目录

[1.动态代理 1](#_Toc162258249)

[Jdk动态代理 1](#_Toc162258250)

[Cglib动态代理 2](#_Toc162258251)

[Jdk动态代理为什么只能代理有接口的类 2](#_Toc162258252)

[区别 2](#_Toc162258253)

[2.订单表每天新增500W数据，分库分表的方案应该如何设计？在实际业务中存在两个比较多的查询。 3](#_Toc162258254)

[考点： 3](#_Toc162258255)

[3.生产中遇到CPU运行过高 4](#_Toc162258256)

[4. 在亿级别的数据库中，快速把uid=4的数据查出来，其中uid=4的数据查出来，而且这个uid=4的数据还很多。 4](#_Toc162258257)

[考点 4](#_Toc162258258)

[解决方案 5](#_Toc162258259)

[5.BigDecimal为何能解决浮点数运算时精度丢失问题 5](#_Toc162258260)

[6.go语言 5](#_Toc162258261)

[7. Springboot如何解决跨域问题？ 6](#_Toc162258262)

[8.如何在Java中去实现无锁数据结 6](#_Toc162258263)

# 1.动态代理

## Jdk动态代理

Java基于接口的动态代理是一种在运行时动态生成代理类的技术。它是通过反射机制在运行时生成代理对象，对被代理对象的方法进行拦截处理，实现增强功能。

Java基于接口的动态代理需要实现两个接口：InvocationHandler和Proxy。InvocationHandler接口中定义了一个invoke方法，该方法接收一个代理对象、被代理对象的方法和参数，并返回代理对象对被代理对象的方法进行增强后的返回值。Proxy类中则提供了一个静态方法newProxyInstance，用于创建代理对象

用途:记录方法调用时长、打印日志、权限校验

## Cglib动态代理

Java CGLIB 动态代理是一种在运行时生成代理对象的技术，它可以在不修改原始类型（类）的基础上，为该类型创建一个代理子类，该代理子类可以拦截原始类中的方法调用。相较于Java JDK动态代理，它可以代理没有实现接口的类。

## Jdk动态代理为什么只能代理有接口的类

Jdk动态代理本身机制决定的。

Jdk中的代理类是通过Proxy.newProxyInstance方法实现的，该方法需要传入动态代理接口类。之所以要传入接口而不能传入类，取决于底层实现。Jdk会在运行时，会生成一个代理类叫$proxy0，这个动态生成的代理类回去继承一个java.lang.reflect.Proxy类，同时还会实现被代理类的接口，java里面不支持多继承，而每个动态代理类都集成了一个Proxy，所以就导致jdk里面的动态代理只能代理接口，而不能代理实现类。Proxy只是保存了动态代理的一个处理器叫InvocationHandler，如果不抽出来，直接设置到$Proxy0这个动态代理里面，我认为也是可以得，如果这么去实现的话，就可以针对实现类来做动态代理了。

作者为什么这么设计，我认为还是有几个方面的原因：

1. 动态代理本身的使用场景或者需求，只是对原始实现的一个拦截，然后去做一些功能过的增强或者扩展。而实际的开发模式都是基于面向接口来开发的，所以基于接口来实现动态代理，从需求和场景，我认为都是吻合的。当然确实会存在一些没有实现接口的实现类，这个时候去实现动态代理，jdk很显然是无法满足的。
2. Java里面类的继承关系的设计更多的是考虑到共性能力的抽象，从而去提高代码的重用性和扩展性，动态代理也在做这样一件事，它封装了动态代理类的生成的抽象逻辑以及判断一个类是否是动态代理类，把这些抽象逻辑放到Proxy这个父类里面，很显然也是一个比较正常的思路。技术的方案设计是解决特定的场景问题的，如果一定要正对普通类做动态代理，我们可以选择cglib，它会动态生成一个被代理类的子类，子类重写父类的所有非final修饰的方法

