采.单线程的.案了（毕竟采.多线程会有很多.烦）。 .试官：嗯，是的。那你能说说 Redis是单线程的，为什么还能这么快吗？ 我：可以这么说吧，总结.下有如下四点：

Redis完全基于内存，绝.部分请求是纯粹的内存操作，.常迅速，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是 O(1)。数据结构简单，对数据操作也简单。采.单线程，避免了不必要的上下.切换和竞争条件，不存在多线程导致的 CPU切换，不.去考虑各种锁的问题，不存在加锁释放锁操作，没有死锁问题导致的性能消耗。使.多路复. IO模型，.阻塞 IO。 Redis和Memcached的区别 .试官：嗯嗯，说的很详细。那你为什么选择 Redis的缓存.案.不. Memcached呢？ 我：原因有如下四点：

存储.式上：Memcache会把数据全部存在内存之中，断电后会挂掉，数据不能超过内存...。Redis有部分数据存在硬盘上，这样能保证数据的持久性。数据.持类型上：Memcache对数据类型的.持简单，只.持简单的 key-value，，. Redis.持五种数据类型。使.底层模型不同：它们之间底层实现.式以及与客.端之间通信的应.协议不.样。Redis直接..构建了VM机制，因为.般的系统调.系统函数的话，会浪费.定的时间去移动和请求。 Value的..：Redis可以达到 1GB，. Memcache只有1MB。 淘汰策略 .试官：那你说说你知道的 Redis的淘汰策略有哪些？ 我：Redis有六种淘汰策略，如下图：

补充.下：Redis 4.0加.了 LFU（least frequency use）淘汰策略，包括 volatile-lfu和 allkeys-lfu，通过统计访问频率，将访问频率最少，即最不经常使.的 KV淘汰。持久化 .试官：你对 Redis的持久化机制了解吗？能讲.下吗？我：Redis为了

保证效率，数据缓存在了内存中，但是会周期性的把更新的数据写.磁盘或者把修改操作写.追加的记录.件中，以保证数据的持久化。 Redis的持久化策略有两种：

RDB：快照形式是直接把内存中的数据保存到.个 dump的.件中，定时保存，保存策略。 AOF：把所有的对 Redis的服务器进.修改的命令都存到.个.件.，命令的集合。Redis默认是快照RDB的持久化.式。

当 Redis重启的时候，它会优先使. AOF .件来还原数据集，因为 AOF .件保存的数据集通常. RDB .件所保存的数据集更完整。你甚.可以关闭持久化功能，让数据只在服务器运.时存。 .试官：那你再说下 RDB是怎么.作的？ 我：默认Redis是会以快照"RDB"的形式将数据持久化到磁盘的.个.进制.件dump.rdb。 .作原理简单说.下：当 Redis需要做持久化时，Redis会 fork .个.进程，.进程将数据写到磁盘上.个临时 RDB . 件中。当.进程完成写临时.件后，将原来的 RDB替换掉，这样的好处是可以copy-on-write。我：RDB的优点是：这种.件.常适合.于备份：.如，你可以在最近的 24 .时内，每.时备份.次，并且在每个.的每. 天也备份.个 RDB.件。这样的话，即使遇上问题，也可以随时将数据集还原到不同的版本。RDB .常适合灾难恢复。RDB的缺点是：如果你需要尽量避免在服务器故障时丢失数据，那么RDB不合适你。.试官：那你要不再说下 AOF？我：（说就.起说下吧）使. AOF做持久化，每.个写命令都通过write函数追加到appendonly.aof中，配置.式如下： appendfsyncyes appendfsyncalways#每次有数据修改发.时都会写.AOF.件。 appendfsynceverysec#每秒钟同步.次，该策略为AOF的缺省策略。 AOF可以做到全程持久化，只需要在配置中开启 appendonly yes。这样Redis每执..个修改数据的命令，都会把它添加到AOF.件中，当 Redis重启时，将会读取 AOF.件进.重放，恢复到Redis关闭前的最后时刻。我顿了.下，继续说：使. AOF的优点是会让 Redis变得.常耐久。可以设置不同的 Fsync策略，AOF的默认策略是每秒钟Fsync.次，在这种配置下，就算发.故障停机，也最多丢失.秒钟的数据。缺点是对于相同的数据集来说，AOF的.件体积通常要.于 RDB .件的体积。根据所使.的 Fsync策略，AOF的速度可能会慢于 RDB。.试官.问：你说了这么多，那我该.哪.个呢？我：如果你.常关.你的数据，但仍然可以承受数分钟内的数据丢失，那么可以额只使.RDB持久。AOF将 Redis执.的每.条命令追加到磁盘中，处理巨.的写.会降低Redis的性能，不知道你是否可以接受。数据库备份和灾难恢复：定时.成 RDB快照.常便于进.数据库备份，并且 RDB恢复数据集的速度也要. AOF恢复的速度快。当然了，Redis .持同时开启 RDB和 AOF，系统重启后，Redis会优先使. AOF来恢复数据，这样丢失的数据会最少。主从复制.试官：Redis单节点存在单点故障问题，为了解决单点问题，.般都需要对 Redis配置从节点，然后使.哨兵来监听主节点的存活状态，如果主节点挂掉，从节点能继续提供缓存功能，你能说说Redis主从复制的过程和原理吗？我有点懵，这个说来就话.了。但幸好提前准备了：主从配置结合哨兵模式能解决单点故障问题，提.Redis可.性。从节点仅提供读操作，主节点提供写操作。对于读多写少的状况，可给主节点配置多个从节点，从.提.响应效率。我顿了.下，接着说：关于复制过程，是这样的：

从节点执. slaveof[masterIP][masterPort]，保存主节点信息。 从节点中的定时任务发现主节点信息，建.和主节点的Socket连接。

从节点发送 Ping信号，主节点返回 Pong，两边能互相通信。连接建.后，主节点将所有数据发送给从节点（数据同步）。主节点把当前的数据同步给从节点后，便完成了复制的建.过程。接下来，主节点就会持续的把写命令发送给从节点，保证主从数据.致性。 .试官：那你能详细说下数据同步的过程吗？ （我.想：这也问的太细了吧）我：可以。Redis 2.8之前使. sync[runId][o.set]同步命令，Redis 2.8之后使. psync[runId][o.set]命令。 两者不同在于，Sync命令仅.持全量复制过程，Psync.持全量和部分复制。 介绍同步之前，先介绍.个概念：

runId：每个 Redis节点启动都会.成唯.的 uuid，每次Redis重启后，runId都会发.变化。o.set：主节点和从节点都各.维护..的主从复制偏移量o.set，当主节点有写.命令时，o.set=o.set+命令的字节.度。从节点在收到主节点发送的命令后，也会增加..的 o.set，并把..的 o.set发送给主节点。这样，主节点同时保存..的 o.set和从节点的o.set，通过对.o.set来判断主从节点数据是否.致。 repl_backlog_size：保存在主节点上的.个固定.度的先进先出队列，默认..是1MB。主节点发送数据给从节点过程中，主节点还会进..些写操作，这时候的数据存储在复制缓冲区中。 从节点同步主节点数据完成后，主节点将缓冲区的数据继续发送给从节点，.于部分复制。 主节点响应写命令时，不但会把命名发送给从节点，还会写.复制积压缓冲区，.于复制命令丢失的数据补救。

上.是Psync的执.流程，从节点发送psync[runId][o.set]命令，主节点有三种响应：

FULLRESYNC：第.次连接，进.全量复制 CONTINUE：进.部分复制ERR：不.持 psync命令，进.全量复制.试官：很好，那你能具体说下全量复制和部分复制的过程吗？ 我：可以！

上.是全量复制的流程。主要有以下.步：

从节点发送 psync ?-1命令（因为第.次发送，不知道主节点的 runId，所以为?，因为是第.次复制，所以 o.set=-1）。主节点发现从节点是第.次复制，返回 FULLRESYNC {runId} {o.set}，run Id是主节点的 run Id，o.set是主节点.前的 o.set。从节点接收主节点信息后，保存到 info中。主节点在发送 FULLRESYNC后，启动 bgsave命令，.成RDB.件（数据持久化）。主节点发送 RDB.件给从节点。到从节点加载数据完成这段期间主节点的写命令放.缓冲区。从节点清理..的数据库数据。从节点加载 RDB.件，将数据保存到..的数据库中。如果从节点开启了AOF，从节点会异步重写AOF .件。关于部分复制有以下.点说明： ①部分复制主要是 Red is针对全量复制的过.开销做出的.种优化措施，使. psync[run Id] [o.set]命令实现。 当从节点正在复制主节点时，如果出现.络闪断或者命令丢失等异常情况时，从节点会向主节点要求补发丢失的命令数据，主节点的复制积压缓冲区将这部分数据直接发送给从节点。这样就可以保持主从节点复制的.致性。补发的这部分数据.般远远.于全量数据。②主从连接中断期间主节点依然响应命令，但因复制连接中断命令.法发送给从节点，不过主节点内的复制积压缓冲区依 然可以保存最近.段时间的写命令数据。 ③当主从连接恢复后，由于从节点之前保存了..已复制的偏移量和主节点的运. ID。因此会把它们当做 psync参数发送给主节点，要求进.部分复制。④主节点接收到 psync命令后.先核对参数run Id是否与...致，如果.致，说明之前复制的是当前主节点。之后根据参数 o.set在复制积压缓冲区中查找，如果 o.set之后的数据存在，则对从节点发送+COUT INUE命令，表.可 以进.部分复制。因为缓冲区..固定，若发.缓冲溢出，则进.全量复制。⑤主节点根据偏移量把复制积压缓冲区.的数据发送给从节点，保证主从复制进.正常状态。 哨兵 .试官：那主从复制会存在哪些问题呢？ 我：主从复制会存在以下问题：.旦主节点宕机，从节点晋升为主节点，同时需要修改应..的主节点地址，还需要命令所有从节点去复制新的主节点，整个过程需要...预。主节点的写能.受到单机的限制。主节点的存储能.受到单机的限制。原.复制的弊端在早期的版本中也会.较突出，.如：Redis复制中断后，从节点会发起psync。此时如果同步不成功，则会进.全量同步，主库执.全量备份的同时，可能会造成毫秒或秒级的卡顿。

.试官：那.较主流的解决.案是什么呢？ 我：当然是哨兵啊。 .试官：那么问题.来了。那你说下哨兵有哪些功能？

我：如图，是Redi il i il s Sent ne（哨兵）的架构图。Red s Sent ne（哨兵）主要功能包括主节点存活检测、主从运.情况检测、.动故障转移、主从切换。 RedisSentinel最.配置是.主.从。Redis的 Sentinel系统可以.来管理多个Redis服务器。该系统可以执.以下四个任务：

监控：不断检查主服务器和从服务器是否正常运.。通知：当被监控的某个 Redis服务器出现问题，Sentinel通过API脚本向管理员或者其他应.程序发出通知。.动故障转移：当主节点不能正常.作时，Sentinel会开始.次.动的故障转移操作，它会将与失效主节点是主从关系的其中.个从节点升级为新的主节点，并且将其他的从节点指向新的主节点，这样...预就可以免了。配置提供者：在 Redis Sentinel模式下，客.端应.在初始化时连接的是 Sentinel节点集合，从中获取主节点的信息。 .试官：那你能说下哨兵的.作原理吗？我：话不多说，直接上图：

①每个 Sentinel节点都需要定期执.以下任务：每个 Sentinel以每秒.次的频率，向它所知的主服务器、从服务器以及其他的Sentinel实例发送.个 PING命令。（如上图）

②如果.个实例距离最后.次有效回复 PING命令的时间超过 down-after-milliseconds所指定的值，那么这个实例会被 Sentinel标记为主观下线。（如上图）

③如果.个主服务器被标记为主观下线，那么正在监视这个服务器的所有 Sent inel节点，要以每秒.次的频率确认主服务器的确进.了主观下线状态。

④如果.个主服务器被标记为主观下线，并且有.够数量的 Senti l 围内同意这.判断，那么这个主服务器被标记为客观下线。 ne（.少要达到配置.件指定的数量）在指定的时间范

⑤.般情况下，每个 Sent inel会以每 10秒.次的频率向它已知的所有主服务器和从服务器发送 INFO命令。 当.个主服务器被标记为客观下线时，Sent inel向下线主服务器的所有从服务器发送 INFO命令的频率，会从 10秒.次改为每秒.次。

⑥Sentinel和其他 Sent inel协商客观下线的主节点的状态，如果处于 SDOWN状态，则投票.动选出新的主节点，将剩余从节点指向新的主节点进.数据复制。

⑦当没有.够数量的 Sentinel同意主服务器下线时，主服务器的客观下线状态就会被移除。当主服务器重新向 Sentinel的 PING命令返回有效回复时，主服务器的主观下线状态就会被移除。.试官：不错，.试前没少下.夫啊，今天 Redis这关你过了，明天找个时间我们再聊聊其他的。（露出欣慰的微笑）我：没问题。 总结本.在.次.试的过程中讲述了 Redis是什么，Redis的特点和功能，Redis缓存的使.，Redis为什么能这么快，Redis缓存的淘汰策略，持久化的两种.式，Redis.可.部分的主从复制和哨兵的基本原理。只要功夫深，铁杵磨成针，平时准备好，.试不.慌。虽然.试不.定是这样问的，但万变不离其“宗”。 有偿投稿：欢迎投稿原创技术博.，.旦采.将给予50元-200元不等稿费，要求个.原创，图.并茂。可以是

职场经验、.试经历，也可是技术教程，学习笔记。投稿请联系微信「web527zsd」，备注投稿。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，欢迎添加.编微信「focusoncode」，每.朋友圈更新.篇.质量技术博.（..告）。 ↓扫描.维码添加.编↓

推荐阅读 1.不会IntelliJIDEA项.配置?

2.

SpringBoot异步请求和异步调.

3.删库跑路!

4..频使.的Git命令

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

经常.HashMap？这6个问题回答下！ Java后端 2019-08-27 以下.章来源于孤独烟，作者孤独烟

孤独烟 孤独烟说java,.来分享.业内的java技术和架构！ .、HashMap的实现原理? 此题可以组成如下连环炮来问

你看过HashMap源码嘛，知道原理嘛? 为什么.数组+链表？ hash冲突你还知道哪些解决办法？ 我.LinkedList代替数组结构可以么? 既然是可以的,为什么HashMap不.LinkedList,.选.数组? 1.你看过HashMap源码嘛，知道原理嘛? 针对这个问题，嗯，当然是必须看过HashMap源码。.于原理，下.那张图很清楚了:

HashMap采.Entry数组来存储key-value对，每.个键值对组成了.个Entry实体，Entry类实际上是.个单向的链表结构，它具有Next指针，可以连接下.个Entry实体。 只是在JDK1.8中，链表.度.于8的时候，链表会转成红.树！ 2.为什么.数组+链表？ 数组是.来确定桶的位置，利.元素的key的hash值对数组.度取模得到. 链表是.来解决hash冲突问题，当出现hash值.样的情形，就在数组上的对应位置形成.条链表。ps:这.的hash值并不是指hashcode，.是将hashcode.低.六位异或过的。.于为什么要这么做，继续往下看。 3.hash冲突你还知道哪些解决办法？ .较出名的有四种(1)开放定址法(2)链地址法(3)再哈希法(4)公共溢出区域法ps:.家有兴趣拓展的，..去搜.下就懂了，这个就不拓展了！ 4.我.LinkedList代替数组结构可以么? 这.我稍微说明.下，此题的意思是，源码中是这样的

ps：Entry就是.个链表节点。那我.下.这样表.

是否可.?答案很明显，必须是可以的。 5.既然是可以的,为什么HashMap不.LinkedList,.选.数组? 因为.数组效率最.！ 在HashMap中，定位桶的位置是利.元素的key的哈希值对数组.度取模得到。此时，我们已得到桶的位置。显然数组的查找效率.LinkedList.。那ArrayList，底层也是数组，查找也快啊，为啥不.ArrayList? (烟哥写到这.的时候，不禁觉得..真有想法，..把..问死了，还好我灵机.动想出了答案)因为采.基本数组结构，扩容机制可以..定义，HashMap中数组扩容刚好是2的次幂，在做取模运算的效率.。 .ArrayList的扩容机制是1.5倍扩容，那ArrayList为什么是1.5倍扩容这就不在本.说明了。 .、HashMap在什么条件下扩容? 此题可以组成如下连环炮来问

HashMap在什么条件下扩容?为什么扩容是2的n次幂?为什么为什么要先.16位异或低16位再取模运算? 1.HashMap在什么条件下扩容? 如果bucket满了(超过load factor*currentcapacity)，就要resize。 load factor为0.75，为了最.程度避免哈希冲突currentcapacity为当前数组..。 2.为什么扩容是2的次幂? HashMap为了存取.效，要尽量较少碰撞，就是要尽量把数据分配均匀，每个链表.度.致相同，这个实现就在把数据存到 哪个链表中的算法；这个算法实际就是取模，hash%length。 但是，.家都知道这种运算不如位移运算快。 因此，源码中做了优化hash&(length-1)。 也就是说hash%length==hash&(length-1) 那为什么是2的n次.呢？ 因为2的n次.实际就是1后.n个0，2的n次.-1，实际就是n个1。例如.度为8时候，3&(8-1)=32&(8-1)=2，不同位置上，不碰撞。 ..度为5的时候，3&(5-1)=02&(5-1)=0，都在0上，出现碰撞了。所以，保证容积是2的n次.，是为了保证在做(length-1)的时候，每.位都能&1，也就是和1111……1111111进.与运算。 3.为什么为什么要先.16位异或低16位再取模运算? 我先晒.下，jdk1.8.的hash.法。1.7的.较复杂，咱就不看了。

hashmap这么做，只是为了降低hash冲突的.率。 打个..，当我们的length为16的时候，哈希码(字符串“abcabcabcabcabc”的key对应的哈希码)对(16-1)与操作，对于多个key.成的hashCode，只要哈希码的后4位为0，不论不论.位怎么变化，最终的结果均为0。如下图所.

.加上.16位异或低16位的"扰动函数"后，结果如下可以看到:扰动函数优化前：1954974080 % 16 = 1954974080 & (16 -1) =0扰动函数优化后：1955003654 % 16 = 1955003654& (16 -1)=6很显然，减少了碰撞的.率。

三、讲讲hashmap的get/put的过程? 此题可以组成如下连环炮来问

知道hashmap中put元素的过程是什么样么?知道hashmap中get元素的过程是什么样么？你还知道哪些hash算法？说说String中hashcode的实现?(此题很多..问过) 1.知道hashmap中put元素的过程是什么样么? 对key的hashCode()做hash运算，计算index;如果没碰撞直接放到bucket.；如果碰撞了，以链表的形式存在buckets后；如果碰撞导致链表过.(.于等于TREEIFY_THRESHOLD)，就把链表转换成红.树(JDK1.8中的改动)；如果节点已经存在就替换oldvalue(保证key的唯.性)如果bucket满了(超过load factor*currentcapacity)，就要resize。 2.知道hashmap中get元素的过程是什么样么? 对key的hashCode()做hash运算，计算index;如果在bucket.的第.个节点.直接命中，则直接返回；如果有冲突，则通过key.equals(k)去查找对应的Entry;

若为树，则在树中通过key.equals(k)查找，O(logn)； 若为链表，则在链表中通过key.equals(k)查找，O(n)。 3.你还知道哪些hash算法？ 先说.下hash算法.嘛的，Hash函数是指把.个.范围映射到.个.范围。把.范围映射到.个.范围的.的往往是为了节省空间，使得数据容易保存。.较出名的有MurmurHash、MD4、MD5等等 4.说说String中hashcode的实现?(此题频率很.) 1 public int hashCode() { 2 int h = hash; 3 if (h == 0 && value.length > 0) 4 { 5 char val[] = value 6 ; 7 for (int i = 0; i < value.length; i++) { //欢迎关注微信公众号 Web项.聚集地 8 9 h=31 * h + val[i] 10 ; 11 } hash = h; } return h ; } String类中的hashCode计算.法还是.较简单的，就是以31为权，每.位为字符的ASCII值进.运算，..然溢出来等效取模。哈希计算公式可以计为s[0]31^(n-1)+s[1]31^(n-2)+ …+s[n-1] 那为什么以31为质数呢? 主要是因为31是.个奇质数，所以31*i*=*32*i-i=(i<<5)-i，这种位移与减法结合的计算相..般的运算快很多。 四、为什么hashmap的在链表元素数量超过8时改为红.树? 此题可以组成如下连环炮来问

知道jdk1.8中hashmap改了啥么? 为什么在解决hash冲突的时候，不直接.红.树?.选择先.链表，再转红.树? 我不.红.树，..叉查找树可以么? 那为什么阀值是8呢? 当链表转为红.树后，什么时候退化为链表? 1.知道jdk1.8中hashmap改了啥么?

由数组+链表的结构改为数组+链表+红.树。 优化了.位运算的hash算法：h^(h>>>16) 扩容后，元素要么是在原位置，要么是在原位置再移动2次幂的位置，且链表顺序不变。 最后.条是重点，因为最后.条的变动，hashmap在1.8中，不会在出现死循环问题。 2.为什么在解决hash冲突的时候，不直接.红.树?.选择先.链表，再转红.树? 因为红.树需要进.左旋，右旋，变.这些操作来保持平衡，.单链表不需要。 当元素.于8个当时候，此时做查询操作，链表结构已经能保证查询性能。当元素.于8个的时候，此时需要红.树来加快查询速度，但是新增节点的效率变慢了。因此，如果.开始就.红.树结构，元素太少，新增效率..较慢，.疑这是浪费性能的。 3.我不.红.树，..叉查找树可以么? 可以。但是.叉查找树在特殊情况下会变成.条线性结构（这就跟原来使.链表结构.样了，造成很深的问题），遍历查找会.常慢。 4.那为什么阀值是8呢? 不知道，等jdk作者来回答。这道题，.上能找到的答案都是扯淡。我随便贴.个.客.的答案，如下图所.

看出bug没？交点是6.64？交点分明是4，好么。 log4=2，4/2=2。 jdk作者选择8，.定经过了严格的运算，觉得在.度为8的时候，与其保证链表结构的查找开销，不如转换为红.树，改为维 持其平衡开销。 5.当链表转为红.树后，什么时候退化为链表? 为6的时候退转为链表。中间有个差值7可以防.链表和树之间频繁的转换。假设.下，如果设计成链表个数超过8则链表转换成树结构，链表个数.于8则树结构转换成链表，如果.个HashMap不停的插.、删除元素，链表个数在8左右徘徊，就会频繁的发.树转链表、链表转树，效率会很低。 五、HashMap的并发问题? 此题可以组成如下连环炮来问

HashMap在并发编程环境下有什么问题啊?在jdk1.8中还有这些问题么?

你.般怎么解决这些问题的？HashMap在并发编程环境下有什么问题啊? (1)多线程扩容，引起的死循环问题

(2) 多线程put的时候可能导致元素丢失

(3) put.null元素后get出来的却是null在jdk1.8中还有这些问题么?在jdk1.8中，死循环问题已经解决。其他两个问题还是存在。你.般怎么解决这些问题的？.如ConcurrentHashmap，Hashtable等线程安全等集合类。

六、你.般.什么作为HashMap的key? 此题可以组成如下连环炮来问

健可以为Null值么? 你.般.什么作为HashMap的key? 我.可变类当HashMap的key有什么问题? 如果让你实现.个.定义的class作为HashMap的key该如何实现？ 1.健可以为Null值么? 必须可以，key为null的时候，hash算法最后的值以0来计算，也就是放在数组的第.个位置。

2.你.般.什么作为HashMap的key? .般.Integer、String这种不可变类当HashMap当key，.且String最为常.。 (1) 因为字符串是不可变的，所以在它创建的时候hashcode就被缓存了，不需要重新计算。这就使得字符串很适合作为Map中的键，字符串的处理速度要快过其它的键对象。这就是HashMap中的键往往都使.字符串。

(2) 因为获取对象的时候要.到equals()和hashCode().法，那么键对象正确的重写这两个.法是.常重要的,这些类已经很规范的覆写了hashCode()以及equals().法。

3.我.可变类当HashMap的key有什么问题? hashcode可能发.改变，导致put进去的值，.法get出，如下所. 1 HashMap<List, Object> changeMap = new HashMap<>(); 2 List list = new ArrayList<>(); 3 list.add("hello"); 4 Object objectValue = new Object(); 5 changeMap.put(list, objectValue); 6 System.out.println(changeMap.get(list)); 7 list.add("hello world");//hashcode发.了改变 8 System.out.println(changeMap.get(list)); 输出值如下

4.如果让你实现.个.定义的class作为HashMap的key该如何实现？ 此题考察两个知识点

重写hashcode和equals.法注意什么? 如何设计.个不变类 针对问题.，记住下.四个原则即可 (1) 两个对象相等，hashcode.定相等

(2) 两个对象不等，hashcode不.定不等

(3) hashcode相等，两个对象不.定相等

(4) hashcode不等，两个对象.定不等针对问题.，记住如何写.个不可变类

(1)类添加.n al修饰符，保证类不被继承。

如果类可以被继承会破坏类的不可变性机制，只要继承类覆盖.类的.法并且继承类可以改变成员变量值，那么.旦.类以.类的形式出现时，不能保证当前类是否可变。

(2) 保证所有成员变量必须私有，并且加上.nal修饰

通过这种.式保证成员变量不可改变。但只做到这.步还不够，因为如果是对象成员变量有可能再外部改变其值。所以第4点弥补这个不.。

(3) 不提供改变成员变量的.法，包括setter避免通过其他接.改变成员变量的值，破坏不可变特性。

(4) 通过构造器初始化所有成员，进.深拷.(deepcopy)

如果构造器传.的对象直接赋值给成员变量，还是可以通过对传.对象的修改进.导致改变内部变量的值。例如： 1 public final class ImmutableDemo { 2 3 private final int[] myArray; 4 public ImmutableDemo(int[] array) { 5 6 this.myArray = array; // wrong。欢迎关注微信公众号 Web项.聚集地 } }

这种.式不能保证不可变性，myArray和array指向同.块内存地址，..可以在ImmutableDemo之外通过修改array对象的值来改变myArray内部的值。 为了保证内部的值不被修改，可以采.深度copy来创建.个新内存保存传.的值。正确做法： 1 public final class MyImmutableDemo { 2 3 private final int[] myArray; 4 public MyImmutableDemo(int[] array) { 5 6 this.myArray = array.clone();//欢迎关注微信公众号 Web项.聚集地 } } 阅读原. 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ 记.次SpringBoot项.启动卡住问题排查记录 陈凯玲Java后端 2019-12-02 点击上.Java后端，选择设为星标优质.章，及时送达

作者|陈凯玲来源|my.oschina.net/keking/blog/3058921 .个SpringBoot开发的项.，SpringBoot版本是1.5.7，携带的spring版本是4.1.3。开发反馈，突然在本地启动不起来了，表象特征就是在本地IDEA上运.时，进程卡住也不退出，应.启动时加载相关组件的.志也不输出。症状如下图：

问题分析 因为没有有.的.志信息，所以不能从.志这个层.上排查问题。但是像这种没有输出.志的话，.般情况下，肯定是程序内部启动流程卡在什么地.了，只能通过打印下当前线程堆栈信息了解下。.般情况下，在服务器环境，我们会使.java.具包中的jstack.具来查看：如jstackpid（应.java进程）。 但是，在IDEA本地开发的话，IDEA内置了.个.具，可以直接查看当前应.的线程上线.信息，如：

