1. JAVA 基础

一、jvm jre jdk 分别是什么?

a. JDK(Java Development Kit) 是 Java 语言的软件开发工具包(SDK)

b. JRE(Java Runtime Environment ,Java 运行环境），包含 JVM 标准实现及 Java 核心类库。JRE 是 Java 运行环境，并不是一个开发环境，所以没有包含任何开发工具（如编译器和调试器

c. JVM 是 Java Virtual Machine（Java 虚拟机）的缩写，JVM 是一种用于计算设备的规范，它是一个虚构出来的计算机

二、Java 三大特性

a. **封装**： 将类的某些信息隐藏在类的内部，不允许外部程序访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问

b. 继承是类与类的一种关系

c. 多态指对象的多种引用形态，继承是多态的前提

三、Java 八大基本数据类型

a. 浮点型：float，double

b. 字符型：char

c. 整型：byte，short，int,long

d. Boolean 型：boolean

四、float f=3.4;是否正确

答案：不正确。

原因：精度不准确,应该用强制类型转换，如下所示：float f=(float)3.4 或 float f = 3.4f

五、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

答：对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

六、&、| 和&&、||的区别

答：&、|有两种用法：(1)按位运算；(2)逻辑运算。逻辑运算时不会短路

例：5&4 （0101&0100=0100）=4

例：5|4（0101|0100=0101）=5

&&、||逻辑运算符，会短路。

七、Math.round(11.5) 等于多少？Math.round(-11.5)等于多少

答：Math.round(11.5)的返回值是 12，Math.round(-11.5)的返回值是-11

八、hashcode 与 equal

1.equal()相等的两个对象他们的 hashCode()肯定相等，

2.hashCode()相等的两个对象他们的 equal()不一定相等，

所有对于需要大量并且快速的对比的话如果都用 equal()去做显然效率太低，所以解决方式是，每当需要对比 的时候，首先用 hashCode()去对比，如果 hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用 equal()去再对比了）,如果 hashCode()相同，此时再对比他们的 equal()，如果 equal()也相同，则表示这两个对 象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

九、String StringBuilder StringBuffer 的区别

1.String 为字符串常量，而 StringBuilder 和 StringBuffer 均为字符串变量，即 String 对象一旦创建之后该对象是不可更改的，但后两者的对象是变量，是可以更改的。

2.在线程安全上，StringBuilder 是线程不安全的，而 StringBuffer 是线程安全的

十、方法重载与重写的区别

重写：也叫子类的方法覆盖父类的方法，要求返回值、方法名和参数都相同。

子类抛出的异常不能超过父类相应方法抛出的异常。(子类异常不能超出父类异常)

子类方法的的访问级别不能低于父类相应方法的访问级别(子类访问级别不能低于父类访问级别)

重载：是在同一个类中的两个或两个以上的方法，拥有相同的方法名，但是参数却不相同，方法体也不相同，

十一、讲下 java 集合

答：Java 集合类存放于 java.util 包中，是一个用来存放对象的容器。Collection 是 List 接口和 Set 接口的父接口

List 实现类：

Vector 底层数据结构是数组，查询快增删慢;线程安全，效率低,

ArrayList 底层数据结构是数组查询快增删慢;线程不安全，效率高，

ArrayList 初始容量为 10 扩容算法（原来的容量*3）/2+1

LinkedList 底层数据结构是链表，查询慢，增删快;线程不安全，效率高

Set 实现类：

HashSet:不能保证元素的顺序不可重复；不是线程安全的；集合元素可以为空不可以重复，有序

因为底层采用 链表 和 哈希表的算法。链表保证元素的添加顺序，哈希表保证元素的唯一性

TreeSet:有序；不可重复，底层使用 红黑树算法，擅长于范围查询。

Map 的实现类

1、HashMap

它是线程不安全的 Map，方法上都没有 synchronize 关键字修饰(初始容量为 16 ，加载因子为 0.75 ，扩容为 2 倍)

2、HashTable

hashTable 是线程安全的一个 map 实现类,它实现线程安全的方法是在各个方法上添加了 synchronize 关键字。但是现在已经不再推荐使用 HashTable 了，因为现在有了 ConcurrentHashMap 这个专门用于多线程场景下的 map 实现类，其大大优化了多线程下的性能。

3、ConcurrentHashMap

这个 map 实现类是在 jdk1.5 中加入的，其在 jdk1.6/1.7 中的主要实现原理是 segment 段锁，它不再使用和 HashTable 一样的 synchronize 一样的关键字对整个方法进行枷锁，而是转而利用 segment 段落锁来对其进行加锁，以保证 Map 的多线程安全。

其实可以理解为，一个 ConcurrentHashMap 是由多个 HashTable 组成，所以它允许获取不用段锁的线程同时持有该资源，segment 有多少个，理论上就可以同时有多少个线程来持有它这个资源。

其默认的 segment 是一个数组，默认长度为 16。也就是说理论商可以提高 16 倍的性能
在 JAVA 的 jdk1.8 中则对 ConcurrentHashMap 又再次进行了大的修改，取消了 segment 段锁字段，采用了 CAS+Synchronize 技术来保障线程安全。底层采用数组+链表+红黑树的存储结构

十二、线程和进程的区别

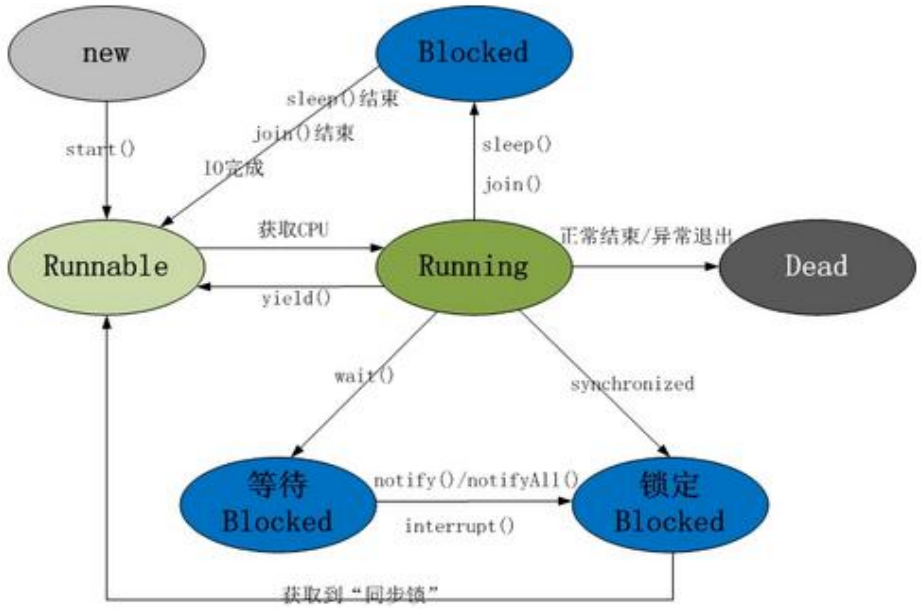
1.一个进程至少有一个线程

2.进程是资源的分配和调度的一个独立单元，而线程是 CPU 调度的基本单元.

3.一个线程只能属于一个进程，但是一个进程可以拥有多个线程。多线程处理就是允许一个进程中在同一时刻执行多个任务。

十三、线程的生命周期

关于Java中线程的生命周期，首先看一下下面这张较为经典的图：



新建状态（New）：当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread();

就绪状态（Runnable）：当调用线程对象的 start()方法，线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待 CPU 调度执行，并不是说执行了 start()此线程立即就会执行；

运行状态（Running）：当 CPU 开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就　　绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

阻塞状态（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对 CPU 的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被 CPU 调用以进入到运行状态。根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行状态中的线程执行 wait()方法，使本线程进入到等待阻塞状态；

2.同步阻塞 -- 线程在获取 synchronized 同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

3.其他阻塞 -- 通过调用线程的 sleep()或 join()或发出了 I/O 请求时，线程会进入到阻塞状态。当 sleep()状态超时、join()等待线程终止或者超时、或者 I/O 处理完毕时，线程重新转入就绪状态。

死亡状态（Dead）：线程执行完了或者因异常退出了 run()方法，该线程结束生命周期。

十四、sleep yield wait join 方法的区别

sleep()方法

运行中的线程，执行 sleep()方法，放弃 CPU，转到阻塞状态。但不会放弃占有的资源，例如锁。sleep()时间结束，进入可运行状态。

yield()方法

执行 yield()方法，如果此时有相同或更高优先级的其他线程处于就绪状态，那么 yield()方法把当前线程放到可运行池中。如果只有相同优先级的线程，那么，该线程可能接着马上执行。如果没有相同或更高优先级线程，则什么也不做。yield()方法并不会释放锁。

wait()方法

wait 方法用于线程通信，与 notify，nofityall 配合使用。它必须在 synchronized 语句块内使用。wait()方法使当前线程暂停执行并释放对象锁标示，让其他线程可以进入 synchronized 数据块，当前线程被放入对象等待池。有关具体的内容，在下面一篇文章中详细讲解。需要注意，该方法是 java.lang.Object 的方法。

join()方法

当前运行的线程执行另一个线程的 join()方法，当前线程进入阻塞状态，直到另一个线程运行结束，它才会恢复到可就绪状态。join()方法，调用了 wait()，因此它会释放锁。

十五、创建线程的三种方式

一、继承 Thread 类创建线程类

（1）定义 Thread 类的子类，并重写该类的 run 方法，该 run 方法的方法体就代表了线程要完成的任务。因此把 run()方法称为执行体。

（2）创建 Thread 子类的实例，即创建了线程对象。

（3）调用线程对象的 start()方法来启动该线程。

二、通过 Runnable 接口创建线程类

（1）定义 runnable 接口的实现类，并重写该接口的 run()方法，该 run()方法的方法体同样是该线程的线程执行体。

（2）创建 Runnable 实现类的实例，并依此实例作为 Thread 的 target 来创建 Thread 对象，该 Thread 对象才是真正的线程对象。

（3）调用线程对象的 start()方法来启动该线程。

三、通过 Callable 和 Future 创建线程

（1）创建 Callable 接口的实现类，并实现 call()方法，该 call()方法将作为线程执行体，并且有返回值。

（2）创建 Callable 实现类的实例，使用 FutureTask 类来包装 Callable 对象，该 FutureTask 对象封装了该 Callable 对象的 call()方法的返回值。

