**Mysql数据库优化方案**

|  |
| --- |
| **1、 数据库选型：**首先应该从数据库的选型来看，首先是关系型数据库，目前最流行的是mysql和Oracle，mysql更适合于中小型项目，安装使用比较简单，免费开源。Oracle适合于大型项目，安装包比较大，使用比较耗内存，收费，但是效率高、可靠性好，使用行级锁，比mysql的表级锁粒度更细，对并发性的支持要好的多。NOSQL数据库（比如：mongdb），事务性比较差，但效率高，主要适用于数据量比较大，但关系型要求不是很强的数据，比如评论数据。缓存数据库（比如：redis），由于保存在内存中，所以速度特别快，但是不适应持久化保存。所以缓存数据库一般作为应用和数据库的中间层来使用。  **2、 设计规范：**其次应该从数据库设计规范来考虑。好的规范，可以是数据库运行效率更高，优化更方便。以mysql为例；  Ø 由于mysql对大小写在windows和linux上敏感型不一致，为了避免麻烦，我们一般建议数据库名、表名、字段名都用小写+下划线命名；  Ø 数据库名最后于项目名称一致。  Ø 表名最好采用：“数据库名\_表名”的方式；  Ø 字段名最后见名知意，不要太长；  Ø 字段选用合适的字段类型，比如姓名可以用varchar（10），而不要用默认的长度255，日期最好用date或datetime或时间戳。金额最好用decimal，而不是double、float等。  Ø 关联表以外的表最好都有主键id、创建时间、创建人、最后修改时间、最后修改人字段。  Ø 数据库设计时，以三大范式为参考，但是必要时，可以使用冗余；  Ø 对于预期数据量特别大的表，可能没有都会有很大增长的表，要考虑水平分表，比如系统日志表，可以考虑表名中加入月份，每月生成新表。  Ø 对于字段比较多的表，比如企业信息表，可以考虑纵向分表，把表分成两张1对1的表。把经常查询的字段放在一张表里，不经常用到的和大文本字段放在另外一张表里。  Ø 开发阶段，介意每个表只有主键，不应该包含其他外键。因为开发过程中，系统不稳定，可能出现很多垃圾数据，有了外键，数据的清理会给我们开发照成很多不便。要求开发人员用代码维护表之间的关系，这样更能增加代码的健壮性。  **3、 索引：**开发测试完成以后，一定要记得给数据库添加索引，增加sql的执行效率。添加索引也要注意，索引应该添加到经常作为查询条件或者排序字段、或者作为外键的列上。而那些不经常查询的字段、大文本字段、可选值比较少的字段（比如性别）等不需要创建索引。另外，sql中的不等于、is null、函数、like两边同时使用%等，会使索引失效，应该尽量避免使用；  **4、 配置：**数据库的配置优化（my.ini)，比如：最大连接数、最大缓存数、最大线程数、物理内存等；  **5、 读写分离**、主从复制优化。因为一般查询多、修改少，索引可以增加查询效率，但会影响修改效率。所以做了读写分离以后，可以把索引只建立在读的数据库上。并且查询不需要事务，所以可以用非事务行引擎；  **6、 数据库集群**，数据库压力过大时，用集群来分担压力是个很不错的方案；                       SQL优化  <https://mikechen.cc/3305.html> |

# **JAVA基础之多线程部分**

1、java中有几种方法可以实现一个线程？

继承Thread类

实现Runnable接口

实现Callable接口，需要实现的是call()方法

2、守护线程和本地线程区别？

守护线程：在start()方法之前调用Thread.setDaemon(boolon)设置为true;

区别：

唯一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自动创建的线程（但不一定），用户线程是程序创建的线程；比如JVM的垃圾回收线程是一个守护线程，当所有线程已经撤离，不再产生垃圾，守护线程自然就没事可干了，当垃圾回收线程是Java虚拟机上仅剩的线程时，Java虚拟机会自动离开。

