1、**多态（了解） 的含义**约等同于“同一个方法对于不同类型的输入参数均能做出正确的处理过程，并给出人们所期望获得的结果”

C++的多态性是C++实现面向对象技术的基础。具体的说，通过一个指向基类的指针调用虚成员函数的时候，

运行时系统将能够根据指针所指向的实际对象调用恰当的成员函数实现。

主要是**通过父类子类的继承和虚函数来实现**，当基类和子类拥有同名同参同返回的方法，且该方法声明为**虚方法**，当一个基类对象、指针、引用指向的是子类的对象的时候，基类对象、指针、引用在调用父类的方法，实际上调用的是子类方法。

多态的具体实现：

声明一个类时，如果类中有虚方法，则自动在类中增加一个**虚函数指针，**该指针指向的是一个虚函数表，**虚函数表中存着每个虚函数真正对应的函数地址**。但是拥有虚函数的类，在调用虚函数时，首先去**查虚函数表**，然后在确定调用的是哪一个函数，所以，调用的函数是在运行时才会确定的。把父[类的成员函数](https://baike.baidu.com/item/%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E8%99%9A%E5%87%BD%E6%95%B0/_blank)设为**virtual**

静态多态靠编译器来实现，简单来说就是编译器对原来的函数名进行修饰，const。

虚表指针初始化问题

当创建子类对象时，编译器的执行顺序其实是这样的：

对象在创建时，由编译器对 vptr 进行初始化

子类的构造会先调用父类的构造函数，这个时候 vptr 会先指向父类的虚函数表

子类构造的时候，vptr 会再指向子类的虚函数表

对象的创建完成后，vptr 最终的指向才确定

### **2、虚函数表的构成（了解）**

无覆盖的公有单继承：****虚函数按声明顺序存在于虚表中； 在派生类中，前面是基类的虚函数，后面是派生类的虚函数。****

有覆盖的公有单继承：****先拷贝基类的虚表；如果派生类重写了基类的虚函数，则修改同位置的基类虚函数；最后面是生类新定义的虚函数。****

****多继承时：****按顺序，父类1->子类->父类2

即如果子类有新定义的虚函数，则存放在继承顺序第一的基类续表的最后面

3、拷贝构造函数（了解）

拷贝构造函数调用时机：a.对象需要通过另外一个对象进行初始化b.对象以值传递的方式从函数返回c.对象以值传递的方式传入函数参数。定义方式：（const 类名 & 变量名）

4、指针与引用的区别（重点）

1）指针：**指针**是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而**引用**跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已，在内存占有同一个存储单

元）。

2）指针的值可以为空，但是引用的值不能为NULL，并且引用在定义的时候必须初始化；

3）指针可以有多级，但是引用只能是一级。

4）可以有const指针，但是没有const引用

5）”sizeof引用”得到的是所指向的变量(对象)的大小，而”sizeof指针”得到的是指针本身的大小；

6）指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了。

**7）如果返回动态内存分配的对象或者内存，必须使用指针，引用可能引起内存泄漏。**

5、C++中volatile关键字的作用，什么场景一定要用volatile关键字呢：

保证了变量的可见性（visibility）。被volatile关键字修饰的变量，如果值发生了变更，其他线程立马可见，避免出现脏读的现象

场景1.volatile****最适用一个线程写，多个线程读的场合****

场景2、结合使用 volatile 和 synchronized 实现 “开销较低的读－写锁”，volatile 允许多个线程执行读操作，因此当使用 volatile 保证读代码路径时，要比使用锁执行全部代码路径获得更高的共享度

1. 说一下C++是怎么实现哈希表的（重点）

哈希表的目的就是为了根据数据的部分内容（关键字），直接计算出存放完整数据的地址（哈希值）。

****哈希函数的构造方法：****

1. 直接定址法
2. 数字分析法
3. 平方取中法
4. 折叠法
5. 除留余数法
6. 随机数法

哈希冲突：多个数据同一个哈希值

****链地址法、开放定址法****

1. map在底层使用了红黑树来实现；unordered\_map是它的底层是使用hash表的形式来完成映射的功能。动态数据结构时，红黑树仍旧是较好的选择。

unordered\_map：hashtable处理hash冲突的方法就是在相同hash值的元素位置下面挂buket（桶），当数据量在8以内使用链表来实现桶，当数据量大于8 则自动转换为红黑树结构 也就是有序map的实现结构

