



# 目錄

介绍	1.1
第一章 概念	1.2
第二章 云计算技术	1.3
1 结构化数据与非结构化数据	1.3.1
第三章 Hadoop是什么	1.4
1.基本概念	1.4.1
2.原理漫画	1.4.2
3.技术选型	1.4.3
第四章 Hadoop的安装	1.5
1.Hadoop 2.6.2伪分布式	1.5.1
2.Hadoop 2.6.2完全分布式	1.5.2
3 Hadoop 2.7.2完全分布式	1.5.3
4.Hadoop的组件	1.5.4
5.SSH免密钥登录	1.5.5
第五章 Ambari集群管理	1.6
1.Ambari集群安装	1.6.1
第六章 Hive	1.7
1.Hive 1.1.1的安装	1.7.1
2.Hive 2.1.0的安装	1.7.2
3.Hive的基本使用与导入导出	1.7.3
4.Hive数据类型与文件格式	1.7.4
5 Hive的存储架构与HQL语法	1.7.5
6 Hive的模式设计与事务性	1.7.6
7 Hive综合案例实战	1.7.7
8 Hive开发	1.7.8
9 Hive安全	1.7.9
10 FAQ	1.7.10

第七章 Sqoop	1.8
1.Sqoop2的安装	1.8.1
2.Sqoop1的安装	1.8.2
3.Sqoop综合案例	1.8.3
4.Sqoop综合案例	1.8.4
5.DataX性能对比	1.8.5
第八章 HBASE	1.9
1.HBASE的伪分布安装与分布式安装	1.9.1
2.HBASE常用的Shell命令	1.9.2
3.基于HBASE的Java开发	1.9.3
4.基于HBASE的Python开发	1.9.4
5.HBASE与传统数据库的区别	1.9.5
6.HBASE安装疑难杂症	1.9.6
7.Hive与Hbase的区别	1.9.7
第九章 HBASE实战	1.10
1.HBASE基于Java开发	1.10.1
2.整合SQL引擎层	1.10.2
3.基本数据迁移	1.10.3
4.基于Bulk load的数据迁移	1.10.4
5.使用管理工具	1.10.5
6 Hbase 数据备份及恢复	1.10.6
7 监控与诊断	1.10.7
副录-HBase 资源收集	1.10.8
第十章 Spark	1.11
1. 基YARN安装	1.11.1
2. 基于Mesos安装	1.11.2
3.Hadoop与Spark的区别	1.11.3
第十一章 CDH的发行版本	1.12
1.基于Centos的安装	1.12.1
2.基于Ubuntu的安装	1.12.2

第十二章 TDH发行版本	1.13
1.安装	1.13.1
2.Inceptor-SQL使用	1.13.2
3.使用JDBC、ODBC工具连接Inceptor	1.13.3
4 Sqoop的使用	1.13.4
5 使用JDBC、ODBC工具连接Inceptor	1.13.5
6 SQL兼容测试	1.13.6
7 SQL语法知识	1.13.7
附录 POC前的准备工作	1.13.8
第十三章 HUE安装与配置	1.14
第十四章 数据采集与爬虫	1.15
第十五章 Hadoop相关资源	1.16
1 github相关资源收集	1.16.1
第十五章 Hadoop100问	1.17
第十六章 大数据框架合集	1.18
1.电商网站架构案例	1.18.1
第十七章 运维	1.19
1.运维常用工具	1.19.1
第十八章 机器学习入门	1.20
第十九章 Centos下的Oracle安装	1.21
第二十章 基于JFinal的大数据框架快速开发	1.22
1 基于Gradle的项目构建	1.22.1
第二十一章 ELK	1.23
附录 Hadoop运维技能要求	1.24

本书来源：开源书籍：大数据实验手册 (by 楚广明)

报名参与：Star/fork GitHub 仓库 并发送 Pull Request

关注我们：扫描二维码 关注 @楚广明 微博和微信公众号

赞助我们：赞助 8.99¥，更多原创开源书籍期待您的支持 ^o^

# 大数据实验手册

v 0.1

这是一本关于大数据学习记录的手册,主要针对初学者.做为一个老IT工作者,学习是一件很辛苦的事情.希望这本手册对帮助大家快速的学习与认识大数据(特指Hadoop Spark),为了不让初学者一下接触爆炸式的新概念,我们会以实验先行,概念跟进的方式进行课程学习,这样有利于大家快速进入状态,而不至于一直深陷逻辑概念出不来,但是每个人的学习方式不一样,仁者见仁智者见智吧.大家如果有意见请给我发邮件 chu888chu888@qq.com — 楚广明

## 介绍

- 项目首页：<https://github.com/chu888chu888>
- 代码仓库：<https://github.com/chu888chu888/HadoopAndSparkDataStudy>
- 在线阅读：<https://chu888chu888.gitbooks.io/hadoopstudy/content/>

## 纠错

欢迎大家指出不足，如有任何疑问，请邮件联系 [5211486@qq.com](mailto:5211486@qq.com) 或者直接修复并提交 Pull Request。

## 版权

本书采用  协议发布，详细版权信息请参考 [CC BY NC ND 4.0](#)。

---

## 赞助我们

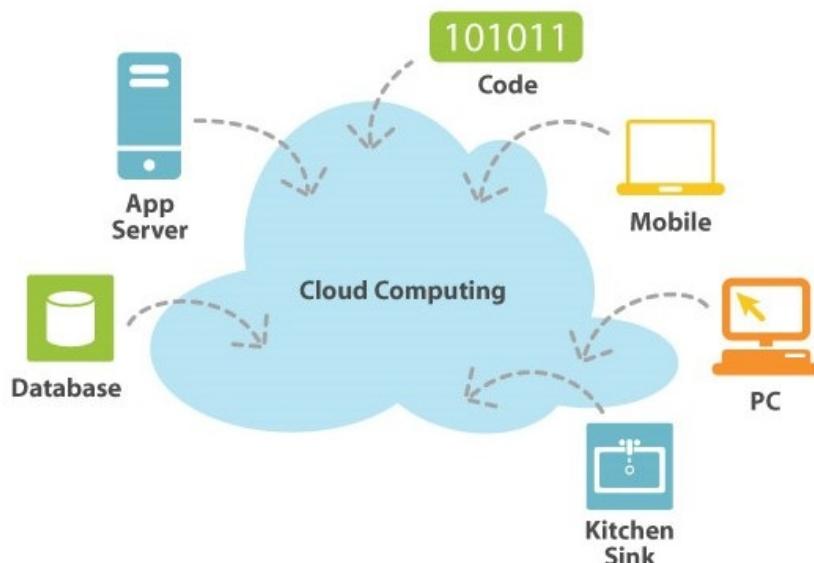
更多原创开源书籍

## 概述

大家知道什么叫做云计算吗？事实上，目前并没有一个确定的定义。然而概括来讲，所谓的云计算，指的就是把你的软件和服务统一部署在数据中心，统一管理，从而实现高伸缩性。

云计算具有以下特性：

- 虚拟化和自动化
- 服务器，存储介质，网络等资源都可以随时替换
- 所有的资源都由云端统一管理
- 高度的伸缩性以满足业务需求
- 集中于将服务传递给业务.



云计算示意图

云计算的部署方式

从部署方式来说，总共有两类云计算

- 私有云：数据中心部署在企业内部，由企业自行管理。微软为大家提供了**Dynamic Data Center Toolkit**，来方便大家管理自己的数据中心。
- 公共云：数据中心由第三方的云计算供应商提供，供应商帮助企业管理基础设施（例如硬件，网络，等等）。企业将自己的软件及服务部属在供应商提供的数据中心，并且支付一定的租金。**Windows Azure**正是这样一个公共云平台。

## 云计算的运营方式

从运营方式来说，总共有三类云计算：

- 软件即服务（**SaaS**）：云计算运营商直接以服务的形式供应软件，供最终用户使用。有些服务还提供了**SDK**，从而使得第三方开发人员可以进行二次开发。在这种运营模式下，开发人员通常只能针对现有的产品开发插件，而无法充分挖掘平台和操作系统的功能，而必须从头开始实现。微软的**Bing**，**Windows Live**，**Microsoft Business Productivity Online**等产品就属于这一类型。
- 平台即服务（**PaaS**）：云计算运营商将自己的开发及部署平台提供给第三方开发人员，第三方开发人员在这个平台上开发自己的软件和服务，供自己或其它用户使用。在这种运营模式下，开发人员有了更多的自由，可以发挥出平台的强大功能，而不受现有产品的束缚。**Windows Azure**正是这样一个产品。
- 基础设施即服务（**IaaS**）：云计算运营商提供但不管理基础设施，第三方开发人员将开发好的软件和服务交给自己公司的**IT**管理员，由**IT**管理员负责部署及管理。在这种运营模式下，开发人员和**IT**管理员有最大限度的自由，然而由于必须自行管理部分基础设施，因此成本通常也会较大，对管理员的要求也会较高。目前微软尚未提供**IaaS**的云计算运营模式，不过我们正在考虑如何给予开发人员和**IT**管理员更多的自由。

## 总结

&nbsp;云计算指的就是把你的软件和服务统一部署在数据中心，统一管理，从而实现高伸缩性。从部署方式来说，云计算可以分为私有云和公共云。从运营方式来说，云计算可以分成**SaaS**，**PaaS**，**IaaS**三类。

## 第二章 云计算技术

### 云计算技术：

首先让大家明白什么是云端，所谓云端需要两层理解

- 服务不在本地，这一层可以理解为服务器
- 它和普通的服务器是不一样的，这些云端的服务器的资源是共享的，一旦一个服务器不能承受，将会把任务分配给其他机器。

### 云技术与其他技术的区别：

云技术可以使用的语言有java,c++等。云技术的开发，并没有发展什么新语言，而是在其他语言的基础上。比如Java语言。与其他技术，最显著的区别，不是在开发上，而是在于架构上，最显著的特点是分布式。

### 成熟的云计算技术：

- Hadoop

Hadoop是一个框架，它是由Java语言来实现的。Hadoop是处理大数据技术.Hadoop可以处理云计算产生大数据，需要区分hadoop并不是云计算。它和云计算密不可分。

Hadoop产生是互联网的产物，也是必然。大家都知道，我们上网时需要服务器的。假如世界上只有一台电脑，根本不需要服务器。如果有10台服务器，100台，1000台，上万台，那么我们该如何让大家相互通信，共享知识，所以我们产生了互联网。

互联网产生，全世界都可以通信，知识如此居多，我们像获取更多的知识，想获取新技术，获取新知识，通过什么，国内通过百度，国外也有许多，比如Google。可是百度和谷歌的用户有多少，多了不说，最起码有上亿的用户。并且这些用户每天上百度，上谷歌，又会产生多少数据，查询多少数据。那么他们怎么承受如此多用户。

这不是一台电脑、一台服务器能完成的事情。Hadoop就是一个解决方案。Hadoop是一个分布式方案，能够把压力分摊到其他服务器。至于如何做到的，可以深入了解Hadoop的mapreduce等知识。

- **openstack**

openstack是搭建云平台技术，可以搭建公有云，私有云，和混合云。OpenStack是开源的云管理平台，用来统一管理多个虚拟化集群的框架。openstack目前分为两种:openstack的运维与openstack的二次开发

- **Cloud Foundry**

Cloud Foundry是一个开源的平台即服务产品，它提供给开发者自由度去选择云平台，开发框架和应用服务。Cloud Foundry最初由 VMware 发起，得到了业界广泛的支持，它使得开发者能够更快更容易的开发，测试，部署和扩展应用。Cloud Foundry是一个开源项目，用户可以使用多种私有云发行版，也可以使用公共云服务。

- **nosql**

nosql即not only sql。nosql数据库是一种比较低级的数据库，关系型数据库是由nosql数据库发展而来。什么是关系型数据库，这里不从概念上区别，常用的SqlServer，mysql,oracle都是关系型数据库。关系型数据库顾名思义，数据库关系明确严谨。而nosql则是一种数据关系不严谨的数据库。一个key和value。

# 1 结构化数据与非结构化数据

相对于结构化数据（即行数据，存储在数据库里，可以用二维表结构来逻辑表达实现的数据）而言，不方便用数据库二维逻辑表来表现的数据即称为非结构化数据，包括所有格式的办公文档、文本、图片、XML、HTML、各类报表、图像和音频/视频信息等等。

非结构化数据库是指其字段长度可变，并且每个字段的记录又可以由可重复或不可重复的子字段构成的数据库，用它不仅可以处理结构化数据（如数字、符号等信息）而且更适合处理非结构化数据（全文文本、图象、声音、影视、超媒体等信息）。

非结构化WEB数据库主要是针对非结构化数据而产生的，与以往流行的关系数据库相比，其最大区别在于它突破了关系数据库结构定义不易改变和数据定长的限制，支持重复字段、子字段以及变长字段并实现了对变长数据和重复字段进行处理和数据项的变长存储管理，在处理连续信息（包括全文信息）和非结构化信息（包括各种多媒体信息）中有着传统关系型数据库所无法比拟的优势。

- 结构化数据(即行数据,存储在数据库里,可以用二维表结构来逻辑表达实现的数据)
- 非结构化数据,包括所有格式的办公文档、文本、图片、XML、HTML、各类报表、图像和音频/视频信息等等

所谓半结构化数据，就是介于完全结构化数据（如关系型数据库、面向对象数据库中的数据）和完全无结构的数据（如声音、图像文件等）之间的数据，HTML文档就属于半结构化数据。它一般是自描述的，数据的结构和内容混在一起，没有明显的区分。

## 数据模型：

- 结构化数据：二维表（关系型）
- 半结构化数据：树、图
- 非结构化数据：无
- RMDBS的数据模型有：如网状数据模型、层次数据模型、关系型

- 结构化数据：先有结构、再有数据
- 半结构化数据：先有数据，再有结构

随着网络技术的发展，特别是Internet和Intranet技术的飞快发展，使得非结构化数据的数量日趋增大。这时，主要用于管理结构化数据的关系数据库的局限性暴露地越来越明显。因而，数据库技术相应地进入了“后关系数据库时代”，发展进入基于网络应用的非结构化数据库时代。

## 第三章 Hadoop是什么？

### Hadoop是什么？

- Hadoop是一个开源的框架，可编写和运行分布式应用处理大规模数据，是专为离线和大规模数据分析而设计的，并不适合那种对几个记录随机读写的在线事务处理模式。Hadoop=HDFS（文件系统，数据存储技术相关）+ Mapreduce（数据处理），Hadoop的数据来源可以任何形式，在处理半结构化和非结构化数据上与关系型数据库相比有更好的性能，具有更灵活的处理能力，不管任何数据形式最终会转化为key/value，key/value是基本数据单元。用函数式变成Mapreduce代替SQL，SQL是查询语句，而Mapreduce则是使用脚本和代码，而对于适用于关系型数据库，习惯SQL的Hadoop有开源工具hive代替。
- Hadoop就是一个分布式计算的解决方案.

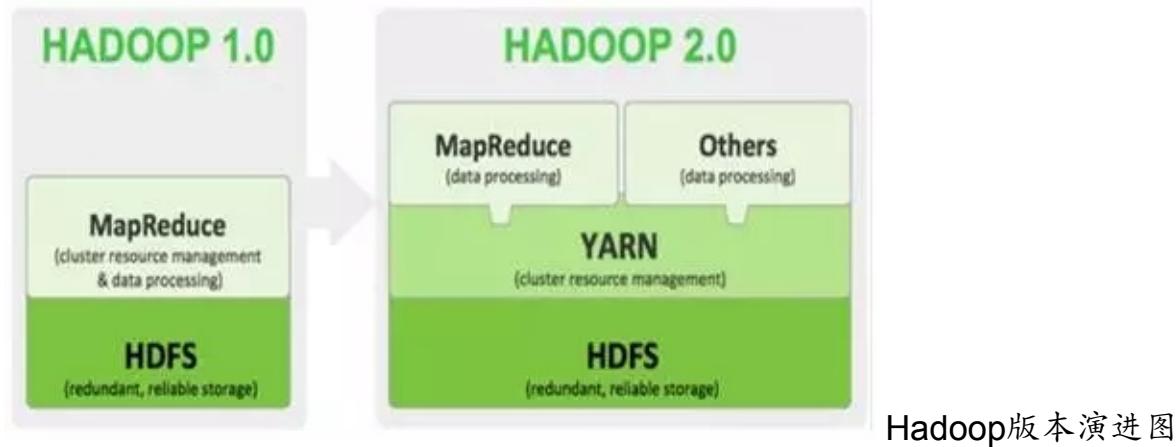
### hadoop能做什么？

- hadoop擅长日志分析，facebook就用Hive来进行日志分析，2009年时facebook就有非编程人员的30%的人使用HiveQL进行数据分析；淘宝搜索中的自定义筛选也使用的Hive；利用Pig还可以做高级的数据处理，包括Twitter、LinkedIn上用于发现您可能认识的人，可以实现类似Amazon.com的协同过滤的推荐效果。淘宝的商品推荐也是！在Yahoo！的40%的Hadoop作业是用pig运行的，包括垃圾邮件的识别和过滤，还有用户特征建模。（2012年8月25新更新，天猫的推荐系统是hive，少量尝试mahout！）

从现在企业的使用趋势来看，Pig慢慢有点从企业的视野中淡化了。

### Hadoop的版本演进

当前Hadoop有两大版本：Hadoop 1.0和Hadoop 2.0，如下图所示。



Hadoop1.0被称为第一代Hadoop，由分布式文件系统HDFS和分布式计算框架MapReduce组成，其中，HDFS由一个NameNode和多个DataNode组成，MapReduce由一个JobTracker和多个TaskTracker组成，对应Hadoop版本为0.20.x、0.21.X，0.22.x和Hadoop 1.x。其中0.20.x是比较稳定的版本，最后演化为1.x，变成稳定版本。0.21.x和0.22.x则增加了NameNode HA等新特性。

第二代Hadoop被称为Hadoop2.0，是为克服Hadoop 1.0中HDFS和MapReduce存在的各种问题而提出的，对应Hadoop版本为Hadoop 0.23.x和2.x。

针对Hadoop1.0中NameNode HA不支持自动切换且切换时间过长的风险，Hadoop2.0提出了基于共享存储的HA方式，支持失败自动切换切回。

针对Hadoop 1.0中的单NameNode制约HDFS的扩展性问题，提出了HDFS Federation机制，它允许多个NameNode各自分管不同的命名空间进而实现数据访问隔离和集群横向扩展。

针对Hadoop 1.0中的MapReduce在扩展性和多框架支持方面的不足，提出了全新的资源管理框架YARN，它将JobTracker中的资源管理和作业控制功能分开，分别由组件ResourceManager和ApplicationMaster实现。其中，ResourceManager负责所有应用程序的资源分配，而ApplicationMaster仅负责管理一个应用程序。相比于 Hadoop 1.0，Hadoop 2.0框架具有更好的扩展性、可用性、可靠性、向后兼容性和更高的资源利用率以及能支持除了MapReduce计算框架外的更多的计算框架，Hadoop 2.0目前是业界主流使用的Hadoop版本。

## Hadoop的使用场景

项目一：数据整合

称之为“企业级数据中心”或“数据湖”，这个想法是你有不同的数据源，你想对它们进行数据分析。这类项目包括从所有来源获得数据源（实时或批处理）并且把它们存储在hadoop中。有时，这是成为一个“数据驱动的公司”的第一步；有时，或许你仅仅需要一份漂亮的报告。“企业级数据中心”通常由HDFS文件系统和HIVE或IMPALA中的表组成。未来，HBase和Phoenix在大数据整合方面将大展拳脚，打开一个新的局面，创建出全新的数据美丽新世界。

销售人员喜欢说“读模式”，但事实上，要取得成功，你必须清楚的了解自己的用例将是什么（Hive模式不会看起来与你在企业数据仓库中所做的不一样）。真实的原因是一个数据湖比Teradata和Netezza公司有更强的水平扩展性和低得多的成本。许多人在做前端分析时使用Tableau和Excel。许多复杂的公司以“数据科学家”用Zeppelin或IPython笔记本作为前端。

#### 项目二：专业分析

许多数据整合项目实际上是从你特殊的需求和某一数据集系统的分析开始的。这些往往是令人难以置信的特定领域，如在银行领域的流动性风险/蒙特卡罗模拟分析。在过去，这种专业的分析依赖于过时的，专有的软件包，无法扩大数据的规模经常遭受一个有限的功能集（大部分是因为软件厂商不可能像专业机构那样了解的那么多）。

在Hadoop和Spark的世界，看看这些系统大致相同的数据整合系统，但往往有更多的HBase，定制非SQL代码，和更少的数据来源（如果不是唯一的）。他们越来越多地以Spark为基础。

#### 项目三：Hadoop作为一种服务

在“专业分析”项目的任何大型组织（讽刺的是，一个或两个“数据整理”项目）他们会不可避免地开始感觉“快乐”（即，疼痛）管理几个不同配置的Hadoop集群，有时从不同的供应商。接下来，他们会说，“也许我们应该整合这些资源池，”而不是大部分时间让大部分节点处于资源闲置状态。它们应该组成云计算，但许多公司经常会因为安全的原因（内部政治和工作保护）不能或不会。这通常意味着很多Docker容器包。

#### 项目四：流分析

很多人会把这个“流”，但流分析是不同的，从设备流。通常，流分析是一个组织在批处理中的实时版本。以反洗钱和欺诈检测：为什么不在交易的基础上，抓住它发生而不是在一个周期结束？同样的库存管理或其他任何。

在某些情况下，这是一种新的类型的交易系统，分析数据位的位，因为你将它并联到一个分析系统中。这些系统证明自己如Spark或Storm与Hbase作为常用的数据存储。请注意，流分析并不能取代所有形式的分析，对某些你从未考虑过的事情而言，你仍然希望分析历史趋势或看过去的数据。

#### 项目五：复杂事件处理

在这里，我们谈论的是亚秒级的实时事件处理。虽然还没有足够快的超低延迟（皮秒或纳秒）的应用，如高端的交易系统，你可以期待毫秒响应时间。例子包括对事物或事件的互联网电信运营商处理的呼叫数据记录的实时评价。有时，你会看到这样的系统使用Spark和HBase——但他们一般落在他们的脸上，必须转换成Storm，这是基于由LMAX交易所开发的干扰模式。

在过去，这样的系统已经基于定制的消息或高性能，从货架上，客户端-服务器消息产品-但今天的数据量太多了。我还没有使用它，但Apex项目看起来很有前途，声称要比Storm快。

#### 项目六：ETL流

有时你想捕捉流数据并把它们存储起来。这些项目通常与1号或2号重合，但增加了各自的范围和特点。（有些人认为他们是4号或5号，但他们实际上是在向磁盘倾倒和分析数据。），这些几乎都是Kafka和Storm项目。Spark也使用，但没有理由，因为你不需要在内存分析。

#### 项目七：更换或增加SAS

SAS是精细，是好的但SAS也很贵，我们不需要为你的数据科学家和分析师买存储你就可以“玩”数据。此外，除SAS可以做或产生漂亮的图形分析外，你还可以做一些不同的事情。这是你的“数据湖”。这里是IPython笔记本（现在）和Zeppelin（以后）。我们用SAS存储结果。

当我每天看到其他不同类型的Hadoop，Spark，或Storm项目，这些都是正常的。如果你使用Hadoop，你可能了解它们。几年前我已经实施了这些项目中的部分案例，使用的是其它技术。如果你是一个老前辈太害怕“大”或“做”大数据Hadoop，不要担心。事情越变越多，但本质保持不变。你会发现很多相似之处的东西你用来部署和时髦的技术都是围绕Hadooposphere旋转的。

#### 举个例子

设想一下这样的应用场景.

我有一个100M的数据库备份的sql文件.我现在想在不导入到数据库的情况下直接用grep操作通过正则过滤出我想要的内容。

例如：某个表中 含有相同关键字的记录那么有几种方式.

- 用linux的命令grep
- 通过编程来读取文件,然后对每行数据进行正则匹配得到结果好了现在是100M的数据库备份.

上述两种方法都可以轻松应对.

那么如果是1G,1T甚至1PB的数据呢,上面2种方法还能行得通吗?

答案是不能.毕竟单台服务器的性能总有其上限.那么对于这种超大数据文件怎么得到我们想要的结果呢?

有种方法 就是分布式计算,分布式计算的核心就在于利用分布式算法把运行在单台机器上的程序扩展到多台机器上并行运行.从而使数据处理能力成倍增加.但是这种分布式计算一般对编程人员要求很高,而且对服务器也有要求.导致了成本变得非常高.

Hadoop 就是为了解决这个问题诞生的.Hadoop 可以很轻易的把很多linux的廉价pc组成分布式结点,然后编程人员也不需要知道分布式算法之类,只需要根据mapreduce的规则定义好接口方法,剩下的就交给Hadoop. 它会自动把相关的计算分布到各个结点上去,然后得出结果.

例如上述的例子 : Hadoop 要做的事 首先把 1PB的数据文件导入到HDFS中,然后编程人员定义好map和reduce,也就是把文件的行定义为key,每行的内容定义为value , 然后进行正则匹配,匹配成功则把结果 通过reduce聚合起来返回.Hadoop 就会把这个程序分布到N 个结点去并行的操作. 那么原本可能需要计算好几天,在有了足够多的结点之后就可以把时间缩小到几小时之内.

这也就是所谓的 大数据 云计算了.如果还是不懂的话再举个简单的例子比如 1亿个 1 相加 得出计算结果, 我们很轻易知道结果是1亿.但是计算机不知道.那么单台计算机处理的方式做一个一亿次的循环每次结果+1那么分布式的处理方式则变成我用1万台计算机,每个计算机只需要计算 1万个 1 相加然后再有一台计算机把1万台计算机得到的结果再相加从而得到最后的结果.

理论上讲,计算速度就提高了1万倍.当然上面可能是一个不恰当的例子.但所谓分布式,大数据,云计算大抵也就是这么回事了.

## 引用一篇腾讯大数据的文章

大数据本身是个很宽泛的概念，Hadoop生态圈（或者泛生态圈）基本上都是为了处理超过单机尺度的数据处理而诞生的。你可以把它比作一个厨房所需要的各種工具。锅碗瓢盆，各有各的用处，互相之间又有重合。你可以用汤锅直接当碗吃饭喝汤，你可以用小刀或者刨子去皮。但是每个工具有自己的特性，虽然奇怪的组合也能工作，但是未必是最佳选择。

## 大数据，首先你要能存的下大数据

传统的文件系统是单机的，不能横跨不同的机器。HDFS（Hadoop Distributed FileSystem）的设计本质上是为了大量的数据能横跨成百上千台机器，但是你看到的是一个文件系统而不是很多文件系统。比如你说我要获取/hdfs/tmp/file1的数据，你引用的是一个文件路径，但是实际的数据存放在很多不同的机器上。你作为用户，不需要知道这些，就好比在单机上你不关心文件分散在什么磁道什么扇区一样。HDFS为你管理这些数据。

存下的数据之后，你就开始考虑怎么处理数据。虽然HDFS可以为你整体管理不同机器上的数据，但是这些数据太大了。一台机器读取成T上P的数据（很大的数据哦，比整个东京热有史以来所有高清电影的大小甚至更大），一台机器慢慢跑也许需要好几天甚至好几周。对于很多公司来说，单机处理是不可忍受的，比如微博要更新24小时热博，它必须在24小时之内跑完这些处理。那么我如果要用很多台机器处理，我就面临了如何分配工作，如果一台机器挂了如何重新启动相应的任务，机器之间如何互相通信交换数据以完成复杂的计算等等。这就是MapReduce/Tez/Spark的功能。

MapReduce是第一代计算引擎，Tez和Spark是第二代。MapReduce的设计，采用了很简化的计算模型，只有Map和Reduce两个计算过程（中间用Shuffle串联），用这个模型，已经可以处理大数据领域很大一部分问题了。

## 那什么是**Map**什么是**Reduce**？

考虑如果你要统计一个巨大的文本文件存储在类似HDFS上，你想要知道这个文本里各个词的出现频率。你启动了一个MapReduce程序。Map阶段，几百台机器同时读取这个文件的各个部分，分别把各自读到的部分分别统计出词频，产生类似（hello, 12100次），（world, 15214次）等等这样的Pair（我这里把Map和Combine放在一起说以便简化）；这几台机器各自都产生了如上的集合，然后又

有几百台机器启动Reduce处理。Reducer机器A将从Mapper机器收到所有以A开头的统计结果，机器B将收到B开头的词汇统计结果（当然实际上不会真的以字母开头做依据，而是用函数产生Hash值以避免数据串化。因为类似X开头的词肯定比其他要少得多，而你不希望数据处理各个机器的工作量相差悬殊）。然后这些Reducer将再次汇总， $(\text{hello}, 12100) + (\text{hello}, 12311) + (\text{hello}, 345881) = (\text{hello}, 370292)$ 。每个Reducer都如上处理，你就得到了整个文件的词频结果。

这看似是个很简单的模型，但很多算法都可以用这个模型描述了。Map+Reduce的简单模型很黄很暴力，虽然好用，但是很笨重。第二代的Tez和Spark除了内存Cache之类的新feature，本质上来说，是让Map/Reduce模型更通用，让Map和Reduce之间的界限更模糊，数据交换更灵活，更少的磁盘读写，以便更方便地描述复杂算法，取得更高的吞吐量。

有了MapReduce，Tez和Spark之后，程序员发现，MapReduce的程序写起来真麻烦。他们希望简化这个过程。这就好比你有了汇编语言，虽然你几乎什么都能干了，但是你还是觉得繁琐。你希望有个更高层更抽象的语言层来描述算法和数据处理流程。于是就有了Pig和Hive。Pig是接近脚本方式去描述MapReduce，Hive则用的是SQL。它们把脚本和SQL语言翻译成MapReduce程序，丢给计算引擎去计算，而你就从繁琐的MapReduce程序中解脱出来，用更简单更直观的语言去写程序了。

有了Hive之后，人们发现SQL对比Java有巨大的优势。一个是它太容易写了。刚才词频的东西，用SQL描述就只有一两行，MapReduce写起来大约要几十上百行。而更重要的是，非计算机背景的用户终于感受到了爱：我也会写SQL！于是数据分析人员终于从乞求工程师帮忙的窘境解脱出来，工程师也从写奇怪的一次性的处理程序中解脱出来。大家都开心了。Hive逐渐成长成了大数据仓库的核心组件。甚至很多公司的流水线作业集完全是用SQL描述，因为易写易改，一看就懂，容易维护。

自从数据分析人员开始用Hive分析数据之后，它们发现，Hive在MapReduce上跑，太慢了！流水线作业集也许没啥关系，比如24小时更新的推荐，反正24小时内跑完就算了。但是数据分析，人们总是希望能跑更快一些。比如我希望看过去一个小时內多少人在可穿戴手环页面驻足，分别停留了多久，对于一个巨型网站海量数据下，这个处理过程也许要花几十分钟甚至很多小时。而这个分析也许只是你万里长征的第一步，你还要看多少人浏览了电子产品多少人看了拉赫曼尼诺夫的CD，以便跟老板汇报，我们的用户是屌丝男闷骚女更多还是文艺青年／少女更多。你无法忍受等待的折磨，只能跟工程师说，快，快，再快一点！

于是Impala，Presto，Drill诞生了（当然还有无数非著名的交互SQL引擎，就不一一列举了）。三个系统的核心理念是，MapReduce引擎太慢，因为它太通用，太强壮，太保守，我们SQL需要更轻量，更激进地获取资源，更专门地对SQL做优