3、死锁与活锁的区别，死锁与饥饿的区别？

死锁：两个或多个线程争夺CPU资源而处于互相等待的状态，此状态必须通过人为程序进行处理。

活锁：线程之间由于一直不满足某些条件导致重试、失败，活锁可自行解开。

饥饿：线程因为种种原因无法获得所需要的资源，导致一直无法执行的状态。

饥饿的原因：高优先的线程会吞并低优先级线程的cpu时间。

线程被永久堵塞在一个等待进入同步块的状态。

线程在等待一个调用了wait方法的对象。

3、如何停止一个正在运行的线程？

使用共享变量的方式

使用interrupt方法终止线程

4、notify()和notifyAll()有什么区别？

当一个线程进入wait之后，就必须等其他线程notify/notifyall,使用notifyall,可以唤醒所有处于wait状态的线程，使其重新进入锁的争夺队列中，而notify只能唤醒一个。如果没把握，建议notifyAll，防止notify因为信号丢失而造成程序异常。

5、什么是锁的可重入性？

可以重复利用已经获取的锁资源。

synchronized、ReentrantLock都是可重入的锁。

6、乐观锁和悲观锁的理解及如何实现，有哪些实现方式？

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人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁。

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人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量。

7、volatile有什么用？能否用一句话说明下volatile的应用场景？

volatile保证内存可见性和禁止指令重排。

volatile用于多线程环境下的单次操作(单次读或者单次写)。

8、在java中wait和sleep方法的不同？

最大的不同是在等待时wait会释放锁，而sleep一直持有锁。Wait通常被用于线程间交互，sleep通常被用于暂停执行。

9、什么是线程池？为什么要使用它？

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长，而且一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干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5开始，JavaAPI提供了Executor框架让你可以创建不同的线程池。

10、Thread类中的yield方法有什么作用？

将当前线程的执行状态变为就绪状态；

12、为什么wait(),notify()和notifyAll()必须在同步方法或者同步块中被调用？

当一个线程需要调用对象的wait()方法的时候，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入等待状态直到其他线程调用这个对象上的notify()方法。同样的，当一个线程需要调用对象的notify()方法时，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。由于所有的这些方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方法或者同步块中被调用。

13、同步方法和同步块，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）。同步方法会锁住整个对象，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止执行并需要等待获得这个对象上的锁。同步块更要符合开放调用的原则，只在需要锁住的代码块锁住相应的对象，这样从侧面来说也可以避免死锁。

深入解析多线程

14、创建线程的有哪些方式？

1、继承Thread类创建线程类

2、通过Runnable接口创建线程类

3、通过Callable和Future创建线程(future通过get()获取返回值)

4、通过线程池创建

15、什么是线程池？有哪几种创建方式？

线程池就是提前创建若干个线程，如果有任务需要处理，线程池里的线程就会处理任务，处理完之后线程并不会被销毁，而是等待下一个任务。由于创建和销毁线程都是消耗系统资源的，所以当你想要频繁的创建和销毁线程的时候就可以考虑使用线程池来提升系统的性能。

线程池常用的参数（

corePoolSize 最小线程数

、maximumPoolSize 最大线程数

、keepAliveTime 空闲线程存活时间

、unit 空闲线程存活时间单位

、workQueue 工作队列

、threadFactory 线程工厂

、handler 拒绝策略）

WorkQueue有四种

（基于数组的有界阻塞队列、基于链表的无界阻塞队列、不缓存队列、具有优先级的无界阻塞队列）

Handler JDK提供的四种策略（直接抛弃任务、直接丢弃任务并抛出异常、直接丢弃任务啥都不做、丢弃掉最早的任务并 将新任务加进来）

线程池工作的原理：当新任务来临时先判断线程是否空闲，如果空闲则直接执行，否则加入到队列中等待，当队列中加满任务后就会创建新线程去执行任务，如果达到最大线程数还有新任务则采取拒绝策略。

如何使用多线程求和？

1、创建线程池

2、调用sumit();

3、将futrue的get（）累计求和

四种线程池的创建：

1、newCachedThreadPool创建一个可缓存线程池2、newFixedThreadPool创建一个定长线程池，可控制线程最大并发数。3、newScheduledThreadPool创建一个定长线程池，支持定时及周期性任务执行。4、newSingleThreadExecutor创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务。

