文本复制检测报告单(全文标明引文)

№:ADBD2018R_201805301531272018053015482144017393840 2 检测时间:2018-05-30 15:48:21

检测文献: 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现

作者: 徐浩

检测范围: 中国学术期刊网络出版总库

中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库

中国重要会议论文全文数据库

中国重要报纸全文数据库

中国专利全文数据库

图书资源

优先出版文献库

大学生论文联合比对库

互联网资源(包含贴吧等论坛资源)

英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)

港澳台学术文献库

互联网文档资源

CNKI大成编客-原创作品库

个人比对库

时间范围: 1900-01-01至2018-05-30

可能已提前检测，检测时间：2018/5/9 14:49:02，检测结果：41.8%

检测结果

总文字复制比：0.5% 跨语言检测结果：0%

去除引用文献复制比：0.5% 去除本人已发表文献复制比：0.5%

单篇最大文字复制比：0.5%

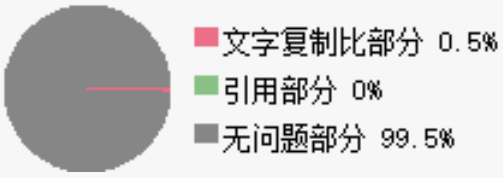
重复字数： [174] 总段落数： [8]

总字数： [34404] 疑似段落数： [1]

单篇最大重复字数： [171] 前部重合字数： [0]

疑似段落最大重合字数：[174] 后部重合字数： [174]

疑似段落最小重合字数：[174]



指 标：疑似剽窃观点 疑似剽窃文字表述 疑似自我剽窃 疑似整体剽窃 过度引用

表格： 0 公式： 0 疑似文字的图片： 0 脚注与尾注：0

（注释：无问题部分 文字复制比部分 引用部分）

1. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第1部分

总字数：3628

相似文献列表 文字复制比：0%(0) 疑似剽窃观点：(0)

原文内容 红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

吉林大学学士学位论文（设计）承诺书

本人郑重承诺：所呈交的学士学位毕业论文（设计），是本人在指导教师的指导下，独立进行实验、设计、调研等工作基础上取得的成果。除文中已经注明引用的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写的作品成果。对本人实验或设计中做出重要贡献的个人或集体，均已在文中以明确的方式注明。本人完全意识到本承诺书的法律结果由本人承担。

学士学位论文（设计）作者签名：

2018年5月20日

基于Hadoop的论坛日志分析系统的设计与实现

摘要

从20世纪60年代开始，人类进入信息时代，各种信息呈指数增长。面对如此繁多的信息，如何保存并提取有效信息成了当务之急，Hadoop应运而生。

当今时代趋势是智能化，而智能化是建立在大量数据的分析之上的。个性化广告推荐，海量日志处理，云计算等都需要大数据作为支撑，因而大数据技术是顺应时代潮流的，是不可或缺的。

Hadoop是一个分布式数据存储与分析系统，其中的HDFS文件系统支持对大量数据的存储；MapReduce模型借助映射和规约支持对大量数据的分析；Yarn是一个资源管理器，负责进行资源管理调度。三者各司其职，共同完成数据分析任务。另外，Hadoop往往需需要依赖Zookeeper实现分布式集群。

大数据也因其可扩展性而广受关注，与数字汇流的结合，和物联网，区块链等未来科技的相辅相成，都可圈可点，未来，必定万物互联的世界，不仅仅是手机电脑，这些电子设备，还有智能家居，出行设备，人所涉及到的，都会连接，由此产生的大量数据，绝对价值连城。具体点，增强现实，跟虚拟现实必将普及，并且走进大众，到时候，也将并入互联网，也将产生数据，有数据就需要分析提取；语音识别，更加人性化的科技，也是建立在大量语音样本的基础之上。所以呢，大数据掌握着未来并不是没有道理。

本文将基于Hadoop平台，搭建一个高可用、高可靠的分布式集群，借助Hadoop及其衍生产品做一个论坛日志分析项目。利用Hadoop对日志清洗并进行多维分析，同时提供对原始数据和清洗后数据的查看功能，最后会通过MPAndroidChart对结果进行图表展示。

关键字：Hadoop ，分布式 ， MPAndroidChart

Design and Implementation of Forum Log Analysis System Based on Hadoop

Abstract

Since the 1960s, mankind entered the information age, with exponential growth of various kinds of information. The

face of so many messages, how to save and extract valid information has become a top priority, Hadoop came into being.

The current trend in the world is intellectualization, and intelligence is based on the analysis of large amounts of data. Personalized advertising recommendations, massive log processing, cloud computing and so require big data as a support, so big data technology is to adapt to the trend of the times, is indispensable.

Hadoop is a distributed data storage and analysis system in which HDFS file systems support the storage of large amounts of data; MapReduce models support the analysis of large amounts of data with mapping and reducing; and Yarn is a resource manager responsible for resource management scheduling. Three of them perform their duties and work together to complete the task of data analysis. In addition, Hadoop often needs to rely on Zookeeper for distributed clustering.

Big data is also attracting attention due to its scalability. The combination of digital convergence and future technologies such as the Internet of Things and blockchain are all worthy of mutual respect. In the future, the world of things will certainly be more than a mobile phone. Computers, these electronic devices, as well as smart homes, travel devices, and people involved, will all be connected. The resulting large amounts of data are absolutely valuable. Specific points, augmented reality, and virtual reality will surely become popular, and they will enter the public. At that time, they will also be integrated into the Internet. Data will also be generated, and data will need to be extracted and extracted. Speech recognition and more humane technology will also be established. Based on a large number of speech samples. Therefore, it is not unreasonable for big data to hold the future.

This article will be based on the Hadoop platform to build a highly available, highly reliable distributed cluster, using Hadoop and its derivatives to make a forum log analysis project. Use Hadoop log cleaning and multi-dimensional analysis, while providing raw data and cleaned data query function, and finally the results will be charted through the MPAndroidChart.

Keywords：Hadoop ，distributed ，MPAndroidChart

目录

2. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第2部分

总字数：723

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第1章绪论

1.1 研究背景及国内外现状

随着科技的高速发展,社会各个领域的数据呈几何式增长.尽管这些数据量大而杂,却有极大用处.为了快速而有效地挖掘这些价值,Hadoop框架应运而生.

Hadoop是阿帕奇下的一个处理frame,可以通过搭建分布式集群实现对大量数据的存储和计算.

Hadoop长于日志的analyze,脸书曾经用过这frame.淘宝搜索也用了Hadoop的相关技术.

Hadoop是一个分布式系统。主要是用来进行海量数据存储，海量数据分析计算的。

研究历史：

06年3月，Map/Reduce 和 Nutch Distributed File System (NDFS) 分别被归入称为 Hadoop 的这个项目中。

Hadoop不同于以往的文件系统，它可以并行处理数据，对数据处理速度的增加比较明朗。

Hadoop其实来自于谷歌的其中某个MapReduce项目编程包，这项目是会把一个应用程序解析成mutiple的并行指令，并能跨节点计算海量数据集。Hadoop刚开始的时候只与一些网站索引有关，但之后很快的发展壮大。

研究现状：

Hadoop如今是大数据分析的重要工具，Hadoop及其衍生产品（Hbase，Hive等）已经形成一个完整的大数据分析体系，被大量互联网公司使用。

截止到2018年3月1日，最新版本为3.0.0，github的stars量5.9k，活跃量惊为天人呀。

1.2主要研究内容

本论文将利用Hadoop框架搭建一个分布式日志分析系统,分析GB级别的日志信息,日志内容是某论坛的数据信息.通过日志分析出论坛用户的浏览行为,对论坛的版块调整等提供参考.

