1.HBase与MySQL的区别

答：HBase是一个数据库----可以提供数据的实时随机读写，HBASE与mysql、oralce、db2、sqlserver等关系型数据库不同，它是一个NoSQL数据库（非关系型数据库）

数据类型：HBase只有简单的字节类型，所有的类型都是交由用户自己处理。而关系数据库有丰富的类型和存储方式。

数据操作：HBase只有很简单的插入、查询、删除、清空等操作，表和表之间是分离的，没有复杂的表和表之间的关系，而传统数据库通常有各式各样的函数和连接操作。

存储模式：HBase是基于列存储的，每个列族都由几个文件保存，不同的列族的文件是分离的。而传统的关系型数据库是基于表格结构和行模式保存的

数据维护：HBase的更新操作不应该叫更新，它实际上是插入了新的数据，而传统数据库是替换修改

可伸缩性：HBase这类分布式数据库就是为了这个目的而开发出来的，所以它能够轻松增加或减少硬件的数量，并且对错误的兼容性比较高。而传统数据库通常需要增加中间层才能实现类似的功能

2.堆排？复杂度怎么算?

答：首先，将数组生成大根堆O(N)，然后排序，每次将最后一个数与大根堆的头节点交换，然后调整堆为大根堆，复杂度为O(NlogN)。

3.什么样的操作会导致数据相同的key很多?

答：如一个订单表一个用户表，join后，同一个用户会有好多记录

4.Hadoop调优

答：总体上来看，管理员需从**硬件选择、操作系统参数调优、JVM参数调优和Hadoop参数调优**等四个角度入手，为Hadoop用户提供一个高效的作业运行环境。

硬件选择：

主节点要好于从节点

操作系统：

关闭swap分区（物理内存不足会使用swap分区）[root@Hadoop ~]# echo "vm.swappiness = 0 " >> /etc/sysctl.conf

调整内存分配策略，操作系统内核根据vm.overcommit_memory的值来决定内存分配策略，并且通过vm.overcommit_ratio的值来设定超过物理内存的比例，前者一般设置为2，

后者根据实际情况调整。 [root@NameNode ~]# echo 2 > /proc/sys/vm/overcommit_memory

修改net.core.somaxconn参数，表示socket监听backlog的上限，backlog是套接字的监听队列，默认是128，socket服务器会一次性处理backlog中的所有请求，当服务器繁忙的时候，128远远不够的，建议大于等于32768

禁止文件系统的访问时间

关闭THP，THP（Transparent Huge Pages）是一个使管理Huge Pages自动化的抽象层，**THP会引起CPU占用率偏高**，需要将其关闭

JVM参数调优：

JVM垃圾回收以吞吐量优先为原则：Parallel Scavenge + Parallel Old

Hadoop参数调优：

hdfs-site.xml

```
<property>
    <name>dfs.blocksize</name>
    <value>134217728</value>
</property>
```

该参数表示Hadoop文件块大小，通常设置为128MB或者256MB。

```
<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>
```

该参数表示控制HDFS文件的副本数，默认是3，当许多任务同时读取一个文件时，读取可能造成瓶颈，这时增大副本数能有效缓解，但会造成大量的磁盘空间占用，这时只修改Hadoop客户端的配置，这样，从Hadoop客户端上传的文件的副本数以Hadoop客户端的为准。

```
<property>
    <name>dfs.namenode.handler.count</name>
    <value>40</value>
</property>
```

该参数表示NameNode同时和DataNode通信的线程数，默认是10，将其增大为40。

core-site.xml

```
<property>
    <name>io.file.buffer.size</name>
    <value>65536</value>
</property>
```

缓冲区的大小用于读写HDFS文件，默认4KB，增加64KB。

Yarn-site.xml

```
<property>
    <name>yarn.scheduler.increment-allocation-mb</name>
    <value>512</value>
</property>
```

该参数表示内存申请大小的规整化单位，默认是1024MB，即如果申请的内存是1.5GB，将被计算为2GB。

mapred-site.xml

```
<property>
    <name>mapreduce.map.output.compress</name>
    <value>true</value>
</property>
```

该参数表示Map任务的中间结果是否进行压缩，设定为true时，会对中间结果进行压缩，这样会减少数据传输时带宽的需要。

```
<property>
    <name>mapreduce.job.jvm.numtasks</name>
    <value>1</value>
</property>
```

