|  |
| --- |
| 代理模式就是给一个对象提供一个代理，由代理对象来控制原对象的引用，然后代理模式是能够协调调用者和被调用者，在一定程度上降低了系统的耦合性，在生活中买机票的例子，中介（中间商）。项目中利用代理模式都干些什么。 框架自带的。 |

|  |
| --- |
| 在spring的设计中，IOC和DI虽然是两个不同的名字 ，但实际上从两种不同角度去 阐述一个事情，IOC 的实现实际上是spring利用他的接口beanfactory，application context 他是加载我们的配置文件，去创建我们的Bean，然后保存到我们的容器中，就是动解析，反射，工厂的方式去是实现的，aop的实现就是动态代理，两种常见的动态代理就jdk动态代理，和cglib动态代理， |

|  |
| --- |
| 线程sleep和wait的区别   1. 这两个方法来自不同的类分别是Thread和Object 2. sleep方法没有释放锁，wait方法释放了锁，使得其他线程可以使用同步控制块或者方法 3. sleep必须捕获异常，而wait，notify，notifyAll 不需要捕获异常 4. wait ，notify（通知，通告） ，notifyAll 只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用 |

|  |
| --- |
| Int 和 integer 有什么区别  Int 它是一个原始类型，integer是一个包装类型，int是当时jvm开发的时候，为了性能和开销的考虑，引入了int，不是完全面向对象处理的，interger是完全面向对象处理的 ，jvm里面有对象头，也会加载它的Class对象，实际使用的时候的，int有一个默认值，integer默认值就是null，int比较的时候调用的是==，integer时调用的equal。interger是有一个缓存存在的，-127~+128之间，对调用==，临时的变量用int好一点，他是分布在栈上的。定义pojo 的时候，最好用interger，在序列化和反序列化的时候，可以告诉对方有没有这个值，给数据库的对应关系也会好一点，数据库int默认是null。 |

|  |
| --- |
| 谈谈jvm分为那些方法区，每个区都是干嘛的？  方法区，堆，栈，程序计数器。堆就是jvm堆，它可以使用对外的内存，主要是把大量的内存分配在堆上面，栈的话就是线程栈，jvm运行的时候，每个线程都会保有一个线程栈，线程会有一个程序计数器，jvm他是一个虚拟机，所以它还会有一个本地方法栈，她会去调用一些C库，所以栈分为本地方法栈和jvm方法栈，方法区，jvm会把一些类，静态的全局变量，引用也会存在方法区， |

|  |
| --- |
| Varchar：可变长度字符串，会根据实际长度分配空间，节省空间 但是速度慢  Char ：定长字符串，使用不恰当会浪费空间， 速度比varchar快。 |

|  |
| --- |
| 存储引擎就是一个表存储/组织数据的方式，  MYISAM存储引擎，管理的表，使用三个文件来表示每个表。  格式文件-存储表结构的定义（.frm文件）  数据文件-存储表的内容（.MYD文件）  索引文件-存储表上的索引（.MYI文件）  不支持事务，安全性能比较低  ！！！对于一个表来说，只要有主键，或者加有unique约束的字段上会自动创建索引  这种存储引擎的优势 可被转换为压缩，只读表来节省空间（优点！！！！！） |

|  |
| --- |
| InnoDB存储引擎 MySQL默认的存储引擎，重量级的存储引擎  InnoDB支持事务，支持数据库崩溃后自动回滚机制  InnoDB存储引擎最主要的特点是，非常安全。  特征：   1. 每个InnoDB表在数据库目录中以.frm格式文件表示 2. InnoDB表空间tablespace被用于存储表的内容（表空间是也给逻辑名称，表空间+索引） 3. 提供一组用来记录事务性活动的日志文件 4. 用COMMIT（提交）SAVEPOINT 及ROLLBACK（回滚）支持事务处理 5. 提供全ACID(四个特性)兼容 6. 在MYSQL服务器崩溃后提供自动恢复 7. 多版本（MVCC）和行级锁定 8. 支持外键及引用的完整性，包括级联删除和更新   InnoDB最大的特点就是支持事务：MYSQL很注重数据安全  以保证数据的安全，效率不是很高，并且也不能压缩，不能转换为只读，不能更好的节省空间 |