1.3论文结构

3. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第3部分

总字数：365

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第一章介绍研究背景及发展趋势；

第二章主要介绍Hadoop核心知识：HDFS文件存储system，MapReduce这个计算框架，Yarn这个资源调配。同时包括其相关生态体系：zookeeper；数据分析处理Hive；数据导进来导出去Flume和sqoop。最后介绍数据展示模块：利用android实现数据呈现，GreenDao数据库操作，MPAndroidChart这个开源的图表显示的框架等；

第三章是系统设计阶段。主要步骤有：数据导入，数据清洗，数据分析，报表展示。囊括了一个基本完整的数据分析链；

第四章是系统测试阶段。分为功能测试和性能测试。功能测试目标是实现对数据的分析处理并展示处理结果；性能测试一定的得测试系统的可依赖性和可用性，测试在某些机器宕机的情况下是否还能正常提供服务。

最后是参考文献和致谢。

4. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第4部分

总字数：10630

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第2章技术储备

2.1Hadoop架构

Hadoop的特点

可扩展性：存储PB这种级别的数据不在话下。

经济：数据将通过由很常见的机器组成的服务器场进行分配和处理。 服务器可以按照需求组成几百乃至几千节点都没问

题。

高效率：因为是一个集群，相互之间是并行执行一个work的，这就使得效率异乎寻常的高。

非常的可靠：任务执行失败的话，Hadoop会自己从头布置任务的执行，另外呀，他还有不少的副本。

2.1.1HDFS

分布式文件系统来源及特点：

随着数据量越来越大，一台机器负担不起，所以多台机器进行组装，而多台机器之间需要协调机制，并生成一个分布式文件管理的system。

它是一种文件系统，可以通过网络传输共享文件资源，并且可以同时共享文件和存储空间。

渗透性。 虽然通过网络访问资源，但它会让用户感觉像访问本地磁盘。

容错。 纵然系统中的某些节点处在脱机的状态，例如某些节点已关闭，系统仍可以在没有数据丢失的情况下运行。即使数据丢失，也存在恢复机制。

分布式文件的管理system不少，hdfs算是其中一员了。 它的特点是它适合一次写多个查询，不支持并发写入data，不是很适合小文件。这个系统被使用的很广泛了。

HDFS shell命令

-help [cmd] //帮助信息

-ls(r) <path> //显示所有文件

-text <src> //在终端显示文件内容

-du(s) <path> //文件大小

-getmerge <src> <localdst> //将源目录中的所有文件排序合并到一个文件中

-count[-q] <path> //显示目录中文件数量

-moveFromLocal //从本地文件移动到hdfs

-mv <src> <dst> //移动多个文件到目标目录

-touchz <path> //创建一个空文件

-rm(r) //删除文件(夹)

-put <localsrc> <dst> //本地文件复制到hdfs

-copyFromLocal //同put

-cat <src> //在终端显示文件内容

-get [-ignoreCrc] <src> <localdst> //复制文件到本地，可以忽略crc校验

-copyToLocal [-ignoreCrc] <src> <localdst> //复制到本地

-moveToLocal <src> <localdst>

-cp <src> <dst> //复制多个文件到目标目录

NameNode Metadata

NameNode 的元数据格式如下

(FileName, replicas, block-ids,id2host...)

FileName--》文件名字（HDFS系统下的文件全路径）

Replicas--》副本数量

block-ids--》组成文件的块数

id2host--》是一个数组，表述每个块所在的主机名字

文件有这些内容呀：

fsimage:一种元数据的信息。它呢，主要是存储某时的NameNode信息。（与NameNode内存互为镜像）；

edits:日志文件，记录用户的写操作记录；

fstime:保存最近一次checkpoint的time

NameNode元数据管理机制：

为了快速访问及提供可靠性，Hadoop提供了一套元数据管理机制

NameNode内存时刻保存所有元数据信息，提供给client读取操作。

当client进行写操作时，NameNode首先会把写信息存入edits日志中，当写操作成功，edits日志并入内存，保证NameNode最新。

NameNode同时存在磁盘文件fsimage，是内存的镜像文件，当edits文件写满，进行checkpoint操作。

checkpoint需要合并fsimage和edits，需要cpu资源，因此需要单独开启一个进程，也就是secondarynamenode。secondarynamenode首先通知namenode停止写入edits文件，新建edits.new暂时替代edits功能；然后通过网络下载fsimage文件和edits文件；之后进行合并操作，产生新文件fsimage_checkpoint；最后网络传输给namenode fsimage_checkpoint替代旧的fsimage，edits_new更名为edits，一次合并结束。

HDFS的架构

主从结构
NN节点的信息：
它主要是来管理文件系统的取名空间，存贮一些document的数据的
管理者这个系统的document和一些目录之类的结构，保证着某些映射关系
聆听操作者的一些请求，写下doucment所在数据块的节点信息
维护文件系统的目录结构
管理文件与block之间关系，block与datanode之间关系
DataNode
存储文件
文件被分成block存储在磁盘上
为保证数据安全，文件会有多个副本
存储并检索数据块
向NameNode更新数据块信息
HDFS优点：
适合大文件存储，有副本策略可以构建在廉价机器，有容错机制
支持流式数据访问，一次写入，多次读取
HDFS缺点：
不适合小文件存储
不适合文件随机修改
不支持随机读写
HDFS写流程：
客户端请求NameNode写数据，NameNode返回可写入的数据节点，客户端对数据进行分块，逐块写入数据节点，数据节点写入成功，返回NameNode，NameNode返回客户端信息，客户端写入剩余数据块。数据的备份是数据节点自动完成的。
HD FS的读流程：
客户端请求NameNode读数据，NameNode返回数据节点（距离最近的数据节点），客户端读取数据。
RPC实现机制：
RPC——就是一种远端调用某个类方法的一种方式
服务端有一个业务类实现特定接口，并实现接口的方法，
Socket作为代理监听指定接口发送来的请求，并根据接口协议调用指定业务方法，返回给调用端；
调用端同时拥有该接口，并通过socket代理出传入接口协议，版本信息，方法参数等信息，远程调用服务端，并获取结果。
RPC使用：
服务端代码
接口
```
public interface softInterface {
//指定版本号
public static final long versionID=1L;
//接口方法
public String dosomething(String arg1,String arg2);
}
```
实现类
```
public class softInterfaceImpl implements softInterface {
@override
public String dosomething(String arg1,String arg2){
return “your args”+arg1+” ”+arg2;
}
}
```
服务主进程
```
public class LoginController {
```
Main函数的入口呀：
声明配置文件 conf；
```
Builder builder = new RPC.Builder(conf );
```

builder .setBindAddress(“extreme”);//主机名
builder .setPort(7777);//端口号
builder .setProtocol(softInterface .class);//设置接口协议
builder .setInstance(softInterfaceImpl .class);//设置实例
Server server = builder.build();
Server.start();}
客户端代码
接口
public interface softInterface {
//指定版本号
public static final long versionID=1L;
//接口方法
public String dosomething(String arg1,String arg2);
}
客户端主进程
public class DemoClient{
LoginServiceInterface proxy = RPC.getProxy(softInterface .class, 1L, new InetSocketAddress("extreme", 7777), new Configuration());
String result = proxy.login("hello", "world");
System.out.println(result);
}
NN高可用方案：
当集群只有一个NameNode节点时，虽然能够保证数据不丢失，但是一旦NN宕机，就不能提供服务了，可用性大大降低，因此多台NN节点就有了存在的意义。
本系统采用两台NN节点，不能让两台机器同时响应客户端请求，会造成读写不一致，读写覆盖等问题。可以选择让其中一台处于active状态，另一台处于standby状态。为了保证两台NN的无缝快速切换，需要让二者的元数据保持一致，qjournal由此产生，qjournal底层依赖于zookeeper实现，是一个分布式应用，其节点成为journalnode，主要作用就是保存edits文件，并提供高可靠性。当客户端向active的NN申请写数据，NN会向qjournal写入edits日志，standby的NN会经常读取qjournal上的文件，然后和自己的fsimage合并，这样就保证了两台NN的元数据同步问题。
两个NameNode节点切换时可能会出现这样的问题:NN1没有挂掉，NN2以为它挂了，于是自己也变为active，成为brain split。为了解决这问题，hadoop2.x之后有这样机制：每个NN都会存在zkfc进程，用来监控NN的状态，并写入qjournal信息，另一台会读取信息，一旦宕机，待定状态的NN首先发送kill指令给挂掉的NN，成功返回，自己启动；隔一段时间没有响应，执行自定义脚本（自己编写），这样很大程度降低了脑裂发生的概率。
2.1.2 MapReduce
分为四个阶段：
Split ，map ， shuffle ， reduce
序列化机制
如果想传输自定义bean，需要实现序列化
举例
public class Tobeserilizedclass implements WritableComparable<FlowBean>{
private String string;
private long num;
//对象数据序列化到流中
@Override
public void write(DataOutput out) throws IOException {
out.writeUTF(string字符串string);
out.writeLong(long型数据num);
..... }
//数据流中反序列出对象的数据
//必须跟序列化时的顺序保持一致
@Override
public void readFields(DataInput in) throws IOException {
string= in.readUTF();

num= in.readLong();

.....