上层的框架可能会随着业务的发展而变化，但是底层技术原理和设计理念是不会变化的。底层逻辑是对一个事务本质上的一个认识，而不是停留在表面上。

## 区别

JDK 代理 : 基于接口的动态代理技术·：利用拦截器（必须实现InvocationHandler）加上反射机制生成

一个代理接口的匿名类，再调用具体方法前调用InvokeHandler来处理，从而实现方法增强

CGLIB 代理 : 基于父类的动态代理技术：动态生成一个要代理的子类，子类重写要代理的类的所有不是

final的方法。在子类中采用方法拦截技术拦截所有的父类方法的调用，顺势织入横切逻辑，对方法进行

增强

# 2.订单表每天新增500W数据，分库分表的方案应该如何设计？在实际业务中存在两个比较多的查询。

根据订单id查询订单信息

根据用户id杏询订单列表

## 考点：

分库分表的理解，加两个分片键。

分库分表，避免出现容量满的情况，容量满了再去做数据迁移和容量扩充，成本会很高；

单表数据量过大的问题；

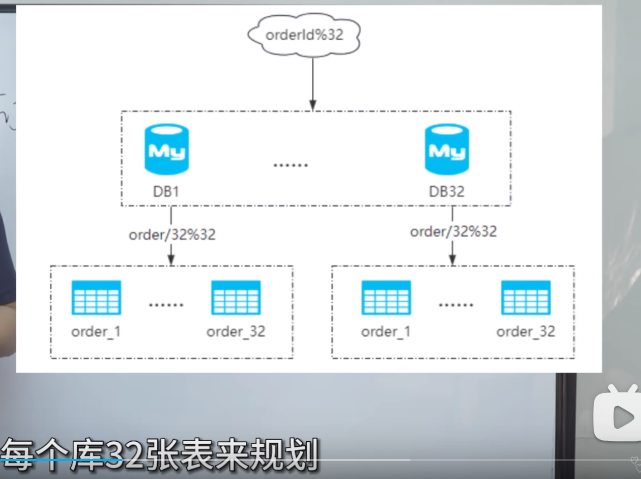
一天500万，一年有18亿的数量。保留两年的热数据，大概是40亿，空间预留是50亿数据。按32个库，每个库32张表来规划，每个表存放500万数据量。满足50亿数据容量规划。

选择orderid作为分片键，采用一致性hash算法进行路由。采用基因算法，来确保用户ID对应的订单ID,路由到同一个库，同一个表。把用户id的基因片段拼装到订单ID上，这样不管是通过订单ID还是用户ID都能路由的同一个表。

在性能方面

假设每个库正常写入的并发量是1000,那32个库可以承载32000的并发量。如果每个库的写性能再优化到1500，就意味着这个方案能支持接近5w每秒的写并发

热数据保特一年,一年以后的数据追多到历史库。历史库也就是冷库保存2年,超过这个时间的数据直接归档，从而实现一个1+2的存储方案



# 3.生产中遇到CPU运行过高

Top指令看具体进行，看那个进程cpu占用高。 然后去进程里看那个线程占用cpu高。将线程id转16进制。然后用jstack看根据线程id去找代码。

# 4. 在亿级别的数据库中，快速把uid=4的数据查出来，其中uid=4的数据查出来，而且这个uid=4的数据还很多。

传统的索引查询方式，可能因为数据量过大而变得效率低下。

## 考点

对数据库优化的理解；

是否有处理大量数据量的经验；

## 解决方案

1. 分库分表，根据uid使用一致性hash取模运算，把数据均匀的分配到不同的库和表中。降低表的数据量提升效率。
2. 使用搜索引擎，将uid同步到搜索引擎，搜索引擎可以提高更加高效的查询。
3. 数据冷热分离，访问频率高和频率低的数据分离，热数据存储在高性能的存储设备上（SSD）,冷数据存储在一般的磁盘上（进行备份和归档），减少数据表的规模来提升查询效率
4. 数据仓库和OLAP系统，

# 5.BigDecimal为何能解决浮点数运算时精度丢失问题

十进制整数在转化为二进制数时不会有精度问题，所以将十进制小数扩大N被让它在整数维度上进行计算（BigInteger类型），并记录小数点位置即可；

BigDecimal进行运算时分解为两部分，BigInteger间的计算，以及小数点位置scale的更新；例如加法运算时，先按整数计算，然后更新小数点的位置为两者小数位小的那个值