|  |
| --- |
| MEMORY存储引擎 断电即消失 以前被称为堆内存  使用MEMORY存储引擎的表，其数据存储在内存中，且行的长度固定，  这两个特点使得MEMORY存储引擎非常快  MEMORY存储引擎管理的表：   1. 在数据库目录内，每个表均已.frm文件表示。 2. 表数据及索引被存储在内存中。（目的就是快，查询快） 3. 表级锁机制 4. 不能包含TEXT或BLOB字段   MEMORY引擎的优点：查询效率是最高的。不需要和硬盘交互  缺点，不安全、关机即消失。因为数据和索引都是在内存中 |

|  |
| --- |
| 什么是事务？  一个事务就是一个完整的业务逻辑，是一个最小的工作单元，不可再分  说到底就是，两条及以上的DML语句同时成功，或者同时失败  事务的四个特性（ACID）  A:原子性  C:一致性  I:隔离性  D:持久性 |

|  |
| --- |
| 重点研究这个事务的隔离性  A教室和B教室中间有一道墙，这道墙可以很厚，也可以很薄，这就是事务的隔离级别，  这道墙越厚，表示隔离级别就越高  一共有四个隔离级别  读未提交：read uncommitted(最低的隔离级别)  事务A可以读取到事务B未提交的数据 。  这种隔离级别存在的问题就是 脏读（Dirty read）！  就是读到了脏数据  这种隔离级别一般都是理论上的，大多数的数据库隔离级别都是二档起步  读已提交：read committed  事务A只能读取到事务B提交之后的数据  解决了脏读现象 但是产生了不可重复读的问题  什么是不可重复读？  在事务开启之后第一次读到的数据是3条，当前事务还没有结束，可能第二次再读取的时候，读到的数据是4条，3不等于4 称为不可重复读  这种隔离级别是比较真实的数据，每一次读到的数据是绝对真实的数据，Oracle数据库默认的隔离级别是 读已提交  可重复读 repeatable read （就是读以提交的）  解决了不可重复读的问题，但是造成了幻读的问题，Mysql默认的隔离级别就是这个  序列化/串行化 serializable （最高的隔离级别）  效率最低，但是解决了全部问题。  这种隔离级别表示事务排队，不能并发， 和synchronized类似。数据都是真实的  幻读就是 第一次查到/是不一样的 （只是对于查看来说）  不可重复读 是对于增删改来说的 两次执行的select出来的结果不一样 |

|  |
| --- |
| 在MySQL数据库中索引也是需要排序的，并且这个的排序和Treeset数据结构相同，TreeSet底层是一个自平衡二叉树，在mysql中索引是一个B-Tree数据结构  采用左小右大，中序遍历取数据 |

|  |
| --- |
| 索引的原理就是缩小扫描范围，避免全表扫描  提醒1. 在任何数据库当中主键上都会自动添加索引对象，id字段上自动有索引，另外在mysql中，一个字段如果有unique约束的话，也会自动创建索引对象  提醒 2 . 在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有一个硬盘的物理存储编号  提醒3. 在mysql中索引是一个单独的对象，不同的存储引擎以不同的形式存在，在myisam存储引擎中，索引存储在一个 .MYI文件中，在InnoDB存储引擎中，索引存储在一个逻辑名称叫做table space的中，在MEMORY存储引擎当中索引被存储在内存中，不管索引存储在哪里，索引在mysql当中都是一个树的形式存在（自平衡二叉树，B-Tree） |

|  |
| --- |
| 在mysql中主键上，以及unique字段上都会自动添加索引的  什么条件下，我们会考虑到加索引呢？  条件1. 数据量庞大，（到底有多大，需要测试，因为每个硬件环境不同）  条件2. 该字段经常出现在where后面，以条件的形式存在，也就是说这个字段总是被扫描  条件3. 该字段很少的DML（增删改 ）操作（因为DML之后，索引需要重新排序）  建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。  建议通过主键查询，建议通过unique约束的字段进行查询，效率是比较高的 |