一个支持并发的unordered\_map的话要怎么实现，本身不支持，可用多线程；或者**ConcurrentHashMap 支持。**

1. 平衡二叉树和红黑树（重点，百度一下）：

平衡二叉树：它是一 棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。这个方案很好的解决了二叉查找树退化成链表的问题，把插入，查找，删除的时间复杂度最好情况和最坏情况都维持在O(logN)。

红黑树：是一种平衡二叉查找树的变体，它的左右子树高差有可能大于 1。性质1. 结点是红色或黑色。 性质2. 根结点是黑色。 性质3. 所有叶子都是黑色。（叶子是NIL结点） 性质4. 每个红色结点的两个子结点都是黑色。（从每个叶子到根的所有路径上不能有两个连续的红色结点）性质5. 从任一结点到其每个叶子的所有路径都包含相同数目的黑色结点。

1. 归并排序：分治法将问题**分**(divide)成一些小的问题然后递归求解，而**治(conquer)**的阶段则将分的阶段得到的各答案"修补"在一起，即分而治之
2. 机器内存比较小，数据量比较大的情况下要怎么排序？本来有序：直接插入、希尔、冒泡，不然就快速、归并、堆排
3. 一个进程里面的内存管理是怎样的

代码段(text segment)：用来存放程序的代码，具有可执行，可读权限。程序不可能在运行的过程中更改自己的代码。

数据段(data segment)：用来存放全局变量和静态变量(用static声明的)。

堆空间(heap storage)：堆空间的大小是动态变化的，用来做动态内存分配。

栈空间(stack segment)：栈空间用来保存局部变量，函数调用的实参，返回地址等。

环境变量：由系统设置的。

12、操作系统里面的内存管理的模型是什么样的

# 分区式内存管理：分配在一块连续的地址空间，一个区只能装入一个进程

# 分页式内存管理：不需要把所有部分完整的装入到内存中，只需要保证运行需要的部分在实际的物理内存中就行，虚拟内存的主要思想（重点）

# 分段式内存管理：进程对内存空间的应用主要分为两部分，程序(code)和数据(data)。单个地址空间划分成两个独立的空间

# 段页式内存管理

**13、linux进程与线程**（重点）**：**

进程是正在运行的程序，线程是CPU调度的最小单位。一个进程切换到另一个进程：****当发生进程调度时，进行进程切换就是上下文切换。****进程切换涉及到虚拟地址空间的切换。每个进程都有自己的虚拟地址空间，进程内的所有线程共享进程的虚拟地址空间。**因此同一个进程中的线程进行线程切换时不涉及虚拟地址空间的转换。**

**当操作系统决定要把控制权从当前进程转移到某个新进程时，就会进行上下文切换**，**即保存当前进程的上下文，恢复新进程的上下文，然后将控制权传递到新进程，新进程就会从上次停止的地方开始，**独占的使用硬件。

****14、Linux系统用户模式和内核模式：操作系统分为内核空间和用户空间，分别执行内核和用户进程，****一个运行在内核模式的进程可以执行指令集中的任何指令，并且可以访问系统中任何存储器位置。****用户模式中的进程不允许执行特权指令****，比如停止处理器，改变模式位，或者发起一个I/O操作。也不允许用户模式中的进程直接引用地址空间中内核区内的代码和数据。

15、Linux的gdb调式工具：

Run运行、break(简写 b) ：格式 b 行号，在某行设置断点；单步执行使用 continue继续执行，到下一个断点处、step单步调试如果有函数调用，则进入函数、next单步跟踪程序，当遇到函数调用时，也不进入此函数体命令；delete 断点号n：删除第n个断点；disable 断点号n：暂停第n个断点；enable 断点号n：开启第n个断点；clear 行号n：清除第n行的断点；

where/bt ：当前运行的堆栈列表；

bt backtrace 显示当前调用堆栈

up/down 改变堆栈显示的深度

set args 参数:指定运行时的参数

show args：查看设置好的参数

info program： 来查看程序的是否在运行，进程号

16、内部命令外部命令

内部命令：系统加载运行时shell就被加载并驻留在系统内存，可以直接执行，shell命令；外部命令：需要时才将其调进内存，其命令执行过程是由shell 程序控制。