该参数表示JVM重用次数，默认是1。

参考：

Cloudera Hadoop运维管理与性能调优.note

5.你能讲讲Spark吗

答：Spark是一种**快速、通用、可扩展**的大数据分析引擎，Spark是基于内存计算的大数据并行计算框架。**基于MapReduce的计算引擎通常会将中间结果输出到磁盘上**，进行存储和容错，如果处理逻辑复杂，则需要频繁进行磁盘的读写，磁盘IO有很大瓶颈，而Spark则可

以将中间结果写入到内存，大大降低了磁盘IO，与Hadoop相比Spark通常要快10到100倍。**Spark提供了很多高级算子**，可以使用户快速构建不同的应用。**Spark提供了一站式解决方案**，Spark可以用于批处理、交互式查询（Spark SQL）、实时流处理（Spark Streaming）、机器学习（Spark MLlib）和图计算（GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。**Spark的兼容性也很好**，可以直接跑在Yarn上。**RDD是Spark的核心**。RDD（Resilient Distributed Dataset）叫做弹性分布式数据集，是Spark中最基本的数据抽象，它代表一个不可变、可分区、里面的元素可并行计算的集合。用户感知不到数据的分布，操作RDD就如同操作本地集合一样。**RDD有五大特性**，一系列分区，会有一个函数作用在每个切片上，RDD和RDD之间存在依赖关系，（可选）如果是RDD中装的是KV类型的，那么Shuffle时会有一个分区器，默认是HashPartitioner，（可选）如果只从HDFS中读取数据，会感知数据的位置，将Executor启动在数据所在的机器上。

RDD上的操作有Transformation和Action两种，Transformation是懒加载的，一旦遇到Action，Spark就会构建DAG，从后往前，遇到宽依赖就切分成不同的Stage，然后将相同的任务以TaskSet的形式提交进行计算。这就是我理解的Spark。

6.讲讲你的职业规划

答：**长远的规划**，我没想太具体，主要是想深耕这个领域，争取早日成为一位大数据专家。**至于短期的规划**，我因为刚毕业，项目经验还不是很充足，但是如果我能获得这次宝贵的机会，那么我会努力学习，多请教前辈同事，提高自己的项目，在公司的大数据项目上做出自己的贡献。

7.讲讲Python中的yield

8.你能讲讲Java多线程吗

答：线程的状态->线程的创建方式->线程安全(synchronize和ReentrantLock)->并发包

9.你能讲讲Kafka吗

答：Kafka最初由Linkedin公司开发，是一个分布式、分区、多副本、多订阅者，基于zookeeper协调的分布式日志系统（也可以当做MQ系统）。主要应用场景是：日志收集系统和消息系统。

Broker：消息中间件处理结点，一个Kafka节点就是一个Broker，多个Broker可以组成一个Kafka集群。

Topic：一类消息，Kafka集群能够同时负责多个topic的分发。

Partition：topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列。
Segment：partition物理上由多个segment组成。
offset：每个partition都由一系列有序的、不可变的消息组成，这些消息被连续的追加到partition中。partition中的每个消息都有一个连续的序列号叫做offset，用于partition唯一标识一条消息。
Producer：负责发布消息到Kafka broker。
Consumer：消息消费者，向Kafka broker读取消息的客户端。
Consumer Group：每个Consumer属于一个特定的Consumer Group。

Kafka的数据传输事物默认保证**at-least-once**
Kafka是基于硬盘的，但是它之所以这么快，是有很多原因的，写入时顺序写+mmap，读取时page cache + sendfile

10.Http访问页面的流程
答：
1，浏览器首先会查询本机的系统，获取主机名对应的IP地址。
2，若本机查询不到相应的IP地址，则会发起DNS请求，获取主机名对应的IP地址。
3，使用查询到的IP地址，直接访问目标服务器。
4，浏览器发送HTTP请求。
　　HTTP请求由三部分组成，分别是：**请求行、消息报头、请求正文**
5， 从请求信息中获得客户机想访问的主机名。
6，从请求信息中获取客户机想要访问的web应用。（web应用程序指提供浏览器访问的程序，简称web应用）
7，从请求信息中获取客户机要访问的web资源。（web资源，即各种文件，图片，视频，文本等）
8，读取相应的主机下的web应用，web资源。
9，用读取到的web资源数据，创建一个HTTP响应。
10，服务器回送HTTP响应。
　　HTTP响应也是由三个部分组成，分别是：**状态行、消息报头、响应正文**
参考：
一个web页面的访问的过程.note