线程池的优点？

1、重用存在的线程，减少对象创建销毁的开销。2、可有效的控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。3、提供定时执行、定期执行、单线程、并发数控制等功能。

16、synchronized的作用？

在Java中，synchronized关键字是用来控制线程同步的，就是在多线程的环境下，控制synchronized代码段不被多个线程同时执行。synchronized既可以加在一段代码上，也可以加在方法上。

# **synchronized和lock区别**

　　1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；

　　2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；

　　3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；

　　4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。

　　5）Lock可以提高多个线程进行读操作的效率。

17、volatile关键字的作用？

对于可见性，Java提供了volatile关键字来保证可见性。当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。从实践角度而言，volatile的一个重要作用就是和CAS结合，保证了原子性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger。

18、什么是CAS？

CAS是compareandswap的缩写，即我们所说的比较交换。cas是一种基于锁的操作，而且是乐观锁。在java中锁分为乐观锁和悲观锁。悲观锁是将资源锁住，等一个之前获得锁的线程释放锁之后，下一个线程才可以访第201页共485页问。而乐观锁采取了一种宽泛的态度，通过某种方式不加锁来处理资源，比如通过给记录加version来获取数据，性能较悲观锁有很大的提高。CAS操作包含三个操作数——内存位置（V）、预期原值（A）和新值(B)。如果内存地址里面的值和A的值是一样的，那么就将内存里面的值更新成B。CAS是通过无限循环来获取数据的，若果在第一轮循环中，a线程获取地址里面的值被b线程修改了，那么a线程需要自旋，到下次循环才有机会执行。java.util.concurrent.atomic包下的类大多是使用CAS操作来实现的(AtomicInteger,AtomicBoolean,AtomicLong)。

CAS的问题？

CAS容易造成ABA问题一个线程a将数值改成了b，接着又改成了a，此时CAS认为是没有变化，其实是已经变化过了，而这个问题的解决方案可以使用版本号标识，每操作一次version加1。在java5中，已经提供了AtomicStampedReference来解决问题。

2、不能保证代码块的原子性CAS机制所保证的只是一个变量的原子性操作，而不能保证整个代码块的原子性。比如需要保证3个变量共同进行原子性的更新，就不得不使用synchronized了。

3、3、CAS造成CPU利用率增加之前说过了CAS里面是一个循环判断的过程，如果线程一直没有获取到状态，cpu资源会一直被占用。

19、数据库锁（悲观锁、乐观锁、行锁、表锁、页锁）

<https://mikechen.cc/5018.html>

20、锁优化

减少锁持有时间（能给代码块加锁尽量不要给整个方法加锁）

减少锁粒度（将大对象拆分成小对象，大大增加并行度，降低锁竞争eg：concurrentHashMap）

读写分离锁替代独占锁(使用ReadWriteLock将读写锁分离开来，有效提高系统并发能力)