}

}

自定义排序

举例

对Tobeserilizedclass 添加 implements WritableComparable<FlowBean>

实现方法

@Override

public int compareTo(Tobeserilizedclass o) {

return num>o.getNum()?-1:1;

}

Shuffle机制

Map任务数量由split切片大小决定（block也是影响因素），一个map会处理不同分组数据，而且数据量可能会非常大，内存放不下（可在配置文件中配置）于是便写入本地磁盘，写入的格式--》每个分组在一起（默认根据key排序），当出现多个磁盘片，shuffle机制使其进行合并操作，同一组在一起，产生一个大的磁盘片，之后根据分组情况传入各个对应的reduceTask中（期间可以先进行一下处理，就是combiner，这样可以提高传输速度）。MAPPmaster控制map任务和reduce任务分配在哪台主机，是一个协调工作者，首先，在某个Nodemanager下的容器里边指定map任务数量，一般和split切片相关，之后shuffle，就是对map处理完的结果进行分组，分组依据，重写的getPar方法的返回值确定，先自己整理:内存放不下，溢出到磁盘，传送之前，合并一下，一个组的在一起，默认key排好序。完成后向协调者发送成功消息，Master再找一个执行reduce任务，告诉它需要的map数据在哪个主机，文件名，等信息，reduce这个任务去下载（多个map），先合并一下下载来的数据，最后进行reduce任务，输出想要结果。

分组：

默认情况下map后的数据会进入同一组，这是由于HashPartitioner返回值问题，

Return （。。。。。）%(reduceTask.num)，%后边的数是1，所以只有一组。

实现自定义分组：

A.自定义Partitioner

public class ExtremePartitioner extends Partitioner<Text, KpiWritable> {

@Override

public int getPartition(Text key, KpiWritable value, int numPartitions) {

//分组逻辑，每组的code不同

Return code；

}

}

B.指定reduceTask任务数

job.setPartitionerClass(ExtremePartitioner .class);

job.setNumReduceTasks(分组数目);

2.1.3Yarn

Yarn提供资源调度服务

一个完整的任务过程：

Job.waitforCompletion（ ）提交后，RunJar进程请求RM进行资源分配，RM返回任务jar包及配置文件的提交目录，返回，然后么，RunJar这个进程就开始向指定文件夹（其实是向hdfs里面写数据）放文件，提供一个唯一的jbID，放到队列边，等待轮到自己，之后分发任务，根据当然是NM节点的状态，看看cpu占有率呀，内存使了多少呀，他就是一个整体的资源调度器，给NM开启一个container，这是个抽象的概念，本质呢，就是各种资源的综合，开启成功后，nodemanager相当于领取任务了。全局的分配就结束了，他会在某一台主机上开一个Appmaster进程，鉴于此，RM的使命结束，这个主人进程，了不得，它是细粒度的资源发配，是MR任务的主控制器。它，先向RM请求自己到底能主宰哪些主机呀，然后，就毫不客气地开始指派任务了，依据任务的情况，分配map任务主机，接着，开始处理数据了，map完成，再分配台主机，来个reduce任务，把数据传过来，处理，最后呀，结果出来，一个任务over，master开始想RM注销，收回那些资源。

2.2 Zookeeper分布式调度

背景：

不得不感慨，如今是技术boom的时代，互联网这个新兴行业发展贼快，一些企业对电脑的要求变得高高的，比如你的cpu速度呀，支持并发不呀，你能存多大的数据量呀，等等，都是应当tnhink的目标。

于是呢，在这种situation下，单兵作战之类的不适合了，集群--产生了，

企业的那些架构师都从小型集中转向集群，也就是分布式了。分布式其实概念也不麻烦，就是分-合的一种模式而已，比如吧，有一个job要处理一下，这个job非常复杂，一个单纯的想法就是切分小块呀，一小块一小块的handle它，对了，把这每个小块块发到一台电脑，让他们单独计算结果，最后呢，把结果汇总一下，

这个任务是不是就完成啦，这，就是分布式。

Zookeeper是Apache下的开源项目，目前GitHub的star数量为4.3k，并且一直有相关人员进行维护（修改时间一天之内），它主要实现高可靠的分布式协调。

Zookeeper的老东家，是雅虎，他的定义，high性能，一致性解决，分布式数据。说到底把，它就是封装了一下底层，这个底层呢，他会帮你把集群的主机应当协调的数据给归拢归拢。当然了，这个框架，非常好用的，因为接口实在是太人性化啦，简单容易上手。

他主要是：用于维护配置信息，起名，提供一下同步呀，一个组的服务，最广的用处就是分布式的协调了。

ZooKeeper的优势

1、源代码开放（必须在github上，不开源可惜了）

2、用的企业，多如牛毛，用的时间也不短了，经过了考验，比较完美的工业品。

3、用到的场景还是不少的：举个栗子把，Hadoop,HBase,Storm,Solr

典型应用场合：

1.广播一些消息，选择性接收，或者不收听

见名思意，一端发一些消息，另一端得到信息。

一般呢，两种方法：push model跟pull model，推就是服务器这边主动，拉就是消费者那边主动，主动，是指对消息的兴趣。Zk当然是两种都用。

服务器会发一些信息，订阅的人可以自己选喜欢啥消息，然后跟服务器吭一声，（就是在服务端注册个字符串），那天消息变了，服务器就和订阅者说一下，客户端就收到这个message了。

2.load leveling

A。Zk节点两种，一种永存的，不会无缘消失，一种有寿命，暂时的，节点会自检，发现自己不行了（无法访问，数据损坏，），马上通知ZK，然后自我报销（删除自己），ZK更新为最新的，全都好使，可用列表健壮性就不错了。

B.消费者要读写database时，先去ZK瞅瞅能用的数据库都有哪些，会得到一个可用的列表。

C.要是连接的不能用，消费者俩选择，再去Zk上瞅瞅可用列表，或者直接把这个无用的删了，换一个别的（在上次请求表里的）。

3.起名服务

就是给起名的服务，举例，数据库主键，有无规则字符串型，还有有规则增长型，各有千秋，自增的也就单个数据库了，集群里不好用，UUID适用后者，但是这玩意太长了，不好记，不好看。所以，ZK集二者之长，又能自增，又能有规则，ID就交给Zk处理了。

4.相互通信

集群里，最常见就是通信儿啦，我们得知道那个电脑还好使，网络可以用ping来试试，能发送bytes，哎，这俩能对上话，都好使，要不然，主机挂了。Zk有个大好处，初始化时，集群机器都屁颠屁颠注册自己信息，啥时候自己挂了，信息就没啦，这样看一个电脑好不好使，直接取Zookeeper取列表就行了，大大简化了复杂度，这个设计贼人性化。

Zookeeper基础知识：

1.会话

就是消费者生产者搭个话，建立了一个连接，文雅点就是一个session，其实，是一个厂的TCP的连接哩。

维护会话就是网络通信，可以through心跳这个机制保持长期有效会话，客户端可以发起连接，服务端来响应。

2.cluster角色

领导，职员，观察者

Leader：服务器核心，协调数据，比如，写数据，一半以上都写进去就成功了。具体点，写流程大致这样，某个server接收客户端请求，准备write数据啦，这个ser发送给领导，领导再转发给其他职员，收集职员更新的情况，一半以上ok，领导就觉得，我这任务完成了，返回给ser结果，ser再依次应答给客户端。这样，一个写过程结束。

Observer：和follower差不多，不过不参加选举。为啥要存在这么个东西？它，其实是3.3后边出来的，需求旺盛，应当扩展，加server把，这一加就出问题了，选举时间太慢了，这可不行，影响效率呢，怎么办，折中了，加服务器节点目的就是更快响应客户端请求，好的，基于此，我再新来一种节点类型，它只提供服务，响应客户端啦，传递给领导写信息啦，一个不缺，但是，我没有选举的这个功能。这样，一举两得了。

Follower:集群开启，选个领导，这就是election，具体执行者，读数据，写数据（传给领导，再写入），脏活累活都它干了。

3节点type

节点2种

1.1台加入到集群的电脑就是一个node

2.节点存储信息是一种目录结构，下边可以来个znode，还可以来个真实数据。但有点需要注意下，那种朝生暮死的绝对不会有子节点，必须的，不然子节点也成了临时的，显然是及其不合适。这个信息呀，不是一种，可以存多个version ，想得到的时候，除了带上路径，还得捎带着版本号。你这个节点类型不得随便改，要不然就乱套了，所以，创建完后就不得改了。还有，读，一次读个够，不能读一半。那个朝夕不保类型的节点，客户端访问后，直接就kill了，那种永久的，除非你点名要除掉，不然会保存的灰常好。

Znode里边是能保存少量数据的，大量的不行，存储空间不太够。

悲观锁与乐观锁

悲观锁有个小名，叫并发锁定。觉得两个进程同时访问数据一定不是好事，一定冲突，于是，乖乖按流程来，一个一个进，这样就没问题了。适用于数据经常性改，竞争灰常激烈情形。