不要使用BigDecimal（double），存在精度损失风险，因为实际存储的值是double数值的二进制表示，可能会有精度损失

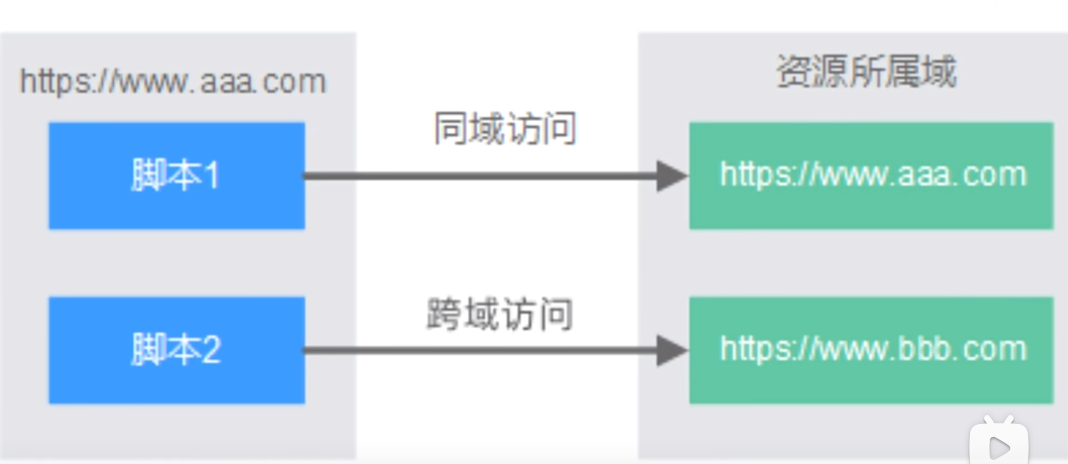
使用除法注意事项：使用roundingMode指明结果的保留规则，否则遇到无限循环小数时，会报错

BigDecimal等值比较时，使用CompareTo（），而不是equals（），因为equals会比较精度，而compareTo会忽略精度，那些无实际意义的0会自动忽略

# 6.go语言

提供简单，高效可靠的编程体验。Go语言的性能优势，在处理大规模云计算工作复杂的时候，

# 7. Springboot如何解决跨域问题？



就是在不破坏同源策略的情况下，能够安全地实现数据共享和交互。

在服务端配置那些域名支持跨域访问。

# 8.如何在Java中去实现无锁数据结

## 无锁数据结构

1.使用原子类

2.使用CAS操作

## Mysql update

3.mysql的update操作，即可以是行锁，也可以是表锁。

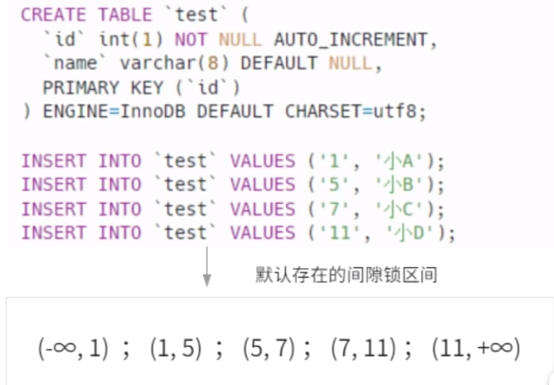
行锁：where条件包含了索引列,且只更新一条数据；

表锁：如果where条件中不包含索引列

## 行锁，表锁，临键锁，间隙锁



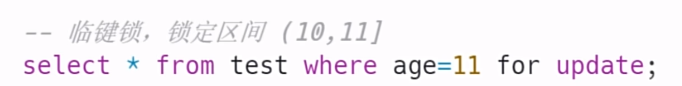
间隙锁表示锁定一个索引区间，在普通锁或者唯一索引的列上，由于索引是基于B+树的一个存储结构，所以默认会存在一个索引的一个区间；间隙锁就是某个事物对索引列加锁的时候默认锁定对应索引的左右开区间的一个范围，