锁分离

锁粗化

# **JAVA基础部分**

1、Java基本数据类型有哪些？

Byte、short、int、long、char、float、double、boollen

2、==和equals的区别？

==既可比较基本数据类型又可比较应用数据类型；

比较基本数据类型比较的是值，引用数据类型比较的是对象地址；

Equals 只能比较引用数据类型，且比较的是对象地址；一般在类库中会被按照要求重写；

3、public 、private、protected、default区别？

4、&和&&的区别？

&&双与表示左边条件不成立，后面就不会进行下去，多用于非空判断也可减少一些空指针；

&单与表示即使左边条件不成立，后面条件也会去执行；

5、static关键字的作用？

static关键字可以修饰类、方法、成员变量

Static修饰的类为静态类（里面只能是静态方法）

修饰的方法为静态方法（只能调用静态方法，普通方法可以调用静态方法和非静态方法）

修饰的成员变量为类变量（只可修饰全局变量）

普通类调用他们不需要实例化通过类名可直接调用；

6、final关键字的作用？

Final可修饰类、方法、变量

由final修饰的类不能被继承比如String；

修饰的方法不能被重写，修饰的变量只可赋值一次；

7、final、finally、finalize的区别？

Final 是关键字用于修饰类、方法、成员变量

Finally是异常处理结构的一部分，总是优先执行

Finalize 是GC回收垃圾对象的方法；

8、重写和重载的区别？

重写发生在父子类中方法名称和参数等都一样；

重载则发生在同一个类中，方法名可以相同参数列表不同；

9、构造器可以被重写或重载吗？

构造器可以被重载，但不能被重写；

10、私有方法和静态方法可以被重载或重写吗？

私有方法和静态方法可以被重载，不能被重写；

11、静态方法可以被继承吗？

可以被继承，但没有被重写，只会被隐藏起来；

12、java异常的分类有哪些？

Java 分为ERROR和Exception

ERROR属于非程序错误，例如内存溢出，无法修改程序来解决；

Exception属于程序异常又被分为运行时异常和非运行时异常

运行时异常是指程序启动后可正常编译，可处理可不处理，例如：空指针、数组越界

非运行时异常是指程序启动无法通过编译，必须人为去处理，如：IO异常、ClassNotFound等；

13、如何避免空指针？

14、throw和throws的区别？

Throws：类似一种规则抛出的是可能要发生但实际又不一定发生的异常；

Throw：通常用于抛出一个实实在在正在发生的异常；

15、Switch case支持哪几种数据类型？

Byte、short、int、char、string、枚举，不支持long

16、String可变吗？为什么？

17、String类可以被继承吗？

18、String字符串如何反转？

19、字符串分割方法有哪些？

20、String、StringBuffer、StringBuilder的区别？

21、普通类和抽象类的区别？

22、静态内部类和普通内部类的区别？

23、普通方法与静态方法的区别？

24、静态变量和普通变量的区别？

25、接口和抽象类的区别？

26、接口里面可以写方法实现吗？

27、抽象类能使用final修饰吗？

28、抽象类是否可以继承具体了？

29、Equals和hashcode的联系、区别？

30、try里面return，finally还会执行吗？

31、java8的新特性？

32、方法引用是什么，分为哪几类？

33、JDK实现定时任务有哪些方式？

# **JAVA基础之集合部分**

1、HashMap JDK1.7和1.8之后的区别 （数组+链表+红黑树）

Java8 对 HashMap 进行了一些修改，最大的不同就是利用了红黑树，所以其由 数组+链表+红黑树 组成。 根据 Java7 HashMap 的介绍，我们知道，查找的时候，根据 hash 值我们能够快速定位到数组的具体下标，但是之后的话，需要顺着链表一个个比较下去才能找到我们需要的，时间复杂度取决于链表的长度，为 O(n)。为了降低这部分的开销，在 Java8 中，当链表中的元素>= 8 数组长度>=64，会将链表转换为红黑树，在这些位置进行查找的时候可以降低时间复杂度为 O(logN)。 初始大小16，负载因子0.75，map中entry数量>阙值时扩容为原来2倍

2、TreeMap（可排序）

TreeMap实现SortedMap接口，能够把它保存的记录根据键排序，默认是按键值的升序排序，也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排过序的。 如果使用排序的映射，建议使用TreeMap。 在使用TreeMap时，key必须实现Comparable接口或者在构造TreeMap传入自定义的Comparator，否则会在运行时抛出java.lang.ClassCastException类型的异常。

3、LinkHashMap（记录插入顺序）

LinkedHashMap是HashMap的一个子类，保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的，也可以在构造时带参数，按照访问次序排序。

4、ArrayList（数组）初始大小10

ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力，就要将已经有数组的数据复制到新的存储空间中。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。

扩容为原来的1.5倍

5、LinkList（链表）无扩容、初始大小

 LinkedList是用链表结构存储数据的，很适合数据的动态插入和删除，随机访问和遍历速度比较慢。另外，他还提供了List接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆栈、队列和双向队列使用。