11.实现负载均衡的几种方式
答：

（1）HTTP重定向负载均衡

这种负载均衡方案的优点是比较简单；

缺点是浏览器需要每次请求两次服务器才能拿完成一次访问，性能较差。

（2）DNS域名解析负载均衡

优点是将负载均衡工作交给DNS，省略掉了网络管理的麻烦；

缺点就是DNS可能缓存A记录，不受网站控制。

（3）反向代理负载均衡

优点是部署简单；

缺点是反向代理服务器是所有请求和响应的中转站，其性能可能会成为瓶颈。

（4）IP负载均衡

优点：IP负载均衡在内核进程完成数据分发，较反向代理均衡有更好的处理性能。

缺点：负载均衡的网卡带宽成为系统的瓶颈。

（5）数据链路层负载均衡

避免负载均衡服务器网卡带宽成为瓶颈，是目前大型网站所使用的最广的一种负载均衡手段。

参考：

几种负载均衡技术的实现.note

12.Java 线程之间如何通信

答：同步、while轮询、wait/notify、管道

分布式系统中说的两种通信机制：共享内存机制和消息通信机制。感觉前面的①中的synchronized关键字和②中的while轮询 “属于” 共享内存机制，由于是轮询的条件使用了volatile关键字修饰时，这就表示它们通过判断这个“共享的条件变量“是否改变了，来实现进程间的交流。

而管道通信，更像消息传递机制，也就是说：通过管道，将一个线程中的消息发送给另一个。

参考：

JAVA线程间通信的几种方式.note

13.你能不能谈谈，java GC是在什么时候，对什么东西，做了什么事情?

答：

什么时候：程序员不能具体控制时间，系统在不可预测的时间调用System.gc()函数的时候；当然可以通过调优，用NewRatio**控制newObject和oldObject的比例，**用MaxTenuringThreshold **控制进入oldObject的次数，使得oldObject 存储空间延迟达到full gc,从而使得计时器引发gc时间延迟**OOM的时间延迟，**以延长对象生存期。**

对什么东西：超出了作用域或引用计数为空的对象；从gc root开始搜索找不到的对象，而且经过一次标记、清理，仍然没有复活的对象。

做了什么事情：删除不使用的对象，回收内存空间；运行默认的finalize,当然程序员想立刻调用就用dipose调用以释放资源如文件句柄，JVM用from survivor、to survivor对它进行标记清理，对象序列化后也可以使它复活。

参考：

面试题：“你能不能谈谈，java GC是在什么时候，对什么东西，做了什么事情？”.note

14.Java什么时候发生死锁

答：创建两个字符串a和b，再创建两个线程A和B，让每个线程都用synchronized锁住字符串（A先锁a，再去锁b；B先锁b，再锁a），如果A锁住a，B锁住b，A就没办法锁住b，B也没办法锁住a，这时就陷入了死锁

15.如何避免死锁

答：破坏死锁的四个必要条件（**互斥条件，请求与保持条件，不可剥夺条件，循环等待条件**）

参考：

死锁的四个必要条件？如何避免与预防死锁？.note

16.Spark看过哪些源码

答：**我主要说下它的RPC通信和Shuffle**

RpcEndpoint：需要通信的个体

RpcEndpointRef：对RpcEndpoint的一个引用

RpcEnv：为RpcEndpoint提供处理消息的环境，提供了停止、注册、获取endpoint等方法的定义

Dispatcher：分发器，用于转发客户端发来的消息，RpcEndpoint就是注册在这里

在standalone模式中，worker会定时发心跳消息（SendHeartbeat）给master

worker发送流程：

1.**forwordMessageScheduler线程**定时自己发送SendHeartbeat消息

2.worker的receive函数接收到消息后，调用自己的sendToMaster(message)函数

3.然后在函数中通过masterRef.send(message)=>nettyEnv.send(message)方法发送消息，**如果是本地消息就调用 dispatcher.postOneWayMessage(message)，远程消息调用postToOutbox(message.receiver,**