在基于索引类的范围查询中，无论是否是唯一索引都会触发一个间隙锁，比如基于between范围查找的过程中就会产生一个左右开区间的间隙锁



临键锁=行锁+间隙锁的一个组合，它的锁定范围即包含了索引记录，也包含了索引区间



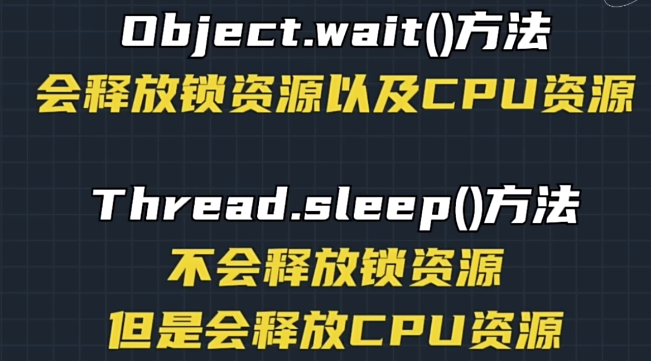
行锁，间隙锁，临键锁只是表示锁的一个数据范围，最终目的是解决幻读这个问题，使用非唯一索引进行精准匹配的时候，会默认加一个临键锁，因为它需要去锁定匹配的这一行记录，还需要去锁定这一行数据对应的左开右闭区间，因此在实际使用过程中，尽可能的使用唯一锁或者主键索引进行一个数据查询，避免大面积的锁造成性能影响

## Mvcc会加锁吗

通常不需要加锁来控制并发的访问，每个事务都可以读取到已提交事务的快照，而不需要去获取共享锁或者排他锁，在写操作的时候，mvcc会使用一种叫写时复制的技术，在修改数据之前，先将数据复制一份，从而创建一个新的快照，当一个事务需要修改数据的时候，mvcc会首先检查修改数据的版本号是否与该事务的快照版本一致，如果一致则可以修改这个数据，否则这个事务需要等其他事务完成对该数据的修改。另外这个事务在新快照之上修改的结果，不会影响原始数据，其它事务可以继续读取原快照，从而解决脏读和不可重复读的问题。有个mvcc机制，让多个事务对同一个数据进行读写的时候，不需要加锁，也不会出现读写冲突的问题。

## Wait和sleep

Wait和sleep是否会触发锁的释放，以及cpu资源的释放



凡是让线程进入阻塞状态的方法，操作系统层面都会去重新调度实现cpu时间的切换，这个设计是为了提升CPU的一个利用率。

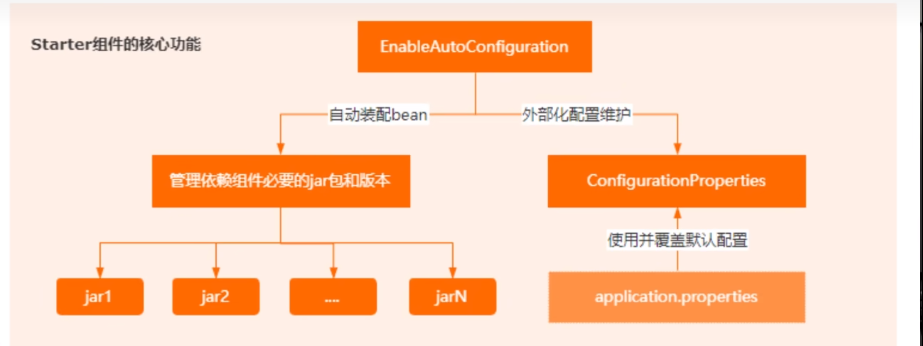
# 9.线程池中execute和submit的区别是什么

Submit要获取结果，execute直接执行不管。

# 10如何理解springboot中的starter组件

Stater是springboot的四大核心功能特性之一；自动装配，Actuator监控；只需要关系业务逻辑，减少对配置和外部环境的依赖。

Starter组件的作用：

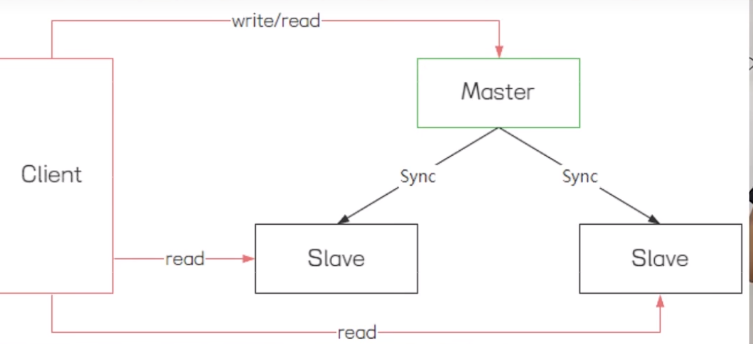


1. 以功能为维度，来维护对应jar包的版本依赖，无需关注版本冲突，导入所有jar依赖
2. 集成自动装配这个机制，程序依赖对应的starter组件以后，会自动集成到Spring的生态里面，对于相关bean的管理也是基于自动装配机制完成的。
3. 依赖starter组件后，外部化配置，会自动集成springboot里面，只需要在application.yml文件里维护就行
4. Stater组件完美的体现springboot约定大于配置这个理念。