化，而且不需要那么多容错性保证（因为系统出错了大不了重新启动任务，如果整个处理时间更短的话，比如几分钟之内）。这些系统让用户更快速地处理SQL任务，牺牲了通用性稳定性等特性。如果说MapReduce是大砍刀，砍啥都不怕，那上面三个就是剔骨刀，灵巧锋利，但是不能搞太大太硬的东西。

这些系统，说实话，一直没有达到人们期望的流行度。因为这时候又两个异类被造出来了。他们是Hive on Tez /Spark和SparkSQL。它们的设计理念是，MapReduce慢，但是如果我用新一代通用计算引擎Tez或者Spark来跑SQL，那我就能跑的更快。而且用户不需要维护两套系统。这就好比如果你厨房小，人又懒，对吃的精细程度要求有限，那你可以买个电饭煲，能蒸能煲能烧，省了好多厨具。上面的介绍，基本就是一个数据仓库的构架了。底层HDFS，上面跑MapReduce／Tez／Spark，在上面跑Hive，Pig。或者HDFS上直接跑Impala，Drill，Presto。这解决了中低速数据处理的要求。

## 那如果我要更高速的处理呢？

如果我是一个类似微博的公司，我希望显示不是24小时热博，我想看一个不断变化的热播榜，更新延迟在一分钟之内，上面的手段都将无法胜任。于是又一种计算模型被开发出来，这就是Streaming（流）计算。Storm是最流行的流计算平台。流计算的思路是，如果要达到更实时的更新，我何不在数据流进来的时候就处理了？比如还是词频统计的例子，我的数据流是一个一个的词，我就让他们一边流过我就一边开始统计了。流计算很牛逼，基本无延迟，但是它的短处是，不灵活，你想要统计的东西必须预先知道，毕竟数据流过就没了，你没算的东西就无法补算了。因此它是个很好的东西，但是无法替代上面数据仓库和批处理系统。

还有一个有些独立的模块是KV Store，比如Cassandra，HBase，MongoDB以及很多很多很多其他的（多到无法想象）。所以KV Store就是说，我有一堆键值，我能很快速滴获取与这个Key绑定的数据。比如我用身份证号，能取到你的身份数据。这个动作用MapReduce也能完成，但是很可能要扫描整个数据集。而KV Store专用来处理这个操作，所有存和取都专门为此优化了。从几个P的数据中查找一个身份证号，也许只要零点几秒。这让大数据公司的一些专门操作被大大优化了。比如我网页上有个根据订单号查找订单内容的页面，而整个网站的订单数量无法单机数据库存储，我就会考虑用KV Store来存。KV Store的理念是，基本无法处理复杂的计算，大多没法JOIN，也许没法聚合，没有强一致性保证（不同数据分布在不同机器上，你每次读取也许会读到不同的结果，也无法处理类似银行转账那样的强一致性要求的操作）。但是丫就是快。极快。

每个不同的KV Store设计都有不同取舍，有些更快，有些容量更高，有些可以支持更复杂的操作。必有一款适合你。除此之外，还有一些更特制的系统／组件，

比如Mahout是分布式机器学习库，Protobuf是数据交换的编码和库，ZooKeeper是高一致性的分布存取协同系统，等等。有了这么多乱七八糟的工具，都在同一个集群上运转，大家需要互相尊重有序工作。所以另外一个重要组件是，调度系统。现在最流行的是Yarn。你可以把他看作中央管理，好比你妈在厨房监工，哎，你妹妹切菜切完了，你可以把刀拿去杀鸡了。只要大家都服从你妈分配，那大家都能愉快滴烧菜。你可以认为，大数据生态圈就是一个厨房工具生态圈。为了做不同的菜，中国菜，日本菜，法国菜，你需要各种不同的工具。而且客人的需求正在复杂化，你的厨具不断被发明，也没有一个万用的厨具可以处理所有情况，因此它会变的越来越复杂。

# Hadoop的一些基本概念

## 一 Hadoop的构建模块

- NameNode(名字节点)
- DataNode(数据节点)
- Secondary NameNode(次名字节点)
- JobTracker(作业跟踪节点)
- TaskTracker(任务跟踪节点)

## 1.HDFS系统的文件特征

- 存储极大数目的信息 (terabytes or petabytes) , 将数据保存到大量的节点当中。支持很大单个文件。
- 提供数据的高可靠性，单个或者多个节点不工作，对系统不会造成任何影响，数据仍然可用。
- 提供对这些信息的快速访问，并提供可扩展的方式。能够通过简单加入更多服务器的方式就能够服务更多的客户端。
- HDFS是针对MapReduce设计的，使得数据尽可能根据其本地局部性进行访问与计算。

## 2.HDFS的缺陷

- 低延迟数据访问
  - 比如毫秒级 比如低延迟与高吞吐
- 小文件存取
  - 占用NameNode大量内存 寻道时间超过读取时间
- 并发写入/文件随机修改
  - 一个文件只能有一个写者 仅支持append

## 3.MapReduce的系统工作原理

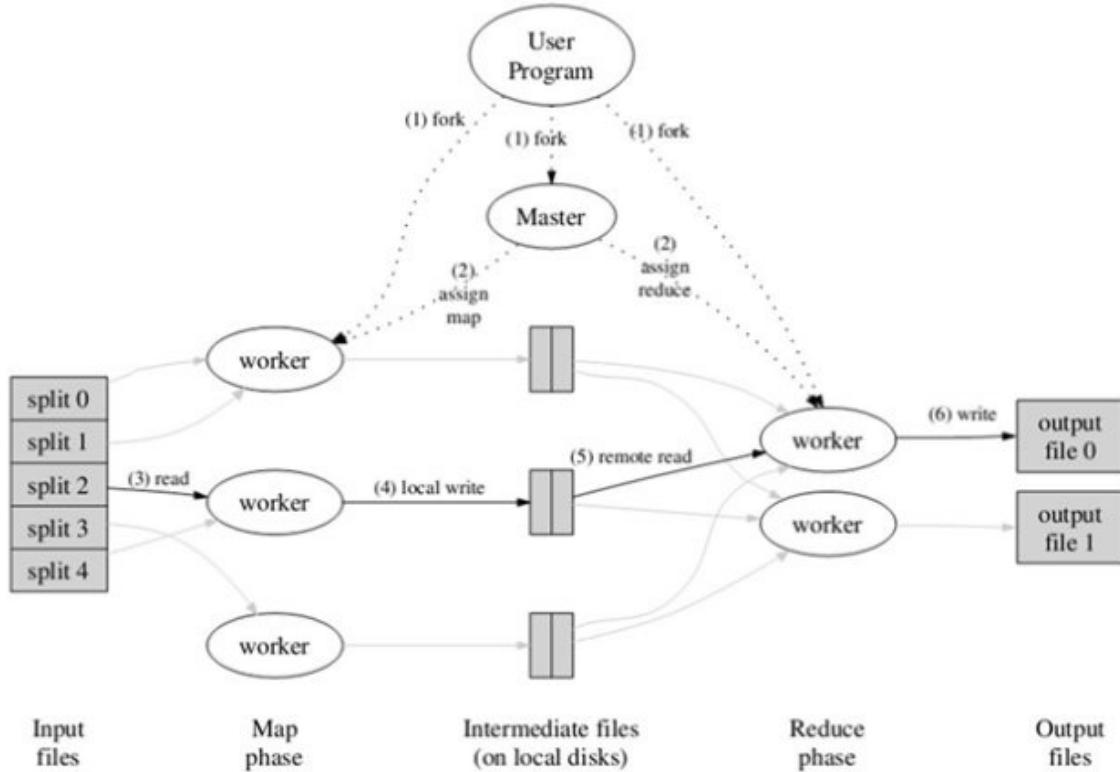


Figure 1: Execution overview

3 / 3

## 4. NameNode

NameNode是Hadoop守护进程中最重要的一个,Hadoop在分布式计算与分布式存储中都采用了主/从(Master/slave)结构.分布式存储系统被称为Hadoop文件系统,或简单称为HDFS.NameNode位于HDFS的主端,它指导DataNode执行底层的IO任务.

运行NameNode消耗大量的内存与IO资源.因此,为了减轻机器的负载,驻留NameNode的服务器通常不会存储用户数据或者执行MapReduce程序的计算任务.这意味着NameNode服务器不会同时是DataNode或者TaskTracker.

不过NameNode的重要性也带来一个负面影响-Hadoop集群的失效.

## 5. DataNode

每一个集群上的从节点都会驻留一个DataNode守护进程,来执行分布式文件系统的繁重工作,将HDFS数据块读取或者写入到本地文件系统的实际文件中.当希望对HDFS文件进行读写时,文件被分割为多个块,由NameNode告知客户端每个数据块驻

留在那个DataNode。客户端直接与DataNode守护进程通信,来处理与数据块相对应的本地文件。然而DataNode会与其他DataNode进行通信,复制这些数据块以实现冗余。

## 6. Secondary NameNode

SNN是一个监测HDFS集群状态的辅助守护进程,它通常独占一台服务器,该服务器不会运行其他DataNode或者TaskTracker守护进程。SNN与NameNode的不同在于它不接收或者记录HDFS的任何实时变化。相反它与NameNode通信,感觉集群所配置的时间间隔获取HDFS元数据的快照。

NameNode是Hadoop集群的单一故障点,而SNN的快照可以有助于减少停机的时间并降低数据丢失的风险。然而NameNode的失效处理需要人工的干预,即手动地重新配置集群,将SNN用作主要的NameNode。

## 7. JobTracker

JobTracker守护进程是应用程序和Hadoop之间的纽带。一旦提交代码到集群上,JobTracker就会确定执行计划,包括决定处理哪些文件,为不同的任务分配节点以及监控所有任务的运行。如果任务失败,JobTracker将自动重启任务,但所分配的节点可能会不同,同时受到预定义的重试次数限制。

每一个Hadoop集群只有一个JobTracker守护进程,它通常运行在服务器集群的主节点上。

## 8. TaskTracker

与存储的守护进程一样,计算的守护进程也遵循主从架构:JobTracker作为主节点,监测MapReduce作业的整个执行过程,同时,TaskTracker管理各个任务在每个从节点上的执行情况。

TaskTracker的一个职责就是负责持续不断地与JobTracker通讯。如果JobTracker在指定的时间内没有收到来自TaskTracker的心跳,它会假定TaskTracker已经崩溃了,进而重新提交相应的任务到集群的其他节点中。

# HDFS文件系统工作原理

Hadoop分布式文件系统(HDFS)是一种被设计成适合运行在通用硬件上的分布式文件系统。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。它能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。要理解HDFS的内部工作原理，首先要理解什么是分布式文件系统。

## 1. 分布式文件系统

多台计算机联网协同工作(有时也称为一个集群)就像单台系统一样解决某种问题，这样的系统我们称之为分布式系统。分布式文件系统是分布式系统的一个子集，它们解决的问题就是数据存储。换句话说，它们是横跨在多台计算机上的存储系统。存储在分布式文件系统上的数据自动分布在不同的节点上。

分布式文件系统在大数据时代有着广泛的应用前景，它们为存储和处理来自网络和其它地方的超大规模数据提供所需的扩展能力。

## 2. 分离元数据和数据：NameNode和DataNode

存储到文件系统中的每个文件都有相关联的元数据。元数据包括了文件名、i节点(inode)数、数据块位置等，而数据则是文件的实际内容。

在传统的文件系统里，因为文件系统不会跨越多台机器，元数据和数据存储在同一台机器上。

为了构建一个分布式文件系统，让客户端在这种系统中使用简单，并且不需要知道其他客户端的活动，那么元数据需要在客户端以外维护。HDFS的设计理念是拿出一台或多台机器来保存元数据，并让剩下的机器来保存文件的内容。

NameNode和DataNode是HDFS的两个主要组件。其中，元数据存储在NameNode上，而数据存储在DataNode的集群上。NameNode不仅要管理存储在HDFS上内容的元数据，而且要记录一些事情，比如哪些节点是集群的一部分，某个文件有几份副本等。它还要决定当集群的节点宕机或者数据副本丢失的时候系统需要做什么。

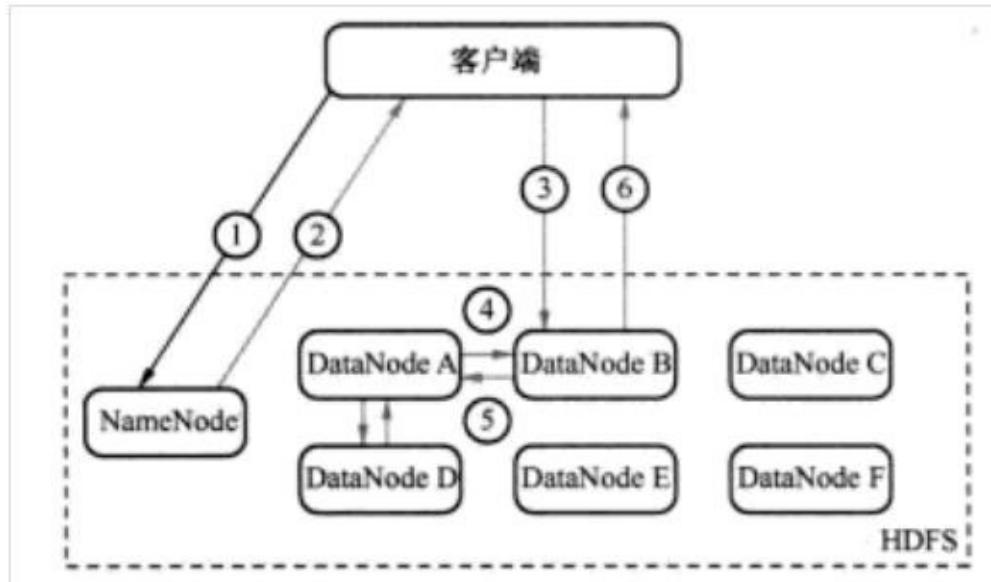
存储在HDFS上的每份数据片有多份副本(replica)保存在不同的服务器上。在本质上，NameNode是HDFS的Master(主服务器)，DataNode是Slave(从服务器)。

## 3. HDFS写过程

## 1. 基本概念

NameNode 负责管理存储在HDFS上所有文件的元数据，它会确认客户端的请求，并记录下文件的名字和存储这个文件的DataNode集合。它把该信息存储在内存中的文件分配表里。

例如，客户端发送一个请求给NameNode，说它要将“zhou.log”文件写入到HDFS。那么，其执行流程如图1所示。具体为： HDFS 写 图1 HDFS写过程示意图



第一步：客户端发消息给NameNode，说要将“zhou.log”文件写入。(如图1中的①)

第二步：NameNode发消息给客户端，叫客户端写到DataNode A、B和D，并直接联系DataNode B。(如图1中的②)

第三步：客户端发消息给DataNode B，叫它保存一份“zhou.log”文件，并且发送一份副本给DataNode A和DataNode D。(如图1中的③)

第四步：DataNode B发消息给DataNode A，叫它保存一份“zhou.log”文件，并且发送一份副本给DataNode D。(如图1中的④)

第五步：DataNode A发消息给DataNode D，叫它保存一份“zhou.log”文件。(如图1中的⑤)

第六步：DataNode D发确认消息给DataNode A。(如图1中的⑥)

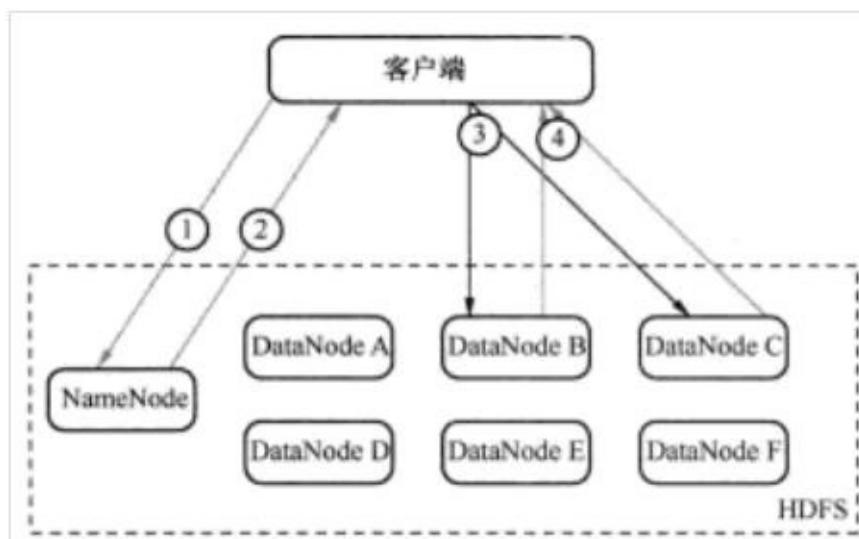
第七步：DataNode A发确认消息给DataNode B。(如图1中的⑦)

第八步：DataNode B发确认消息给客户端，表示写入完成。(如图1中的⑧)

在分布式文件系统的设计中，挑战之一是如何确保数据的一致性。对于HDFS来说，直到所有要保存数据的DataNodes确认它们都有文件的副本时，数据才被认为写入完成。因此，数据一致性是在写的阶段完成的。一个客户端无论选择从哪个DataNode读取，都将得到相同的数据。

## 4. HDFS读过程

为了理解读的过程，可以认为一个文件是由存储在DataNode上的数据块组成的。客户端查看之前写入的内容的执行流程如图2所示，具体步骤为：HDFS读写图2 HDFS读过程示意图



第一步：客户端询问NameNode它应该从哪里读取文件。(如图2中的①)

第二步：NameNode发送数据块的信息给客户端。(数据块信息包含了保存着文件副本的DataNode的IP地址，以及DataNode在本地硬盘查找数据块所需要的数据块ID。)(如图2中的②)

第三步：客户端检查数据块信息，联系相关的DataNode，请求数据块。(如图2中的③)

第四步：DataNode返回文件内容给客户端，然后关闭连接，完成读操作。(如图2中的④)

客户端并行从不同的DataNode中获取一个文件的数据块，然后联结这些数据块，拼成完整的文件。

## 5. 通过副本快速恢复硬件故障

当一切运行正常时，DataNode会周期性发送心跳信息给NameNode(默认是每3秒钟一次)。如果NameNode在预定的时间内没有收到心跳信息(默认是10分钟)，它会认为DataNode出问题了，把它从集群中移除，并且启动一个进程去恢复数据。

DataNode可能因为多种原因脱离集群，如硬件故障、主板故障、电源老化和网络故障等。

对于HDFS来说，丢失一个DataNode意味着丢失了存储在它的硬盘上的数据块的副本。假如在任意时间总有超过一个副本存在(默认3个)，故障将不会导致数据丢失。当一个硬盘故障时，HDFS会检测到存储在该硬盘的数据块的副本数量低于要求，然后主动创建需要的副本，以达到满副本数状态。

## 6. 跨多个DataNode切分文件

在HDFS里，文件被切分成数据块，通常每个数据块64MB~128MB，然后每个数据块被写入文件系统。同一个文件的不同数据块不一定保存在相同的DataNode上。这样做的好处是，当对这些文件执行运算时，能够通过并行方式读取和处理文件的不同部分。

当客户端准备写文件到HDFS并询问NameNode应该把文件写到哪里时，NameNode会告诉客户端，那些可以写入数据块的DataNode。写完一批数据块后，客户端会回到NameNode获取新的DataNode列表，把下一批数据块写到新列表中的DataNode上。

## 7. HDFS数据存储单元(block)

文件被切分成固定大小的数据块

- 默认数据块大小为64MB, 可配置
- 若文件大小不到64MB, 则单独保存到一个block

一个文件存储方式

- 按大小切分成若干个block, 存储到不同节点上
- 默认情况下每个block都有三个副本
- block大小和副本数通过Client端上传文件时设置, 文件上传成功后副本数可以变更, Block Size不可以变更

## 二 MPP与MapReduce之争

这些年大数据概念已经成为IT界的热门，我们经常也会在新闻和报纸中看到。大数据概念中最为关键的技术就是数据库管理系统，伴随着hadoop和MapReduce技术的流行，大数据的数据库中Hive和Spark等新型数据库脱颖而出；而另一个技术流派是基于传统的并行数据库技术演化而来的大规模并行处理（MPP）数据库比如GreenPlum和HAWQ也在最近几年突飞猛进，这两种流派都有对应的比较知名的产品，他们都已得到了市场的认可。

然而对于一个不是搞大数据的门外汉，如何理解大数据概念，如何根据自身的需求来选择对应的数据库管理系统，就成了很多用户很关心的话题。我们将采用比较通俗易懂的介绍，让大家从本质上认识这些大数据库管理系统的技术实现和应用场景。

MPP是一种海量数据实时分析架构，MPP作为一种不共享架构，每个节点运行自己的操作系统和数据库，节点之间信息交互只能通过网络连接实现，横向扩展是MPP数据库的主要设计目标，MPP数据库的核心仍是关系型数据库模型。

## MPP和Hadoop的区别与联系

其实MPP架构的关系型数据库与Hadoop的理论基础是极其相似的，都是将运算分布到节点中独立运算后进行结果合并。区别仅仅在于前者跑的是SQL，后者底层处理则是MapReduce程序。MPP也支持横向扩展，但是这种扩展一般是扩到100左右，而Hadoop一般可以扩展1000+，这也是主要区别之一。原因可以从CAP理论上解释。因为MPP始终还是DB，一定要考虑C(Consistency)，其次考虑A(Availability)，最后才在可能的情况下尽量做好P(Partition-tolerance)。而Hadoop就是为了并行处理和存储设计的，所有数据都是以文件存储，所以优先考虑的是P，然后是A，最后再考虑C。所以后者的可扩展性当然好于前者。

以下几个方面制约了MPP数据库的扩展

- 高可用：MPP DB是通过Hash计算来确定数据行所在的物理机器(而Hadoop无需此操作)，对存储位置的不透明导致MPP的高可用很难办。
- 并行任务：数据是按照Hash来切分了，但是任务没有。每个任务，无论大小都要到每个节点去走一圈。

- 文件系统：数据切分了，但是文件数没有变少，每个表在每个节点上一定有一到多个文件。同样节点数越多，存储的表就越多，导致每个文件系统上有上万甚至十万多个文件。
- 网络瓶颈：MPP强调对等的网络，点对点的连接也消耗了大量的网络带宽，限制了网络上的线性扩展(想象一台机器可能要给1000台机器发送信息)。更多的节点并没有提供更高的网络带宽，反而导致每个组节点间平均带宽降低。
- 其他关系数据库的枷锁：比如锁、日志、权限、管理节点瓶颈等均限制了MPP规模的扩大。

但是MPP数据库有对SQL的完整兼容和一些事务处理功能，对于用户来说，在实际的使用场景中，如果数据扩展需求不是特别大，需要的处理节点不多，数据都是结构化数据，习惯使用传统RDBMS的很多特性的场景，可以考虑MPP如Greenplum/Gbase等。但如果有很多非结构化数据，或者数据量巨大，有需要扩展到成百上千个数据节点需求的，这个时候Hadoop是更好的选择。

## 2. 原理漫画

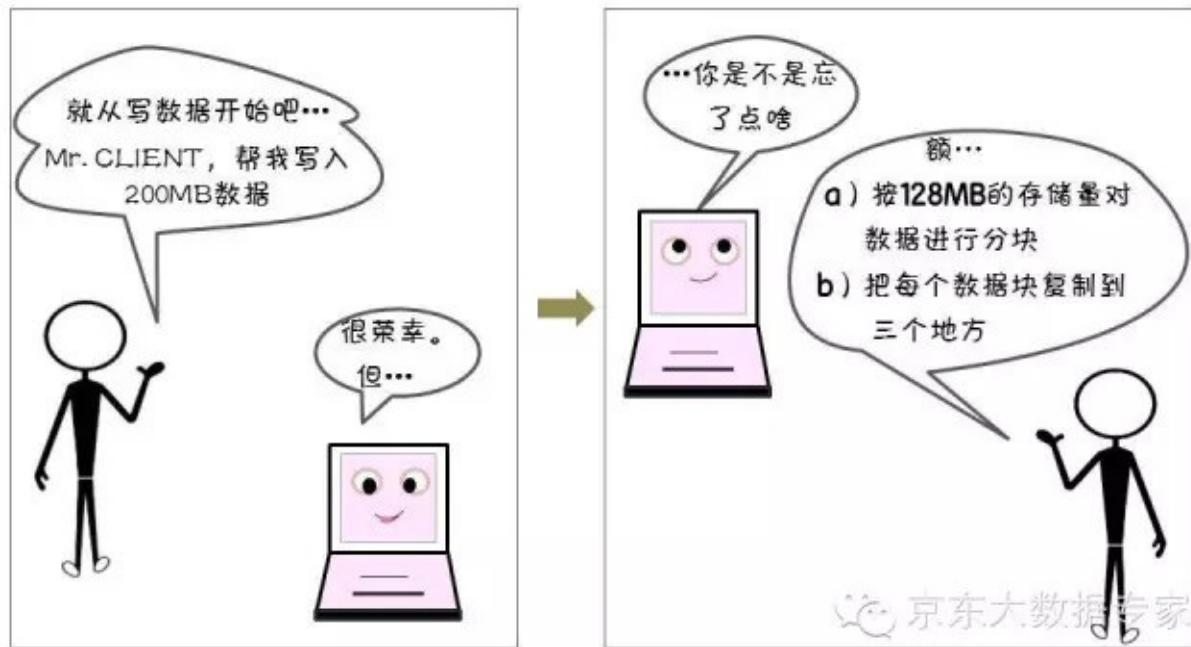
### HDFS存储原理

#### 1. 发送写数据请求



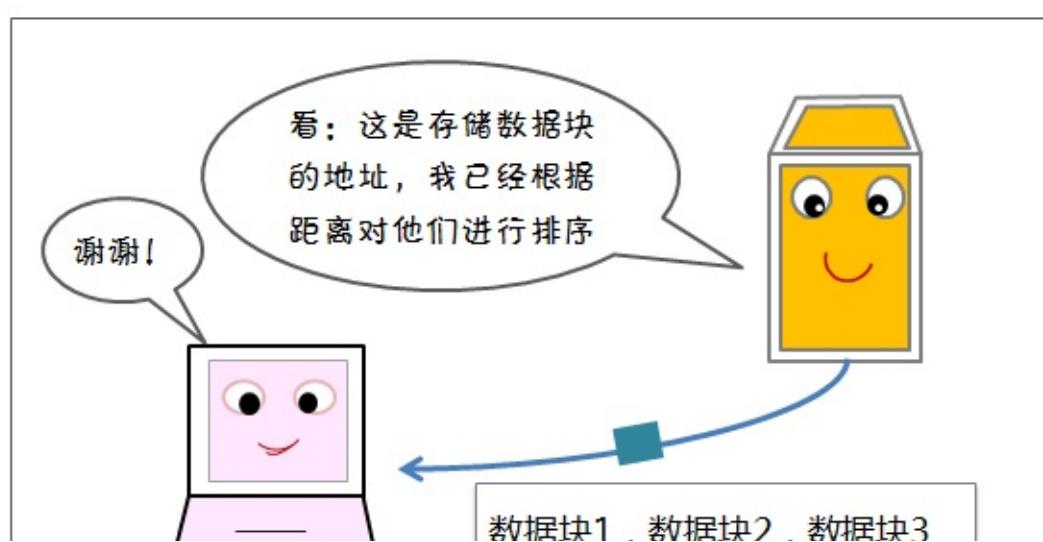
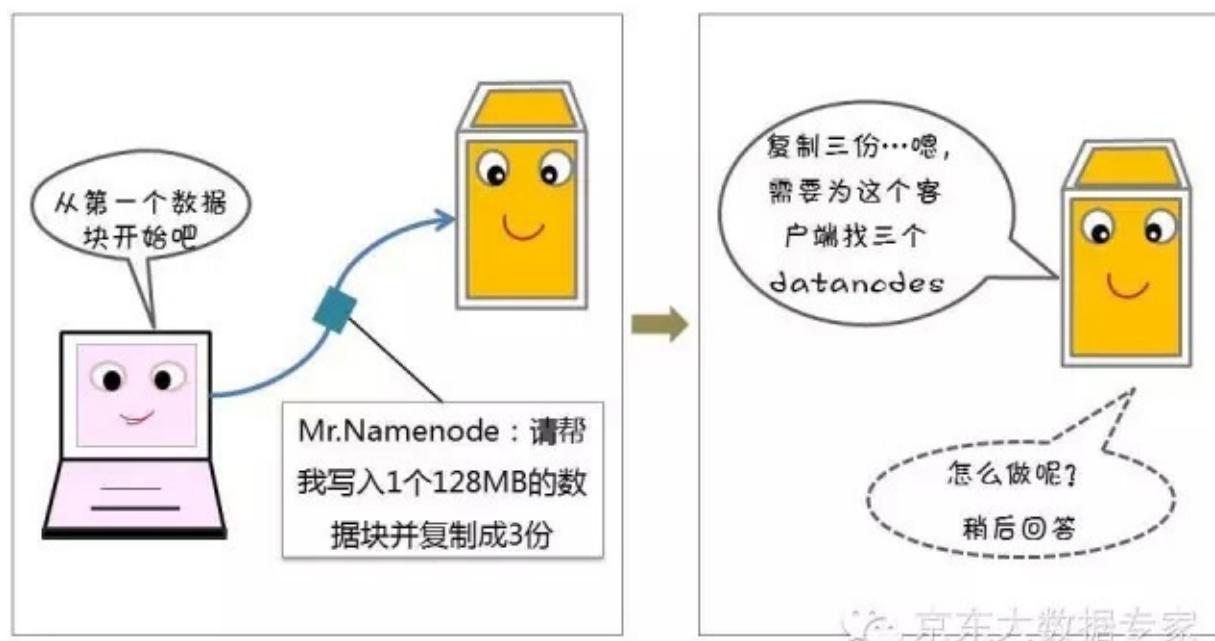
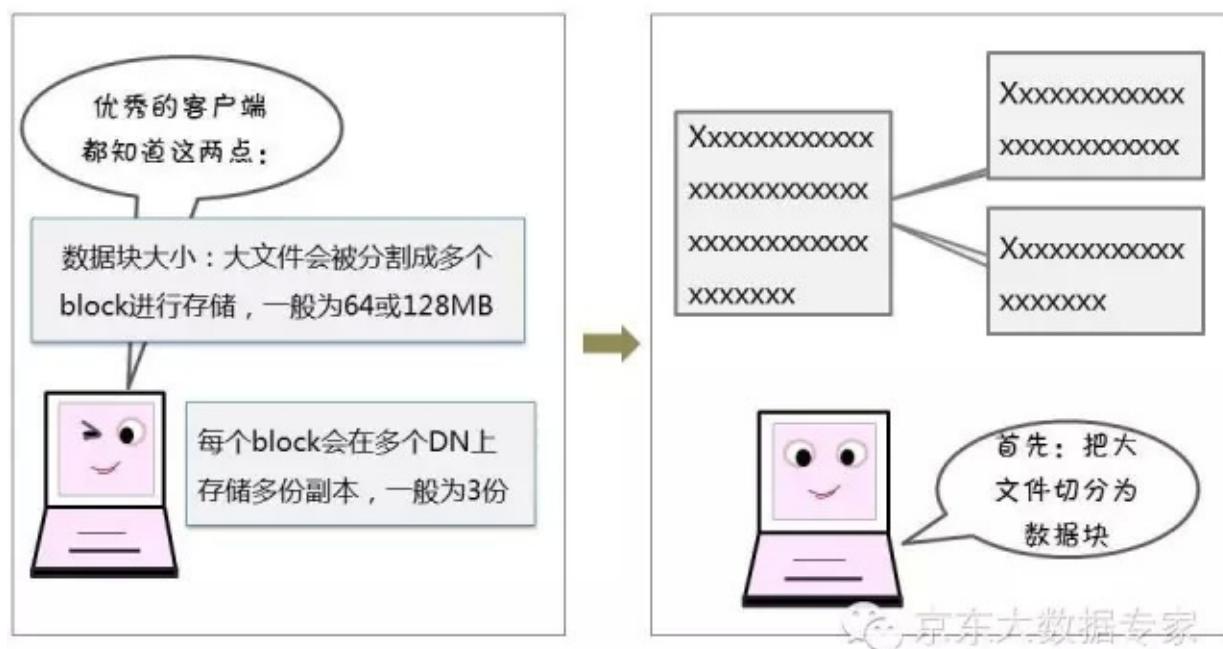
#### 2. 文件切分