|  |
| --- |
| Select \* from emp where ename like ’% T’  这时候索引就会失效， 原因是因为模糊匹配当中以“%”开头了  尽量避免模糊匹配当中以%开头  这是一种优化的手段/策略 失效的第一种情况  使用or的时候会失效，如果使用or那么要求or两边的条件字段都要有索引，才会走索引，如果其中一边有一个字段没有索引，那么另一个字段上的索引也会失效，所以这就是为什么不建议使用or的原因 失效的第二种情况  使用复合索引的时候，没有使用左侧的列查找，索引失效  失效的第三种情况  在where中索引列参加了运算，索引失效  失效的第四种情况  在where当中索引列使用了函数  失效的第五种情况 |

|  |
| --- |
| 索引是各种数据库进行优化的重要手段，优化的时候优先考虑的因素就是索引，  单一索引 一个字段上添加索引，  符合索引 两个字段或者更多字段上添加索引  主键索引 主键上添加索引  唯一性索引 具有unique 约束的字段上添加索引 |

|  |
| --- |
| SpringBoot 自动装配原理  SpringbootApplication 注解主要包含三个重要的注解，@springBootConfiguration @EnableAutoConfiguration @ComponentScan  @ComponentScan 是扫描那些包 到我们的容器中  @SpringBootConfiguration 是加载我们相关的 配置  @EnableAutoConfiguration 这个里面也有很多注解，其中@Import（AutoConfigurationImportSelector.class）选择器  AutoConfigurationImportSelector 实际是实现ImportSelector接口的类  其中ImportSelector 中有一个比较重要的类 selectImports 从字面意思，它选取一些类，注入进来，本质上是把Bean注入的一个操作  紧接着问application.properties 里面的配置是怎么加进去的呢  SpringBoot 启动后，会去加载spring factorys里面的类  里面有一个ServletWebServerFactoryAutoConfiguration ,这个就是springBoot加载servlet的配置类，里面有一个EnableConfigurationProperties会把后面括号里那个类加载进来，然后里面又有一个@ConfigurationProperties 这个就是把配置文件中的keyVaule 键值对读入进来  老金版本：只要是spring Boot启动的项目，先从starter启动器开始，都会运行SpringFactoriesLoader，他会读MEIT-INF/spring.factories 文件，然后挨个加载（只要有就会加载）  总结一下：spring Boot自动装配流程  Spring在启动的时候，他是去寻找myData下的springfactory文件，而这个文件里面放的是key-value 的键值对，它包含了spring启动所包含的所有类的信息，就是这些bean通过IOC容器给实例化，加到容器中，进行存储，配置文件中的信息怎么加入到容器中？ 是通过@ConfigurationProperties这个注解把配置文件中的属性和类中的字段绑定在一起，这样实现一个人为修改的一个目的 |