（3）使用 FutureTask 对象作为 Thread 对象的 target 创建并启动新线程。

（4）调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值

十六、Io 流

1、IO 流的分类

根据处理数据类型的不同分为：字符流和字节流

根据数据流向不同分为：输入流和输出流

A）输入流和输出流区别

输入流只能进行读操作，输出流只能进行写操作，程序中需要根据待传输数据的不同特性而使用不同的流。

B) 字符流和字节流区别

读写单位不同：字节流以字节（8bit）为单位，字符流以字符为单位，根据码表映射字符，一次可能读多个字节。

处理对象不同：字节流能处理所有类型的数据（如图片、视频等），而字符流只能处理字符类型的数据

字节流：一次读入或读出是 8 位二进制。

字符流：一次读入或读出是 16 位二进制。

设备上的数据无论是图片或者视频，文字，它们都以二进制存储的。二进制的最终都是以一个 8 位为数据单元进行体现，所以计算机中的最小数据单元就是字节。意味着，字节流可以处理设备上的所有数据，所以字节流一样可以处理字符数据。

结论：只要是处理纯文本数据，就优先考虑使用字符流。 除此之外都使用字节流。

C)缓冲流：

BufferedInputStrean 、BufferedOutputStream、 BufferedReader、 BufferedWriter 增加缓冲功能，避免频繁读写硬盘

十七、什么是反射

答：JAVA 反射机制是在运行状态中，对于任意一个实体类，都能够知道这个类的所有属性和方法；对于任意一个 对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为 java 语言的反射机制。

十八、创建对象的 5 种方式

使用new关键字	} → 调用了构造函数
使用Class类的newInstance方法	} → 调用了构造函数
使用Constructor类的newInstance方法	} → 调用了构造函数
使用clone方法	} → 没有调用构造函数
使用反序列化	} → 没有调用构造函数

十九、Synchronized lock volatile 的区别

一、synchronized 与 Lock 的区别

1. 首先 synchronized 是 java 内置关键字，在 jvm 层面，Lock 是个 java 类；
2. synchronized 无法判断是否获取锁的状态，Lock 可以判断是否获取到锁；
3. synchronized 会自动释放锁(a 线程执行完同步代码会释放锁 ； b 线程执行过程中发生异常会释放锁)，Lock 需在 finally 中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；
4. 用 synchronized 关键字的两个线程 1 和线程 2，如果当前线程 1 获得锁，线程 2 线程等待。如果线程 1 阻塞，线程 2 则会一直等待下去，而 Lock 锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；
5. synchronized 的锁可重入、不可中断、非公平，而 Lock 锁可重入、可判断、可公平（两者皆可）
6. Lock 锁适合大量同步的代码的同步问题，synchronized 锁适合代码少量的同步问题。

二、synchronized 和 volatile 区别

1. volatile 关键字解决的是变量在多个线程之间的可见性；而 sychronized 关键字解决的是多个线程之间访问共享资源的同步性。
2. volatile 只能用于修饰变量，而 synchronized 可以修饰方法，以及代码块。（volatile 是线程同步的轻量级实现，所以 volatile 性能比 synchronized 要好，
3. 多线程访问 volatile 不会发生阻塞，而 sychronized 会出现阻塞。
4. volatile 能保证变量在多个线程之间的可见性，但不能保证原子性；而 sychronized 可以保证原子性，也可以间接保证可见性，因为它会将私有内存和公有内存中的数据做同步。

线程安全包含原子性和可见性两个方面。

对于用 volatile 修饰的变量，JVM 虚拟机只是保证从主内存加载到线程工作内存的值是最新的。

三、公平锁和非公平锁的区别

公平锁，就是很公平，在并发环境中，每个线程在获取锁时会先查看此锁维护的等待队列，如果为空，或者当前线程是等待队列的第一个，就占有锁， 否则就会加入到等待队列中，以后会按照 FIFO 的规则从队列中取到自己

非公平锁比较粗鲁，上来就直接尝试占有锁，如果尝试失败，就再采用类似公平锁那种方式

二十、写出 java 5 种运行时异常

NullPointerException - 空指针引用异常　　ClassCastException - 类型强制转换异常。

IllegalArgumentException - 传递非法参数异常。 ArithmeticException - 算术运算异常

IndexOutOfBoundsException - 下标越界异常　NumberFormatException - 数字格式异常

二十一、final、finally、finalize 区别

final 用于修饰类、成员变量和成员方法。final 修饰的类，不能被继承（String、StrngBuilder、StringBuffer、Math，不可变类），其中所有的方法都不能被重写，所有不能同时用 abstract 和 final 修饰（abstract 修饰的是抽象类，抽象类是用于被子类继承的，和 final 起相反的作用）；final 修饰的方法不能被重写，但是子类可以用父类中 final 修饰的方法；final 修饰的成员变量是不可变的，如果成员变量是基本数据类型，初始化之后成员变量的值不能被改变，如果成员变量是引用类型，那么它只能指向初始化时指向的那个对

finally 是在异常处理时提供 finally 块来执行任何清除操作。不管有没有异常被抛出、捕获都会被执行。try 块中的内容是在无异常时执行到结束。catch 块中的内容，是在 try 块内容发生 catch 所声明的异常时，跳转到 catch 块中执行。finally 块则是无论异常是否发生都会执行 finally 块的内容，所以在代码逻辑中有需要无论发生什么都必须执行的代码，可以放在 finally 块中。

finalize 是方法名，java 技术允许使用 finalize（）方法在垃圾收集器将对象从内存中清楚出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的，它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize（）方法以整理系统资源或者执行其他清理工作。 finalize 方法是在垃圾收集器删除对象之前对这个对象调用的。

Finally 一定会被执行吗？

至少有两种情况下 finally 语句是不会被执行的：
（1）try 语句没有被执行到，如在 try 语句之前就返回了，这样 finally 语句就不会执行，这也说明了 finally 语句被执行的必要而非充分条件是：相应的 try 语句一定被执行到。
（2）在 try 块中有 System.exit(0);这样的语句，System.exit(0);是终止 Java 虚拟机 JVM 的，连 JVM 都停止了，所有都结束了，当然 finally 语句也不会被执行到

finally 语句是在 try 的 return 之前执行还是之后执行？？？

finally 块的语句在 try 或 catch 中的 return 语句执行之后 返回之前执行且 finally 里的修改语句不能影响 try 或 catch 中 return 已经确定的返回值 ,若 finally 里也有 return 语句则覆盖 try 或 catch 中的 return 语句直接返

2. 数据库

一、数据库三大范式

第一范式（1NF）是指数据表中的每个字段必须是不可拆分的最小单元，也就是确保每一列的原子性。

第二范式（2NF）是指满足 1NF 后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情。

第三范式（3NF）是指必须先满足第二范式（2NF），另外要求表中的每一列只与主键直接相关而不是间接相关。

二、数据库五大约束

1. 主键约束（Primay Key Coustraint） 唯一性，非空性；

2. 唯一约束 （Unique Counstraint）唯一性，可以空，但只能有一个；

3. 默认约束 (Default Counstraint) 该数据的默认值；

4. 外键约束 (Foreign Key Counstraint) 需要建立两表间的关系；

5. 非空约束（Not Null Counstraint）:设置非空约束，该字段不能为空。

三、分页

Mysql: select * from emp limit 0 ,5;

Oracle: select e.* from (select emp.* ,rownum rn from emp) e where rn between 1 and 5;

四、oracel 与 mysql 的区别

A).自动增长的数据类型处理

MYSQL 有自动增长的数据类型，插入记录时不用操作此字段，会自动获得数据值。ORACLE 没有自动增长的数据类型，需要建立一个自动增长的序列号，插入记录时要把序列号的下一个值赋于此字段。

B).单引号的处理

MYSQL 里可以用双引号包起字符串，ORACLE 里只可以用单引号包起字符串。在插入和修改字符串前必须做单引号

的替换：把所有出现的一个单引号替换成两个单引号。

C).空字符的处理

MYSQL 的非空字段也有空的内容，ORACLE 里定义了非空字段就不容许有空的内容。按 MYSQL 的 NOT NULL 来定义 ORACLE 表结构，导数据的时候会产生错误。因此导数据时要对空字符进行判断，如果为 NULL 或空字符，需要把它改成一个空格的字符串。

D).日期字段的处理

MYSQL 日期字段分 DATE 和 TIME 两种，ORACLE 日期字段只有 DATE，包含年月日时分秒信息用当前数据库的系统时间为 SYSDATE，精确到秒

E).并发性

mysql 以表级锁为主，对资源锁定的粒度很大，如果一个 session 对一个表加锁时间过长，会让其他 session 无法更新此表中的数据。虽然 InnoDB 引擎的表可以用行级锁，但这个行级锁的机制依赖于表的索引，如果表没有索引或者 sql 语句没有使用索引，那么仍然使用表级锁。

oracle 使用行级锁，对资源锁定的粒度要小很多，只是锁定 sql 需要的资源，并且加锁是在数据库中的数据行上不依赖与索引。所以 oracle 对并发性的支持要好很多

F).提交方式

oracle 默认不自动提交，需要用户手动提交。mysql 默认是自动提交。

五、数据库优化

（1）Sql 语句优化

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断。

3.应尽量避免在 where 子句中使用!=或<>操作符， MySQL 只有对以下操作符才使用索引：<，<=，=，>，>=

5.in 和 not in 也要慎用，否则会导致全表扫描，对于连续的数值，能用 between 就不要用 in 了

6.应尽量避免在 where 子句中对字段进行表达式操作，应尽量避免在 where 子句中对字段进行函数操作

7.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，

（2）索引优化

1.数据量超过百万级别的表应该有索引；

2.经常与其他表进行连接的表，在连接字段上应该建立索引；

3.经常出现在 Where 子句 order by group by 中的字段，特别是大表的字段，应该建立索引；

4.索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；

（3）使用缓存

（4）读写分离，分表分库分区，活跃数据分离

六、索引

（1）什么索引

a．什么是索引？？？索引是对数据库表中一列或多列的值进行排序的一种结构

（2）索引的优点

①建立索引的列可以保证行的唯一性，生成唯一的 rowId

②建立索引可以有效缩短数据的检索时间

③建立索引可以加快表与表之间的连接

④为用来排序或者是分组的字段添加索引可以加快分组和排序顺序

（3）索引的缺点

①创建索引和维护索引需要时间成本，这个成本随着数据量的增加而加大

②创建索引和维护索引需要空间成本，每一条索引都要占据数据库的物理存储空间，数据量越大，占用空间也越大（数据表占据的是数据库的数据空间）

③会降低表的增删改的效率，因为每次增删改索引需要进行动态维护，导致时间变长

（4）索引什么时候会失效

1.like 查询是以%开头

2.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引

3.对索引列进行运算.需要建立函数索引.否则索引会失效

（5）什么样的列需要建立索引

1.在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

2.在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询 时间

3.在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度.