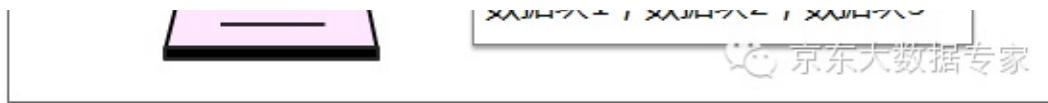
## 2. 原理漫画



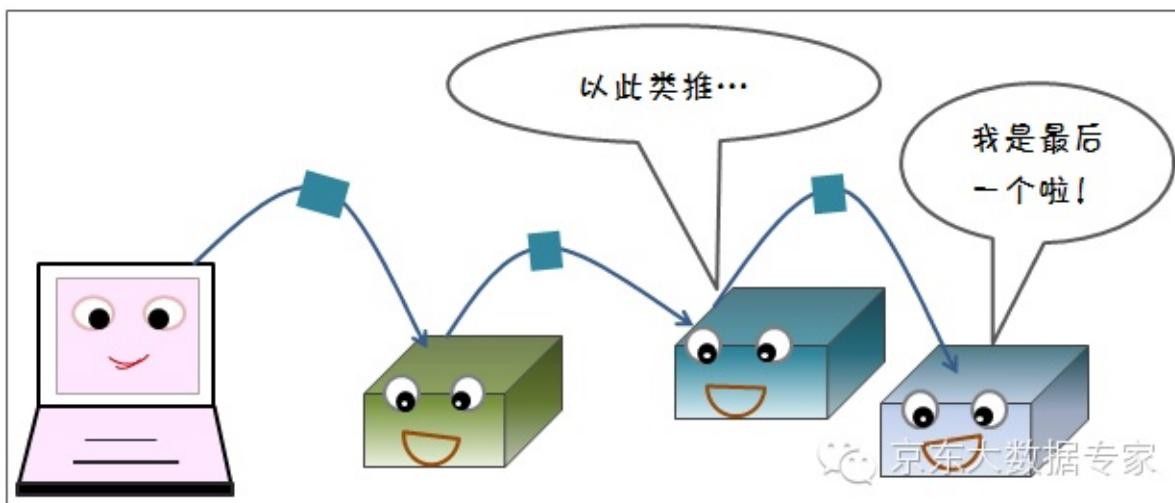
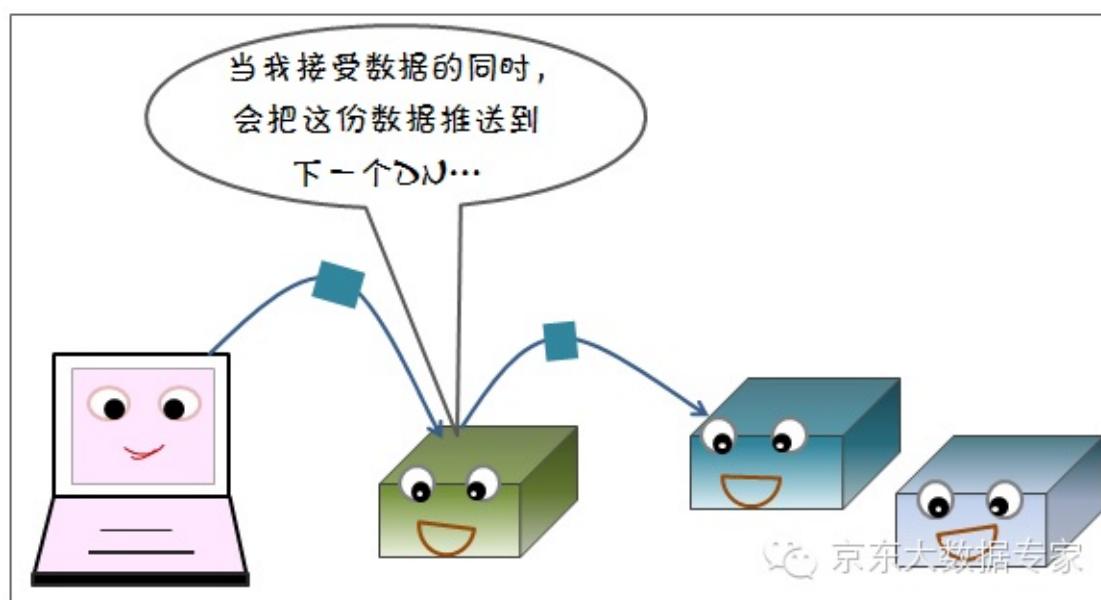
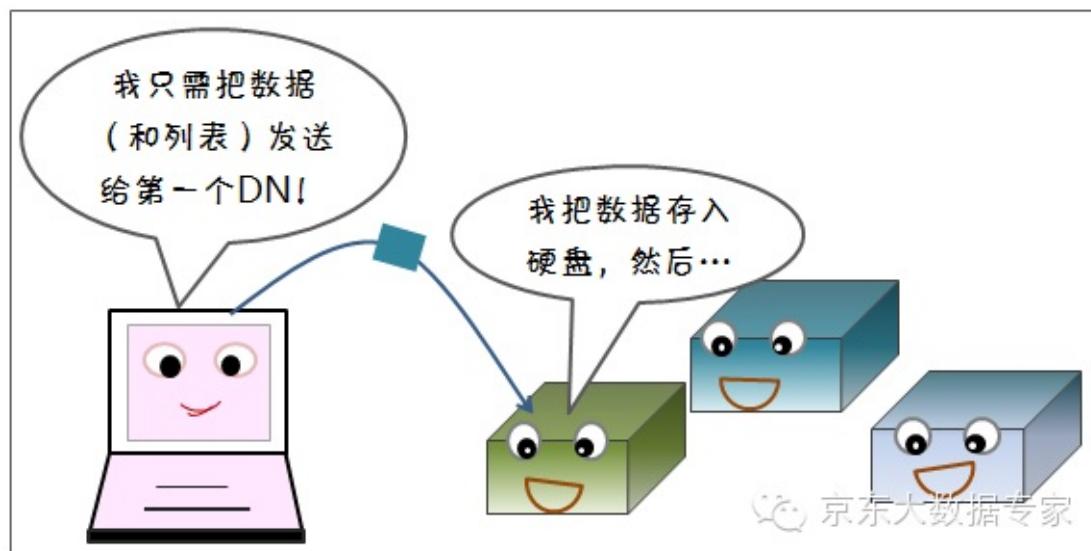
## 3.DN分配

## 2. 原理漫画



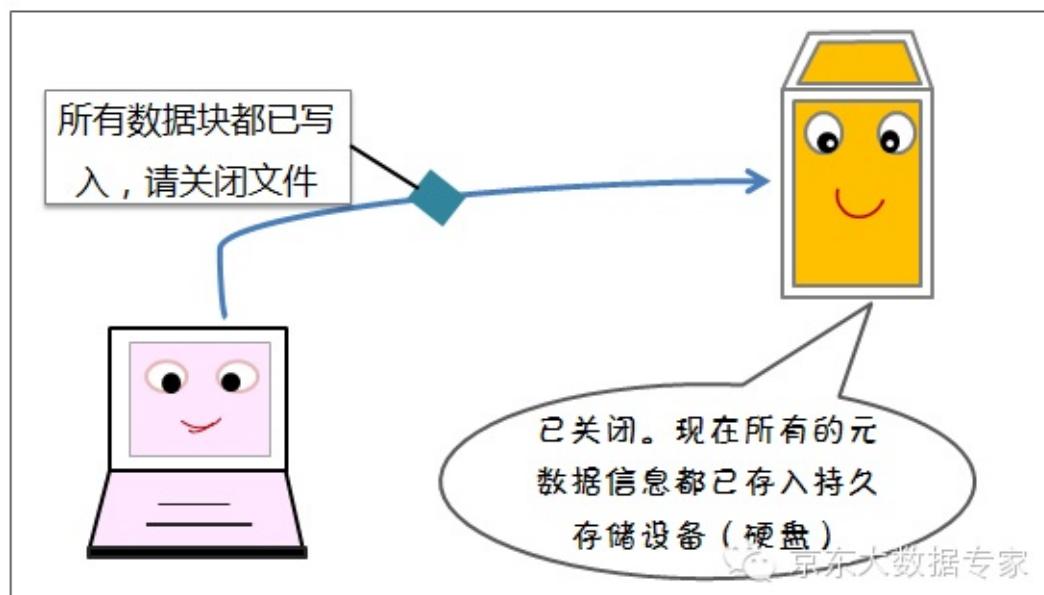
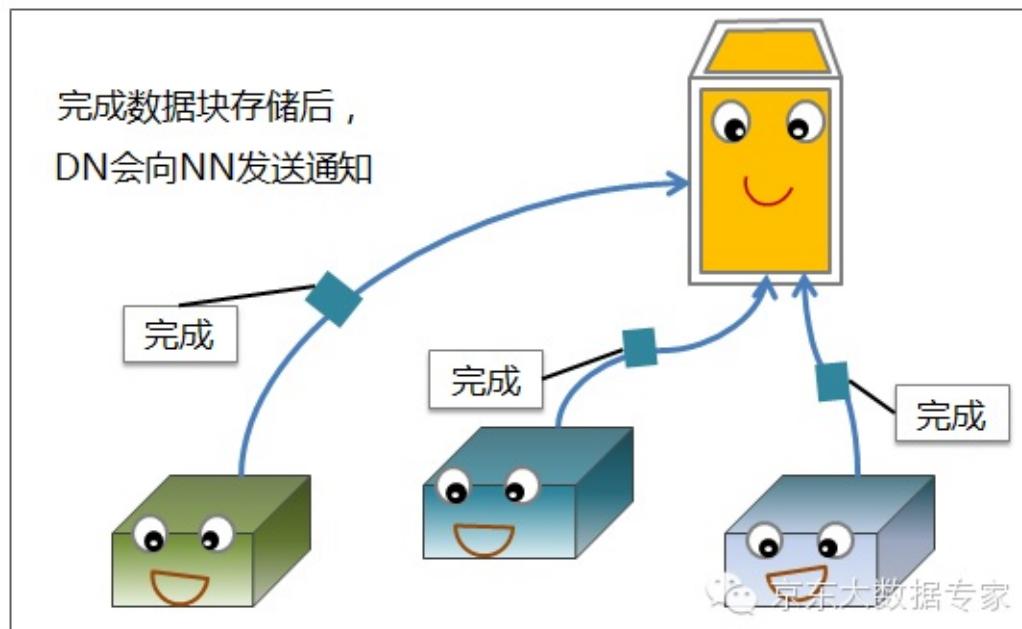


## 4. 数据写入



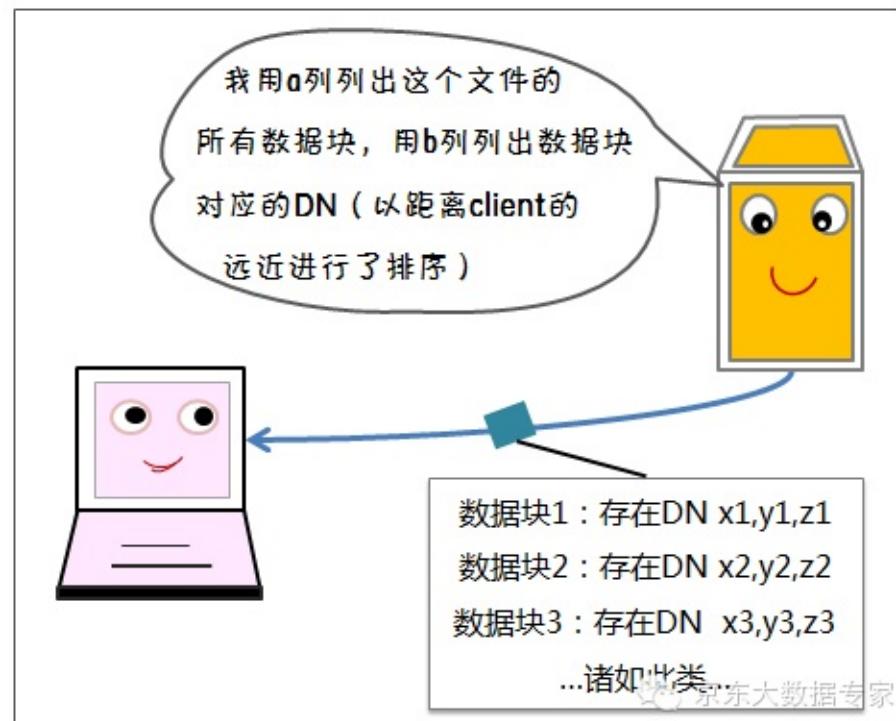
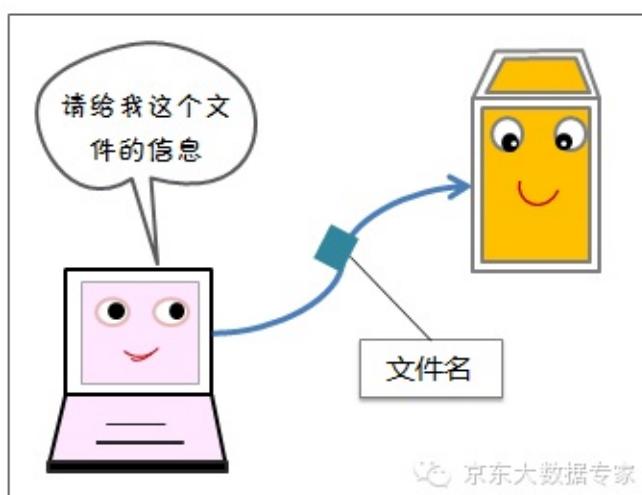
## 5. 完成写入

## 2. 原理漫画



## 6. 用户读文件

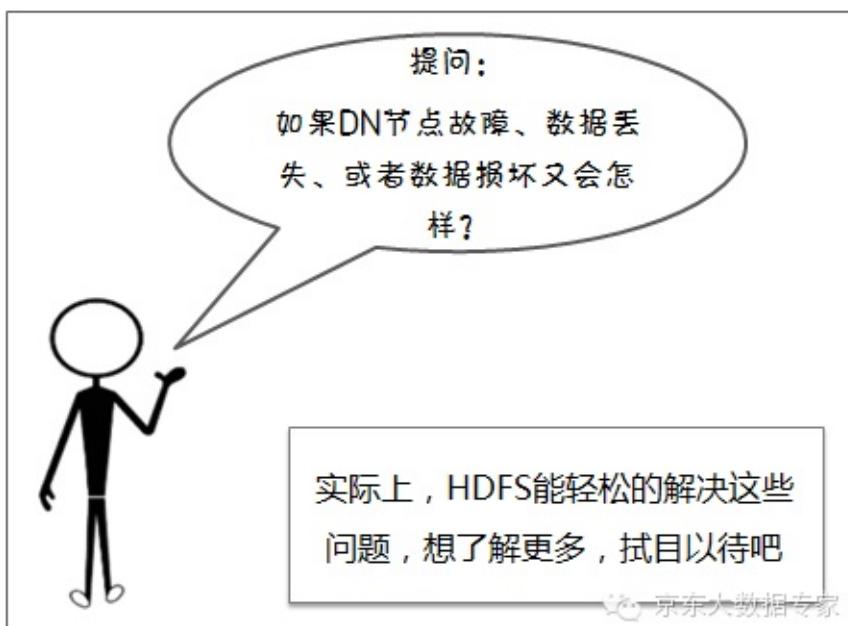
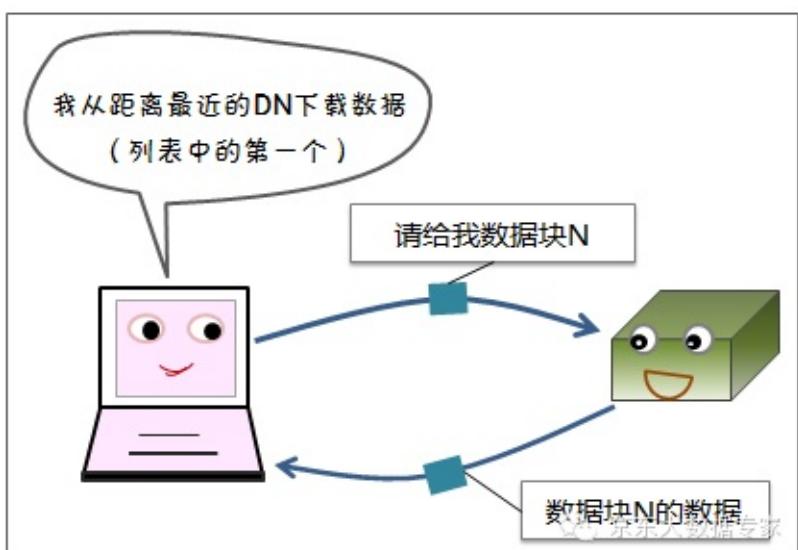
## 2. 原理漫画



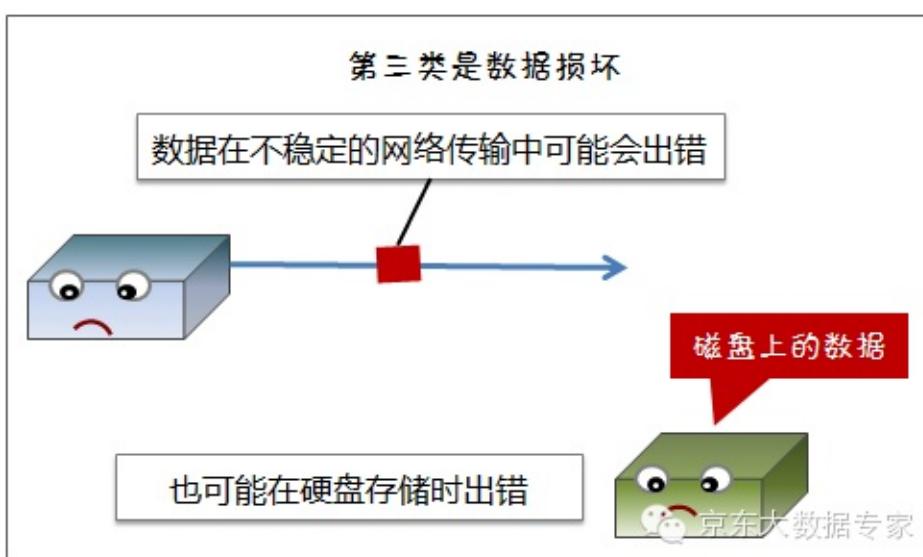
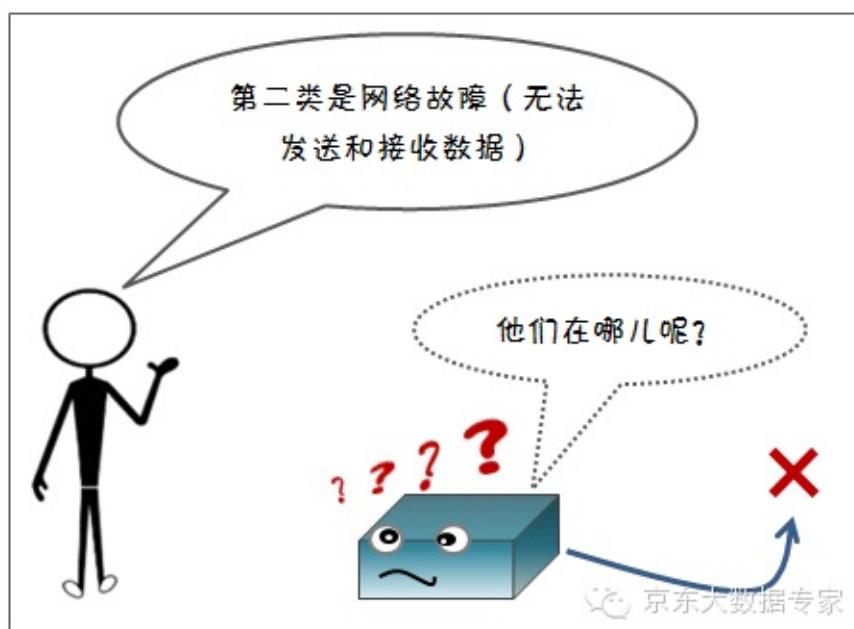
## 2. 原理漫画



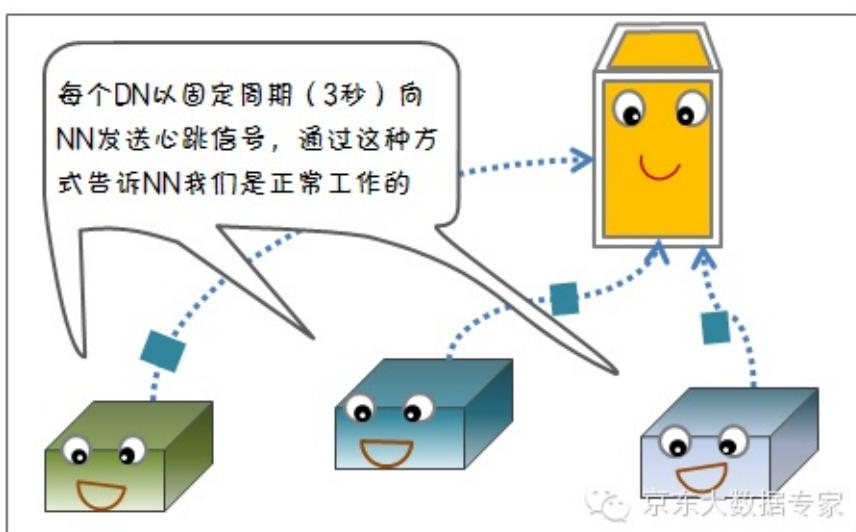
## 7. 下载数据



## 8. HDFS容错机制

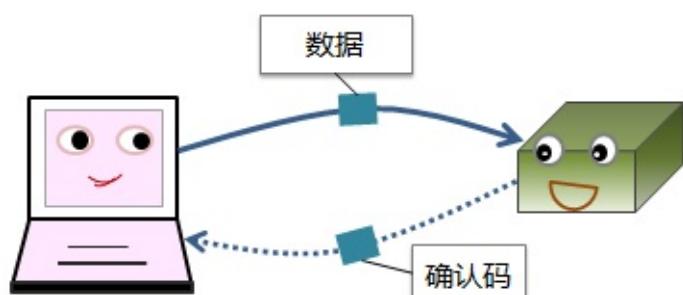


## 9. 节点监测机制



## 2. 原理漫画

只要发送了数据，接收方就会返回确认码



如果经过几次重试后，还是没有收到确认码，

发送方会认为主机挂了或者网络~~在故障~~数据专家

所有DN都会定期向NN发送数据块的存储状况

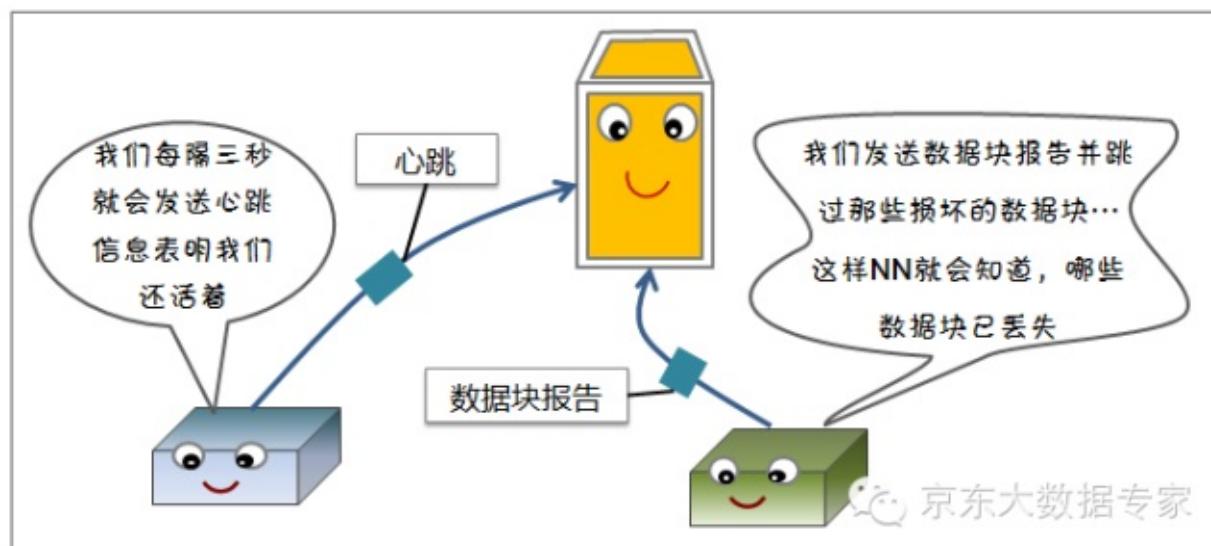
数据块存储清单  
京东大数据专家

在发送数据块报告前，  
我会先检查总和校验码是否正确。  
如果数据存在错误我就不发送

该数据块的信息

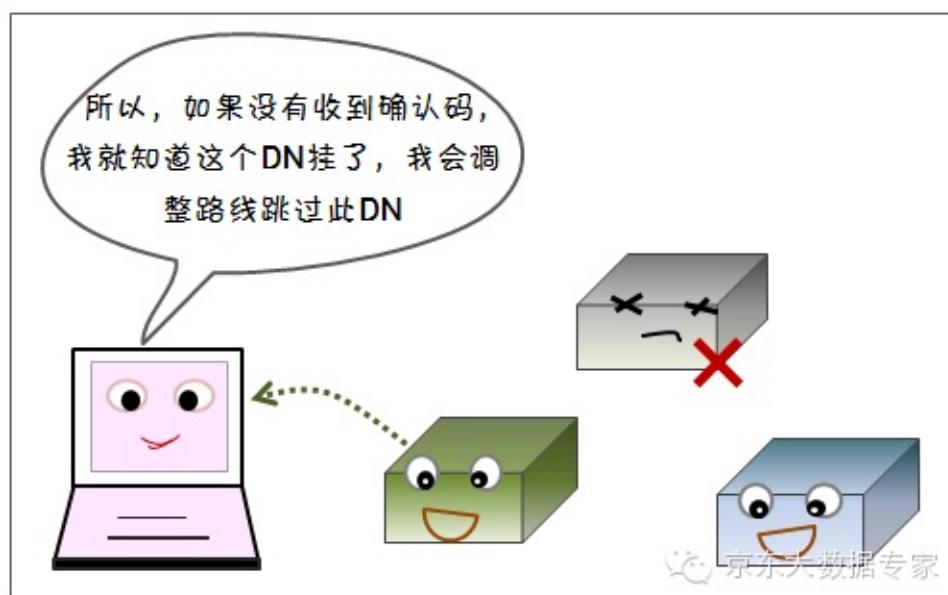
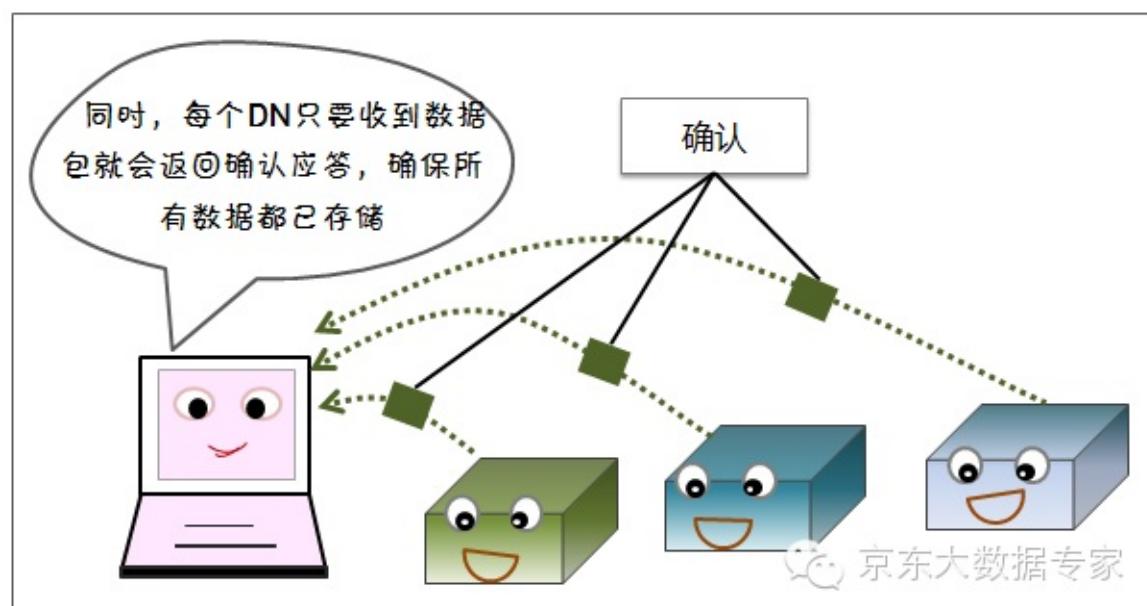
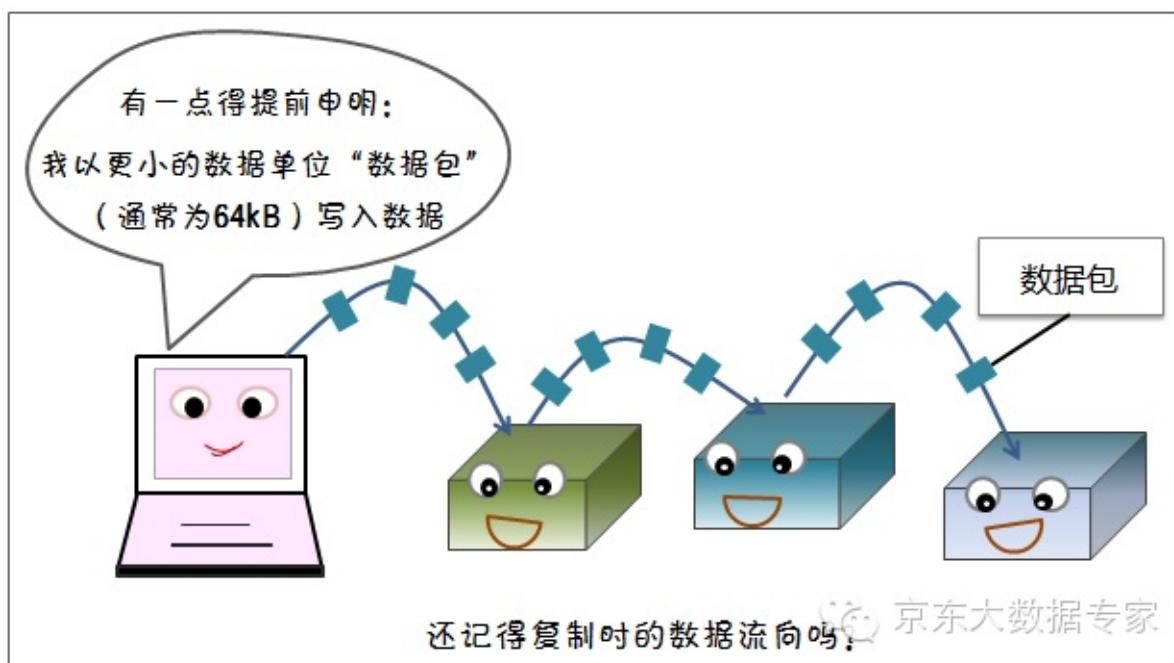
我存有四  
个数据块