|  |
| --- |
|  |

|  |
| --- |
| Java 的集合容器都包含那些吗？  主要是三个接口 list set map，以及实现了这么多接口的诸多子类，只不过我们在谈到jdk容器的时候，经常会把这三者放在一起谈。  最基本的概念上的区别  List他是有序的，这个有序指的是插入元素的顺序，其中可以存放重复的值和null值，  Set他是无序的，其中的值都是各不相同的，所以里面只能存放一个null值，  Map它存放的是entry键值对，键是唯一的，所以里面只能存放一个键为null值，但是可以存放多个值为null的entry。  关于HashMap你能说说他的理解嘛？  HashMap正如她的名字一样，它的key是通过hashcode进行存储的，这就要谈到它的内部结构了，HashMap的内部结构主要用到了三个数据结构，数组++链表，以及在jdk1.8之后为了提升它的查询速度，加入的红黑树结构。  你可以具体的讲一下吗？  首先HashMap中维护了一个数组，数组类型为entry，entry就是键值对，每个entry要添加到map中的时候，首先要使用hash算法，来计算它的hashcode，然后对hashcode进行取模操作，然后根据值再添加到数组中的位置，所以说hash算法是比较重要的，好的算法出来是比较散列，均匀的，这样认为它的算法是比较好， 而且查找时间复杂度是O(1), 如果它的算法比较差的话，他们的取模结果都是一样的，整个数组就会蜕化成一个链表，这样它的查询速度就是非常慢，时间复杂度就是 O(n), jdk8中对此做了一些优化，一个节点上的节点超过8的时候，就将这个链表转化成红黑树，而红黑树的查询速度是O（log n）这样就起到了一定的优化左右。  那么为什么这个临界值是8呢？  这个临界值为什么是8，这是和泊松分布有关的，这是开发者在空间和时间上做了一些取舍  看过它的源码嘛?说说他的实现方式吧，  Hashmap的数据结构是数组+链表，其中数组存储的元素叫Node， Node是对entry的实现，entry是一个接口，node是他的实现类，node主要有四个属性，hash，key，value，next，next是指向下一个节点的区域，组成一个链，还重写了hashcode和equals方法，  接下来看hashmap的另外属性，除了table是也给Node类型的数组，还有entrySet,size, modCount, threshold ,loadFactor entrySet是map.entry类型的set集合，size集合中被使用的实际数量，modCount是一个计数器，主要用来hashmap结构发生变化的次数  比如put一个值，或者扩容一次，threshold等于size\*loadFactor，loadFactor是负载因子， 默认值是0.75  transient这个修饰符不是很常见，你能说一下吗？  Java中有一个serializable（序列化）的接口，一个类实现了这个接口的话就会自动序列化，那什么是自动序列化呢，就是java进程之间进行对象传输的时候，或者说对对象进行持久化操作的时候呢，就要将对象序列化为二进制流，便于传输或者存储 ，那transient修饰一个属性后，那么这个属性将不会被自动序列化，比如说一些敏感的字段，开发者是不希望他被序列化的，比如一个含有敏感字符有这个transient修饰符的类，把他序列化，再反序列化将看不到这个transient修饰的属性  hashMap的扩容机制  hashMap默认长度是16，默认的负载因子是0.75，负载因子的意思就是当数组使用度到了75%以上的时候，threshold就是 size\*loadFactor，扩容都是成倍的，都是之前的两倍，扩容是一个比较耗资源的操作，他的需要给数组分配给多的空间，好处就是数组更长了，他的hash算法就会比较均匀，碰撞也会更少，个数和碰撞几率它是服从泊松分布的，发现在0.75处碰撞几率是最小的，所以使用0.75作为默认的负载因子，当然面临特殊情况的时候呢，我们可以通过构造器手动设置它的大小，但是不建议去改动默认值，  揣测开发者为什么负载因子是0.75，扩容又是2倍扩容呢？  至于为什么table的长度是2的次幂，是为了在取模的时候做优化，当判断node值进入按个桶的时候，要对hash值进行取模，但是取模是一个比较重的操作，比较耗性能，而当数组的长度总是2的N次方的时候呢，用hash值和长 度-1 来进行 位与 运算的时候呢，是等于对长度直接取模的 ，相较于取模的操作，位与运算更好。 效率更高  为什么hashmap要在Object的hashcode的基础上进行异或或右移的操作呢？  Hash与它的右移16位进行与运算是为了减少碰撞。  你能说一个put的整个流程吗？  进来判断table是否为空或者length是否等于0 来判断是否扩容，其他都是加入节点的时候扩容  介绍一下hashmap的线程安全情况吗？  hashMap不是线程安全的，因为它的很多操作都不是用synchronized来修饰的，扩容就不是线程安全的，会出现环形链表异常  hashtable就是给hashmap加锁版本，  1.7concurrentHashMap维护了一个分段锁，就是在厕所的坑位上分别加锁，而不是一下把厕所全锁住，里面维护了segment数组，这个长度可以自己设置， |