OneWayOutboxMessage(serialize(message)))，最终通过**TransportClient**的send方法将消息发给master

master接收流程

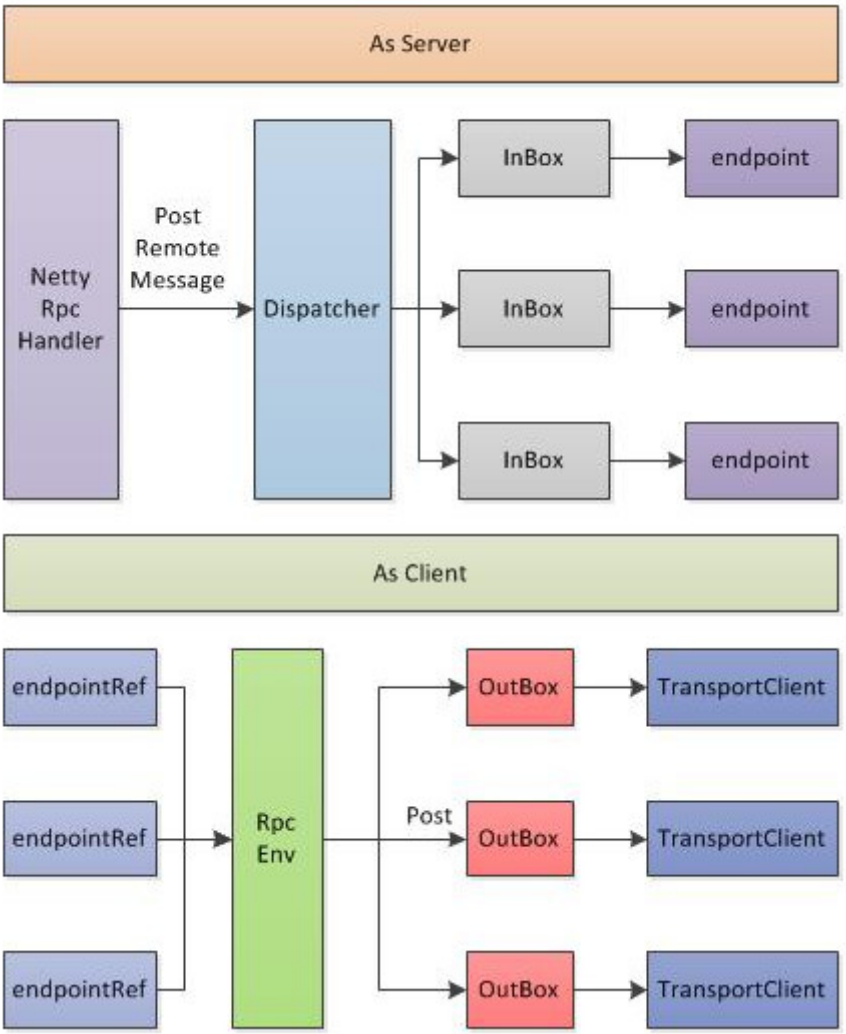
1.在master端TransportRequestHandler的handle方法中，由于心跳信息在worker端被分装成了OneWayMessage，所以在该handle方法中，将调用processOneWayMessage进行处理

2.processOneWayMessage函数将调用rpcHandler的实现类NettyRpcHandler中的receive方法。在该方法中，首先通过internalRecieve将消息解包成RequestMessage。然后该消息通过dispatcher的postOneWayMessage分发给对应的endpoint

3.在Dispatcher的postMessage方法中，可以看到，首先根据对应的endpoint的EndpointData信息（主要是该endpoint及其应用以及其信箱（inbox）），然后将消息塞到给endpoint（此例中的master）的信箱中，最后将消息塞到**recievers的阻塞队列**中

4.**在Dispatcher中有一个线程池threadpool在MessageLoop类的run方法中**，将receivers中的对象取出来，交由信箱的process方法去处理。

5.在inbox的proces方法中，首先取出消息，然后根据消息的类型（此例中是oneWayMessage），最终将调用endpoint的receiver方法进行处理（也就是master中的receive方法）。至此，整个一次rpc调用的流程结束



Spark的sort shuffle

spill到磁盘中的文件由blockmanager管理

shuffle write

主要看**SortShuffleWriter.write()**方法，使用**ExternalSorter**容器，内部包装了两个数据结构。

当计算结果需要map combine，则new ExternalSorter[K, V, C](context, dep.aggregator, Some(dep.partitioner), **dep.keyOrdering**, dep.serializer)，**在spill时对partition和key 进行排序**，否则new ExternalSorter[K, V, V](context, aggregator = None, Some(dep.partitioner), **ordering = None**, dep.serializer)，**在spill时只对Partition进行排序**。排序规则是先按照keyComparator对key排序，然后使用partitionID大小排序，**keyComparator根据key的hashCode进行排序**。

map combine使用的数据结构是**PartitionedAppendOnlyMap**，否则使用**PartitionedPairBuffer**。默认大小为**spark.shuffle.spill.initialMemoryThreshold=5MB**或者结构中元素数目达到**spark.shuffle.spill.numElementsForceSpillThreshold=Long.MaxValue**时，将内存