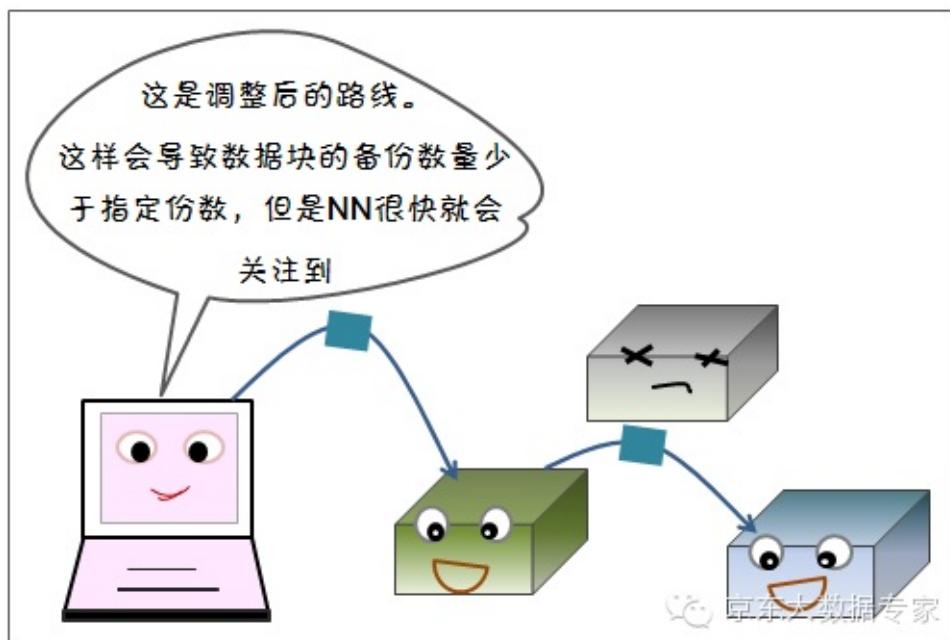
我想他存有五个数  
据块…这么说是有一  
个数据块损坏了  
京东大数据专家



## 10. 写容错

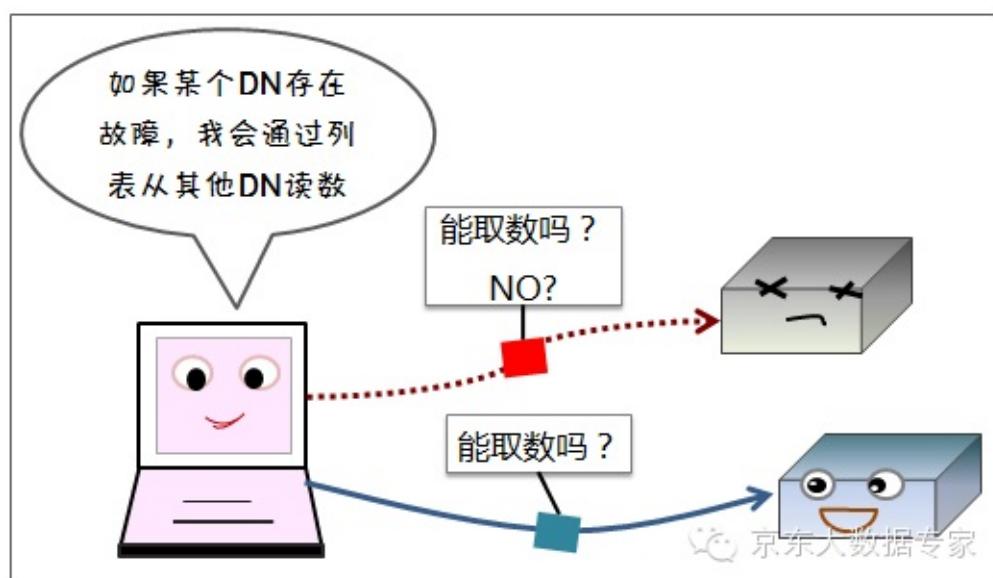
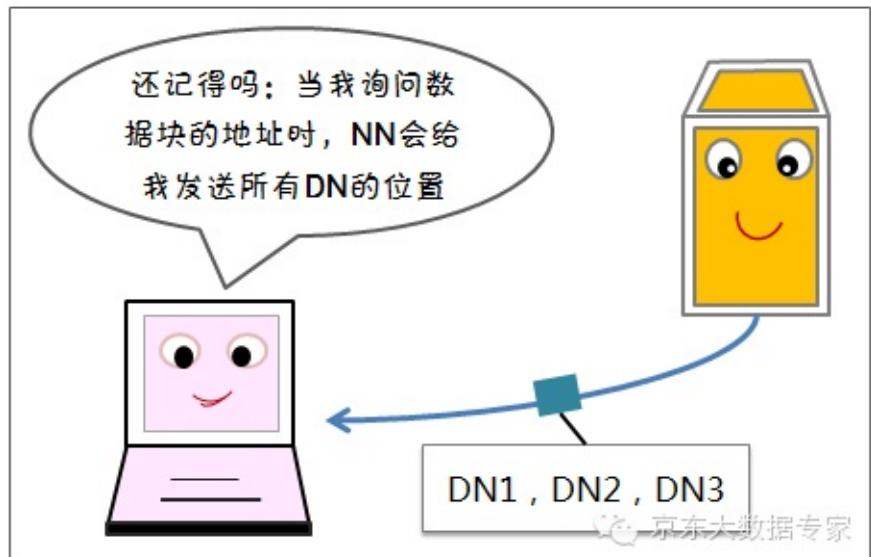
## 2. 原理漫画





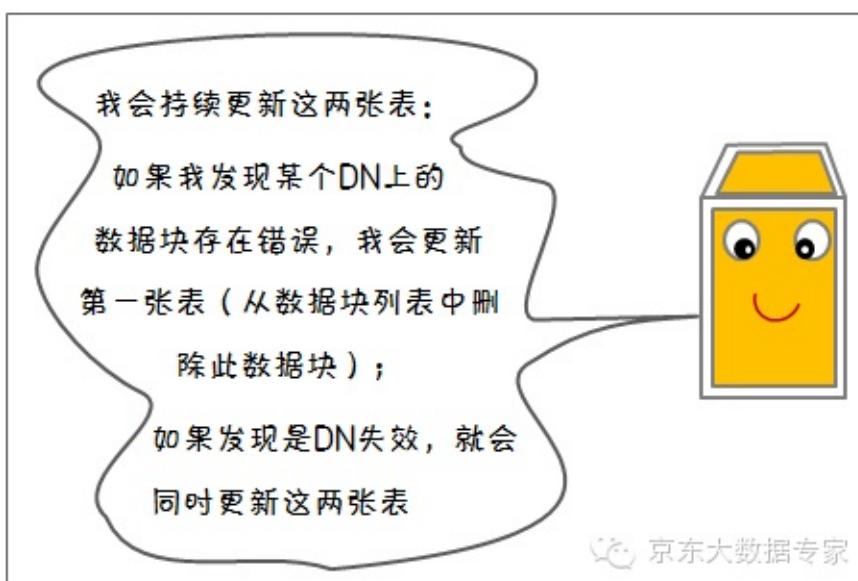
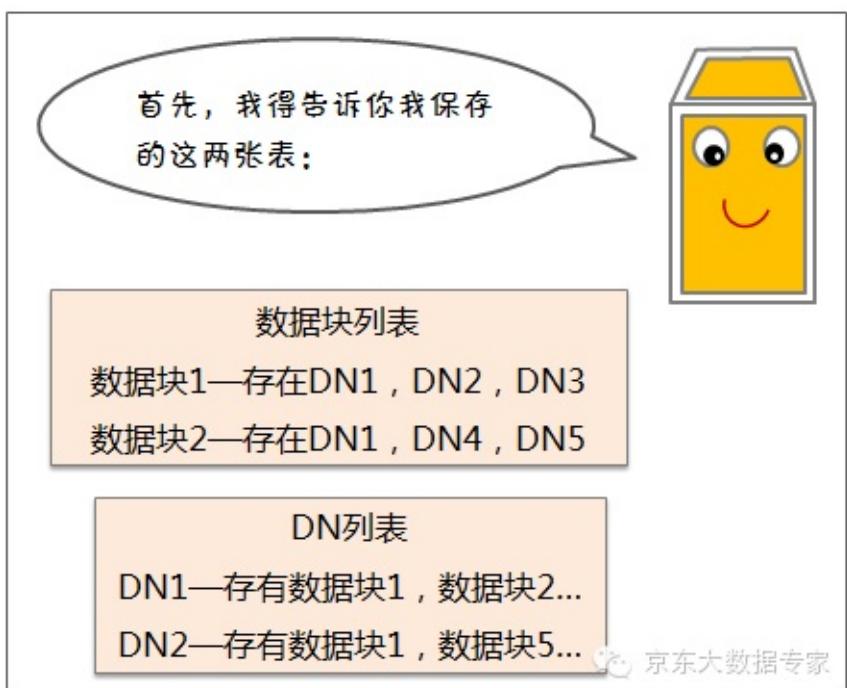
## 11. 读容错

## 2. 原理漫画



## 12.DN失效

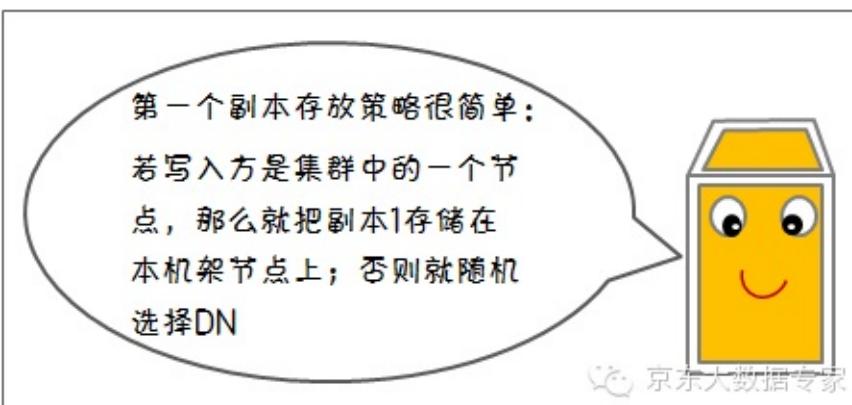
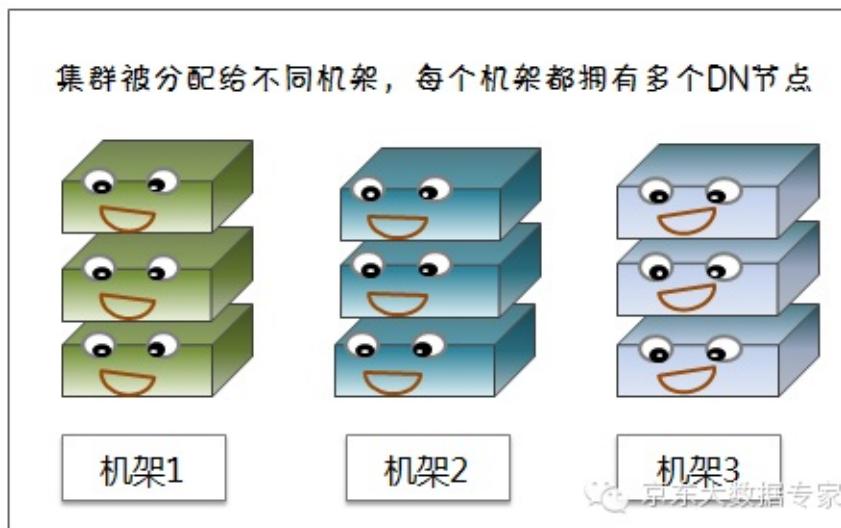
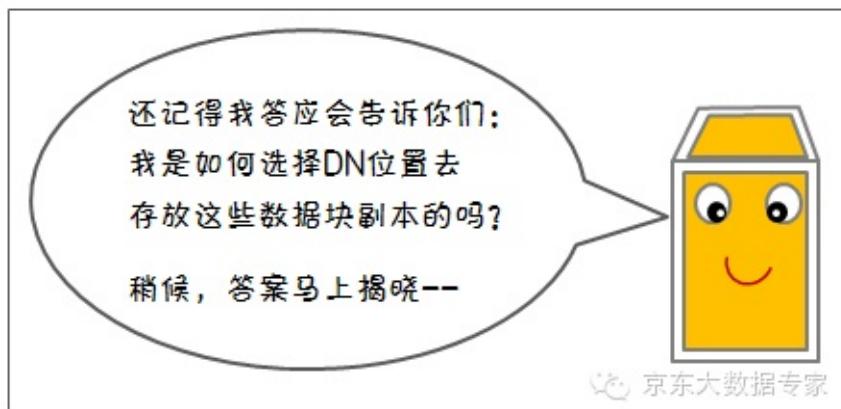
## 2. 原理漫画

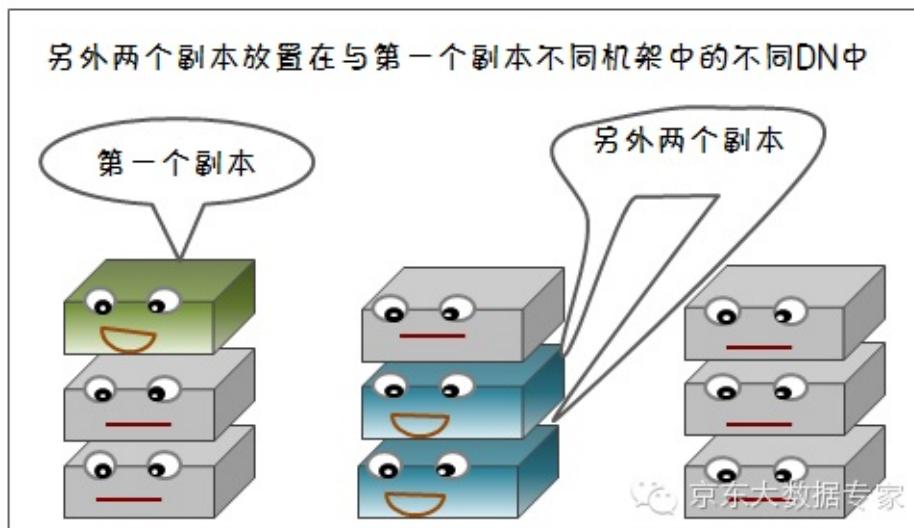




### 13. 备份规则

## 2. 原理漫画





## 3.Hadoop部署技术选型

### 一、背景介绍

生产环境中，hadoop的版本选择是一个公司架构之时，很重要的一个考虑因素。

**Apache Hadoop** : Apache Hadoop 是一款支持数据密集型分布式应用并以Apache 2.0 许可协议发布的开源软件框架。它支持在商品硬件构建的大型集群上运行的应用程序。Hadoop是根据Google公司发表的MapReduce和Google档案系统的论文自行实作而成。称为社区版Hadoop。

**第三方发行版Hadoop** : Hadoop遵从Apache开源协议，用户可以免费地任意使用和修改Hadoop，也正因此，市面上出现了很多Hadoop版本。其中有很多厂家在Apache Hadoop的基础上开发自己的Hadoop产品，比如Cloudera的CDH， Hortonworks的HDP，MapR的MapR产品等。

### 二、社区版本与第三方发行版本的比较

#### Apache社区版本

- 优点：

完全开源免费。社区活跃 文档、资料详实

- 缺点：

复杂的版本管理。版本管理比较混乱的，各种版本层出不穷，让很多使用者不知所措。复杂的集群部署、安装、配置。通常按照集群需要编写大量的配置文件，分发到每一台节点上，容易出错，效率低下。复杂的集群运维。对集群的监控，运维，需要安装第三方的其他软件，如ganglia，nagois等，运维难度较大。复杂的生态环境。在Hadoop生态圈中，组件的选择、使用，比如Hive，Mahout，Sqoop，Flume，Spark，Oozie等等，需要大量考虑兼容性的问题，版本是否兼容，组件是否有冲突，编译是否能通过等。经常会浪费大量的时间去编译组件，解决版本冲突问题。

#### 第三方发行版本（如**CDH**，**HDP**，**MapR**等）

- 优点：

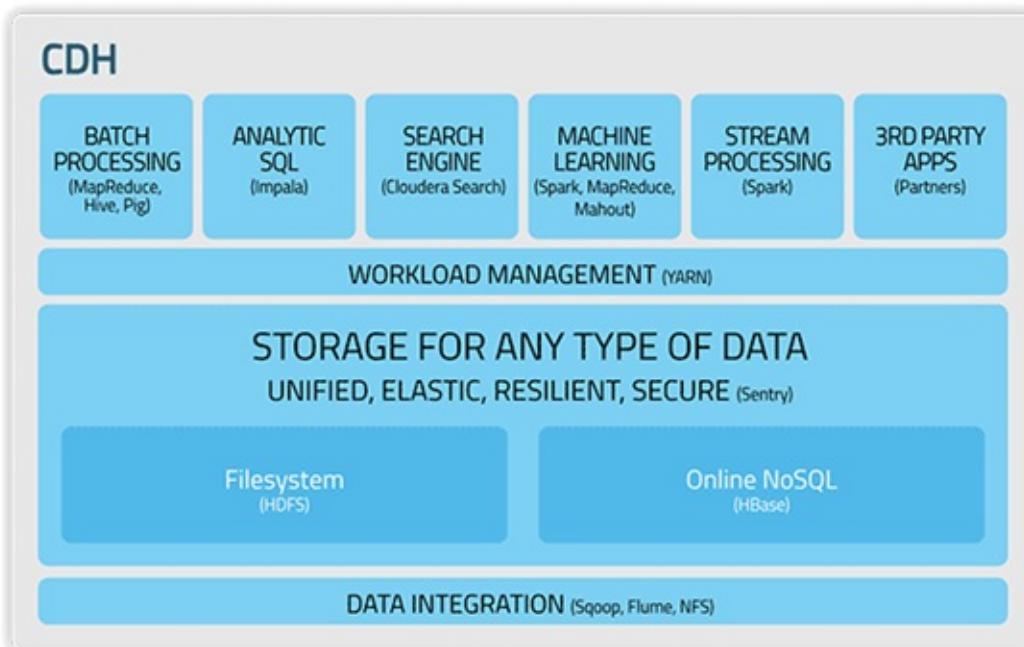
基于Apache协议，100%开源。版本管理清晰。比如Cloudera，CDH1，CDH2，CDH3，CDH4等，后面加上补丁版本，如CDH4.1.0 patch level 923.142，表示在原生态Apache Hadoop 0.20.2基础上添加了1065个patch。比Apache Hadoop在兼容性、安全性、稳定性上有增强。第三方发行版通常都经过了大量的测试验证，有众多部署实例，大量的运行到各种生产环境。版本更新快。通常情况，比如CDH每个季度会有一个update，每一年会有一个release。基于稳定版本Apache Hadoop，并应用了最新Bug修复或Feature的patch 提供了部署、安装、配置工具，大大提高了集群部署的效率，可以在几个小时内部署好集群。运维简单。提供了管理、监控、诊断、配置修改的工具，管理配置方便，定位问题快速、准确，使运维工作简单，有效。

- 缺点：

涉及到厂商锁定的问题。（可以通过技术解决）

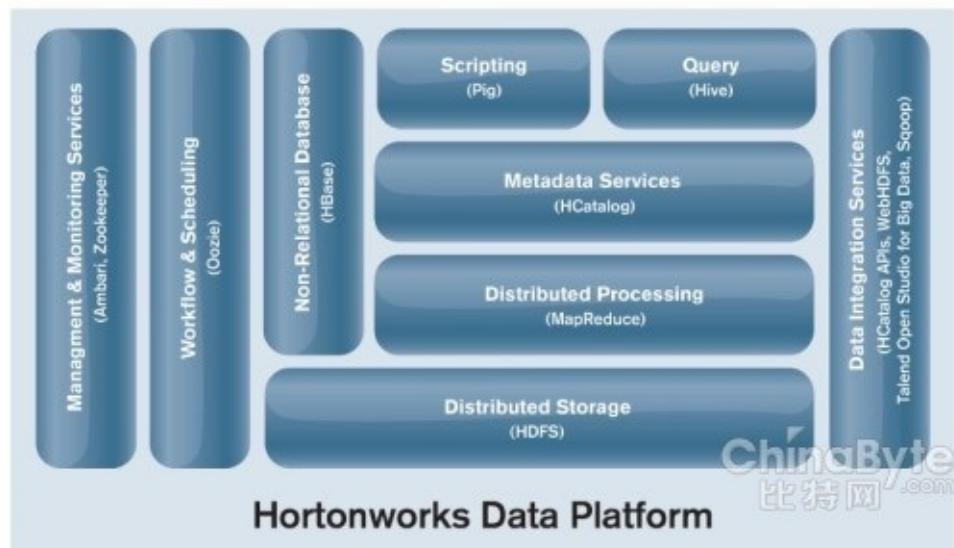
### 三、第三方发行版本的比较

**Cloudera**：最成型的发行版本，拥有最多的部署案例。提供强大的部署、管理和监控工具。Cloudera开发并贡献了可实时处理大数据的Impala项目。

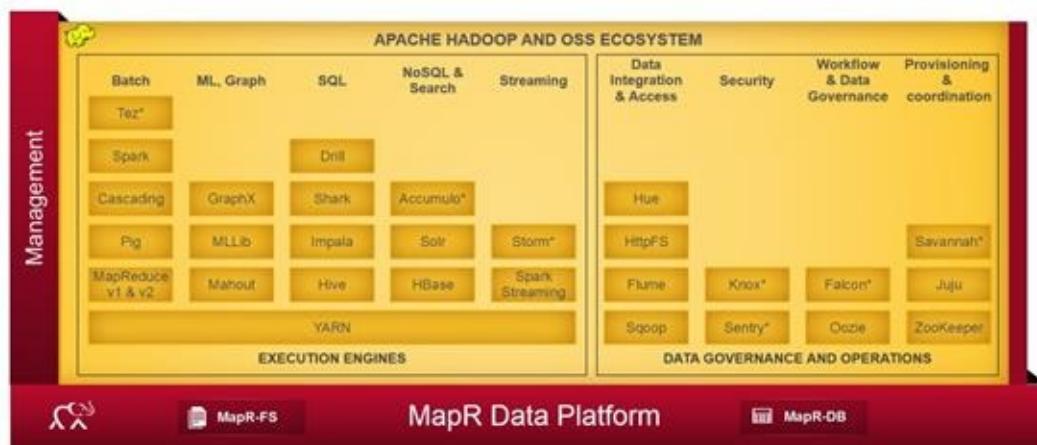


**Hortonworks**：不拥有任何私有（非开源）修改地使用了100%开源Apache Hadoop的唯一提供商。Hortonworks是第一家使用了Apache HCatalog的元数据服务特性的提供商。并且，它们的Stinger开创性地极大地优化了Hive项目。Hortonworks为

入门提供了一个非常好的，易于使用的沙盒。Hortonworks开发了很多增强特性并提交至核心主干，这使得Apache Hadoop能够在包括Windows Server和Windows Azure在内的Microsoft Windows平台上本地运行。



**MapR**：与竞争者相比，它使用了一些不同的概念，特别是为了获取更好的性能和易用性而支持本地Unix文件系统而不是HDFS（使用非开源的组件）。可以使用本地Unix命令来代替Hadoop命令。除此之外，MapR还凭借诸如快照、镜像或有状态的故障恢复之类的高可用性特性来与其他竞争者相区别。该公司也领导着Apache Drill项目，本项目是Google的Dremel的开源项目的重新实现，目的是在Hadoop数据上执行类似SQL的查询以提供实时处理。



**Amazon Elastic Map Reduce (EMR)**：区别于其他提供商的是，这是一个托管的解决方案，其运行在由Amazon Elastic Compute Cloud (Amazon EC2) 和 Amazon Simple Storage Service (Amazon S3) 组成的网络规模的基础设施之上。除了Amazon的发行版本之外，你也可以在EMR上使用MapR。临时集群是主要的使用情形。如果你需要一次性的或不常见的大数据处理，EMR可能会为你节省大笔开支。然而，这也存在不利之处。其只包含了Hadoop生态系统中Pig和Hive项目，在

默认情况下不包含其他很多项目。并且，EMR是高度优化成与S3中的数据一起工作的，这种方式会有较高的延时并且不会定位位于你的计算节点上的数据。所以处于EMR上的文件IO相比于你自己的Hadoop集群或你的私有EC2集群来说会慢很多，并有更大的延时。

以上为具有代表性的第三方发行版，另外的发行版则不一一列举了。

## 四、选择决定

当我们决定是否采用某个软件用于开源环境时，通常需要考虑以下几个因素：

- (1) 是否为开源软件，即是否免费。
- (2) 是否有稳定版，这个一般软件官方网站会给出说明。
- (3) 是否经实践验证，这个可通过检查是否有一些大点的公司已经在生产环境中使用知道。
- (4) 是否有强大的社区支持，当出现一个问题时，能够通过社区、论坛等网络资源快速获取解决方法。

综上所述，考虑到大数据平台高效的部署和安装，中心化的配置管理，使用过程中的稳定性、兼容性、扩展性，以及未来较为简单、高效的运维，遇到问题低廉的解决成本。

个人建议使用第三方发行版本。

## 第三章 Hadoop的伪分布式搭建

### 一 JDK的安装

1. 下载JDK安装包,建议去Oracle官方下载,地址自行百度
2. 下载Hadoop2.6的安装包,建议官方下载,地址自行百度
3. 如果是在Windows端进行终端操作,建议使用XFTP与XShell,有Free版本
4. 之后用XFTP将JDK安装包与Hadoop安装包上传到实验主机上
5. 将Java SDK解压,并将解压文件复制到/usr/lib/jvm中
6. 配置环境变量
7. 如果系统中已经有默认的OpenJavaSDK的话,这里我们需要修改默认SDK
8. 检测JavaSDK是否成功安装

```
#新安装的服务器,建议更新一下安装源列表,同时安装nano,用VI编辑不是太方便
#sudo aptitude update
#sudo aptitude upgrade
#sudo aptitude install nano

chu888chu888@ubuntu1:~$ tar xvfz jdk-8u65-linux-x64.gz
chu888chu888@ubuntu1:~$ sudo cp -r jdk1.8.0_65/ /usr/lib/jvm/
chu888chu888@ubuntu1:/usr/lib/jvm$ sudo nano /etc/profile

#修改内容如下,注意大小写
#在环境变量中的配置中,有一点需要指出就是如果只是编辑~/.profile的话这个变量
#的生效只是针对当前用户的.
#如果想要其在全局生效的话,建议更新/etc/profile,这是一个全局的.

export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

chu888chu888@ubuntu1:/usr/lib/jvm$ source /etc/profile
chu888chu888@ubuntu1:/usr/lib/jvm$ env
chu888chu888@ubuntu1:/usr/lib/jvm$ java -version
java version "1.8.0_65"
```

```
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

#有一种极端情况就是,如果在本机已经安装了OpenJavaSDK,怎么办?
sudo update-alternatives --install /usr/bin/java java /usr/lib/j
vm/java/bin/java 300
sudo update-alternatives --install /usr/bin/javac javac /usr/lib
/jvm/java/bin/javac 300
sudo update-alternatives --install /usr/bin/jar jar /usr/lib/jvm
/java/bin/jar 300
sudo update-alternatives --install /usr/bin/javah javah /usr/lib
/jvm/java/bin/javah 300
sudo update-alternatives --install /usr/bin/javap javap /usr/lib
/jvm/java/bin/javap 300
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

## 二 Hadoop的用户创建

1. 创建hadoop用户组
2. 创建hadoop用户
3. 给hadoop用户添加权限,打开/etc/sudoers文件

```
chu888chu888@ubuntu1:/$ sudo addgroup hadoop
chu888chu888@ubuntu1:/$ sudo adduser -ingroup hadoop hadoop
chu888chu888@ubuntu1:/$ sudo nano /etc/sudoers

root    ALL=(ALL:ALL) ALL
hadoop  ALL=(ALL:ALL) ALL
```

## 三 SSH无密码登录

```
#第一次都要用ssh密码登录太麻烦,我们想办法采用无密码登录
chu888chu888@ubuntu2:~$ cd ~/.ssh/
chu888chu888@ubuntu2:~/ssh$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chu888chu888/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/chu888chu888/.ssh/id_rsa.
Your public key has been saved in /home/chu888chu888/.ssh/id_rsa.pub.
The key fingerprint is:
3a:51:9b:99:3e:65:9d:9a:b4:60:35:4f:79:d9:5b:89 chu888chu888@ubuntu2
The key's randomart image is:
+--[ RSA 2048]----+
|                   |
|                   ..o.|
|       . o oEo.o|
|       . * = o   o|
|       . S + +   .|
|       = = +      |
|       o o +      |
|       . .        |
|                   |
+-----+
chu888chu888@ubuntu2:~/ssh$ cat ./id_rsa.pub >./authorized_keys
chu888chu888@ubuntu2:~/ssh$ ssh localhost
#但是这里面有一个小问题就是,我是用chu888chu888这个用户做的,如果您想用
#hadoop用户登录的话,这个过程需要再来一次
hadoop@ubuntu2:~$ ssh hadoop@localhost
```

## 四 Hadoop的基本安装

Hadoop 2 可以通过

地址1:<http://mirror.bit.edu.cn/apache/hadoop/common/>

地址2:<http://mirrors.cnnic.cn/apache/hadoop/common/>

下载，本教程选择的是 2.6.0 版本，下载时请下载 hadoop-2.x.y.tar.gz 这个格式的文件，这是编译好的，另一个包含 src 的则是 Hadoop 源代码，需要进行编译才可使用。

下载时强烈建议也下载 hadoop-2.x.y.tar.gz.mds 这个文件，该文件包含了检验值可用于检查 hadoop-2.x.y.tar.gz 的完整性，否则若文件发生了损坏或下载不完整，Hadoop 将无法正常运行。

本文涉及的文件均通过浏览器下载，默认保存在“下载”目录中（若不是请自行更改 tar 命令的相应目录）。另外，如果你用的不是 2.6.0 版本，则将所有命令中出现的 2.6.0 更改为你所使用的版本。