注意下.那个箭头指向的像照相机.样的图标，故图思意，就是打印当前线程快照的的意思。点击后，就出现了右边那些线程上下.信息了，可以看到有很多的线程，我们主要关注下main线程，线程状态确实是waiting的，接着点击箭头所指向的main线程，可以看到如下内容： "main@1"prio=5tid=0x1nid=NAwaitingjava.lang.Thread.State:WAITING atsun.misc.Unsafe.park(Unsafe.java:-1)atjava.util.concurrent.locks.LockSupport.park(LockSupport.java:175)atjava.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)atjava.util.concurrent.locks.AbstractQueuedSynchronizer.doAcquireSharedInterruptibly(AbstractQueuedSynchronizer.java:997)atjava.util.concurrent.locks.AbstractQueuedSynchronizer.acquireSharedInterruptibly(AbstractQueuedSynchronizer.java:1304)atjava.util.concurrent.CountDownLatch.await(CountDownLatch.java:231)atorg.springframework.boot.autoconfigure.BackgroundPreinitializer.onApplicationEvent(BackgroundPreinitializer.java:63)atorg.springframework.boot.autoconfigure.BackgroundPreinitializer.onApplicationEvent(BackgroundPreinitializer.java:45)atorg.springframework.context.event.SimpleApplicationEventMulticaster.doInvokeListener(SimpleApplicationEventMulticaster.java:172)atorg.springframework.context.event.SimpleApplicationEventMulticaster.invokeListener(SimpleApplicationEventMulticaster.java:158)atorg.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:139)atorg.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:127)atorg.springframework.boot.context.event.EventPublishingRunListener.finished(EventPublishingRunListener.java:115)atorg.springframework.boot.SpringApplicationRunListeners.callFinishedListener(SpringApplicationRunListeners.java:79)atorg.springframework.boot.SpringApplicationRunListeners.finished(SpringApplicationRunListeners.java:72)atorg.springframework.boot.SpringApplication.handleRunFailure(SpringApplication.java:745)atorg.springframework.boot.SpringApplication.run(SpringApplication.java:314)atorg.springframework.boot.builder.SpringApplicationBuilder.run(SpringApplicationBuilder.java:134)-locked<0xea6> (ajava.util.concurrent.atomic.AtomicBoolean)atorg.springframework.cloud.bootstrap.BootstrapApplicationListener.bootstrapServiceContext(BootstrapApplicationListener.java:175)atorg.springframework.cloud.bootstrap.BootstrapApplicationListener.onApplicationEvent(BootstrapApplicationListener.java:98)atorg.springframework.cloud.bootstrap.BootstrapApplicationListener.onApplicationEvent(BootstrapApplicationListener.java:64)atorg.springframework.context.event.SimpleApplicationEventMulticaster.doInvokeListener(SimpleApplicationEventMulticaster.java:172)atorg.springframework.context.event.SimpleApplicationEventMulticaster.invokeListener(SimpleApplicationEventMulticaster.java:165)atorg.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:139)atorg.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:127)atorg.springframework.boot.context.event.EventPublishingRunListener.environmentPrepared(EventPublishingRunListener.java:74)atorg.springframework.boot.SpringApplicationRunListeners.environmentPrepared(SpringApplicationRunListeners.java:54)atorg.springframework.boot.SpringApplication.prepareEnvironment(SpringApplication.java:325)atorg.springframework.boot.SpringApplication.run(SpringApplication.java:296)atcn.keking.project.customerManagement.KekingCustomerManagement.main(KekingCustomerManagement.java:36) 可以看到是通过CountDownLatch.await()阻塞了线程，接着看下.那.，所在代码块如下：
privatestaticfinalCountDownLatchpreinitializationComplete=newCountDownLatch(1); @Override publicvoidonApplicationEvent(SpringApplicationEventevent) {if(eventinstanceofApplicationEnvironmentPreparedEvent){if(preinitializationStarted.compareAndSet(false,true)) {performPreinitialization(); }} if(eventinstanceofApplicationReadyEvent||eventinstanceofApplicationFailedEvent){ try{ preinitializationComplete.await(); } catch(InterruptedExceptionex){ Thread.currentThread().interrupt();}}} 这是springboot中的.个安全初始化资源的.个类，代码所.为监听SpringApplicationEvent事件，可以肯定的是，它的逻辑执.到了 preinitializationComplete.await();这.，导致了线程阻塞了。 正常情况下，spring会触发ApplicationEnvironmentPreparedEvent事件完成资源的初始化，这.先不深究Spring为什么这么做，主要 通过程序逻辑看下为什么卡这.了，在preinitializationComplete.await();所在.打个断点，看下event对象.的信息，如下：

原来event是.个Spring上下.初始化失败的异常事件对象，对象.包含了具体的异常信息，如箭头所指，关键异常信息如：
NoSuchMethodError："org.springframework.util.ObjectUtils.unwrapOptional(Ljava/lang/Object;)Ljava/lang/Object;" 假设问题 通过上.的分析，基本定位到Springboot应.启动卡住这个表象背后的真实原因了，.且也定位到了异常信息。出现NoSuchMethodError异常，是因为调..法的时候，找不到.法了。.般出现在两个有关联的jar包，但是版本对不上了，也就是常说的jar版本依赖冲突。查看了下，ObjectUtils是spring-core包.的.个类，当前的4.1.3版本确实没有这个unwrapOptional.法，spring-core-5.x的版本才新增了这个.法。因为之前的依赖是没有问题，为什么现在spring上下.会调.5.x的版本的.法呢？ 所以先假设近期有开发在pom.xml.添加了新的的依赖，导致了这个问题。 ..求证 有了找问题的.向就好办了，因为代码都是git管理维护的，所以查看下pom.xml.件近期的提交记录即可，查看后，确实发现了近期对

pom.xml有改动，添加了.个依赖 org.springframework spring-context 5.1.6.RELEASE 这.还涉及到.点Maven依赖优先级的问题，在pom.xml.直接这样添加的依赖优先于其他jar中pom.xml依赖的，也就是说，即使 springboot1.5.7.带了Spring-context.4.1.3，但是这样指定后，应.最后依赖的还是5.1.6的版本。 具体的Maven依赖关系，可以参考我的博.《关于Maven的使.，这些你都了解了么？》。结合之前的分析，.九不离.了就是因为加了这个依赖导致的问题，spring-context.5.1.6配合spring-core.4.1.3肯定得出问题啊。直接移除这个依赖，然后启动系统.切正常，.志打印了Spring加载上线.的信息。 问题总结 定位这个问题的关键在于要了解java中线程堆栈的知识，在没有.够异常.志情况下通过线程快照排查问题。 在定位到问题后，如NoSuchMethodError这样的异常，需要平时的经验积累来假设问题的真实原因，然后在追本溯源验明问题所在根本原因。找问题本质.定要这种循序渐进的思路。 举例，出现这种问题，如果你直接去搜索引擎搜：“Springboot应.启动卡住了”，是搜不出来什么东西的，但是当你发现了是由于jar冲突。去搜索引擎搜索：“NoSuchMethodError："org.springframework.util.ObjectUtils.unwrapOptional(Ljava/lang/Object;)Ljava/lang/Object;"”。就会有很多的内容，很容易解决问题。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. 你在公司项...看过哪些操蛋的代码?

2.你知道 Spring Batch吗?

3.

.试题：Lucene、Solr、ElasticSearch

4.

3分钟带你彻底搞懂 Java泛型背后的秘密

5.团队开发中 Git最佳实践

喜欢.章，点个在看

 阿.员.最常.的问题排查.具单 红魔七号Java后端 2019-11-06 点击上.Java后端，选择设为星标 优质.章，及时送达

作者|红魔七号 链接|https://urlify.cn/aiqYni 这是.篇来源于阿.内部技术论坛的.章，原.在阿.内部获得.致好评。作者已经把这篇.章开放到云栖社区中供外.访问。 Hollis对.章内容做了部分删减，主要删减掉了其中只有阿.内部才能使.的.具的介绍，并删减掉部分只有通过阿.内.才能访问到的链接。

前.平时的.作中经常碰到很多疑难问题的处理，在解决问题的同时，有.些.具起到了相当.的作.，在此书写下来，.是作为笔记，可以让..后续忘记了可快速翻阅，.是分享，希望看到此.的同学们可以拿出...常觉得帮助很.的.具，.家.起进步。 闲话不多说，开搞。

Linux命令类 tail 最常.的tail-f tail-300fshopbase.log#倒数300.并进.实时监听.件写.模式 grep grepforestf.txt#.件查找 grepforestf.txtcpf.txt#多.件查找 grep'log'/home/admin-r-n#.录下查找所有符合关键字的.件 catf.txt|grep-ishopbase grep'shopbase'/home/admin-r-n--include*.{vm,java}#指定.件后缀 grep'shopbase'/home/admin-r-n--exclude*.{vm,java}#反匹配 seq10|grep5-A3#上匹配 seq10|grep5-B3#下匹配 seq10|grep5-C3#上下匹配，平时.这个就妥了 catf.txt|grep-c'SHOPBASE' awk 1基础命令 awk'{print$4,$6}'f.txt awk'{printNR,$0}'f.txtcpf.txt awk'{printFNR,$0}'f.txtcpf.txt awk'{printFNR,FILENAME,$0}'f.txtcpf.txt awk'{printFILENAME,"NR="NR,"FNR="FNR,"$"NF"="$NF}'f.txtcpf.txt echo1:2:3:4|awk-F:'{print$1,$2,$3,$4}' 2匹配 awk'/ldb/{print}'f.txt#匹配ldb awk'!/ldb/{print}'f.txt#不匹配ldb awk'/ldb/&&/LISTEN/{print}'f.txt#匹配ldb和LISTEN awk'$5~/ldb/{print}'f.txt#第五列匹配ldb 3内建变量 NR:NR表.从awk开始执.后，按照记录分隔符读取的数据次数，默认的记录分隔符为换.符，因此默认的就是读取的数据.数，NR可以理解为NumberofRecord的缩写。 FNR:在awk处理多个输..件的时候，在处理完第.个.件后，NR并不会从1开始，.是继续累加，因此就出现了FNR，每当处理.个新.件的时候，FNR就从1开始计数，FNR可以理解为FileNumberofRecord。 NF:NF表..前的记录被分割的字段的数.，NF可以理解为NumberofField。 Tips：关注微信公众号：Java后端，每.获取博.的推送。 find sudo-uadminfind/home/admin/tmp/usr-name*.log(多个.录去找)find.-iname*.txt(..写都匹配)find.-typed(当前.录下的所有..录)find/usr-typel(当前.录下所有的符号链接)find/usr-typel-name"z*"-ls(符号链接的详细信息eg:inode,.录)find/home/admin-size+250000k(超过250000k的.件，当然+改成-就是.于了)find/home/adminf-perm777-execls-l{};

(按照权限查询.件)find/home/admin-atime-1 1天内访问过的.件 find/home/admin-ctime-1 1天内状态改变过的.件 find/home/admin-mtime-1 1天内修改过的.件 find/home/admin-amin-1 1分钟内访问过的.件 find/home/admin-cmin-1 1分钟内状态改变过的.件 find/home/admin-mmin-1 1分钟内修改过的.件 pgm 批量查询vm-shopbase满.条件的.志 pgm-A-fvm-shopbase'cat/home/admin/shopbase/logs/shopbase.log.2017-01-17|grep2069861630' tsar tsar是咱公司..的采集.具。很好.,将历史收集到的数据持久化在磁盘上，所以我们快速来查询历史的系统数据。当然实时的应.情况也是可以查询的啦。.部分机器上都有安装。 tsar###可以查看最近.天的各项指标

top top除了看.些基本信息之外，剩下的就是配合来查询vm的各种问题了 ps-ef|grepjava top-H-ppid 获得线程10进制转16进制后jstack去抓看这个线程到底在.啥 其他

排查利器 btrace .当其冲的要说的是btrace。真是.产环境&预发的排查问题.杀器。简介什么的就不说了。直接上代码. 1、查看当前谁调.了ArrayList的add.法，同时只打印当前ArrayList的size.于500的线程调.栈

@OnMethod(clazz="java.util.ArrayList",method="add",location=@Location(value=Kind.CALL,clazz="/./",*method*="/./"))publicstaticvoidm(@ProbeClassNameStringprobeClass,@ProbeMethodNameStringprobeMethod,@TargetInstanceObject instance,@TargetMethodOrFieldStringmethod){ if(getInt(field("java.util.ArrayList","size"),instance)>479) {println("checkwhoArrayList.addmethod:"+probeClass+"#"+probeMethod +",method:"+method+",size:"+getInt(field("java.util.ArrayList","size"),instance));jstack(); println(); println("============================="); println(); }} 2、监控当前服务.法被调.时返回的值以及请求的参数 @OnMethod(clazz="com.taobao.sellerhome.transfer.biz.impl.C2CApplyerServiceImpl",method="nav",location=@Location (value=Kind.RETURN)) publicstaticvoidmt(longuserId,intcurrent,intrelation,Stringcheck,StringredirectUrl,@ReturnAnyTyperesult){ println("parameter#userId:"+userId+",current:"+current+",relation:"+relation+",check:"+check+",redirectUrl:"+redirectUrl+",result:"+result);} 更多内容，感兴趣的请移步：https://github.com/btraceio/btrace 注意: 1. 经过观察，1.3.9的release输出不稳定，要多触发.次才能看到正确的结果

2.

正则表达式匹配trace类时范围.定要控制，否则极有可能出现跑满CPU导致应.卡死的情况

3.

由于是字节码注.的原理，想要应.恢复到正常情况，需要重启应.。

Greys 说.个挺棒的功能(部分功能和btrace重合): sc-dfxxx :输出当前类的详情,包括源码位置和classloader结构 traceclassmethod :相当喜欢这个功能!很早前可以早JProfiler看到这个功能。打印出当前.法调.的耗时情况，细分到每个.法。 javOSize 就说.个功能 classes：通过修改了字节码，改变了类的内容，即时.效。所以可以做到快速的在某个地.打个.志看看输出，缺点是对代码的侵.性太.。但是如果..知道..在.嘛，的确是不错的玩意.。 其他功能Greys和btrace都能很轻易做的到，不说了。 JProfiler 之前判断许多问题要通过JProfiler，但是现在Greys和btrace基本都能搞定了。再加上出问题的基本上都是.产环境(.络隔离)，所以基本不怎么使.了，但是还是要标记.下。官.请移步https://www.ej-technologies.com/products/jprofiler/overview.html

.杀器 eclipseMAT 可作为eclipse的插件，也可作为单独的程序打开。详情请移步http://www.eclipse.org/mat/ jps java三板斧，噢不对，是七把 我只..条命令：

jstack 普通.法:

native+java栈: sudo-uadmin/opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jstack-m2815

jinfo 可看系统启动的参数，如下

jmap 两个.途1.查看堆的情况 sudo-uadmin/opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap-heap2815

2.dump sudo-uadmin/opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap-dump:live,format=b,file=/tmp/heap2.bin2815 或者 sudo-uadmin/opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap-dump:format=b,file=/tmp/heap3.bin2815 3.看看堆都被谁占了?再配合zpro filer和btrace，排查问题简直是如.添翼 sudo-uadmin/opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap-histo2815|head-10

jstat jstat参数众多，但是使..个就够了

jdb 时.今.，jdb也是经常使.的。 jdb可以.来预发debug,假设你预发的java_home是/opt/taobao/java/，远程调试端.是8000.那么 sudo-uadmin/opt/taobao/java/bin/jdb-attach8000 .

出现以上代表jdb启动成功。后续可以进.设置断点进.调试。具体参数可.oracle官.说明 http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html CHLSDB CHLSDB感觉很多情况下可以看到更好玩的东西，不详细叙述了。查询资料听说jstack和jmap等.具就是基于它的。 sudo-uadmin/opt/taobao/java/bin/java-classpath/opt/taobao/java/lib/sa-jdi.jarsun.jvm.hotspot.CLHSDB 更详细的可.R.此贴 http://rednaxelafx.iteye.com/blog/1847971

pluginofintellijidea keypromoter 快捷键.次你记不住，多来.次你总能记住了吧?

mavenhelper 分析maven依赖的好帮.。

VMoptions 1、你的类到底是从哪个.件加载进来的？ -XX:+TraceClassLoading结果形如 [Loadedjava.lang.invoke.MethodHandleImpl$LazyfromD:\programme\jdk\jdk8U74\jre\lib\rt.jar] 2、应.挂了输出dump.件 -XX:+HeapDumpOnOutOfMemoryError-XX:HeapDumpPath=/home/admin/logs/java.hprof

jar包冲突 把这个单独写个.标题不过分吧？每个.或多或少都处理过这种烦.的case。我特么下边这么多.案不信就搞不定你?

打出所有依赖

只打出指定groupId和artifactId的依赖关系

vm启动脚本加.。在tomcat启动脚本中可.加载类的详细信息

vm启动脚本加.。在tomcat启动脚本中可.加载类的详细信息

greys的sc命令也能清晰的看到当前类是从哪.加载过来的

通过以下url可以获知当前类是从哪.加载的 curlhttp://localhost:8006/classloader/locate?class=org.apache.xerces.xs.XSObjec 其他

dmesg 如果发现..的java进程悄.声息的消失了，.乎没有留下任何线索，那么dmesg.发，很有可能有你想要的。sudodmesg|grep-ikill|less 去找关键字oom_killer。找到的结果类似如下: [6710782.021013]javainvokedoom-killer:gfp_mask=0xd0,order=0,oom_adj=0,oom_scoe_adj=0 [6710782.070639][]?oom_kill_process+0x68/0x140 [6710782.257588]Taskin/LXC011175068174killedasaresultoflimitof/LXC011175068174 [6710784.698347]Memorycgroupoutofmemory:Killprocess215701(java)score854orsacrificechild [6710784.707978]Killedprocess215701,UID679,(java)total-vm:11017300kB,anon-rss:7152432kB,file-rss:1232kB 以上表明，对应的java进程被系统的OOMKiller给.掉了，得分为854.解释.下OOM killer（Out-Of-Memory killer），该机制会监控机器的内存资源消耗。当机器内存耗尽前，该机制会扫描所有的进程（按照.定规则计算，内存占.，时间等），挑选出得分最.的进程，然后杀死，从.保护机器。 dmesg.志时间转换公式: log实际时间=格林威治1970-01-01+(当前时间秒数-系统启动.今的秒数+dmesg打印的log时间)秒数： date-d"1970-01-01UTC `echo"$(date+%s)-$(cat/proc/uptime|cut-f1-d'')+12288812.926194"|bc` seconds" 剩下的，就是看看为什么内存这么.，触发了OOM-Killer了。

新技能get RateLimiter 想要精细的控制QPS?.如这样.个场景，你调.某个接.，对.明确需要你限制你的QPS在400之内你怎么控制？这个时候RateLimiter就有了.武之地。详情可移步http://ifeve.com/guava-ratelimite -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读

1. Spring Boot全局异常处理整理 2.细说 Java主流.志.具库

3.9个爱不释.的 JSON .具

4.

12306的架构到底有多.逼?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试了N个候选.后，我总结出这份Java.试准备技巧！ Java后端 2019-11-04 点击上.Java后端，选择设为星标 优质.章，及时送达

作者|hsm_computer来源|cnblogs.com/JavaArchitect/p/10011253.html .录： 1. 框架是重点，但别让.感觉你只会.寨别.的代码2.别只看单机版的框架，分布式也需要了解3.对于数据库，别只知道增删改查，得了解性能优化

4.

Java核.，围绕数据结构和性能优化准备.试题

5.

Linux..，.少了解如何看.志排查问题

6.

通读.段底层代码，作为加分项

7.

切记切记，把上述技能嵌.到你的项..

8.

.结：本.更多讲述的准备.试的.法

在上周，我密集.试了若.位Java后端的候选.，.作经验在3到5年间。我的标准其实不复杂：

第.能.活

第.Java基础要好

第三最好熟悉些分布式框架相信其它公司招初级开发时，应该也照着这个标准来.的。 我也知道，不少候选.能.其实不差，但.试时没准备或不会说，这样的.可能进团队.活后确实能达到期望，但可能.法通过.试，.试官只根据.试情况来判断。 要知道，我们平时.活更偏重于业务，不可能.量接触到算法，数据结构，底层代码这类.试必问的问题点。 换句话说，.试准备点和平时.作要点匹配度很.。 作为.试官，我只能根据候选.的回答来决定.试结果。不过，与..便...便。 所以我在本..，将通过.些常.的问题来介绍.试的准备技巧。 .家在看后.定会感叹：只要.法得当，准备.试第.不难，第..的时间也不会太多。 1、框架是重点，但别让.感觉你只会.寨别.的代码 在.试前，我会阅读简历以查看候选.在框架..的项.经验，在候选.的项.介绍的环节，我也会着重关注候选.最近的框架经验，.前.较热.的是SSM。 不过，.般.作在5年内的候选.，.多仅仅是能".寨"别.的代码，也就是说能在现有框架的基础上，照着别.写的流程，扩展出新的功能模块。 .如要写个股票挂单的功能模块，是会模仿现有的下单流程，然后从前端到后端再到数据库，依样画葫芦写.遍，最多把功能相关的代码点改掉。 其实我们每个.都这样过来的，但在.试时，如果你仅仅表现出这样的能.，就和.多数.的.平差不多了，在这点就没法体现出你的优势了。我们知道，如果单纯使.SSM框架，.多数项.都会有痛点。.如数据库性能差，或者业务模块.较复杂，并发量. 较.，.Spring MVC.的Controller.法满.跳转的需求。所以我.般还会主动问：你除了依照现有框架写业务代码时，还做了哪些改动？我听到的回答有：增加了Redis缓存，以避免频繁调..些不变的数据。或者，在MyBitas的xml.，select语句where条件有isnull，即这个值有就增加.个where条件，对此，会对任何 .个where增加.个不带isnull的查询条件，以免该语句当传.参数都是null时，做全表扫描。 或者.脆说，后端异步返回的数据量很.，时间很.，我在项..就调.了异步返回的最.时间，或者对返回信息做了压缩处理，以增加.络传输性能。对于这个问题，我不在乎听到什么回答，我只关.回答符不符逻辑。.般只要答对，我就会给出"在框架层.有.. 的体会，有.定的了解"的.试评价。否则，我就只会给出"只能在项.经理带领下编写框架代码，对框架本.了解不多"。其实，在准备.试时，归纳框架.的要点并不难，我就

不信所有.在做项.时.点积累也没，只要你说出来，可以说， 这..你就碾压了将近7成的竞争者。 2、别只看单机版的框架，分布式也要了解 此外，在描述项..框架技术时，最好你再带些分布式的技术。下.我列些.家可以准备的分布式技术。

1.

反向代理..，nginx的基本配置，.如如何通过lua语.设置规则，如何设置session粘滞。如果可以，再看些 nginx的底层，.如协议，集群设置，失效转移等。

2.

远程调.dubbo..，可以看下dubbo和zookeeper整合的知识点，再深.步，了解下dubbo底层的传输协议和序列化.式。

3.

消息队列..，可以看下kafka或任意.种组件的使..式，简单点可以看下配置，.作组的设置，再深.点，可以看下Kafka集群，持久化的.式，以及发送消息是..连接还是短拦截。

以上仅仅是.3个组件举例，.家还可以看下Redis缓存，.志框架，MyCAT分库分表等。准备的.式有两.类：

第.是要会说怎么.，这.较简单，能通过配置.件搭建成.个功能模块即可

第.是可以适当读些底层代码，以此了解下协议，集群和失效转移之类的.级知识点。

如果能在.试中侃侃.谈分布式组件的底层，那么得到的评价就会.较好了，.如"深.了解框架底层"，或"框架经验丰富"，这样就算去.试架构师也.了，更何况是.级开发。 Tips：欢迎.家关注微信公众号：Java后端，来获取更多推送。 3、对于数据库，别只知道增删改查，得了解性能优化 在实际项..，.多数程序员.到的可能仅仅是增删改查，当我们.Mybatis时，这个情况更普遍。不过如果你.试时也这样表现，估计你的能.就和其它竞争者差不多了。这..，你可以准备如下的技能。
1. SQL.级..，.如group by, having，左连接，.查询（带in），.转列等.级.法。

2.

建表..，你可以考虑下，你项.是.三范式还是反范式，理由是什么？

3.

尤其是优化，你可以准备下如何通过执.计划查看SQL语句改进点的.式，或者其它能改善SQL性能的.式（.如建索引等）。

4.

如果你感觉有能.，还可以准备些MySQL集群，MyCAT分库分表的技能。.如通过LVS+Keepalived实现MySQL负载均衡，MyCAT的配置.式。同样，如果可以，也看些相关的底层代码。

哪怕你在前三点表现.般，那么.少也能超越将近.半的候选.，尤其当你在SQL优化..表现.常好，那么你在.试.级开发时，数据库层..定是达标的。 如果你连第四点也回答.常好，那么恭喜你，你在数据库..的能.甚.达到了初级架构的级别。 4、Java核.，围绕数据结构和性能优化准备.试题 Java核.这块，.上的.试题很多，不过在此之外，.家还应当着重关注集合（即数据结构）和多线程并发这两块在此基础上，.家可以准备些设计模式和虚拟机的说辞。下.列些我.般会问的部分问题：

1. String a = "123"; String b = "123";a==b的结果是什么？这包含了内存，String存储.式等诸多知识点
2.

HashMap.的hashcode.法和equal.法什么时候需要重写？如果不重写会有什么后果？对此.家可以进.步了解HashMap（甚.ConcurrentHashMap）的底层实现

3. ArrayList和LinkedList底层实现有什么差别？它们各.适.于哪些场合？对此.家也可以了解下相关底层代码。
4. volatile关键字有什么作.？由此展开，.家可以了解下线程内存和堆内存的差别。

5.

JVM.，new出来的对象是在哪个区？再深..下，问下如何查看和优化JVM虚拟机内存。

5. CompletableFuture，这个是JDK1.8.的新特性，通过它怎么实现多线程并发控制？ 7.Java的静态代理和动态代理有什么差别？最好结合底层代码来说。 通过上述的问题点，我其实不仅仅停留在"会."级别，.如我不会问如何在ArrayList.放元素。 .家可以看到，上述问题包含了"多线程并发"，"JVM优化"，"数据结构对象底层代码"等细节，.家也可以举.反三，通过看.些.级知识，多准备些其它类似.试题。 我们知道，.前Java开发是以Web框架为主，那么为什么还要问Java核.知识点呢？我这个是有切.体会的。 之前在我团队.，我.过两个.，.个是就会.活，具体表现是会.Java核.基本的API，.且也没有深.了解的意愿（估计不知道该怎么深.了解），另.位平时专.会看些Java并发，虚拟机等的.级知识。 过了半年以后，后者的能.快速升级到.级开发，由于对JAVA核.知识点了解很透彻，所以看.些分布式组件的底层实现没什么.问题。.前者，.直在重复劳动，能.也只.直停留在"会.活"的层.。 .在现实的.试中，如果不熟悉Java核.知识点，估计升.级开发都难，更别说是.试架构师级别的岗位了。 5、Linux..，.少了解如何看.志排查问题 如果候选.能证明..有"排查问题"和"解决问题"的能.，这绝对是个加分项，但怎么证明？ .前.多数的互联.项.，都是部署在Linux上，也就是说，.志都是在Linux，下.归纳些实际的Linux操作。 1.能通过 less命令打开.件，通过Shift+G到达.件底部，再通过?+关键字的.式来根据关键来搜索信息

6.

能通过grep的.式查关键字，具体.法是, grep关键字 .件名，如果要两次在结果.查找的话，就.grep关键字1 .件名 |关键字2 --color。最后--color是.亮关键字

3.能通过v i来编辑.件

4.

能通过chmod来设置.件的权限

当然，还有更多更实.的Linux命令，但在实际.试过程中，不少候选.连.条linux命令也不知道。还是这句话，你哪怕知道些很基本的，也..般.强了。 6、通读.段底层代码，作为加分项 如何证明..对.个知识点.常了解?莫过于能通过底层代码来说明。 我在和不少.作经验在5年之内的程序员沟通时，不少.认为这很难？确实，如果要通过阅读底层代码了解分布式组件，那难度不.，但如果如下部分的底层代码，并不难懂。

1. ArrayList,LinkedList的底层代码.，包含着基于数组和链表的实现.式，如果.家能以此讲清楚扩容，"通过枚举器遍历"等.式，绝对能证明..。
2.

HashMap直接对应着Hash表这个数据结构，在HashMap的底层代码.，包含着hashcode的put，get等的操作，甚.在ConcurrentHashMap.，还包含着Lock的逻辑。如果.家在.试中，看看..

ConcurrentHashMap，再结合在纸上边说边画，那.定能征服.试官。

3.