中的集合spill到一个有序文件中。之后SortShuffleWriter.write中会调用sorter.writePartitionedFile来merge它们同时进行排序。
sorter.insertAll(records)将数据写入内存数据结构，若大小超出阈值，则将其溢写到磁盘数据文件，溢出时已经按照key的hashcode排好序。最后将多个溢出文件进行合并并使用TimSort排序。

shuffle read
调用的是rdd.ShuffledRDD.compute，然后调用SortShuffleManager的getReader方法，然后调用**BlockStoreShuffleReader.read()**方法，接着**通过mapOutputTracker获取存储数据位置的元数据**，然后去拉取数据，**每个reduce任务分配一个缓冲区，大小为spark.reducer.maxSizeInFlight=48MB**。
判断是否需要进行map端聚合(对于下一个Shuffle来说)，若需要dep.aggregator.get.combineCombinersByKey(combinedKeyValuesIterator, context)，否则dep.aggregator.get.combineValuesByKey(keyValuesIterator, context)，**内部都是使用ExternalAppendOnlyMap.insertAll对数据进行处理**，ExternalAppendOnlyMap也是一个容器，包装了一个**SizeTrackingAppendOnlyMap**，和ExternalSorter一样都是Spillable的子类，这里的spill准则同上，spill时使用HashComparator。**判断是否需要对key进行排序(这个看map端的算子，如果聚合了这里就要排序)**，如果需要对key进行排序则使用ExternalSorter，否则直接返回结果aggregatedIter。之后读取溢写文件时会用TimSort进行排序。

广度优先遍历，又称为广度优先搜索，简称BFS。举例说明，在一套房子里找一个钥匙，利用深度优先搜索就是搜索每一个房间，而广度优先搜索，是先看看钥匙有没有放在各个房间的明显位置，如果没有，再看看各个房间的抽屉有没有，这样逐步扩大查找的范围的方式我们称之为广度优先搜索。
利用广度优先遍历生成stage，如果RDD所在的stage已经生成就直接返回。
利用深度优先遍历提交stage，一条路走到底，先提交之前的stage。

17.http1.0和1.1的区别
答：**HTTP 1.0规定浏览器与服务器只保持短暂的连接**，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。 **HTTP 1.1的持续连接**，也需要增加新的请求头来帮助实现，例如，**Connection请求头**的值为**Keep-Alive**时，客户端通知服务器返回本次请求结果后保持连接；Connection请求头的值为**close**时，客户端通知服务器返回本次请求结果后关闭连接。**HTTP 1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头**。

HTTP 1.1增加host字段，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。**HTTP/1.1加入了一个新的状态码100（Continue）**，100 (Continue) 状态代码的使用，允许客户端在发request消息body之前先用request header试探一下server，看server要不要接收request body，再决定要不要发request body。

参考：

HTTP1.0和HTTP1.1的区别.note

18.Java类的加载过程

答：

加载：通过一个类的全限定名来获取定义此类的二进制字节流，并在内存中生成代表这个类的Class对象。加载阶段完成后，虚拟机外部的 二进制字节流就按照虚拟机所需的格式存储在方法区之中，而且在Java堆中也创建一个java.lang.Class类的对象，**这样便可以通过该对象访问方法区中的这些数据。**

验证：确保 Class 文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全，比如检查魔数。

准备：准备阶段为类变量分配内存并设置初始值，使用的是方法区的内存。

解析：将常量池的符号引用替换为直接引用的过程。

两个重点：

- 符号引用。即一个字符串，但是这个字符串给出了一些能够唯一性识别一个方法，一个变量，一个类的相关信息。
- 直接引用。可以理解为一个内存地址，或者一个偏移量。比如类方法，类变量的直接引用是指向方法区的指针；而实例方法，实例变量的直接引用则是从实例的头指针开始算起到这个实例变量位置的偏移量

举个例子来说，现在调用方法hello()，这个方法的地址是1234567，那么hello就是符号引用，1234567就是直接引用。

在解析阶段，虚拟机会把所有的类名，方法名，字段名这些符号引用替换为具体的内存地址或偏移量，也就是直接引用。

初始化：初始化阶段才真正开始执行类中定义的 Java 程序代码。初始化阶段即虚拟机执行类构造器 <clinit>() 方法的过程。

类初始化的触发条件:只有当对类的主动使用的时候才会导致类的初始化。

(1)遇到new、getstatic、putstatic或invokestatic这4条字节码指令时，如果类没有进行过初始化，则需要先触发其初始化。生成这4条指令的最常见的Java代码场景是：使用new关键字实例化对象的时候，读取或设置一个类的静态字段（被final修饰、已在编译期把结果放入常量池的静态字段除外）的时候，以及调用一个类的静态方法的时候。