```
# 列出md5检验值
# cat ~/下载/hadoop-2.6.0.tar.gz.mds | grep 'MD5'
# 2.7.1版本格式变了，可以用这种方式输出
# head -n 6 ~/下载/hadoop-2.7.1.tar.gz.mds
# 计算md5值，并转化为大写，方便比较
# md5sum ~/下载/hadoop-2.6.0.tar.gz | tr "a-z" "A-Z"
```

1. 将hadoop解压到/usr/local下
2. 修改bashrc的配置,加入内容
3. 修改hadoop-env.sh的配置
4. 测试

```
chu888chu888@ubuntu1:~$ sudo tar xvfz hadoop-2.6.0.tar.gz
chu888chu888@ubuntu1:~$ sudo cp -r hadoop-2.6.0 /usr/local/hadoop
chu888chu888@ubuntu1:~$ sudo chmod -R 775 /usr/local/hadoop/
chu888chu888@ubuntu1:~$ sudo chown -R hadoop:hadoop /usr/local/hadoop
chu888chu888@ubuntu1:~$ sudo nano ~/.bashrc
#加入以下内容

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
```

```
export PATH=$PATH:$HADOOP_INSTALL/bin  
export PATH=$PATH:$JAVA_HOME/bin  
export PATH=$PATH:$HADOOP_INSTALL/sbin  
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL  
export HADOOP_COMMON_HOME=$HADOOP_INSTALL  
export HADOOP_HDFS_HOME=$HADOOP_INSTALL  
export YARN_HOME=$HADOOP_INSTALL  
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native  
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"  
#HADOOP VARIABLES END
```

```
chu888chu888@ubuntu1:~$ source ~/.bashrc  
chu888chu888@ubuntu1:/usr/local/hadoop$ sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh  
chu888chu888@ubuntu1:~$ cd /usr/local/hadoop/
```

```
#hadoop安装后的查看hadoop的版本  
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/hadoop version  
Hadoop 2.6.0  
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r  
e3496499ecb8d220fba99dc5ed4c99c8f9e33bb1  
Compiled by jenkins on 2014-11-13T21:10Z  
Compiled with protoc 2.5.0  
From source with checksum 18e43357c8f927c0695f1e9522859d6a  
This command was run using /usr/local/hadoop/share/hadoop/common  
/hadoop-common-2.6.0.jar  
hadoop@ubuntu2:/usr/local/hadoop$
```

### #实验一 Hadoop单机配置

Hadoop 默认模式为非分布式模式，无需进行其他配置即可运行。非分布式即单 Java 进程，方便进行调试。

现在我们可以执行例子来感受下 Hadoop 的运行。Hadoop 附带了丰富的例子（运行 `./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar` 可以看到所有例子），包括 `wordcount`、`terasort`、`join`、`grep` 等。

在此我们选择运行 `grep` 例子，我们将 `input` 文件夹中的所有文件作为输入，筛选当中符合正则表达式 `dfs[a-z.]+` 的单词并统计出现的次数，最后输出结果到 `output` 文件夹中。

```
chu888chu888@ubuntu1:/usr/local/hadoop$ sudo mkdir input
chu888chu888@ubuntu1:/usr/local/hadoop$ sudo cp README.txt input
chu888chu888@ubuntu1:/usr/local/hadoop$ bin/hadoop jar share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.6.0-sources.jar org.apache.hadoop.examples.WordCount input output
#这里面有一个小插曲，其实就是权限的事件，如果出现问题的话，就是这个程序无法在当前目录创建，简单粗暴一点chmod -R 777 /usr/local/hadoop
#如果成功的话，那前目录就会有一个output目录自动生成
#目录内容如下
chu888chu888@ubuntu1:/usr/local/hadoop/output$ ls
part-r-00000 _SUCCESS

#查看输出结果
hadoop@ubuntu2:/usr/local/hadoop$ ls
bin etc include input lib libexec LICENSE.txt NOTICE.txt
output README.txt sbin share
hadoop@ubuntu2:/usr/local/hadoop$ cd output/
hadoop@ubuntu2:/usr/local/hadoop/output$ ls
part-r-00000 _SUCCESS
hadoop@ubuntu2:/usr/local/hadoop/output$ cd ..
hadoop@ubuntu2:/usr/local/hadoop$ cat ./output/*
(BIS), 1
(ECCN) 1
(TSU) 1
(see 1
5D002.C.1, 1
740.13) 1
<http://www.wassenaar.org/> 1

#注意，Hadoop 默认不会覆盖结果文件，因此再次运行上面实例会提示出错，需要先将 ./output 删除。
```

## 五 开始伪分布式的配置文件修改

Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点既作为 NameNode 也作为 DataNode，同时，读取的是 HDFS 中的文件。

Hadoop 的配置文件位于 /usr/local/hadoop/etc/hadoop/ 中，伪分布式需要修改2个配置文件 core-site.xml 和 hdfs-site.xml 。Hadoop的配置文件是 xml 格式，每个配置以声明 property 的 name 和 value 的方式来实现。

修改配置文件 core-site.xml

```
chu888chu888@ubuntu1:/$ sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
```

#内容如下

```
<configuration>
<property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
</property>
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

```
chu888chu888@ubuntu1:/$ sudo nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

#内容如下

```
<configuration>
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
</property>
```

```
</configuration>

hadoop@ubuntu2:/usr/local/hadoop$ ./bin/hdfs namenode -format
16/01/13 21:26:01 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = ubuntu2/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.6.0
16/01/13 21:26:02 INFO namenode.NNStorageRetentionManager: Going
to retain 1 images with txid >= 0
16/01/13 21:26:02 INFO util.ExitUtil: Exiting with status 0
16/01/13 21:26:02 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at ubuntu2/127.0.1.1
*****/
```

成功的话，会看到“successfully formatted”和“Exitting with status 0”的提示，若为“Exitting with status 1”则是出错。

### >>>注意

在这一步时若提示 Error: JAVA\_HOME is not set and could not be found . 的错误，则需要在文件 ./etc/hadoop/hadoop-env.sh 中设置 JAVA\_HOME 变量，即在该文件中找到：

```
export JAVA_HOME=${JAVA_HOME}
```

将这一行改为 JAVA 安装位置：

```
export JAVA_HOME=/usr/lib/jvm/
```

再重新尝试格式化即可。

#接着开启 NaneNode 和 DataNode 守护进程。

```
hadoop@ubuntu2:/usr/local/hadoop$ ./sbin/start-dfs.sh
16/01/13 21:29:20 WARN util.NativeCodeLoader: Unable to load nat
ive-hadoop library for your platform... using builtin-java class
es where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/
hadoop-hadoop-namenode-ubuntu2.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/
hadoop-hadoop-datanode-ubuntu2.out
Starting secondary namenodes [0.0.0.0]
```

```
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.  
ECDSA key fingerprint is 87:f6:48:6b:0f:52:1f:27:3f:62:8c:c0:39:  
2d:87:e3.  
Are you sure you want to continue connecting (yes/no)? yes  
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the lis-  
t of known hosts.  
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoo-  
p/logs/hadoop-hadoop-secondarynamenode-ubuntu2.out  
16/01/13 21:29:39 WARN util.NativeCodeLoader: Unable to load nat-  
ive-hadoop library for your platform... using builtin-java class-  
es where applicable
```

启动时可能会出现如下 WARN 提示：WARN util.NativeCodeLoader: Unable t o load native-hadoop library for your platform... using builtin-ja va classes where applicable WARN 提示可以忽略，并不会影响正常使用。

启动完成后，可以通过命令 jps 来判断是否成功启动，若成功启动则会列出如下进程：“NameNode”、“DataNode” 和 “SecondaryNameNode”（如果 SecondaryName Node 没有启动，请运行 sbin/stop-dfs.sh 关闭进程，然后再次尝试启动尝试）。如果没有 NameNode 或 DataNode ，那就是配置不成功，请仔细检查之前步骤，或通过查看启动日志排查原因。

```
hadoop@ubuntu2:/usr/local/hadoop$ jps  
11841 NameNode  
12309 Jps  
12188 SecondaryNameNode  
11998 DataNode  
hadoop@ubuntu2:/usr/local/hadoop$
```

通过查看启动日志分析启动失败原因

有时 Hadoop 无法正确启动，如 NameNode 进程没有顺利启动，这时可以查看启动日志来排查原因，注意几点：

启动时会提示形如 “DBLab-XMU: starting namenode, logging to /usr/loc al/hadoop/logs/hadoop-namenode-DBLab-XMU.out”，其中 DBLab-XMU 对应你的机器名，但其实启动日志信息是记录在 /usr/local/hadoop/logs/h

adoop-hadoop-namenode-DBLab-XMU.log 中，所以应该查看这个后缀为 .log 的文件；

每一次的启动日志都是追加在日志文件之后，所以得拉到最后面看，看下记录的时间就知道了。

一般出错的提示在最后面，通常是写着 Fatal、Error 或者 Java Exception 的地方。

可以在网上搜索一下出错信息，看能否找到一些相关的解决方法。

成功启动后，可以访问 Web 界面 <http://localhost:50070> 查看 NameNode 和 Datanode 信息，还可以在线查看 HDFS 中的文件。

The screenshot shows the Hadoop Web UI Overview page for the active cluster 'localhost:9000'. The top navigation bar includes links for Hadoop, Overview, Datanodes, Snapshot, Startup Progress, and Utilities.

**Overview 'localhost:9000' (active)**

Started:	Fri Jan 08 10:58:35 CST 2016
Version:	2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled:	2014-11-13T21:10Z by jenkins from (detached from e349649)
Cluster ID:	CID-a369a38e-40e9-454a-b9c2-af8fb3d5d18
Block Pool ID:	BP-1433882767-127.0.1.1-145221324432

**Summary**

Security is off.  
Safemode is off.  
16 files and directories, 0 blocks = 16 total filesystem object(s).  
Heap Memory used 59.59 MB of 149.5 MB Heap Memory. Max Heap Memory is 889 MB.  
Non Heap Memory used 44.79 MB of 45.78 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	0 B
----------------------	-----

## Browse Directory

The screenshot shows the Hadoop Web UI Browse Directory page for the path '/tmp/hadoop-yarn/staging/history'. The top navigation bar includes links for Hadoop, Overview, Datanodes, Snapshot, Startup Progress, and Utilities.

**/tmp/hadoop-yarn/staging/history**

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxrwx---	chu888chu888	supergroup	0 B	0	0 B	done
drwxrwxrwt	chu888chu888	supergroup	0 B	0	0 B	done_intermediate

## 运行伪分布式实例

1 上面的单机模式，grep 例子读取的是本地数据，伪分布式读取的则是 HDFS 上的数据。要使用 HDFS，首先需要在 HDFS 中创建用户目录：

```
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/dfs dfs -mkdir -p /user/hadoop
16/01/13 21:37:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@ubuntu2:/usr/local/hadoop$
```

2 接着将 ./etc/hadoop 中的 xml 文件作为输入文件复制到分布式文件系统中，即将 /usr/local/hadoop/etc/hadoop 复制到分布式文件系统中的 /user/hadoop/input 中。我们使用的是 hadoop 用户，并且已创建相应的用户目录 /user/hadoop ，因此在命令中就可以使用相对路径如 input，其对应的绝对路径就是 /user/hadoop/input:

```
#!/bin/dfs dfs -mkdir input
#!/bin/dfs dfs -put ./etc/hadoop/*.xml input
```

3 复制完成后，可以通过如下命令查看文件列表：

```
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/dfs dfs -ls input
16/01/13 21:41:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 8 items
-rw-r--r-- 1 hadoop supergroup          4436 2016-01-13 21:40 inp
ut/capacity-scheduler.xml
-rw-r--r-- 1 hadoop supergroup          1071 2016-01-13 21:40 inp
ut/core-site.xml
-rw-r--r-- 1 hadoop supergroup         9683 2016-01-13 21:40 inp
ut/hadoop-policy.xml
-rw-r--r-- 1 hadoop supergroup         1133 2016-01-13 21:40 inp
ut/dfs-site.xml
-rw-r--r-- 1 hadoop supergroup          620  2016-01-13 21:40 inp
ut/httpfs-site.xml
-rw-r--r-- 1 hadoop supergroup         3523 2016-01-13 21:40 inp
ut/kms-acls.xml
-rw-r--r-- 1 hadoop supergroup         5511 2016-01-13 21:40 inp
ut/kms-site.xml
-rw-r--r-- 1 hadoop supergroup          690  2016-01-13 21:40 inp
ut/yarn-site.xml
hadoop@ubuntu2:/usr/local/hadoop$
```

4 伪分布式运行 MapReduce 作业的方式跟单机模式相同，区别在于伪分布式读取的是HDFS中的文件（可以将单机步骤中创建的本地 input 文件夹，输出结果 output 文件夹都删掉来验证这一点）。

```
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar grep input output 'dfs[a-z.]+'
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/dfs dfs -cat output/*
hadoop@ubuntu2:/usr/local/hadoop$ ./bin/dfs dfs -cat output/*
16/01/13 21:42:55 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
1      dfsadmin
1      dfs.replication
1      dfs.namenode.name.dir
1      dfs.datanode.data.dir
```

### 5 将结果取回本地

```
# rm -r ./output
# 先删除本地的 output 文件夹（如果存在）
# ./bin/dfs dfs -get output ./output
# 将 HDFS 上的 output 文件夹拷贝到本机
# cat ./output/*
```

### 6 关闭hadoop

```
./sbin/stop-dfs.sh
```

### 7 开启hadoop

下次启动 hadoop 时，无需进行 NameNode 的初始化，只需要运行 ./sbin/start-dfs.sh 就可以！

## 六 启动YARN

### 1 （伪分布式不启动 YARN 也可以，一般不会影响程序执行）

有的读者可能会疑惑，怎么启动 Hadoop 后，见不到书上所说的 JobTracker 和 TaskTracker，这是因为新版的 Hadoop 使用了新的 MapReduce 框架（MapReduce V2，也称为 YARN，Yet Another Resource Negotiator）。

YARN 是从 MapReduce 中分离出来的，负责资源管理与任务调度。YARN 运行于 MapReduce 之上，提供了高可用性、高扩展性，YARN 的更多介绍在此不展开，有兴趣的可查阅相关资料。

上述通过 `./sbin/start-dfs.sh` 启动 Hadoop，仅仅是启动了 MapReduce 环境，我们可以启动 YARN，让 YARN 来负责资源管理与任务调度。

首先修改配置文件 `mapred-site.xml`，这边需要先进行重命名：

```
hadoop@ubuntu2:/usr/local/hadoop$ mv ./etc/hadoop/mapred-site.xml.template ./etc/hadoop/mapred-site.xml
hadoop@ubuntu2:/usr/local/hadoop$ nano ./etc/hadoop/mapred-site.xml
hadoop@ubuntu2:/usr/local/hadoop$

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

2 修改接着修改配置文件 `yarn-site.xml`：

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

3 启动YARN

```
hadoop@ubuntu2:/usr/local/hadoop$ ./sbin/start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn
-hadoop-resourcemanager-ubuntu2.out
localhost: starting nodemanager, logging to /usr/local/hadoop/lo
gs/yarn-hadoop-nodemanager-ubuntu2.out
hadoop@ubuntu2:/usr/local/hadoop$ ./sbin/mr-jobhistory-daemon.sh
start historyserver
starting historyserver, logging to /usr/local/hadoop/logs/mapred
-hadoop-historyserver-ubuntu2.out
hadoop@ubuntu2:/usr/local/hadoop$
```

4 查看进程 开启后通过 jps 查看，可以看到多了 NodeManager 和 ResourceManager 两个后台进程，如下图所示。启动 YARN 之后，运行实例的方法还是一样的，仅仅是资源管理方式、任务调度不同。观察日志信息可以发现，不启用 YARN 时，是“mapred.LocalJobRunner”在跑任务，启用 YARN 之后，是“mapred.YARNRunner”在跑任务。启动 YARN 有个好处是可以通过 Web 界面查看任务的运行情况：<http://localhost:8088/cluster>，如下图所示。

```
hadoop@ubuntu2:/usr/local/hadoop$ jps
12880 ResourceManager
11841 NameNode
13329 JobHistoryServer
13398 Jps
13016 NodeManager
12188 SecondaryNameNode
11998 DataNode
hadoop@ubuntu2:/usr/local/hadoop$
```

The screenshot shows the Hadoop YARN ResourceManager UI at the URL 192.168.1.192:8088/cluster. The main title is "All Applications". On the left, there's a sidebar with "Cluster Metrics" and a table of application logs. The table has columns for ID, User, Name, Application Type, Queue, StartTime, FinishTime, State, FinalStatus, Progress, and Tracking UI. A search bar and pagination controls are also present.

但 YARN 主要是为集群提供更好的资源管理与任务调度，然而这在单机上体现不出价值，反而会使程序跑得稍慢些。因此在单机上是否开启 YARN 就看实际情况了。

不启动 YARN 需重命名 mapred-site.xml 如果不想启动 YARN，务必把配置文件 mapred-site.xml 重命名，改成 mapred-site.xml.template，需要用时改回来就行。否则在该配置文件存在，而未开启 YARN 的情况下，运行程序会提示“Retrying connect to server: 0.0.0.0/0.0.0.0:8032”的错误，这也是为何该配置文件初始文件名为 mapred-site.xml.template。

### 5 关闭YARN

```
#!/sbin/stop-yarn.sh  
#!/sbin/mr-jobhistory-daemon.sh stop historyserver
```

## Unable to load native-hadoop library for your platform 错误

错误现象

```
hadoop@Master:~$ hadoop fs -ls input
16/01/27 18:52:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 9 items
-rw-r--r-- 1 hadoop supergroup          4436 2016-01-25 22:10 inp
ut/capacity-scheduler.xml
-rw-r--r-- 1 hadoop supergroup          1072 2016-01-25 22:10 inp
ut/core-site.xml
-rw-r--r-- 1 hadoop supergroup         9683 2016-01-25 22:10 inp
ut/hadoop-policy.xml
-rw-r--r-- 1 hadoop supergroup         1257 2016-01-25 22:10 inp
ut/hdfs-site.xml
-rw-r--r-- 1 hadoop supergroup          620  2016-01-25 22:10 inp
ut/httpfs-site.xml
-rw-r--r-- 1 hadoop supergroup         3523 2016-01-25 22:10 inp
ut/kms-acls.xml
-rw-r--r-- 1 hadoop supergroup         5511 2016-01-25 22:10 inp
ut/kms-site.xml
-rw-r--r-- 1 hadoop supergroup         1103 2016-01-25 22:10 inp
ut/mapred-site.xml
-rw-r--r-- 1 hadoop supergroup          924  2016-01-25 22:10 inp
ut/yarn-site.xml
```

### 解决

```
1首先编辑core-site.xml
<property>
    <name>hadoop.native.lib</name>
    <value>true</value>
    <description>Should native hadoop libraries, if present, be used</description>
</property>
2之后修改环境变量
export JAVA_LIBRARY_PATH=/usr/local/hadoop/lib/native
```



## 第四章 完全分布式搭建

### 1网络拓扑

- 192.168.1.80 Master
- 192.168.1.82 Slave1
- 192.168.1.84 Slave2

### 2安装JDK

所有实验主机都需要正确的安装JDK,具体操作方法

```
chu888chu888@ubuntu1:~$ tar xvfz jdk-8u65-linux-x64.gz
chu888chu888@ubuntu1:~$ sudo cp -r jdk1.8.0_65/ /usr/lib/jvm/
chu888chu888@ubuntu1:/usr/lib/jvm$ sudo nano /etc/profile

#修改内容如下,注意大小写
#在环境变量中的配置中,有一点需要指出就是如果只是编辑~/.profile的话这个变量
的生效只是针对当前用户的.
#如果想要其在全局生效的话,建议更新/etc/profile,这是一个全局的.

export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

chu888chu888@ubuntu1:/usr/lib/jvm$ source /etc/profile
chu888chu888@ubuntu1:/usr/lib/jvm$ env
chu888chu888@ubuntu1:/usr/lib/jvm$ java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

#有一种极端情况就是,如果在本机已经安装了OpenJavaSDK,怎么办?
sudo update-alternatives --install /usr/bin/java java /usr/lib/j
vm/java/bin/java 300
sudo update-alternatives --install /usr/bin/javac javac /usr/lib
/jvm/java/bin/javac 300
sudo update-alternatives --install /usr/bin/jar jar /usr/lib/jvm
/java/bin/jar 300
sudo update-alternatives --install /usr/bin/javah javah /usr/lib
/jvm/java/bin/javah 300
sudo update-alternatives --install /usr/bin/javap javap /usr/lib
/jvm/java/bin/javap 300
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

这里面我简单补充一下,其他相关知识,因为涉及到主机之间的安装文件传递,我们可以使用sftp命令进行.

```
chu888chu888@ubuntu-hadoop:~$ sftp chu888chu888@192.168.1.84
The authenticity of host '192.168.1.84 (192.168.1.84)' can't be
established.
ECDSA key fingerprint is 6c:00:fb:9b:43:6c:3b:29:96:98:a8:28:d1:
23:11:13.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.84' (ECDSA) to the list of
known hosts.
chu888chu888@192.168.1.84's password:
Connected to 192.168.1.84.
sftp> put jdk-8u65-linux-x64.gz
Uploading jdk-8u65-linux-x64.gz to /home/chu888chu888/jdk-8u65-1
inux-x64.gz
jdk-8u65-linux-x64.gz
                                         100%  173MB  28.8MB/s   00
:06
sftp>
```

## 3Hadoop用户的创建

```
创建hadoop用户组
创建hadoop用户
给hadoop用户添加权限,打开/etc/sudoers文件
chu888chu888@ubuntu1:/$ sudo addgroup hadoop
chu888chu888@ubuntu1:/$ sudo adduser -ingroup hadoop hadoop chu8
88chu888@ubuntu1:/$ sudo nano /etc/sudoers
# User privilege specification
root    ALL=(ALL:ALL) ALL
hadoop  ALL=(ALL:ALL) ALL
```

## 4hosts文件修改

所有的主机的**hosts**都需要修改,在这里我吃了一个大亏,如果在**etc**配置文件中直接用**Ip**的话,可能会出现**Datanode**链接不上**Namenode**的现象.

```
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ more /etc/hosts
127.0.0.1      localhost
192.168.1.80    Master
192.168.1.82    Slave1

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$
```

## 5SSH无密码登录

所有的主机都要进行操作

这个操作是要让Master节点可以在无密码的状态下SSH登录到各个Slave节点上  
首先生成Master节点的公钥，在Master节点的终端中执行

```
#如果没有该目录,先执行一次ssh localhost
$ cd ~/.ssh
#删除之前生成的公钥
$ rm ./id_rsa*
#一直按回车就可以了
$ ssh-keygen -t rsa
```

让Master节点需能无密码的SSH本机，在Master节点上执行

```
$cat ./id_rsa.pub>>./authorized_keys
完成后可执行ssh Master验证一下，接着需要把Master节点的公钥上传输到Slave1
节点上
$ssftp hadoop@Slave1
```

接着在Slave1节点上，将ssh公钥加入授权

```
$mkdir ~/.ssh
$cat ~/id_rsa.pub>>~/.ssh/authorized_keys
$rm ~/id_rsa.pub
如果有其他的Slave节点，也要执行将Master公钥传输到Slave节点，在Slave节点加
入授权这两步。
这样，在Master节点就可以无密码SSH到各个Slave节点了。
```

## 6 Hadoop的安装

```
chu888chu888@ubuntu1:~$ sudo tar xvfz hadoop-2.6.0.tar.gz
chu888chu888@ubuntu1:~$ sudo cp -r hadoop-2.6.0 /usr/local/hadoo
p
chu888chu888@ubuntu1:~$ sudo chmod -R 775 /usr/local/hadoop/
chu888chu888@ubuntu1:~$ sudo chown -R hadoop:hadoop /usr/local/h
adoop
```

这里面有一个小的体验技巧，我建议将所有需要的环境变量配置加入到/etc/profile中，  
这是全局变量。

```

export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

```

还有一个问题就是，在启动hadoop的时候经常会出现，找不到JAVA\_HOME的问题，这个问题可以通过修改hadoop环境变量来解决，直接写死变量就可以了。

```

$ more hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/

```

## 6配置集群环境192.168.1.80 NameNode

集群/分布式模式需要修改 /usr/local/hadoop/etc/hadoop 中的5个配置文件，更多设置项可点击查看官方说明，这里仅设置了正常启动所必须的设置项： slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml 。

## 7文件slaves

文件 slaves，将作为 DataNode 的主机名写入该文件，每行一个，默认为 localhost，所以在伪分布式配置时，节点即作为 NameNode 也作为 DataNode。分布式配置可以保留 localhost，也可以删掉，让 Master 节点仅作为 NameNode 使

用。

本教程让 Master 节点仅作为 NameNode 使用，因此将文件中原来的 localhost 删除，只添加一行内容：Slave1。

```

hadoop@ubuntu-hadoop:~$ cd /usr/local/hadoop/etc/hadoop/
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ sudo nano slaves
[sudo] password for hadoop:
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ more slaves
Slave1

hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$
```

**8**文件 **core-site.xml** 改为下面的配置：

```

<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://Master:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
</configuration>
```

**9**文件 **hdfs-site.xml**，**dfs.replication** 一般设为 **3**，但我们只有一个 **Slave** 节点，所以 **dfs.replication** 的值还是设为 **1**：

```

<configuration>
  <property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>Master:50090</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
  </property>
</configuration>

```

**10**文件 **mapred-site.xml**（可能需要先重命名，默认文件名为 **mapred-site.xml.template**），然后配置修改如下：

```

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>Master:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>Master:19888</value>
  </property>
</configuration>

```

## 11文件 yarn-site.xml :

```
<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>Master</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

## 在其他**Slave**节点需要做的

首先通过sftp把Master配置好的hadoop打包,之后传输到Slave节点上,配置好环境变量JDK PATH SSH 基本上与Master是一样的.

配置好后,将 Master 上的 /usr/local/Hadoop 文件夹复制到各个节点上。因为之前有跑过伪分布式模式,建议在切换到集群模式前先删除之前的临时文件。

在 Master 节点上执行：

```
cd /usr/local
# 删除 Hadoop 临时文件
sudo rm -r ./hadoop/tmp
# 删除日志文件
sudo rm -r ./hadoop/logs/*
# 先压缩再复制
tar -cvfz ~/hadoop.master.tar.gz ./hadoop
```

在Slave节点上执行:

```
sudo rm -r /usr/local/hadoop      # 删掉旧的(如果存在)
sudo tar -xvfz ~/hadoop.master.tar.gz -C /usr/local
sudo chown -R hadoop:hadoop /usr/local/hadoop
```

开始启动集群

```
hdfs namenode -format      # 首次运行需要执行初始化，之后不需要  
在Master上执行：  
$start-dfs.sh  
$start-yarn.sh  
$mr-jobhistory-daemon.sh start historyserver  
  
Centos6.X需要关闭防火墙  
sudo service iptables stop    # 关闭防火墙服务  
sudo chkconfig iptables off   # 禁止防火墙开机自启，就不用手动关闭了  
Cent7  
systemctl stop firewalld.service    # 关闭firewall  
systemctl disable firewalld.service # 禁止firewall开机启动
```

之后分别在Master与Slave上执行jps,会看到不同的结果.缺少任一进程都表示出错。另外还需要在 Master 节点上通过命令 hdfs dfsadmin -report 查看 DataNode 是否正常启动，如果 Live datanodes 不为 0，则说明集群启动成功。例如我这边一共有 1 个 Datanodes：

```
$jps  
$hdfs dfsadmin -report
```

可以访问<http://192.168.1.80:50070/> 查看结果

```
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ jps  
11056 Jps  
10561 ResourceManager  
10420 SecondaryNameNode  
10839 JobHistoryServer  
10207 NameNode  
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ █
```

```
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ jps  
4290 Jps  
4164 NodeManager  
4026 DataNode  
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ █
```

## 执行分布式的实验-分布式存储

执行分布式实例过程与伪分布式模式一样，首先创建 HDFS 上的用户目录：

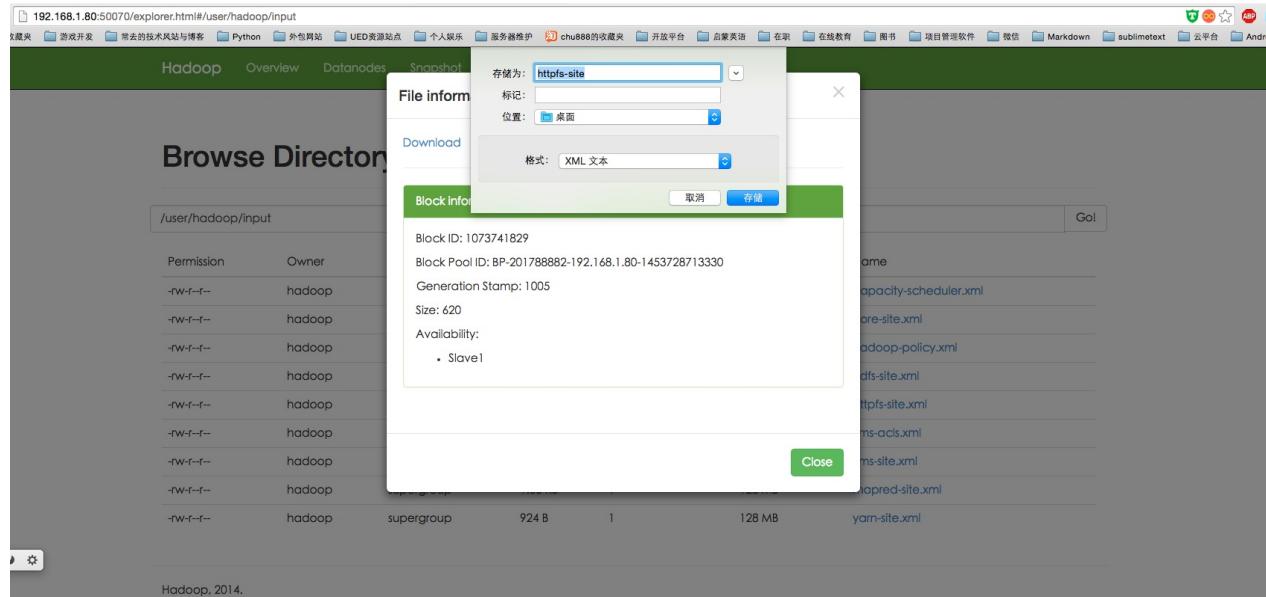
```
$ hdfs dfs -mkdir -p /user/hadoop
```

将 /usr/local/hadoop/etc/hadoop 中的配置文件作为输入文件复制到分布式文件系统中：

```
$ hdfs dfs -mkdir input
```

```
$ hdfs dfs -put /usr/local/hadoop/etc/hadoop/*.xml input
```

通过查看 DataNode 的状态（占用大小有改变），输入文件确实复制到了 DataNode 中，如下图所示：



## 执行分布式的实验-MapReduce

执行MapReduce作业

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar grep /user/hadoop/input /user/hadoop/output  
'dfs[a-z.]+'
```

查看 <http://192.168.1.80:8088/cluster> 看结果

运行时的输出信息与伪分布式类似，会显示 Job 的进度。

可能会有点慢，但如果迟迟没有进度，比如 5 分钟都没看到进度，那不妨重启 Hadoop 再试试。若重启还不行，则很有可能是内存不足引起，建议增大虚拟机的内存，或者通过更改 YARN 的内存配置解决。

## 2.Hadoop 2.6.2完全分布式

同样可以通过 Web 界面查看任务进度 <http://master:8088/cluster>，在 Web 界面点击“Tracking UI”这一列的 History 连接，可以看到任务的运行信息，如下图所示：

The screenshot shows the 'FINISHED Applications' section of the Hadoop Web interface. On the left, there's a sidebar with cluster metrics and a 'Scheduler' section. The main area displays a table of finished applications. One row is visible:

ID	User	Name	ApplicationType	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1453730825595_0001	hadoop	grep-search	MAPREDUCE	default	Mon, 25 Jan 2016 14:12:45 GMT	Mon, 25 Jan 2016 14:13:42 GMT	FINISHED	SUCCEEDED		<a href="#">History</a>

Below the table, it says "Showing 1 to 1 of 1 entries".