17、Shell指令:

grep 命令用于查找文件里符合条件的字符串，find查找相应后缀的文本文件；

# sort排序，-r逆序；uniq求文件的交集，-d并集，-u差集；paste:将多个文件内容合并；cut将文件中指定行切割；

# sed按行用来自动编辑一个或多个文件简化对文件的反复操作，a下一行新增，i前一行插入，c替换行，d删除行，s替换匹配内容；awk可以按行列均可处理文件；xargs过滤器；

ls显示文件和目录；cd切换；cp复制，rm删除,mv移动,ln建立链接，

cat显示文件内容，vi编辑,touch创建一个空文件，

su切换用户，top查看系统cpu、内存等使用情况，free查看内存和swap分区使用情况，uptime查看系统运行时间，系统负载，vmstat监视虚拟内存使用情况，ps显示进程，kill干掉某个进程，进程号可以通过ps命令得到，

df检查文件系统的磁盘空间占用情况，du检测一个目录和所有它的子目录中占用的磁盘空间，mount挂载各种文件系统，fdisk对磁盘进行分区，useradd添加用户，passwd更改用户密码，chmod改变文件的权限

gzip -v  压缩文件，gunzip  -f  解压缩，tar，-zcvf  压缩文件或，格式tar.gz，-zxvf 解压缩文件或，格式tar.gz

**记以下指令较常用：ifconfig显示网络信息和临时配置ip，netstat显示网络状态-an  查看网络端口信息，-nr 查看路由表信息，默认DNS的文件的位置为：/etc/resolv.conf  ，ping、traceroute路由跟踪，nslookup命令域名解析排错，ssh、telnet远程登录，**

**vim 具有程序编辑的能力，会自动补全，程序的语法判断式，正则表达式的搜索，方便程序设计；vi是普通的文本编辑器，linux系统都内置。**

18、设置环境变量,环境变量是默认设定的参数：临时设置：export指令;用户变量永久设置~/.bash\_profile文件，系统变量设置修改在.etc/profile文件，export后面加参数。

19、为什么要用多线程？（充分利用多核CPU）

# **20、进程间通讯的7种方式**（重点）

**管道pipe**：管道是一种半双工的通信方式，亲缘关系的进程间使用

命名管道FIFO：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信

**消息队列MessageQueue**

**共享存储SharedMemory**

信号量Semaphore

**套接字Socket：套解口也是一种进程间通信机制**

**信号 ( sinal )**

21、网络层：IP协议提供一种无连接、不可靠的、尽力而为的数据包传输服务。

22、用一些可靠传输的原理——差错检测，重传，接收确认等机制。

传输层的Socket套接字看作不同主机间的进程进行双间通信的端点，是支持[TCP/IP协议](https://baike.baidu.com/item/TCP/IP%E5%8D%8F%E8%AE%AE/212915" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)的路通信的基本操作单元。流套接字之所以能够实现可靠的数据服务，原因在于其使用了[传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE/9727741" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)。数据报套接字使用UDP。**TCP连接是接收方和 发送方两个端系统自己所维护的一种连接状态，不断发送心跳，用一些可靠传输的原理——差错检测，重传，接受确认等机制。（了解）**

23、TCP为了保证可靠传输有哪些机制呢（重点）

##### **确认应答机制、超时重传机制、窗口流量控制（**接收端把窗口大小告诉发送端）**、拥塞控制。**

24、说一下拥塞控制的几个算法（重点）

慢开始( slow-start )、拥塞避免( congestion avoidance )、快重传( fast retransmit )和快恢复( fast recovery )。

25、为什么TCP的连接要三次握手啊?