乐观锁，觉得，两个以上来就来吧，你们对数据的访问对彼此没啥大影响，这样访问速度当然快了不少，但基数大了，肯定出事，于是想个弥补办法，读数据之前，我先看看是不是我预期想读的数据版本，是，没问题，继续操作，并加上自己这次修改的版本号；不是，可以抛个异常，具体处理交给异常的流程来整。当然，还可以回退这个事务，重来一次。

Zookeeper的安装：

1.上传安装包

2.解压

3.配置（先在一台节点上配置）

3.1添加一个zoo.cfg配置文件

在$ZOOKEEPER/conf目录下

mv zoo_sample.cfg zoo.cfg

3.2修改配置文件（zoo.cfg）

dataDir=/home/hadoop/app/zookeeper-3.4.5/data

server.1=zk01:2888:3888

server.2=zk02:2888:3888

server.3=zk03:2888:3888

3.3在（dataDir=/home/hadoop/app/zookeeper-3.4.5/data）创建一个myid文件，里面内容是server.N中的N（server.2里面内容为2）

echo "1" > myid

3.4将配置好的zk拷贝到其他节点

scp -r /../zookeeper-3.4.5/ zk02:/../

scp -r /../zookeeper-3.4.5/ zk03:/../

3.5注意：在其他节点上一定要修改myid的内容

在zk02应该讲myid的内容改为2（echo "2" > myid）

在zk03应该讲myid的内容改为3（echo "3" > myid）

4.启动集群

分别启动zk

./zkServer.sh start

查看启动状态

./zkServer.sh status

在集群中，必须有一半以上的主机启动才可以

2.3 HBase及Hive

2.3.1 HBase

HBase是阿帕奇的开源项目，是分布式，可扩展，非关系型，大数据数据库，追根溯源，Google的Bigtable论文。

HBase的两大特点：海量数据存储，百亿行乘以百万列；准实时查询，查询速度灰常快。

HBase，实际业务应用场景：交通行业，摄像头拍照信息；金融行业，金钱交易信息；电商行业，买家卖家交易信息，物流信息；移动行业，通话，地理位置等信息。

HBase的特点：

容量大：相比于传统关系型数据库，容量很大。数据矩阵横向和纵向维度所支持的数据集都非常有弹性；

面向列（和普通关系型不一样）：其数据在数据库是按照列存储，这样在查询某几个字段的时候，大大减少了查询时间；

多版本：每一个列都有多个version；

稀疏性：为空的列不占用存储空间，表的设计非常稀疏；

扩展性：底层依赖于HDFS，如果存储空间不够，可以动态增加Datanode节点；

高可用性：因为是基于HDFS的，所以有多个备份数据，具备分布式的特点；

高性能：底层，随机定向的读取，该性能针对rowkey的查询，达到毫秒级别。
综合来看，HBase列动态增加，数据自动切分，高并发读写，不支持复杂的条件查询。
HBase架构体系：
A.一般要搭配使用Zk和hdfs
Zookeeper用处：
确保运行的时候，集群的running master有且只能存在这么一个（单例）
地址的寻找，都由他保管着
是一个监管者，监控着Region Server的一举一动，通报master他的活动信息
保存表的相关数据，表名是啥呀，表的列有哪些呀（列族）。
B.俩主进程：
Master：
管理进程，监听regionServer的状态消息，和Zookeeper通消息。
Master 爱启动几个Hmaster就启动几个（前提是资源足够），election当然得保证这个单例的存在了。
regionServer：
服务进程，每张表被分为多个区域，分别对应一个regionServer。负责和HDFS交互（读取写入数据），同时将状态信息传给Master。

5. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第5部分

总字数：12174

相似文献列表　文字复制比：1.4%(174)　疑似剽窃观点：(0)

1	201605210929343976_孔舒晨_基于hadoop的数据分析	1.4%（171）
	孔舒晨 -《大学生论文联合比对库》- 2016-05-21	是否引证：否
2	912101400113_黄泽涛_基于Hadoop的制造过程大数据存储平台构建	0.7%（83）
	黄泽涛 -《大学生论文联合比对库》- 2016-05-26	是否引证：否
3	曾鹏飞-201210913247-基于HADOOP集群的日志分析系统论文-1242班-焦合军	0.6%（70）
	曾鹏飞 -《大学生论文联合比对库》- 2016-05-27	是否引证：否
4	曾鹏飞-201210913247-基于HADOOP集群的日志分析系统论文-1242班-焦合军	0.6%（70）
	曾鹏飞 -《大学生论文联合比对库》- 2016-05-29	是否引证：否
5	20125023_高玉洁_基于mapreduce的海量数据的处理方法与应用研究	0.6%（70）
	高玉洁 -《大学生论文联合比对库》- 2016-05-16	是否引证：否
6	基于hadoop的数据分析	0.6%（70）
	孔舒晨 -《大学生论文联合比对库》- 2016-05-17	是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

HBase的数据模型：
基于列族存储，不需要提前指定有哪些列，只需指定列族即可。
一张表列族不会超过5个，每个列族的列数没有限制，列只有在插入数据后存在，列在列族中有序。
主键是用来检索记录的主键，访问hbase table中的行，只有三种方式
通过单个row key访问
通过row key的range
全表扫描
HBase的配置安装：
1.下载安装包，解压；
2.改一下hbase- env.sh
//jdk路径
export JAVA_HOME=/home/hadoop/java/jdk1.7.0_56
//人家带着一个，这里得使自己配滴
3.添加一个文件regionservers
指定主机名字
2.3.2 Hive
Hive是阿帕奇的，数据仓库管理软件，可以使用HQL语言管理驻留在分布式系统的大型数据集，可以将结构投影到已存储的数据上因此Hive一般都是执行各种查询操作，而不是进行更新删除等处理。
利用JDBC驱动，客户端连接到Hive，其工作原理是，把HQL语句转化为MapReduce任务，其本质是建立在HDFS上的一

个数据库。

数据仓库一般不随时间的变化而变化。包含三种操作：ETL。Extract，提取信息；Transform，转换；load，加载；数据仓库模型可分为两种：星型模型，基于一个中心构建的框架；星型模型的进一步演化，雪花模型，对附属部件的进一步细化。

Hive允许开发者自己编写MapReduce逻辑来应对内建程序无法处理的复杂操作。Hive的表其实就是HDFS的目录。Hive将元数据存储在metastore中，数据库默认是dergby数据库，但是这种数据库在不同目录的数据表不共享，因此一般使用mysql，oracle等数据库。Hive的元数据有这些：表滴名，列滴属性，表滴属性（内部表，外部表），table相应目录（数据所在位置），表的分区属性等。HQL执行过程如下：通过三个组件完成任务，解析器，编译器和优化器。解析器负责解析HQL语句，编译器产生执行任务，优化器负责对生成的任务进行优化，加快MapReduce的执行效率。

Hive体系结构：大致分为三层。最底层：linux；中间层：Hadoop集群层，包括NameNode，DataNode，JobTracker等。HQL解析完成后产生的MapReduce任务会分发给JobTracker，然后把执行结果保存到DataNode中；上层：Hive层。包括Client端口，通过驱动访问的Thrift Server（java程序编写），web 控制台（只可以进行查询操作）。

Hive的安装分为三种模式：嵌入模式，使用默认derby数据库，只有一个连接，适用demo；本地模式，使用mysql本地数据库；远程模式，数据库和Hive不在同一台主机。本地模式配置方式：官网下载hive压缩包；解压；把mysql驱动包放到lib目录下；配置conf目录下的hive-site.xml(默认hive-default.xml.template)，

因为Hive是基于Hadoop的，因此使用Hive之前需要开启Hadoop集群。Hive的表在HDFS的存储位置：/user/hive/warehouse/...