（6）什么样的列不需要建立索引

1、频繁更新的字段不适合创建索引，因为每次更新不单单是更新记录，还会更新索引，保存索引文件；

2、表记录太少，不需要创建索引；

3、.经常增删改的表；

4、.数据重复且分布平均的字段，因此为经常查询的和经常排序的字段建立索引。注意某些数据包含大量重复数据，因此他建立索引就没有太大的效果，例如性别字段，只有男女，不适合建立索引。

（7）常见索引

Mysql 常见索引有：主键索引、唯一索引、普通索引、全文索引、组合索引
普通索引：最基本的索引，没有任何限制
唯一索引：与"普通索引"类似，不同的就是：索引列的值必须唯一，但允许有空值。
主键索引：它 是一种特殊的唯一索引，不允许有空值。
全文索引：仅可用于 MyISAM 表，针对较大的数据，生成全文索引很耗时好空间。
组合索引：用多个列组成的索引。

七、存储过程 存储函数

存储过程/函数是什么？存储过程和存储函数是事先经过编译并存储在数据库中的一段 sql 语句的集合

存储过程与存储函数的区别？

1.存储函数必须有返回值，而存储过程没有返回值。

2.存储过程的参数可以是 in ，out 类型，存储函数的参数类型只能是 in

（1）优点

• 运行速度：对于复杂的业务逻辑，因为在存储过程创建的时候，数据库已经对其进行了一次解析和优化。存储过程一旦执行，在内存中就会保留一份这个存储过程，这样下次再执行同样的存储过程时，可以从内存中直接调用，所以执行速度会比普通 sql 快。

•减少网络传输：存储过程直接就在数据库服务器上跑，所有的数据访问都在数据库服务器内部进行，不需要传输数据到其它服务器，所以会减少一定的网络传输。但是在存储过程中没有多次数据交互，那么实际上网络传输量和直接 sql 是一样的。而且我们的应用服务器通常与数据库是在同一内网，大数据的访问的瓶颈会是硬盘的速度，而不是网速。

（2）缺点

开发调试复杂，由于 IDE 的问题，存储过程的开发调试要比一般程序困难

•没办法应用缓存。虽然有全局临时表之类的方法可以做缓存，但同样加重了数据库的负担。如果缓存并发严重，经常要加锁，那效率实在堪忧

•不支持群集，数据库服务器无法水平扩展，或者数据库的切割（水平或垂直切割）。数据库切割之后，存储过程并不清楚数据存储在哪个数据库中

•移植性差

八、事务的四大特性

原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都提交，要么都回滚。 比如：转账转过去的加和转时候的减必须一次发生

一致性（Consistency）

事务必须使数据库从一个一致性状态变换到另外一个一致性状态。 比如：转账时双方的总数在转账的时候保持一致

隔离性（Isolation）

事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

比如：多个用户操纵，防止数据干扰，就要为每个客户开启一个自己的事务；

持久性（Durability）

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

九、事务的传播机制

REQUIRED（默认）：支持使用当前事务，如果当前事务不存在，创建一个新事务。

SUPPORTS：支持使用当前事务，如果当前事务不存在，则不使用事务。

MANDATORY：中文翻译为强制，支持使用当前事务，如果当前事务不存在，则抛出 Exception。

REQUIRES_NEW：创建一个新事务，如果当前事务存在，把当前事务挂起。

NOT_SUPPORTED：无事务执行，如果当前事务存在，把当前事务挂起。

NEVER：无事务执行，如果当前有事务则抛出 Exception。

NESTED：嵌套事务，如果当前事务存在，那么在嵌套的事务中执行。如果当前事务不存在，则表现跟 REQUIRED 一样。

十、MyISAM 和 InnoDB 两者之间区别

1）存储结构

MyISAM：每个 MyISAM 在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm 文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。
InnoDB：所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB。

2）事务支持

MyISAM：强调的是性能，每次查询具有原子性,其执行数度比 InnoDB 类型更快，但是不提供事务支持。
InnoDB：提供事务支持事务，外键。 具有事务(commit)、回滚(rollback)功能

3）表锁差异

MyISAM：只支持表级锁，用户在操作 myisam 表时，select，update，delete，insert 语句都会给表自动加锁，如果加锁以后的表满足 insert 并发的情况下，可以在表的尾部插入新的数据。
InnoDB：支持事务和行级锁，是 innodb 的最大特色。行锁大幅度提高了多用户并发操作的新能。但是 InnoDB 的行锁，只是在 WHERE 的主键是有效的，非主键的 WHERE 都会锁全表的。

MyISAM 锁的粒度是表级，而 InnoDB 支持行级锁定。简单来说就是，InnoDB 支持数据行锁定，而 MyISAM 不支持行锁定，只支持锁定整个表。即 MyISAM 同一个表上的读锁和写锁是互斥的，MyISAM 并发读写时如果等待队列中既有读请求又有写请求，默认写请求的优先级高，即使读请求先到，所以 MyISAM 不适合于有大量查询和修改并存的情况，那样查询进程会长时间阻塞。因为 MyISAM 是锁表，所以某项读操作比较耗时会使其他写进程饿死。

4）全文索引

MyISAM：支持(FULLTEXT 类型的)全文索引
InnoDB：不支持(FULLTEXT 类型的)全文索引

5）表主键

MyISAM：允许没有任何索引和主键的表存在，索引都是保存行的地址。
InnoDB：如果没有设定主键或者非空唯一索引，就会自动生成一个 6 字节的主键(用户不可见)，数据是主索引的一部分，附加索引保存的是主索引的值。InnoDB 的主键范围更大，最大是 MyISAM 的 2 倍

6）外键

MyISAM：不支持
InnoDB：支持

1）MyISAM 管理非事务表。它提供高速存储和检索，以及全文搜索能力。如果应用中需要执行大量的 SELECT 查询，那么 MyISAM 是更好的选择。
2）InnoDB 用于事务处理应用程序，具有众多特性，包括 ACID 事务支持。如果应用中需要执行大量的 INSERT 或 UPDATE 操作，则应该使用 InnoDB，这样可以提高多用户并发操作的性能。

3. JAVA WEB

一、Servlet 是什么

答：Servlet（Server Applet）是 Java Servlet 的简称，称为小服务程序或服务连接器，用 Java 编写的服务器端程序，主要功能在于交互式地浏览和修改数据，生成动态 Web 内容。

二、Serlvet 工作原理

答：首先客户发送个请求，Servlet 容器会创建特定于这个请求的 ServletRequest 对象和 ServletResponse 对象然后调用 Servlet 的 service()方法。Service()方法从 ServletRequest 对象获得客户请求信息，处理该请求，然后通过 ServletResponse 对象向客户返回响应信息。

三、Servlet 生命周期

1，初始化阶段 调用 init()方法

Servlet 容器创建一个 Servlet 实例并且调用 Servlet 的 init()方法进行初始化。在 Servlet 的整个生命周期内，init()方法只被调用一次。

2，响应客户请求阶段 调用 service()方法

对于用户到达 Servlet 的请求，Servlet 容器会创建特定于这个请求的 ServletRequest 对象和 ServletResponse 对象，然后调用 Servlet 的 service 方法。service 方法从 ServletRequest 对象获得客户请求信息，处理该请求，并通过 ServletResponse 对象向客户返回响应信息。

3，终止阶段 调用 destroy()方法

当 WEB 应用被终止，或 Servlet 容器终止运行，或 Servlet 容器重新装载 Servlet 新实例时，Servlet 容器会先调用 Servlet 的 destroy()方法，在 destroy()方法中可以释放掉 Servlet 所占用的资源

四、Servlet 什么时候创建

1，默认情况下，当 WEB 客户第一次请求访问某个 Servlet 的时候，WEB 容器将创建这个 Servlet 的实例。

2，当 web.xml 文件中如果<servlet>元素中指定了<load-on-startup>子元素时，Servlet 容器在启动 web 服务器时，将按照顺序创建并初始化 Servlet 对象。

五、servlet 与 jsp 的区别和联系

1.jsp 经编译后就变成了 Servlet.

2.JSP 的本质就是 Servlet，JVM 只能识别 java 的类，不能识别 JSP 的代码,

3.Web 容器将 JSp 的代码编译成 JVM 能够识别 java 类

4.jsp 更擅长表现于页面显示,servlet 更擅长于逻辑控制.