(2) 使用java.lang.reflect包的方法对类进行反射调用的时候，如果类没有进行过初始化，则需要先触发其初始化。

(3) 当初始化一个类的时候，如果发现其父类还没有进行过初始化，则需要先触发其父类的

初始化。

(4) 当虚拟机启动时，用户需要指定一个要执行的主类（包含main()方法的那个类），虚拟机会先初始化这个主类。

19.Java反射

答：**JAVA反射机制是在运行状态中**，对于任意一个类，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一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

反射的作用：通过反射运行配置文件内容，通过反射越过泛型检查

参考：

Java基础之—反射（非常重要）.note

20.Spark调优总结

1）防止GC是的shuffle拉取数据失败

spark.shuffle.io.maxRetries=60(3)　　spark.shuffle.io.retryWait=60(5)

2）更改缓冲区大小

spark.shuffle.file.buffer=128k(32k)

3）更改sortBuffer大小

spark.reducer.maxSizeInFlight=128m(48m)

4）SparkSQL修改shuffle partition

spark.sql.shuffle.partition=500(200)

5）对多次使用的RDD进行持久化

6）使用filter后进行coalesce操作

7）使用repartitionAndSortWithinPartitions替代repartition与sort类操作

8）使用Kyro优化序列化性能

```
// 创建SparkConf对象。
val conf = new SparkConf().setMaster(...).setAppName(...)
// 设置序列化器为KryoSerializer。
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
// 注册要序列化的自定义类型。
conf.registerKryoClasses(Array(classOf[MyClass1], classOf[MyClass2]))
```

9）广播共享变量

21.MySQL的事务隔离级别

答：

事物的基本要素(ACID)：原子性，一致性，隔离性，持久性

事物的并发问题：脏读，不可重复读，幻读

MySQL事物的隔离级别：读未提交，读已提交，可重复读，串行化

参考：

MySQL的四种事务隔离级别.note

22.CAP原理

答：CAP原则又称CAP定理，指的是在一个分布式系统中， Consistency（一致性）、Availability（可用性）、Partition tolerance（分区容错性），三者不可得兼。

一致性：读操作总是能读取到之前完成的写操作结果，满足这个条件的系统称为强一致系统，这里的“之前”一般对同一个客户端而言；

可用性：读写操作在单台机器发生故障的情况下仍然能够正常执行，而不需要等待发生故障的机器重启或者其上的服务迁移到其他机器；

分区可容忍性：机器故障、网络故障、机房停电等异常情况下仍然能够满足一致性和可用性。

为什么要一般一定要满足分区容错性：放弃P的同时意味着放弃了系统的可扩展性(将所有的数据放在一个分布式节点上)。

一个分布式系统里面，节点组成的网络本来应该是连通的。然而可能因为一些故障，使得有些节点之间不连通了，整个网络就分成了几块区域。数据就散布在了这些不连通的区域中。这就叫分区。

当你一个数据项只在一个节点中保存，那么分区出现后，和这个节点不连通的部分就访问不到这个数据了。这时分区就是无法容忍的。

提高分区容忍性的办法就是一个数据项复制到多个节点上，那么出现分区之后，这一数据项就可能分布到各个区里。容忍性就提高了。

然而，要把数据复制到多个节点，就会带来一致性的问题，就是多个节点上面的数据可能是不一致的。要保证一致，每次写操作就都要等待全部节点写成功，而这等待又会带来可用性的问题。

总的来说就是，数据存在的节点越多，分区容忍性越高，但要复制更新的数据就越多，一致性就越难保证。为了保证一致性，更新所有节点数据所需要的时间就越长，可用性就会降低。

参考：

一致性（Consistency），可用性（Avilable）,分区容错性（Tolerance of network Partition）.note

23.HBase的读写流程

答：

HBase写入流程

(1) **用户提交put请求后，HBase客户端会将put请求添加到本地buffer中。**HBase默认设置autoflush=true，表示put请求直接会提交给服务器进行处理；用户可以设置autoflush=false，这样的话put请求会首先放到本地buffer，等到本地buffer大小超过一定阈值（默认为2M，可以通过配置文件配置）之后才会提交。