6、ConcurrentHashMap的并发度是什么？

ConcurrentHashMap把实际map划分成若干部分来实现它的可扩展性和线程安全。这种划分是使用并发度获得的，它是ConcurrentHashMap类构造函数的一个可选参数，默认值为16，这样在多线程情况下就能避免争用。在JDK8后，它摒弃了Segment（锁段）的概念，而是启用了一种全新的方式实现,利用CAS算法。同时加入了更多的辅助变量来提高并发度，具体内容还是查看源码吧。

ConcurrentHashMap 和 HashMap 思路是差不多的，但是因为它支持并发操作，所以要复杂一些。整个 ConcurrentHashMap 由一个个 Segment 组成，Segment 代表”部分“或”一段“的意思，所以很多地方都会将其描述为分段锁。注意，行文中，我很多地方用了“槽”来代表一个 segment。

3.4.2.2. 线程安全（Segment 继承 ReentrantLock 加锁）

简单理解就是，ConcurrentHashMap 是一个 Segment 数组，Segment 通过继承 ReentrantLock 来进行加锁，所以每次需要加锁的操作锁住的是一个 segment，这样只要保证每个 Segment 是线程安全的，也就实现了全局的线程安全。

Spring基础

1、IOC（控制反转）

将创建对象的控制权进行转移，由springIOC进行负责创建、管理对象；

2、spring在启动的时读取Bean配置信息，并在spring容器中生成一份Bean配置注册表，

根据Bean配置注册表实例化Bean，将实例化的bean放入到缓存池中并配置好bean之间的依赖关系。

3、SpringBean生命周期

（实例化——>设置属性——>初始化Bean——>使用Bean——>销毁并清理Bean）

4、四种注入方式

构造注入、setter注入、静态工厂注入、实例工厂注入

5、SpringAOP

"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，

并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共

同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未

来的可操作性和可维护性

使用"横切"技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流

程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生

在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP 的作用在于分离系统

中的各种关注点，将核心关注点和横切关注点分离开来。

**6、AOP 两种代理方式**

默认的策略是如果目标类是接口，

则使用 JDK 动态代理技术，否则使用 Cglib 来生成代理。

事务传播行为

## **PROPAGATION\_REQUIRED 有事务则支持事务，没有则开启新的事务**

## **PROPAGATION\_SUPPORTS 有事务则支持，否则就非事务执行**

## **PROPAGATION\_MANDATORY 有事务则支持，否则就抛异常**

PROPAGATION\_REQUIRES\_NEW 有事务则挂起并新建事务

## **PROPAGATION\_NOT\_SUPPORTED 挂起所有事务并执行非事务**

## **PROPAGATION\_NEVER 支持非事务执行，发现事务则抛异常**

## **PROPAGATION\_NESTED外层事务失败会回滚影响内层事务操作，内层事务失败不会影响外层事务**

Springboot启动注解（

Target、Documented被javadoc引用、ComponentScan扫描文件、springbootConfigtion

）

**7、Spring MVC 原理**

客户端请求提交到 DispatcherServlet，由 DispatcherServlet 控制器查询一个或多个 HandlerMapping，找到处理请求的

Controller。DispatcherServlet 将请求提交到 Controller，Controller 调用业务逻辑处理后，返回 ModelAndView。

DispatcherServlet 查询一个或多个视图解析器找到ModelAndView试图，试图负责将内容显示到客户端。

**7、Mybatis 缓存**

Mybatis 中有一级缓存和二级缓存，默认情况下一级缓存是开启的，而且是不能关闭的。一级缓存

是指 SqlSession 级别的缓存，当在同一个 SqlSession 中进行相同的 SQL 语句查询时，第二次以

后的查询不会从数据库查询，而是直接从缓存中获取，一级缓存最多缓存 1024 条 SQL。二级缓存

是指可以跨 SqlSession 的缓存。是 mapper 级别的缓存，对于 mapper 级别的缓存不同的

sqlsession 是可以共享的。

8、如何解决Spring循环依赖问题

构造注入、作用域等引发的循环依赖无法解决。

**Setter（）引发的循环依赖通过三级缓存**

**原理：在初始化的时候先查找一级缓存，如果查不到就查询二级缓存，如果在查不到就ObjectFactory.GetObject()通过三级缓存获取数据。**

BeanA首先完成了初始化的第一步（实例化），并且将自己放到singletonFactories（三级缓存）中，此时进行初始化的第二步（属性填充），发现自己依赖对象BeanB，此时就尝试去get(BeanB)，发现BeanB还没有被create，所以走create流程。