|  |
| --- |
| Elasticsearch底层写流程 包括查询性能高的原因  首先客户端发一个请求，要写数据（增删改），数据是存在索引index中的，索引分成若干个分片（shard）,不管是主分片还是副分片，都是存在硬盘上的，落实成文件，这是必然流程，一定是这样的，但是要做写操作的时候，往硬盘里面写数据，是I/O操作，重量级操作，很慢，不符合ES的近实时（NRT,Near RunTime）特性,无论是写还是查，1秒内返回，要是使用I/O流，未必能达到1秒返回。  默认 ES会在内存里面开一个缓存，缓存的大小无限制，但里面只存1秒的数据，记录完一秒的数据，下一秒重新开一个缓存来记录下一秒的数据， 这一秒记录好的数据，会放在内存的另外一个地方-分段（segment），类似java里面的File，一秒的数据一个分段，记录一段时间后，内存里面的分段数量明显变多，然后ES会请求操作系统协作，每隔一段时间会把分段交给系统缓存（OS cache）管理工具，每个操作系统都有缓存管理工具，不管是windows 还是linux，缓存管理基于虚拟内存（拿硬盘做缓存）和内存缓存两个一起用。  ES就会把分段交给操作系统，让它协作管理，在操作系统的系统缓存区里面放了很多分段（segment），时间不断变长后，缓存里面的分段数量也会变多，操作系统拥有全资源控制能力，但是资源是有限的，不可能无限使用。然后它也会把数据往硬盘里面刷，刷到硬盘里面是一个个的分段， 就会变成文件，但是文件的数量是很多的。  这么多分段文件，ES这么处理的目的就是，查询。每秒一个缓存，缓存里的数据会被立刻查到，上一秒的数据在分段里面，分段里面的文件也是被开启状态（OPEN）也可以做搜索查询，在操作系统缓存里面放置的分段文件也是开启的，也可以做查询操作，在硬盘里面的文件同样是这样，也可以做搜索查询。它又是多线程操作，硬盘上的文件由长期I/O去管理。这就是ES查询快的原因。  但是在整个流程里面会有问题，  不管是缓存，还是分段（segment），系统缓存（OS cache），一旦电脑断电，硬盘还没有存数据，就会丢数据。ES为了这个情况发生，它会在写操作的同时，也会记录translog日志文件（记录命令），日志永久在硬盘上，而且是长I/O，长期定位。万一断电发生，在重启的时候会全看日志，把日志的命令还原到内存里面。  硬盘的文件数量会随着时间的变长而变多，所以ES会在每隔一段时间或者空闲的时候，会把系统硬盘里面大小相近的分段文件读到内存里，然后合并，合并成一个更庞大的segment文件，在合并过程中，把所有已标记删除的数据清除，把文件压缩，压缩完后，还是交个系统缓存（OS cache），所以硬盘上的文件是逐渐变多，再经由合并，再主键变少，如果长期没有写操作，最终会变成一个分片（一个文件），不是一个索引一个文件，一个分片若干个分段。  日志文件  随着时间的增长，日志文件的大小也会吃尽硬盘的所有容量，ES会在日志处做一个周期化的提交（commit point）比如说合并文件的周期30分钟，那么translog的提交周期就是30分钟，只要把数据写到硬盘里面，说明没有问题了，ES就会把老的命令数据清除，这样最极限的情况就是存60分钟的命令，不至于发生把硬盘吃尽的情况发生，  不考虑文件是否排列紧密，先做标记删除和再新增，使用周期化处理，读取硬盘中大小相近的分段文件做合并整理，合并整理的时候清除标记删除的数据，做一下压缩化处理，按照整体流程不断循环往复，让自己的应用文件数量保持在一个合理范围，缓存中，硬盘中的文件都是可搜索状态（OPEN）让增删改查都是近实时（NRT,Near RunTime）处理，达到秒级处理 |

|  |
| --- |
| 中间件：tomcat就是中间件，运行在两者之间的东西，tomcat运行在web和jvm两者之间。  RabbitMQ就是基于AMQP协议去设计的一款软件，在RabbitMQ中一定有交换器和队列概念，中间的那部分就是一个borker，一个borker中可以运行多个v-host（虚拟主机），默认的RabbitMQ启动后里面就有一个v-host(/[根])，一个v-host中可以有多个交换器和队列组成（多个交换器多个队列），交换器和队列之间通过路由键进行绑定，路由键可选范围字符串+‘\*’+‘#‘，通过‘.’进行分割，\*号代表一个不包含‘.‘的完整的单词，‘#’代表所有，一切，路由键，队列和交换器之间的对应关系，在进行处理映射的时候，称为绑定（Bingding），通过路由键（Routing key）绑定。一个交换器可以绑定多个队列，一个队列可以绑定多个交换器，它只看路由键是否正确，来决定本次交换器收到的消息放到那个队列里面。  客户端发消息的称为publisher，消息信息包括头和体，头包括路由键，他的优先级，是否永久存储，就是是否持久化，等等属性，而路由键是一定存在的，只有一种情况下不存在—广播。体是由二进制数组描述的，理论上什么都可以存，实际存的是一个二进制数组，字节流数据。publisher发消息必须发送到指定的交换器上，前提必须知道把消息放到那个交换器上。放到交换器后，交换器会找消息的头，找到路由键，绑定到队列，并把消息投递到里面去。  Consumer消费者可以绑定若干个队列，绑定队列代表消费这个队列中的消息，而队列可以被多个消费者同时绑定，多个队列绑定同一个消费者的时候，轮询消费消息，忽略消费者的性能。一个消费者可以绑定若干个队列，但是就建立一个连接（connect），这个连接有多个信道（channel），一个连接可以有多个信道，这就是软连接（虚连接） 每个信道对应一个队列。通过连接（connection）连接到服务器，用TCP连接，AMQP协议，信道（channel）连接到队列。用信道不用连接的原因就是资源，资源消耗少，耗时短，性能高。 |