5.JSP 侧重于视图，Servlet 主要用于控制逻辑

六、Session 与 cookie 的区别

一、对于 cookie：

①cookie 是创建于服务器端
②cookie 保存在浏览器端
③cookie 的生命周期可以通过 cookie.setMaxAge(2000);来设置，如果没有设置 setMaxAge，则 cookie 的生命周期当浏览器关闭的时候，就消亡了

①存在的位置：

cookie 存在于客户端，临时文件夹中
session：存在于服务器的内存中，一个 session 域对象为一个用户浏览器服务

②安全性

cookie 是以明文的方式存放在客户端的，安全性低，可以通过一个加密算法进行加密后存放
session 存放于服务器的内存中，所以安全性好

③网络传输量

cookie 会传递消息给服务器，session 本身存放于服务器，不会有传送流量

④生命周期(以 30 分钟为例)

(1)cookie 的生命周期是累计的，从创建时，就开始计时，30 分钟后，cookie 生命周期结束，

(2)session 的生命周期是间隔的，从创建时，开始计时如在 30 分钟，没有访问 session，那么 session 生命周期被销毁

但是，如果在 30 分钟内（如在第 29 分钟时）访问过 session，那么，将重新计算 session 的生命周期

(3) 关机会造成 session 生命周期的结束，但是对 cookie 没有影响

七、转发于重定向的区别

1.转发只访问服务器一次。重定向访问服务器两次

2.转发地址栏没有变化；重定向地址栏有变化

3.转发在服务器端完成的；重定向是在客户端完成的

4.转发不会执行转发后的代码；重定向会执行重定向之后的代码

5 转发必须是在同一台服务器下完成；重定向可以在不同的服务器下完成

八、jQuery 选择器

1.基本选择器

基本选择器是 JQuery 最常用的选择器，也是最简单的选择器，它通过元素 id、class 和标签名来查找 DOM 元素

2.层级选择器

如果想通过 DOM 元素之间的层次关系来获取特定元素，例如后代元素，子元素，相邻元素，兄弟元素等，则需要使用层次选择器。

3.过滤选择器

找到一堆页面元素，我们可以对这些元素加过滤条件，找到我们想要的这些元素，然后进行过滤。通过特定的过滤规则来筛选出所需的 DOM 元素，该选择器都以 ":" 开头。

按照不同的过滤规则，过滤选择器可以分为基本过滤，内容过滤，可见性过滤，属性过滤，子元素过滤和表单对象属性过滤选择器。

参考博客：https://blog.csdn.net/pseudonym_/article/details/76093261

九、Ajax

AJAX 全称为“Asynchronous JavaScript and XML”（异步 JavaScript 和 XML），是一种创建交互式网页应用的网页开发技术

十、JDBC

1．Jdbc 是什么

（1）JDBC（Java DataBase Connectivity,java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准，使数据库开发人员能够编写数据库应用程序

2．jdbc 连接数据库步骤

1，使用 jdbc 编程需要连接数据库，注册驱动和数据库信息

2，操作 Connection，打开 Statement 对象

3，通过 Statement 对象执行 SQL，返回结果到 ResultSet 对象

4，使用 ResultSet 读取数据，然后通过代码转化为具体的实体对象

5，关闭数据库相关的资源】

3.Jdbc 缺点

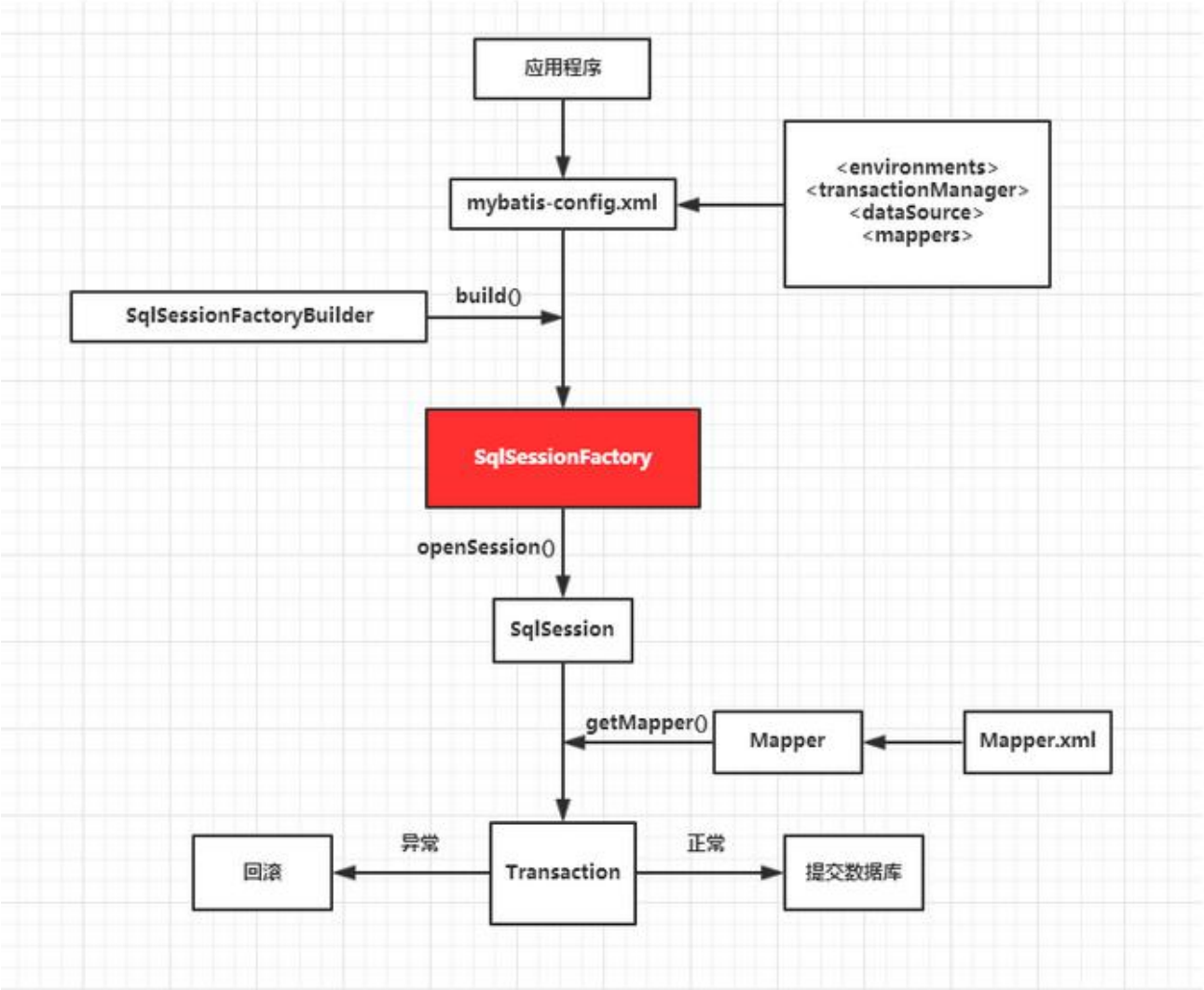
工作量比较大，需要连接，然后处理 jdbc 底层事务，处理数据类型，还需要操作 Connection，Statement 对象和 ResultSet 对象去拿数据并关闭他们

4.Jdbc 优点

执行效率要比 hibernate mybatis 要快，hibernate ，mybatis 框架都是对 jdbc 的封装

4. 框架

一、Mybatis 工作原理



1.加载 mybatis 全局配置文件（数据源、mapper 映射文件等），解析配置文件，MyBatis 基于 XML 配置文件生成 Configuration，和一个个 MappedStatement（包括了参数映射配置、动态 SQL 语句、结果映射配置），其对应着<select | update | delete | insert>标签项。

2、SqlSessionFactoryBuilder 通过 Configuration 对象生成 SqlSessionFactory，用来开启 SqlSession。

3、SqlSession 对象完成和数据库的交互：

4、用户程序调用 mybatis 接口层 api（即 Mapper 接口中的方法）

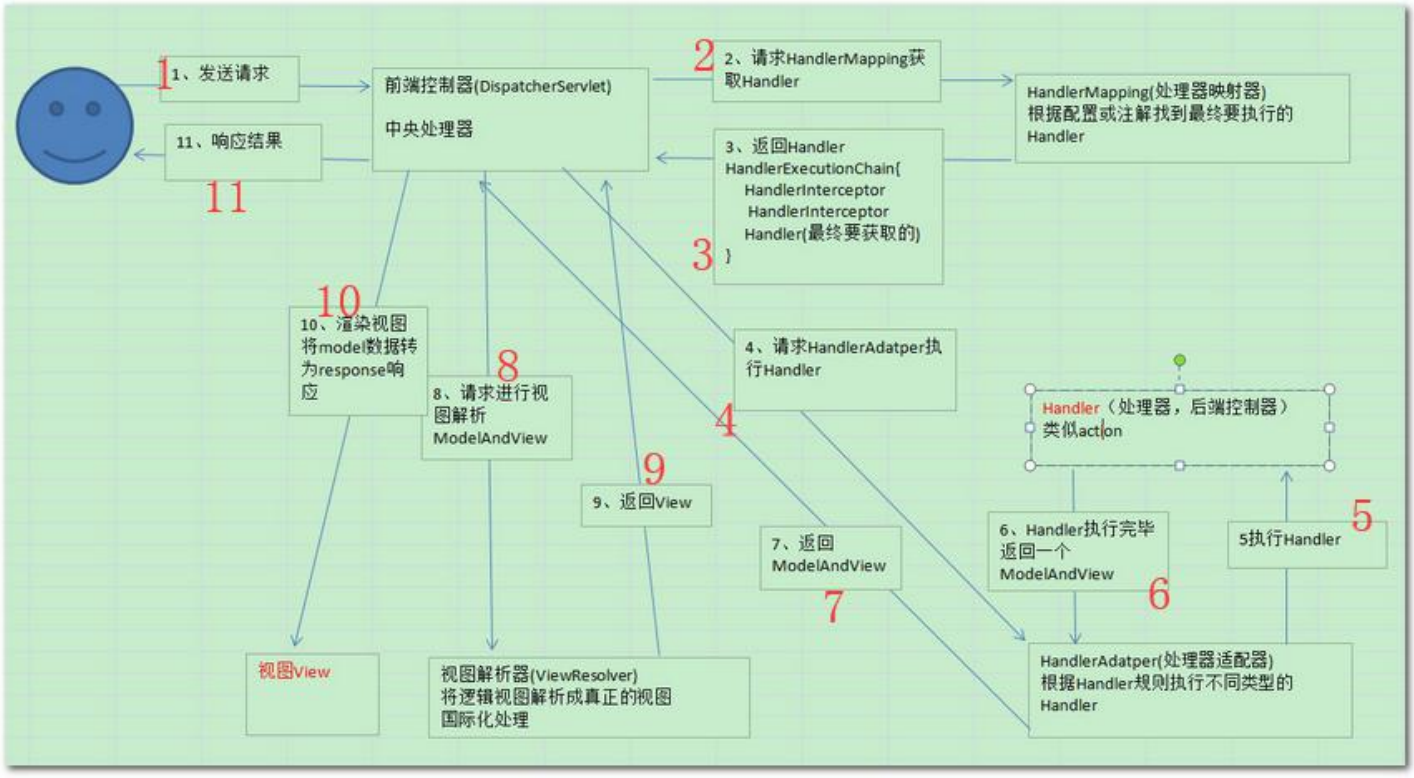
5、SqlSession 通过调用 api 的 Statement ID 找到对应的 MappedStatement 对象