可以看下静态代理和动态代理的实现.式，再深..下，可以看下Spring AOP.的实现代码。

4.或许Sp irng IOC和MVC的底层实现代码.较难看懂，但.家可以说些关键的类，根据关键流程说下它们的实现.式。

其实准备的底层代码未必要多，.且也不限于在哪个..，.如集合.基于红.树的TreeSet，基于NIO的开源框架，甚.分布式组件的Dubbo，都可以准备。 .且准备时未必要背出所有的底层（事实上很难做到），你只要能结合.些重要的类和.法，讲清楚思路即可（.如讲清楚HashMap如何通过hashCode快速定位）。 那么在.试时，如何找到个好机会说出你准备好的上述底层代码？ 在.试时，总会被问到集合，Spring MVC框架等相关知识点，你在回答时，顺便说.句，"我还了解这块的底层实现"，那么.试官.定会追问，那么你就可以说出来了。 不要.看这个对候选.的帮助，.旦你讲了，只要意思到位，那么最少能得到个"积极专业"的评价，如果描述很清楚，那么评价就会升级到"熟悉Java核.技能（或Spring MVC），且基本功扎实"。 要知道，.试中，很少有.能讲清楚底层代码，所以你抛出了这个话题，哪怕最后没达到预期效果，.试官也不会由此对你降低评价。 所以说，准备这块绝对是"有百利...害"的挣钱买卖。 7、切记切记，把上述技能嵌.到你的项.. 在.试过程中，我经常会听到.些.较遗憾的回答，.如候选.对SQL优化技能讲得头头是道，但最后得知，这是

他平时.学时掌握的，并没.在实际项..。 当然这总.不说要好，所以我会写下“在平时.学过SQL优化技能”，但如果在项..实践过，那么我就会写下“有实际数据库SQL优化的技能”。 .家可以对.下两者的差别，.个是偏重理论，.个是直接能.活了。其实，很多场景.，我就不信在实际项...定没有实践过SQL优化技能。 从这个案例中，我想告诉.家的是，你之前费了千.万苦（其实.法.向得到，也不.费太.精.）准备的很多技能和说辞，最后应该落实到你的实际项..。 .如你有过在Linux.志.查询关键字排查问题的经验，在描述时你可以带.句，在之前的项..我就这样.的。 .如，你通过看底层代码，了解了TreeSet和HashSet的差别以及它们的适.范围，那么你可以回想你之前做的项.，是否有个场景仅仅适.于TreeSet？ 如果有，那么你就可以适当描述下项.的需求，然后说，通过读底层代码，我了解了两者的差别，.且在这个实际需求.，我就.了TreeSet，.且我还专.做了对.性试验，发现.TreeSet.HashSet要.xx个百分点。请记得，“实践经验”.定.“理论经验”值钱，.且.多数你知道的理论上的经验，.定在你的项...过。所以，如果你仅仅让.试官感觉你只有“理论经验”，那就太亏了。 8、.结：本.更多讲述的准备.试的.法 本.给出的.试题并不多，但本.并没有打算给出太多的.试题。从本..，.家更多看到的是.试官发现的诸多候选.的痛点。 本.的.意是让.家别再重蹈别.的覆辙，因此给出了不少准备.试的.法。 你的能.或许.别.出众，但如果你准备.试的.式和别.差不多，或者就拿你在项...的活来说事，.没有归纳出你在项.中的亮点，那么.试官还真的会看扁你。 作者|hsm_computer来源|cnblogs.com/JavaArchitect/p/10011253.html -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. 推荐.位.神，.握 GitHub 16000 star

2.附源码！Spring Boot并发登录.数控制

3.为什么 Redis单线程却能.撑.并发?

4.

.货！MySQL数据库开发规范

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

.试再问ThreadLocal，别说你不会 Java后端 2019-11-14 点击上.Java后端，选择设为星标 优质.章，及时送达

作者|坚持就是胜利链接|juejin.im/post/5d427f306fb9a06b122f1b94 ThreadLocal是什么 以前.试的时候问到ThreadLocal总是.脸懵逼，只知道有这个哥们，不了解他是.来做什么的，更不清楚他的原理了。表.上看他是和多线程，线程同步有关的.个.具类，但其实他与线程同步机制.关。 线程同步机制是多个线程共享同.个变量，.ThreadLocal是为每个线程创建.个单独的变量副本，每个线程都可以改变..的变量副本.不影响其它线程所对应的副本。 官.API上是这样介绍的： 该类提供了线程局部(thread-local)变量。这些变量不同于它们的普通对应物，因为访问某个变量（通过其get或set.法）的每个线程都有..的局部变量，它独.于变量的初始化副本。ThreadLocal实例通常是类中的privatestatic字段，它们希望将状态与某.个线程（例如，..ID或事务ID）相关联。 ThreadLocal的API ThreadLocal定义了四个.法: get():返回此线程局部变量当前副本中的值 set(Tvalue):将线程局部变量当前副本中的值设置为指定值 initialValue():返回此线程局部变量当前副本中的初始值 remove():移除此线程局部变量当前副本中的值 ThreadLocal还有.个特别重要的静态内部类ThreadLocalMap，该类才是实现线程隔离机制的关键。get()、set()、remove()都是基于该内部类进.操作，ThreadLocalMap.键值对.式存储每个线程变量的副本，key为当前的ThreadLocal对象，value为对应线程的变量副本。试想，每个线程都有..的ThreadLocal对象，也就是都有..的ThreadLocalMap，对..的ThreadLocalMap操作，当然是互不影响的了，这就不存在线程安全问题了，所以ThreadLocal是以空间来交换安全性的解决思路。 使.实例 假设每个线程都需要.个计数值记录..做某件事做了多少次，各线程运.时都需要改变..的计数值.且相互不影响，那么 ThreadLocal就是很好的选择，这.ThreadLocal.保存的当前线程的局部变量的副本就是这个计数值。 publicclassSeqCount{ privatestaticThreadLocalseqCount=newThreadLocal(){@Override protectedIntegerinitialValue(){ return0;}}; publicintnextSeq(){seqCount.set(seqCount.get()+1); returnseqCount.get(); } publicstaticvoidmain(String[]args) {SeqCountseqCount=newSeqCount(); SeqThreadseqThread1=newSeqThread(seqCount);SeqThreadseqThread2=newSeqThread(seqCount);SeqThreadseqThread3=newSeqThread(seqCount);SeqThreadseqThread4=newSeqThread(seqCount); seqThread1.start();seqThread2.start();seqThread3.start();seqThread4.start(); } publicstaticclassSeqThreadextendsThread{ privateSeqCountseqCount; publicSeqThread(SeqCountseqCount)

{this.seqCount=seqCount;} @Override publicvoidrun(){for(inti=0;i<3;i++)
{System.out.println(Thread.currentThread().getName()+"seqCount:"+seqCount.nextSeq());}}}} 运.结果：

解决SimpleDateFormat的线程安全 我们知道SimpleDateFormat在多线程下是存在线程安全问题的，那么将SimpleDateFormat作为每个线程的局部变量的副本就是每个线程都拥有..的SimpleDateFormat，就不存在线程安全问题了。 publicclassSimpleDateFormatDemo{ privatestaticfinalStringDATE_FORMAT="yyyy-MM-ddHH:mm:ss"; privatestaticThreadLocalthreadLocal=newThreadLocal<>(); /***获取线程的变量副本，如果不覆盖initialValue.法，第.次get将返回null,故需要创建.个DateFormat，放.threadLocal中 *@return/* publicDateFormatgetDateFormat() {DateFormatdf=threadLocal.get();if(df==null){ df=newSimpleDateFormat(DATE_FORMAT); threadLocal.set(df);}returndf; } publicstaticvoidmain(String[]args) {SimpleDateFormatDemoformatDemo=newSimpleDateFormatDemo(); MyRunnablemyRunnable1=newMyRunnable(formatDemo);MyRunnablemyRunnable2=newMyRunnable(formatDemo);MyRunnablemyRunnable3=newMyRunnable(formatDemo); Threadthread1=newThread(myRunnable1);Threadthread2=newThread(myRunnable2);Threadthread3=newThread(my Runnable3);thread1.start();thread2.start();thread3.start(); } publicstaticclassMyRunnableimplementsRunnable{ privateSimpleDateFormatDemodateFormatDemo; publicMyRunnable(SimpleDateFormatDemodateFormatDemo) {this.dateFormatDemo=dateFormatDemo;} @Override publicvoidrun(){ } System.out.println(Thread.currentThread().getName()+"当前时 间："+dateFormatDemo.getDateFormat().format(newDate())); }} 运.结果：

源码分析 ThreadLocalMap ThreadLocalMap内部是利.Entry来进.key-value的存储的。 staticclassEntryextendsWeakReference<ThreadLocal>{ /**ThevalueassociatedwiththisThreadLocal.*/ Objectvalue; Entry(ThreadLocalk,Objectv){super(k); value=v; }} 上.源码中key就是ThreadLocal，value就是值，Entry继承WeakReference，所以Entry对应key的引.（ThreadLocal实例）是.个弱引.。 set(ThreadLocalkey,Objectvalue) /** *Setthevalueassociatedwithkey. * *@paramkeythethreadlocalobject *@paramvaluethevaluetobeset */privatevoidset(ThreadLocalkey,Objectvalue){Entry[]tab=table;intlen=tab.length;//根据ThreadLocal的散列值，查找对应元素在数组中的位置 inti=key.threadLocalHashCode&(len-1);//采.线性探测法寻找合适位置 for(Entrye=tab[i];e!=null;e=tab[i=nextIndex(i,len)]){ThreadLocalk=e.get();//key存在，直接覆盖 if(k==key) {e.value=value;return;}//key==null，但是存在值（因为此处的e!=null），说明之前的ThreadLocal对象已经被回收了 if(k==null){replaceStaleEntry(key,value,i);return;} }//ThreadLocal对应的key实例不存在，new.个 tab[i]=newEntry(key,value);intsz=++size; //清楚陈旧的Entry(key==null的) //如果没有清理陈旧的Entry并且数组中的元素.于了阈值，则进.rehash if(!cleanSomeSlots(i,sz)&&sz>=threshold)rehash();} 这个set操作和集合Map解决散列冲突的.法不同，集合Map采.的是链地址法，这.采.的是开放定址法（线性探测）。set().法中的replaceStaleEntry()和cleanSomeSlots()，这两个.法可以清除掉key==null的实例，防.内存泄漏。 getEntry() privateEntrygetEntry(ThreadLocalkey){inti=key.threadLocalHashCode&(table.length-1);Entrye=table[i];if(e!=null&&e.get()==key) returne;else returngetEntryAfterMiss(key,i,e);} 由于采.了开放定址法，当前keu的散列值和元素在数组中的索引并不是..对应的，.先取.个猜测数（key的散列值），如果所对应的key是我们要找的元素，那么直接返回，否则调.getEntryAfterMiss privateEntrygetEntryAfterMiss(ThreadLocalkey,inti,Entrye) {Entry[]tab=table;intlen=tab.length; while(e!=null){ThreadLocal<?>k=e.get();if(k==key) returne;if(k==null)expungeStaleEntry(i); else i=nextIndex(i,len); e=tab[i];}returnnull; } 这..直在探测寻找下.个元素，知道找的元素的key是我们要找的。这.当key==null时，调.expungeStaleEntry有利于GC的回收，.于防.内存泄漏。 ThreadLocal为什么会内存泄漏 ThreadLocalMap的key为ThreadLocal实例，他是.个弱引.，我们知道弱引.有利于GC的回收，当key==null时，GC就会回收这部分空间，但value不.定能被回收，因为他和CurrentThread之间还存在.个强引.的关系。 由于这个强引.的关系，会导致value.法回收，如果线程对象不消除这个强引.的关系，就可能会出现OOM。有些时候，我们调.ThreadLocalMap的remove().法进.显式处理。

总结 ThreadLocal不是.来解决共享变量的问题，也不是协调线程同步，他是为了.便各线程管理..的状态.引.的.个机制。每个ThreadLocal内部都有.个ThreadLocalMap,他保存的key是ThreadLocal的实例，他的值是当前线程的局部变量的副本的值。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. SSM实现.付宝扫码.付功能（图.详解）

2. Spring Boot微信点餐系统

3. Spring MVC到 Spring Boot的简化之路
4.

12306的架构到底有多.逼?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试准备-分布式系统CAP理论 pkwendaJava后端 2019-12-29 点击上.Java后端，选择设为星标优质.章，及时送达

链接|github.com/pkwenda/Blog/issues/28 CAP理论是.个被说烂了也听烂了的话题，但是还是选择花点时间做整理，做些横向拓展，加点..的感受，认认真真讨论.下。之前看阮.峰.师写的.章不错，我之前觉得总有点难懂，现在我觉得是有点问题 我觉得阮.师说的Availability理解不对，分布式和具体「哪台」没什么关系。Partitiontolerance的解释也没让我豁然开朗。 CAP简介 1998年，有个哥们提出了分布式系统的三个指标： CAP(Consistency、Availablity、Partition-tolerance)其中： Consistency

.致性 这哥们说，分布式系统所有数据备份，在「同.时刻」，必须是「同样的」。

.如图床服务：后台是三台机器A、B、C做.撑，上传.张图.，通过.关最终可能是A处理的，随后「.上」请求这张图.，通过.关轮询算法，这个请求极有可能不落在A上了，那么他说：不管落在B、C哪个机器上，该图.必须存在并返回，否则，就不是『.致性』 如果就有多个分区要执.写操作，如果分布式存储系统「分.」同步、或集群系统的「主从」同步总会有个先后顺序，在这个先后顺序中，就产.了不.致的问题，这时候你：

要么接受同.个数据在各个存放点上同步过程中的「暂时」不.致。要么你要「强.致性」，那么同步操作就会影响你的「可.性」。 常.的.致性的级别整理，想看的可以看看 Availability

可.性 这哥们说:在集群中.部分节点故障后，集群整体是否还能响应客.端的读写请求，即使集群中的某个结点宕机了，依旧不影响你的任何请求。 这条.常好理解，就是字.意思 Partitiontolerance

分区容忍性 这哥们说：如果集群中的机器被分成了两部分，这两部分不能互相通信，系统是否能继续正常.作这个CAP中的P是最误导我的，以.于今天我对CAP依旧.法侃侃.谈，特此整理的原因。 我对「分区容忍性」的疑问 很多.说".致性，可.性，分区容忍性，只能任意选择其中两个"，乍.看是没错，但是AC满.吗？普遍到「百度百科」原话也是这样写的： CAP原则的精髓就是要么AP，要么CP，要么AC，但是不存在CAP。 我的疑问：保证了AC，那么P就没办法保证，试想：机器间不能通信，.络分区，如何保证数据的.致性？如何同步数据？APCP我都能理解，平常.作中也都能看到影.：.如AP（Eureka）选择了可.性、CP（Zookeeper、HDFS）都选择了.致性，但是AC的模型在哪.？如何做到不会通信失败呢？有.说：CAP理论作者太教条！只要各地的分布式机器保持.个量级，且把.法通信的机器踢下去，那就是CAP！.有.说：那不.，如果.络抖动，你踢的太多，数据分.达不到要求也不.！参考资料□ clouderafacebook原.（cloudera官.原.已失效） 参考上.翻译 作者的意思是，只能CP或者AP因为由于.络问题，将系统的成员隔离成了2个区域，互相.法知道对.的状态，这在分布式环境下是.常常.的。所以说只能从C和A中选.个。所以P必选。要建.个永远不发.多相关故障的.络，对于分布式系统来说是不切实际的。所以设计者必须在.致性（C）和可.性（A）之间做选择。 当然以上都是教条的按照CAP理论进.的讨论，其实在现实中不需要如此纠结。 总结 在分布式环境下.，P是铁定存在的，也就是只要我们有多台机器，那么.络隔离分区就.定不可避免，所以在设计系统的时候我们就要选择到底是设计的是AP系统还是CP系统，但实际上，我们只要深.理解下CAP，就会发现其实有时候系统设计上.没必要这么纠结，主 要表现在： .络分区出现的概率虽然市场发.，但是我们可以感知.跳强.剔除下线，将流量平均分往其他节点，毕竟谁也不能保证.络百分百稳定，不出现.络分区。.然.然的没必要刻意A、C中选.个，.是可以都做得不错。也就是「百度百科」说的AC系统。 CAP..规定A是100%的可.性，但实际上，我们只需要提供highavailability，也就是像.关.样，全年不宕不可能，但是满.99.99%或者99.999%等.个9就可以了。 收获 对CAP的P.有了进.步的概念，有了.些新的理解，.看了下Paxos，实在是.较复杂本来想也写写，但是理解的还不好怕写不好下次再说。 参考资料 阮.峰.师：https://www.ruanyifeng.com/blog/2018/07/cap.html clouderacap：https://pt-br.facebook.com/notes/cloudera/cap-confusion-problems-with-partition-tolerance/385772907002/ clouderacap翻译：http://zzyongx.github.io/blogs/cap-confusion-problems-with-partition-tolerance.html 常.的.致性的级别：https://github.com/pkwenda/Blog/blob/master/snippets/consistency.md -END - 推荐阅读 1. Java线程有哪些不太为.所知的技巧与.法?

2.关于 CPU的.些基本知识总结

3.

发布没有答案的.试题，都是耍流氓

4.什么是.致性 Hash算法?

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试字节跳动，我被怼了。 三级狗Java后端 2019-12-09 点击上.Java后端，选择设为星标优质.章，及时送达

作者.三级狗链接|zhihu.com/question/31225105/answer/582508111 本.来.知乎知名博主「三级狗」的回答，看完问学.希望有所收获，同时欢迎点击阅读原.关注该博主。 .们都说，这个世界上有两种.注定单.，.种是太优秀的，另.种是太平凡的。 我.听呀？那我这岂不是就不优秀了吗，于是毅然决然和.朋友分了.。 .们都说，互联.寒冬来了，这个时候还在..积招.的公司，必然是.逼的公司。.这个时候勇敢跳槽的.，必然是.逼的.。 于是2019年1.25.下午14:00，我开始了字节跳动的社招.试。 为了这天，我前.天排..的队理了个利落的发型，胡.刮得...净。穿上崭新的新百伦999，连袜.都是崭新的NB，墨绿..装裤配酒红.卫.，外.再搭.件精致的...绒.坎肩，准时准点出现在电脑屏幕前，准备开始视频.试。 去.试头条，你最好有点.理准备。我.试过头条的好兄弟给了我.句忠告。不过邮件对考查内容写的.清.楚，所以..，我并没有做太强的.理建设，事实证明我可能对通.型业务和软性技能的理解还不够透彻。

时间到了，画..出现.个.积很.的房间，.试官坐在..调节姿势和座椅靠背。这不是我第.次，但我却.第.次还要紧张。.试官跟我的.位闷骚朋友很像，.相、.质、说话声.如出.辙，我.乎差点问他这些年跟郑州那个.孩.还有没有联系，孩.到底是不是你的..... 确认过眼神，我们在初次..的紧张和局促中正式开始，前戏的部分.同.异，.我介绍，公司职能，项.简介。.作坊.出来的.猿，项.经验并不出彩，没有.并发，没有分布式，前端后台服务器和数据库部署在同.台机器上。为了不显的太low.假装不经意提了提..在GitHub上有..的开源框架，但是他并不care。我的花样耍完了，就到了他肆意挥舞..鞭的环节。 “我看你项.上都是偏独.开发，或者你.个.在项.中承担了很.的.重。那你能不能说说你们的项.在部署的时候.的是怎样的架构？” “主要就是Django部署那.套，nginx+uwsgi+Django+mysql。” “那你详细说.下nginx在部署的时候有哪些必要的配置？” “这个我在nginx.配置了端.的转发，对外监听80或443，然后转发到内部uwsgi的端.，由uwsgi来处理业务请求，部署 Django项.。”显然我对nginx的认知仅仅局限在配置个端.反向代理，偷换概念企图蒙混过关。 “不是，我的意思是nginx当中不是有很多参数配置嘛，你在项.部署的时候那些参数是必要的配置，有哪些可以调优” 对.识破了我的.伎俩，并不给我蒙混过关的机会，.奈我只得低头认怂，表.并没有.过其他配置，出师未捷就挨了当头. 棒。 “那你再说说uwsgi它的.作原理吧，它的底层是怎么.作的。” 因为紧张的缘故，原本不会的知识，.下.变得还是不会，囫囵吞枣地讲它代理.个服务端.，分配不同的线程处理客.端浏览器的请求。 “那他的底层是怎么实现的？看来我如果还没求饶，他就不打算放过我。 .....（这.省略.些不清楚、不知道、忘记了之类的词藻，保留.丝尊严。） 接下来是关于redis哲学三连“是什么？为什么？怎么.？” 我把肚..仅有的关于Redis的.滴墨.挤成了三滴，仍然没有给出他想要的。继续追问Redis的数据存储.式，操作.法，读写操作在底层都是如何实现。 啊.，好深。 除了低头委屈说不会，内.的我已经意识到了问题的严重性，平常.试你只要.喊不.，对.就会体贴地换.个.向继续深.，这次我都快被搞哭了，他居然换了个姿势继续往深怼，这谁顶得住啊..... 到了这.步我的.理防线基本宣布告罄（qìng）。往后.问了关系型数据库表的存储结构，我隐约记得是毕加索还是毕加树来 着，完后.追问到索引的实现原理，创建.个索引怎么它就能加快查询效率。 其实.试官的套路.多如此，每个技术都尽可能的深.，深到你不会为.，并不是.得要把你折磨的不能.理，.是为了考察你的技术.平到底如何。但是事到如今我的.脑已经.法给..完成.理建设了，摧枯拉朽般智商情商性商逆商全线崩盘。 当然这也不能全怪我，上来就是三个.闷棍，就是李云.来也让旅.打成懵逼了，哪.还能腾出精.开意.利炮。 下.个节.到了数据结构，.试官稳准狠.步到位点到了我的敏感地带：堆。什么是最.堆？什么是最.堆？在堆中怎么插..个元素？ 这个问题上值得庆幸的是，我居然急中.智，.使神差地.了四种不同的.式表达我不会，每种.式都尴尬.不委婉，并且还不重样。好..试.波.不能让.家觉得我..是处，多少还算有点.笔.... 其实我平时稍微多看哪怕.眼，知道堆的实现.式是平衡.叉树，这.连串的问题不.于答的这么惨，.试的时候脑..记得看过栈和队列，完全不知道堆是怎么实现，下来才知道就是个平衡.叉树。 这个时候的我已经被折磨的..俱疲了，.试已经持续了将近四.分钟，嘴开始打漂，情绪紧张思维也难以集中，回答中开始.量出现我感觉、我猜、可能、应该是等.危词汇。 死亡轰炸还没有结束，接着是MQ，我所使.的MQ的技术选型，为什么选它，项.中如何应.，最后..例外，它底层是如何实现？怎么保证的稳定的消费者.产者队列？回答的中间穿插着我不.信的连接词“嗯.啊.哼.哦.”。每次我招架不住求饶喊着不要不要的时候，我都能看到.试官嘴.狡黠（xiá）的微笑，由内.外的快感浮现在脸上憋都憋不住。..暗暗发誓迟早有.天，我也要在上.。 最后以.道算法编程题结束，如何给.个双向链表排序？我失去控制的.

头.作主张地把话锋导向了快速排序，那，你来写.个双向链表的快速排序吧，这个视频..上旁边有.个编辑器，语.你随便，.c++或python都可以，.概10分钟的时间，可以吧。 “那，我.c++试.下吧” 我随即在编辑器..信的def了.个quickSort函数，参数是.个int数组，还有两个int值代表low和high两个档位，函数末尾加上冒号，下..开始缩进四个空格以.尊敬，当然代码段的区分花括号必须得有。 写完这.句，双.这才收到了.脑已经在.分钟前宕机的消息，于是两..摊表..能为.。 时间才过去.分钟，剩下时间我开始拿纸笔低头写字，沉默的.试官抬头看到还以为我在纸上排演算法，可实际上我是在拼尽最后.丝脑.回顾刚才他问我的问题，倒不是为了.试后好好复习，主要是我.会.要写知乎，哈哈，我可真是个.机灵.。 期间.试官也...的代码快速的敲击键盘，听声.猜.概是在吐槽HR，筛选简历的时候.点.，找来.试的这都什么玩意.。 时间差不多了，再.次厚着脸.主动承认..写不出来。实际上，我哪怕正经.c++写.个数组的快速排序也算话，然.当时的 我满.只剩下疲惫，痛苦和羞辱。 .试官终于放我..，.达.个.时的激情视频聊天终于结束，关掉视频的.刹那，随着浑..阵抽搐，整个.瘫软在椅.上。我是谁？我在哪.？刚才发.了什么？ 哦，刚才发.了那种事情。我之前对那种事情还挺向往的，内.对此充满了激情和憧憬，怎么现在，对这件事情.点兴趣也没了？现在只想这么瘫着什么也不想做。要不要点根烟抽抽？哦对了我不抽烟。原来贤者时间脑..这么多想法…… 飘飘忽忽浑浑噩噩，脑海..直回荡着周董的.乐：这感觉已经不对我最后才了解，...不忍翻阅的情节我好累，你沉默看我掉过.次泪多憔悴，.我.碎你受罪你的offer，我不配...！ .试过程和提到的问题.致如此，为了阅读体验有部分艺术修饰，但是内容全部都是真实的。接下来咱们说点正经的。 讲真的现在的我只想把头插到..再也不出来，之前写过超级浓的鸡汤，讲学习.法，被技术号拿去直接顶置到现在；写过.赞，讲连连看的外挂，到现在还有.私信问我能不能做外挂的私活.，.天上千的利润；作为培训讲师讲培训机构内幕，有上海北京的培训机构给到..时1k了我不想去，想趁年轻进..做技术。在.家.顿商业吹捧之下就真的以为..是个.佬了。 有极少数的.在评论区怼我说花拳绣腿，说我.本渣渣热衷于误..弟，我都不以为然，..想着等我有.天拿着..的offer回来证明..！到今天总算知道他们说的是对的，打嘴炮和真有能耐是不.样的。之所以.直没有.佬来怼，可能是因为.佬不刷知乎。 今天这轮.试，算是被扒了个体.完肤，可以说是程序员.试的反.典型，当然我也不怕说出来，菜就是菜，吹.就是吹.，努.把吹过的.早.兑现，迟早有.天能成真正的.佬！ 这次作为loser再说说在求职前期准备过程中暴露出的问题，望.家引以为戒。 1、尽量早早做好准备 .试不应该是准备好了才去，.是时刻都准备好了。如果.要说从什么时候开始准备离职跳槽，我的建议是半年。别像我.样，.试了才开始看算法和数据结构。 2、先找.公司..次进.状态，再投.公司.举拿下。 我蠢到.开始就动.了我所有的.脉，投了bat及其他.线..所有的内推。算上字节跳动已经是我.掉的第三个响当当..了，踩着..当垫脚.攒.试经，除了觉得..蠢，我还觉得..真特么.逼。 3、你的简历，就是复习.纲 总有.拿.上找到的拿了bato.er的.提供的复习.纲复习，结果发现并没啥卵.。我这次字节的.试，他提出的所有的问题全部是基于我简历中的技能清单。 我认为很多.但凡能做到简历中技能清单描述的样.，就已经算了不起了。如果简历中感觉.料可写？我推荐.个好办法： 1.听说过名字，就写了解； 2.跟着帖.写过demo，就写熟悉；

3.

项.当中.过，就写熟练掌握；

4.

项.当中经常.，就写精通；

我就是这么写的，结果很显然，出来混，总是要还的，敢装逼，就活该被.凌辱。 4、警惕舒适区 不得不承认我在现公司呆的真的.常舒服，领导.常器重，项.和授课两不耽误，去.学实训总是受到学.的追捧，..时不时还能接到价格公道的私活.。 之所以想.，...是因为.中对..和技术的执念，另...是因为现在已经到了给..规划的时间，碰巧赶上互联.寒冬是没有办法的事，但是我仍然坚信真正.逼的.是不会被寒冬所阻拦。 最近的.波..试直接把我打到怀疑..了，.试完后双..神的望着天花板不知道该如何评价...。授课到现在带过的学.虽然不多，但是送进..线..的没有..个也有..个了（我们机构不造假学历，都是实习.进的），怎么我这个.师找个.作这么费劲？是道德的沦丧还是.性.扭曲？是..真的飘了还是我确实拿不动.了？ “如果程序员对职场感到迷茫，对眼下的舒适感到不安，我建议他出去.试，不.得要.，但是你要出去，听听市场对..的评价。” 这句话是在.个求职公众号.看到的，我觉得不光程序员，任何职场.都该深以为然。我对职场虽不迷茫，但是.试.波.就发现，我在毫.意识的情况下，在.作三年这个节点成.上已经被.甩下..截，更要命的是对基础知识的掌握程度完全.不上应届毕业的实习.。 往后...赶紧..实实看书，踏踏实实做事.，早.兑现..曾经吹过的.逼。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。↓扫描.维码进群↓

推荐阅读 .为什么你学不会递归？ .MySQL：Left Join避坑指南 .AJAX请求真的不安全么？ ..个..不主动联系你还有机会吗？ .团队开发中 Git最佳实践

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

.试官最常问的SpringCloud.试题 Java后端 2.2.