|  |
| --- |
| Redis的分布式锁的实现思想    （每步缺一不可）   1. 获取锁的时候，使用setnx加锁，并使用expire命令为锁添加一个超时时间，超过该时间则自动释放锁，锁的value值为一个随机生成的UUID，通过此再释放锁的时候进行判断 2. 获取锁的时候还设置一个获取的超时时间，若超过这个时间则放弃获取锁 3. 释放锁的时候，通过UUID判断是不是该锁，若是该锁，则执行delete进行锁释放   （Redisson为了兼容新老版本，只要不是原子操作，就用lua脚本）  问题一：超时时间长短问题  解决：在加锁成功后，启动一个守护线程（Redission中的Watch dog），守护线程每隔1/3的锁的超时时间就去延迟锁的超时时间，当业务代码执行完成，关闭守护线程。  问题二：Redis主从切换导致锁失效问题  Redlock具有很大的争议。**Redisson并不能有效的解决Redis的主从切换问题的**，**目前推荐使用Zookeeper分布式锁来解决。** |

|  |
| --- |
| 并行和并发的区别：  并发是指一个处理器同时处理多个任务  并行是多个处理器或者多核处理器同时处理不同的任务  比喻：并发是一个人同时吃三个馒头，并行是三个人同时吃三个馒头  线程池的核心参数？  CorePoolSize：线程池的基本大小，即在没有任务需要执行的时候线程池的大小。  queueCapacity 任务队列容量，当核心线程达到最大时，新任务会放在队列中排队等待执行  maxPoolSize：最大线程数  keepAliveTime：线程空闲时间，当线程空闲时间达到keepAliveTime时，线程会退出，直到线程数量=corePoolSize。当allowCoreThreadTimeOut=true 则会直到线程数量=0  allowCoreThreadTimeOut 允许核心线程超时  rejectExecutionHandler 任务拒绝处理器  JDK自带的线程池的4中拒绝策略  ThreadPoolExecutor.AbortPolicy丢弃任务，并抛出RejectedExecutionException异常；  ThreadPoolExecutor.CallerRunPolicy:该任务被线程池拒绝，由调用execute方法的线程执行该任务。  ThreadPoolExecutor.DiscardOldestPolicy:抛弃队列最前的任务，然后重新尝试执行任务。  ThreadPoolExecutor.DiscardPolicy,丢弃任务，不过也不抛出异常。  线程池的执行流程    一个任务通过execute方法被添加到线程池，任务就一个runnable类型的对象，就是执行对象的run方法，  如果此时的线程数量小于corePoolSize，就会创建线程执行任务，  如果线程数量大于等于corePoolSize ，就会判断缓存队列是否已满，没满的话，进入缓存队列，等待调用。如果满了的话，接着判断是否大于最大线程数量，如果大于的话就会执行拒绝，不大于就会重新创建线程。  什么是阻塞队列？  阻塞队列是一个支持两个附加操作的队列。   1. 支持阻塞的插入方法：当队列满时，队列会阻塞插入元素的线程，直到队列不满 2. 支持阻塞是移除方法：当队列为空时，获取元素的线程会被阻塞，等到队列变为非空   种类：ArrayBlockingQueue:一个由数组结构组成的有界阻塞队列  LinkedBlockingQueue: 一个由链表结构组成的有界阻塞队列（7个）  原理就是通知模式：就是当生产者往满的队列添加元素时会阻塞住生产者，当消费者消费了一个队列中的元素后，会通知生产者当前队列可用。  核心线程会不会被回收？  ThreadPoolExecutor回收线程都是等getTask( )获取不到任务，返回null时，调用processWorkerExit方法从set集合中remove掉线程，getTask（）返回又分为2种场景   1. 线程正常执行完任务，并且已经等到超过keepAliveTime时间，大于核心线程数，那么会返回null，结束外层的runWorker种的while循环。 2. 当调用shutdown（）方法，会将线程池状态设置为shutdown，并且需要等待正在执行的任务执行完，阻塞队列中的任务执行完才能返回null。   什么是对象锁？  对象锁也叫方法锁，是针对一个对象实例的，它只是该对象的某个内存位置声明一个标识该对象是否拥有锁，所以它只会锁住当前的对象，而并不会对其他对象实例的锁产生任何影响，不同对象访问同一个 被synchronized修饰的方法 的时候不会阻塞。  Synchronized加在代码块和方法上有什么区别？  代码块上就是 先获取对象地址的锁，这样就可以先执行那部分代码。执行完就放开锁其他线程就可以进来了。  方法上就是 如果访问同一个类同一个实例的方法，就会被同步。  放在方法和类上没什么区别，只是可读性的问题。  Wait（）为什么必须在代码块中运行？  因为wait和notify和notifyAll（）来实现线程间通信。比如在消费者和生产者维妮塔之间必须实现线程同步。不在代码块中就会报illegalMonitorStateException（非法监视器声明状态）  怎么唤醒指定线程？  