```
关闭集群
stop-yarn.sh
stop-dfs.sh
mr-jobhistory-daemon.sh stop historyserver
```

# Hadoop 2.7.2完全分布式的搭建

## 一 生产环境描述

### 1 网络环境

- hadoopmaster 192.168.1.159
- hadoopslave1 192.168.1.76
- hadoopslave2 192.168.1.166

### 2 软件环境

- JDK 7U79
- Hadoop 2.7.2

## 二 hadoopmaster主机配置

### 1 JDK设置

```
chu888chu888@hadoopmaster:~$ ls  
jdk-7u79-linux-x64.gz  
chu888chu888@hadoopmaster:~$ sudo tar xvfz jdk-7u79-linux-x64.gz  
  
chu888chu888@hadoopmaster:~$ ls  
jdk1.7.0_79 jdk-7u79-linux-x64.gz  
chu888chu888@hadoopmaster:~$ sudo cp -r jdk1.7.0_79/ /usr/lib/jvm/  
chu888chu888@hadoopmaster:~$ cd /usr/lib/jvm  
chu888chu888@hadoopmaster:/usr/lib/jvm$ ls  
bin db jre LICENSE README.html src.zip THIRDPARTYLICENSEREADME.t  
xt  
COPYRIGHT include lib man release THIRDPARTYLICENSEREADME-JAVAFX  
.txt  
chu888chu888@hadoopmaster:/usr/lib/jvm$  
  
chu888chu888@ubuntu1:/usr/lib/jvm$ sudo nano /etc/profile
```

修改内容如下,注意大小写,在环境变量中的配置中,有一点需要指出就是如果只是编辑~/.profile的话这个变量的生效只是针对当前用户的.如果想要其在全局生效的话,建议更新/etc/profile,这是一个全局的.

/etc/profile的内容如下:

```
export JAVA_HOME=/usr/lib/jvm/  
export JRE_HOME=${JAVA_HOME}/jre  
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib  
export PATH=${JAVA_HOME}/bin:$PATH
```

这里面有一个小的体验技巧,我建议将所有需要的环境变量配置加入到/etc/profile中,这是全局变量.

```

export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"

```

还有一个问题就是,在启动hadoop的时候经常会出现,找不到JAVA\_HOME的问题,这个问题可以通过修改hadoop环境变量来解决,直接写死变量就可以了.

测试环境变量是不是生效

```

chu888chu888@hadoopmaster:/usr/lib/jvm$ source /etc/profile
chu888chu888@hadoopmaster:/usr/lib/jvm$ env
XDG_SESSION_ID=1
TERM=xterm-256color
SHELL=/bin/bash
SSH_CLIENT=192.168.1.23 49818 22
OLDPWD=/home/chu888chu888
SSH_TTY=/dev/pts/0
JRE_HOME=/usr/lib/jvm//jre
USER=chu888chu888
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;3
5:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41
:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.a
rj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=
01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz
=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;3

```

```

1:* .tz=01;31:* .deb=01;31:* .rpm=01;31:* .jar=01;31:* .war=01;31:* .e
ar=01;31:* .sar=01;31:* .rar=01;31:* .ace=01;31:* .zoo=01;31:* .cpio=
01;31:* .7z=01;31:* .rz=01;31:* .jpg=01;35:* .jpeg=01;35:* .gif=01;35
:* .bmp=01;35:* .pbm=01;35:* .pgm=01;35:* .ppm=01;35:* .tga=01;35:* .x
bm=01;35:* .xpm=01;35:* .tif=01;35:* .tiff=01;35:* .png=01;35:* .svg=
01;35:* .svgz=01;35:* .mng=01;35:* .pcx=01;35:* .mov=01;35:* .mpg=01;
35:* .mpeg=01;35:* .m2v=01;35:* .mkv=01;35:* .webm=01;35:* .ogm=01;35
:* .mp4=01;35:* .m4v=01;35:* .mp4v=01;35:* .vob=01;35:* .qt=01;35:* .n
uv=01;35:* .wmv=01;35:* .ASF=01;35:* .rm=01;35:* .rmvb=01;35:* .flc=0
1;35:* .avi=01;35:* .fli=01;35:* .flv=01;35:* .gl=01;35:* .dl=01;35:*
.xcf=01;35:* .xwd=01;35:* .yuv=01;35:* .cgm=01;35:* .emf=01;35:* .axv
=01;35:* .anx=01;35:* .ogv=01;35:* .ogx=01;35:* .aac=00;36:* .au=00;3
6:* .flac=00;36:* .mid=00;36:* .midi=00;36:* .mka=00;36:* .mp3=00;36:
* .mpc=00;36:* .ogg=00;36:* .ra=00;36:* .wav=00;36:* .axa=00;36:* .oga
=00;36:* .spx=00;36:* .xspf=00;36:
MAIL=/var/mail/chu888chu888
PATH=/usr/lib/jvm//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
PWD=/usr/lib/jvm
JAVA_HOME=/usr/lib/jvm/
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/chu888chu888
LANGUAGE=en_US:en
LOGNAME=chu888chu888
CLASSPATH=.: /usr/lib/jvm//lib:/usr/lib/jvm//jre/lib
SSH_CONNECTION=192.168.1.23 49818 192.168.1.159 22
LESSOPEN=| /usr/bin/lesspipe %
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %
_= /usr/bin/env
chu888chu888@hadoopmaster:/usr/lib/jvm$ java -version
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
chu888chu888@hadoopmaster:/usr/lib/jvm$
```

## 2 IP设置

```
chu888chu888@hadoopmaster:/etc/network$ ls
if-down.d  if-post-down.d  if-pre-up.d  if-up.d  interfaces  int
erfaces.d  run
chu888chu888@hadoopmaster:/etc/network$ sudo nano interfaces
```

**interfaces** 的内容如下

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.1.159
netmask 255.255.255.0
gateway 192.168.1.1
```

**/etc/resolv.conf** 的内容

```
nameserver 192.168.1.1
```

## 3 Hadoop 相关环境变量的设置

创建hadoop用户组

创建hadoop用户

```
chu888chu888@hadoopmaster:~$ sudo addgroup hadoop  
[sudo] password for chu888chu888:  
Adding group `hadoop' (GID 1001) ...  
Done.  
chu888chu888@hadoopmaster:~$ sudo adduser -ingroup hadoop hadoop  
Adding user `hadoop' ...  
Adding new user `hadoop' (1001) with group `hadoop' ...  
Creating home directory `/home/hadoop' ...  
Copying files from `/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for hadoop  
Enter the new value, or press ENTER for the default  
    Full Name []:  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n] y
```

给hadoop用户添加权限,打开/etc/sudoers文件

```
root      ALL=(ALL:ALL)  ALL  
hadoop   ALL=(ALL:ALL)  ALL
```

## 4 hosts文件修改

所有的主机的**hosts**都需要修改,在这里我吃了一个大亏,如果在**etc**配置文件中直接用**Ip**的话,可能会出现**Datanode**链接不上**Namenode**的现象.

```
127.0.0.1      localhost  
192.168.1.159  hadoopmaster  
192.168.1.76   hadoopslave1  
192.168.1.166  hadoopslave2
```

## 5 SSH无密码登录

步骤一用**ssh-key-gen**在**hadoopmaster**主机上创建公钥与密钥

需要注意一下，一定要用hadoop用户生成公钥，因为我们是免密钥登录用的是hadoop

```

hadoop@hadoopmaster:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
cf:98:06:01:29:81:ff:82:5b:d3:6d:5d:53:b6:a7:75 hadoop@hadoopmas
ter
The key's randomart image is:
+--[ RSA 2048]----+
| .. . .
| . . . o |
| . . . o . |
| . . . o . o E|
| . o ... S. . + . |
| . + o o..= . |
| o o . + o |
| . . . |
| |
+-----+
```

步骤二保证**hadoopmaster**登录自己是有效的

```

cd .ssh
cat ./id_rsa.pub >> ./authorized_keys
```

步骤三将公钥拷贝到其他主机上

```
scp ~/.ssh/id_rsa.pub hadoop@hadoopslave1:/home/hadoop/  
scp ~/.ssh/id_rsa.pub hadoop@hadoopslave2:/home/hadoop/
```

#### 步骤四 在其他二个节点上做的工作

```
mkdir ~/.ssh          # 如果不存在该文件夹需先创建，若已存在则忽略  
cat ~/id_rsa.pub >> ~/.ssh/authorized_keys  
rm ~/id_rsa.pub      # 用完就可以删掉了
```

## 6 资源限制配置

此步骤要求在hadoopmaster hadoopslave1 hadoopslave2都要进行配置,以防止处理达到机器文件上限

### 1通过修改/etc/security/limits.conf追击参数进行配置

HBASE和其他的数据库软件一样会同时打开很多文件,Linux默认的ulimit值是1024,这对HBASE来说太小了,当使用诸如bulkload这种工具导入数据的时候会得到这样的异常信息:java.io.IOException:Too many open files.我们需要改变这个值.这是对操作系统的操作,而不是通过HBASE配置文件完成的,我们可以大致估算ulimit值需要配置为多大,例如:每个列族至少有一个存储文件(HFile),每个被加载的Region可能管理多达5或6个列族所对应的存储文件,用于存储文件个数乘于列族数再乘以每个RegionServer中的Region数量得到RegionServer主机管理的存储文件数量.假如每个Region有3个列族,每个列族平均有3个存储文件,每个RegionServer有100个region,将至少需要 $3 \times 100 = 900$ 个文件.这些存储文件会被客户端大量的操作,涉及大量的磁盘操作.

```

#*           soft   core      0
#root        hard   core    100000
#*           hard   rss     10000
#@student    hard   nproc    20
#@faculty    soft   nproc    20
#@faculty    hard   nproc    50
#ftp         hard   nproc    0
#ftp         -      chroot  /ftp
#@student    -      maxlogins 4
hadoop -    nofile 65535
hadoop -    nproc  32000

```

**2** 通过修改 **hdfs-site.xml**解决同时处理文件上限的参数 Hadoop的Datanode有一个用于设置同时处理文件的上限个数的参数,这个参数叫**xcievers**,在启动之前,先确认有没有配置hadoop的这个参数,默认值是256,这对于一个任务很多的集群来说,实在太小了.

```

<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>12500</value>
</property>

```

### 三 hadoopslave1 2主机配置

#### 1 JDK安装

同上

#### 2 IP设置

同上

#### 3 Hadoop相关环境变量的设置

同上

## 4 hosts文件修改

同上

## 5 SSH无密码登录

同上

## 四 hadoopmaster的安装

### 1 安装

以下操作都要以**hadoop**用户身份进行

```
chu888chu888@hadoopslave1:~$ sudo tar xvfz jdk-7u79-linux-x64.gz  
hadoop@hadoopmaster:~$ sudo cp -r hadoop-2.7.2 /usr/local/hadoop  
hadoop@hadoopmaster:~$ sudo chmod -R 775 /usr/local/hadoop/  
hadoop@hadoopmaster:~$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

这里面有一个小的体验技巧,我建议将所有需要的环境变量配置加入到**/etc/profile**中,这是全局变量.

```
export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
```

还有一个问题就是,在启动hadoop的时候经常会出现,找不到JAVA\_HOME的问题,这个问题可以通过修改hadoop环境变量来解决,直接写死变量就可以了.

```

hadoop@hadoopmaster:/usr/local/hadoop/etc/hadoop$ ls
capacity-scheduler.xml  hadoop-metrics2.properties  httpfs-signature.secret
log4j.properties          ssl-client.xml.example
configuration.xsl        hadoop-metrics.properties  httpfs-site.xml
mapred-env.cmd            ssl-server.xml.example
container-executor.cfg    hadoop-policy.xml         kms-acls.xml
mapred-env.sh              yarn-env.cmd
core-site.xml             hdfs-site.xml           kms-env.sh
mapred-queues.xml.template  yarn-env.sh
hadoop-env.cmd            httpfs-env.sh           kms-log4j.properties
properties               mapred-site.xml.template  yarn-site.xml
hadoop-env.sh             httpfs-log4j.properties  kms-site.xml
slaves
hadoop@hadoopmaster:/usr/local/hadoop/etc/hadoop$ sudo nano hadoop-env.sh

$ more hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/

```

## 2 配置集群环境

集群/分布式模式需要修改 /usr/local/hadoop/etc/hadoop 中的5个配置文件，更多设置项可点击查看官方说明，这里仅设置了正常启动所必须的设置项： slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml。

## 3 配置文件内容-slaves

文件 slaves，将作为 DataNode 的主机名写入该文件，每行一个，默认为 localhost，所以在伪分布式配置时，节点即作为 NameNode 也作为 DataNode。分布式配置可以保留 localhost，也可以删掉，让 hadoopmaster 节点仅作为 NameNode 使用。

本教程让 hadoopmaster 节点仅作为 NameNode 使用，因此将文件中原来的 localhost 删除，只添加二行内容：hadoopslave1 hadoopslave2。

```
hadoop@hadoopmaster:/usr/local/hadoop/etc/hadoop$ more slaves
hadoopslave1
hadoopslave2
```

## 4 配置文件内容 -core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://hadoopmaster:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
</configuration>
```

## 5 配置文件内容 hdfs-site.xml

**dfs.replication** 一般设为 **3**，但我们只有二个 **Slave** 节点，所以 **dfs.replication** 的值还是设为 **2**：

```
<configuration>
  <property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>hadoopmaster:50090</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
  </property>
</configuration>
```

## 6 配置文件 - mapred-site.xml

(可能需要先重命名，默认文件名为 **mapred-site.xml.template**），然后配置修改如下：

```

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>hadoopmaster:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>hadoopmaster:19888</value>
  </property>
</configuration>

```

## 7 配置文件 - **yarn-site.xml**

```

<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>hadoopmaster</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>

```

## 五 **hadoopslave1 2节点需要做的**

首先通过sftp把hadoop配置好的hadoop打包,之后传输到Slave节点上,配置好环境变量JDK PATH SSH 基本上与Master是一样的.

配置好后,将 Master 上的 /usr/local/Hadoop 文件夹复制到各个节点上。因为之前有跑过伪分布式模式,建议在切换到集群模式前先删除之前的临时文件。

在 Master 节点上执行:

```
cd /usr/local
sudo rm -r ./hadoop/tmp
sudo rm -r ./hadoop/logs/*
tar -cvfz ~/hadoop.master.tar.gz ./hadoop
```

在Slave节点上执行:

```
sudo rm -r /usr/local/hadoop      # 删掉旧的（如果存在）
sudo tar -xvfz ~/hadoop.master.tar.gz -C /usr/local
sudo chown -R hadoop:hadoop /usr/local/hadoop
```

## 六 开始启动集群

```
hdfs namenode -format          # 首次运行需要执行初始化，之后不需要
在Master上执行:
$start-dfs.sh
$start-yarn.sh
$mr-jobhistory-daemon.sh start historyserver
```

Centos6.X需要关闭防火墙

```
sudo service iptables stop    # 关闭防火墙服务
sudo chkconfig iptables off  # 禁止防火墙开机自启，就不用手动关闭了
Cent7
systemctl stop firewalld.service    # 关闭firewall
systemctl disable firewalld.service # 禁止firewall开机启动
```

之后分别在Master与Slave上执行jps,会看到不同的结果.缺少任一进程都表示出错。另外还需要在 Master 节点上通过命令 hdfs dfsadmin -report 查看 DataNode 是否正常启动，如果 Live datanodes 不为 0，则说明集群启动成功。例如我这边一共有 1 个 Datanodes :

```
$jps  
$hdfs dfsadmin -report
```

可以访问<http://192.168.1.159:50070/> 查看结果

```
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ jps  
11056 Jps  
10561 ResourceManager  
10420 SecondaryNameNode  
10839 JobHistoryServer  
10207 NameNode  
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ █
```

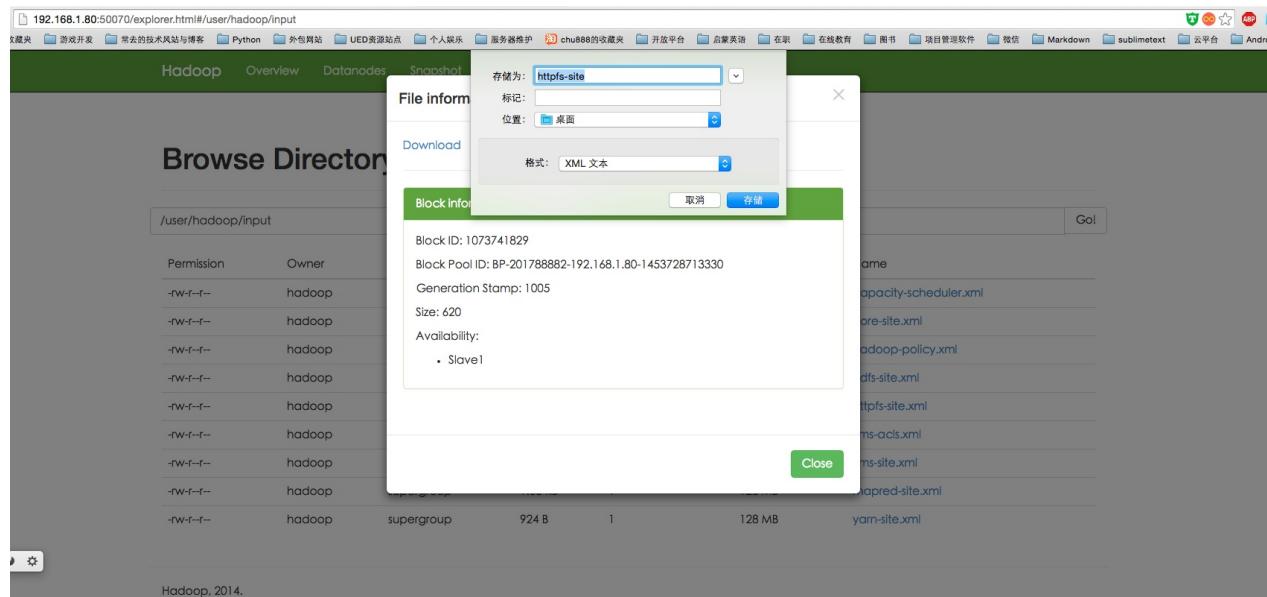
```
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ jps  
4290 Jps  
4164 NodeManager  
4026 DataNode  
hadoop@ubuntu-hadoop:/usr/local/hadoop/etc/hadoop$ █
```

## 七 执行分布式的实验-分布式存储

执行分布式实例过程与伪分布式模式一样，首先创建 HDFS 上的用户目录：

```
$ hdfs dfs -mkdir -p /user/hadoop  
将 /usr/local/hadoop/etc/hadoop 中的配置文件作为输入文件复制到分布式文  
件系统中：  
$ hdfs dfs -mkdir input  
$ hdfs dfs -put /usr/local/hadoop/etc/hadoop/*.xml input
```

通过查看 DataNode 的状态（占用大小有改变），输入文件确实复制到了 DataNode 中，如下图所示：



## 八 执行分布式的实验-MapReduce

执行 MapReduce 作业

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar grep /user/hadoop/input /user/hadoop/output 'dfs[a-z.]+'  
查看 http://192.168.1.159:8088/cluster 看结果
```

运行时的输出信息与伪分布式类似，会显示 Job 的进度。

可能会有点慢，但如果迟迟没有进度，比如 5 分钟都没看到进度，那不妨重启 Hadoop 再试试。若重启还不行，则很有可能是内存不足引起，建议增大虚拟机的内存，或者通过更改 YARN 的内存配置解决。

同样可以通过 Web 界面查看任务进度 <http://master:8088/cluster>，在 Web 界面点击“Tracking UI”这一列的 History 连接，可以看到任务的运行信息，如下图所示：

The screenshot shows the Hadoop Web UI at the URL 192.168.1.80:8088/cluster/apps/FINISHED. The title bar says "FINISHED Applications". On the left, there's a sidebar with "Cluster Metrics" and "Applications" sections. The "Applications" section lists one application: "application\_1453730825593\_0001" with status "FINISHED". The main table shows cluster metrics like memory usage and active nodes.

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	1	1	1	2 GB	8 GB	0 B	1	8	0	1	0	0	0	0

```
关闭集群  
stop-yarn.sh  
stop-dfs.sh  
mr-jobhistory-daemon.sh stop historyserver
```

## 九 同步时间

在安装hadoop环境的过程中,时间同步是非常重要的一个服务.因为分布式系统如果时间不同步会造成系统中的许多问题.

### 1 在线安装

ntp在线安装的方式很简单，只需要执行以下命令即可帮你安装好ntp以及所有的依赖包

```
sudo apt-get install ntp
```

### 2 离线安装

如果要离线安装，那么就需要下载ntp安装包和依赖包。我们可以在一个有线环境下运行上面的在线安装，然后到/var/cache/apt/archives这个目录下拷贝完整的ntp安装包和依赖包。

```
dpkg -i libopts25_1%3a5.12-0.1ubuntu1_amd64.deb  
dpkg -i ntp_1%3a4.2.6.p3+dfsg-1ubuntu3.1_amd64.deb
```

当然还有更加简单的方法，将下载的deb包拷贝到/var/cache/apt/archives目录下，然后在执行一下命令同样可以安装。

```
sudo apt-get install ntp
```

安装完毕以后我们可以查看服务是否启动，执行以下命令：

```
hadoop@hadoopmaster:~$ sudo service --status-all
[ + ]  acpid
[ + ]  apparmor
[ ? ]  apport
[ + ]  atd
[ ? ]  console-setup
[ + ]  cron
[ - ]  dbus
[ ? ]  dns-clean
[ + ]  friendly-recovery
[ - ]  grub-common
[ ? ]  irqbalance
[ ? ]  killprocs
[ ? ]  kmod
[ ? ]  networking
[ + ]  ntp
[ ? ]  ondemand
[ ? ]  pppd-dns
[ - ]  procps
[ ? ]  rc.local
[ + ]  resolvconf
[ - ]  rsync
[ + ]  rsyslog
[ ? ]  screen-cleanup
[ ? ]  sendsigs
[ - ]  ssh
[ - ]  sudo
[ + ]  udev
[ ? ]  umountfs
[ ? ]  umountnfs.sh
[ ? ]  umountroot
[ - ]  unattended-upgrades
[ - ]  urandom
```

可以看到ntp服务已经启动（[+]表示已经启动。）

### 3 配置文件 /etc/ntp.conf

建议在配置之前进行备份

```
driftfile /var/lib/ntp/ntp.drift
statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
server ntp.ubuntu.com
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery
restrict 192.168.1.0 mask 255.255.255.0 nomodify
restrict 127.0.0.1
restrict ::1
```

## 4 同步时间

```
sudo ntpdate 192.168.1.159
```

## 十 FAQ

### 1 出现的问题

```
hadoop@hadoopmaster:~$ start-dfs.sh
16/07/18 20:45:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [hadoopmaster]
hadoopmaster: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-namenode-hadoopmaster.out
hadoopsslave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-datanode-hadoopsslave1.out
hadoopsslave2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-datanode-hadoopsslave2.out
Starting secondary namenodes [hadoopmaster]
hadoopmaster: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-secondarynamenode-hadoopmaster.out
16/07/18 20:45:20 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

#### 解决方法

首先下载hadoop-native-64-2.4.0.tar :

<http://dl.bintray.com/sequenceiq/sequenceiq-bin/hadoop-native-64-2.4.0.tar>

如果你是hadoop2.6的可以下载下面这个：

<http://dl.bintray.com/sequenceiq/sequenceiq-bin/hadoop-native-64-2.6.0.tar>

下载完以后，解压到hadoop的native目录下，覆盖原有文件即可。操作如下：

```
tar -x hadoop-native-64-2.4.0.tar -C hadoop/lib/native/
```

再将环境变量导入到系统中.修改/etc/profile

```
export JAVA_LIBRARY_PATH=/usr/local/hadoop/lib/native
```

#### 时钟不正确的问题

```
fatal org.apache.hadoop.hbase.regionserver.hregionserver: master  
rejected startup because clock is out of sync  
org.apache.hadoop.hbase.clockoutofsyncexception: org.apache.hadoop.hbase.  
clockoutofsyncexception: server suc-pc,60020,1363269953  
286 has been rejected; reported time is too far out of sync with  
master. time difference of 39375ms > max allowed of 30000ms
```

小问题，一看就知道错误发生在哪。在hbase中，允许小的时间偏差，但是上面39秒的时间偏差就有点大了。如果你是联网的话，可以用ntpdate 219.158.14.130进行同步。219.158.14.130是网通北京的时间服务器，如果不行你可以用别的服务器进行同步。

这里面有一段小插曲就是,我在使用的时候,由于使用了外网的时钟服务器,但是resolv.conf反复被覆盖.所以不成功.

/etc/resolv.conf中设置dns之后每次重启Ubuntu Server时该文件会被覆盖，针对这种情况找了一些个解决方法

防止/etc/resolv.conf被覆盖的方法

方法一

1. 需要创建一个文件/etc/resolvconf/resolv.conf.d/tail

```
sudo vi /etc/resolvconf/resolv.conf.d/tail
```

2. 在该文件中写入自己需要的dns服务器，格式与/etc/resolv.conf相同

```
nameserver 8.8.8.8
```

3. 重启下resolvconf程序

```
sudo /etc/init.d/resolvconf restart
```

再去看看/etc/resolv.conf文件，可以看到自己添加的dns服务器已经加到该文件中

方法二

在/etc/network/interfaces中

```
###interfaces中#####
auto eth0
iface eth0 inet static
address 192.168.3.250
netmask 255.255.255.0          #子网掩码
gateway 192.168.3.1           #网关
dns-nameservers 8.8.8.8 8.8.4.4 #设置dns服务器
```

**zookeeper**服务器未设置或者/etc/hosts设置有误（**hbase**）

```

2013-03-11 19:41:08,263 info org.apache.zookeeper.clientcnxn: opening socket connection to server localhost/127.0.0.1:2181. will not attempt to authenticate using sasl (unknown error)
2013-03-11 19:41:08,266 warn org.apache.zookeeper.clientcnxn: session 0x0 for server null, unexpected error, closing socket connection and attempting reconnect
java.net.ConnectException: 拒绝连接
    at sun.nio.ch.SocketChannelImpl.checkConnect(native method)
    at sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:692)
    at org.apache.zookeeper.clientcnxnSocketnio.dotransport(clientcnxnSocketnio.java:350)
    at org.apache.zookeeper.clientcnxn$sendthread.run(clientcnxn.java:1068)

```

这个问题的出现，会伴随一个非常奇怪的现象。在master所在的pc上启动start-all时，内容提示所有的regionserver已经全部启动。但是，如果你去查看masterip : 60010时会发现其他的regionserver并没有启动，regionserver的数量只有一台。因为已经有一台regionserver是活着的，所以hbase还是能继续使用的，这此文来自：马开东博客 转载请注明出处 网址：<http://www.makaidong.com>会迷惑你。查看别的机器的日志后，你就会发现上述错误。zookeeper的定位居然定位到127.0.0.1去了，这个不科学。最后，查阅资料才发现hbase.zookeeper.quorum这个属性设置时，默认本机即为zookeeper服务器（单机使用）。这就很简单了，只需要增加这个属性就可以了。

```

<property>
    <name>hbase.zookeeper.quorum</name>
    <value>10.82.58.213</value>
</property>

```

同时，也发现如果/etc/hosts设置错误也会发生类似问题。/etc/hosts中，localhost和本机pc名都需要为127.0.0.1，因为本机pc名默认是127.0.1.1。

参考：[http://mail-archives.apache.org/mod\\_mbox/hbase-user/201106.mbox/%3cbanlktimcghr-1mdtdo3netzmrqxbjy=da@mail.gmail.com%3e](http://mail-archives.apache.org/mod_mbox/hbase-user/201106.mbox/%3cbanlktimcghr-1mdtdo3netzmrqxbjy=da@mail.gmail.com%3e)

# Hadoop组件

## HDFS文件操作

HDFS是一种文件系统,专为MapReduce这类框架下的大规模分布式数据处理而设计,你可以把一个大数据集(比如说100TB)在HDFS中存储为单个文件,而大多数其他的文件系统无力实现这一点。HDFS并不是一个天生的UNIX文件系统,不支持像ls和cp这种标准的UNIX文件命令,也不支持如fopen()和fread()这样的标准文件读写操作.另一方面,Hadoop确也提供了一套与Linux文件命令类似的命令行工具.

### 基本文件命令

Hadoop的文件命令采取的形式为

```
hadoop fs -cmd <args>
```

基中cmd是具体的文件命令,而 <args> 是一组数据可变的参数. cmd 的命名通常与 unix对应的命令名相同.例如,文件形表命令为

```
hadoop fs -ls

hadoop fs -mkdir /user/chuck
hadoop fs -ls /
hadoop fs -put example.txt /user/chuck
hadoop fs -ls
hadoop fs -get example.txt .
hadoop fs -cat example.txt
hadoop fs -rm example.txt
```

### cat命令

将路径指定文件的内容输出到stdout

```
hadoop@Master:~$ hadoop dfs -cat input/core-site.xml
```

## **chgrp命令**

改变文件所属组.使用-R将使改变在目录结构下递归进行.命令的使用者必须是文件的所有者或者超级用户.

## **chmod命令**

修改文件权限

## **chown命令**

改变文件的拥有者

## **cp命令**

将文件从源路径复制到目标路径,这个命令允许有多个源路径,此时目标路径必须是一个目录

```

hadoop@Master:/usr/local/hadoop/etc/hadoop$ hdfs dfs -cp input/*
  output/
hadoop@Master:/usr/local/hadoop/etc/hadoop$ hdfs dfs -ls output
Found 11 items
-rw-r--r--  1 hadoop supergroup          0 2016-01-25 22:14 out
put/_SUCCESS
-rw-r--r--  1 hadoop supergroup        4436 2016-01-27 19:16 out
put/capacity-scheduler.xml
-rw-r--r--  1 hadoop supergroup        1072 2016-01-27 19:16 out
put/core-site.xml
-rw-r--r--  1 hadoop supergroup       9683 2016-01-27 19:16 out
put/hadoop-policy.xml
-rw-r--r--  1 hadoop supergroup       1257 2016-01-27 19:16 out
put/hdfs-site.xml
-rw-r--r--  1 hadoop supergroup        620 2016-01-27 19:16 out
put/httpfs-site.xml
-rw-r--r--  1 hadoop supergroup      3523 2016-01-27 19:16 out
put/kms-acls.xml
-rw-r--r--  1 hadoop supergroup      5511 2016-01-27 19:16 out
put/kms-site.xml
-rw-r--r--  1 hadoop supergroup      1103 2016-01-27 19:16 out
put/mapred-site.xml
-rw-r--r--  1 hadoop supergroup        107 2016-01-25 22:14 out
put/part-r-00000
-rw-r--r--  1 hadoop supergroup       924 2016-01-27 19:16 out
put/yarn-site.xml

```

## du命令

显示目录中所有文件的大小,或者当只指定一个文件时,显示此文件的大小.

```

hadoop@Master:/usr/local/hadoop/etc/hadoop$ hadoop fs -du input/
core-site.xml

```

## expunge命令

清空回收站 除了文件权限之外,还有一个保护机制可以防止在HDFS上意外删除文件,这就是回收站,默认情况下该功能是被禁用.当它启用后,用于删除的命令行不会立即删除文件.