三次握手：主机A通过向主机B 发送，告诉B可以用哪个序列号作为起始数据段来回应A，主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)告诉A可以用哪个序列号作为起始数据段来回应B，主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了。

为什么要三次握手标准回答：“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误，解决网络中存在延迟的重复分组”的问题。

解释：“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，**而是在某个网络结点长时间的滞留了**，**以致延误到连接释放以后的某个时间才到达server**。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。

26、TCP与UDP的区别（重点）

TCP：面向连接、面向字节、可靠传输、只能一对一、信道要求是全双工

UDP：是无连接的、面向报文、无差错控制、尽最大能力传输、可以一对一、一点多、多对一、多对多、首部简单

27、TCP网络服务器具体的工作

客户端：socket()、connect()

服务端：socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP)、bind()套接字和IP、端口绑定、listen()、accept()、recvmsg()、send()接收和发送

28、lambda匿名函数，执行一条表达式或者计算，不用声明参数类型，直接识别，简单方便。

左值：非临时地表达式，可取地址；右值：临时的表达式，如果立即数、函数返回。左值引用：int a=10;int &b=a;常引用：const int &a=10，只能读;右值引用：int &&a=10;右值引用可以读写，功能与常引用相似，可避免拷贝，实现move，避免重载的复杂性。左值引用可以作为函数返回引用、参数传递、别名。

29、c++ stl容器

顺序容器主要有：vector、list、deque等。其中vector表示一段连续的内存地址，基于数组的实现，list表示非连续的内存，基于链表实现。deque与vector类似，但是对于首元素提供删除和插入的双向支持。关联容器主要有map和set。map是key-value形式的，set是单值。map和set只能存放唯一的key值，multimap和multiset可以存放多个相同的key值。STL 中包含三种适配器：栈stack 、队列queue 和优先级队列priority\_queue 。

哪些容器内存分配在栈上，vector、set和map

容器全是支持动态增长的吗 vector、deque、list、set和map

容器的迭代器，begin 指向首地址，end 指向结束的地址

C++多线程使用，多线程操作的thread类，mutex保持数据同步。

以下内容为数据库相关知识：

30、什么是索引

索引是一种数据结构，用于帮助我们在大量数据中快速定位到我们想要查找的数据，索引的作用相当于图书的[目录](https://baike.baidu.com/item/%E7%9B%AE%E5%BD%95/96183" \t "https://baike.baidu.com/item/%E7%B4%A2%E5%BC%95/_blank)，提供指向存储在表的指定列中的数据值的指针，数据库使用索引以找到特定值，然后顺指针找到包含该值的行。这样可以使对应于表的SQL语句执行得更快，可快速访问数据库表中的特定信息

##### 31、MySQL的B+树是一个什么样的索引（重点）？

B+ 树非叶子节点上是不存储数据的，仅存储键值，而 B 树节点中不仅存储键值，也会存储数据

如果不存储数据，那么就会存储更多的键值，相应的树的阶数（节点的子节点树）就会更大，树就会更矮更胖，如此一来我们**查找数据进行磁盘的 IO 次数又会再次减少，数据查询的效率也会更快**。B+ 树的阶数是等于键值的数量的，**范围查找，排序查找，分组查找以及去重查找变得异常简单**。

32、数据库的并发安全是怎么做到的呢

## 按照不同隔离级别，隔离性越高，数据越安全，但性能越低。怎样实现隔离性：一种是**读写**锁，即当前事务将所有涉及操作的对象加锁，每次读操作需要获取一个共享锁，每次写操作需要获取一个写锁操作完成后释放给其它对象使用；一种是乐观锁，即不同的事务可以同时看到同一对象（一般是数据行）的不同历史版本。多版本并发控制（MVCC）：**类似快照的方式将数据保存下来，这样读锁就和写锁不冲突了，这种方式真正实现了非阻塞读，只有在写操作时才需要加行级锁，因此并发效率更高**。