BeanB在初始化第一步（实例化）的时候发现自己依赖了对象BeanA，于是尝试get(BeanA)，尝试从一级缓存singletonObjects(肯定没有，因为BeanA还没初始化完全)，尝试从二级缓存earlySingletonObjects（也没有），尝试从三级缓存singletonFactories中获取，由于BeanA通过ObjectFactory将自己提前曝光了，所以BeanB能够通过ObjectFactory.getObject()拿到BeanA对象(虽然BeanA还没有初始化完全，但是可以被发现)，

BeanB拿到BeanA对象后顺利完成了初始化阶段1、2、3（实例化、填充属性、初始化），完全初始化之后将自己放入到一级缓存singletonObjects中。并且将BeanA放到二级缓存中，移除三级缓存中的BeanA。

此时返回BeanA中，BeanA此时能拿到BeanB的对象顺利完成自己的初始化阶段2、3（填充属性和初始化），最终BeanA也完成了初始化，将BeanA也添加到了一级缓存singletonObjects中。

MySQL基础介绍

1、存储引擎主要有： 1. MyIsam , 2. InnoDB, 3. Memory, 4. Archive, 5. Federated

2、MyIsa支持表锁更适用于以查询为主，只有少量按索引条件更新数据的应用

3、InnoDB支持行锁更适用于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用

**4、只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁**

**InnoDB（B+树）**

InnoDB 底层存储结构为B+树， B树的每个节点对应innodb的一个page，page大小是固定的，

一般设为 16k。其中非叶子节点只有键值，叶子节点包含完成数据。

B+树特性（索引字段尽量小、最左匹配）

数据项占空间越小，数据项越多，B+树高度越低，IO次数越少；

真实数据放入叶子节点

**MyIASM**

MyIASM是 MySQL默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，

因此当 INSERT(插入)或 UPDATE(更新)数据时即写操作需要锁定整个表，效率便会低一些。

ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。在设计之初就预想数据组织

成有固定长度的记录，按顺序存储的。---ISAM 是一种静态索引结构。

缺点是它不 支持事务处理。

**数据库是事务**

（原子性、一致性、隔离性、持久性）

**存储过程(特定功能的 SQL 语句集)**

一组为了完成特定功能的 SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次

编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过

程是数据库中的一个重要对象。

**存储过程优化思路：**

1. 尽量利用一些 sql 语句来替代一些小循环，例如聚合函数，求平均函数等。

2. 中间结果存放于临时表，加索引。

3. 少使用游标。sql 是个集合语言，对于集合运算具有较高性能。而 cursors 是过程运算。比如

对一个 100 万行的数据进行查询。游标需要读表 100 万次，而不使用游标则只需要少量几次

读取。

4. 事务越短越好。sqlserver 支持并发操作。如果事务过多过长，或者隔离级别过高，都会造成

并发操作的阻塞，死锁。导致查询极慢，cpu 占用率极地。

5. 使用 try-catch 处理错误异常。

6. 查找语句尽量不要放在循环内。

**数据库并发策略**

并发控制一般采用三种方法，分别是乐观锁和悲观锁以及时间戳。

**数据库锁**

行级锁是一种排他锁，防止其他事务修改此行；在使用以下语句时，Oracle 会自动应用行级锁：

1. INSERT、UPDATE、DELETE、SELECT … FOR UPDATE [OF columns] [WAIT n | NOWAIT];

2. SELECT … FOR UPDATE 语句允许用户一次锁定多条记录进行更新

3. 使用 COMMIT 或 ROLLBACK 语句释放锁。

表级锁

表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分 MySQL 引擎支持。最常使

用的 MYISAM 与 INNODB 都支持表级锁定。表级锁定分为表共享读锁（共享锁）与表独占写锁

（排他锁）。

**页级锁**

Mysql回表

普通索引查询方式，则需要先搜索 k 索引树，得到 ID 的值为 500，再到 ID 索引树搜索一次。这个过程称为回表（无法一次性命中所有字段，只能通过回表的方式查询未命中的字段大大降低查询效率）。