如果删除/user/hive/warehouse/，Hive执行会出错。

HQL大部分情况下都会转换成MapReduce程序。当查询表的所有数据时，不会转换；可以使用source命令执行一个sql文件;进入到静默模式:hive -S(只显示结果,不显示调试信息)；linux命令行界面执行HQL方法：hive -e。

Hive的数据类型：

简单类型：整数类型，浮点类型，字符型（varchar指定最长长度，char固定长度），布尔型；

复杂数据类型：数组类型，集合类型，结构类型。

Hive的数据模型：

内部表：表对应一个目录。所有的table之中的数据都保存在这个目录下。删除表，目录中的数据也被删除；

Example了:create table demo_1 (id int,name varchar(20)) 按行读取行之间分隔符是 ' '

分区表：

partition和目录文件一一对应，数据都在目录下边

Example:create table demo_2 (id int,name varchar(20)) partitioned by (year string) row format delimited fields terminated by ' '

外部表：

外部表只是一个链接。删除外部表不会删除对应数据。

示例：create external table demo_3 (id int,name varchar(20)) row format delimited fields terminated by ' '

桶表：

对数据进行Hash取值，放到不同的文件中存储

视图：

是一种虚表，一种逻辑概念，可以跨越多张表。

视图可以监护复杂查询。Hive不支持物化视图。

Hive进行数据导入：

A.把本地文件导入到Hive

语句：load data local inpath 本地路径（既可以是单个文件也可以是文件目录）（overwrite关键字表示是否覆盖表原来的数据）into 表名

注意事项：表的分隔符和导入数据的分隔符要一致（表的默认分隔符是制表符）。

B.把HDFS上的文件导入到Hive

语句：load data inpath HDFS路径（既可以是单个文件也可以是文件目录）（overwrite关键字表示是否覆盖表原来的数据）into 表名

C.把文件导入到Hive的制定分区

语句：load data inpath HDFS路径（既可以是单个文件也可以是文件目录）（overwrite关键字表示是否覆盖表原来的数据）into 表名 partition（分区条件）

Hive的查询：

A.关于NULL

nvl函数--》nvl（prize，0）--》如果prize是null，则转换为0；

判断是否为空--》where prize is null。

B.关于Fetch Task

在Hive的0.10.0版本中，如果是简单的查询，则不用生成Mapreduce任务。
配置方法：
set hive.fetch.task.conversion=more
hive --hiveconf hive.fetch.task.conversion=more
修改hive-site.xml文件
Hive 0.14.0之后的版本已经中默认为more了,不需要修改fetch.task
C.使用过滤
模糊查询（_代表任意char，当然得转义啦）
含有null进行排序：升序排在前面；降序排在后面
Hive的函数：
A.数学函数
Round（四舍五入）
Ceil（上取个整）
Floor（下取整）
B.字符函数
Lower（小写）
Upper（大写）
Length（字符数）
Concat（拼接字符）
Substr（取子串）
Trim（去除前后空格）
Lpad，rpad（左右填充）
C.收集函数和转换函数
Size（map集合的大小）
Cast（转换数据类型）
Hive的表连接
分为四类：等值连接，不等值连接，外连接，自链接。
Hive的JDBC客户端操作
首先要启动hive的远程服务
Hive --service hiveserver
分为五步：（具体步骤和连接mysql数据库大同小异）
1.获取连接
2.创建运行环境
3.执行HQL
4.处理结果
5.释放资源
String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
Class.forName(驱动名字);
创建conn连接，必须指定url地址呀
Statement stmt = con.createStatement();
stmt.execute("HQL语句");//直接执行
ResultSet res = stmt.executeQuery(sql); //获取返回结果
Con.close();//如果不为空关闭
Stmt.close();//如果不为空关闭
Res.close();//如果不为空关闭
Hive的Thrift客户端操作
基于socket实现
Final TSocket socket = new TSocket (192.168.233.128,10000);
指定协议
最终拿到客户端的端口了
Socket.open();
Client.execute("HQL语句");
2.4 Sqoop

简介：

sqoop是在hadoop体系和关系型数据库之间进行数据传输的工具，原理很巧妙，把语句解析，产生mapreduce的任务，执行，得到期望的答案。

安装：

下载压缩包，解压。注意修改配置文件 sqoop-env.sh

export HADOOP_COMMON_HOME=/home/hadoop/app/hadoop-2.4.1（指定hadoop的主目录）

export HADOOP_MAPRED_HOME=/home/hadoop/app/hadoop-2.4.1（指定MapReduce主目录）

配置环境变量

export JAVA_HOME=/home/hadoop/app/jdk1.7.0_65

export HADOOP_HOME=/home/hadoop/app/hadoop-2.4.1

export SQOOP_HOME=/home/hadoop/app/sqoop

改变PATH变量的值，并重新赋给self

将数据库连接驱动拷贝到$SQOOP_HOME/lib里

使用Sqoop进行数据导入导出：

A.数据库里边数据导入到HDFS上

sqoop import

--connect jdbc:mysql://49.140.77.211:3306/datastore

--username root --password root

--table demo

--columns 'id, name,password'

--target-di r linux下某目录（导入hdfs后虚拟位置）

--规定分隔符 '\t'

-m 2（指定做导入处理时的map 任务数 ）

--where 'id>3'（增加的条件）

B.把HDFS数据导出到数据库

如果是远程登录，需要配置mysql的远程连接

GRANT ALL PRIVILEGES ON nn1.* TO 'root'@'49.140.77.211' IDENTIFIED BY 'root' WITH GRANT OPTION;

FLUSH PRIVILEGES;

2.5 Android

2.5.2 OkHttp

是一个网络框架，github star26.4k，隶属于square公司

特点：

支持HTTPS/HTTP2/WebSocket（在OkHttp3.7中已经剥离对Spdy的支持，变为大力支持HTTP2）

内部维护任务队列线程池，尤其支持并发访问

内部维护自己的连接池，支持多路复用，减少连接创建开销

socket支持极好路由，一个字，快

提供拦截器链（InterceptorChain），实现了请求与响应的分层处理(如透明GZIP压缩等)

架构层面如何设计的：

使用简易，俩方式，同步调一下和异步调一下，先说同步的：一开始，就加一个锁，判断调用过没有，如果调过了，没办法，抛出来啦一个exception，否则的话，

把那个布尔类型的变量executed设置一下，成为true（最开始的时候是false），

之后，借助客户端的executed方法，来执行实际操作，用什么来接收尼，Response对象呀，返回链的get方法获取，其实上边的executed方法是把调用加到队列里面，获得返回对象之后呢，finish代码块执行，再把这个从队列移除。

另外一个就是异步啦，异步的处理前半部分和同步一致，指定一个是否执行过的。标志位，之后，会调用AsyncCall的方法，这个类其实实现了runnable接口滴。

同许多框架类似，这个也是有拦截器的，而且这个拦截器是双向的，既可以拦截请求的，还可以拦截响应滴，神奇吧，设计者想的很全面呀。

下面来说一下okHttp的总架构把：

最上边肯定是接口层了，就是我们调用的接口，Call类，dispatcher，负责和程序员打交道的。

OkHttpClient是客户端的接口啊，每产生一个实例，调用了，这个实例都会维护自己独有的一套，缓存啦，任务队列啦，拦截器啦，连接池啦。所以开很多实例是消耗资源的，自己写应用的时候，来一个全局的就ok啦。每个Call代表依次请求，但他只是实现了接口，实际是产生了AscCall的实例，所以每次请求都是一个线程在执行。任务队列就是dispatcher，它负责把空闲的线程利用起来，执行execute方法。

再下一层是协议层了，okHttp是支持很多种协议的，HTTP2，https都支持，

Realconnnection是连接的实例，它统一了socket的连接，并支持多路复用，支持空闲时重复再用一下下

再往下是连接层，就是两个端口进行tcp连接，处理二者之间的参数信息等，

再往下是他的特色之一，缓存区，这个可以大大节省不必要的访问次数，

最后是IO层，就是真正的进行数据读写的地方了。这也算他的一大特点，因为IO效率实在是太高了，当然了，归功于他的库okio。

拦截器详解呀：

请求会经过每个拦截器，具体原理，依次拦截器执行拦截方法，最后返回response对象。拦截器的顺序及功能，RetryAndFollowUpInterceptor，一次连接失败后重新链接一下，需要重定向的时候，要不要复用一下当前的连接呀。BridgeInterceptor，处理解压任务，访问浏览器等，加上一个cookie的信息，还有，设置一下其他的http的报文信息。CacheInterceptor，提供缓存机制的相关处理，失效的话，直接删了就行。CallServerInterceptor，真正的请求信息。

任务队列：

符合队列特点呀，先进先出。

共有三种类型，等待中的异步请求，运行中的异步请求，运行中同步请求，自己有一个线程池，请求进来会分配线程池里的线程，60s空闲了，自动关闭。最出色的设计，即使请求失败了，一样会从任务队列移除。

Cache：

HTTP也有一些缓存策略，Expires，一个头信息，作用：标记资源的过期时间，请求在时间之内，就没必要再请求了，浪费时间，浪费资源，直接从上次的缓存拿。

请求端头， Last-Modified-Date，服务器告诉客户端，文件上次修改时间，再请求一下，时间没改，就直接拿缓存了。Etag表示文件有没有变化。

他是嵌入在拦截器中的，对应类名是CacheInterceptor。内部存在同文件系统交互的类，DiskLruCache，支持文件写。缓存策略，俩选择，直接读缓存的数据，还是网络请求。怎么判断，先尝试获取一下缓存，有，改一下命中率，方便后续是否从缓存读数据。还得看看符不符合要求，不符合，直接把缓存删了。存在一种情况，有网但是不用，就直接使缓存了，有网还用，返回的还是未修改，说明缓存也是有效的；返回的修改了，缓存就没用了，关了。怎么实现的？getCandidate函数中具体逻辑，就是http的那些常见的缓存策略了，一结合，就行了。每次读写缓存，都有DiskLruCache写文件日志，并且还按照一定格式。DiskLruCache特点相当鲜明：LinkedHashMap来替换LRU这个东西，自己保存读写日志来确保缓存原子性，还会对日志进行一点简化的处理。一个缓存，两个副本，CLEAN代表着现在可用，外边访问的快照全部无一例外的是CLEAN状态；DIRTY是更新状态缓存。这在一定程度上实现了读和写之间的分开。一缓存，四个文件，俩状态,每个状态对着俩文件：一个存放meta数据，一个文件存着缓存的信息。