作者|moakun blog.csdn.net/moakun/article/details/82817757 今天跟.家分享下SpringCloud常..试题的知识。 1什么是Spring Cloud？ Spring cloud流应.程序启动器是基于SpringBoot的Spring集成应.程序，提供与外部系统的集成。Spring cloud Task，.个.命周期短暂的微服务框架，.于快速构建执.有限数据处理的应.程序。 2使.Spring Cloud有什么优势？ 使.Spring Boot开发分布式微服务时，我们.临以下问题:与分布式系统相关的复杂性-这种开销包括.络问题，延迟开销，带宽问题，安全问题。服务发现-服务发现.具管理群集中的流程和服务如何查找和互相交谈。它涉及.个服务.录，在该.录中注册服 务，然后能够查找并连接到该.录中的服务。冗余-分布式系统中的冗余问题。负载平衡--负载平衡改善跨多个计算资源的.作负荷，诸如计算机，计算机集群，.络链路，中央处理单元，或磁 盘驱动器的分布。性能-问题由于各种运营开销导致的性能问题。部署复杂性-Devops技能的要求。 3服务注册和发现是什么意思？Spring Cloud如何实现？ 当我们开始.个项.时，我们通常在属性.件中进.所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，.某些位置可能会发.变化。.动更改属性可能会产.问题。Eureka服务注册和发现可以在这种情况下提供帮助。由于所有服务都在Eureka服务器上注册并通过调.Eureka服务器完成查找，因此.需处理服务地点的任何更改和处理。 4负载平衡的意义什么？ 在计算中，负载平衡可以改善跨计算机，计算机集群，.络链接，中央处理单元或磁盘驱动器等多种计算资源的.作负载分布。负载平衡旨在优化资源使.，最.化吞吐量，最.化响应时间并避免任何单.资源的过载。使.多个组件进.负载平衡.不是单个组件可能会通过冗余来提.可靠性和可.性。负载平衡通常涉及专.软件或硬件，例如多层交换机或域名系统服务器进程。 5什么是Hystrix？它如何实现容错？ Hystrix是.个延迟和容错库，旨在隔离远程系统，服务和第三.库的访问点，当出现故障是不可避免的故障时，停.级联故障并在复杂的分布式系统中实现弹性。通常对于使.微服务架构开发的系统，涉及到许多微服务。这些微服务彼此协作。思考以下微服务

假设如果上图中的微服务9失败了，那么使.传统.法我们将传播.个异常。但这仍然会导致整个系统崩溃。 随着微服务数量的增加，这个问题变得更加复杂。微服务的数量可以.达1000.这是hystrix出现的地.我们将使.Hystrix在这种情况下的Fallback.法功能。我们有两个服务employee-consumer使.由employee-consumer公开的服务。 简化图如下所.

现在假设由于某种原因，employee-producer公开的服务会抛出异常。我们在这种情况下使.Hystrix定义了.个回退.法。这种后备.法应该具有与公开服务相同的返回类型。如果暴露服务中出现异常，则回退.法将返回.些值。 6什么是Hystrix断路器？我们需要它吗？ 由于某些原因，employee-consumer公开服务会引发异常。在这种情况下使.Hystrix我们定义了.个回退.法。如果在公开服务中发.异常，则回退.法返回.些默认值。

如果.rstPage method()中的异常继续发.，则Hystrix电路将中断，并且员.使.者将.起跳过.rtsPage.法，并直接调.回退.法。断路器的.的是给第...法或第...法可能调.的其他.法留出时间，并导致异常恢复。可能发.的情况是，在负载较.的情况下，导致异常的问题有更好的恢复机会。

7什么是Net. ix Feign？它的优点是什么？ Feign是受到Retro.t，JAXRS-2.0和WebSocket启发的java客.端联编程序。Feign的第.个.标是将约束分.的复杂性统.到http apis，.不考虑其稳定性。在employee-consumer的例.中，我们使.了employee-producer使.REST模板公开的REST服务。 但是我们必须编写.量代码才能执.以下步骤:使.功能区进.负载平衡。获取服务实例，然后获取基本URL。利.REST模板来使.服务。前.的代码如下: 1 @Controller
2 public class ConsumerControllerClient {
3 @Autowired
4 private LoadBalancerClient loadBalancer;
5 public void getEmployee() throws RestClientException, IOException
6 {
7 ServiceInstance serviceInstance=loadBalancer.choose("employee-producer");
8 System.out.println(serviceInstance.getUri());
9 String baseUrl=serviceInstance.getUri().toString();
10 baseUrl=baseUrl+"/employee"

之前的代码，有像NullPointer这样的例外的机会，并不是最优的。我们将看到如何使.Net. ix Feign使呼叫变得更加轻松和清洁。如果Net. ix Ribbon依赖关系也在类路径中，那么Feign默认也会负责负载平衡。 8什么是Spring Cloud Bus？我们需要它吗？ 考虑以下情况：我们有多个应.程序使.Spring Cloud Con.g读取属性，.Spring Cloud Con.g从GIT读取这些属性。 下.的例.中多个员..产者模块从Employee Con.g Module获取Eureka注册的财产。

如果假设GIT中的Eureka注册属性更改为指向另.台Eureka服务器，会发.什么情况。在这种情况下，我们将不得不重新启动服务以获取更新的属性。 还有另.种使.执.器端点/刷新的.式。但是我们将不得不为每个模块单独调.这个url。例如，

如果EmployeeProducer1部署在端.8080上，则调.http：// localhost：8080 / refresh。同样对于EmployeeProducer2 http：// localhost：8081 / refresh等等。这.很.烦。这就是Spring Cloud Bus发挥作.的地.。

Spring CloudBus提供了跨多个实例刷新配置的功能。因此，在上.的.例中，如果我们刷新EmployeeProducer1，则会.动刷新所有其他必需的模块。如果我们有多个微服务启动并运.，这特别有.。这是通过将所有微服务连接到单个消息代理来实现的。.论何时刷新实例，此事件都会订阅到侦听此代理的所有微服务，并且它们也会刷新。可以通过使.端点/总线/刷新来实现对任何单个实例的刷新。 9 SpringCloud和Dubbo SpringCloud和Dubbo都是现在主流的微服务架构 SpringCloud是Apache旗下的Spring体系下的微服务解决.案Dubbo是阿.系的分布式服务治理框架从技术维度上,其实SpringCloud远远的超过Dubbo,Dubbo本.只是实现了服务治理,.SpringCloud现在以 及有21个.项.以后还会更多所以其实很多.都会说Dubbo和SpringCloud是不公平的但是由于RPC以及注册中.元数据等原因,在技术选型的时候我们只能.者选其.,所以我们常常为.他俩来对.服务的调..式Dubbo使.的是RPC远程调.,.SpringCloud使.的是 Rest API,其实更符合微服务官.的 定义服务的注册中.来看,Dubbo使.了第三.的ZooKeeper作为其底层的注册中.,实现服务的注册和发 现,SpringCloud使.Spring Cloud Net.ixEureka实现注册中.,当然SpringCloud也可以使.ZooKeeper实现,但.般我们不会这样做服务.关,Dubbo并没有本.的实现,只能通过其他第三.技术的整合,.SpringCloud有Zuul路由.关,作为路 由服务器,进.消费者的请求分发,SpringCloud还.持断路器,与git完美集成分布式配置.件.持版本控制,事务总线实现配置.件的更新与服务.动装配等等.系列的微服务架构要素 10技术选型 .前国内的分布式系统选型主要还是Dubbo毕竟国产,.且国内.程师的技术熟练程度.,并且Dubbo在其他维度上的缺陷可以由其他第三.框架进.集成进.弥补 .SpringCloud.前是国外.较流.,当然我觉得国内的市场也会慢慢的偏向SpringCloud,就连刘军作为Dubbo重启的负责.也发表过观点,Dubbo的发展.向是积极适应SpringCloud.态,并不是起冲突 11 Rest和RPC对. 其实如果仔细阅读过微服务提出者.丁福勒的论.的话可以发现其定义的服务间通信机制就是Http Rest RPC最主要的缺陷就是服务提供.和调..式之间依赖太强,我们需要为每.个微服务进.接.的定义,并通过持续继承发布,需要严格的版本控制才不会出现服务提供和调.之间因为版本不同.产.的冲突 .REST是轻量级的接.,服务的提供和调.不存在代码之间的耦合,只是通过.个约定进.规范,但也有可能出现.档和接.不.致.导致的服务集成问题,但可以通过swagger.具整合,是代码和.档.体化解决,所以REST在分布式环境下.RPC更加灵活 这也是为什么当当.的DubboX在对Dubbo的增强中增加了对REST的.持的原因 12 .档质量和社区活跃度 SpringCloud社区活跃度远.于Dubbo,毕竟由于梁.团队的原因导致Dubbo停.更新迭代五年,.中.型公司.法承担技术开发的成本导致Dubbo社区严重低落,.SpringCloud异军突起,迅速占领了微服务的市场,背靠Spring混的...起 Dubbo经过多年的积累.档相当成熟,对于微服务的架构体系各个公司也有稳定的现状 13 SpringBoot和SpringCloud SpringBoot是Spring推出.于解决传统框架配置.件冗余,装配组件繁杂的基于Maven的解决.案,旨在快速 搭建单个微服务 .SpringCloud专注于解决各个微服务之间的协调与配置,服务之间的通信,熔断,负载均衡等,技术维度并相同,并且SpringCloud是依赖于SpringBoot的,.SpringBoot并不是依赖与SpringCloud,甚.还可以和Dubbo进.优秀的整合开发 总结: SpringBoot专注于快速.便的开发单个个体的微服务 SpringCloud是关注全局的微服务协调整理治理框架,整合并管理各个微服务,为各个微服务之间提供,配置管理, 服务发现,断路器,路由,事件总线等集成服务 SpringBoot不依赖于SpringCloud,SpringCloud依赖于SpringBoot,属于依赖关系 SpringBoot专注于快速,.便的开发单个的微服务个体,SpringCloud关注全局的服务治理框架 14 Eureka和ZooKeeper都可以提供服务注册与发现的功能,请说说两个的区别 1.ZooKeeper保证的是CP,Eureka保证的是APZooKeeper在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可.的Eureka各个节点是平等关系,只要有.台Eureka就可以保证服务可.,.查询到的数据并不是最新的.我保护机制会导致： Eureka不再从注册列表移除因.时间没收到.跳.应该过期的服务Eureka仍然能够接受新服务的注册和查询请求,但是不会被同步到其他节点(.可.)当.络稳定时,当前实例新的注册信息会被同步到其他节点中(最终.致性)Eureka可以很好的应对因.络故障导致部分节点失去联系的情况,.不会像ZooKeeper.样使得整个注册系统 瘫痪 2. ZooKeeper有Leader和Follower..,Eureka各个节点平等

3.

ZooKeeper采.过半数存活原则,Eureka采..我保护机制解决分区问题

4.

Eureka本质上是.个.程,.ZooKeeper只是.个进程

15微服务之间是如何独.通讯的 微服务通信机制 系统中的各个微服务可被独.部署，各个微服务之间是松耦合的。每个微服务仅关注于完成.件任务并很好地完 成该任务。 围绕业务能.组织服务、.动化部署、智能端点、对语.及数据的去集中化控制。 将组件定义为可被独.替换和升级的软件单元。 以业务能.为出发点组织服务的策略。 倡导谁开发，谁运营的开发运维.体化.法。 RESTfulHTTP协议是微服务架构中最常.的通讯机制。 每个微服务可以考虑选.最佳.具完成(如不同的编程语.)。 允许不同微服务采.不同的数据持久化技术。 微服务.常重视建.架构及业务相关指标的实时监控和.志机制，必须考虑每个服务的失败容错机制。 注重快速更新，因此系统会随时间不断变化及演进。可替代性模块化设计。

微服务通信.式： 同步：RPC，REST等 异步：消息队列。要考虑消息可靠传输、.性能，以及编程模型的变化等。 消息队列中间件如何选型 1.协议：AMQP、STOMP、MQTT、私有协议等。2.消息是否需要持久化。 3.吞吐量。 4. .可..持，是否单点。

5.

分布式扩展能.。

6.

消息堆积能.和重放能.。7.开发便捷，易于维护。8.社区成熟度。 RabbitMQ是.个实现了AMQP(.级消息队列协议)协议的消息队列中间件。RabbitMQ.持其中的最多.次和

最少.次两种。.易蜂巢平台的服务架构，服务间通过RabbitMQ实现通信。 16什么是服务熔断?什么是服务降级 在复杂的分布式系统中,微服务之间的相互调.,有可能出现各种各样的原因导致服务的阻塞,在.并发场景下,服务的阻塞意味着线程的阻塞,导致当前线程不可.,服务器的线程全部阻塞,导致服务器崩溃,由于服务之间的调.关系是同步的,会对整个微服务系统造成服务雪崩，为了解决某个微服务的调.响应时间过.或者不可.进.占.越来越多的系统资源引起雪崩效应就需要进.服务熔断和服务降级处理。 所谓的服务熔断指的是某个服务故障或异常.起类似显.世界中的"保险丝"当某个异常条件被触发就直接熔断整个服务，.不是.直等到此服务超时。 服务熔断就是相当于我们电闸的保险丝,.旦发.服务雪崩的,就会熔断整个服务,通过维护.个..的线程池,当线程达到阈值的时候就启动服务降级,如果其他请求继续访问就直接返回fallback的默认值。 17微服务的优缺点分别是什么?说下你在项.开发中碰到的坑 优点:每.个服务.够内聚,代码容易理解开发效率提.,..个服务只做.件事微服务能够被.团队单独开发微服务是松耦合的,是有功能意义的服务可以.不同的语.开发,.向接.编程易于与第三.集成微服务只是业务逻辑的代码,不会和HTML,CSS或者其他界.组合开发中,两种开发模式前后端分离全栈.程师 可以灵活搭配,连接公共库/连接独.库 缺点: 分布式系统的负责性;多服务运维难度,随着服务的增加,运维的压.也在增.;系统部署依赖;服务间通信成本;数据.致性;系统集成测试;性能监控. 18你所知道的微服务技术栈有哪些?请列举.. 多种技术的集合体 我们在讨论.个分布式的微服务架构的话,需要哪些维度 维度(SpringCloud) 1服务开发 2 SpringBoo 3 t 4 Sprin 5 g 6 SpringMV 7 C 8服务配置与管理 9 10 Netfilx公司的Archaiusm,阿.的Diamond 11服务注册与发 12现 13 Eureka,ZooKeeper 14服务调 15 . 16 Rest,RPC,gRPC 17服务熔断 18器 19 Hystri 20 x 21服务负载均 22衡 23 Ribbon,Nginx 24服务接.调 . Feig n 消息队列 Kafka,RabbitMq,ActiveMq 服务配置中.管理 SpringCloudConfing服务路由(API.关) Zuu

推荐阅读 1. 村.部们的智慧 2.IntelliJ发布2020RoadMap

3.

互联.公司的抗疫情.动！ 4.如何获取靠谱的新型冠状病毒疫情

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试官问：Redis内存满了怎么办？我想不到！ 千.Java后端 2019-11-27 点击上.Java后端，选择设为星标优质.章，及时送达

来源：https://tinyurl.com/y6qctca6 Redis是基于内存的key-value数据库，因为系统的内存..有限，所以我们在使.Redis的时候可以配置Redis能使.的最.的内存..。 1、通过配置.件配置 通过在Redis安装.录下.的redis.conf配置.件中添加以下配置设置内存..

redis的配置.件不.定使.的是安装.录下.的redis.conf.件，启动redis服务的时候是可以传.个参数指定redis的配置.件的 2、通过命令修改 Redis.持运.时通过命令动态修改内存..如果不设置最.内存..或者设置最.内存..为0，在64位操作系统下不限制内存..，在32位操作系统下最多使.3GB

内存 Redis的内存淘汰 既然可以设置Redis最.占.内存..，那么配置的内存就有.完的时候。那在内存.完的时候，还继续往Redis..添加数据不就没内存可.了吗？实际上Redis定义了.种策略.来处理这种情况： noeviction(默认策略)：对于写请求不再提供服务，直接返回错误（DEL请求和部分特殊请求除外） allkeys-lru：从所有key中使.LRU算法进.淘汰 volatile-lru：从设置了过期时间的key中使.LRU算法进.淘汰 allkeys-random：从所有key中随机淘汰数据 volatile-random：从设置了过期时间的key中随机淘汰volatile-ttl：在设置了过期时间的key中，根据key的过期时间进.淘汰，越早过期的越优先被淘汰 当使.volatile-lru、volatile-random、volatile-ttl这三种策略时，如果没有key可以被淘汰，则和noeviction.样返回错误 如何获取及设置内存淘汰策略 获取当前内存淘汰策略：

通过配置.件设置淘汰策略（修改redis.conf.件）：

通过命令修改淘汰策略：

LRU算法 什么是LRU? 上.说到了Redis可使.最.内存使.完了，是可以使.LRU算法进.内存淘汰的，那么什么是LRU算法呢？ LRU(LeastRecentlyUsed)，即最近最少使.，是.种缓存置换算法。在使.内存作为缓存的时候，缓存的...般是固定的。当缓存被占满，这个时候继续往缓存..添加数据，就需要淘汰.部分.的数据，释放内存空间.来存储新的数据。这个时候就可以使.LRU算法了。其核.思想是：如果.个数据在最近.段时间没有被.到，那么将来被使.到的可能性也很.，所以就可以被淘汰掉。 使.java实现.个简单的LRU算法 public class LRUCache <k,v>{ //容量 private int capacity; //当前有多少节点的统计 private int count; //缓存节点 private Map <k, Node <k,v>>nodeMap; private Node <k,v>head; private Node <k,v>tail; public LRUCache (int capacity){ if (capacity< 1){ throw new IllegalArgumentException (String .valueOf(capacity)); } this .capacity=capacity; this .nodeMap= new HashMap <>(); //初始化头节点和尾节点，利.哨兵模式减少判断头结点和尾节点为空的代码 Node headNode= new Node (null , ll);nu Node tailNode= new Node (null , ll);nu headNode. next =tailNode; tailNode.pre=headNode; this .head=headNode; this .tail=tailNode; } public void put(kkey,vvalue){ Node <k,v>node=nodeMap. get (key); if (node== null){ if (count>=capacity){ //先移除.个节点 removeNode(); } node= new Node <>(key,value); //添加节点 addNode(node); }lse {e //移动节点到头节点 moveNodeToHead(node); } } public Node <k,v> get (kkey){ Node <k,v>node=nodeMap. get (key); if (node!= null){ moveNodeToHead(node); } return node; } private void removeNode(){ Node node=tail.pre; //从链表..移除 removeFromList(node); nodeMap.remove(node.key);

count--; } private void removeFromList(Node <k,v>node){ Node pre=node.pre; Node next =node. next ; pre. next

next ; next .pre=pre; node. next

null ; node.pre= null ; (} private void addNode Node (<k,v>node){ //添加节点到头部 addToHead(node); nodeMap.put(node.key,node); count++; } private void addToHead(Node <k,v>node){ Node next =head. next ; next .pre=node; node. next

next ; node.pre=head; head. next =node; } public id mvooveNodeToHead(Node <k,v>node){ //从链表..移除 removeFromList(node); //添加节点到头部 addToHead(node); } class Node <k,v>{ kkey; vvalue; Node pre;

上.这段代码实现了.个简单的LUR算法，代码很简单，也加了注释，仔细看.下很容易就看懂。 LRU在Redis中的实现 近似LRU算法 Redis使.的是近似LRU算法，它跟常规的LRU算法还不太.样。近似LRU算法通过随机采样法淘汰数据，每次随机出5（默认）个key，从..淘汰掉最近最少使.的key。 可以通过maxmemory-samples参数修改采样数量：例：maxmemory-samples10maxmenory-samples配置的越.，淘汰的结果越接近于严格的LRU算法 Redis为了实现近似LRU算法，给每个key增加了.个额外增加了.个24bit的字段，.来存储该key最后.次被访问的时间。 Redis3.0对近似LRU的优化 Redis3.0对近似LRU算法进.了.些优化。新算法会维护.个候选池（..为16），池中的数据根据访问时间进.排序，第.次随机选取的key都会放.池中，随后每次随机选取的key只有在访问时间.于池中最.的时间才会放.池中，直到候选池被放满。当放满后，如果有新的key需要放.，则将池中最后访问时间最.（最近被访问）的移除。 当需要淘汰的时候，则直接从池中选取最近访问时间最.（最久没被访问）的key淘汰掉就.。 LRU算法的对. 我们可以通过.个实验对.各LRU算法的准确率，先往Redis..添加.定数量的数据n，使Redis可.内存.完，再往Redis..添加n/2的新数据，这个时候就需要淘汰掉.部分的数据，如果按照严格的LRU算法，应该淘汰掉的是最先加.的n/2的数据。.成如下各LRU算法的对.图（[图.来源]

你可以看到图中有三种不同颜.的点： 浅灰.是被淘汰的数据灰.是没有被淘汰掉的.数据绿.是新加.的数据 我们能看到Redis3.0采样数是10.成的图最接近于严格的LRU。.同样使.5个采样数，Redis3.0也要优于Redis2.8。 LFU算法 LFU算法是Redis4.0..新加的.种淘汰策略。它的全称是LeastFrequentlyUsed，它的核.思想是根据key的最近被访问的频率进.淘汰，很少被访问的优先被淘汰，被访问的多的则被留下来。 LFU算法能更好的表..个key被访问的热度。假如你使.的是LRU算法，.个key很久没有被访问到，只刚刚是偶尔被访问了.次，那么它就被认为是热点数据，不会被淘汰，.有些key将来是很有可能被访问到的则被淘汰了。如果使.LFU算法则不会出现这种情况，因为使..次并不会使.个key成为热点数据。LFU.共有两种策略： volatile-lfu：在设置了过期时间的key中使.LFU算法淘汰key allkeys-lfu：在所有的key中使.LFU算法淘汰数据 设置使.这两种淘汰策略跟前.讲的.样，不过要注意的.点是这两周策略只能在Redis4.0及以上设置，如果在Redis4.0以下设置会报错 问题 最后留.个.问题，可能有的.注意到了，我在.中并没有解释为什么Redis使.近似LRU算法.不使.准确的LRU算法，可以在评论区给出你的答案，.家.起讨论学习。 来源：https://tinyurl.com/y6qctca6 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1...掌握 Redis常.知识点 -图.结合

2..试官：讲.下 Mybatis初始化原理

3.我们再来聊.聊 Java的单例吧

4.我采访了.位 Pornhub .程师

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

.试官问：为什么String的hashCode选择31作为乘.？..波Java后端 2019-12-11 点击上.Java后端，选择设为星标 优质.章，及时送达

作者 |..波 链接 |http://www.tianxiaobo.com 某天，我在写代码的时候，.意中点开了StringhashCode.法。然后.致看了.下hashCode的实现，发现并不是很复杂。但是我从源码中发现了.个奇怪的数字，也就是本.的主.31。这个数字居然不是.常量声明的，所以没法从字.意思上推断这个数字的.途。后来带着疑问和好奇.，到.上去找资料查询.下。在看完资料后，默默的感叹了.句，原来是这样啊。那么到底是哪样呢？在接下来章节.，请.家带着好奇.和我揭开数字31的.途之谜。 选择31的原因 在详细说明StringhashCode.法选择数字31的作为乘.的原因之前，我们先来看看StringhashCode.法是怎样实现的，如下： publicinthashCode(){inth=hash;if(h==0&&value.length>0){ charval[]=value; for(inti=0;i<value.length;i++){ h=31*h+val[i];}hash=h; }returnh;} 上.的代码就是StringhashCode.法的实现，是不是很简单。实际上hashCode.法核.的计算逻辑只有三.，也就是代码中的for循环。我们可以由上.的for循环推导出.个计算公式，hashCode.法注释中已经给出。如下： s[0]31^(n-1)+s[1]31^(n-2)+...+s[n-1] 这.说明.下，上.的 s数组即源码中的 val数组，是 String内部维护的.个 char类型数组。这.我来简单推导.下这个公式： 假设n=3 i=0->h=310+val[0]i=1->h=31(310+val[0])+val[1]i=2->h=31*(31*(310+val[0])+val[1])+val[2] h=3131310+3131val[0]+31val[1]+val[2]h=31^(n-1)val[0]+31^(n-2)val[1]+val[2] 上.的公式包括公式的推导并不是本.的重点，.家了解了解即可。接下来来说说本.的重点，即选择31的理由。从.上的资料来看，.般有如下两个原因： 第. 31是.个不.不.的质数，是作为 hashCode乘.的优选质数之.。另外.些相近的质数，.如37、41、43等等，也都是不错的选择。那么为啥偏偏选中了31呢？请看第.个原因。 第.31可以被JVM优化， 31i=(i<<5)-i。 上.两个原因中，第.个需要解释.下，第.个.较简单，就不说了。下.我来解释第.个理由。.般在设计哈希算法时，会选择.个特殊的质数。.于为啥选择质数，我想应该是可以降低哈希算法的冲突率。.于原因，这个就要问数学家了，我.乎可以忽略的数学.平解释不了这个原因。上.说到，31是.个不.不.的质数，是优选乘.。那为啥同是质数的2和101（或者更.的质数）就不是优选乘.呢，分析如下。 这.先分析质数2。.先，假设 n=6，然后把质数2和n带.上.的计算公式。并仅计算公式中次数最.的那.项，结果是 2^5 = 32，是不是很。所以这.可以断定，当字符串.度不是很.时，.质数2做为乘.算出的哈希值，数值不会很.。也就是说，哈希值会分布在.个较.的数值区间内，分布性不佳，最终可能会导致冲突率上升。 上.说了，质数2做为乘.会导致哈希值分布在.个较.区间内，那么如果..个较.的.质数101会产.什么样的结果呢？根据上.的分析，我想.家应该可以猜出结果了。就是不.再担.哈希值会分布在.个.的区间内了，因为 101^5 = 10,510,100,501。但是要注意的是，这个计算结果太.了。如果. int类型表.哈希值，结果会溢出，最终导致数值信息丢失。尽管数值信息丢失并不.定会导致冲突率上升，但是我们暂且先认为质数101（或者更.的质数）也不是很好的选择。最后，我们再来看看质数31的计算结果：31^5=28629151，结果值相对于 32和 10,510,100,501来说。是不是很n ice，不.不.。 上..了.较简陋的数学.段证明了数字31是.个不.不.的质数，是作为hashCode乘.的优选质数之.。 接下来我会.详细的实验来验证上.的结论，不过在验证前，我们先看看 Stack Over.ow上关于这个问题的讨论，Why does Java's hashCode() in String use 31 as a multiplier?（地址：https://stackover.ow.com/quest ions/299304/why-does-javas-hashcode-in-string-use-31-as-a-multiplier）。其中排名第.的答案引.了《E.ect iveJava》中的.段话，这.也引..下：Thevalue31waschosenbecauseitisanoddprime.Ifitwereevenandthemultiplicationover.owed,informationwouldbelost,asmultiplicationby2isequivalenttoshifting.Theadvantageofusingaprimeislessclear,butitistraditional.Anicepropertyof31isthatthemultiplicationcanbereplacedbyashiftandasubtractionforbetterperformance: 31i==(i<<5)-i .ModernVMsdo thissortofoptimizationautomatically. 简单翻译.下： 选择数字31是因为它是.个奇质数，如果选择.个偶数会在乘法运算中产.溢出，导致数值信息丢失，因为乘.相当于移位运算。选择质数的优势并不是特别的明显，但这是.个传统。同时，数字31有.个很好的特性，即乘法运算可以被移位和减法运算取代，来获取更好的性能： 31i==(i<<5)-i，现代的Java虚拟机可以.动的完成这个优化。 排名第.的答案设这样说的：AsGoodrichandTamassiapointout,Ifyoutakeover50,000Englishwords(formedastheunionofthewordlistsprovidedintwo variantsofUnix),usingtheconstants31,33,37,39,and41willproducelessthan7collisionsineachcase.Knowingthis,itshould comeasnosurprisethatmanyJavaimplementationschooseoneoftheseconstants. 这段话也翻译.下： 正如Goodr ich和Tamass ia指出的那样，如果你对超过50,000个英.单词（由两个不同版本的Un ix字典合并.成）进.hashcode运算，并使.常数31,33,37,39和41作为乘.，每个常数算出的哈希值冲突数都.于7个，所以在上..个常数中，常数31被Java实现所选.也就不.为奇了。 上.的两个答案完美的解释了Java源码中选.数字31的原因。接下来，我将针对第.个答案就.验证，请.家继续往下看。 实验及数据可视化 本节，我将使.不同的数字作为乘.，对超过23万个英.单词进.哈希运算，并计算哈希算法的冲突率。同时，我也将针对不同 乘.算出的哈希值分布情况进.可视化处理，让.家可以直观的看到数据分布情况。本

次实验所使.的数据是 Unix/Linux平台 中的英.字典.件，.件路径为 /usr/share/dict/words。 哈希值冲突率计算 计算哈希算法冲突率并不难，.如可以.次性将所有单词的 hash code算出，并放. Set中去除重复值。之后拿单词数减去 set.size() 即可得出冲突数，有了冲突数，冲突率就可以算出来了。当然，如果使. JDK8提供的流式计算 API，则可更.便算 出，代码.段如下： publicstaticIntegerhashCode(Stringstr,Integermultiplier){inthash=0;for(inti=0;i<str.length();i++) {hash=multiplier*hash+str.charAt(i);} returnhash;} /***计算hashcode冲突率，顺便分析.下hashcode最.值和最.值，并输出 *@parammultiplier *@paramhashs / publicstaticvoidcalculateConflictRate(Integermultiplier,Listhashs){Comparatorcp=(x,y)->x>y?1:(x<y?-1:0);intmaxHash=hashs.stream().max(cp).get();intminHash=hashs.stream().min(cp).get(); //计算冲突数及冲突率 intuniqueHashNum=(int)hashs.stream().distinct().count();intconflictNum=hashs.size()-uniqueHashNum;doubleconflictRate=(conflictNum1.0)/hashs.size(); System.out.println(String.format("multiplier=%4d,minHash=%11d,maxHash=%10d,conflictNum=%6d,conflictRate=%.4f%%",multiplier,minHash,maxHash,conflictNum,conflictRate100));} 结果如下：