6、通过 Executor（负责动态 SQL 的生成和查询缓存的维护）将 MappedStatement 对象进行解析，sql 参数转化、动态 sql 拼接，生成 jdbc Statement 对象

7、JDBC 执行 sql。

8、借助 MappedStatement 中的结果映射关系，将返回结果转化成 HashMap、JavaBean 等存储结构并返回。

二、springmvc 工作流程



1、 用户发送请求至前端控制器 DispatcherServlet。

2、 DispatcherServlet 收到请求调用 HandlerMapping 处理器映射器。

3、 处理器映射器找到具体的处理器(可以根据 xml 配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给 DispatcherServlet。

4、 DispatcherServlet 调用 HandlerAdapter 处理器适配器。

5、 HandlerAdapter 经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、 Controller 执行完成返回 ModelAndView。

7、 HandlerAdapter 将 controller 执行结果 ModelAndView 返回给 DispatcherServlet。

8、 DispatcherServlet 将 ModelAndView 传给 ViewReslover 视图解析器。

9、 ViewReslover 解析后返回具体 View。

10、DispatcherServlet 根据 View 进行渲染视图（即将模型数据填充至视图中）。

11、 DispatcherServlet 响应用户。

DispatcherServlet：作为前端控制器，整个流程控制的中心，控制其它组件执行，统一调度，降低组件之间的耦合性，提高每个组件的扩展性。

HandlerMapping：通过扩展处理器映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

HandlAdapter：通过扩展处理器适配器，支持更多类型的处理器。

ViewResolver：通过扩展视图解析器，支持更多类型的视图解析，例如：jsp、freemarker、pdf、excel 等。

三、Spring ioc/aop

IoC（控制反转）,将类的创建和依赖关系写在配置文件里，由配置文件注入，实现了松耦合.底层使用了工厂模式、 单例模式和反射

AOP（面向切面编程） 将安全，事务等于程序逻辑相对独立的功能抽取出来，利用 spring 的配置文件将这些功能插进去，实现了按照方面编程，提高了复用性.底层使用了代理模式

四、Hibernate

1．工作原理

（1）首先，应用程序先调用 Configration 类，该类读取 hibernate 的配置文件及映射文件中的信息（如果我们使用注解的话则不需要读取映射文件），并使用这些信息生成一个 SessionFactory 对象。接着，从 SessionFacctory 生成一个 Session 对象，并用 Session 对象生成 Transaction 对象。通过 Session 对象的进行 CRUD 等方法对 C 对象进行加载，保存，更新删除等操作

2．三种状态

transient(瞬时)

刚 new 出来一个对象 obj，obj 在内存孤立存在,obj 是携带信息的载体,obj 没有被保存到数据库中(换句话：obj 没有被 session 进行持久化

persistent(持久化)

该状态的对象在数据库中具有对应的记录,并拥有一个持久化标识

detached(游离)

游离对象拥有数据库的识别值,但已不在持久化管理范围之内（session 中不存在该对象）

3．优点

（0）它使用时只需要操纵对象，使开发更对象化，抛弃了数据库中心的思想，完全的面向对象思想

（1）移植性会很好，缓存机制。提供一级缓存和二级缓存

（2）开发效率高。hibernate 提供了大量的封装，很多数据操作以及关联关系等都被封装的很好，开发者不需写大量的 sql 语句，这就极大的提高了开发者的开发效率

4．缺点

（1）由于对持久层封装过于完整，导致开发人员无法对 SQL 进行优化，无法灵活使用 JDBC 的原生 SQL，Hibernate 封装了 JDBC，所以没有 JDBC 直接访问数据库效率高。要使用数据库的特定优化机制的时候，不适合用 Hibernate

五、Spring 的常用注解

@Component:用于创建对象的,作用于类。

@Controller： 一般用于表现层的注解。

@Service： 一般用于业务层的注解。

@Repository： 一般用于持久层的注解。

@Bean 用于把当前方法的返回值作为 bean 对象存入 spring 的 ioc 容器中

@ComponentScan 用于指定 spring 在初始化容器时要扫描的包

@Autowired:自动按照类型注入

@Scope:作用：指定 bean 的作用范围。

5. 分布式/微服务

一、Nginx 是什么

是一个高性能的反向代理服务器能够出色的完成动静分离负载均衡。

二、Nginx 轮询方式

Ip_hash:

采用 ip_hash 的轮询方法,每个 ip 在一定时间内会被固定的后端服务器,这样我们不用解决 session 共享问题

Url_hash

同一个 url（也就是同一个资源请求）会到达同一台机器

least_conn(最小连接)

算法很简单，首选遍历后端集群，比较每个后端的连接数，选取该值最小的后端。如果有多个后端的连接数值同为最小的，那么对它们采用加权轮询算法。

轮询（默认）

轮询算法是把请求平均的转发给各个后端，使它们的负载大致相同。当然也可以设置 weight 值 weight 越大权重越大

三、集群如何做 session 共享

集群 session 共享问题：

session 存在 memcache 或者 redis 中以这种方式来同步 session，不会加大数据库的负担，并且安全性比用 cookie 大大的提高，把 session 放到内存里面，比从文件中读取要快很多。

四、Redis 是什么

Redis[Remote Dictionary Server(远程字典服务器)]

作用主要用来做数据缓存，可以用来做分布式锁，排行榜，计数器，消息队列等。

五、Redis 有哪些数据类型及使用场景

字符串（strings），

string——适合最简单的 k-v 存储，类似于 memcached 的存储结构，短信验证码，配置信息等，就用这种类型来存储。

散列（hashes）， 适用于：存储、读取、修改用户属性。

列表（lists），适用于：消息队列。

集合（sets），在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。Redis 还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能，对上面的所有集合操作，你还可以使用不同的命令选择将结果返回给客户端还是存集到一个新的集合中。

有序集合（sorted sets）：适用于：排行榜

六、Redis 持久化

有 rdb(redis database) 和 aof(append only file)默认开启 rdb，如果两者同时开用 aof

RDB 是 Redis 默认的持久化方案。在指定的时间间隔内，执行指定次数的写操作，则会将内存中的数据写入到磁盘中。即在指定目录下生成一个 dump.rdb 文件。Redis 重启会通过加载 dump.rdb 文件恢复数据。

RDB 优点：

1 适合大规模的数据恢复。

2 如果业务对数据完整性和一致性要求不高，RDB 是很好的选择。

rdb 缺点：

1 数据的完整性和一致性不高，因为 RDB 可能在最后一次备份时宕机了。

2 备份时占用内存，因为 Redis 在备份时会独立创建一个子进程，将数据写入到一个临时文件（此时内存中的数据是原来的两倍哦），最后再将临时文件替换之前的备份文件

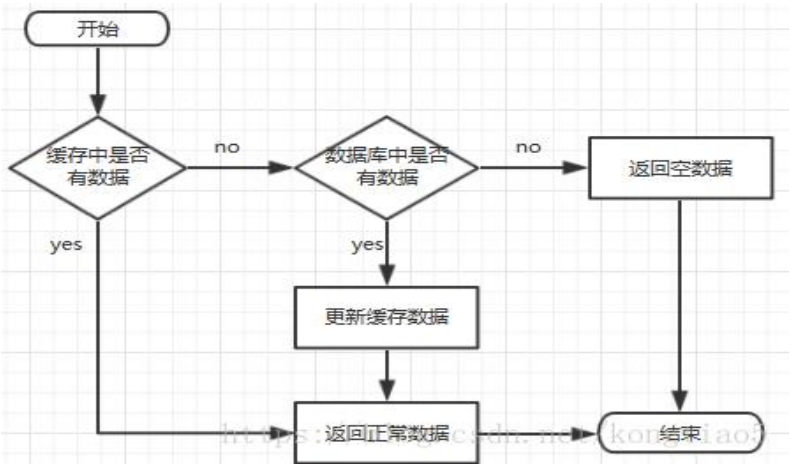
AOF ：Redis 默认不开启。它的出现是为了弥补 RDB 的不足（数据的不一致性），所以它采用日志的形式来记录每个写操作，并追加到文件中。Redis 重启的会根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。

AOF 优点：数据的完整性和一致性更高

AOF 缺点：因为 AOF 记录的内容多，文件会越来越大，数据恢复也会越来越慢。

七、Redis 缓存处理流程

前台请求，后台先从缓存中取数据，取到直接返回结果，取不到时从数据库中取，数据库取到更新缓存，并返回结果，数据库也没取到，那直接返回空结果。



八、Redis 缓存穿透

描述： 缓存穿透是指缓存和数据库中都没有的数据，而用户不断发起请求，如发起为 id 为“-1”的数据或 id 为特别大不存在的数据。这时的用户很可能是攻击者，攻击会导致数据库压力过大。

解决方案： 接口层增加校验，如用户鉴权校验，id 做基础校验，id<=0 的直接拦截；

从缓存取不到的数据，在数据库中也没有取到，这时也可以将 key-value 对写为 key-null，缓存有效时间可以设置短点，如 30 秒（设置太长会导致正常情况也没法使用）。这样可以防止攻击用户反复用同一个 id 暴力攻击

九、Redis 缓存雪崩

描述： 缓存雪崩是指缓存中数据大批量到过期时间，而查询数据量巨大，引起数据库压力过大甚至 down 机。和缓存击穿不同的是， 缓存击穿指并发查同一条数据，缓存雪崩是不同数据都过期了，很多数据都查不到从而查数据库。

解决方案： 缓存数据的过期时间设置随机，防止同一时间大量数据过期现象发生。

如果缓存数据库是分布式部署，将热点数据均匀分布在不同搞得缓存数据库中。 设置热点数据永远不过期。

十、Redis 缓存击穿

描述： 缓存击穿是指缓存中没有但数据库中有的数据（一般是缓存时间到期），这时由于并发用户特别多，同时读缓存没读到数据，又同时去数据库去取数据，引起数据库压力瞬间增大，造成过大压力

解决方案：设置热点数据永远不过期。加互加互斥锁

十一、RabbitMQ 是什么

答：RabbitMQ 是一个开元的消息代理和队列服务器，用来通过普通协议在完全不同的应用之间共享数据，RabbitMQ 是使用 Erlang 语言来编写的。

ConnectionFactory（连接管理器）：应用程序与 Rabbit 之间建立连接的管理器，程序代码中使用；

Channel（信道）：消息推送使用的通道；

Exchange（交换器）：用于接受、分配消息；

Queue（队列）：用于存储生产者的消息；

RoutingKey（路由键）：用于把生成者的数据分配到交换器上；

BindingKey（绑定键）：用于把交换器的消息绑定到队列上；

常用的 MQ 还有：activeMQ,zeroMQ ,kafka

十二、RabbitMQ 消息持久化

消息持久化

Rabbit 队列默认情况下重启服务器会导致消息丢失，那么怎么保证 Rabbit 在重启的时候不丢失呢？答案就是消息持久化。

设置消息持久化必须先设置队列持久化，要不然队列不持久化，消息持久化，队列都不存在了，消息存在还有什么意义。消息持久化需要将交换机持久化、队列持久化、消息持久化，才能最终达到持久化的目的。

持久化的缺点

消息持久化的优点显而易见，但缺点也很明显，那就是性能，因为要写入硬盘要比写入内存性能较低很多，从而降低了服务器的吞吐量，

十三、RabbitMQ vhost

答：每个 Rabbit 都能创建很多 vhost，我们称之为虚拟主机，每个虚拟主机其实都是 mini 版的 RabbitMQ，拥有自己的队列，交换器和绑定，拥有自己的权限机制。

十四、如何确保消息正确地发送至 RabbitMQ?