讨论一下多线程，为啥要用呢？当然是为了提高cpu使用率啦。并行，多个电脑同时执行，一段代码。并发，一台电脑，cpu调度一下，出现多个线程，看起来像多个job一块干，其实不然的。快了，需要付出点代价，线程不安全的代价，就得解决，啥是安全？多个线程执行顺序不样，结果还是一模一样的。解决一下，就得同步，保证共享资源每次接待一个客人，安全是比快优先级更高滴。同步关键字内部才能使用wait，notify这方法，否则出错，内存模型这块，分主存，线程内存，一般情况下，线程取主存值，执行完后把数返回主存，有问题。于是，出现volatile，一旦对数操作，就和主存进行交互，保证了数据的最新。普通的异常处理，捕捉不到线程的异常。Concurrent是个非常好的工具类，TheadLocal，每个线程自己维护变量副本，各自为王，互不干扰。原子类，相当于锁关键字。

再看一下线程池，存在有什么必要？有的线程周期贼短，创建销毁时间代价忒大，真正的执行时间却少的可怜。所以抽象出一个线程管理类，这类就是Executors，这货许多静态方法，能创建线程池，缓存线程池，是在必要创建线程，活的时间不长，闲着，60s就销毁了，但是灵活啊，数量多变。固定TheadPool，线程数固定的，闲着状态，也不干掉他。一个线程的池子，按照顺序执行那些线程呀。Scheduled，按照约定好的执行。说了这么多，咋用？先，调用静态方法创建出来池子，再用submit把线程放进去，不需要更多可以shutdown，池子里的线程怎么管尼？提交后会返回一个future对象，这方法，可以取消任务，看是否取消了，看是否完了，完成后返回个结果。

池原理：

线程池就是不让cpu闲着，高效的俩原因：复用了线程。创建和销毁时间集中，可指定空闲时处理。内部成员，线程数组，保存多个线程，执行实际任务的，数组size，最多线程并发数，队列，爆满时，缓冲任务，暂时保存到队列。核心，客户端提交线程，线程池把其加入队列，判断自己的池子有没有空闲的，有，放进去执行，没有，等一哈，队列等待先，执行完了，空出多于的，再从队列取。

原子变量类，对多线程是一种比较好的处理。例如，网页访问，统计访问次数，多个用户访问，所以多线程，需要定全局变量，保存次数，并累加。如果，直接long，肯定会不精确。因为，count++这个而操作，表面上一局操作，实际上，三个原子操作，读取c，把c加上1，赋值给c。正确解决办法，AtomicLong count ，

加操作，调用incrementandget，问题就解决啦。

再唠一下synchronized关键字，非静态方法和关键字（this）锁住的是当前对象，只对同一个对象起作用，要想锁住全部类对象，加static关键字，或者锁（类.class）。对重入锁，获得了锁，还可以再次申请获得锁，一般加一个计数，0，没有人

获得锁，依次类推下去。存在的意义，其实为了防止死锁的发生。

http1.0时代，tcp连接简单明了，建立连接，服务端写数据，然后读数据，完成一次会话，但是有致命缺陷。光是请求一个页面，就需要贼多资源，每个资源一个tcp，这就跟连接池类似，创建连接，三次握手，销毁连接，四次分手，浪费大量时间资源，许多没必要的连接被创建，当然慢了，网络负载也贼高。1.1着手解决了问题，用的机制，keep-alive，说白了，就是传输完了，先别急，不关闭连接，如果还有和目标主机一样的请求，接着复用一下就可以了，这样，好处灰常明显，减少了创建销毁时间呀。速度，必须快。当然，1代还是很多毛病，并行必须多个连接，请求响应报头不会主动压缩，那就浪费流量了。千呼万唤，2代来了，解决了不少问题，hpack这玩意进行报头压缩，分层帧加入，分开了请求响应，可以交替发送，到另一头，再自己组装，更快了。帧有了优先级，加进来一个流控制。数据流诞生，就是两端之间的逻辑数据信息，帧变为原子单位。现在，实现通信，一个tcp，承载双向的数据传输，帧可以装头信息，还可以实际数据，这就是多路复用机制。

okHttp连接池保存有一个routedatabase，就是保存连接失败的路由地址。

2.5.2 GreenDao

是一个数据库框架，github 9.9k start量，是一个非常好用滴，十分快捷滴ORM解决方案。

使用

Build.gradle（project）

添加jcenter，当然还有maven库，

最后，别忘了插件的依赖呀

Build.gradle（app）

依赖的库，一定要注意版本号，最新的往往是最好的（稳定版本）

```
greendao {
schemaVersion 1 //数据库版本号
daoPackage 'com.extreme.demo' //代码生成所在包
targetGenDir 'src/main/java' //所在目录
}
```

创建一个实体类（make project 之后...）

Make project

会自动生成三个类：DaoMaster，DaoSession，ExtremeDao

获取数据库接口

```
extremeDao = daoSession.getExtremeDao();
```

增删改查操作

```
//add
extremeDao.insert(new Extreme(...));
//delete
extremeDao.deleteByKey(1);
//update
Extreme extreme = new Extreme(id:1...);
extremeDao.update(extreme);
//query
extremeDao.queryBuilder().where(...).list();
```

数据库升级：

Greendao默认数据库升级会删除旧数据库数据，需要改进一下，保存旧数据

创建临时表-->删旧表-->建新滴表-->复制临表数据到新表删除临时表；

原生的数据库API啥情况呢：

其实，安卓的数据持久手段不少，来看一看啦。数据也分两种，一种活不长的，暂时的，寄存在内存里，朝夕不保,app一关就消失殆尽了。我们保存的分析结果可不能这个样子，所以得把信息长期留下来，这就是长期的data。保存在哪儿呢，这儿有仨地方，文件系统里边，数据库里边，sharedpreferences里面。最简单的就是文件存了，这种，适合txt文件，简单类型，要不然没有固定格式，取都不好取。Context有个方法，就是为这个服务的，名叫openfileoutput（），

参数有俩，前者文件名字，注意，这地方不能有路径，为啥？文件会默认的，保存在指定文件夹，.../files，就不劳烦你制定了。后者写入的模式，两种选择，存在文件命了，直接覆盖，或者呢，追加到源文件后边去。名叫private和append。

这方法有个返回值，返回一个文件写出流对象。事务都两面性，有写就有读。

方法openfileinput（），参数简单，文件名就行，返回输入流对象。第二种存储方式，用键值对存数据，支持的类型非常多，自定义类型都行的。获取SharedPreferences方法一般三个。上下文对象的get方法，指定文件名，模式；活动get方法，默认当前类名作为文件名，PreferenceManager的获得默认的方法。存数据大致三步。Edit方法拿来一个编辑对象。第二步和map差不多了，就是调用put方法，存什么put什么，比如long，就putlong，最后，千万不要忘了提交，调用apply这个

method。内部，使用xml这种比较常见的格式保存数据的。读数据也很简单，读什么就get啥，比如，getBoolean，俩参数，第一个，名字，第二个，找不到了，默认返回啥。上面俩方法适合存一些结构单一数据，一旦复杂，存储麻烦。因此，sqlite数据库来了。它，最大特点，轻量级，不要名字密码，就能用，只要几百kb内存，速度还不慢，要的资源不很多。是一种标准化，关系类型数据库，而且还是自己嵌入的，贼方便。咋用？有个帮助类，叫SQLiteOpenHelper，可惜，这是抽象类，不能直接来使，自己建个类，继承吧，实现方法，主要是onCreate，onUpgrade这俩方法，一个负责创建时候建表，一个库升级用的。还得来构造方法，四个参数，上下文，你要新建的库名，null，最后比较重要，是记录版本号滴，新建对象，调用getWritableDatabase（），数据库就诞生啦。想看看数据库，得用个adb shell工具，到数据库文件夹，sqlite3 名，就看到了。升级也不麻烦，传入比原版本大的数，就会执行升级方法。数据操作无外乎，增，删，改，查，一一介绍。插入，insert，三个参数，表名，你往哪个表里插数据呀；null，这个用不到，最后是个对象，ContentValues，里面能装大量数据，就是一个封装了，很多的put重载，清空有个clear方法。Put俩参数，一个列名，一个值。更新，参数多点，四个，表名，你要改的数据放到一个封装对象里，和之前类似，后俩，是一种约束，传null，对所有起作用，改的时候一定慎重，不然容易出事故。删除，和改比较，没有封装这个参数，别的一致。查询最麻烦，就是那个最短的，还得7个参数，表名，查询哪些列，这个参数很好，精确定位，避免无效的盲目的查，三四就是行的约束了，再后边俩跟分组关系大，一个分组，一个还得再次筛选，最后查询结果拍一下顺序。会返回个游标，最开始在-1位置，一行一行取。当然了，它支持原生sql，用的execSQL方法，前边参数占位符，后边补上param。