从上图可以看出，使.较.的质数做为乘.时，冲突率会很.。尤其是质数2，冲突率达到了55.14%。同时我们注意观察质数2作为乘.时，哈希值的分布情况。可以看得出来，哈希值分布并不是很.，仅仅分布在了整个哈希空间的正半轴部分，即 0 ~231-1。.负半轴-231~-1，则.分布。 这也证明了我们上.断.，即质数2作为乘.时，对于短字符串，.成的哈希值分布性不佳。然后再来看看我们之前所说的31、37、41这三个不.不.的质数，表现都不错，冲突数都低于7个。.质数101和199表现的也很不错，冲突率很低，这也说明哈希值溢出并不.定会导致冲突率上升。但是这两个家伙..不合就溢出，我们认为他们不是哈希算法的优选乘.。最后我们再来看看32和36这两个偶数的表现，结果并不好，尤其是32，冲突率超过了了50%。尽管36表现的要好.点，不过和31，37相.，冲突率还是.较.的。当然并.所有的偶数作为乘.时，冲突率都会.较.，.家有兴趣可以..验证。 哈希值分布可视化 上.节分析了不同数字作为乘.时的冲突率情况，这.节来分析.下不同数字作为乘.时，哈希值的分布情况。在详细分析之前，我先说说哈希值可视化的过程。我原本是打算将所有的哈希值..维散点图进.可视化，但是后来找了.圈，也没找到合适的画图.具。加之后来想了想，.维散点图可能不合适做哈希值可视化，因为这.有超过23万个哈希值。也就意味着会在图上显.超过23万个散点，如果不出意外的话，这23万个散点会聚集的很密，有可能会变成.个..块，就失去了可视化的意义了。 所以这.选择了另.种可视化效果更好的图表，也就是 excel中的平滑曲线的.维散点图（下.简称散点曲线图）。当然这.同样没有把23万散点都显.在图表上，太多了。所以在实际绘图过程中，我将哈希空间等分成了64个.区间，并统计每个区间内的哈希值数量。最后将分区编号做为X轴，哈希值数量为Y轴，就绘制出了我想要的.维散点曲线图了。 这.举个例.说明.下吧，以第0分区为例。第0分区数值区间是[-2147483648, -2080374784)，我们统计落在该数值区间内哈希值的数量，得到<分区编号,哈希值数量>数值对，这样就可以绘图了。分区代码如下： /***将整个哈希空间等分成64份，统计每个空间内的哈希值数量 *@paramhashs */ publicstaticMap<Integer,Integer>partition(Listhashs){ //step=2^32/64=2^26 finalintstep=67108864;Listnums=newArrayList<>();Map<Integer,Integer>statistics=newLinkedHashMap<>();intstart=0;for(longi=Integer.MIN_VALUE;i<=Integer.MAX_VALUE;i+=step){ finallongmin=i;finallongmax=min+step;intnum=(int)hashs.parallelStream() .filter(x->x>=min&&x<max).count(); statistics.put(start++,num);nums.add(num);} //为了防.计算出错，这.验证.下 inthashNum=nums.stream().reduce((x,y)->x+y).get();asserthashNum==hashs.size(); returnstatistics;} 本.中的哈希值是.整形表.的，整形的数值区间是[-2147483648, 2147483647]，区间..为 2^32。所以这.可以将区间等 分成64个.区间，每个..区间..为2^26。详细的分区对照表如下：

如下：

上.的图还是很..了然的，乘.2算出的哈希值.乎全部落在第32分区，也就是[0,67108864)数值区间内，落在其他区间内的哈希值数量.乎可以忽略不计。这也就不难解释为什么数字2作为乘.时，算出哈希值的冲突率如此之.的原因了。所以这样的哈希算法要它有何.啊，拖出去斩了吧。接下来看看数字3作为乘.时的表现：

3作为乘.时，算出的哈希值分布情况和2很像，只不过稍微好了那么.点点。从图中可以看出绝.部分的哈希值最终都落在了第32分区.，哈希值的分布性很差。这个也没啥.，拖出去枪毙5分钟吧。在看看数字17的情况怎么样：

数字17作为乘.时的表现，明显.上.两个数字好点了。虽然哈希值在第32分区和第34分区有.定的聚集，但是相.较上.2和3，情况明显好好了很多。除此之外，17作为乘.算出的哈希值在其他区也均有分布，且较为均匀，还算是.个不错的乘.吧。接下来来看看我们本.的主.31了，31作为乘.算出的哈希值在第33分区有.定的.聚集。不过相.于数字17，主.31的表现.好了.些。.先是哈希值的聚集程度没有17那么严重，其次哈希值在其他区分布的情况也要好于17。总之，选31，准没错啊。

最后再来看看.质数101的表现，不难看出，质数101作为乘.时，算出的哈希值分布情况要好于主.31，有点喧宾夺主的意思。不过不可否认的是，质数101的作为乘.时，哈希值的分布性确实更加均匀。所以如果不在意质数101容易导致数据信息丢失问题，或许其是.个更好的选择。 写在最后 经过上.的分析与实践，我想.家应该明.了StringhashCode.法中选择使.数字31作为乘.的原因了。本.本质是.篇简单的科普..已，并没有银弹。如果.家读完后觉得.涨知识了，那这篇.章的.的就达到了。最后，本篇.章的配图画的还是很.苦的，所以如果.家觉得.章不错，不妨就给个赞吧，就当是对我的.励了。 另外，如果.章中有不妥或者错误的地.，也欢迎指出来。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1.送 9个智能.机，.粉丝请绕道！

2. Nginx反向代理、负载均衡图.教程！ 3.为什么你学不会递归?
3.

.个..不主动联系你还有机会吗?

5.团队开发中 Git最佳实践

喜欢.章，点个在看

阅读原. 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试官：Redis内存满了怎么办？ 千.qianshanJava后端 2019-10-05 Redis占.内存.. 我们知道Redis是基于内存的key-value数据库，因为系统的内存..有限，所以我们在使.Redis的时候可以配置Redis能使.的最.的内存..。 1、通过配置.件配置 通过在Redis安装.录下.的redis.conf配置.件中添加以下配置设置内存..

redis的配置.件不.定使.的是安装.录下.的redis.conf.件，启动redis服务的时候是可以传.个参数指定redis的配置.件的 2、通过命令修改 Redis.持运.时通过命令动态修改内存..

如果不设置最.内存..或者设置最.内存..为0，在64位操作系统下不限制内存..，在32位操作系统下最多使.3GB内存 Redis的内存淘汰 既然可以设置Redis最.占.内存..，那么配置的内存就有.完的时候。那在内存.完的时候，还继续往Redis..添加数据不就没内存可.了吗？实际上Redis定义了.种策略.来处理这种情况： noeviction(默认策略)：对于写请求不再提供服务，直接返回错误（DEL请求和部分特殊请求除外） allkeys-lru：从所有key中使.LRU算法进.淘汰 volatile-lru：从设置了过期时间的key中使.LRU算法进.淘汰 allkeys-random：从所有key中随机淘汰数据 volatile-random：从设置了过期时间的key中随机淘汰 volatile-ttl：在设置了过期时间的key中，根据key的过期时间进.淘汰，越早过期的越优先被淘汰 当使.volatile-lru、volatile-random、volatile-ttl这三种策略时，如果没有key可以被淘汰，则和noeviction.样返回 错误 如何获取及设置内存淘汰策略 获取当前内存淘汰策略：

通过配置.件设置淘汰策略（修改redis.conf.件）：

通过命令修改淘汰策略：

LRU算法 什么是LRU? 上.说到了Redis可使.最.内存使.完了，是可以使.LRU算法进.内存淘汰的，那么什么是LRU算法呢? LRU(LeastRecentlyUsed)，即最近最少使.，是.种缓存置换算法。在使.内存作为缓存的时候，缓存的...般是固定的。当缓存被占满，这个时候继续往缓存..添加数据，就需要淘汰.部分.的数据，释放内存空间.来存储新的数据。这个时候就可以使.LRU算法了。其核.思想是：如果.个数据在最近.段时间没有被.到，那么将来被使.到的可能性也很.，所以就可以被淘汰掉。 使.java实现.个简单的LRU算法 1 public class LRUCache<k, v> 2 { 3 //容 4量 5 private capacity; 6 //当前有多少节int点的统 7计 8 private int count; 9 //缓存节 10点 11 private Map<k, Node<k, v>> nodeMap; 12 private Node<k, v> head; 13 private Node<k, v> tail; 14 15 public LRUCache(int capacity) 16 { 17 if (capacity < 1) 18 { 19 throw new IllegalArgumentException(String.valueOf(capacity)); 20 } this.capacity = capacity; this.capacity capacity;
this.nodeMap = new HashMap<>();
码 //初始化头节点和尾节点，利 .哨兵模式减少判断头结点和尾节点为空的代 Node headNode = new Node(null, null);
Node tailNode = new Node(null, null);
headNode.next = tailNode;
tailNode.pre = headNode;
this.head = headNode;

```
this.tail = tailNode;
}
public void put(k key, v value)
{
Node<k, v> node = nodeMap.get(key);
if (node == null)
{
if (count >= capacity) {
//先移除 .个节 点
removeNode();
}
node = new Node<>(key, value);
//添加节 点
addNode(node);
} else
{
//移动节点到头节 点
moveNodeToHead(node);
}
}
public Node<k, v> get(k key)
{
Node<k, v> node = nodeMap.get(key);
if (node != null)
{
moveNodeToHead(node);
}
return node
;
}
private void removeNode()
{
Node node = tail.pre;
//从链表 ..移 除
removeFromList(node);
nodeMap.remove(node.key);
count--;

75 76 } count ;
77 78 79 80 81 { private void removeFromList(Node<k, Node pre = node.pre; Node next = node.next; v> node)
82 83 84 pre.next next.pre = = next; pre;
85 86 87 88 89 90 ; ; } node.next = null node.pre = null
91 92 93 94 95 96 97 98 99 { 部 private void addNode(Node<k, v> node) //添加节点到头addToHead(node);
nodeMap.put(node.key, node); count++; }
100 101 102 { private void addToHead(Node<k, Node next = head.next; v> node)
next.pre = node;
node.next = next;
node.pre = head;
head.next = node;
}
public void moveNodeToHead(Node<k, v> node)
{
//从链表 ..移 除
```

```
removeFromList(node);
//添加节点到头 部
addToHead(node);
}
class Node<k, v>
{
k key;
v value;
Node pre;
Node next;
public Node(k key, v value)
{
this key = key

this.key key ; this.value = value ; } } }
```

上.这段代码实现了.个简单的LUR算法，代码很简单，也加了注释，仔细看.下很容易就看懂。 LRU在Redis中的实现 近似LRU算法 Redis使.的是近似LRU算法，它跟常规的LRU算法还不太.样。近似LRU算法通过随机采样法淘汰数据，每次随机出5（默认）个key，从..淘汰掉最近最少使.的key。 可以通过maxmemory-samples参数修改采样数量：例：maxmemory-samples10maxmenory-samples配置的越.， 淘汰的结果越接近于严格的LRU算法 Redis为了实现近似LRU算法，给每个key增加了.个额外增加了.个24bit的字段，.来存储该key最后.次被访问的时间。 Redis3.0对近似LRU的优化 Redis3.0对近似LRU算法进.了.些优化。新算法会维护.个候选池（..为16），池中的数据根据访问时间进.排序，第.次随机选取的key都会放.池中，随后每次随机选取的key只有在访问时间.于池中最.的时间才会放.池中，直到候选池被放满。当放满后，如果有新的key需要放.，则将池中最后访问时间最.（最近被访问）的移除。 当需要淘汰的时候，则直接从池中选取最近访问时间最.（最久没被访问）的key淘汰掉就.。 LRU算法的对. 我们可以通过.个实验对.各LRU算法的准确率，先往Redis..添加.定数量的数据n，使Redis可.内存.完，再往Redis..添加n/2的新数据，这个时候就需要淘汰掉.部分的数据，如果按照严格的LRU算法，应该淘汰掉的是最先加.的n/2的数据。.成如下各LRU算法的对.图（图.来源）：

你可以看到图中有三种不同颜.的点：

浅灰.是被淘汰的数据

灰.是没有被淘汰掉的.数据

绿.是新加.的数据

我们能看到Redis3.0采样数是10.成的图最接近于严格的LRU。.同样使.5个采样数，Redis3.0也要优于Redis2.8。 LFU算法 LFU算法是Redis4.0..新加的.种淘汰策略。它的全称是LeastFrequentlyUsed，它的核.思想是根据key的最近被访问的频率进.淘汰，很少被访问的优先被淘汰，被访问的多的则被留下来。 LFU算法能更好的表..个key被访问的热度。假如你使.的是LRU算法，.个key很久没有被访问到，只刚刚是偶尔被访问了.次，那么它就被认为是热点数据，不会被淘汰，.有些key将来是很有可能被访问到的则被淘汰了。如果使.LFU算法则不会出现这种情况，因为使..次并不会使.个key成为热点数据。 LFU.共有两种策略：

volatile-lfu：在设置了过期时间的key中使.LFU算法淘汰key

allkeys-lfu：在所有的key中使.LFU算法淘汰数据

设置使.这两种淘汰策略跟前.讲的.样，不过要注意的.点是这两周策略只能在Redis4.0及以上设置，如果在Redis4.0以下设置会报错 问题 最后留.个.问题，可能有的.注意到了，我在.中并没有解释为什么Redis使.近似LRU算法.不使.准确的LRU算法，可以在评论区给出你的答案，.家.起讨论学习。 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试官：怎么保证缓存与数据库的双写.致性？ 你是我的海啸Java后端 2019-09-15 点击上.蓝.字体，选择"标星公众号"优质.章，第.时间送达

来源：blog.csdn.net/chang384915878 分布式缓存是现在很多分布式应.中必不可少的组件，但是.到了分布式缓存，就可能会涉及到缓存与数据库双存储双写，你只要是双写，就.定会有数据.致性的问题，那么你如何解决.致性问题？ CacheAsidePattern 最经典的缓存+数据库读写的模式，就是CacheAsidePattern。读的时候，先读缓存，缓存没有的

话，就读数据库，然后取出数据后放.缓存，同时返回响应。更新的时候，先更新数据库，然后再删除缓存。 为什么是删除缓存，.不是更新缓存？ 原因很简单，很多时候，在复杂点的缓存场景，缓存不单单是数据库中直接取出来的值。.如可能更新了某个表的.个字段，然后其对应的缓存，是需要查询另外两个表的数据并进.运算，才能计算出缓存最新的值的。 另外更新缓存的代价有时候是很.的。是不是说，每次修改数据库的时候，都.定要将其对应的缓存更新.份？也许有的场景是这样，但是对于.较复杂的缓存数据计算的场景，就不是这样了。如果你频繁修改.个缓存涉及的多个表，缓存也频繁更新。但是问题在于，这个缓存到底会不会被频繁访问到？ 举个栗.，.个缓存涉及的表的字段，在1分钟内就修改了20次，或者是100次，那么缓存更新20次、100次；但是这个缓存在1分钟内只被读取了 1次，有.量的冷数据。实际上，如果你只是删除缓存的话，那么在 1分钟内，这个缓存不过就重新计算.次.已，开销.幅度降低，.到缓存才去算缓存。 其实删除缓存，.不是更新缓存，就是.个 lazy计算的思想，不要每次都重新做复杂的计算，不管它会不会.到，.是让它到需要被使.的时候再重新计算。像mybatis，hibernate，都有懒加载思想。查询.个部.，部.带了.个员.的list，没有必要说每次查询部.，都..的 1000个员.的数据也同时查出来啊。80%的情况，查这个部.，就只是要访问这个部.的信息就可以了。先查部.，同时要访问..的员.，那么这个时候只有在你要访问..的员.的时候，才会去数据库..查询1000个员.。 最初级的缓存不.致问题及解决.案 问题：先修改数据库，再删除缓存。如果删除缓存失败了，那么会导致数据库中是新数据，缓存中是旧数据，数据就出现了不.致。解决思路：先删除缓存，再修改数据库。如果数据库修改失败了，那么数据库中是旧数据，缓存中是空的，那么数据不会不.致。因为读的时候缓存没有，则读数据库中旧数据，然后更新到缓存中。

.较复杂的数据不.致问题分析 数据发.了变更，先删除了缓存，然后要去修改数据库，此时还没修改。.个请求过来，去读缓存，发现缓存空了，去查询数据库，查到了修改前的旧数据，放到了缓存中。随后数据变更的程序完成了数据库的修改。 完了，数据库和缓存中的数据不.样了。。。 为什么上亿流量.并发场景下，缓存会出现这个问题？ 只有在对.个数据在并发的进.读写的时候，才可能会出现这种问题。其实如果说你的并发量很低的话，特别是读并发很低，每天访问量就 1万次，那么很少的情况下，会出现刚才描述的那种不.致的场景。但是问题是，如果每天的是上亿的流量，每秒并发读是.万，每秒只要有数据更新的请求，就可能会出现上述的数据库+缓存不.致的情况。 解决.案如下： 更新数据的时候，根据数据的唯.标识，将操作路由之后，发送到.个 jvm内部队列中。读取数据的时候，如果发现数据不在缓存中，那么将重新读取数据+更新缓存的操作，根据唯.标识路由之后，也发送同.个jvm内部队列中。 .个队列对应.个.作线程，每个.作线程串.拿到对应的操作，然后.条.条的执..这样的话，.个数据变更的操作，先删除缓存，然后再去更新数据库，但是还没完成更新。此时如果.个读请求过来，读到了空的缓存，那么可以先将缓存更新的请求发送到队列中，此时会在队列中积压，然后同步等待缓存更新完成。 这.有.个优化点，.个队列中，其实多个更新缓存请求串在.起是没意义的，因此可以做过滤，如果发现队列中已经有.个更新缓存的请求了，那么就不.再放个更新请求操作进去了，直接等待前.的更新操作请求完成即可。 待那个队列对应的.作线程完成了上.个操作的数据库的修改之后，才会去执.下.个操作，也就是缓存更新的操作，此时会从数据库中读取最新的值，然后写.缓存中。 如果请求还在等待时间范围内，不断轮询发现可以取到值了，那么就直接返回；如果请求等待的时间超过.定时.，那么这.次直接从数据库中读取当前的旧值。

.并发的场景下，该解决.案要注意的问题： 1、读请求.时阻塞 由于读请求进.了.常轻度的异步化，所以.定要注意读超时的问题，每个读请求必须在超时时间范围内返回。 该解决.案，最.的.险点在于说，可能数据更新很频繁，导致队列中积压了.量更新操作在..，然后读请求会发..量的超时，最后导致.量的请求直接.数据库。务必通过.些模拟真实的测试，看看更新数据的频率是怎样的。 另外.点，因为.个队列中，可能会积压针对多个数据项的更新操作，因此需要根据..的业务情况进.测试，可能需要部署多个服务，每个服务分摊.些数据的更新操作。如果.个内存队列.居然会挤压 100个商品的库存修改操作，每隔库存修改操作要耗费 10ms去完成，那么最后.个商品的读请求，可能等待10*100=1000ms=1s后，才能得到数据，这个时候就导致读请求的.时阻塞。 .定要做根据实际业务系统的运.情况，去进..些压.测试，和模拟线上环境，去看看最繁忙的时候，内存队列可能会挤压多少更新操作，可能会导致最后.个更新操作对应的读请求，会 hang多少时间，如果读请求在 200ms返回，如果你计算过后，哪怕是最繁忙的时候，积压10个更新操作，最多等待200ms，那还可以的。 如果.个内存队列中可能积压的更新操作特别多，那么你就要加机器，让每个机器上部署的服务实例处理更少的数据，那么每个内存队列中积压的更新操作就会越少。 其实根据之前的项.经验，.般来说，数据的写频率是很低的，因此实际上正常来说，在队列中积压的更新操作应该是很少的。像这种针对读.并发、读缓存架构的项.，.般来说写请求是.常少的，每秒的QPS能到.百就不错了。 实际粗略测算.下 如果.秒有500的写操作，如果分成5个时间.，每200ms就100个写操作，放到20个内存队列中，每个内存队列，可能就积压5个写操作。每个写操作性能测试后，.般是在20ms左右就完成，那么针对每个内存队列的数据的读请求，也就最多hang.会.，200ms以内肯定能返回了。 经过刚才简单的测算，我们知道，单机.撑的写 QPS在.百是没问题的，如果写 QPS扩.了 10倍，那么就扩容机器，扩容 10倍的机器，每个机器20个队列。 2、读请求并发量过. 这.还必须做好压.测试，确保恰巧碰上上述情况的时候，还有.个.险，就是突然间.量读请求会在..毫秒的延时 hang在服务上，看服务能不能扛的住，需要多少机器才能扛住最.的极限情况的峰值。 但是因为并不是所有的数据都在同.时间更新，缓存也不会同.时间失效，所以每次可能也就是少数数据的缓

存失效了，然后那些数据对应的读请求过来，并发量应该也不会特别.。 3、多服务实例部署的请求路由 可能这个服务部署了多个实例，那么必须保证说，执.数据更新操作，以及执.缓存更新操作的请求，都通过 Nginx服务器路由到相同的服务实例上。 .如说，对同.个商品的读写请求，全部路由到同.台机器上。可以..去做服务间的按照某个请求参数的hash路由，也可以. Nginx的hash路由功能等等。 4、热点商品的路由问题，导致请求的倾斜 万.某个商品的读写请求特别.，全部打到相同的机器的相同的队列..去了，可能会造成某台机器的压.过.。就是说，因为只有在商品数据更新的时候才会清空缓存，然后才会导致读写并发，所以其实要根据业务系统去看，如果更新频率不是太.的话，这个问题的影响并不是特别.，但是的确可能某些机器的负载会..些。 如果喜欢本篇.章，欢迎转发。关注订阅号「Web项.聚集地」，回复「进群」即可进...告技术交流。 推荐阅读 1. 仅推荐.个置顶的程序员公众号

2.

寓教于乐，.玩游戏的.式学习Git

3.

在浏览器输.URL回.之后发.了什么

4.

在阿..了5年，.试个.公司挂了...

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

.试官：数据量很.，分.查询很慢，有什么优化.案？ 悠悠Java后端 2019-11-08 点击上.Java后端，选择设为星标优质.章，及时送达

来源|cnblogs.com/youyoui/p/7851007.html 当需要从数据库查询的表有上万条记录的时候，.次性查询所有结果会变得很慢，特别是随着数据量的增加特别明显，这时需要使.分.查询。对于数据库分.查询，也有很多种.法和优化的点。下.简单说.下我知道的.些.法。 准备.作 为了对下.列举的.些优化进.测试，下.针对已有的.张表进.说明。

表名：order_history描述：某个业务的订单历史表主要字段：unsignedintid，tinyint(4)inttype字段情况：该表.共37个字段，不包含text等.型数据，最.为varchar(500)，id字段为索引，且为递增。数据量：5709294MySQL版本：5.7.16线下找.张百万级的测试表可不容易，如果需要..测试的话，可以写shell脚本什么的插.数据进.测试。以下的sql所有语句执.的环境没有发.改变，下.是基本测试结果： selectcount(*)fromorders_history; 返回结果：5709294 三次查询时间分别为：

8903ms 8323ms 8401ms .般分.查询 .般的分.查询使.简单的limit.句就可以实现。limit.句声明如下：SELECT*FROMtableLIMIT[offset,]rows|rowsOFFSEToffset LIMIT.句可以被.于指定SELECT语句返回的记录数。需注意以下.点：

第.个参数指定第.个返回记录.的偏移量，注意从 0开始 第.个参数指定返回记录.的最.数. 如果只给定.个参数：它表.返回最.的记录.数.

第.个参数为-1表.检索从某.个偏移量到记录集的结束所有的记录.初始记录.的偏移量是0(.不是1)下.是.个应.实例：select*fromorders_historywheretype=8limit1000,10;该条语句将会从表orders_h istory中查询 oset:1000开始之后的10条数据，也就是第1001条到第1010条数据（ 1001<= 数据表中的记录默认使.主键（.般为id）排序，上.的结果相当于：select*fromorders_historywheretype=8orderbyidlimit10000,10; 三次查询时间分别为：

3040ms 3063ms 3018ms针对这种查询.式，下.测试查询记录量对时间的影响：select*fromorders_historywheretype=8limit10000,1;select*fromorders_historywheretype=8limit10000,10;select*fromorders_historywheretype=8limit10000,100;select*fromorders_historywheretype=8limit10000,1000;select*fromorders_historywheretype=8limit10000,10000; 三次查询时间如下：

查询1条记录：3072ms3092ms3002ms查询10条记录：3081ms3077ms3032ms查询100条记录：3118ms3200ms3128ms查询1000条记录：3412ms3468ms3394ms查询10000条记录：3749ms3802ms3696ms另外我还做了.来次查询，从查询时间来看，基本可以确定，在查询记录量低于100时，查询时间基本没有差距，随着查询记录量越来越.，所花费的时间也会越来越多。 针对查询偏移量的测试：

select*fromorders_historywheretype=8limit100,100;select*fromorders_historywheretype=8limit1000,100;select*fromorders_historywheretype=8limit10000,100;select*fromorders_historywheretype=8limit100000,100;select*fromorders_historywheretype=8limit1000000,100; 三次查询时间如下：

查询100偏移：25ms24ms24ms

查询1000偏移：78ms76ms77ms

查询10000偏移：3092ms3212ms3128ms

查询100000偏移：3878ms3812ms3798ms

查询1000000偏移：14608ms14062ms14700ms 随着查询偏移的增.，尤其查询偏移.于10万以后，查询时间急剧增加。这种分.查询.式会从数据库第.条记录开始扫描，所以越往后，查询速度越慢，.且查询的数据越多，也会拖慢总查询速度。 使..查询优化 这种.式先定位偏移位置的id，然后往后查询，这种.式适.于id递增的情况。 select*fromorders_historywheretype=8limit100000,1;selectidfromorders_historywheretype=8limit100000,1;select*fromorders_historywheretype=8and id>=(selectidfromorders_historywheretype=8limit100000,1)limit100;select*fromorders_historywheretype=8limit100000,100; 4条语句的查询时间如下：

第1条语句：3674ms第2条语句：1315ms第3条语句：1327ms第4条语句：3710ms

针对上.的查询需要注意：

.较第1条语句和第2条语句：使.selectid代替select*速度增加了3倍.较第2条语句和第3条语句：速度相差..毫秒.较第3条语句和第4条语句：得益于selectid速度增加，第3条语句查询速度增加了3倍

这种.式相较于原始.般的查询.法，将会增快数倍。 使.id限定优化 这种.式假设数据表的id是连续递增的，则我们根据查询的.数和查询的记录数可以算出查询的id的范围，可以使.idbetweenand来查询：select*fromorders_historywheretype=2 andidbetween1000000and1000100limit100; *查询时间：15ms12ms9ms 这种查询.式能够极.地优化查询速度，基本能够在..毫秒之内完成。限制是只能使.于明确知道id的情况，不过.般建.表的时候，都会添加基本的id字段，这为分.查询带来很多便利。Tips:关注微信公众号：Java后端，获取每.技术博.推送。 还可以有另外.种写法：* select*fromorders_historywhereid>=1000001limit100;当然还可以使.in的.式来进.查询，这种.式经常.在多表关联的时候进.查询，使.其他表查询的id集合，来进.查询： select*fromorders_historywhereidin (selectorder_idfromtrade_2wheregoods='pen')limit100;*这种in查询的.式要注意：某些mysql版本不.持在in.句中使.limit。使.临时表优化 这种.式已经不属于查询优化，这.附带提.下。 *对于使.id限定优化中的问题，需要id是连续递增的，但是在.些场景下，.如使.历史表的时候，或者出现过数据缺失问题时，可以考虑使.临时存储的表来记录分.的id，使.分.的id来进.in查询。这样能够极.的提.传统的分.查询速度，尤其是数据量上千万的时候。 关于数据表的id说明 .般情况下，在数据库中建.表的时候，强制为每.张表添加id递增字段，这样.便查询。 如果像是订单库等数据量.常庞.，.般会进.分库分表。这个时候不建议使.数据库的id作为唯.标识，.应该使.分布式的.并发唯.id.成器来.成，并在数据表中使.另外的字段来存储这个唯.标识。 使.先使.范围查询定位id（或者索引），然后再使.索引进.定位数据，能够提.好.倍查询速度。即先 select*id，*然后再 *select*； 本.才疏学浅，难免犯错，若发现.中有错误遗漏，望不吝赐教。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。↓扫描.维码进群↓

推荐阅读 1. Java代码是如何.步步输出结果的?