隔离性是指，多个用户的并发事务访问同一个数据库时，一个用户的事务不应该被其他用户的事务干扰，多个并发事务之间要相互隔离。

33、数据库的隔离级别（重点）

**读未提交（**Read uncommitted）：

这种事务隔离级别下，select语句不加锁。

此时，可能读取到不一致的数据，即“读脏 ”。这是并发最高，一致性最差的隔离级别。

2.**读已提交**（Read committed）：

可避免 脏读 的发生。

在互联网大数据量，高并发量的场景下，几乎 不会使用 上述两种隔离级别。

3.**可重复读**（Repeatable read）：RR级别

MySql默认隔离级别。

可避免 脏读 、不可重复读 的发生。

4.**串行化（**Serializable ）：SI级别

可避免 脏读、不可重复读、幻读 的发生。完全的隔离性，在语义上相当于事务的执行没有并发一样；

34、说一下事务的属性

**原子性（atomicity）、一致性（consistency、隔离性**（isolation）、**持久性**（durability）

**隔离性**（isolation）：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

35、事务的原子性是怎么保证的？

**日志的管理和回滚，**将所有对数据的更新操作都写入日志，如果一个事务中的一部分操作已经成功，但以后的操作，由于断电/系统崩溃/其它的软硬件错误而无法继续，则通过回溯日志，将已经执行成功的操作撤销，从而达到“全部操作失败”的目的。

36、事务的提交过程是怎么提交的呢?

开启binlog后事务分成两阶段提交分别是prepare阶段和commit阶段

先记录 undo/redo log，确保日志刷到磁盘上持久存储。

更新数据记录，缓存操作并异步刷盘。

将事务日志持久化到 binlog。

提交事务，在 redo log 中写入commit记录

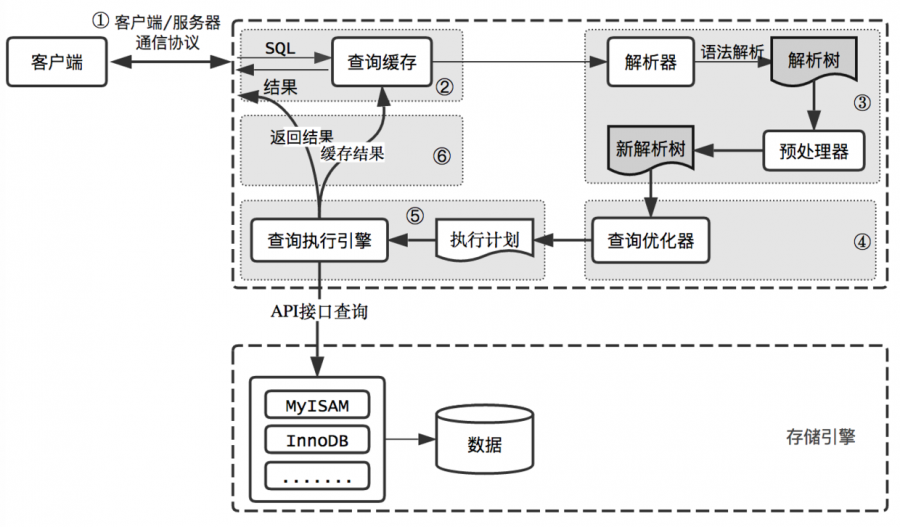
37、幻读

幻读是事务非独立执行时发生的一种现象。**指当一个事物读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。**例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

38、MySQL的存储引擎：MyISAM、InnoDB、Heap(Memory)、NDB

Myisam是Mysql的默认存储引擎，每个MyISAM在磁盘上存储成三个文件，文件名都和表名相同；InnoDB存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。

39、mysql的查询过程是什么样的



40、.MySQL的join是怎么实现的,底层实现

mysql只支持一种join算法：Nested-Loop Join（嵌套循环连接），但Nested-Loop Join有三种变种；

1. ****Simple Nested-Loop Join，****左右列
2. ****Index Nested-Loop Join（索引嵌套）****要求非驱动表（匹配表s）上有索引，可以通过索引来减少比较，加速查询
3. ****Block Nested-Loop Join，如果****没有索引，将驱动表的所有join相关的列都先缓存到join buffer中，然后批量与匹配表进行匹配

41、如果MySQL里面有一条查询跑的比较慢,你要怎么找呢?