RabbitMQ 使用发送方确认模式，确保消息正确地发送到 RabbitMQ。

发送方确认模式：将信道设置成 confirm 模式（发送方确认模式），则所有在信道上发布的消息都会被指派一个唯一的 ID。一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一 ID）。如果 RabbitMQ 发生内部错误从而导致消息丢失，会发送一条 nack（not acknowledged，未确认）消息。

十五、如何防止生产者弄丢数据

答：如果要确保说写 rabbitmq 的消息别丢，可以开启 confirm 模式，在生产者那里设置开启 confirm 模式之后，你每次写的消息都会分配一个唯一的 id，然后如果写入了 rabbitmq 中，rabbitmq 会给你回传一个 ack 消息，告诉你说这个消息 ok 了。如果 rabbitmq 没能处理这个消息，会回调你一个 nack 接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息 id 的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。

补充：也可以用事务机制 但事务机制和 cnofirm 机制最大的不同在于，事务机制是同步的，你提交一个事务之后会阻塞在那儿，但是 confirm 机制是异步的，你发送个消息之后就可以发送下一个消息，然后那个消息 rabbitmq 接收了之后会异步回调你一个接口通知你这个消息接收到了。

十六、如何防止 Rabbit 弄丢数据

这个必须开启 rabbitmq 的持久化，就是消息写入之后会持久化到磁盘，哪怕是 rabbitmq 自己挂了，恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是，rabbitmq 还没持久化，自己就挂了，可能导致少量数据会丢失的，但是这个概率较小。

设置持久化有两个步骤，第一个是创建 queue 的时候将其设置为持久化的，这样就可以保证 rabbitmq 持久化 queue 的元数据，但是不会持久化 queue 里的数据；第二个是发送消息的时候将消息的 deliveryMode 设置为 2，就是将消息设置为持久化的，此时 rabbitmq 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行，rabbitmq 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 里的数据。

而且持久化可以跟生产者那边的 confirm 机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者 ack 了，所以哪怕是在持久化到磁盘之前，rabbitmq 挂了，数据丢了，生产者收不到 ack，你也是可以自己重发的。

十七、如何避免消息重复投递或重复消费？

在消息生产时，MQ 内部针对每条生产者发送的消息生成一个 id，作为去重和幂等的依据（消息投递失败并重传），避免重复的消息进入队列；在消息消费时，要求消息体中必须要有一个 id（对于同一业务全局唯一，如支付 ID、订单 ID、帖子 ID 等）作为去重和幂等的依据，避免同一条消息被重复消费。

十八、RabbitMQ 常用交换器

direct：如果路由键完全匹配，消息就被投递到相应的队列

fanout：如果交换器收到消息，将会广播到所有绑定的队列上

topic：通配交换器

十九、Mycat 是什么

答：MyCat 是一个开源的分布式数据库系统，是一个实现了 MySQL 协议的服务器，前端用户可以把它看作是一个数据库代理，用 MySQL 客户端工具和命令行访问，而其后端可以用 MySQL 原生协议与多个 MySQL 服务器通信，也可以用 JDBC 协议与大多数主流数据库服务器通信，其核心功能是分表分库。

二十、Mycat 有哪些功能和特性

支持 JDBC 连接多数据库

支持 NoSQL 数据库

自动故障切换，高可用性

支持读写分离，支持 Mysql 双主多从，以及一主多从的模式

支持自增长主键、支持 Oracle 的序列 机制

二十一、Lucene 是什么

Lucene 是 apache 软件基金会一个开放源代码的全文检索引擎工具包，是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎。

二十二、什么是 ElasticSearch?

Elasticsearch 是一个基于 Lucene 的搜索引擎。它提供了具有 HTTP Web 界面和无架构 JSON 文档的分布式，多租户能力的全文搜索引擎。Elasticsearch 是用 Java 开发的，根据 Apache 许可条款作为开源发布。

二十三、ElasticSearch 的倒排索引是什么

传统的我们的检索是通过文章，逐个遍历找到对应关键词的位置。

而倒排索引，是通过分词策略，形成了词和文章的映射关系表，这种词典+映射表即为倒排索引

二十四、ElasticSearch 索引 类型 文档 字段是什么

```
Relational DB -> Databases -> Tables -> Rows -> Columns
Elasticsearch -> Indices   -> Types  -> Documents -> Fields
```

索引：相当于关系型数据库

类型：相当于表

文档：相当于表中的一行

字段：相当行中的列

二十五、Springboot 是什么

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

二十六、Springboot 有什么优点

敏捷开发
嵌入式 Tomcat，Jetty 容器，
无需部署 WAR 包
支持热部署
提供大量 starter 简化 Maven 配置
对主流开发框架的无配置集成

二十七、Springboot 与 Springmvc 的区别

spring boot 就是一个大框架里面包含了许许多多的东西，其中 spring 就是最核心的内容之一，包含 spring mvc。

spring mvc 是只是 spring 处理 web 层请求的一个模块。

二十八、Springboot 常用 starter

```
spring-boot-starter-data-jpa
spring-boot-starter-data-mongodb
spring-boot-starter-amqp
spring-boot-starter-data-elasticsearch
spring-boot-starter-web
```

二十九、什么是微服务架构

微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相协作（通常是基于 HTTP 协议的 RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境。

三十、微服务架构有什么优势和劣势

微服务架构的优点：

每个服务都比较简单，只关注于一个业务功能。

微服务架构方式是松耦合的，可以提供更高的灵活性。

微服务可通过最佳及最合适的不同的编程语言与工具进行开发

每个微服务可由不同团队独立开发，互不影响，加快推出市场的速度。

微服务架构的缺点：

运维工作量增加，应用运维管理复杂。
服务器内存占用量更高
事务处理麻烦

三十一、Springcloud 是什么

SpringCloud=分布式微服务架构下的一站式解决方案，是各个微服务架构落地技术的集合体，俗称微服务全家桶

三十二、Springcloud 与 SpringBoot 有什么区别

SpringBoot 可以离开 SpringCloud 独立使用开发项目，但是 SpringCloud 离不开 SpringBoot，属于依赖的关系.

SpringBoot 专注于快速、方便的开发单个微服务个体，SpringCloud 关注全局的服务治理框架。

三十三、Eureka（注册发现）

Eureka 是 Netflix 开发的服务注册发现框架，本身是一个基于 REST 的服务。

Eureka 包含两个组件：Eureka Server 和 Eureka Client。

Eureka Server 提供服务注册服务，各个节点启动后，会在 Eureka Server 中进行注册，这样 EurekaServer 中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。

Eureka Client 是一个 java 客户端，用于简化与 Eureka Server 的交互，客户端同时也就是一个内置的、使用轮询(round-robin)负载算法的负载均衡器。

三十四、Spring Cloud Ribbon（负载均衡）

- Spring Cloud Ribbon是一个基于HTTP和TCP的客户端负载均衡工具，它基于Netflix Ribbon实现。通过Spring Cloud 的封装，可以让我们轻松地将面向服务的 REST 模版请求自动转换成客户端负载均衡的服务调用。Spring Cloud Ribbon 虽然只是一个工具类框架，它不像服务注册中心、配置中心、API 网关那样需要独立部署

三十五、Spring Cloud Feign（负载均衡）

Feign 是一种负载均衡的 HTTP 客户端，使用 Feign 调用 API 就像调用本地方法一样

Feign 集成了 Ribbon

三十六、Spring Cloud hystrix（断路器）

Hystrix 是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix 能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性.