2. IntelliJ IDEA详细图解最常.的配置 3.Maven实战问题和最佳实践

3.

12306的架构到底有多.逼?

5.团队开发中 Git最佳实践

喜欢.章，点个在看

作者|亦. 原.|https://urlify.cn/zaYRJv 对于任何框架..，在使.前都要进..系列的初始化，MyBatis也不例外。本章将通过以下.点详细介绍MyBatis的初始化过程。

1. MyBatis的初始化做了什么
2.

MyBatis基于XML配置.件创建Con.guration对象的过程

3.

.动加载XML配置.件创建Con.guration对象完成初始化，创建并使.SqlSessionFactory对象

4.涉及到的设计模式 .、MyBatis的初始化做了什么 任何框架的初始化，..是加载..运.时所需要的配置信息。MyBatis的配置信息，.概包含以下信息，其.层级结构如下： × con.guration配置 ×properties属性 ×settings设置 ×typeAliases类型命名 ×typeHandlers类型处理器 ×objectFactory对象.. ×plugins插件 ×environments环境 ×environment环境变量 ×transactionManager事务管理器 ×dataSource数据源 ×映射器 MyBatis的上述配置信息会配置在XML配置.件中，那么，这些信息被加载进.MyBatis内部，MyBatis是怎样维护的呢？ MyBatis采.了.个.常直.和简单的.式---使.org.apache.ibatis.session.Con.guration对象作为.个所有配置信息的容器，Con.guration对象的组织结构和XML配置.件的组织结构.乎完全.样（当然，Con.guration对象的功能并不限于此，它还负责创建.些MyBatis内部使.的对象，如Executor等，这将在后续的.章中讨论）。如下图所.：

MyBatis根据初始化好Con.gurat ion信息，这时候..就可以使.MyBat is进.数据库操作了。 可以这么说，MyBat is初始化的过程，就是创建Con.gurat ion对象的过程。 MyBatis的初始化可以有两种.式：

基于XML配置.件：基于XML配置.件的.式是将MyBat is的所有配置信息放在XML.件中，MyBat is通过加载并 XML配置.件，将配置.信息组装成内部的Con.gurat ion对象 基于Java API：这种.式不使.XML配置.件，需要MyBat is使.者在Java代码中，.动创建Con.gurat ion对象， 然后将配置参数set进.Con.gurat ion对象中 （PS: MyBatis具体配置信息有哪些，.分别表.什么意思，不在本.的叙述范围，读者可以参考我的《Java Persistence withMyBatis 3(中.版)》的第.章引导MyBat is中有详细的描述） 接下来我们将通过基于XML配置.件.式的MyBat is初始化，深.探讨MyBat is是如何通过配置.件构建Con.gurat ion对象，并使.它的。 .、MyBatis基于XML配置.件创建Con.guration对象的过程 现在就从使.MyBatis的简单例...，深.分析.下MyBatis是怎样完成初始化的，都初始化了什么。看以下代码：

有过MyBatis使.经验的读者会知道，上述语句的作.是执.com.foo.bean.BlogMapper.queryAllBlogInfo定义的SQL语句，返回.个List结果集。总的来说，上述代码经历了mybatis初始化-->创建SqlSession-->执.SQL语句返回结果三个过程。上述代码的功能是根据配置.件mybatis-con.g.xml配置.件，创建SqlSessionFactory对象，然后产.SqlSession，执.SQL语句。.mybatis的初始化就发.在第三句：SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);现在就让我们看看第三句到底发.了什么。 MyBatis初始化基本过程：SqlSessionFactoryBuilder根据传.的数据流.成Con.guration对象，然后根据Con.guration对象创建默认的SqlSessionFactory实例。 初始化的基本过程如下序列图所.：

由上图所.，mybatis初始化要经过简单的以下.步： 1.调.Sq lSessionFactoryBuilder对象的build(inputStream).法；

2. SqlSessionFactoryBuilder会根据输.流inputStream等信息创建XMLCon.gBuilder对象;
3. SqlSessionFactoryBuilder调.XMLCon.gBuilder对象的parse().法；
4.

XMLConfigBuilder对象返回Configuration对象；

5.

SqlSessionFactoryBuilder根据Configuration对象创建.个DefaultSessionFactory对象；

6.

SqlSessionFactoryBuilder返回DefaultSessionFactory对象给Client，供Client使.。

SqlSessionFactoryBuilder相关的代码如下所.：

上述的初始化过程中，涉及到了以下.个对象：

SqlSessionFactoryBuilder：SqlSessionFactory的构造器，.于创建SqlSessionFactory，采.了Builder设计模式 Con.guration：该对象是mybatis-con.g.xml.件中所有mybatis配置信息 SqlSessionFactory：SqlSession..类，以..形式创建SqlSession对象，采.了Factory..设计模式 XmlCon.gParser：负责将mybatis-con.g.xml配置.件解析成Con.guration对象，共 SqlSessonFactoryBuilder使.，创建SqlSessionFactory 创建Con.guration对象的过程 接着上述的 MyBatis初始化基本过程讨论，当Sq lSessionFactoryBuilder执.bu ild().法，调.了XMLCon.gBu ilder的parse ().法，然后返回了Con.gurat ion对象。那么parse( ).法是如何处理XML.件，.成Con.gurat ion对象的呢?

1. XMLCon.gBuilder会将XML配置.件的信息转换为Document对象，.XML配置定义.件DTD转换成XMLMapperEntityResolver对象，然后将.者封装到XpathParser对象中，XpathParser的作.是提供根据Xpath表达式获取基本的DOM节点Node信息的操作。如下图所.：
2.

之后XMLCon.gBu ilder调.parse( ).法：会从XPathParser中取出 <con.guration>节点对应的Node对象，然后解析此Node节点的.Node：propert ies, settings, typeAliases,typeHandlers, objectFactory, objectWrapperFactory,plugins,environments,databaseIdProvider,mappers

注意：在上述代码中，还有.个.常重要的地.，就是解析XML配置.件.节点的.法 mapperElements(root.evalNode("mappers")),它将解析我们配置的Mapper.xm l配置.件，Mapper配置.件可以说是MyBat is的核.，MyBat is的特性和理念都体现在此Mapper的配置和设计上，我们将在后续的.章中讨论它，敬请期待.

3.

然后将这些值解析出来设置到Con.gurat ion对象中。

解析.节点的过程这.就不..介绍了，..可以参照MyBat is源码仔细揣摩，我们就看上述的 environmentsElement(root.evalNode("environments"));.法是如何将env ironments的信息解析出来，设置到Con.guration对象中的：

4.

返回Con.gurat ion对象我们将上述的MyBat is初始化基本过程的序列图细化。

三、.动加载XML配置.件创建Con.guration对象完成初始化，创建并使.SqlSessionFactory对象 我们可以使.XMLCon.gBuilder.动解析XML配置.件来创建Con.guration对象，代码如下：

四、涉及到的设计模式 初始化的过程涉及到创建各种对象，所以会使..些创建型的设计模式。在初始化的过程中，Builder模式运.的.较多。 Builder模式应.1：SqlSessionFactory的创建对于创建SqlSessionFactory时，会根据情况提供不同的参数，其参数组合可以有以下.种：

由于构造时参数不定，可以为其创建.个构造器Bu ilder，将Sq lSessionFactory的构建过程和表.分开：

MyBatis将Sq lSessionFactoryBuilder和Sq lSessionFactory相互独.。 Builder模式应.2：数据库连接环境Env ironment对象的创建在构建Con.gurat ion对象的过程中，XMLCon.gParser解析 mybat is XML配置.件节点节点时，会有以下相应的代码：

在Env ironment内部，定义了静态内部Bu ilder类：

以上就是本.《深.理解mybatis原理》Mybatis初始化机制详解的全部内容，希望对.家有所帮助！上述内容如有不妥之处，还请读者指出，共同探讨，共同进步！ 【END】 推荐阅读 1.我采访了.位 Pornhub .程师，聊了这些纯纯的话题

2.

常.排序算法总结 -Java实现 3.Java：如何更优雅的处理空值?

4.

MySQL：数据库优化，可以看看这篇.章

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试官：说.说SpringBoot.动配置原理吧，我懵逼了... 你在我家..Java后端 2019-09-22

作者|你在我家.. juejin.im/post/5ce5effb6fb9a07f0b039a14 推荐：公众号.质量博.整理（请戳我） .伙伴们是否想起曾经被SSM整合.配的恐惧？相信很多.伙伴都是有过这样的经历的，..堆配置问题，各种排除扫描，导..个新的依赖.得添加新的配置。.从有了SpringBoot之后，咋们就起.了！各种零配置开箱即.，.我们之所以开发起来能够这么爽，.动配置的功劳少不了，今天我们就.起来讨论.下SpringBoot.动配置原理，看完..有个.概，不.于被.试官问的.红...。 本.主要分为三.部分： SpringBoot源码常.注解拾遗 SpringBoot启动过程 SpringBoot.动配置原理 1.SpringBoot源码常.注解拾遗 这部分主要讲.下SpringBoot源码中经常使.到的注解，以扫清后.阅读源码时候的障碍。 组合注解 当可能.量同时使.到.个注解到同.个类上，就可以考虑将这.个注解到别的注解上。被注解的注解我们就称之为组合注解。 元注解：可以注解到别的注解上的注解。组合注解：被注解的注解我们就称之为组合注解。 @Value @Value就相当于传统xml配置.件中的value字段。假设存在代码：

1 @Component 2 public class Person { 3 4 @Value("i am name") 5 6 private String name; 7 } 上.代码等价于的配置.件： 2 </property 3 1

> 我们知道配置.件中的value的取值可以是： 字.量 通过 ${key}.式从环境变量中获取值 通过 ${key}.式全局配置.件中获取值 #{SpEL} 所以，我们就可以通过 @Value(${key})的.式获取全局配置.件中的指定配置项。微信搜索web_resource回复爆.送你.质量技术博.。 @ConfigurationProperties 如果我们需要取N个配置项，通过 @Value的.式去配置项需要.个.个去取，这就显得有点low了。我们可以使 . @ConfigurationProperties。 标有@ConfigurationProperties的类的所有属性和配置.件中相关的配置项进.绑定。（默认从全局配置.件中获取配置值），绑定之后我们就可以通过这个类去访问全局配置.件中的属性值了。 下.看.个实例： 1. 在主配置.件中添加如下配置

2.

创建配置类，由于篇幅问题这.省略了setter、getter.法，但是实际开发中这个是必须的，否则.法成功注.。另外，@Component这个注解也还是需要添加的。

1 @Component 2 @ConfigurationProperties(prefix = "person") 3 4 public class Person { 5

这. @ConfigurationProperties有.个prefix参数，主要是.来指定该配置项在配置.件中的前缀。 3.测试，在SpringBoot环境中，编写个测试.法，注.Person类，即可通过Person对象取到配置.件的值。 @Import @Import注解.持导.普通java类，并将其声明成.个bean。主要.于将多个分散的javaconfig配置类融合成.个更.的 config类。 @Import注解在4.2之前只.持导.配置类。在4.2之后@Import注解.持导.普通的java类,并将其声明成.个bean。 @Import三种使..式 直接导.普通的Java类。配合.定义的ImportSelector使.。配合ImportBeanDefinitionRegistrar使.。 1. 直接导.普通的Java类

1.

创建.个普通的Java类。

1 public class Circle { 2 3 public void sayHi() { 4 System.out.println("Circle sayHi()"); 5 } 6 7 }

2.创建.个配置类，..没有显式声明任何的Bean，然后将刚才创建的Circle导.。微信搜索web_resource回复爆.送你.质量技术博.。 1 @Import({Circle.class}) 2 @Configuration 3 public class MainConfig { 4 5 } 3.创建测试类。

4.运.结果： CirclesayHi() 可以看到我们顺利的从IOC容器中获取到了Circle对象，证明我们在配置类中导.的Circle类，确实被声明为了.个Bean。 2.配合.定义的ImportSelector使. ImportSelector是.个接.，该接.中只有.个selectImports.法，.于

返回全类名数组。所以利.该特性我们可以给容器动态导.N个Bean。 1.创建普通Java类Triangle。 1 public class Triangle { 2 3 public void sayHi(){ 4 System.out.println("Triangle sayHi()"); 5 } 6 7 }

2.创建 ImportSelector实现类，selectImports返回Triangle的全类名。

3.

创建配置类，在原来的基础上还导.了MyImportSelector。

5.

运.结果：

1 public class MyImportSelector implements ImportSelector {
2
3 @Override
4 public String [] selectImports(AnnotationMetadata annotationMetadata) {
5 return new String[]{"annotation.importannotation.waytwo.Triangle"};
6
7 }
8
}

4.创建测试类 1 public static void main(String[] args) {
2
3 ApplicationContext context = new AnnotationConfigApplicationContext(MainConfigTwo.class);
4 Circle circle = context.getBean(Circle.class);
5 Triangle triangle = context.getBean(Triangle.class);
6 circle.sayHi();
7 triangle.sayHi();
8
9 }

CirclesayHi()TrianglesayHi() 可以看到Triangle对象也被IOC容器成功的实例化出来了。 3.配合ImportBeanDefinitionRegistrar使. ImportBeanDefinitionRegistrar也是.个接.，它可以.动注册bean到容器中，从.我们可以对类进.个性化的定制。(需要搭配@Import与@Configuration.起使.。）微信搜索web_resource回复爆.送你.质量技术博.。 1.创建普通Java类Rectangle。 1 public class Rectangle { 2 3 public void sayHi() { 4 System.out.println("Rectangle sayHi()"); 5 } 6 7 }

2.创建 ImportBeanDefinitionRegistrar实现类，实现.法直接.动注册.个名叫rectangle的Bean到IOC容器中。

3.

创建配置类，导.MyImportBeanDefinitionRegistrar类。

1 public class MyImportBeanDefinitionRegistrar implements ImportBeanDefinitionRegistrar {
2
3 @Override
4 public void registerBeanDefinitions(Anno tati onMetadata anno tationMetadata, BeanDefinitionRegistry beanDefinitionRe
5
6 RootBeanDefinition rootBeanDefinition = new RootBeanDefinition(Rectangle.class);
7 //注册.个名字叫做 rectangle的 bean
8 beanDefinitionRegistry.registerBeanDefinition("rectangle", rootBeanDefinition);
9 }

```
10
11 }
```

```
1 @Import({Circle.class, MyImportSelector.class, MyImportBeanDefinitionRegistrar.class}) 2 @Configuration 3 public class MainConfigThree { 4 5 } 4.创建测试类。 1 public static void main(String[] args) {
2
3 ApplicationContext context = new AnnotationConfigApplicationContext(MainConfigThree.class);
4
5 Circle circle = context.getBean(Circle.class);
6
7 Triangle triangle = context.getBean(Triangle.class);
8 Rectangle rectangle = context.getBean(Rectangle.class);
9 circle.sayHi();
10 triangle.sayHi();
11 rectangle.sayHi();
```

} 5.运.结果 CirclesayHi() TrianglesayHi() RectanglesayHi() 嗯对，Rectangle对象也被注册进来了。 @Conditional @Conditional注释可以实现只有在特定条件满.时才启..些配置。 下.看.个简单的例.： 1. 创建普通Java类ConditionBean，该类主要.来验证Bean是否成功加载。

2.创建Cond ition实现类，@Cond itional注解只有.个Cond ition类型的参数，Cond ition是.个接.，该接.只有.个返回布尔值的matches ().法，该.法返回true则条件成.，配置类.效。反之，则不.效。在该例.中我们直接返回true。

```
1 public class MyCondition implements Condition { 2 3 @Override 4 public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata annotatedTypeMetadata) { 5 return true; 6 } 7 8 }
```

3.

创建配置类，可以看到该配置的@Cond itional传了我们刚才创建的Cond ition实现类进去，.作条件判断。微信搜索web_resource回复爆.送你.质量技术博.。

4.

编写测试.法。

```
1 @Configuration
2 @Conditional(MyCondition.class)
3 public class ConditionConfig {
4
5 @Bean
6 public ConditionBean conditionBean(){
7
8 return new ConditionBean();
9 }
10
}
```

```
1 public static void main(String[] args) {
2
3 ApplicationContext context = new AnnotationConfigApplicationContext(ConditionConfig.class);
4 ConditionBean conditionBean = context.getBean(ConditionBean.class);
5 conditionBean.sayH i();
6
7 }
```

5.结果分析 因为Cond ition的matches.法直接返回了true，配置类会.效，我们可以把matches改成返回 false，则配置类就不会.效了。 除了.定义Cond ition，Spr ing还为我们扩展了.些常.的Cond ition。

2.SpringBoot启动过程 在看源码的过程中，我们会看到以下四个类的.法经常会被调.，我们需要对.下.个类有点印象：ApplicationContextInitializer ApplicationRunner CommandLineRunner SpringApplicationRunListener 下.开始源码分析，先从SpringBoot的启动类的run().法开始看，以下是调.链：SpringApplication.run()->run(new Class[]{primarySource},args)->newSpringApplication(primarySources)).run(args)。 .直在run，终于到重点了，我们直接看newSpringApplication(primarySources)).run(args)这个.法。 上.的.法主要包括两.步骤： 创建SpringApplication对象。运.run().法。 创建SpringApplication对象 1 public SpringApplication(ResourceLoader resourceLoader, Class... primarySources) { 2 3 this.sources = new LinkedHashSet(); 4 this.bannerMode = Mode.CONSOLE; 5 this.logStartupInfo = true; 6 this.addCommandLineProperties = true; 7 this.addConversionService = true; 8 9 this.headless = true; 10 this.registerShutdownHook = true; 11 12 this.additionalProfiles = new HashSet(); 13 this.isCustomEnvironment = false; this.resourceLoader = resourceLoader; 14 Assert.notNull(primarySources, "PrimarySources must not be null");15 //保存主配置类（这.是.个数组，说明可以有多个主配置类）16 this.primarySources = new LinkedHashSet(Arrays.asList(primarySources));17 //判断当前是否是.个 Web应. 18 this.webApplicationType = WebApplicationType.deduceFromClasspath();19 //从类路径下找到 META/INF/Spring.factories配置的所有 ApplicationContextInitializer，然后保存起来 20 this.setInitializers(this.getSpringFactoriesInstances(ApplicationContextInitializer.class));21 //从类路径下找到 META/INF/Spring.factories配置的所有 ApplicationListener，然后保存起来 22 this.setListeners(this.getSpringFactoriesInstances(ApplicationListener.class)); 23 24 //从多个配置类中找到有 main.法的主配置类（只有.个） this.mainApplicationClass = this.deduceMainApplicationClass(); 25 } 运.run().法 1 public ConfigurableApplicationContext run(String... args) { 2 3 //创建计时器 4 StopWatch stopWatch = new StopWatch(); 5 stopWatch.start(); 6 //声明 IOC容器 7 ConfigurableApplicationContext context = null; 8 9 Collection exceptionReporters = new ArrayList(); 10 this.configureHeadlessProperty(); 11 //从类路径下找到 META/INF/Spring.factories获取 SpringApplicationRunListeners 12 SpringApplicationRunListeners listeners = this.getRunListeners(args); 13 //回调所有 SpringApplicationRunListeners的 starting().法 14 listeners.starting(); 15 Collection exceptionReporters; 16 try { 17 //封装命令.参数 18 ApplicationArguments applicationArguments = new DefaultApplicationArguments(args); 19 //准备环境，包括创建环境，创建环境完成后回调 SpringApplicationRunListeners#environmentPrepared().法，表.环境准备完成 20 ConfigurableEnvironment environment = this.prepareEnvironment(listeners, applicationArguments); 21 this.configureIgnoreBeanInfo(environment); 22 //打印 Banner 23 printedBanner = this.printBanner(environment); 24 //创建BannerIOC容器（决定创建 web的 IOC容器还是普通的 IOC容器） 25 context = this.createApplicationContext(); 26 exceptionReporters = this.getSpringFactoriesInstances(SpringBootExceptionReporter.class, new Class[]{ConfigurableA 27 /* 28 *准备上下.环境，将 environment保存到 IOC容器中，并且调. applyInitializers().法 29 * applyInitializers().法回调之前保存的所有的 ApplicationContextInitializer的 initialize().法 30 *然后回调所有的 SpringApplicationRunListener#contextPrepared().法 31 *最后回调所有的 SpringApplicationRunListener#contextLoaded().法 32 */ 33 this.prepareContext(context, environment, listeners, applicationArguments, printedBanner); 34 //刷新容器，IOC容器初始化（如果是 Web应.还会创建嵌.式的 Tomcat），扫描、创建、加载所有组件的地. 35 this.refreshContext(context); //从 IOC容器中获取所有的 ApplicationRunner和 CommandLineRunner进.回调//从 IOC容器中获取所有的 ApplicationRunner和 CommandLineRunner进.回调 36 this.afterRefresh(context, applicationArguments); 37 stopWatch.stop(); 38 if (this.logStartupInfo) { 39 (new StartupInfoLogger(this.mainApplicationClass)).logStarted(this.getApplicationLog(), stopWatch); 40 } 41 //调.所有 SpringApplicationRunListeners#started().法42 listeners.started(context); 43 this.callRunners(context, applicationArguments); 44 } catch (Throwable var10) { 45 this.handleRunFailure(context, var10, exceptionReporters, listeners); 46 throw new IllegalStateException(var10); 47 } 48 try { 49 listeners.running(context); 50 return context; 51 } catch (Throwable var9) { 52 this.handleRunFailure(context, var9, exceptionReporters, (SpringApplicationRunListeners)null); 53 throw new IllegalStateException(var9); 54 } 55 }

.结： run()阶段主要就是回调本节开头提到过的4个监听器中的.法与加载项.中组件到IOC容器中，.所有需要回调的监听器都是从类路径下的META-INF/Spring.factories中获取，从.达到启动前后的各种定制操作。微信搜索web_resource回复爆.送你.质量技术博.。 3.SpringBoot.动配置原理 @SpringBootApplication注解 SpringBoot项.的.切都要从@SpringBootApplication这个注解开始说起。 @SpringBootApplication标注在某个类上说明： 这个类是SpringBoot的主配置类。 SpringBoot就应该运.这个类的main.法来启动SpringBoot应.。 该注解的定义如下： 1

@SpringBootConfiguration 2 @EnableAutoConfiguration 3 @ComponentScan(4 excludeFilters = {@Filter(5 6 type = FilterType.CUSTOM, 7 classes = {TypeExcludeFilter.class} 8 9), @Filter(10

可以看到SpringBootApplication注解是.个组合注解（关于组合注解.章的开头有讲到），其主要组合了.下三个注解：@SpringBootConfiguration：该注解表.这是.个SpringBoot的配置类，其实它就是.个@Configuration注解.已。@ComponentScan：开启组件扫描。 @EnableAutoConfiguration：从名字就可以看出来，就是这个类开启.动配置的。嗯，.动配置的奥秘全都在这个注解..。 @EnableAutoConfiguration注解 先看该注解是怎么定义的：

@AutoConfigurationPackage 从字.意思理解就是.动配置包。点进去可以看到就是.个@Import注解：@Import({Registrar.class})，导.了.个 Registrar的组件。关于@Import的.法.章上.也有介绍哦。 我们在Registrar类中的registerBeanDefinitions.法上打上断点，可以看到返回了.个包名，该包名其实就是主配置类所在的包。微信搜索web_resource回复爆.送你.质量技术博.。

.句话：@AutoConfigurationPackage注解就是将主配置类（@SpringBootConfiguration标注的类）的所在包及下.所有.包..的所有组件扫描到Spring容器中。所以说，默认情况下主配置类包及.包以外的组件，Spring容器是扫描不到的。@Import({AutoConfigurationImportSelector.class}) 该注解给当前配置类导.另外的N个.动配置类。（该注解详细.法上.有提及）。 配置类导.规则 那具体的导.规则是什么呢？我们来看.下源码。在开始看源码之前，先啰嗦两句。就像..哥说的，我们看源码不.全部都看，不.每..代码都弄明.是什么意思，我们只要抓住关键的地.就可以了。 我们知道AutoConfigurationImportSelector的selectImports就是.来返回需要导.的组件的全类名数组的，那么如何得到这些数组呢？ 在selectImports.法中调.了.个getAutoConfigurationEntry().法。

由于篇幅问题我就不..截图了，我直接告诉你们调.链：在 getAutoConfigurationEntry()->getCandidateConfigurations()->loadFactoryNames()。 在这. loadFactoryNames().法传.了EnableAutoConfiguration.class这个参数。先记住这个参数，等下会.到。

loadFactoryNames()中关键的三步： 从当前项.的类路径中获取所有META-INF/spring.factories这个.件下的信息。将上.获取到的信息封装成.个Map返回。从返回的Map中通过刚才传.的EnableAutoConfiguration.class参数，获取该key下的所有值。

META-INF/spring.factories探究 听我这样说完可能会有点懵，我们来看.下 META-INF/spring.factories这类.件是什么就不懵了。当然在很多第三.依赖中都会有这个.件，.般每导..个第三.的依赖，除了本.的jar包以外，还会有.个xxx-spring-boot-autoConfigure，这个就是第三.依赖..编写的.动配置类。我们现在就以spring-boot-autocongigure这个依赖来说。微信搜索web_resource回复爆.送你.质量技术博.。可以看到EnableAutoConfiguration下.有很多类，这些就是我们项.进..动配置的类。

.句话：将类路径下META-INF/spring.factories..配置的所有EnableAutoConfiguration的值加.到Spring容器中。HttpEncodingAutoConfiguration 通过上..式，所有的.动配置类就被导进主配置类中了。但是这么多的配置类，明显有很多.动配置我们平常是没有使.到的，没理由全部都.效吧。 接下来我们以HttpEncodingAutoConfiguration为例来看.个.动配置类是怎么.作的。为啥选这个类呢？主要是这个类.较的简单典型。 先看.下该类标有的注解： 1 @Configuration 2 @EnableConfigurationProperties({HttpProperties.class}) 3 @ConditionalOnWebApplication(4 type = Type.SERVLET 5) 6 @ConditionalOnClass({CharacterEncodingFilter.class}) 7 @ConditionalOnProperty(8 prefix = "spring.http.encoding", 9 10 value = {"enabled"}, 11 12 matchIfMissing = true 13) 14 public class HttpEncodingAutoConfiguration { } @Configuration：标记为配置类。 @ConditionalOnWebApplication：web应.下才.效。 @ConditionalOnClass：指定的类（依赖）存在才.效。 @ConditionalOnProperty：主配置.件中存在指定的属性才.效。@EnableConfigurationProperties({HttpProperties.class})：启动指定类的ConfigurationProperties功能；将配置.件中对应的值和HttpProperties绑定起来；并把HttpProperties加.到IOC容器中。 因为@EnableConfigurationProperties({HttpProperties.class})把配置.件中的配置项与当前HttpProperties类绑定上 了。然后在HttpEncodingAutoConfiguration中.引.了HttpProperties，所以最后就能在HttpEncodingAutoConfiguration中使.配置.件中的值了。 最终通过@Bean和.些条件判断往容器中添加组件，实现.动配置。（当然该Bean中属性值是从HttpProperties中获取） HttpProperties HttpProperties通过@ConfigurationProperties注解将配置.件与..属性绑定。所有在配置.件中能配置的属性都是在xxxProperties类中封装着；配置.件能配置什么就可以参照某个功能对应的这个属性类。微信搜索web_resource回复爆.送你.质量技术博.。

1 @ConfigurationProperties(2 prefix = "spring.http" 3 4)//从配置.件中获取指定的值和bean的属性进.绑定 5 public class HttpProperties { } .结： SpringBoot启动会加载.量的.动配置类。 我们看需要的功能有没有SpringBoot默认写好的.

动配置类。 我们再来看这个.动配置类中到底配置了那些组件（只要我们要.的组件有，我们就不需要再来配置了）。给容器中.动配置类添加组件的时候，会从properties类中获取某些属性。我们就可以在配置.件中指定这些属性的值。 xxxAutoConfiguration：.动配置类给容器中添加组件。 xxxProperties：封装配置.件中相关属性。 不知道.伙伴们有没有发现，很多需要待加载的类都放在类路径下的META-INF/Spring.factories.件下，.不是直接写死这代码中，这样做就可以很.便我们..或者是第三.去扩展，我们也可以实现..starter，让SpringBoot去加载。现在明.为什么SpringBoot可以实现零配置，开箱即.了吧！ -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. 经济疲软，公司盈利困难，程序员这样也能涨.资？

2.

标星1.3W！GitHub热榜第.，全.最.掰的12306抢票神器 3.Java最常.的208道.试题

4.

在浏览器输.URL回.之后发.了什么？ 5.接私活必备的10个开源项.