Jdk6之后 LockSupport  优势：LockSupport不需要在同步代码块里，所以线程间不需要维护一个共享的同步对象了，实现了线程间的解耦。 unpark函数可以先于park调用，所以不需要担心线程间的执行的先后顺序。 其实有很多地方都用到了这个，比如java里面最常用的类 ThreadPoolExector里面的get方法是如何塞住当前线程，和线程池执行完任务又是如何唤醒线程的呢？都是利用lockSupport。  还有就是利用condition+ReentrantLock 实现 condition就可以想象成一个条件队列，Condition必须咬配合锁一起使用，因为对共享状态变量的访问发生在多线程环境下。一个Condition的实例必须与一个Lock绑定，因此Condition一般都是作为Lock的内部实现。  什么是重入锁？  Java.util.concurrent.locks.ReentrantLock 这是JDK1.5添加的一种颗粒度更小的锁，它完全可以替代synchronized关键字来实现它的所有功能，而且ReentrantLock锁的灵活度要远远大于synchronized关键字。 synchronized也是重入锁。  怎么看待mysql日志先行？  所谓的日志先行就是mysql数据库的WAL机制，WAL全称是Write-Ahead Logging，它的关键点是先写日志，再写磁盘。WAL机制由来：用户如果对数据库中的数据进行了修改，必须保证日志先于数据落盘。当日志落盘后，就可以给用户返回操作成功，并不需要保证当时对数据的修改也落盘。如果数据库在日志落盘前crash（崩溃），那么相应的数据修改会回滚。当日志落盘后crash，会保证相应的修改不丢失。  Mysql中有那些类型的日志？  重做日志（redo log） 确保事务的持久性。redo日志记录事务执行后的状态，用来恢复未写入data file的已成功事务更新的数据。防止在发生故障的时间点，尚有脏页未写入磁盘，在重启mysql服务的时候，根据redo log进行重做，从而达到事务的持久性这一特性。  回滚日志：（undo log） 保证数据的原子性，保存了事务发生之前的数据的一个版本，可以用于回滚，同时可以提供多版本并发控制下的读（MVCC），也即非锁定读。  二进制日志：（binlog） 用于复制的，在主从复制中，从库利用主库上的binlog进行重播，实现主从同步。  主键索引和唯一索引的区别：   1. 主键是一种约束，唯一索引是一种索引，两者在本质上是不同的 2. 主键创建后一定包含一个唯一性索引，唯一性索引并不一定是主键 3. 唯一性索引允许空值，而主键列不允许为空值 4. 主键列在创建时，已经默认为空值+唯一索引了。 5. 主键可以被其他表引用为外键，而唯一索引不能。 6. 一个表最多只能创建一个主键，但可以创建多个唯一索引 7. 主键更适合都不容易更改的唯一标识，比如身份证号 8. 在RBO模式下，主键的执行计划优先级要高于唯一索引，两者都可以提高查询速度。   什么是回表？如何优化回表情况？避免二次查找？  回表就是先通过数据库索引扫描出数据所在行，再通过行主键id取出索引中未提供的数据。即基于非主键索引的查询需要多扫描一颗索引树。  因此可以通过索引先查询出id字段，再通过主键id字段，查询行中的字段数据，即通过再次查询提供MySQL查询速度。说白了就是不是主键索引查出来的数据不全，得通过id主键再次查找，查询的数据有重复（二次查找）。直接通过索引去查询id，用id直接查。  Spring注入Bean的几种方式？  通过注解@Configuration注入，也是最简单的一种。  通过构造方法。  通过set方法。  通过属性去注入Bean。  通过集合去注入Bean。  Set比构造器好，因为set用到的时候才会创建对象，构造器不是。  Spring怎么解决循环依赖问题？   1. 使用@Lazy注解，延迟加载 2. 使用@DependOn注解，指定加载先后关系 3. 修改文件名称，改变循环依赖类的加载顺序。   之前看过一篇文章，Spring内部是有三级缓存的来解决这个循环依赖问题，但是二级缓存也可以解决。只不过在满足Spring自身创建的原则下，是必须的。  Spring事务概念  Spring事务管理为我们提供了三个高层的抽象接口，分别是TransactioonProxyFactoryBean（事务管理器），TransactionDefinition（定义事务基本属性），TransactionStatus（事务状态）   1. 事务管理器----Spring框架并不直接管理事务，而是通过这个接口为不同的持久层框架提供了不同的接口实现类。 2. 事务的基本属性---隔离级别，传播行为，是否只读，事务超时，回滚规则（遇到什么异常就会回滚）。 3. 事务状态   Spring事务的实现方式  编程式事务 很少用 声明式事务—三种方式 代理方式，AspectJ的XML方式，基于注解的方式。  为什么private修饰的方法不能被事务回滚 也不报错  因为AOP底层是代理，jdk是代理接口，私有方法必然不会存在在接口里，所以不会被拦截到。cglib是子类，private的方法照样不会出现在子类里，也不能被拦截。 |