2.5.3 MPAndroidChart

这东西是一款功能灰常厉害并且容易使用的安卓表库

支持线条，条形，散点图，烛台图，泡泡，饼图和蜘蛛网图

支持缩放，拖动（平移），选择，动画

适用于Android 2.2（API 8）及以上的版本

此图标库可以跨平台使用：Android和iOS

添加依赖关系

MPAndroidChart不offically支持及时滴数据，但是呢，对添一些新数据以图表或remove数据动态地，存在允许添加或delete的各种方法Entry的对象到现有DataSet或DataSet从现有的对象/ ChartData对象。

动态添加/删除数据的可能性

类DataSet（和所有子类）：

addEntry(Entry e)：将给定的Entry对象添加到DataSet。

这个类ChartData：

给定添加Entry到DataSet指定的数据集索引处。

addDataSet(DataSet d)：将给定的DataSet对象添加到ChartData对象。

除此之外，还有一些动态删除数据的方法：

类DataSet（和其所有子类）：

public boolean removeFirst()：从条目数组中删除此DataSet的第一个条目（在索引0处）。

从中抹掉给定的x-index DataSet。如果成功就来return一个true。

类ChartData：

public boolean removeDataSet(DataSet ds)：DataSet从ChartData对象中除掉给定对象。If成功则返回true。

ViewPortHandler这类，专门来处理图表，视图interface。对眼所能及的他都管，缩放状态，图表大小以及绘图区域和当前偏移量。在ViewPortHandler允许直接访问上述所有特性和修改。

图表内容

clear()：清空处理。

clearValues()：清除所有DataSet对象的图表，从而清除all条目。不会从图表中删除提供的x值。

invalidate()就是更新一下图表。

getData()：返回自己设滴数据对象。

getViewPortHandler()：将返回ViewPortHandler包含有关图表大小和边界（偏移量，内容区域）以及图表当前比例（缩放）和平移（滚动）状态信息的图表对象。

getHighestVisibleXIndex()：返回图表上仍然可见的最高x索引（x轴上的值）。

一些更多的方法（Chart类的）

saveToGallery(String title)：将当前图表状态保存为图库的图像。不要忘记将“WRITE_EXTERNAL_STORAGE”权限添加到您的清单。

saveToPath(String 标题, String 路径)：保存到本地。千万记住把“WRITE_EXTERNAL_STORAGE”权限添加到manifest清单。

getChartBitmap()：返回图表滴对象，这Bitmap，总包含chart的最新状态。

setHardwareAccelerationEnabled(boolean b)：可以支持硬件滴加速，仅API级别11+

第3章系统设计与实现

3.1 分布式集群搭建

使用CentOS6虚拟机，七台机器

A.设置ip地址对应关系

192.168.233.128 zk01

192.168.233.129 zk02

192.168.233.130 zk03

192.168.233.131 nn1

192.168.233.132 nn2

192.168.233.133 rm1

192.168.233.134 rm2

节点说明：

NN配置两台主机，活跃状态和待定状态

RM配置两台，所属子进程nodemanager配置在三台主机

DN配置三台，尽量和NM在一起，原因：转移计算结果不转移数据，更高效

Qj，子进程journalnode分别在三台

Zkfc，分别对应两台NN

ZK配置三台，用于jn的edits同步，NN切换，DN通信，NN通信，NM通信

B.

vi（主机名文件目录）/etc/sysconfig/network 修改名

vi（IP地址映射）/etc/hosts 修改ip

C.安装JDK，配置JDK和其他bin指令的环境变量

把java添加到环境变量里面

vi（变量所在目录）/etc/profile

#追加文件

export JAVA_HOME=（用的jdk是7版本）/home/hadoop/app/jdk-7u_65-i585

export PATH=$PATH:$JAVA_HOME/bin

#重新加载

source /etc/profile

D.安装Hadoop--》版本是2.4.1

上传hadoop的安装包到服务器上/home/hadoop/（主目录下.）

这里方法很多，可以用sftp，方便快捷。（put方法）

E.修改hadoop的配置文件（基于HA的分布式高可用配置）

1.hadoop-env.sh（jdk环境变量经常获取不到，因此直接写死）

vi hadoop-env.sh（在conf目录下，下同）

export JAVA_HOME=/.../java/jdk1.7.0_65

2.core-site.xml(HA机制)

<name>fs.defaultFS</name>

<!--如果是单个应该为hdfs：//主机名：9000/-->

<value>hdfs://ns1/</value>

<!-- 临时目录,存放数据，可细分NN，DN存放目录 -->

3.mapred-site.xml（指定资源分配方式）

4.hdfs-site.xml

5.yarn-site.xml

F.改改slaves文件

G.配置无密登陆：

因为是一个集群，因此启动的时候需要多台主机的配合，每次都需要输入密码，所以无密登录就显得比较便捷了。

原理：

基于ssh，ssh是应用层的安全协议。

操作：

对两台主机a和b，a登录b实现无密登录：第一，a生成密钥对，ssh-keygen -t rsa，会在主目录的.ssh目录下生成公钥和私钥；第二，把公钥传到b主机，scp

当前公钥 b:/.. ；第三，b主机把公钥放到自己的授权列表，在.ssh目录下新建一个authorized_keys，此文件的权限为600，chmod 600 authorized_keys ，把公钥追加到授权列表，cat 公钥 >> authorized_keys

H.启动集群

1.启动zookeeper集群（zk01 zk02 zk03)

./zkServer.sh start-->启动，产生QuorumPeerMain进程

./zkServer.sh status-->可以查看每台机器状态，leader一个，follower两个

2.启动journalnode（zk01,zk02,zk03)

hadoop-daemon.sh start journalnode

可以用java自带的jps指令查看进程是否启动

3.第一次启动的时候需要NN之间同步（nn1）

首先需要格式化hfds

hdfs namenode -format

Hadoop会自动生成一个tmp文件（在配置文件中指定位置），

把其复制到另一台nn2

scp -r tmp/ nn2:/home/hadoop/app/hadoop-2.4.1/

然后格式化zkfc

hdfs zkfc -formatZK

4.启动集群（nn1）

start-dfs.sh

这句指令会启动两个namenode，同时启动3个datanode，（包括journalnode），两个zkfc

5.启动yarn框架（rm1 rm2）

Yarn脚本不够完善

start-yarn.sh (rm1)

yarn-daemon.sh start resourcemanager (rm2)

3.2 日志导入

因为数据量比较小，所以使用hadoop的shell命令即可。

每天从指定文件夹传送文件到HDFS系统

脚本代码如下

hadoop fs -put /home/hadoop/log/${day} /data

${day}是变量，/data是hdfs文件目录

3.3 数据清洗

脚本如下

#clean data

hadoop jar /home/hadoop/demo.jar com.Cleaner /data/${day} /data/out/${day}/ 1>/dev/null

原始文件每行格式遵循apache-log标准

处理后的文件格式为：

Ip地址时间（格式yyyy_MM_dd_HH_mm_SS） 访问资源路径

MyMapper

MyReducer

LogParser

DataCleaner

打包jar

并且放到/home/hadoop/下，命名为demo.jar

3.4 数据分析

数据分析主要用到了hive，

Shell脚本执行hql语句产生mapreduce程序对清洗后的数据分析

hive-init.sh

```
#!/bin/sh
#create table
Hive-daily.sh
#handle logs
cd /home/hadoop/log
for file in $(ls *)
do
hive-handle.sh $file
rm -r -f $file
Done
Hive-handle.sh
#!/bin/sh
#get date
#clean data
#add PARTITION
#analysis data
#export to mysql
sqoop export --connect jdbc:mysql://nn1:3306/datastore --username root --password root --table result --fields-terminated-by '\001' --export-dir '/user/hive/warehouse/cleandata_'${day}
```