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试官：集群部署时，分布式session如何实现？ 王炸炸Java后端 2019-09-30 点击上.Java后端，选择"设为星标"优质.章，及时送达

原.来.GitHub开源社区Doocs，欢迎Star此项.，如果你有独到的.解，同样可以参与贡献此项.。 .试题 集群部署时的分布式session如何实现？ .试官.理分析 .试官问了你.堆dubbo是怎么玩.的，你会玩.dubbo就可以把单块系统弄成分布式系统，然后分布式之后接踵.来的就是.堆问题，最.的问题就是分布式事务、接.幂等性、分布式锁，还有最后.个就是分布式session。当然了，分布式系统中的问题何.这么.点，.常之多，复杂度很.，但是这.就是说下常.的.个，也是.试的时候常问的.个。 .试题剖析 session是啥？ 浏览器有个cookie，在.段时间内这个cookie都存在，然后每次发请求过来都带上.个特殊的jsessionidcookie，就根据这个东西，在服务端可以维护.个对应的session域，..可以放点数据。.般只要你没关掉浏览器，cookie还在，那么对应的那个 session就在，但是如果cookie没了，session也就没了。常.于什么购物.之类的东西，还有登录状态保存之类的。这个不多说了，懂Java的都该知道这个。单块系统的时候这么玩.session没问题，但是你要是分布式系统呢，那么多的服务。 session状态在哪.维护啊？其实.法很多，但是常.常.的是以下.种： 完全不.session 使.JWTToken储存...份，然后再从数据库或者cache中获取其他的信息。这样.论请求分配到哪个服务器都.所谓。tomcat + redis 这个其实还挺.便的，就是使.session的代码，跟以前.样，还是基于tomcat原.的session.持即可，然后就是..个叫做TomcatRedisSessionManager的东西，让所有我们部署的tomcat都将session数据存储到redis即可。在tomcat的配置.件中配置： <ValveclassName="com.orangefunction.tomcat.redissessions.RedisSessionHandlerValve"/> <ManagerclassName="com.orangefunction.tomcat.redissessions.RedisSessionManager" host="{redis.host}" port="{redis.port}" database="{redis.dbnum}" maxInactiveInterval="60"/> 复制代码然后指定redis的host和port就ok了。 <ValveclassName="com.orangefunction.tomcat.redissessions.RedisSessionHandlerValve"/> <ManagerclassName="com.orangefunction.tomcat.redissessions.RedisSessionManager" sentinelMaster="mymaster" sentinels=":26379,:26379,:26379" maxInactiveInterval="60"/> 复制代码还可以.上.这种.式基于red is哨兵.持的red is.可.集群来保存sess ion数据，都是ok的。微信搜索web_resource获取更多推送 spring session + redis 第.种.式会与tomcat容器重耦合，如果我要将web容器迁移成 jetty，难道还要重新把 jetty都配置.遍？因为上.那种 tomcat+redis的.式好.，但是会严重依赖于web容器，不好将代码移植到其他web容器上去，尤其是你要是换了技术栈咋整？.如换成了spr ingcloud或者是spr ingboot之类的呢？所以现在.较好的还是基于Java.站式解决.案，也就是 spring。.家spr ing基本上包掉了.部分我们需要使.的框架，sp irngcloud做微服务，spr ingboot做脚.架，所以.sp ing session是.个很好的选择。在pom.xm l中配置： org.springframework.session spring-session-data-redis 1.2.1.RELEASE redis.clients jedis 2.8.1 复制代码在spr ing配置.件中配置： <beanid="redisHttpSessionConfiguration" class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttpSessionConfiguration"> <propertyname="maxInactiveIntervalInSeconds"value="600"/> <beanid="jedisPoolConfig"class="redis.clients.jedis.JedisPoolConfig"> <propertyname="maxTotal"value="100"/> <propertyname="maxIdle"value="10"/> <beanid="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"destroy-method="destroy"> <propertyname="hostName"value="${redis_hostname}"/> <propertyname="port"value="${redis_port}"/> <propertyname="password"value="${redis_pwd}"/> <propertyname="timeout"value="3000"/>

<propertyname="usePool"value="true"/> <propertyname="poolConfig"ref="jedisPoolConfig"/> 复制代码在web.xm l中配置： springSessionRepositoryFilter org.springframework.web.filter.DelegatingFilterProxy springSessionRepositoryFilter /* 复制代码.例代码： @Controller @RequestMapping("/test")publicclassTestController{ @RequestMapping("/putIntoSession")@ResponseBody publicStringputIntoSession(HttpServletRequestrequest,Stringusername){request.getSession().setAttribute("name", “leo”); return"ok";} @RequestMapping("/getFromSession")@ResponseBody publicStringgetFromSession(HttpServletRequestrequest,Modelmodel) {Stringname=request.getSession().getAttribute("name"); returnname; }} 复制代码上.的代码就是ok的，给spingsession配置基于redis来存储session数据，然后配置了.个springsession的过滤器，这样的话，session相关操作都会交给springsession来管了。接着在代码中，就.原.的session操作，就是直接基于 springsesion从redis中获取数据了。实现分布式的会话有很多种.式，我说的只不过是.较常.的.种.式，tomcat+redis早期.较常.，但是会重耦合到tomcat中；近些年，通过springsession来实现。 作者|yanglbme 来源|https://github.com/doocs/advanced-java -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2. IDEA远程.键部署 Spring Boot到 Docker 3.这 26条，你赞同.个？ 4.7个开源的 Spring Boot前后端分离项.

5.如何设计 API接.，实现统.格式返回?

喜欢.章，点个在看 阅读原.声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试必问的MySQL四种隔离级别，看完吊打.试官 游泳的.头Java后端 2019-11-30 点击上.Java后端，选择设为星标 优质.章，及时送达

作者 |游泳的.头来源 |www.jianshu.com/p/8d735db9c2c0 什么是事务 事务是应.程序中.系列严密的操作，所有操作必须成功完成，否则在每个操作中所作的所有更改都会被撤消。也就是事务具有原.性，.个事务中的.系列的操作要么全部成功，要么.个都不做。 事务的结束有两种，当事务中的所以步骤全部成功执.时，事务提交。如果其中.个步骤失败，将发.回滚操作，撤消撤消之前到事务开始时的所以操作。 事务的 ACID 事务具有四个特征：原.性（Atom icity）、.致性（Cons istency）、隔离性（ Isolation）和持续性（Durab ility）。这四个特性简称为AC ID特性。 原.性。事务是数据库的逻辑.作单位，事务中包含的各操作要么都做，要么都不做.致性。事务执.的结果必须是使数据库从.个.致性状态变到另.个.致性状态。因此当数据库只包含成功事务提交的结果时，就说数据库处于.致性状态。如果数据库系统运.中发.故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的修改有.部分已写.物理数据库，这时数据库就处于.种不正确的状态，或者说是不.致的状态。隔离性。.个事务的执.不能其它事务.扰。即.个事务内部的操作及使.的数据对其它并发事务是隔离的，并发执.的各个事务之间不能互相.扰。持续性。也称永久性，指.个事务.旦提交，它对数据库中的数据的改变就应该是永久性的。接下来的其它操作或故障不应该对其执.结果有任何影响。 Mysql的四种隔离级别 SQL标准定义了4类隔离级别，包括了.些具体规则，.来限定事务内外的哪些改变是可.的，哪些是不可.的。低级别的隔离级.般.持更.的并发处理，并拥有更低的系统开销。 Read Uncommitted（读取未提交内容） 在该隔离级别，所有事务都可以看到其他未提交事务的执.结果。本隔离级别很少.于实际应.，因为它的性能也不.其他级别好多少。读取未提交的数据，也被称之为脏读（D irtyRead）。 Read Committed（读取提交内容） 这是.多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满.了隔离的简单定义：.个事务只能看.已经提交事务所做的改变。这种隔离级别也.持所谓的不可重复读（Nonrepeatab leRead），因为同.事务的其他实例在该实例处理其间可能会有新的comm it，所以同.se lect可能返回不同结果。 Repeatable Read（可重读） 这是MySQL的默认事务隔离级别，它确保同.事务的多个实例在并发读取数据时，会看到同样的数据.。不过理论上，这会导致另.个棘.的问题：幻读（PhantomRead）。简单的说，幻读指当..读取某.范围的数据.时，另.个事务.在该范围内插.了新.，当..再读取该范围的数据.时，会发现有新的“幻影”.。InnoDB和Fa lcon存储引擎通过多版本并发控制（MVCC，Mu ltiversionConcurrencyControl）机制解决了该问题。Serializable（可串.化） 这是最.的隔离级别，它通过强制事务排序，使之不可能相互冲突，从.解决幻读问题。简.之，它是在每个读的数据.上加上共享锁。在这个级别，可能导致.量的超时现象和锁竞争。 这四种隔离级别采取不同的锁类型来实现，若读取的是同.个数据的话，就容易发.问题。例如： 脏读 (Drity Read)：某个事务已更新.份数据，另.个事务在此时读取了同.份数据，由于某些原因，前.个Ro llBack了操作，则后.个事务所读取的数据就会是不正确的。不可重复读 (Non-repeatableread):在.个事务的两次查询之中数据不.致，这可能是两次查询过程中间插.了.个事务更新的原有的数据。幻读 (Phantom Read):在.个事务的两次查询中数据笔数不.致，例如有.个事务查询了.列 (Row)数据，.另.个事务却

在此时插.了新的.列数据，先前的事务在接下来的查询中，就有.列数据是未查询出来的，如果此时插.和另外.个事务插.的数据，就会报错。 在MySQL中，实现了这四种隔离级别，分别有可能产.问题如下所.：

测试 Mysql的隔离级别 下.，将利.MySQL的客.端程序，我们分别来测试.下这.种隔离级别。测试数据库为demo，表为test；表结构：

两个命令.客.端分别为A，B；不断改变A的隔离级别，在B端修改数据。 将A的隔离级别设置为 read uncommitted(未提交读)

A：启动事务，此时数据为初始状态B：启动事务，更新数据，但不提交

A：再次读取数据，发现数据已经被修改了，这就是所谓的"脏读"

B：回滚事务

A：再次读数据，发现数据变回初始状态

经过上.的实验可以得出结论，事务B更新了.条记录，但是没有提交，此时事务A可以查询出未提交记录。造成脏读现象。未提交读是最低的隔离级别。 将客.端 A的事务隔离级别设置为 read committed(已提交读) A：启动事务，此时数据为初始状态

B：启动事务，更新数据，但不提交

A：再次读数据，发现数据未被修改

B：提交事务

A：再次读取数据，发现数据已发.变化，说明B提交的修改被事务中的A读到了，这就是所谓的"不可重复读"

经过上.的实验可以得出结论，已提交读隔离级别解决了脏读的问题，但是出现了不可重复读的问题，即事务A在两次查询的数据不.致，因为在两次查询之间事务B更新了.条数据。已提交读只允许读取已提交的记录，但不要求可重复读。将A的隔离级别设置为 repeatable read(可重复读)

A：启动事务，此时数据为初始状态

B：启动事务，更新数据，但不提交

A：再次读取数据，发现数据未被修改

B：提交事务

A：再次读取数据，发现数据依然未发.变化，这说明这次可以重复读了

B：插..条新的数据，并提交

A：再次读取数据，发现数据依然未发.变化，虽然可以重复读了，但是却发现读的不是最新数据，这就是所谓的"幻读"

A：提交本次事务，再次读取数据，发现读取正常了

由以上的实验可以得出结论，可重复读隔离级别只允许读取已提交记录，.且在.个事务两次读取.个记录期间，其他事务部的更新该记录。但该事务不要求与其他事务可串.化。例如，当.个事务可以找到由.个已提交事务更新的记录，但是可能产.幻读问题 (注意是可能，因为数据库对隔离级别的实现有所差别)。像以上的实验，就没有出现数据幻读的问题。将A的隔离级别设置为可串.化 (Serializable)

A：启动事务，此时数据为初始状态

B：发现B此时进.了等待状态，原因是因为A的事务尚未提交，只能等待（此时，B可能会发.等待超时）

A：提交事务

B：发现插.成功

serializable完全锁定字段，若.个事务来查询同.份数据就必须等待，直到前.个事务完成并解除锁定为.。是完整的隔离级别，会锁定对应的数据表格，因.会有效率的问题。 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. 如果我是.试官，我会问你 Spring那些问题?

2.Spring Boot整合 Spring-cache

3.我们再来聊.聊 Java的单例吧

4.我采访了.位 Pornhub .程师

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试挂在DubboRPC？我把常问.试题整理好了，来拿！ hu1991dieJava后端 2019-09-26 点击上.Java后端，选择"设为星标"优质.章，及时送达

作者|hu1991die链接|www.jianshu.com/p/78f72ccf0377上.篇：计算机专业的学.也太太太太太惨了吧？ RPC.常重要，很多..试的时候都挂在了这个地.！你要是还不懂RPC是什么？他的基本原理是什么？你.定要把下边的内容记起来！好 好研究.下！特别是.中给出的.张关于RPC的基本流程图，重点中的重点，DubboRPC的基本执.流程就是他，RPC框架的基本原理也是他，别说我没告诉你！看了下边的内容你要掌握的内容如下，当然还有很多：

RPC的由来，是怎样.步步演进出来的；RPC的基本架构是什么；RPC的基本实现原理，就是下边的这张图，重点中的重点；REST和SOAP、RPC有何区别呢?整个调.的过程经历了哪.步和SpringMVC的执.流程.样，相当重要；

随着互联.的发展，.站应.的规模不断扩.，常规的垂直应.架构已.法应对，分布式服务架构以及流动计算架构势在必.，亟需.个治理系统确保架构有条不紊的演进。

1、单.应.架构 当.站流量很.时，只需.个应.，将所有功能都部署在.起，以减少部署节点和成本。此时，.于简化增删改查.作量的数据访问框架 (ORM)是关键。 2、垂直应.架构 当访问量逐渐增.，单.应.增加机器带来的加速度越来越.，将应.拆成互不相.的.个应.，以提升效率。此时，.于加速前端..开发的Web框架(MVC)是关键。 3、分布式服务架构 当垂直应.越来越多，应.之间交互不可避免，将核.业务抽取出来，作为独.的服务，逐渐形成稳定的服务中.，使前端应.能更快速的响应多变的市场需求。 此时，.于提.业务复.及整合的分布式服务框架(RPC)，提供统.的服务是关键。例如：各个团队的服务提供.就不要各.实现.套序列化、反序列化、.络框架、连接池、收发线程、超时处理、状态机等"业务之外"的重复技术劳动，造成整体的低效。PS：其实上述三个原因也是为什么要有Dubbo的原因！不信你去Dubbo官.去看！流动计算架构PS：这个属于扩展内容，摘.Dubbo官.，属于架构演进的.个过程 当服务越来越多，容量的评估，.服务资源的浪费等问题逐渐显现，此时需增加.个调度中.基于访问压.实时管理集群容量，提.集群利.率。此时，.于提.机器利.率的资源调度和治理中.(SOA)是关键。 4、另外.个原因 就是因为在.个进程内（应.分布在不同的机器上），.法共.内存空间，或者在.台机器内通过本地调..法完成相关的需求，.如不同的系统之间的通讯，甚.不同组织之间的通讯。此外由于机器的横向扩展，需要在多台机器组成的集群上部署应.等等。 所以，统.RPC框架来解决提供统.的服务。

RPC（Remote Procedure Call Protocol）远程过程调.协议，它是.种通过.络从远程计算机程序上请求服务，.不需要了解底层.络技术的协议。简.之，RPC使得程序能够像访问本地系统资源.样，去访问远端系统资源。.较关键的.些..包括：通讯协议、序列化、资源（接.）描述、服务框架、性能、语..持等。

简单的说，RPC就是从.台机器(客.端)上通过参数传递的.式调.另.台机器(服务器)上的.个函数或.法(可以统称为服务)并得到返回的结果。

.个基本的RPC架构..应该.少包含以下4个组件：1、客.端（Client）:服务调..（服务消费者）2、客.端存根（ClientStub）:存放服务端地址信息，将客.端的请求参数数据信息打包成.络消息，再通过.络传输发送给服务端3、服务端存根（ServerStub）:接收客.端发送过来的请求消息并进.解包，然后再调.本地服务进.处理4、服务端（Server）:服务的真正提供者

具体调.过程： 服务消费者（client客.端）通过调.本地服务的.式调.需要消费的服务； 客.端存根（clientstub）接收到调.请求后负责将.法、.参等信息序列化（组装）成能够进..络传输的消息体； 客.端存根（clientstub）找到远程的服务地址，并且将消息通过.络发送给服务端； 服务端存根（serverstub）收到消息后进.解码（反序列化操作）； 服务端存根（serverstub）根据解码结果调.本地的服务进.相关处理； 本地服务执.具体业务逻辑并将处理结果返回给服务端存根（serverstub）； 服务端存根（serverstub）将返回结果重新打包成消息（序列化）并通过.络发送.消费.； 客.端存根（clientstub）接收到消息，并进.解码（反序列化）； 服务消费.得到最终结果； .RPC框架的实现.标则是将上.的第2-10步完好地封装起来，也就是把调.、编码/解码的过程给封装起来，让..感觉上像调.本地服务.样的调.远程服务。

1、REST 可以看着是HTTP协议的.种直接应.，默认基于JSON作为传输格式,使.简单,学习成本低效率.,但是安全性较低。 2、SOAP SOAP是.种数据交换协议规范,是.种轻量的、简单的、基于XML的协议的规范。.SOAP可以看着是.个重量级的协议，基于XML、SOAP在安全..是通过使.XML-Security和XML-Signature两个规范组成了WS-Security来实现安全控制的,当前已经得到了各个.商的.持。 它有什么优点？简单总结为：易.、灵活、跨语.、跨平台。 3、SOA .向服务架构，它可以根据需求通过.络对松散耦合的粗粒度应.组件进.分布式部署、组合和使.。服务层是SOA的基础，可以直接被应.调.，从.有效控制系统中与软件代理交互的.为依赖性。 SOA是.种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接.进.通讯，不涉及底层编程接.和通讯模型。SOA可以看作是B/S模型、XML（标准通.标记语.的.集）/WebService技术之后的.然延伸。 4、REST和SOAP、RPC有何区别呢? 没什么太.区别，他们的本质都是提供可.持分布式的基础服务，最.的区别在于他们各.的的特点所带来的不同应.场景。

1、如何确定客.端和服务端之间的通信协议？2、如何更.效地进..络通信？3、服务端提供的服务如何暴露给客.端？4、客.端如何发现这些暴露的服务？5、如何更.效地对请求对象和响应结果进.序列化和反序列化操作?

1、需要有.常.效的.络通信，.如.般选择Netty作为.络通信框架；2、需要有.较.效的序列化框架，.如.歌的Protobuf序列化框架；3、可靠的寻址.式（主要是提供服务的发现），.如可以使.Zookeeper来注册服务等等；4、如果是带会话（状态）的RPC调.，还需要有会话和状态保持的功能；

1、动态代理 .成ClientStub（客.端存根）和ServerStub（服务端存根）的时候需要.到Java动态代理技术，可以使.JDK提供的原.的动态代理机制，也可以使.开源的：CGLib代理，Javassist字节码.成技术。 2、序列化和反序列化 在.络中，所有的数据都将会被转化为字节进.传送，所以为了能够使参数对象在.络中进.传输，需要对这些参数进.序列化和反序列化操作。序列化：把对象转换为字节序列的过程称为对象的序列化，也就是编码的过程。反序列化：把字节序列恢复为对象的过程称为对象的反序列化，也就是解码的过程。.前.较.效的开源序列化框架：如Kryo、FastJson和Protobuf等。 3、NIO通信 出于并发性能的考虑，传统的阻塞式IO显然不太合适，因此我们需要异步的IO，即NIO。Java提供了NIO的解决.案，Java7也提供了更 优秀的NIO.2.持。可以选择Netty或者MINA来解决NIO数据传输的问题。 4、服务注册中.可选：Redis、Zookeeper、Consul、Etcd。.般使.ZooKeeper提供服务注册与发现功能，解决单点故障以及分布式部署的问题(注册中.)。

1、RMI 利.java.rmi包实现，基于Java远程.法协议(JavaRemoteMethodProtocol)和java的原.序列化。 2、Hessian 是.个轻量级的remotingonhttp.具，使.简单的.法提供了RMI的功能。基于HTTP协议，采..进制编解码。 3、protobuf-rpc-pro是.个Java类库，提供了基于 Google的 ProtocolBu.ers协议的远程.法调.的框架。基于 Netty底层的 NIO技术。.持 TCP重./ keep-alive、SSL加密、RPC调.取消操作、嵌.式.志等功能。 4、Thrift 是.种可伸缩的跨语.服务的软件框架。它拥有功能强.的代码.成引擎，.缝地.持C + +，C#，Java，Python和PHP和Ruby。thrift允许你定义.个描述.件，描述数据类型和服务接..。依据该.件，编译器.便地.成RPC客.端和服务器通信代码。 最初由facebook开发.做系统内个语.之间的RPC通信，2007年由facebook贡献到apache基.，现在是apache下的opensource之.。.持多种语.之间的RPC.式的通信：php语.client可以构造.个对象，调.相应的服务.法来调.java语.的服务，跨越语.的C/S RPC调.。底层通讯基于SOCKET。 5、Avro 出.Hadoop之.Doug Cutting,在Thrift已经相当流.的情况下推出Avro的.标不仅是提供.套类似Thrift的通讯中间件,更是要建..个新的，标准性的云计算的数据交换和存储的Protocol。.持HTTP，TCP两种协议。 6、Dubbo Dubbo是阿.巴巴公司开源的.个.性能优秀的服务框架，使得应.可通过.性能的 RPC实现服务的输出和输.功能，可以和 Spring框架.缝集成。

PS：这张图.常重点，是PRC的基本原理，请.家.定记住！ 也就是说两台服务器A，B，.个应.部署在A服务器上，想要调.B服务器上应.提供的函数/.法，由于不在.个内存空间，不能直接调.，需要通过.络来表达调.的语义和传达调.的数据。.如说，A服务器想调.B服务器上的.个.法： UsergetUserByName(StringuserName) 1、建.通信 .先要解决通讯的问题：即A机器想要调.B机器，.先得建.起通信连接。 主要是通过在客.端和服务器之间建.TCP连接，远程过程调.的所有交换的数据都在这个连接.传输。连接可以是按需连接，调.结束后就断掉，也可以是.连接，多个远程过程调.共享同.个连接。 通常这个连接可以是按需连接（需要调.的时候就先建.连接，调.结束后就..断掉），也可以是.连接（客.端和服务器建.起连接

之后保持.期持有，不管此时有.数据包的发送，可以配合.跳检测机制定期检测建.的连接是否存活有效），多个远程过程调.共享同.个连接。 2、服务寻址 要解决寻址的问题，也就是说，A服务器上的应.怎么告诉底层的RPC框架，如何连接到B服务器（如主机或IP地址）以及特定的端.，.法的名称名称是什么。 通常情况下我们需要提供B机器（主机名或IP地址）以及特定的端.，然后指定调.的.法或者函数的名称以及.参出参等信息，这样才能完成服务的.个调.。 可靠的寻址.式（主要是提供服务的发现）是RPC的实现基.，.如可以采.Redis或者Zookeeper来注册服务等等。

2.1、从服务提供者的.度看：

当服务提供者启动的时候，需要将..提供的服务注册到指定的注册中.，以便服务消费者能够通过服务注册中.进.查找；

当服务提供者由于各种原因致使提供的服务停.时，需要向注册中.注销停.的服务；

服务的提供者需要定期向服务注册中.发送.跳检测，服务注册中.如果.段时间未收到来.服务提供者的.跳后，认为该服务提供者已经停.服务，则将该服务从注册中.上去掉

2.2、从调.者的.度看： 服务的调.者启动的时候根据..订阅的服务向服务注册中.查找服务提供者的地址等信息； 当服务调.者消费的服务上线或者下线的时候，注册中.会告知该服务的调.者； 服务调.者下线的时候，则取消订阅。 3、.络传输3.1、序列化 当A机器上的应.发起.个RPC调.时，调..法和其.参等信息需要通过底层的.络协议如TCP传输到B机器，由于.络协议是基于.进制的，所有我们传输的参数数据都需要先进.序列化（Serialize）或者编组（marshal）成.进制的形式才能在.络中进.传输。然后通过寻址操作和.络传输将序列化或者编组之后的.进制数据发送给B机器。 3.2、反序列化 当B机器接收到A机器的应.发来的请求之后，.需要对接收到的参数等信息进.反序列化操作（序列化的逆操作），即将.进制信息恢复为内存中的表达.式，然后再找到对应的.法（寻址的.部分）进.本地调.（.般是通过.成代理Proxy去调.,通常会有JDK动态代理、CGLIB动态代理、Javassist.成字节码技术等），之后得到调.的返回值。 4、服务调. B机器进.本地调.（通过代理Proxy和反射调.）之后得到了返回值，此时还需要再把返回值发送回A机器，同样也需要经过序列化操作，然后再经过.络传输将.进制数据发送回A机器，.当A机器接收到这些返回值之后，则再次进.反序列化操作，恢复为内存中的表达.式，最后再交给A机器上的应.进.相关处理，.般是业务逻辑处理操作。 通常，经过以上四个步骤之后，.次完整的RPC调.算是完成了，另外可能因为.络抖动等原因需要重试等。 -END- 如果看到这.，说明你喜欢这篇.章，帮忙转发.下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进.. .告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2.

计算机专业的学.也太太太太太惨了吧?

3..试官：说.说 Spring Boot .动配置原理

4.在浏览器输. URL回.之后发.了什么？ 5.接私活必备的 10个开源项.

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试被问离职原因该如何回答？ cldmxwJava后端 2019-09-30 点击上.Java后端，选择"设为星标"优质.章，及时送达 因，求职者回答得不妥随时可能被淘汰掉。那么，怎样的离职原因能让HR们接受呢？ 求职者应聘时经常碰到.试官问及离职原 1、寻求更.的发展 这是.较经典的回答。.般的回答模式是：我在之前的.作中享受了乐趣（或者和.家相处得很好，再或者学到了很多东西，等等），但是我希望在这个领域更好地发展，去拓展新的未来，去挑战.我（如果是跨.业跳槽的话，可以说想学习更多领域的知识，或者说在这个新.业更能发挥..所.等）。 另外，还可以说现有的企业岗位设置难以满...职业进.步发展的要求，所以只能辞职。.如该企业不太重视财务管理，作为会计，想往更专业的职业.向发展，那就只能离开那个企业。 这个问题回答的关键在于，要突出..是"转职"，.."离职"；..是"为了提升"，.不是"从原公司离开"。.论如何回答，都要向.试官展.出..是为了寻求更多的责任，更好的发展机会和更.的挑战.选择新的公司，并不是单纯由于..对之前.作处境的失望.不得不离开。突出了这.点的回答能够让应聘者在.试中处于.理上的主动。当然，为了维护这个观念，切记要避免对之前.作的不满和负.评价的流露。 2、个.原因 个.原因可以包括.家临时有事，突然中断.作；家庭地址发.变化，导致交通距离太远.法继续在原公司上班；.体原因（包括..病、怀孕..孩等）中断.作；继续深造、脱产学习导致.作需要中断等。 个.原因作为离职原因，是.较让HR接受的，但在描述时也要合情合理，不要让HR认为那是压根就没发.的事，完全是忽悠.的。频繁换.作的.也不适合.这个原因。久未上班的..这个原因.较好。 3、原公司发.了重.客观变化 公司重组或部.内部变动，导致.作内容发.重.变化，.法再继续履.劳动合同，或者直接被裁员等。例如某些职位是为配合企业完成阶段

性任务.特设的，因此当这.阶段任务完成.找不到适合岗位时，就得离开。被裁员的最好不要实话实说，要不HR.般会认为你没能.。将原因归结为公司客观情况变化，导致..职业要重新规划会好.些。 4、和企业.化不匹配 企业.化可以是.些氛围等东西，.如强制加班、..不.由、管理混乱等。在描述这个原因时要注意：每个企业都有这样那样的.化，不能太批驳哪.不好，要不HR就会认为是你..适应能.差了。说出的具体原因，要举例说明，让HR确实认为是企业的责任远.于你的责任，那你就成功了。 5、.作环境差 .作环境差可以包含.作温度.、职业病隐患、某些产品过敏等。这个原因在描述时要合情合理，让HR认为确实客观存在才.。 6、待遇太低 待遇太低是.部分换.作的.的实际离职原因。但在描述这个原因时，我们可以委婉.点，可以说希望..的.作更上.个台阶，待遇上也能上个台阶，迎接更具挑战性的.作，让..职业更深层次的发展。如果直接说那公司待遇低，那HR可能认为是你..能.不.导致待遇低。要知道.才市场上也是.分钱.分货的道理。有的时候采.其中.种原因回答就可以，有时需同时采..种.式组合回答；不管你怎么回答都有.定.险，因为考官还会追问下去，因此你还必须准备好对你有利的解释。频繁换.作的.，离职原因怎么说都会被HR怀疑，所以.作最好别换那么频繁，或者换了不体现在简历上。 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2. IDEA远程.键部署 Spring Boot到 Docker 3.这 26条，你赞同.个？ 4.7个开源的 Spring Boot前后端分离项.