相反它们会暂时的把文件移动到用户工作目录下的.Trash文件夹下.若要启用回收站功能并设置清空回收站的时间延迟,可能通过设置core-site.xml的fs.trash.interval属性(以分钟为单位).

例如如果你希望用户有24个小时的时间来还原已删除的文件,就应该在core-site.xml中设置.

如果将该值设置为0,则将禁用回收站的功能

```
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
</property>
```

## get命令

复制文件到本地文件系统.

```
hadoop fs -get input/hadoop.tar.gz ~/
```

## lsr命令

ls命令的递归版本,类似于Unix中的ls -R

## mkdir命令

接受路径指定的uri作为参数,创建这些目录,其行为类似于Unix的mkdir -p,它会创建路径中的各级父目录.

```
hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
```

## mv命令

将文件从源路径移动到目标路径

```
hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2
```

### put命令

从本地文件系统中复制单个或者多个源路径到目标文件系统.也支持从标准输入中读取输入写入目标文件系统.

```
hadoop fs -put /tmp/*.xml /user/hadoop/
```

### rmr命令

```
hadoop fs -rmr /user/hadoop/chu888chu888
```

### job命令

- \* Job操作
- \* 提交MapReduce Job, Hadoop所有的MapReduce Job都是一个jar包
- \* \$ hadoop jar <local-jar-file> <java-class> <hdfs-input-file><hdfs-output-dir>
- \* \$ hadoop jar sandbox-mapred-0.0.20.jar sandbox.mapred.WordCountJob /user/cl/input.dat /user/cl/outputdir
- \*
- \* 杀死某个正在运行的Job
- \* 假设Job\_Id为 : job\_201207121738\_0001
- \* \$ hadoop job -kill job\_201207121738\_0001

### 系统体检

Hadoop提供的文件系统检查工具叫做fsck,如参数为文件路径时,它会递归检查该路径下所有文件的健康状态,如果参数为/,它就会检查整个文件系统,如下输出一个例子.

```
hadoop@Master:~$ hadoop fsck /
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

16/01/27 22:55:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Connecting to namenode via http://Master:50070
FSCK started by hadoop (auth:SIMPLE) from /192.168.1.80 for path /
at Wed Jan 27 22:55:15 CST 2016
.....Status: HEALTHY
Total size: 878899 B
Total dirs: 21
Total files: 21
Total symlinks: 0
Total blocks (validated): 20 (avg. block size 43944 B)
Minimally replicated blocks: 20 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 1
Average block replication: 1.0
Corrupt blocks: 0
Missing replicas: 0 (0.0 %)
Number of data-nodes: 2
Number of racks: 1
FSCK ended at Wed Jan 27 22:55:15 CST 2016 in 32 milliseconds

The filesystem under path '/' is HEALTHY
```

## 编程读写HDFS

# SSH免密钥登录

## 一 生产环境描述

正常情况下，我们需要连上SSH的控制台输入用户名及其密码才行。如果两者全部正确，我们就可以访问，反之访问被服务端拒绝。不过相比而言还有一种比用密码更安全的登录方式，我们可以在登录SSH时通过加密密钥进行无密码登录。

如果你想启用这个安全的方式，我们只需简单的禁用密码登录并只允许加密密钥登录即可。使用这种方式时，客户端计算机上会产生一对私钥和公钥。接着客户端得把公钥上传到SSH服务端的`authorized_key`文件中去。在授予访问前，服务器及客户端电脑会校验这个密钥对。如果服务器上的公钥与客服端提交的私钥匹配则授予访问权限，否则访问被拒绝。

这是认证到SSH服务器的非常安全的一种做法，如果你想为单一的SSH用户登录实现安全登录，这也是备受推崇的方式。这里快速的过一遍如何启用无密码登录SSH的配置过程。

### 1 安装**OpenSSH**服务端

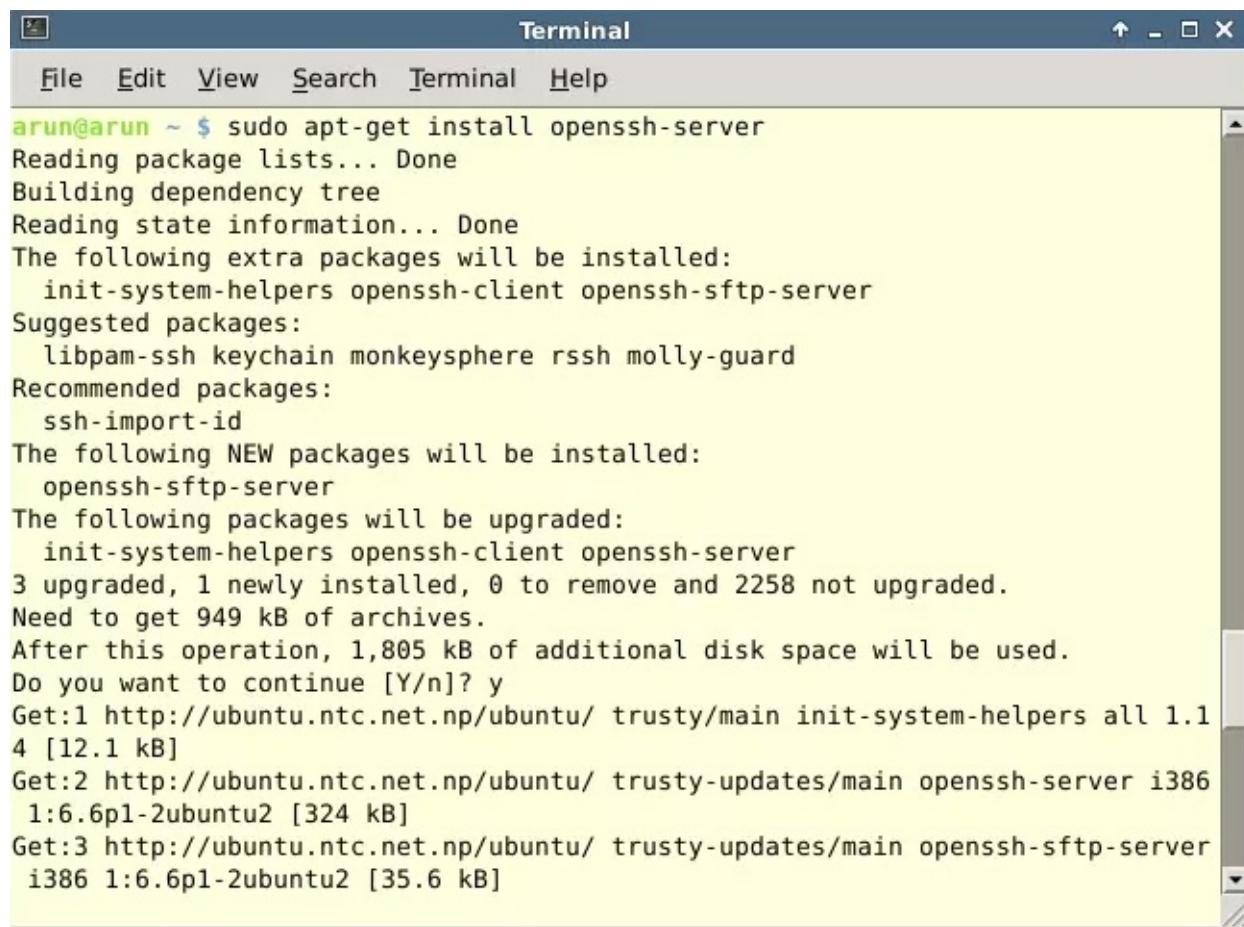
首先，我们需要更新我们的本地库索引。所以如下所见，我们需要先输入“`apt-get update`”

```
$ sudo apt-get update
```

```
arun@arun ~ $ sudo apt-get update
[sudo] password for arun:
Ign http://ubuntu.ntc.net.np trusty InRelease
Ign http://ubuntu.ntc.net.np trusty-updates InRelease
Hit http://ubuntu.ntc.net.np trusty Release.gpg
Hit http://ubuntu.ntc.net.np trusty-updates Release.gpg
Hit http://ubuntu.ntc.net.np trusty Release
Hit http://ubuntu.ntc.net.np trusty-updates Release
Hit http://ubuntu.ntc.net.np trusty/main i386 Packages
Hit http://ubuntu.ntc.net.np trusty/restricted i386 Packages
Hit http://ubuntu.ntc.net.np trusty/universe i386 Packages
Hit http://ubuntu.ntc.net.np trusty/multiverse i386 Packages
Hit http://ubuntu.ntc.net.np trusty/main Translation-en
Hit http://ubuntu.ntc.net.np trusty/multiverse Translation-en
Hit http://ubuntu.ntc.net.np trusty/restricted Translation-en
Hit http://ubuntu.ntc.net.np trusty/universe Translation-en
Hit http://ubuntu.ntc.net.np trusty-updates/main i386 Packages
Hit http://ubuntu.ntc.net.np trusty-updates/restricted i386 Packages
Hit http://ubuntu.ntc.net.np trusty-updates/universe i386 Packages
Hit http://ubuntu.ntc.net.np trusty-updates/multiverse i386 Packages
Hit http://ubuntu.ntc.net.np trusty-updates/main Translation-en
```

现在我们可以通过以下命令安装openssh-server：

```
$ sudo apt-get install openssh-server
```



```
arun@arun ~ $ sudo apt-get install openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  init-system-helpers openssh-client openssh-sftp-server
Suggested packages:
  libpam-ssh keychain monkeysphere rssh molly-guard
Recommended packages:
  ssh-import-id
The following NEW packages will be installed:
  openssh-sftp-server
The following packages will be upgraded:
  init-system-helpers openssh-client openssh-server
3 upgraded, 1 newly installed, 0 to remove and 2258 not upgraded.
Need to get 949 kB of archives.
After this operation, 1,805 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://ubuntu.ntc.net.np/ubuntu/ trusty/main init-system-helpers all 1.1
4 [12.1 kB]
Get:2 http://ubuntu.ntc.net.np/ubuntu/ trusty-updates/main openssh-server i386
 1:6.6p1-2ubuntu2 [324 kB]
Get:3 http://ubuntu.ntc.net.np/ubuntu/ trusty-updates/main openssh-sftp-server
i386 1:6.6p1-2ubuntu2 [35.6 kB]
```

## 2 开启**openSSH**服务

在OpenSSH已经成功安装在Ubuntu14.04操作系统上了之后，我们要启动OpenSSH的服务。以下命令让你启动/开启服务。

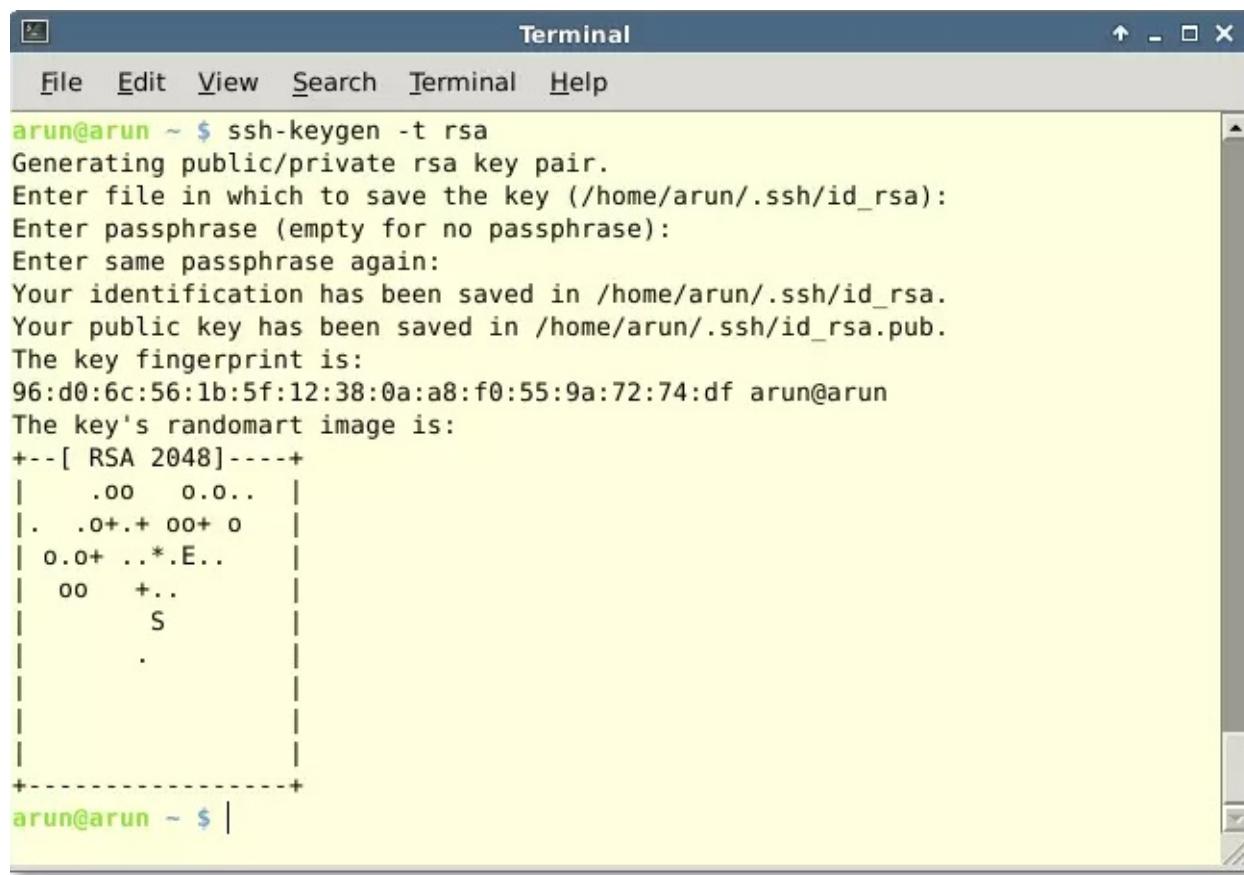
```
$ sudo service ssh start
```

## 3 配置密钥对

在我们安装并启动了OpenSSH服务以后。现在终于到了要我们搞定公私钥对的时候了，在终端中运行以下命令：

```
$ ssh-keygen -t rsa
```

在运行完以上命令了以后，我们需要回答一系列的问题。首先选择保存密钥的路径，按回车将会选择默认路径即家目录的一个隐藏的.ssh文件夹。下一个提示是请输入口令提醒。我个人将此留空（直接回车）。之后密钥对就会创建，大功告成。



```
arun@arun ~ $ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/arun/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/arun/.ssh/id_rsa.
Your public key has been saved in /home/arun/.ssh/id_rsa.pub.
The key fingerprint is:
96:d0:6c:56:1b:5f:12:38:0a:a8:f0:55:9a:72:74:df arun@arun
The key's randomart image is:
+--[ RSA 2048]----+
| .oo 0.. |
| .o+.+ oo+ o |
| o.+ ...*.E.. |
| oo +.. |
| S |
| . |
| |
+-----+
arun@arun ~ $ |
```

在密钥对生成以后，我们需要将客户端上的公钥复制到SSH服务端或者主机，来创建对客户端的信任关系。运行以下命令复制客户端的公钥到服务端。

```
$ ssh-copy-id user@ip_address
```

在公钥上传之后，我们现在可以禁用通过密码登陆SSH的方式了。为此，我们需要通过以下命令用文本编辑器打开/etc/ssh/sshd\_config。

```
$ sudo nano /etc/ssh/sshd_config
```

## 4 重启SSH服务

最后，在我们配置完SSH服务端后，为了使改动生效我们需要重启SSH服务。在终端或控制台运行以下命令重启。

```
$ sudo service ssh restart
```

## SSH配置文件详解

```
# Site-wide defaults for various options
```

带“##”表示该句为注释不起作用，该句不属于配置文件原文，意在说明下面选项均为系统初始默认的选项。说明一下，实际配置文件中也有很多选项前面加有“##”注释，虽然表示不起作用，其实是说明此为系统默认的初始化设置。

```
Host *
```

"Host"只对匹配后面字串的计算机有效，“\*”表示所有的计算机。从该项格式前置一些可以看出，这是一个类似于全局的选项，表示下面缩进的选项都适用于该设置，可以指定某计算机替换\*号使下面选项只针对该算机器生效。

```
ForwardAgent no
```

"ForwardAgent"设置连接是否经过验证代理（如果存在）转发给远程计算机。

```
ForwardX11 no
```

"ForwardX11"设置X11连接是否被自动重定向到安全的通道和显示集（DISPLAY set）。

```
RhostsAuthentication no
```

"RhostsAuthentication"设置是否使用基于rhosts的安全验证。

```
RhostsRSAAuthentication no
```

"RhostsRSAAuthentication"设置是否使用用RSA算法的基于rhosts的安全验证。

```
RSAAuthentication yes
```

"RSAAuthentication"设置是否使用RSA算法进行安全验证。

```
PasswordAuthentication yes
```

"PasswordAuthentication"设置是否使用口令验证。

```
FallBackToRsh no
```

"FallBackToRsh"设置如果用ssh连接出现错误是否自动使用rsh，由于rsh并不安全，所以此选项应当设置为"no"。

```
UseRsh no
```

"UseRsh"设置是否在这台计算机上使用"rlogin/rsh"，原因同上，设为"no"。

```
BatchMode no
```

"BatchMode"：批处理模式，一般设为"no"；如果设为"yes"，交互式输入口令的提示将被禁止，这个选项对脚本文件和批处理任务十分有用。

```
CheckHostIP yes
```

"CheckHostIP"设置ssh是否查看连接到服务器的主机的IP地址以防止DNS欺骗。建议设置为"yes"。

```
StrictHostKeyChecking no
```

"StrictHostKeyChecking"如果设为"yes"，ssh将不会自动把计算机的密匙加入"\$HOME/.ssh/known\_hosts"文件，且一旦计算机的密匙发生了变化，就拒绝连接。

```
IdentityFile ~/.ssh/identity
```

"IdentityFile"设置读取用户的RSA安全验证标识。

**Port 22**

"Port"设置连接到远程主机的端口，ssh默认端口为22。

**Cipher blowfish**

"Cipher"设置加密用的密钥，blowfish可以自己随意设置。

**EscapeChar ~**

"EscapeChar"设置escape字符。

**Port 22**

"Port"设置sshd监听的端口号。

**ListenAddress 192.168.1.1**

"ListenAddress"设置sshd服务器绑定的IP地址。

**HostKey /etc/ssh/ssh\_host\_key**

"HostKey"设置包含计算机私人密匙的文件。

**ServerKeyBits 1024**

"ServerKeyBits"定义服务器密匙的位数。

**LoginGraceTime 600**

"LoginGraceTime"设置如果用户不能成功登录，在切断连接之前服务器需要等待的时间（以秒为单位）。

**KeyRegenerationInterval 3600**

"KeyRegenerationInterval"设置在多少秒之后自动重新生成服务器的密匙（如果使用密匙）。重新生成密匙是为了防止用盗用的密匙解密被截获的信息。

**PermitRootLogin no**

"PermitRootLogin"设置是否允许root通过ssh登录。这个选项从安全角度来讲应设成"no"。

**IgnoreRhosts yes**

"IgnoreRhosts"设置验证的时候是否使用"rhosts"和"shosts"文件。

**IgnoreUserKnownHosts yes**

"IgnoreUserKnownHosts"设置ssh daemon是否在进行RhostsRSAAuthentication安全验证的时候忽略用户的"\$HOME/.ssh/known\_hosts"

**StrictModes yes**

"StrictModes"设置ssh在接收登录请求之前是否检查用户家目录和rhosts文件的权限和所有权。这通常是必要的，因为新手经常会把自己的目录和文件设成任何人都有写权限。

**X11Forwarding no**

"X11Forwarding"设置是否允许X11转发。

**PrintMotd yes**

"PrintMotd"设置sshd是否在用户登录的时候显示"/etc/motd"中的信息。

**SyslogFacility AUTH**

"SyslogFacility"设置在记录来自sshd的消息的时候，是否给出"facility code"。

### LogLevel INFO

"LogLevel"设置记录sshd日志消息的层次。INFO是一个好的选择。查看sshd的man帮助页，已获取更多的信息。

### RhostsAuthentication no

"RhostsAuthentication"设置只用rhosts或"/etc/hosts.equiv"进行安全验证是否已经足够了。

### RhostsRSAAuthentication no

"RhostsRSA"设置是否允许用rhosts或"/etc/hosts.equiv"加上RSA进行安全验证。

### RSAAuthentication yes

"RSAAuthentication"设置是否允许只有RSA安全验证。

### PasswordAuthentication yes

"PasswordAuthentication"设置是否允许口令验证。

### PermitEmptyPasswords no

"PermitEmptyPasswords"设置是否允许用口令为空的帐号登录。

### AllowUsers admin

"AllowUsers"的后面可以跟任意的数量的用户名的匹配串，这些字符串用空格隔开。主机名可以是域名或IP地址。

## FAQ

### 1、安装ssh

```
直接 sudo apt-get install openssh-server
```

### 2、查看ssh运行状态

```
ps -e | grep ssh
```

如果发现 sshd 和 ssh-agent 即表明 ssh服务基本运行正常

### 3、生成公钥和私钥

```
ssh-keygen -t rsa -P ""
```

## 4、将公钥追加到文件

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

## 5、测试ssh localhost

如果发现不用输入密码就可以登录那么 ssh无密码机制就算建立成功了。

## 6、失败原因之一：.ssh及其下属子文件的权限问题：

首选.ssh目录权限是700， 两个dsa 和 rsa的 私钥权限是600，其余文件权限是644。

下面列出.ssh目录及子文件的权限表：

```
drwx----- 2 hadoop hadoop 4096 2014-06-02 15:32 .
drwxr-xr-x 34 hadoop hadoop 4096 2014-06-02 15:06 ..
-rw-r--r-- 1 hadoop hadoop 2252 2014-06-02 15:32 authorized_keys
-rw----- 1 hadoop hadoop 668 2014-06-02 15:11 id_dsa
-rw-r--r-- 1 hadoop hadoop 615 2014-06-02 15:11 id_dsa.pub
-rw----- 1 hadoop hadoop 1675 2014-06-02 15:32 id_rsa
-rw-r--r-- 1 hadoop hadoop 407 2014-06-02 15:32 id_rsa.pub
-rw-r--r-- 1 hadoop hadoop 442 2014-06-02 15:08 known_hosts
```

## 7、.ssh的父目录的权限问题（我的问题就出现在这里）：

.ssh的父目录文件权限应该是755，即所属用户的 用户文件（/home下属的一个用户文件）。

## FAQ2

StrictModes no #修改为no,默认为yes.如果不修改用key登陆是出现server refused our key(如果StrictModes为yes必需保证存放公钥的文件夹的拥有与登陆用户名是相同的.“StrictModes”设置ssh在接收登录请求之前是否检查用户家目录和rhosts文

件的权限和所有权。这通常是必要的，因为新手经常会把自己的目录和文件设成任何人都有写权限。) (来源<http://matt-u.iteye.com/blog/851158>)

# 第八章 Hive的安装与数据导入导出

## 一 Hive介绍

即使像Hadoop这样强大的工具，也不能满足每个人的需求，许多项目如雨后春笋般涌现出来，为特定的扩展了Hadoop，那些比较突出的并且得到很好维护的项目已经正式成为Apache Hadoop项目下的子项目。Hive是Hadoop家族中一款数据仓库产品，Hive最大的特点就是提供了类SQL的语法，封装了底层的MapReduce过程，让有SQL基础的业务人员，也可以直接利用Hadoop进行大数据的操作。就是这一个点，解决了原数据分析人员对于大数据分析的瓶颈。让我们把Hive的环境构建起来，帮助非开发人员也能更好地了解大数据。

### 1 Hive介绍

Hive起源于Facebook，它使得针对Hadoop进行SQL查询成为可能，从而非程序员也可以方便地使用。Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的SQL查询功能，可以将SQL语句转换为MapReduce任务运行。

Hive是建立在Hadoop上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在Hadoop中的大规模数据的机制。

Hive定义了简单的类SQL查询语言，称为HQL，它允许熟悉SQL的用户查询数据。同时，这个语言也允许熟悉MapReduce开发者的开发自定义的mapper和reducer来处理内建的mapper和reducer无法完成的复杂的分析工作。

Hive是建立在Hadoop基础上的数据仓库软件包，在开始阶段，它被Facebook用于处理大量的用户数据和日志数据。它现在是Hadoop的子项目并有许多贡献者。其目标用户仍然是习惯SQL的数据分析师，他们需要在Hadoop规模的数据上做即席查询V汇总和数据分析。通过称为HiveQL的类SQL语言，你可以发起一个查询来实现与Hive的交互。

Hive是基于Hadoop的数据仓库平台，由Facebook贡献，其支持类似SQL的结构化查询功能。Facebook设计开发Hive的初衷就是让那些熟悉sql编程方式的人也可以更好的利用hadoop，hive可以让数据分析人员只关注于具体业务模型，而不需要深

入了解MapReduce的编程细节，但是这并不意味着使用hive不需要了解和学习MapReduce编程模型和hadoop，复杂的业务需求和模型总是存在的，对于Hive分析人员来说，深入了解Hadoop和Hive的原理和Mapreduce模型，对于优化查询总有益处。

Hive不是一个完整的数据库.Hadoop以及HDFS的设计本身约束和局限性地限制了Hive所能胜任的工作.其中最大的限制就是Hive不支持记录级别的更新\插入\删除操作.但是用户可以通过查询生成新表或者将查询结果导入到文件中.同时,因为Hadoop是一个面向批处理的系统,而MapReduce任务(job)的启动过程需要消耗较长时间,所以Hive查询延迟比较严重.传统数据库中在秒级别可以完成的查询,在Hive中,即使用数据集比较小,往往也需要执行更长的时间,最后需要说明的是Hive不支持事务.

因此Hive不支持联机事务处理OLTP,所需要的关键功能,而更接近一个OLAP联机分析技术工具.但是我们将会看到由于HADOOP本身的时间开销巨大,并且HADOOP所被设计用来处理的数据规模非常大,因此提交查询和返回结果是可能有非常大的延迟的,所以HIVE并没有满足OLAP中的联机部分.至少目前并没有满足.

因此Hive最适合数据仓库应用程序,其可以维护海量数据,而且可以对数据进行挖掘,然后形成意见和报告等.