3.5 报表展示

分析后的数据保存到mysql数据库

利用tomcat提供数据服务，安卓端通过网络获取数据

```
Servlet页面
doPost (  ) {
//获取一下名
String name = request.getParameter("AndroidRequestName");
//再获取一下密码
String password = request.getParameter("AndroidRequestPassword");
//打印一下下
System.out.println(name);
//获取到输出流
PrintWriter writer = response.getWriter();
if("root".equals(name)&&"root".equals(password)){
//用户名密码正确
Connection connection = JDBCUtils.getConnection();
声明statement 对象，
再声明一个resultSet
try {
statement = 一条sql语句查询表滴all数据；
//这句执行sql语句得到结果
resultSet = statement.executeQuery();
//获取数据，封装到list数组
List<DataResult> list = new ArrayList<>();
DataResult dataResult = null;
while(resultSet.next()){
dataResult = new DataResult();
dataResult.setDate(resultSet.getString("date"));
dataResult.setPv(resultSet.getInt("pv"));
dataResult.setNewuser(resultSet.getInt("newuser"));
dataResult.setVisitors(resultSet.getInt("visitors"));
dataResult.setJumper(resultSet.getInt("jumper"));
list.add(dataResult);
}
```

//使用gson转换成json数组
Gson gson = new Gson();
//直接转化成字符串
String json = gson.toJson(list);
//在这打印看看对不对
System.out.println(json);
writer.write(json);
} catch (SQLException 异常) {
异常处理一下下；
}finally{
//关闭资源
JDBCUtils.close(connection,statement,resultSet);
}
}else {
//错误返回
writer.write("permission denied");
}
}
doGet(){
doPost(request, response);
}
Web.xml
安卓端将获取的数据保存到sqlite数据库
数据传输阶段用的json数据格式
首先OkHttp网络获取数据，接着GreenDao写入数据库
获取数据并写入是一个app，此app还会开启一个contentProvider，提供给其他app数据。
内容提供者
安卓端利用MPAndroidChart进行数据展示
用户访问量，用户访问数量和用户流失数用柱状图表示
通过内容提供者获取游标，按需求取数据
Pv--》20天数据
Visitors--》前10天
Jumper--》后十天
都是按X轴Y轴动态展示
新增用户数用折线图显示可以保存当前图片到内存

7. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第7部分

总字数：1173

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第4章系统测试

4.1功能测试

数据分析结果

date pv newuser visitors jumper
"2018_1_10" "169866" "28" "10411" "3749"
"2018_1_11" "502404" "223" "24635" "8454"
"2018_1_12" "208336" "83" "13654" "4629"
"2018_1_13" "158674" "20" "10021" "3212"
"2018_1_14" "136548" "18" "10068" "3167"
"2018_1_15" "214639" "95" "15233" "5826"
"2018_1_16" "294314" "112" "18065" "6004"

"2018_1_17" "320875" "128" "20716" "7522"
"2018_1_18" "410226" "147" "21895" "7803"
"2018_1_19" "201865" "86" "12664" "4501"
"2018_1_20" "336954" "115" "21786" "7644"
"2018_1_21" "486230" "230" "23650" "8121"
"2018_1_22" "516194" "255" "25136" "8816"
"2018_1_23" "542318" "221" "23694" "8215"
"2018_1_24" "461294" "189" "20897" "7455"
"2018_1_25" "441364" "206" "21470" "7201"
"2018_1_26" "451298" "199" "20669" "7461"
"2018_1_27" "458710" "187" "21456" "8366"
"2018_1_28" "468744" "204" "22469" "7249"
"2018_1_29" "441990" "190" "23542" "7143"
报表展示结果
新增用户（20天）
访问总量（20天）
访问用户数量（前10天）
用户流失量（后十天）
4.2可用性测试
NN节点高可用
其中一个namenode节点宕机，另一个会无缝切换：
1.杀死namenode进程，迅速切换
2.直接关闭namenode所在主机，等待30秒之后切换（可以自己配置）
DN节点高可用
存在两个及以上DN可用，系统就会提供服务
删掉一个DN节点，可以杀死datanode的进程，也可以直接关机。
再次进行，hdfs相关操作，上传个文件，下载个文件，都可以。

8. 53140123_徐浩_计算机科学与技术_基于Hadoop的论坛日志分析系统的设计与实现.doc_第8部分

总字数：1113

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第5章总结与展望

本次集群的搭建并不顺利，经常遇到各种问题：虚拟机的ip地址设置成了自动获取，导致每次重启虚拟机的时候ip都会变化；防火墙没关，会出现连接失败的问题；单次处理的文件量过大，内存爆表，不得不重新再装一次；数据传输的时候，tomcat会偶尔失灵，其实不是代码问题；数据传输时开着代理，导致ip地址错误，没法连接服务器；greenDao数据库更新的时候会把之前的数据清空，问题比较多，就不一一列举了。

深感自身能力不足，还需提高。比如缓存，okHTTP的缓存机制，底层连接的原理，用到的各种头信息，对连接的复用与优化，都是值得深入挖掘的点。

还有一个感受，开源的框架真的太好用了，感谢那些无私提供项目的开源者，因为是框架对底层的封装，因此用起来方便许多，但这只适用于基础浅显的应用，当数据量上来了，或者需要更快的响应速度，就得去看看底层代码了，看源代码的确挺辛苦的，需要思路清晰，得耐得下性子，枯燥中还有乐趣。实在坚持不下去了，看别人写的源码分析的博客也不错。

对大数据技术本身而言，串联的领域非常广，几乎涵盖了常用的那些技术。收益还是很可观的，尤其是加深了对分布式的理解，说白了，分布式就是一个人干不了这么多活，分给多个人协同解决，这其中就得进行协调，于是出现了协调框架zookeeper，每个人具体分配什么任务，怎么干活，这就是具体的计算逻辑了，比如mapreduce先分后合，spark实时内存处理，storm流式处理，两个一拼接，分布式系统就产生了。

对于个人发展，大数据这门技术大大开阔了自己眼界，比如，分布式协调技术，各种数据处理框架，nginx代理服务器，集群的认识，都大有帮助。同时，也意识到其实机器配置蛮重要的，内存。Cpu，磁盘都得足够大，这样玩虚拟机不会那么累。

对于未来，绝对是属于大数据的，未来的科技就是建立在大数据基础之上的，这种思想，这种技术，类似于编程语言的汇编层面，基石，重要性不言而喻了。

具体到应用层面，还是很广的，最近很火的区块链，就可以结合大数据来用，柯斯塔曾说，数字汇流会对未来形成很大的冲击。个人觉得，大数据最广的领域应该是物联网。连接的设备包括基本的电子社别，还有车载系统，医疗数据，共享单车数据，外卖订购数据，等等，涵盖人的一切活动，衣食住行，这会形成一个庞大的网络，其中当然蕴含无限宝贵的数据，运用大数据技术，会让世界资源分配更加合理，类似大航海时代的功效，还会彻底改善人的生活方式，改变人的一些观念，对社会进步会有巨大的推进作用。

参考文献

[1] 邓涵元,卢山,程光.基于MPP-Hadoop混合架构高校数据集成系统研究[J/OL].计算机技术与发展,2018.

[2] Tom White，Hadoop:the Definitive Guide Storage and Analysi．Hadoop权威指南:大数据的存储与分析(第4版)．王海/华东/刘喻/吕粤海译．清华大学出版社，2017.

[3]陈丽,黄晋,王锐.Hadoop大数据平台安全问题和解决方案的综述[J].计算机系统应用,2018.

[4]]Jyoti V. Gautam,Harshadkumar B. Prajapati,Vipul K. Dabhi,Sanjay Chaudhary. Empirical Study of Job Scheduling Algorithms in Hadoop MapReduce[J]. Cybernetics and Information Technologies,2017,17(1).

[5]李庆君. Hadoop架构下海量空间数据存储与管理[D].武汉大学,2017.

[6]刘红敏.大数据分析平台Hadoop的关键技术[J].电子技术与软件工程,2018.

[7]梁林森.基于hadoop技术的信息系统用户体验智能识别与分析研究[J].中国信息化,2018.

[8]李时玉,孙沫卿,郭建伟.基于科技情报Hadoop平台的系统研究[J].物联网技术,2018.

[9]曾磊,任颖超.基于大数据技术的农业信息管理平台设计[J].南方农机,2018.

[10]李庆君. Hadoop架构下海量空间数据存储与管理[D].武汉大学,2017.

[11]戴海辉. 基于Hadoop的校园卡数据挖掘的研究与实现[D].南昌航空大学,2017.

致谢

感谢 Hadoop权威指南的作者，感谢同学对自己的帮助和支持，感谢老师的指点，感谢百度和谷歌搜索引擎。