三十七、Spring Cloud Config（配置中心）

分布式系统中，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在 Spring Cloud 中，有分布式配置中心组件 springCloud Config ，它支持从远程 Git 仓库中读取配置文件并存放到本地 Git 仓库

三十八、SpringCloud zuul（服务网关）

Zuul 包含了对请求的路由和过滤两个最主要的功能：其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础。

三十九、Dubbo 是什么

Dubbo 是一款高性能、轻量级的开源 Java RPC 框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

四十、Dubbo 有哪些注册中心

Multicast 注册中心

Zookeeper 注册中心

Nacos 注册中心

Redis 注册中心

Simple 注册中心

官网推荐使用 zookeeper

四十一、Springcloud 与 dubbo 的区别

SpringCloud 是一套目前比较网站微服务框架了，整合了分布式常用解决方案遇到了问题注册中心 Eureka、负载均衡器 Ribbon ，客户端调用工具 Rest 和 Feign，分布式配置中心 Config，服务保护 Hystrix，网关 Zuul ，消息总线 Bus 等。

Dubbo 内部实现功能没有 SpringCloud 强大（全家桶），只是实现服务治理，缺少分布式配置中心、网关、链路、总线等，如果需要用到这些组件，需要整合其他框架。

最大区别：SpringCloud 抛弃了 Dubbo 的 RPC 通信，采用的是基于 HTTP 的 REST 方式。

四十二、Eureka 与 zookeeper 的区别

Zookeeper 保证 CP:当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接 down 掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是 zk 会出现这样一种情况，当 master 节点因为网络故障与其他节点失去联系时，剩余节点会重新进行 leader 选举。问题在于，选举 leader 的时间太长，30 ~ 120s, 且选举期间整个 zk 集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得 zk 集群失去 master 节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

Eureka 保存 AP:Eureka 各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而 Eureka 的客户端在向某个 Eureka 注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台 Eureka 还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka 还有一种自我保护机制，如果在 15 分钟内超过 85%的节点都没有正常的心跳，那么 Eureka 就认为客户端与注册中心出现了网络故障

四十三、分布式 CAP 定理是什么

CAP 理论告诉我们，一个分布式系统不可能同时满足一致性（C:Consistency），可用性（A: Availability）和分区容错性（P：Partition tolerance）这三个基本需求，最多只能同时满足其中的 2 个。

选项	描述
C（Consistence）	**一致性**，指数据在多个副本之间能够保持一致的特性（严格的一致性）。
A（Availability）	**可用性**，指系统提供的服务必须一直处于可用的状态，每次请求都能获取到非错的响应——但是不保证获取的数据为最新数据。
P（Network partitioning	**分区容错性**，分布式系统在遇到任何网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务，除非整个网络环境都发生了故障。

组合	分析结果
CA	满足原子和可用，放弃分区容错。说白了，就是一个整体的应用。
CP	满足原子和分区容错，也就是说，要放弃可用。当系统被分区，为了保证原子性，必须放弃可用性，让服务停用。
AP	满足可用性和分区容错，当出现分区，同时为了保证可用性，必须让节点继续对外服务，这样必然导致失去原子性。

四十四、三高 三 V 是什么

大数据时代的 3V：

1. 海量 Volume　多样 Variety　实时 Velocity

互联网需求的 3 高：

1. 高并发　高可拓　高性能

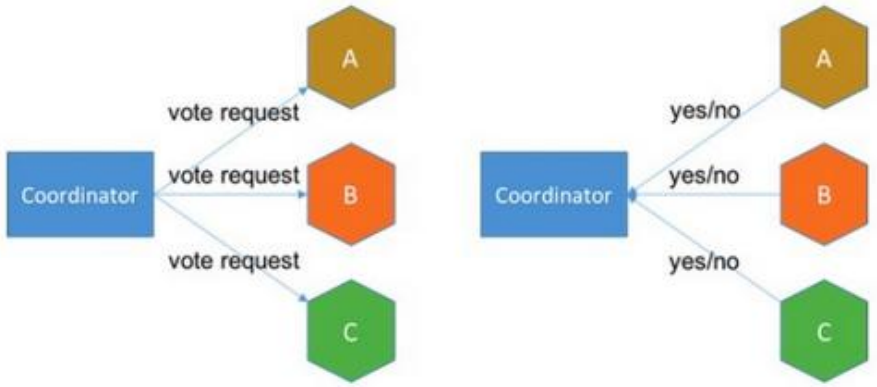
四十五、如何解决分布式事务-2PC

一、两阶段提交（2PC）

两阶段提交（Two-phase Commit，2PC），通过引入协调者（Coordinator）来协调参与者的行为，并最终决定这些参与者是否要真正执行事务

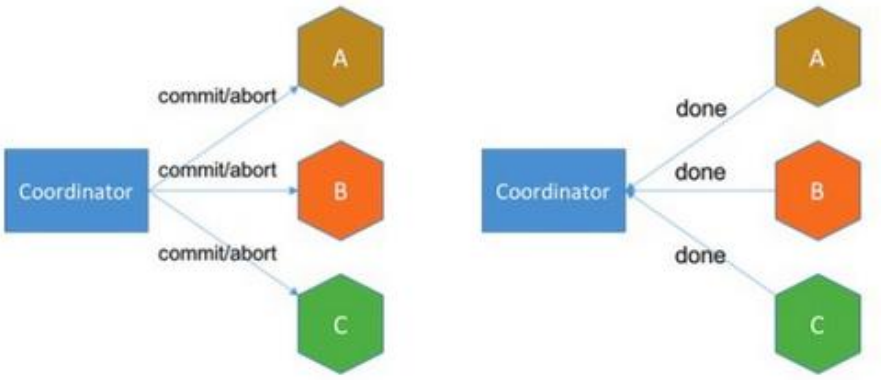
1.1 准备阶段

协调者询问参与者事务是否执行成功，参与者发回事务执行结果。



1.2 提交阶段

如果事务在每个参与者上都执行成功，事务协调者发送通知让参与者提交事务；否则，协调者发送通知让参与者回滚事务。



需要注意的是，在准备阶段，参与者执行了事务，但是还未提交。只有在提交阶段接收到协调者发来的通知后，才进行提交或者回滚。

存在的问题

2.1 同步阻塞 所有事务参与者在等待其它参与者响应的时候都处于同步阻塞状态，无法进行其它操作。

2.2 单点问题 协调者在 2PC 中起到非常大的作用，发生故障将会造成很大影响。特别是在阶段二发生故障，所有参与者会一直等待状态，无法完成其它操作。

2.3 数据不一致 在阶段二，如果协调者只发送了部分 Commit 消息，此时网络发生异常，那么只有部分参与者接收到 Commit 消息，也就是说只有部分参与者提交了事务，使得系统数据不一致。

四十六、如何解决分布式事务-TCC

TCC 其实就是采用的补偿机制，其核心思想是：针对每个操作，都要注册一个与其对应的确认和补偿（撤销）操作。它分为三个阶段：

Try 阶段主要是对业务系统做检测及资源预留

Confirm 阶段主要是对业务系统做确认提交，Try 阶段执行成功并开始执行 Confirm 阶段时，默认 Confirm 阶段是不会出错的。即：只要 Try 成功，Confirm 一定成功。

Cancel 阶段主要是在业务执行错误，需要回滚的状态下执行的业务取消，预留资源释放。

举个例子，假入 Bob 要向 Smith 转账，思路大概是：

我们有一个本地方法，里面依次调用

1、首先在 Try 阶段，要先调用远程接口把 Smith 和 Bob 的钱给冻结起来。

2、在 Confirm 阶段，执行远程调用的转账的操作，转账成功进行解冻。

3、如果第 2 步执行成功，那么转账成功，如果第二步执行失败，则调用远程冻结接口对应的解冻方法 (Cancel)。

优点： 跟 2PC 比起来，实现以及流程相对简单了一些，但数据的一致性比 2PC 也要差一些

缺点： 缺点还是比较明显的，在 2,3 步中都有可能失败。TCC 属于应用层的一种补偿方式，所以需要程序员在实现的时候多写很多补偿的代码，在一些场景中，一些业务流程可能用 TCC 不太好定义及处理。

四十七、如何解决分布式事务-本地事务表(重点)

本地消息表这种实现方式应该是业界使用最多的，其核心思想是将分布式事务拆分成本地事务进行处理，这种思路是来源于 ebay。我们可以从下面的流程图中看出其中的一些细节：

基本思路就是：

消息生产方，需要额外建一个消息表，并记录消息发送状态。消息表和业务数据要在一个事务里提交，也就是说他们要在一个数据库里面。然后消息会经过 MQ 发送到消息的消费方。如果消息发送失败，会进行重试发送。

消息消费方，需要处理这个消息，并完成自己的业务逻辑。此时如果本地事务处理成功，表明已经处理成功了，如果处理失败，那么就会重试执行。如果是业务上面的失败，可以给生产方发送一个业务补偿消息，通知生产方进行回滚等操作。

生产方和消费方定时扫描本地消息表，把还没处理完成的消息或者失败的消息再发送一遍。如果有靠谱的自动对账补账逻辑，这种方案还是非常实用的。

这种方案遵循 BASE 理论，采用的是最终一致性，笔者认为是这几种方案里面比较适合实际业务场景的，即不会出现像 2PC 那样复杂的实现(当调用链很长的时候，2PC 的可用性是非常低的)，也不会像 TCC 那样可能出现确认或者回滚不了的情况。

优点： 一种非常经典的实现，避免了分布式事务，实现了最终一致性。在 .NET 中 有现成的解决方案。

缺点： 消息表会耦合到业务系统中，如果没有封装好的解决方案，会有很多杂活需要处理。

四十八、如何设计一个秒杀系统

秒杀系统的整体架构可以概括为“稳、准、快”几个关键字。

所谓“稳”，就是整个系统架构要满足高可用，流量符合预期时肯定要稳定，超出预期时也同样不能掉链子，你要保证秒杀活动顺利完成，即秒杀商品顺利地卖出去，这个是最基本的前提。

然后就是“准”，你的业务需求是秒杀 10 台 iPhone XS，那就只能成交 10 台，多一台少一台都不行。一旦库存不对，那平台就要承担损失，所以“准”就是要求保证数据的一致性。

最后再看“快”，“快”其实很好理解，它就是说系统的性能要足够高，否则你怎么支撑这么大的流量呢？不光是服务端要做极致的性能优化，而且在整个请求链路上都要做协同的优化，每个地方快一点，整个系统就完美了。

所以，一般优化设计思路：将请求拦截在系统上游，降低下游压力。
在一个并发量大，实际需求小的系统中，应当尽量在前端拦截无效流量，降低下游服务器和数据库的压力，不然很可能造成数据库读写锁冲突，甚至导致死锁，最终请求超时。

限流：前端直接限流，允许少部分流量流向后端。

削峰：瞬时大流量峰值容易压垮系统，解决这个问题是重中之重。常用的消峰方法有异步处理、缓存和消息中间件等技术。

异步处理：秒杀系统是一个高并发系统，采用异步处理模式可以极大地提高系统并发量，其实异步处理就是削峰的一种实现方式。

内存缓存：秒杀系统最大的瓶颈一般都是数据库读写，由于数据库读写属于磁盘 IO，性能很低，如果能够把部分数据或业务逻辑转移到内存缓存，效率会有极大地提升。

消息队列：消息队列可以削峰，将拦截大量并发请求，这也是一个异步处理过程，后台业务根据自己的处理能力，从消息队列中主动的拉取请求消息进行业务处理。

可拓展：当然如果我们想支持更多用户，更大的并发，最好就将系统设计成弹性可拓展的，如果流量来了，拓展机器就好了，像淘宝、京东等双十一活动时会临时增加大量机器应对交易高峰。