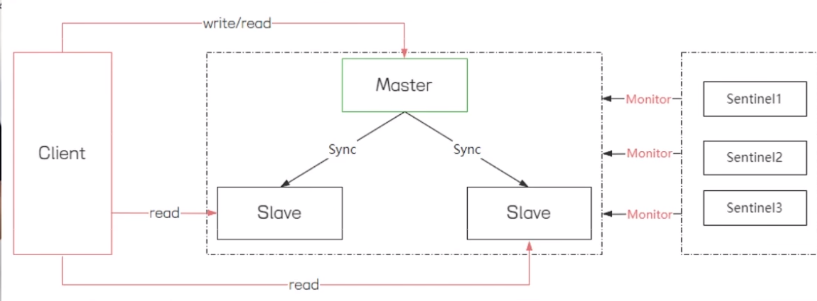
# 11.redis哨兵和集群有什么区别

Redis集群实现方式（主从复制，rediscluster）

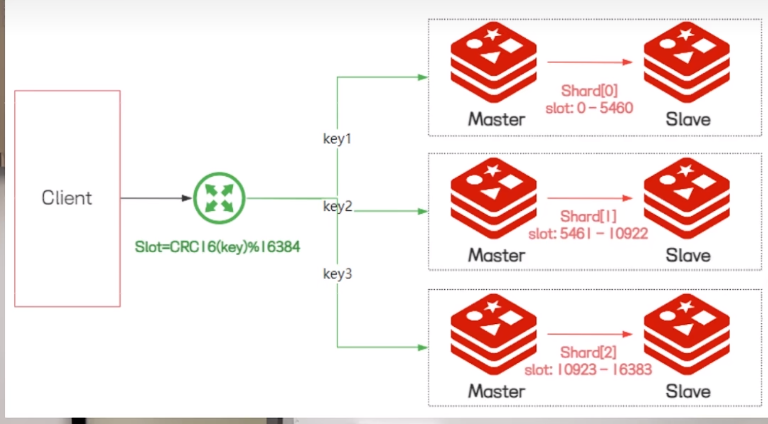
主从集群



实现redis读写分离，提升数据的查询效率。不提供容错和恢复 功能，一旦mastet节点挂了，不会自动选举新的master。哨兵机制用来监听，发生故障后能自动回复。



虽然解决了master的选举问题，但是在线扩容的问题还没有解决，所以有了redis集群



解决了在线扩容以及故障转移的能力。但客户端在实现会有些复杂，slave节点只是个冷备节点，不提供分担读写操作的压力，对于redis里面的批量操作指令会有限制。主从模式和集群模式各有优缺点，根据场景选择。

## 汇总

Redis哨兵集群式基于主从复制，实现读写分离，分担redis读操作的压力，而redis Cluster集群里面的slaver节点，只是实现冷备的一个机制，只有在master宕机后才会工作。

Redis哨兵集群，无法在线扩容，它的并发压力受限于单个服务器资源的一个配置。Redis cluster提供了基于sloat的分片机制，可以实现在线扩容，提升读写的性能。

Redis哨兵集群式一主多从，而redis cluster是一个多主多从的机制

# 12.mybatis是如何分页的

1.逻辑分页，查询所有，放到内存返回符合条件的

2.物理分页，用mysql的分页语法

Mybatis提供给4种分页方式

1. xml中直接写sql，方式灵活，比较简单
2. 一次性加载所有符合条件的目标数据，根据分页参数的值在内存里面实现分页
3. Interceptor拦截器，拦截需要分页的查询语句，然后动态拼接

# 13.Integer和int的区别

Integer默认是null,int默认是0；

Integer是存储在对内存里，int是直接存储在栈空间；

Integer是一个对象类型，它封装了很多的方法和属性，我们在使用的时候更加灵活

为啥要设计封装类型

Java本身是面向对象的语言，一切操作都是以对对象为基础。

# 14.lock和synchronized

功能上：lock和synchronized都是java中用来解决线程问题的一个工具

特性上：synchronized是Java中的同步关键字，lock是juc里面的一个接口，lock比synchronized的灵活性高