5.如何设计 API接.，实现统.格式返回?

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试题：C lass.forName和C lassLoader有什么区别？ 纪莫Java后端 2019-11-12 点击上.Java后端，选择设为星标 优质.章，及时送达

作者 |纪莫来源 |cnblogs.com/jimoer/p/9185662.html 在java中C lass.forName()和C lassLoader都可以对类进.加载。C lassLoader就是遵循双亲委派模型最终调.启动类加载器的类加载器，实现的功能是"通过.个类的全限定名来获取描述此类的.进制字节流"，获取到.进制流后放到JVM中。 Class.forName().法实际上也是调.的CLassLoader来实现的。Class.forName(StringclassName)；这个.法的源码是：

最后调.的.法是 forName0这个.法，在这个 forName0.法中的第.个参数被默认设置为了true，这个参数代表是否对加载的类进.初始化，设置为true时会类进.初始化，代表会执.类中的静态代码块，以及对静态变量的赋值等操作。 也可以调.C lass.forName(Stringname,booleaninitialize,ClassLoaderloader).法来.动选择在加载类的时候是否要对类进.初始化。C lass.forName(Stringname,booleaninitialize,ClassLoaderloader)的源码如下：

源码中的注释只摘取了.部分，其中对参数initialize的描述是：if{@codetrue}theclasswillbeinitialized.意思就是说：如果参数为true，则加载的类将会被初始化。 举例: 下.还是举例来说明结果吧：.个含有静态代码块、静态变量、赋值给静态变量的静态.法的类。

测试.法：

运.结果：

根据运.结果得出Class.forName加载类是将类进了初始化，.ClassLoader的loadClass并没有对类进.初始化，只是把类加载到了虚拟机中。 应.场景 在我们熟悉的Spring框架中的IOC的实现就是使.的ClassLoader。 .在我们使.JDBC时通常是使.Class.forName().法来加载数据库连接驱动。这是因为在JDBC规范中明确要求Driver(数据库驱动)类必须向DriverManager注册...。以MySQL的驱动为例解释：

我们看到Driver注册到DriverManager中的操作写在了静态代码块中，这就是为什么在写JDBC时使.Class.forName()的原因了。 -END-

喜欢.章，点个在看

声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！ .试题：InnoDB中.棵B+树能存多少.数据？ 李平Java后端 2019-10-04 点击上.Java后端，选择设为星标优质.章，及时送达

作者：李平|来源：个.博客 上.篇：彻底理解 Cookie，Session，Token.、InnoDB.棵B+树可以存放多少.数据？ InnoDB.棵B+树可以存放多少.数据？这个问题的简单回答是：约2千万。为什么是这么多呢？因为这是可以算出来的，要搞清楚这个问题，我们先从InnoDB索引数据结构、数据组织.式说起。 我们都知道计算机在存储数据的时候，有最.存储单元，这就好.我们今天进.现.的流通最.单位是..。在计算机中磁盘存储数据最.单元是扇区，.个扇区的..是512字节，..件系统（例如XFS/EXT4）他的最.单元是块，.个块的..是4k，.对于我们的InnoDB存储引擎也有..的最.储存单元——.（Page），.个.的..是16K。 .、下..张图可以帮你理解最.存储单元： .件系统中.个.件..只有1个字节，但不得不占磁盘上4KB的空间。

innodb的所有数据.件（后缀为ibd的.件），他的..始终都是16384（16k）的整数倍。

磁盘扇区、.件系统、 InnoDB存储引擎都有各.的最.存储单元。

在MySQL中我们的 InnoDB.的..默认是16k，当然也可以通过参数设置：

数据表中的数据都是存储在.中的，所以.个.中能存储多少.数据呢？假设..数据的..是1k，那么.个.可以存放16.这样的数据。 如果数据库只按这样的.式存储，那么如何查找数据就成为.个问题，因为我们不知道要查找的数据存在哪个.中，也不可能把所有的.遍历.遍，那样太慢了。所以.们想了.个办法，.B+树的.式组织这些数据。如图所.：

我们先将数据记录按主键进.排序，分别存放在不同的.中（为了便于理解我们这..个.中只存放3条记录，实际情况可以存放很多），除了存放数据的.以外，还有存放键值+指针的.，如图中pagenumber=3的.，该.存放键值和指向数据.的指针，这样的.由N个键值+指针组成。当然它也是排好序的。这样的数据组织形式，我们称为索引组织表。现在来看下，要查找.条数据，怎么查？ 如select*fromuserwhereid=5; 这.id是主键,我们通过这棵B+树来查找，.先找到根，你怎么知道user表的根.在哪呢？其实每张表的根.位置在表空间.件中是固定的，即pagenumber=3的.（这点我们下.还会进.步证明），找到根.后通过.分查找法，定位到id=5的数据应该在指针P5指向的.中，那么进.步去pagenumber=5的.中查找，同样通过.分查询法即可找到id=5的记录： |5|zhao2|27| 现在我们清楚了InnoDB中主键索引B+树是如何组织数据、查询数据的，我们总结.下： 1、InnoDB存储引擎的最.存储单元是.，.可以.于存放数据也可以.于存放键值+指针，在B+树中叶.节点存放数据，.叶.节点存放键值+指针。2、索引组织表通过.叶.节点的.分查找法以及指针确定数据在哪个.中，进.在去数据.中查找到需要的数据； 三、那么回到我们开始的问题，通常.棵B+树可以存放多少.数据？ 这.我们先假设B+树.为2，即存在.个根节点和若.个叶.节点，那么这棵B+树的存放总记录数为：根节点指针数*单个叶.节点记录.数。 上.我们已经说明单个叶.节点（.）中的记录数=16K/1K=16。（这.假设..记录的数据..为1k，实际上现在很多互联.业务数据记录..通常就是1K左右）。 那么现在我们需要计算出.叶.节点能存放多少指针？ 其实这也很好算，我们假设主键ID为bigint类型，.度为8字节，.指针..在InnoDB源码中设置为6字节，这样.共14字节，我们.个.中能存放多少这样的单元，其实就代表有多少指针，即16384/14=1170。那么可以算出.棵.度为2的B+树，能存放1170*16=18720条这样的数据记录。根据同样的原理我们可以算出.个.度为3的B+树可以存放：1170*1170*16=21902400条这样的记录。 所以在InnoDB中B+树.度.般为1-3层，它就能满.千万级的数据存储。在查找数据时.次.的查找代表.次IO，所以通过主键索引查询通常只需要1-3次IO操作即可查找到数据。 四、怎么得到InnoDB主键索引B+树的.度？ 上.我们通过推断得出B+树的.度通常是1-3，下.我们从另外.个侧.证明这个结论。在InnoDB的表空间.件中，约定pagenumber为3的代表主键索引的根.，.在根.偏移量为64的地.存放了该B+树的page level。如果page level为1，树.为2，pagelevel为2，则树.为3。即B+树的.度=pagelevel+1；下.我们将从实际环境中尝试找到这个pagelevel。 在实际操作之前，你可以通过InnoDB元数据表确认主键索引根.的pagenumber为3，你也可以从《InnoDB存储引擎》这本书中得到确认。

执.结果：

可以看出数据库dbt3下的customer表、lineitem表主键索引根.的pagenumber均为3，.其他的.级索引pagenumber为4。关于.级索引与主键索引的区别请参考MySQL相关书籍，本.不在此介绍。 下.我们对数据库表空间.件做想相关的解析：

因为主键索引B+树的根.在整个表空间.件中的第3个.开始，所以可以算出它在.件中的偏移量：16384*3=49152（16384为...）。 另外根据《InnoDB存储引擎》中描述在根.的64偏移量位置前2个字节，保存了page level的值，因此我们想要的page level的值在整个.件中的偏移量为：16384*3+64=49152+64=49216，前2个字节中。 接下来我们.hexdump.具，查看表空间.件指定偏移量上的数据：linetem表的page level为2，B+树.度为page level+1=3； region表的page level为0，B+树.度为page level+1=1；customer表的pagelevel为2，B+树.度为pagelevel+1=3；

这三张表的数据量如下：

五、.结 lineitem表的数据.数为600多万，B+树.度为3，customer表数据.数只有15万，B+树.度也为3。可以看出尽管数据量差异较.，这两个表树的.度都是3，换句话说这两个表通过索引查询效率并没有太.差异，因为都只需要做3次IO。那

么如果有.张表.数是.千万，那么他的B+树.度依旧是3，查询效率仍然不会相差太.。 region表只有5.数据，当然他的B+树.度为1。 六、最后回顾.道.试题 有.道MySQL的.试题，为什么MySQL的索引要使.B+树.不是其它树形结构？.如B树？现在这个问题的复杂版本可以参考本.； 他的简单版本回答是： 因为B树不管叶.节点还是.叶.节点，都会保存数据，这样导致在.叶.节点中能保存的指针数量变少（有些资料也称为扇出），指针少的情况下要保存.量数据，只能增加树的.度，导致IO操作变多，查询性能变低； 七、总结 本.从.个问题出发，逐步介绍了InnoDB索引组织表的原理、查询.式，并结合已有知识，回答该问题，结合实践来证明。当然为了表述简单易懂，.中忽略了.些细枝末节，.如.个.中不可能所有空间都.于存放数据，它还会存放.些少量的其他字段.如pagelevel，indexnumber等等，另外还有.的填充因.也导致.个.不可能全部.于保存数据。关于.级索引数据存取.式可以参考MySQL相关书籍，他的要点是结合主键索引进.回表查询。 参考资料： 1、《MySQL技术内幕:InnoDB存储引擎》2、http://www.innomysql.com/查看-innodb表中每个的索引.度/ 原.地址： www.cnblogs.com/leefreeman/p/8315844.html编辑|Java后端技术 -END- 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. Java后端优质.章整理

2. IDEA远程.键部署 Spring Boot到 Docker 3.这 26条，你赞同.个？ 4.7个开源的 Spring Boot前后端分离项.

5.如何设计 API接.，实现统.格式返回?

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试题：Lucene、So lr、E lasticSearch Java后端 2019-12-01 1、Lucene和So lr和E lasticsearch的区别 Lucene Lucene是apache下的.个.项.，是.个开放源代码的全.检索引擎.具包，但它不是.个完整的全.检索引擎，.是.个全.检索引擎的架构，提供了完整的查询引擎和索引引擎，部分.本分析引擎。官.地址：https:// lucene.apache.org/ Solr Solr是.个.性能，采.Java5开发，基于Lucene的全.搜索服务器。同时对其进.了扩展，提供了.Lucene更为丰富的查询语.，同时实现了可配置、可扩展并对查询性能进.了优化，并且提供了.个完善的功能管理界.，是.款.常优秀的全.搜索引擎。官.地址：http://lucene.apache.org/solr/ Elasticsearch Elasticsearch跟So lr.样，也是.个基于Lucene的搜索服务器，它提供了.个分布式多..能.的全.搜索引擎，基于REST fulweb接.。官.地址：https://www.e lastic.co/products/elasticsearch 1、E lasticsearch的优缺点： 优点： 1. Elasticsearch是分布式的。不需要其他组件，分发是实时的，被叫做"Pushrep lication"。

2.

Elasticsearch完全.持ApacheLucene的接近实时的搜索。

3.

处理多租.（mu ltitenancy）不需要特殊配置，.So lr则需要更多的.级设置。

4.

Elasticsearch采.Gateway的概念，使得完备份更加简单。

5.

各节点组成对等的.络结构，某些节点出现故障时会.动分配其他节点代替其进..作。

缺点：

1.

只有.名开发者（当前E lasticsearchGitHub组织已经不只如此，已经有了相当活跃的维护者）

2.

还不够.动（不适合当前新的 IndexWarmupAPI）

2、So lr的优缺点： 优点 1. Solr有.个更.、更成熟的...、开发和贡献者社区。

2.

.持添加多种格式的索引，如：HTML、PDF、微软O ffice系列软件格式以及JSON、XML、CSV等纯.本格式。 3.Solr.较成熟、稳定。

4.

不考虑建索引的同时进.搜索，速度更快。

缺点

1.

建.索引时，搜索效率下降，实时索引搜索效率不.。

3、Elasticsearch与Solr的.较： 1..者安装都很简单； 2. Solr利.Zookeeper进.分布式管理，.Elasticsearch..带有分布式协调管理功能;

3.

Solr.持更多格式的数据，.Elasticsearch仅.持json.件格式；

4.

Solr官.提供的功能更多，.Elasticsearch本.更注重于核.功能，.级功能多有第三.插件提供；

5.

Solr在传统的搜索应.中表现好于Elasticsearch，但在处理实时搜索应.时效率明显低于Elasticsearch。

6.

Solr是传统搜索应.的有.解决.案，但Elasticsearch更适.于新兴的实时搜索应.。

使.案例： 1.维基百科使.E lasticsearch来进.全.搜做并.亮显.关键词，以及提供search-as-you-type、did-you-mean等搜索建议功能。

2.英国卫报使.E lasticsearch来处理访客.志，以便能将公众对不同.章的反应实时地反馈给各位编辑。

3.

StackOverflow将全.搜索与地理位置和相关信息进.结合，以提供more-like-this相关问题的展现。

4.

GitHub使.Elasticsearch来检索超过1300亿.代码。

5.每天，Go ldmanSachs使.它来处理5TB数据的索引，还有很多投.使.它来分析股票市场的变动。

2、相关.试题 solr如何实现搜索的？倒排索引，先抽取.档中词，并建.词与.档id的映射关系，然后查询的时候会根据词去查询.档id，并查询出.档 Solr过滤器 Solr的过滤器对接收到的标记流（TokenStream）做额外的处理过滤查询，在查询时设置 Solr原理 Solr是基于Lucene开发的全.检索服务器，.Lucene就是.套实现了全.检索的api，其本质就是.个全.检索的过程。全.检 索就是把原始.档根据.定的规则拆分成若.个关键词，然后根据关键词创建索引，当查询时先查询索引找到对应的关键词，并根据关键词找到对应的.档，也就是查询结果，最终把查询结果展.给..的过程 Solr基于什么 基于lucene搜索库的.个搜索引擎框架，lucene是.个开放源码的全.检索引擎.具包 solr怎么设置搜索结果排名靠前 设置.档中域的boost值，值越.相关性越.，排名就靠前 IK分词器原理 本质上是词典分词，在内存中初始化.个词典，然后在分词过程中逐个读取字符，和字典中的字符相匹配，把.档中的所有词语拆分出来的过程 solr的索引查询为什么.数据库要快 Solr使.的是LuceneAPI实现的全.检索。全.检索本质上是查询的索引。.数据库中并不是所有的字段都建.的索引，更何况如果使.like查询时很.的可能是不使.索引，所以使.solr查询时要.查数据库快 solr索引库个别数据索引丢失怎么办 .先Solr是不会丢失个别数据的。如果索引库中缺少数据，那就向索引库中添加 Lucene索引优化 直接使.Lucene实现全.检索已

经是过时的.案，推荐使.solr。Solr已经提供了完整的全.检索解决.案 多张表的数据导.solr(解决id冲突) 在schema.xml中添加uuid，然后solrconfig那边修改update的部分，改为使.uuid.成 solr如何分词，新增词和禁.词如何解决 schema.xml.件中配置.个IK分词器，然后域指定分词器为IK 新增词添加到词典配置.件中ext.dic，禁.词添加到禁.词典配置.件中stopword.dic，然后在schema.xml.件中配置禁.词典： solr多条件组合查询 创建多个查询对象，指定他们的组合关系，Occur.MUST（必须满.and），Occur.SHOULD（应该满.or），Occur.MUST_NOT（必须不满.not） elasticsearch了解多少，说说你们公司es的集群架构，索引数据..，分.有多少，以及.些调优.段。elasticsearch的倒排索引是什么。ElasticSearch（简称ES）是.个分布式、Restful的搜索及分析服务器，设计.于分布式计算；能够达到实时搜索，稳定，可靠，快速。和ApacheSolr.样，它也是基于Lucence的索引服务器，.ElasticSearch对.Solr的优点在于：轻量级：安装启动.便，下载.件之后.条命令就可以启动。 Schemafree：可以向服务器提交任意结构的JSON对象，Solr中使.schema.xml指定了索引结构。多索引.件.持：使.不同的index参数就能创建另.个索引.件，Solr中需要另.配置。 分布式：SolrCloud的配置.较复杂。倒排索引是实现"单词-.档矩阵"的.种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的.档列表。倒排索引主要由两个部分组成："单词词典"和"倒排.件"。 elasticsearch索引数据多了怎么办，如何调优，部署。 使.bulkAPI初次索引的时候，把replica设置为0增.threadpool.index.queue_size增.indices.memory.indexbuffersize增.index.translog.flushthresholdops增.index.translog.sync_interval增.index.engine.robin.refresh_interval http://www.jianshu.com/p/5eeeeb4375d4 lucence内部结构是什么 索引(Index)：在Lucene中.个索引是放在.个.件夹中的。如上图，同..件夹中的所有的.件构成.个Lucene索引。段(Segment)：.个索引可以包含多个段，段与段之间是独.的，添加新.档可以.成新的段，不同的段可以合并。segments.gen和segments_X是段的元数据.件，也即它们保存了段的属性信息。.档(Document)：.档是我们建索引的基本单位，不同的.档是保存在不同的段中的，.个段可以包含多篇.档。新添加的.档是单独保存在.个新.成的段中，随着段的合并，不同的.档合并到同.个段中。 域(Field)： .篇.档包含不同类型的信息，可以分开索引，.如标题，时间，正.，作者等，都可以保存在不同的域.。不同域的索引.式可以不同，在真正解析域的存储的时候，我们会详细解读。 词(Term)： 词是索引的最.单位，是经过词法分析和语.处理后的字符串。 solr和lucene的区别 Solr和Lucene的本质区别有以下三点：搜索服务器，企业级和管理。Lucene本质上是搜索库，不是独.的应.程序，.Solr是。Lucene专注于搜索底层的建设，.Solr专注于企业应..。Lucene不负责.撑搜索服务所必须的管理，.Solr负责。所以说，.句话概括Solr:Solr是Lucene.向企业搜索应.的扩展 Lucene:是.个索引与搜索类库，.不是完整的程序。 Solr：是.个.性能，采.Java5开发，基于Lucene的.个独.的企业级搜索应.服务器，它对外提供类似于Web-service的 API接.。 solr实现全.检索 索引流程：客.端---》solr服务器(发送post请求,xml.档包含filed，solr实现对索引的维护)搜索流程：客.端---》solr服务器(发送get请求，服务器返回.个xml.档) solr和lucene之间的区别 lucene全.检索的.具包,jar包 solr全.检索服务器,单独运.的servlet容器 作者：Ms_lang 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1. 3分钟带你彻底搞懂 Java泛型背后的秘密

2.

聊.聊前后端分离项.中权限数据库的设计

3.

来，.撸.个简版 Redis（附源码）

4. MySQL . limit为什么会影响性能？ 5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快！

.试题：分布式系统接.，如何避免表单的重复提交？ 季.林Java后端 2019-11-21 点击上.Java后端，选择设为星标优质.章，及时送达

作者|季.林链接|opengps.cn/Blog/View.aspx?id=426 上篇|.千.MySQL学习笔记 关于怎么实现承载更多..量的系统，.直是我重点关注的.个技术.向。改造架构提.承载.，通常来讲分为两个..向，互 相配合实现。硬件架构改进，主要是使.阿.云这种多组件的云环境：通过负载均衡SLB，模版克隆的云服务器ECS，云数据库RDS，共享对象存储OSS等不同职责的云产品组合实现。 软件架构优化，主要是软件代码开发的规范：业务解耦合，架构微服务，单机.状态化，.件存储共享等在分布式系统的学习途中也不断.识新的知识点，今天要说的就是软件开发时候对于接.服务的“幂等性”实现！ 幂等性效果：系统对某接.的多次请求，都应该返回同样的结果！（.络访问失败的场景除外）.的：避免因为各种原因，重复请

求导致的业务重复处理 重复请求场景案例： 1，客.端第.次请求后，.络异常导致收到请求执.逻辑但是没有返回给客.端，客.端的重新发起请求2，客.端迅速点击按钮提交，导致同.逻辑被多次发送到服务器简单来划分，业务逻辑..都可以归纳为增删改查！对于查询，内部不包含其他操作，属于只读性质的那种业务必然符合幂等性要求的。对于删除，重复做删除请求.少不会造成数据杂乱，不过也有些场景更希望重复点击提.的是删除成功，.不是.标不存在的提 .。对于新增和修改，这.是今天要重点关注的部分：新增，需要避免重复插.；修改，避免进..效的重复修改； 幂等性的实现.式 实现.法：客.端做某.请求的时候带上识别参数标识，服务端对此标识进.识别，重复请求则重复返回第.次的结果即可。 举个栗.：.如添加请求的表单.，在打开添加表单..的时候，就.成.个AddId标识，这个AddId跟着表单.起提交到后台接..后台接.根据这个AddId，服务端就可以进.缓存标记并进.过滤，缓存值可以是AddId作为缓存key，返回内容作为缓存Value，这样即使添加按钮被多次点下也可以识别出来。这个AddId什么时候更新呢？只有在保存成功并且清空表单之后，才变更这个AddId标识，从.实现新数据的表单提交 【END】 如果看到这.，说明你喜欢这篇.章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下..维码即可进...告交流群。 ↓扫描.维码进群↓

推荐阅读 1..千. MySQL学习笔记

2.

.份.作坚持多久跳槽最合适?

3.项.中常.到的 19条 MySQL优化

4.零基础认识 Spring Boot

5.团队开发中 Git最佳实践

喜欢.章，点个在看 声明：pdf仅供学习使.，.切版权归原创公众号所有；建议持续关注原创公众号获取最新.章，学习愉快!

.试题：说说你对BigDecimal的理解？ Java后端 2019-11-19 点击上.Java后端，选择设为星标 优质.章，及时送达

链接|HikariCP来源|www.jianshu.com/p/c81edc59546c #前. 我们都知道浮点型变量在进.计算的时候会出现丢失精度的问题。如下.段代码： System.out.println(0.05+0.01); System.out.println(1.0-0.42); System.out.println(4.015*100); System.out.println(123.3/100); 输出： 0.060000000000000005 0.5800000000000001 401.49999999999994 1.2329999999999999 可以看到在Java中进.浮点数运算的时候，会出现丢失精度的问题。那么我们如果在进.商品价格计算的时候，就会出现问题。很有可能造成我们.中有0.06元，却.法购买.个0.05元和.个0.01元的商品。 Tips：可以微信搜索：Java后端，关注后加.咱们..的交流群。 因为如上所.，他们两个的总和为0.060000000000000005。这.疑是.个很严重的问题，尤其是当电商.站的并发量上去的时候，出现的问题将是巨.的。可能会导致.法下单，或者对账出现问题。所以接下来我们就可以使.Java中的 BigDecimal类来解决这类问题。 普及.下： Java中float的精度为6-7位有效数字。double的精度为15-16位。 #API 构造器： 构造器描述 BigDecimal(int)创建.个具有参数所指定整数值的对象。 BigDecimal(double)创建.个具有参数所指定双精度值的对象。 BigDecimal(long)创建.个具有参数所指定.整数值的对象。 BigDecimal(String)创建.个具有参数所指定以字符串表.的数值的对象。 函数： .法描述 add(BigDecimal)BigDecimal对象中的值相加，然后返回这个对象。 subtract(BigDecimal)BigDecimal对象中的值相减，然后返回这个对象。 multiply(BigDecimal)BigDecimal对象中的值相乘，然后返回这个对象。 divide(BigDecimal)BigDecimal对象中的值相除，然后返回这个对象。 toString()将BigDecimal对象的数值转换成字符串。 doubleValue()将BigDecimal对象中的值以双精度数返回。 floatValue()将BigDecimal对象中的值以单精度数返回。 longValue()将BigDecimal对象中的值以.整数返回。 intValue()将BigDecimal对象中的值以整数返回。 由于.般的数值类型，例如double不能准确的表.16位以上的数字。 #BigDecimal精度也丢失 我们在使.BigDecimal时，使.它的BigDecimal(String)构造器创建对象才有意义。其他的如BigDecimalb=new BigDecimal(1)这种，还是会发.精度丢失的问题。如下代码：
BigDecimala=newBigDecimal(1.01);BigDecimalb=newBigDecimal(1.02);BigDecimalc=newBigDecimal("1.01");BigDecimald=newBigDecimal("1.02");System.out.println(a.add(b));System.out.println(c.add(d));输出：
2.030000000000000026645352591003756970167160034179687 5 2.03 可.论丢失精度BigDecimal显的更为过分。但是使.Bigdecimal的BigDecimal(String)构造器的变量在进.运算的时候却没有出现这种问题。究其原因计算机组成原理..都有，它们的编码决定了这样的结果。long可以准确存储19位数字，. double只能准备存储16位数字。 double由于有exp位，可以存16位以上的数字，但是需要以低位的不精确作为代价。如果需要.于19位数字的精确存储，则必须.BigInteger来保存，当然会牺牲.些性能。所以我们.般使.BigDecimal来解决商业运算上丢失精度的问题的时候，声明

BigDecimal对象的时候.定要使.它构造参数为String的类型的构造器。 同时这个原则EffectiveJava和MySQL必知必会中也都有提及。float和double只能.来做科学计算和.程计算。商业运算中我们要使.BigDecimal。 .且我们从源码的注释中官.也给出了说明，如下是BigDecimal类的double类型参数的构造器上的.部分注释说明：
*Theresultsofthisconstructorcanbesomewhatunpredictable.
*Onemightassumethatwriting{@codenewBigDecimal(0.1)}in *Javacreatesa{@codeBigDecimal}whichisexactlyequalto
*0.1(anunscaledvalueof1,withascaleof1),butitis *actuallyequalto
*0.1000000000000000055511151231257827021181583404541015625.
*Thisisbecause0.1cannotberepresentedexactlyasa *{@codedouble}(or,forthatmatter,asabinaryfractionof
*anyfinitelength).Thus,thevaluethatisbeingpassed
intotheconstructorisnotexactlyequalto0.1,*appearancesnotwithstanding. ……
*Whena{@codedouble}mustbeusedasasourcefora *{@codeBigDecimal},notethatthisconstructorprovidesan
*exactconversion;itdoesnotgivethesameresultas *convertingthe{@codedouble}toa{@codeString}usingthe *
{@linkDouble#toString(double)}methodandthenusingthe
{@link#BigDecimal(String)}constructor.Togetthatresult,**usethe{@codestatic}{@link#valueOf(double)}method.**
publicBigDecimal(doubleval){this(val,MathContext.UNLIMITED);} 第.段也说的很清楚它只能计算的.限接近这个数，但是.法精确到这个数。第.段则说，如果要想准确计算这个值，那么需要把double类型的参数转化为String类型的。并且使.BigDecimal(String)这个构造.法进.构造。去获取结果。 #正确运.BigDecimal 另外，BigDecimal所创建的是对象，我们不能使.传统的+、-、、/等算术运算符直接对其对象进.数学运算，.必须调.其相对应的.法。.法中的参数也必须是BigDecimal的对象，由刚才我们所罗列的API也可看出。 在.般开发过程中，我们数据库中存储的数据都是float和double类型的。在进.拿来拿去运算的时候还需要不断的转化，这样.分的不.便。这.我写了.个.具类： /
*@author:JiYongGuang. *@date:19:502017/12/14. */ publicclassBigDecimalUtil{ privateBigDecimalUtil(){}
publicstaticBigDecimaladd(doublev1,doublev2){//v1+v2
BigDecimalb1=newBigDecimal(Double.toString(v1));BigDecimalb2=newBigDecimal(Double.toString(v2));
returnb1.add(b2);} publicstaticBigDecimalsub(doublev1,doublev2)
{BigDecimalb1=newBigDecimal(Double.toString(v1));BigDecimalb2=newBigDecimal(Double.toString(v2));
returnb1.subtract(b2);} publicstaticBigDecimalmul(doublev1,doublev2)
{BigDecimalb1=newBigDecimal(Double.toString(v1));BigDecimalb2=newBigDecimal(Double.toString(v2));
returnb1.multiply(b2);} publicstaticBigDecimaldiv(doublev1,doublev2)
{BigDecimalb1=newBigDecimal(Double.toString(v1));BigDecimalb2=newBigDecimal(Double.toString(v2)); //2=保留.数点后两位ROUND_HALF_UP=四舍五. returnb1.divide(b2,2,BigDecimal.ROUND_HALF_UP);//应对除不尽的情况 } } 该.具类提供了double类型的基本的加减乘除运算。直接调.即可