四十九、Docker 是什么

Docker 作为一个软件集装箱化平台，可以让开发者构建应用程序时，将它与其依赖环境一起打包到一个容器中，然后很容易地发布和应用到任意平台中

docker 有 3 大核心：镜像、容器、仓库

五十、Docker 镜像、容器、仓库的关系

Docker 镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数。类似虚拟机镜像

Docker 容器是镜像的一个运行实例，不同的是它带有额外的可写层。 可认为 docker 容器就是独立运行的一个或一组应用，以及它们所运行的必需环境。

Docker 仓库是用来包含镜像的位置，Docker 提供一个注册服务器（Register）来保存多个仓库，每个仓库又可以包含多个具备不同 tag 的镜像。Docker 运行中使用的默认仓库是 Docker Hub 公共仓库。

五十一、Docker 常用命令

① docker images -q 只显示镜像 id
② docker image -a 列出本地所有镜像（含中间映像层）
③ docker images --digests 显示镜像的摘要信息
④ docker images --no-trunc 显示镜像完整信息
⑤ docker search 镜像名 搜索镜像
⑥ docker pull 镜像名 下载镜像
⑦ docker rmi 镜像 id 删除镜像
⑧ docker ps 列出正在运行 的 容器
⑨ docker run -d -p 8080:8080 --name mycat tomcat 创建容器运行示例
⑩ docker start 容器 ID 或容器名 启动容器
⑪ docker restart 容器 ID 或容器名 重启容器
⑫ docker stop 容器 ID 或容器名 停止容器
⑬ docker kill 容器 ID 或容器名 强制停止容器
⑭ docker rm 容器 ID 删除已停止的容器

五十二、补充

6. 其他

一、Linux 常用命令

chmod 修改或文件目录权限 ps 查看进程
ln 创建链接文件 tar 打包压缩或解压文件
touch 创建文件 mkdir 创建目录
rpm 安装 rpm 文件 cut 截取文件内容
df 检查文件系统的磁盘空间占用情况
rm 删除软件

二、文件-rw-r--r--分别代表什么

r:read 就是读权限 --数字 4 表示

w:write 就是写权限 --数字 2 表示

x:excute 就是执行权限 --数字 1 表示

读、写、运行三项权限可以用数字表示，就是 r=4,w=2,x=1。所以，-rw-r--r--用数字表示成 644。

这里总共会有 10 个“-”，第一个表示文件类型，如该文件是文件(-表示)，文件夹(d 表示),连接文件(l 表示)，后面 9 个按照三个一组分，如:

-rwxrwx--- 770 权限表示此文件(文件夹)的拥有着和同组用户有读写及执行权限，其他用户组没任何权限

就是前面三个表示所有者权限，中间三个表示同组用户权限，最后一组表示其他用户权限

三、线程池的创建方式

newSingleThreadExecutor

创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

newFixedThreadPool

创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

newCachedThreadPool

创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，

那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。

newScheduledThreadPool

创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

四、Spring 中用到的设计模式

工厂方法 单例模式 适配器 代理 观察者 策略 模板方法

五、Redis 如何与 mysql 同步

一、**对强一致时**要求比较高的，应采用实时同步方案，即查询缓存查询不到再从 DB 查询，保存到缓存；

二、**高并发时**读操作和上面一样，写操作是异步写，写入 Redis 后直接返回，然后定期写入 MySQL 。可采用异步队列的方式同步，可采用 kafka 等消息中间件处理消息生产和消费。

spring3+提供了注解的方式进行缓存编程

@Cacheable：查询时使用，注意 Long 类型需转换为 Sting 类型，否则会抛异常

@CachePut：更新时使用，使用此注解，一定会从 DB 上查询数据

@CacheEvict：删除时使用；

六、行锁锁 和 表锁 悲观锁 和 乐观锁

（1）行锁：访问数据库的时候，锁定整个行数据，防止并发错误。开销小，加锁快，不会出现死锁；锁定力度大，发生锁冲突概率高，并发度最低

（2）表锁：访问数据库的时候，锁定整个表数据，防止并发错误。 开销大，加锁慢，**会出现死锁**；锁定粒度小，发生锁冲突的概率低，并发度高

（1）悲观锁：顾名思义，就是很悲观，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 block 直到它拿到锁。**

（2）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

（3）悲观锁 和 乐观锁的区别：

两种锁各有优缺点，不可认为一种好于另一种，像**乐观锁适用于写比较少的情况下**，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但**如果经常产生冲突、所以这种情况下用悲观锁就比较合适**。

七、强引用 软引用 弱引用 幻象引用（虚引用）

强引用就是我们最常见的普通对象引用（如 new 一个对象），只要还有强引用指向一个对象，就表明此对象还“活着”。在强引用面前，即使 JVM 内存空间不足，JVM 宁愿抛出 OutOfMemoryError 运行时错误（OOM），让程序异常终止，也不会靠回收强引用对象来解决内存不足的问题。对于一个普通的对象 ,如果没有其他的引用关系 ,只要超过了引用的作用域或者显式地将相应(强)引用赋值为 null，就意味着此对象可以被垃圾收集了。但要注意的是，并不是赋值为 null 后就立马被垃圾回收，具体的回收时机还是要看垃圾收集策略的。

如 Object obj = new Object();

软引用相对强引用要弱化一些，**可以让对象豁免一些垃圾收集。当内存空间足够的时候，垃圾回收器不会回收它。**只有当 JVM 认定内存空间不足时才会去回收软引用指向的对象。JVM 会确保在抛出 OOM 前清理软引用指向的对象，而且 JVM 是很聪明的，会尽可能优先回收长时间闲置不用的软引用指向的对象，对那些刚构建的或刚使用过的软引用指向的对象尽可能的保留。基于软引用的这些特性，**软引用可以用来实现很多内存敏感点的缓存场景**，即如果内存还有空闲，可以暂时缓存一些业务场景所需的数据，当内存不足时就可以清理掉，等后面再需要时，可以重新获取并再次缓存。这样就确保在使用缓存提升性能的同时，不会导致耗尽内存。

```
SoftReference<List<Foo>> ref = new SoftReference<List<Foo>>(new LinkedList<Foo>());

// somewhere else in your code, you create a Foo that you want to add to the list
List<Foo> list = ref.get();
if (list != null)
{
    list.add(foo);
}
else
{
    // list is gone; do whatever is appropriate
}
```

在使用软引用的时候必须检查引用是否为 null。因为垃圾收集器可能在任意时刻回收软引用，如果不做是否 null 的判断，可能会出现 NullPointerException 的异常

弱引用指向的对象是一种十分临近 finalize 状态的情况，当弱引用被清除的时候，就符合 finalize 的条件了。弱引用与软引用最大的区别就是弱引用比软引用的生命周期更短暂。垃圾回收器会扫描它所管辖的内存区域的过程中，只要发现弱引用的对象，不管内存空间是否有空闲，都会立刻回收它。如同前面我说过的，具体的回收时机还是要看垃圾回收策略的，因此那些弱引用的对象并不是说只要达到弱引用状态就会立马被回收。

基于弱引用的这些特性，弱引用同样可以应用在很多需要缓存的场景。

```
Object obj = new Object();
WeakReference<Object> wf = new WeakReference<Object>(obj);
obj = null;
//有时候会返回null
wf.get();
//返回是否被垃圾回收器标记为即将回收的垃圾
wf.isEnQueued();
```

虚引用/幻象引用并不会决定对象的生命周期。即如果一个对象仅持有虚引用，就相当于没有任何引用一样，在任何时候都可能被垃圾回收器回收。不能通过它访问对象，

```
Object obj = new Object();
PhantomReference<Object> pf = new PhantomReference<Object>(obj);
obj=null;
//永远返回null
pf.get();
//返回是否从内存中已经删除
pf.isEnQueued();
```

幻象引用的 get 方法永远返回 null，主要用于检查对象是否已经从内存中删除。

参考博客：https://www.cnblogs.com/heyonggang/p/10254372.html

八、数据结构

九、常见算法

十、二叉树，平衡二叉树，红黑树

十一、补充

7. 笔试题

一、写出两个单例模式（懒汉 饿汉）

```
public class Singleton {
    private static Singleton instance=null;
    private Singleton(){

    }
    public static Singleton getInstance(){
        if(instance==null){
            synchronized(Singleton.class){
                if(instance==null){
                    instance=new Singleton();
                }
            }
        }
        return instance;
    }
}
```

```
public class Singleton {
    private static Singleton instance=new Singleton();
    private Singleton(){

    }
    public static Singleton getInstance(){
        return instance;
    }
}
```

二、数据库去重（删除重复的数据且保留一条）

```
DELETE FROM  表名

WHERE
    (字段名1, 字段名2) IN (
        SELECT 字段名1, 字段名2 FROM 表名 GROUP BY 字段名1,字段名2 HAVING COUNT(*) > 1
    )
AND rowid NOT IN (
        SELECT MIN(rowid) FROM  表名  GROUP BY   字段名1,字段名2  HAVING COUNT(*) > 1
             )
```

三、写出一个冒泡排序

```
public class BubbleSort {
    public static void BubbleSort(int[] arr) {
        int temp;//定义一个临时变量
        for(int i=0;i<arr.length-1;i++){//冒泡趟数
            for(int j=0;j<arr.length-i-1;j++){
                if(arr[j+1]<arr[j]){
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
```

四、二分查找法（面试题给一个数从一个数组中找出它的位置）

```java
/**
 * 使用递归的二分查找
 *title:recursionBinarySearch
 *@param arr 有序数组
 *@param key 待查找关键字
 *@return 找到的位置
 */
public static int recursionBinarySearch(int[] arr,int key,int low,int high){

        if(key < arr[low] || key > arr[high] || low > high){
                return -1;
        }

        int middle = (low + high) / 2;                  //初始中间位置
        if(arr[middle] > key){
                //比关键字大则关键字在左区域
                return recursionBinarySearch(arr, key, low, middle - 1);
        }else if(arr[middle] < key){
                //比关键字小则关键字在右区域
                return recursionBinarySearch(arr, key, middle + 1, high);
        }else {
                return middle;
        }

}
```

五、补充