|  |
| --- |
| RPC就是两个进程之间的相互调用，不管在没在同一个电脑上。不能和TCP、Http比 不是一个东西。Http 应用层的通信协议，RPC是应用层的标准协议  我们写代码的时候，如果涉及到远程调用 用的是RPC协议， 如果搭一个网站，浏览器访问后端 用的是Http协议。 RPC基本都是代码层面的，都是框架实现  Dubbo框架    虚线：异步 实线：同步  Dubbon 必须依赖于容器才能使用，就是Spring，再具体点就是IOC容器  在编写的时候，有角色划分，就像eureka分为applicaionserver和applicationclient一样，它也一样有注册中心服务端和注册中心客户端。注册中心就是registry，这属于服务端，下面的都是客户端，两个客户端之间有调用了，被调用者称为服务的提供者（Provider），调用另外一个服务的叫消费者（Consumer）。写完代码启动的时候，第一时间进行容器初始化，初始化的时候start会把服务的提供者和消费者的对象创建出来，无论是提供者和消费者都会注册，把自己数据写里面。只有服务的消费者才会去发现，发现提供者，无论是发现还是注册都是异步的，发现的方法就会像本地方法一样去调用，调用访问远程，访问过程是一个同步操作。  如果服务的提供者发生了变化，再去注册一下，注册中心可以反向异步通知消费者，让消费者去重新发现，也可以等待消费者以心跳的方式再次发现新的提供者信息。  消费者和提供者都可以把自己的统计信息以异步的方式发送给monitor（监控器），统计了服务的提供者是谁，被调用了多少次，远程的消费者是谁，调用了那个提供者多少次，它只能做统计，别的不能干。  这里面就会注意到注册中心用什么？ 远程调用的时候，拿什么协议去调用？  协议用的是RPC，通信的时候默认用的dubbon协议，（封装的是tcp/ip），单一可复用的长链接。NIO（同步非阻塞） |

|  |
| --- |
| 同步异步是通信服务器之间的说法，比如浏览器和服务器之间 可以说同步异步，  阻塞 线程中sleep和System.in都是阻塞，  非阻塞就是让tomcat所在的操作系统帮忙，进来就会转发给里面的worker线程 ，非阻塞的工作能力更强，可以同时应对多个请求， |