**通过查询日志，**写一个包含所有执行时间超过long\_query\_time 秒的SQL语句的日志文件

**explain来了解SQL执行的状态**

**show processlist 命令的会生成**state列

42、如果它已经走了索引但是还是很慢怎么办呢

使用了主键索引，也有可能是全表扫描，遍历了整个主键索引树；

全索引扫描：指的是语句扫描了整个普通索引；

另外，语句执行速度的快慢与扫描的行数息息相关，索引要有效，就要提高索引的过滤性，减少语句的扫描行数

InnoDB中，数据都是存储在主键索引上的，而普通索引上只记录了普通索引与对应主键索引的映射关系

43、分库分表的理解

单表数据达到千万级别，甚至过亿的量，select个半天都出不来数据，**减小数据库的压力，缩短表的操作时间**。

**垂直切分**（纵向切分），是对不同的表进行切分；**水平切分**（横向切分），是对同一个表中的数据进行切分

44、spark的流处理原理是怎样的呢

Spark 作为一个基于内存的分布式计算引擎

Spark Streaming对于Spark Core进行了API的封装和扩展，接受时时输入数据流，并将数据分批次，然后由Spark engine处理，以批量生成最终的结果流。核心Spark API的扩展，支持可伸缩、高吞吐量、容错实时数据流处理。Spark Core、Spark SQL无缝整合。DStream中的每个RDD都包含了一个时间段内的数据

45、spark是怎么划分stage的呢

一个Job会被拆分为多组Task，每组任务被称为一个Stage。会根据RDD之间的依赖关系将DAG图划分为不同的阶段，对于窄依赖，由于partition依赖关系的确定性，partition的转换处理就可以在同一个线程里完成，窄依赖就被spark划分到同一个stage中，而对于宽依赖，只能等父RDD shuffle处理完成后，下一个stage才能开始接下来的计算。

从后往前推，遇到宽依赖就断开，划分为一个stage；遇到窄依赖就将这个RDD加入该stage中

46、spark rdd容错原理

47、RDD的不同依赖关系导致spark对不同的依赖关系有不同的处理方式。

48、对于宽依赖实质是指一个父RDD的分区会对应一个或多个子RDD多个分区，在此情况下，如果出现部分计算结果丢失，单一计算丢失的数据无法达到效果，便采用计算该步骤的所有数据，从而导致计算数据重复。

49、对于窄依赖而言，由于窄依赖的一个RDD 分区最多对应一个子RDD 分区，在此情况下出现计算结果丢失，由于计算结果只依赖父RDD相关数据有关，所以不需要计算全部数据，只需计算部分数据即可。

50、RDD容错四大要点：

51、spark框架层面的容错机制，主要是三大层面（调度层、RDD lineage、Checkpoint层）。

52、四大核心要点：

a、stage数据失败，高层DAGScheduler重试

b、spark计算中，**Task内部失败，底层调度器重试**。

c、RDD lineage中的宽依赖、窄依赖计算。

d、Checkpoint缓存。

53、spark的内存管理这块有没有了解过?

Spark 为存储内存和执行内存的管理提供了统一的接口——MemoryManager，同一个 Executor 内的任务都调用这个接口的方法来申请或释放内存；在调用这些方法时都需要指定其内存模式（MemoryMode

在 Spark 最初采用的静态内存管理机制下，存储内存、执行内存和其他内存的大小在 Spark 应用程序运行期间均为固定的，但用户可以应用程序启动前进行配置

统一内存管理机制，与静态内存管理的区别在于存储内存和执行内存共享同一块空间，可以动态占用对方的空闲区域

54、C++11允许你使用nullptr取代0或NULL，nullptr会被自动转换为各种pointer类型，std::nullptr\_t是一种基础数据类型，定义的变量必须初始化，nullptr属于一种常量