(2) **在提交之前，HBase会在元数据表.meta.中根据rowkey找到它们归属的region server**，这个定位的过程是通过HConnection 的locateRegion方法获得的。如果是批量请求的话还会把这些rowkey按照HRegionLocation分组，每个分组可以对应一次RPC请求。

(3) 请求写HLog

(4) 请求写MemStore

(5) 当MemStore达到128M后则把数据刷成一个HFile文件

(6) 当多个HFile文件达到一定的大小后，会触发Compact合并操作，合并为一个StoreFile(这里同时进行版本的合并和数据删除)

HBase读取流程

第1步：Client请求ZK获取.META.所在的RegionServer的地址。

第2步：Client请求.META.所在的RegionServer获取访问数据所在的RegionServer地址，client会将.META.的相关信息cache下来，以便下一次快速访问。

第3步：Client请求数据所在的RegionServer，获取所需要的数据。

第4步：获取过的数据会缓存在BlockCache中，以block保存，大小为64k。

24.HBase如何保证数据一致性

答：

HBase是强一致性的

每个值只出现在一个Region中

同一时间一个Region只分配给一个RegionServer服务器

先写在HLog中

行内操作时原子的，要么成功要么失败

HBase的强一致性和HDFS的多副本是否冲突?

不冲突，**数据的更新都是首先写在MemStore中**，只有达到阈值才会刷到磁盘中，而一旦刷到磁盘中，数据就不会改变了

参考：

HBase——强一致性详解.note

25.HBase刷盘策略

答：

Memstore级别限制：当Region中任意一个MemStore的大小达到了上限（hbase.hregion.memstore.flush.size，**默认128MB**），会触发Memstore刷新。

Region级别限制：当Region中所有Memstore的大小总和达到了上限（hbase.hregion.memstore.block.multiplier * hbase.hregion.memstore.flush.size，默认 2* 128M = **256M**），会触发memstore刷新。

Region Server级别限制：当一个Region Server中所有Memstore的大小总和达到了上限（hbase.regionserver.global.memstore.upperLimit * hbase_heapsize，**默认 40%的JVM内存使用量**），会触发部分Memstore刷新。Flush顺序是按照Memstore由大到小执行，先Flush Memstore最大的Region，再执行次大的，直至总体Memstore内存使用量低于阈值（hbase.regionserver.global.memstore.lowerLimit * hbase_heapsize，默认38%的JVM内存使用量）。

当一个Region Server中HLog数量达到上限（可通过参数hbase.regionserver.maxlogs配置）时，系统会选取最早的一个 HLog对应的一个或多个Region进行flush

HBase定期刷新Memstore：**默认周期为1小时**，确保Memstore不会长时间没有持久化。**为避免所有的MemStore在同一时间都进行flush导致的问题，定期的flush操作有20000左右的随机延时。**

手动执行flush：用户可以通过shell命令 flush ‘tablename’ 或者flush ‘region name’ 分别对一个表或者一个Region进行flush。

26.HBase怎么删除数据

答：对数据在内存中先进行标记，在磁盘文件合并时将数据删除

27.HBase并发控制

答：

写写并发

(1) 获取所有待写入（更新）行记录的行锁

(2) 开始执行写入（更新）操作

(3) 写入完成之后再统一释放所有行记录的行锁

读写并发

MVCC--Mutil Version Concurrent Control

(1) 为每一个写（更新）事务分配一个Region级别自增的序列号(SequenceId)

(2) 为每一个读请求分配一个已完成的最大写事务序列号

参考：

数据库事务系列 - HBase行级事务模型.note

28.RegionServer故障恢复

答：

(1) RegionServer的相关信息保存在ZK中，在RegionServer启动的时候，会在Zookeeper中创建对应的临时节点。RegionServer通过Socket和Zookeeper建立session会话，**RegionServer会周期性地向Zookeeper发送ping消息包**，以此说明自己还处于存活状态。**而Zookeeper收到ping包后，则会更新对应session的超时时间。**

(2) 当Zookeeper超过session超时时间还未收到RegionServer的ping包，则Zookeeper会认为该RegionServer出现故障，**ZK会将该RegionServer对应的临时节点删除，并通知Master**，Master收到RegionServer挂掉的信息后就会启动数据恢复的流程。

Master启动数据恢复流程后，其实主要的流程如下：

(3) RegionServer宕机---》ZK检测到RegionServer异常---》Master启动数据恢复---》Hlog切分---》Region重新分配---》Hlog重放---》恢复完成并提供服务

那么宕机回滚的时候如果有写数据怎么办呢?