避免回表可以使用组合索引。

**Redis**

Redis是一个继memcached后的又一个第三方缓存数据库，他比memcached强大很多，支持更多的数据类型（String、list、set、sort set、hash），支持持久化，支持集群；Redis虽然支持持久化，但是他并不适合持久化的保存数据。因为他不是很稳定。但是由于他是保存在内存中，读取速度非常快，所以在项目中一般都用它作为数据库和应用程序直接的中间层来使用，已减轻数据库压力，提高运行效率。

我们项目中很多地方用到了redis；比如商品的三级分类、省市县、关于我们、联系我们、友情链接，常见问题等经常查询但是不经常改变的数据.redis还可以在tomcat集群里边实现session的共享。由于他的单线程的，所以在电商平台里边也经常用他做“防止超卖”, 生成规则的商品编号等。还有就是购物车也用到了redis

 代码中，我们一般都通过spring整合redis官方提供的jedis工具包来操作redis。可以是单机版，也可以是集群。Redis本身就支持集群操作redis\_cluster，另外redis还支持主从复制，还有他独特的哨兵模式，在主服务器宕机时，从服务器可以自动转换为主服务器。另外，他也有他的分片机制，就像mysql的水平分表设计，数据量大时，可以把数据存储到不同的库中，来减轻redis负担。

Redis的持久化方式主要有2种，RDB和AOF，RDB是一种快照方式，默认每隔5分钟创建一个快照副本，这种方式占用空间大，而且会丢失间隔时间5分钟之内的数据，但是他适合做备份，恢复时，可以根据需要恢复任意间隔时间点的数据。AOF是一种日志的持久化记录方式，每秒钟，都把redis中新增的数据记录到日志文件中，这种方式只有一个文件，占用空间少，最多丢失1秒内的数据。相对比较好，但是如果想要恢复5分钟或10分钟前某个时间点的数据，就不行了。所以实际项目中，我们一般会两种方式同时使用。如果搭建集群的话，还可以通过集群互相备份数据，只要集群不同时挂掉，单个redis就可以从集群中的其他服务器获取到最新数据。

还有就是，由于redis不是很稳定，有时候会发生“穿透”和“雪崩”；

缓存雪崩可能是因为数据未加载到缓存中，或者缓存同一时间大面积的失效，从而导致所有请求都去查数据库，导致CPU和内存负载过高，甚至宕机。

解决思路：

1，采用加锁计数，或者使用合理的队列数量来避免缓存失效时对数据库造成太大的压力。这种办法虽然能缓解数据库的压力，但是同时又降低了系统的吞吐量。

2，分析用户行为，尽量让失效时间点均匀分布。避免缓存雪崩的出现。

3，如果是因为某台缓存服务器宕机，可以考虑做主备，比如：Redis主备，但是双缓存涉及到更新事务的问题，update可能读到脏数据，需要好好解决。

缓存穿透是指用户查询数据，在数据库没有，自然在缓存中也不会有。这样就导致用户查询的时候，在缓存中找不到，每次都要去数据库中查询。

解决思路：

1，如果查询数据库也为空，直接设置一个默认值存放到缓存，这样第二次到缓冲中获取就有值了，而不会继续访问数据库，这种办法最简单粗暴。

2，根据缓存数据Key的规则。例如我们公司是做机顶盒的，缓存数据以Mac为Key，Mac是有规则，如果不符合规则就过滤掉，这样可以过滤一部分查询。在做缓存规划的时候，Key有一定规则的话，可以采取这种办法。这种办法只能缓解一部分的压力，过滤和系统无关的查询，但是无法根治。

3，采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的BitSet中，不存在的数据将会被拦截掉，从而避免了对底层存储系统的查询压力。关于布隆过滤器，详情查看：基于BitSet的布隆过滤器(Bloom Filter)

大并发的缓存穿透会导致缓存雪崩