55、IO：

## IO请求一般是请求读写操作，有两个阶段：等待资源阶段，资源被上一个使用者使用没有被释放时，等待，使用资源阶段。在等待数据阶段：阻塞IO： 资源不可用时，IO请求一直阻塞，直到反馈结果；非阻塞IO：资源不可用时，IO请求离开返回，返回数据标识资源不可用。在使用资源阶段，IO分为同步IO和异步IO，同步IO：应用阻塞在发送或接收数据的状态，直到数据成功传输或返回失败；异步IO：应用发送或接收数据后立刻返回，数据写入OS缓存，由OS完成数据发送或接收，并返回成功或失败的信息给应用。大部分的socket接口都是阻塞型的。所谓阻塞型接口是指系统调用（一般是IO接口）不返回调用结果并让当前线程一直阻塞，只有当该系统调用获得结果或者超时出错时才返回。服务端面对多连接时1.一个简单的改进方案是在服务器端使用多线程（或多进程），pthread\_create ()创建新线程，fork()创建新进程。“线程池”旨在减少创建和销毁线程的频率。2.使用非阻塞，设置非阻塞*fcntl( fd, F\_SETFL, O\_NONBLOCK );*循环的调用recv()，根据返回值判断是否成功操作，但是CPU占用率提高了。很少用。3.多路复用，使用select()、epoll(),不断的轮询所负责的所有socket，当某个socket有数据到达了，就通知用户进程。select()同时处理多个connection。 int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds,struct timeval \*timeout)4.事件驱动io，libevent库代替了select()，当内核就绪时以信号通知进程。5。异步io.

56、服务端需要管理多个客户端连接，Select 和Epoll方法。每次调用 Select 都需要将进程加入到所有监视 Socket 的等待队列，每次唤醒都需要从每个队列中移除，进程被唤醒后，程序并不知道哪些 Socket 收到数据，还需要遍历一次。**Epoll将“维护监视队列”和“进程阻塞”分离，在 Select 和 Poll 的基础上引入了 eventpoll 作为中间层，使用了红黑树作为数据结构。**如果内核维护一个“就绪列表”，引用收到数据的 Socket，就能避免遍历。收到数据的 Sock2 和 Sock3 被就绪列表 Rdlist 所引用，当进程被唤醒后，只要获取 Rdlist 的内容，就能够知道哪些 Socket 收到数据。当程序执行到 epoll\_wait 时，如果 Rdlist 已经引用了 Socket，那么 epoll\_wait 直接返回，如果 Rdlist 为空，阻塞进程。

57、为什么IO多路复用要配合非阻塞IO

检查到这个新分节检验和错误，然后丢弃这个分节，这时候调用read则无数据可读，如果socket没有被设置nonblocking，此read将阻塞当前线程。如果是阻塞，在多进程同时监听一个socket，只有一个进程可以accept，别的都会block。

58、怎么创建进程的，fork()，两种方式1.父进程与子进程并发执行，2.父进程等待，直到某个或全部子进程执行完，用wait()/waitpid()系统调用。子进程的产生都需要拷贝父进程的数据作为起始运行状态。僵尸进程是子进程结束，父进程没有wait()释放。孤儿进程是父进程结束，子进程还没结束，父进程会改成init进程。

59、fork返回值是什么？在子进程中返回值是0,在父进程中返回进程ID。**在一个程序的一次执行中，这当然是不可能的；但是你看到的两行输出是来自两个进程，这两个进程来自同一个程序的两次执行。这2个进程共享代码空间，但是数据 空间是互相独立的，子进程数据空间中的内容是父进程的完整拷贝。**子进程与父进程拥有相同的代码空间，程序指针寄存器eip指向相同的下一条指令地址

60、elf执行的具体步骤，加载二进制，链接过程，执行：start( )->\_\_libc\_start\_main( )，\_\_libc\_start\_main( )->init( )->main( )->fini( )->rtld\_fini( )->exit( )

\_start从由内核设置的栈中获取参数和环境变量信息，\_libc\_start\_main初始化必要的数据结构，尤其是C库（比如malloc）和线程环境，然后调用用户的main函数，main函数的返回值由\_\_libc\_start\_main( )接收，并传给exit。

61、raft服务器数目选择，Raft协议会将需要保证一致性的数据保存在多台服务器上，领导者，候选人和追随者，节点 A 成为候选者后，向其他节点发送请求投票 RPC 信息，请它们选举自己为领导者，节点 B 和 节点 C 接收到节点 A 发送的请求投票信息后，在编号为 1 的这届任期内，还没有进行过投票，就把选票投给节点 A，并增加自己的任期编号；节点 A 收到 3 次投票，得到了大多数节点的投票，从候选者成为本届任期内的新的领导者，节点 A 作为领导者，固定的时间间隔给 节点 B 和节点 C 发送心跳信息。在一次选举中，每一个服务器节点最多会对一个任期编号投出一张选票。选举安全、仅领导者追加、日志匹配、领导者完整性。