写不了，恢复了才能写

29.GC参数整理

答：

-XX:+UseSerialGC：在新生代和老年代使用串行收集器

-XX:SurvivorRatio：设置eden区大小和survivior区大小的比例

-XX:NewRatio:新生代和老年代的比

-XX:+UseParNewGC：在新生代使用并行收集器

-XX:+UseParallelGC ：新生代使用并行回收收集器

-XX:+UseParallelOldGC：老年代使用并行回收收集器

-XX:ParallelGCThreads：设置用于垃圾回收的线程数

-XX:+UseConcMarkSweepGC：新生代使用并行收集器，老年代使用CMS+串行收集器

-XX:ParallelCMSThreads：设定CMS的线程数量

-XX:CMSInitiatingOccupancyFraction：设置CMS收集器在老年代空间被使用多少后触发

-XX:+UseCMSCompactAtFullCollection：设置CMS收集器在完成垃圾收集后是否要进行一次内存碎片的整理

-XX:CMSFullGCsBeforeCompaction：设定进行多少次CMS垃圾回收后，进行一次内存压缩

-XX:+CMSClassUnloadingEnabled：允许对类元数据进行回收
-XX:CMSInitiatingPermOccupancyFraction：当永久区占用率达到这一百分比时，启动CMS回收
-XX:UseCMSInitiatingOccupancyOnly：表示只在到达阀值的时候，才进行CMS回收

30.Kafka中zookeeper的作用

答：
1.Broker注册 /brokers/ids
2.Topic注册 /brokers/topics/topicname/3->2 这个节点表示Broker ID为3的一个Broker服务器，对于"login"这个Topic的消息，提供了2个分区进行消息存储
3.生产者负载均衡
4.消费者负载均衡
5.以前版本消息消费进度Offset 记录
6.消费者注册
7.leader选举

31.Kafka的容错

答：假设replica.lag.max.messages设置为4，表明只要follower落后leader不超过3，就不会从同步副本列表中移除。replica.lag.time.max设置为500 ms，表明只要follower向leader发送请求时间间隔不超过500 ms，就不会被标记为死亡,也不会从同步副本列中移除。
通过request.required.acks参数调整生产者是否等消息同步后再发。默认是1，改成-1较好。

如何处理所有Replica都不工作?

这就需要在可用性和一致性当中作出一个简单的折衷。如果一定要等待ISR中的Replica“活”过来，那不可用的时间就可能会相对较长。而且如果ISR中的所有Replica都无法“活”过来了，或者数据都丢失了，这个Partition将永远不可用。**选择第一个“活”过来的Replica作为Leader，而这个Replica不是ISR中的Replica**，那即使它并不保证已经包含了所有已commit的消息，它也会成为Leader而作为consumer的数据源（前文有说明，所有读写都由Leader完成）。Kafka0.8.*使用了第二种方式。根据Kafka的文档，在以后的版本中，Kafka支持用户通过配置选择这两种方式中的一种，从而根据不同的使用场景选择高可用性还是强一致性。

参考：

Kafka设计解析（六）- Kafka高性能架构之道.note

32.get与post的区别

答：

HTTP request 请求 主要是 get 和post， get请求参数在请求行里面，post的参数封装在请求体里面

第一，GET提交的数据会放在URL之后，POST方法是把提交的数据放在HTTP包的Body中.

在客户端，Get方式在通过URL提交数据，参数在请求行里面，数据在URL中可以看到；

POST方式，数据在请求体里面，数据放置在HTML HEADER内提交。

第二，**GET提交的数据大小有限制**（**因为浏览器对URL的长度有限制，一般为1024字节注意HTTP协议没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制，但是具体的浏览器有**），而POST方法提交的数据没有限制.

第三，GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

第四，GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

第五，**get是从服务器上获取数据，post是向服务器传送数据**。

33.Java如何定位内存泄漏

答：首先使用jstat -gcutil 15469 1000 300查看GC信息，如果有大量的FGC就要查询是否有内存泄漏的问题了。然后使用jmap -dump:<dump-options><pid>生成堆转储文件。最后使用Memory Analyzer Tool对dump出来的文件进行分析。

34.Java的进程CPU占用很高，如何分析原因

答：首先通过Top命令查看占用CPU较高的进程PID，执行Top之后按1可以查看每个核占用比例。然后抓取栈信息jstack -F pid > dump.txt