## 2 实现机制

以下先以一个简单的例子说明利用hadoop Map\Reduce程序和Hive实现hadoop word count的例子。

```

public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable();
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
    IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}

```

MapReduce类

```

CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\n')) AS word FROM docs) w
GROUP BY word ORDER BY word;

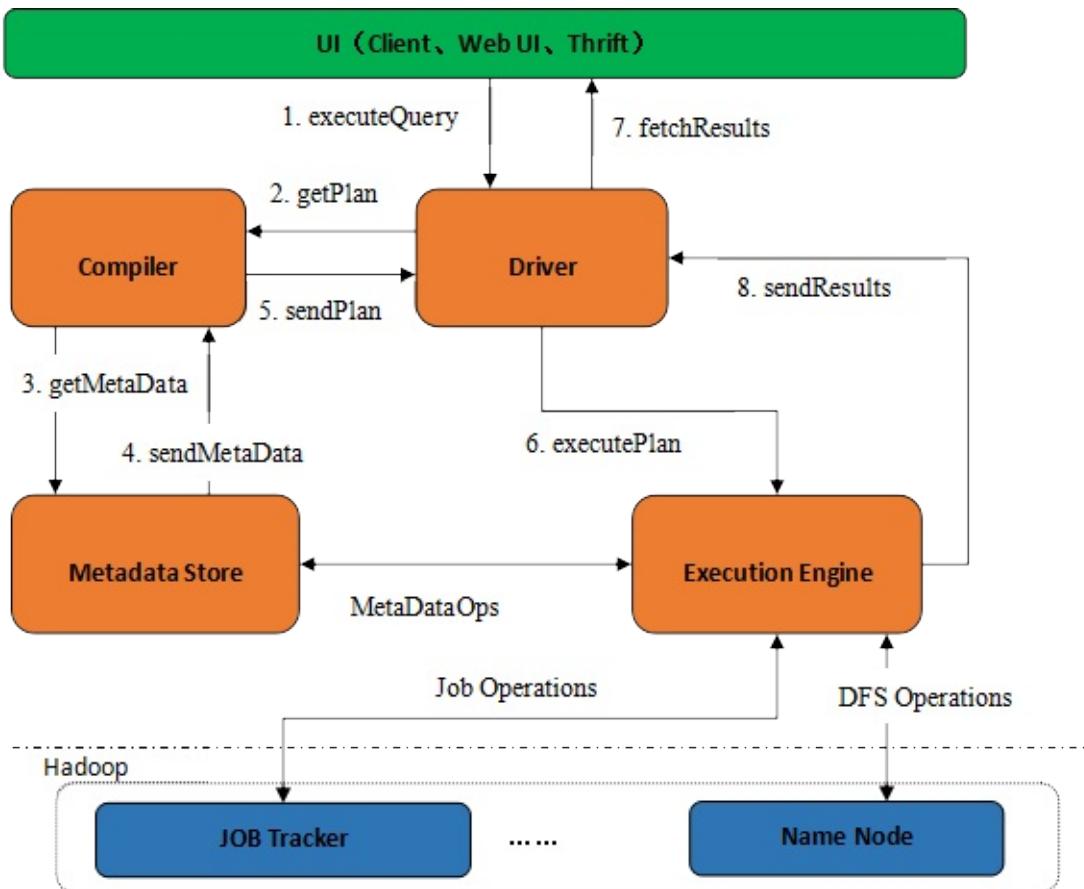
```

Hive查询

通过以上可以看出，hive优点：成本低，可以通过类sql语句快速实现简单或复杂的MapReduce统计。借助于Hadoop和HDFS的大数据存储能力，数据仍然存储于Hadoop的HDFS中，Hive提供了一种类SQL的查询语言：HiveQL（HQL），对数据进行管理和分析，开发人员可以近乎sql的方式来实现逻辑，从而加快应用开发效率。

HQL经过解析和编译，最终会生成基于Hadoop平台的Map Reduce任务，Hadoop通过执行这些任务来完成HQL的执行。Hive的设计体现出它是一个管理和查询结构化数据的系统。通过专注结构化数据，Hive可以实现MapReduce一般所不具备的某些优化和可用性功能。受到SQL影响的Hive语言让用户可以脱离MapReduce一般所不具备的某些优化和可用性功能。受到SQL影响的Hive语言让用户可以脱离MapReduce编程的复杂性。它沿用了关系数据库的常见概念，如表\行\列\Schema，以便于学习。此外，虽然Hadoop天生支持平坦文件，但Hive可以使用目录结构来划分数据，以提高某些查询的性能。为支持这些额外的功能，Hive有一个全新且重要的组件，它就是用于存储schema信息的metastore。这个metastore通常只有在关系型数据库中才有的。

Hive的组件总体上可以分为以下几个部分：用户接口（UI）、驱动、编译器、元数据（Hive系统参数数据）和执行引擎。



- 对外的接口UI包括以下几种：命令行CLI，Web界面、JDBC/ODBC接口；
- 驱动：接收用户提交的查询HQL；
- 编译器：解析查询语句，执行语法分析，生成执行计划；
- 元数据Metadata：存放系统的表、分区、列、列类型等所有信息，以及对应的HDFS文件信息等；
- 执行引擎：执行执行计划，执行计划是一个有向无环图，执行引擎按照各个任务的依赖关系选择执行任务（Job）。

需要注意的是，元数据库一般是通过关系型数据库MySQL或者支持JDBC驱动的关系型数据库来存储。元数据维护了库信息、表信息、列信息等所有内容，例如表T包含哪些列，各列的类型等等。因此元数据库十分重要，需要定期备份以及支持查询的扩展性。

### 读时验证机制

与传统数据库对表数据进行写时严重不同，Hive对数据的验证方式为读时模式，即只有在读表数据的时候，hive才检查解析具体的字段、schema等，从而保证了大数据量的快速加载。既然hive采用的读时验证机制，那么如果表schema与表文件内容不匹配，会发生什么呢？

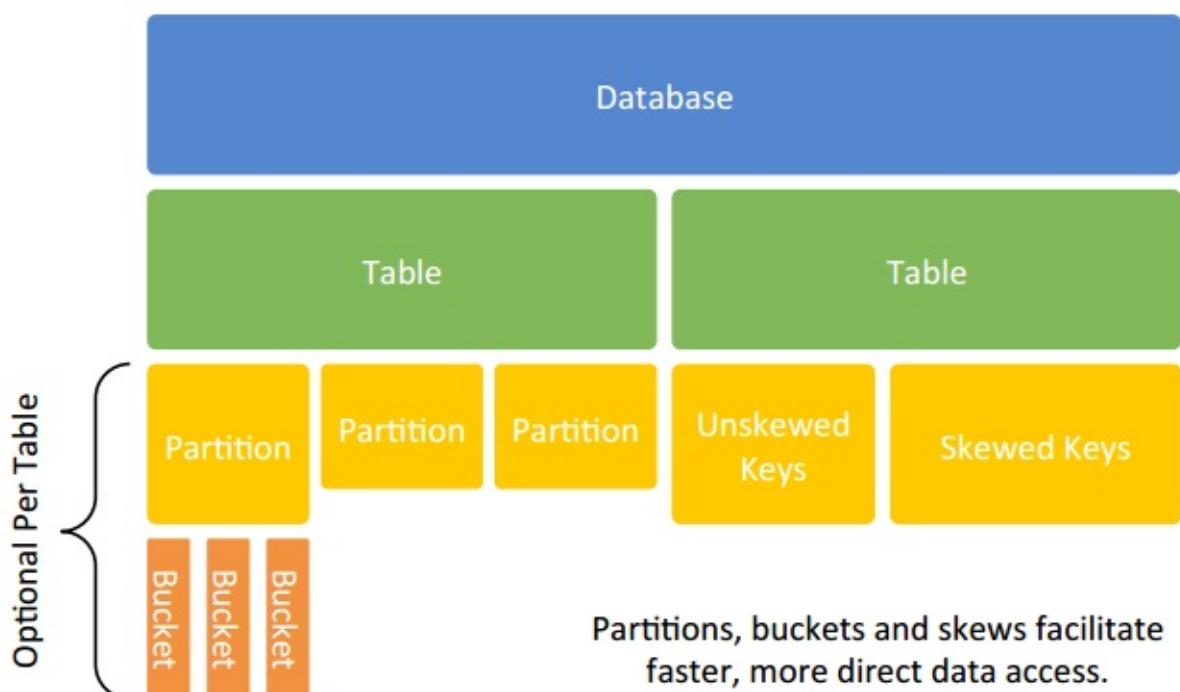
答案是hive会尽其所能的去读数据。如果schema中表有10个字段，而文件记录却只有3个字段，那么其中7个字段将为null；如果某些字段类型定位为数值类型，但是记录中却为非数值字符串，这些字段也将被转换为null。简而言之，hive会努力catch读数据时遇到的错误，并努力返回。

既然Hive表数据存储在HDFS中且Hive采用的是读时验证方式，定义完表的schema会自动生成表数据的HDFS目录，且我们可以以任何可能的方式来加载表数据或者利用HDFS API将数据写入文件，同理，当我们若需要将hive数据写入其他库（如oracle），也可以直接通过api读取数据再写入目标库。在实际生产环境中，当需要数据仓库之间的迁移时，就可以直接利用api将源库的数据直接写入hive库的表文件中，包括淘宝开源的datax数据交换系统都采用类似的方式来交换跨库数据。

再次注意，加载或者写入的数据内容要和表定义的schema一致，否则将会造成字段或者表为空。

### 3.Hive的数据模型

从数据仓库的角度看，Hive是建立在Hadoop上的数据仓库基础架构，可以方便的ETL操作。Hive没有专门的数据存储格式，也没有为数据建立索引，用于可以非常自由的组织Hive中的表，只需要在创建表的时候定义好表的schema即可。Hive中包含4中数据模型：Table、ExternalTable、Partition、Bucket。



- **Table**：类似与传统数据库中的Table，每一个Table在Hive中都有一个相应的目

录来存储数据。例如：一个表t，它在HDFS中的路径为：

\user\hive\warehouse\t。

- **Partition**：类似于传统数据库中划分列的索引。在Hive中，表中的一个Partition对应于表下的一个目录，所有的Partition数据都存储在对应的目录中。例如：t表中包含ds和city两个Partition，则对应于ds=2014，city=beijing的HDFS子目录为：\user\hive\warehouse\t\ds=2014\city=Beijing；需要注意的是，分区列是表的伪列，表数据文件中并不存在这个分区列的数据。
- **Buckets**：对指定列计算的hash，根据hash值切分数据，目的是为了便于并行，每一个Buckets对应一个文件。将user列分数至32个Bucket上，首先对user列的值计算hash，比如，对应hash=0的HDFS目录为：  
\user\hive\warehouse\t\ds=2014\city=Beijing\part-00000；对应hash=20的目录为：\user\hive\warehouse\t\ds=2014\city=Beijing\part-00020。
- **External Table**指向已存在HDFS中的数据，可创建Partition。Managed Table创建和数据加载过程，可以用统一语句实现，实际数据被转移到数据仓库目录中，之后对数据的访问将会直接在数据仓库的目录中完成。删除表时，表中的数据和元数据都会删除。External Table只有一个过程，因为加载数据和创建表是同时完成。数据是存储在Location后面指定的HDFS路径中的，并不会移动到数据仓库中。

## 4.Hive如何转化成Mapreduce

Hive编译器将HQL代码转换成一组操作符（operator），操作符是Hive的最小操作单元，每个操作符代表了一种HDFS操作或者MapReduce作业。Hive中的操作符包括：

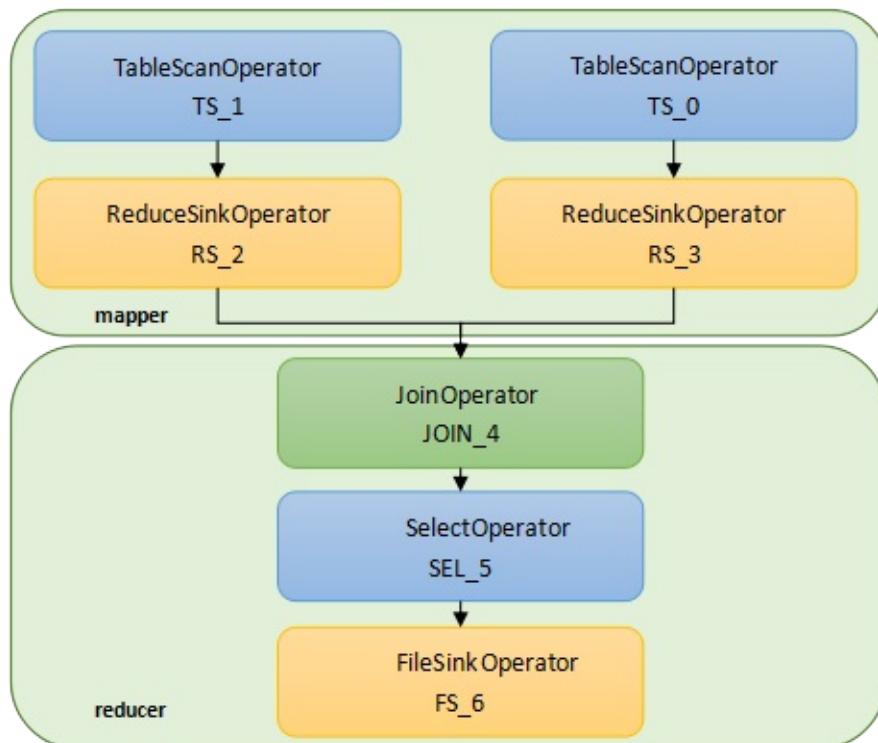
表：Hive执行常用的操作符列表

操作符	描述
TableScanOperator	扫描hive表数据
ReduceSinkOperator	创建将发送到Reducer端的<Key,Value>对
JoinOperator	Join两份数据
SelectOperator	选择输出列
FileSinkOperator	建立结果数据，输出至文件
FilterOperator	过滤输入数据
GroupByOperator	Group By 语句
MapJoinOperator	Mapjoin
LimitOperator	Limit语句
UnionOperator	Union语句

对于MapReduce操作单元，Hive通过ExecMapper和ExecReducer执行MapReduce任务。

对于Hive语句：

```
INSERT OVERWRITE TABLE read_log_tmp
SELECT a.userid,a.bookid,b.author,b.categoryid
FROM user_read_log a JOIN book_info b ON a.bookid = b.bookid;
```



## 二 Hive与其他数据库的区别

由于 Hive 采用了 SQL 的查询语言 HQL，因此很容易将 Hive 理解为数据库。其实从结构上来看，Hive 和数据库除了拥有类似的查询语言，再无类似之处。本文将从多个方面来阐述 Hive 和数据库的差异。数据库可以用在 Online 的应用中，但是 Hive 是为数据仓库而设计的，清楚这一点，有助于从应用角度理解 Hive 的特性。

查询语言	HQL	SQL
数据存储位置	HDFS	Raw Device 或者 Local FS
数据格式	用户定义	系统决定
数据更新	不支持	支持
索引	无	有
执行	MapReduce	Executor
执行延迟	高	低
可扩展性	高	低
数据规模	大	小

### Hive 和数据库的比较

1. 查询语言。由于 SQL 被广泛的应用在数据仓库中，因此，专门针对 Hive 的特性设计了类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。
2. 数据存储位置。Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。
3. 数据格式。Hive 中没有定义专门的数据格式，数据格式可以由用户指定，用户定义数据格式需要指定三个属性：列分隔符（通常为空格、“\t”、“\x001”）、行分隔符（“\n”）以及读取文件数据的方法（Hive 中默认有三个文件格式 TextFile，SequenceFile 以及 RCFile）。由于在加载数据的过程中，不需要从用户数据格式到 Hive 定义的数据格式的转换，因此，Hive 在加载的过程中不会对数据本身进行任何修改，而只是将数据内容复制或者移动到相应的 HDFS 目录中。而在数据库中，不同的数据库有不同的存储引擎，定义了自己的数据格式。所有数据都会按照一定的组织存储，因此，RDBMS 数据库加载数据的过程会比较耗时。
4. 数据更新。由于 Hive 是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive 中不支持对数据的改写和添加，所有的数据都是在加载的时候中确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用 INSERT INTO ... VALUES 添加数据，使用 UPDATE ... SET 修改数据。
5. 索引。之前已经说过，Hive 在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些 Key 建立索引。Hive 要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入，Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。数据库中，通常会针对

一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

6. 执行。Hive 中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的（类似 `select * from tbl` 的查询不需要 MapReduce）。而数据库通常有自己的执行引擎。
7. 执行延迟。之前提到，Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致 Hive 执行延迟高的因素是 MapReduce 框架。由于 MapReduce 本身具有较高的延迟，因此在利用 MapReduce 执行 Hive 查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。hive 执行延迟高，只有在数据规模达到一定程度后，其查询的高效才能弥补其高延迟的劣势。
8. 可扩展性。由于 Hive 是建立在 Hadoop 之上的，因此 Hive 的可扩展性是和 Hadoop 的可扩展性是一致的（世界上最大的 Hadoop 集群在 Yahoo!，2009 年的规模在 4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 Oracle 在理论上的扩展能力也只有 100 台左右。
9. 数据规模。由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

## 1.Hive的安装

### Hive的安装-MySQL作为元数据库

- 安装JDK-略过
- 安装Hadoop-略过
- 安装Mysql-略过

### 1建立Hive数据库,用户,赋予权限

```
#mysql虚拟机的默认密码,在我做试验的时候是123456
#mysql -u root -p
mysql>grant all privileges on *.* to hive@ "%" identified by "hive"
      with grant option;
mysql>flush privileges;
Mysql在Ubuntu中默认安装后,只能在本机访问,如果要开启远程访问,需要做以下两个步骤:
#nano /etc/mysql/my.cnf
找到bind-address=127.0.0.1 ,把这一行注释掉
```

## 2安装Hive

```
hadoop@Master:~$ sudo tar xvfz apache-hive-1.1.1-bin.tar.gz
hadoop@Master:~$ sudo cp -R apache-hive-1.1.1-bin /usr/local/hive
hadoop@Master:~$ sudo chmod -R 775 /usr/local/hive/
hadoop@Master:~$ sudo chown hadoop:hadoop /usr/local/hive/
#修改/etc/profile加入HIVE_HOME的变量
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=. :${JAVA_HOME}/lib:${JRE_HOME}/lib:/usr/local/hive/lib

#修改hive/conf下的几个template模板并重命名为其他
```

```
cp hive-env.sh.template hive-env.sh
cp hive-default.xml.template hive-site.xml

#配置hive-env.sh文件，指定HADOOP_HOME
HADOOP_HOME=/usr/local/hadoop

#修改hive-site.xml文件，指定MySQL数据库驱动、数据库名、用户名及密码，修改
的内容如下所示

<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://192.168.1.178:3306/hive?createDatabaseIfNo
tExist=true</value>
    <description>JDBC connect string for a JDBC metastore</descrip
tion>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>Driver class name for a JDBC metastore</descripti
on>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
    <description>username to use against metastore database</descr
ipton>
</property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
    <description>password to use against metastore database</descr
ipton>
</property>
```

其中：

javax.jdo.option.ConnectionURL参数指定的是Hive连接数据库的连接字符串；  
javax.jdo.option.ConnectionDriverName参数指定的是驱动的类入口名称；  
javax.jdo.option.ConnectionUserName参数指定了数据库的用户名；  
javax.jdo.option.ConnectionPassword参数指定了数据库的密码。

### 3修改**hive/bin**下的**hive-config.sh**文件，设置**JAVA\_HOME,HADOOP\_HOME**

```
export JAVA_HOME=/usr/lib/jvm  
export HADOOP_HOME=/usr/local/hadoop  
export HIVE_HOME=/usr/local/hive
```

### 4下载**mysql-connector-java-5.1.27-bin.jar**文件，并放到**\$HIVE\_HOME/lib**目录下

可以从Mysql的官方网站下载，但是记得一定要解压呀，下载的是一个tar.gz文件

### 5在**HDFS**中创建**/tmp**和**/user/hive/warehouse**并设置权限

```
hadoop fs -mkdir /tmp  
hadoop fs -mkdir /user/hive/warehouse  
hadoop fs -chmod g+w /tmp  
hadoop fs -chmod g+w /user/hive/warehouse
```

### 6启动**hadoop**。进入**hive shell**，输入一些命令查看

```
hive  
show databases;  
show tables;
```

### 7可以在**hadoop**中查看**hive**生产的文件

```
hadoop dfs -ls /user/hive/warehouse
```

## Hive使用实例

在正式讲解HiveQL之前,先在命令行下运行几样命令是有好处的,可以感受一下HiveQL是如何工作的,也可以自己随便探索一下.

## 1查询示例

```
hive> SHOW TABLES;
OK
testuser
Time taken: 0.707 seconds, Fetched: 1 row(s)

hive> DESC testuser;
OK
id          int
username      string

Time taken: 0.38 seconds, Fetched: 2 row(s)
hive> SELECT * from testuser limit 10;
OK
1    sssss
1    sssss
Time taken: 0.865 seconds, Fetched: 2 row(s)
hive>

hive> select count(1) from testuser;
Query ID = hadoop_20160205004747_9d84aaca-887a-43a0-bad9-eddefe4
e2219
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1454604205731_0001, Tracking URL = http://Mas
ter:8088/proxy/application_1454604205731_0001/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_14546
```

## 1.Hive 1.1.1的安装

```
04205731_0001
Hadoop job information for Stage-1: number of mappers: 1; number
of reducers: 1
2016-02-05 00:48:11,942 Stage-1 map = 0%,  reduce = 0%
2016-02-05 00:48:19,561 Stage-1 map = 100%,  reduce = 0%,  Cumula
tive CPU 1.38 sec
2016-02-05 00:48:28,208 Stage-1 map = 100%,  reduce = 100%,  Cumu
lative CPU 2.77 sec
MapReduce Total cumulative CPU time: 2 seconds 770 msec
Ended Job = job_1454604205731_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 2.77 sec  HD
FS Read: 6532 HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 770 msec
OK
2
Time taken: 35.423 seconds, Fetched: 1 row(s)
```

通过这些消息,可以知道该查询生成了一个**Mapreduce**作业,**Hive**之美在于用户根本不需要知道**MapReduce**的存在,用户所需关心的,仅仅是使用一种类似于**SQL**的语言.

多次重复实现大量数据插入

```
hive> insert overwrite table testuser
  > select id,count(id)
  > from testuser
  > group by id;
```

# Hive的安装

## 一 安装环境

- Hadoop 2.7.2
- JDK 1.7 U79
- Hive 2.1.0
- Mysql(apt-get 安装)
- 192.168.1.166为Mysql server meta server安装位置
- 192.168.1.159为Hive数据仓库安装位置

## 二 Hive的安装 -MySQL作为元数据库

- 安装JDK-略过
- 安装Hadoop-略过
- 安装Mysql-略过

## 三 在192.168.1.166上建立Hive meta数据库,用户,赋予权限

mysql虚拟机的默认密码,在我做试验的时候是**123456**

```
$mysql -u root -p  
mysql>grant all privileges on *.* to hive@ "%" identified by "hive"  
e" with grant option;  
mysql>flush privileges;
```

Mysql在Ubuntu中默认安装后,只能在本机访问,如果要开启远程访问,需要做以下两个步骤:

```
$nano /etc/mysql/my.cnf
```

找到bind-address=127.0.0.1 ,把这一行注释掉

```
$service mysql restart
```

## 四 在192.168.1.159上安装Hive

### 1 安装Hive

```
hadoop@hadoopmaster:~$ sudo tar xvfz apache-hive-2.1.0-bin.tar.gz  
hadoop@hadoopmaster:~$ sudo cp -R apache-hive-2.1.0-bin /usr/local/hive  
hadoop@hadoopmaster:~$ sudo chmod -R 775 /usr/local/hive/  
hadoop@hadoopmaster:~$ sudo chown -R hadoop:hadoop /usr/local/hive
```

### 2 修改/etc/profile加入HIVE\_HOME的变量

```
export HIVE_HOME=/usr/local/hive  
export PATH=$PATH:$HIVE_HOME/bin  
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:/usr/local/hive/lib  
$source /etc/profile
```

### 3 修改hive/conf下的几个template模板并重命名为其他

```
cp hive-env.sh.template hive-env.sh  
cp hive-default.xml.template hive-site.xml
```

### 配置hive-env.sh文件，指定HADOOP\_HOME

```
HADOOP_HOME=/usr/local/hadoop
```

### 4 修改hive-site.xml文件，指定MySQL数据库驱动、数据库名、用户名及密码，修改的内容如下所示

```
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://192.168.1.178:3306/hive?createDatabaseIfNo
tExist=true</value>
    <description>JDBC connect string for a JDBC metastore</descrip
tion>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>Driver class name for a JDBC metastore</descripti
on>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
    <description>username to use against metastore database</descr
ipton>
</property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
    <description>password to use against metastore database</descr
ipton>
</property>
```

其中：

javax.jdo.option.ConnectionURL参数指定的是Hive连接数据库的连接字符串；  
javax.jdo.option.ConnectionDriverName参数指定的是驱动的类入口名称；  
javax.jdo.option.ConnectionUserName参数指定了数据库的用户名；  
javax.jdo.option.ConnectionPassword参数指定了数据库的密码。

## 5 缓存目录的问题,如果不配置也会出错的

```
<property>
<name>hive.exec.local.scratchdir</name>
<value>/home/hadoop/iotmp</value>
<description>Local scratch space for Hive jobs</description>
</property>
<property>
<name>hive.downloaded.resources.dir</name>
<value>/home/hadoop/iotmp</value>
<description>Temporary local directory for added resources in the remote file system.</description>
</property>
```

并且需要对目录进行权限设定

```
mkdir -p /home/hadoop/iotmp
chmod -R 775 /home/hadoop/iotmp
```

## 五修改**hive/bin**下的**hive-config.sh**文件，设置**JAVA\_HOME,HADOOP\_HOME**

```
export JAVA_HOME=/usr/lib/jvm
export HADOOP_HOME=/usr/local/hadoop
export HIVE_HOME=/usr/local/hive
```

## 六 下载**mysql-connector-java-5.1.27-bin.jar**文件，并放到**\$HIVE\_HOME/lib**目录下

可以从**Mysql**的官方网站下载,但是记得一定要解压呀,下载的是一个**tar.gz**文件

## 七 在**HDFS**中创建**/tmp**和**/user/hive/warehouse**并设置权限

```
hadoop fs -mkdir /tmp  
hadoop fs -mkdir -p /user/hive/warehouse  
hadoop fs -chmod g+w /tmp  
hadoop fs -chmod g+w /user/hive/warehouse
```

## 1 初始化**meta**数据库

进入之前需要初始化数据库

```
schematool -initSchema -dbType mysql  
  
hadoop@hadoopmaster:/usr/local/hive/lib$ schematool -initSchema  
-dbType mysql  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4  
j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop  
/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLogge  
rBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for  
an explanation.  
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4j  
LoggerFactory]  
Metastore connection URL: jdbc:mysql://192.168.1.166:3306/hive?c  
reateDatabaseIfNotExist=true  
Metastore Connection Driver : com.mysql.jdbc.Driver  
Metastore connection User: hive  
Starting metastore schema initialization to 2.1.0  
Initialization script hive-schema-2.1.0.mysql.sql  
Initialization script completed  
schemaTool completed
```

## 2 测试**hive shell**

```
hive  
show databases;  
show tables;
```

### 3可以在中查看生产的文件

```
hadoop dfs -ls /user/hive/warehouse
```

## 七 Hive shell使用实例

在正式讲解HiveQL之前,先在命令行下运行几样命令是有好处的,可以感受一下HiveQL是如何工作的,也可以自己随便探索一下.

### 1 创建数据(文本以**tab**分隔)

```
~ vi /home/cos/demo/t_hive.txt  
  
16      2      3  
61      12     13  
41      2      31  
17      21     3  
71      2      31  
1       12     34  
11      2      34
```

### 2 创建新表

```
hive> CREATE TABLE t_hive (a int, b int, c int) ROW FORMAT DELIM  
ITED FIELDS TERMINATED BY '\t';  
OK  
Time taken: 0.121 seconds
```

### 3 导入数据t\_hive.txt到t\_hive表

```
hive> LOAD DATA LOCAL INPATH '/tmp/t_hive.txt' OVERWRITE INTO TA  
BLE t_hive ;  
Loading data to table default.t_hive  
OK  
Time taken: 0.609 seconds
```

### 4 查看表

```
hive> show tables;  
OK  
t_hive  
Time taken: 0.099 seconds
```

### 5 正则匹配表名

```
hive>show tables '*t*';  
OK  
t_hive  
Time taken: 0.065 seconds
```

### 6 查看表数据

```
hive> select * from t_hive;  
OK  
16      2      3  
61      12     13  
41      2      31  
17      21     3  
71      2      31  
1       12     34  
11      2      34  
Time taken: 0.264 seconds
```

## 7 查看表结构

```
hive> desc t_hive;
OK
a      int
b      int
c      int
Time taken: 0.1 seconds
```

## 8 增加一个字段

```
hive> ALTER TABLE t_hive ADD COLUMNS (new_col String);
OK
Time taken: 0.186 seconds
hive> desc t_hive;
OK
a      int
b      int
c      int
new_col string
Time taken: 0.086 seconds
```

## 9 重命令表名

```
~ ALTER TABLE t_hive RENAME TO t.hadoop;
OK
Time taken: 0.45 seconds
hive> show tables;
OK
t.hadoop
Time taken: 0.07 seconds
```

## 10 删除表

```
hive> DROP TABLE t_hadoop;
OK
Time taken: 0.767 seconds

hive> show tables;
OK
Time taken: 0.064 seconds
```

## 八 使用beeline

HiveServer2提供了一个新的命令行工具Beeline，它是基于SQLLine CLI的JDBC客户端。关于SQLLine的知识，可以参考这个网站：<http://sqlline.sourceforge.net/#manual>

Beeline工作模式有两种，即本地嵌入模式和远程模式。嵌入模式情况下，它返回一个嵌入式的Hive（类似于Hive CLI）。而远程模式则是通过Thrift协议与某个单独的HiveServer2进程进行连接通信。下面给一个简单的登录Beeline的使用实例：

### 1 首先把驱动拷贝到Lib中

```
sudo cp jdbc/hive-jdbc-2.1.0-standalone.jar /usr/local/hive/lib/
```

### 2 启动hiveserver2的服务

命令行模式：

```
hive --service hiveserver2 --hiveconf hive.server2.thrift.port=10001
```

服务模式：

```
hiveserver2 start
```

### 3 执行操作

```
% bin/beeline
Hive version 0.11.0-SNAPSHOT by Apache
beeline> !connect jdbc:hive2://localhost:10000/default
!connect jdbc:hive2://localhost:10000/default
Connecting to jdbc:hive2://localhost:10000/default
Connected to: Hive (version 0.10.0)
Driver: Hive (version 0.10.0-SNAPSHOT)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000> show tables;
show tables;
+-----+
|   tab_name   |
+-----+
| primitives  |
| src         |
| src1        |
| src_json    |
| src_sequencefile |
| src_thrift  |
| srcbucket   |
| srcbucket2 |
| srcpart     |
+-----+
9 rows selected (1.079 seconds)
```

## 九 FAQ

### 出错信息1

```
hadoop@hadoopmaster:/usr/local/hive/conf$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4
j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop
/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLogge
rBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for
```

```
an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4j
LoggerFactory]

Logging initialized using configuration in jar:file:/usr/local/h
ive/lib/hive-common-2.1.0.jar!/hive-log4j2.properties Async: tru
e

Exception in thread "main" java.lang.RuntimeException: org.apach
e.hadoop.hive.ql.metadata.HiveException: org.apache.hadoop.hive.
ql.metadata.HiveException: MetaException(message:Hive metastore
database is not initialized. Please use schematool (e.g. ./schem
atool -initSchema -dbType ...) to create the schema. If needed,
don't forget to include the option to auto-create the underlying
database in your JDBC connection string (e.g. ?createDatabaseIf
NotExist=true for mysql))
    at org.apache.hadoop.hive.ql.session.SessionState.start(Session
State.java:578)
    at org.apache.hadoop.hive.ql.session.SessionState.beginStart(Se
ssionState.java:518)
    at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:705)
    at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:641
)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcce
ssorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMe
thodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:606)
    at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
    at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
Caused by: org.apache.hadoop.hive.ql.metadata.HiveException: org
.apache.hadoop.hive.ql.metadata.HiveException: MetaException(mes
sage:Hive metastore database is not initialized. Please use sche
matool (e.g. ./schematool -initSchema -dbType ...) to create the
schema. If needed, don't forget to include the option to auto-c
reate the underlying database in your JDBC connection string (e.
g. ?createDatabaseIfNotExist=true for mysql))
    at org.apache.hadoop.hive.ql.metadata.Hive.registerAllFunctions
Once(Hive.java:226)
    at org.apache.hadoop.hive.ql.metadata.Hive.<init>(Hive.java:366
```

```
)  
    at org.apache.hadoop.hive.ql.metadata.Hive.create(Hive.java:310  
)  
    at org.apache.hadoop.hive.ql.metadata.Hive.getInternal(Hive.jav  
a:290)  
    at org.apache.hadoop.hive.ql.metadata.Hive.get(Hive.java:266)  
    at org.apache.hadoop.hive.ql.session.SessionState.start(Session  
State.java:545)  
    ... 9 more  
Caused by: org.apache.hadoop.hive.ql.metadata.HiveException: Met  
aException(message:Hive metastore database is not initialized. P  
lease use schematool (e.g. ./schematool -initSchema -dbType ...)  
to create the schema. If needed, don't forget to include the op  
tion to auto-create the underlying database in your JDBC connect  
ion string (e.g. ?createDatabaseIfNotExist=true for mysql))  
    at org.apache.hadoop.hive.ql.metadata.Hive.getAllFunctions(Hive  
.java:3593)  
    at org.apache.hadoop.hive.ql.metadata.Hive.reloadFunctions(Hive  
.java:236)  
    at org.apache.hadoop.hive.ql.metadata.Hive.registerAllFunctions  
Once(Hive.java:221)  
    ... 14 more  
Caused by: MetaException(message:Hive metastore database is not  
initialized. Please use schematool (e.g. ./schematool -initSchem  
a -dbType ...) to create the schema. If needed, don't forget to  
include the option to auto-create the underlying database in you  
r JDBC connection string (e.g. ?createDatabaseIfNotExist=true fo  
r mysql))  
    at org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:336  
4)  
    at org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:333  
6)  
    at org.apache.hadoop.hive.ql.metadata.Hive.getAllFunctions(Hive  
.java:3590)  
    ... 16 more
```

没有执行

```
schematool -initSchema -dbType mysql
```

执行之后搞定

## 出错信息2

```
hadoop@hadoopmaster:/usr/local/hive/lib$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-2.1.0.jar!/hive-log4j2.properties Async: true
Exception in thread "main" java.lang.IllegalArgumentException: java.net.URISyntaxException: Relative path in absolute URI: ${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D
    at org.apache.hadoop.fs.Path.initialize(Path.java:205)
    at org.apache.hadoop.fs.Path.<init>(Path.java:171)
    at org.apache.hadoop.hive.ql.session.SessionState.createSessionDirs(SessionState.java:631)
    at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:550)
    at org.apache.hadoop.hive.ql.session.SessionState.beginStart(SessionState.java:518)
    at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:705)
    at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:641)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:606)
```

```
at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
Caused by: java.net.URISyntaxException: Relative path in absolute URI: ${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D
at java.net.URI.checkPath(URI.java:1804)
at java.net.URI.<init>(URI.java:752)
at org.apache.hadoop.fs.Path.initialize(Path.java:202)
... 12 more
```

**hive-site.xml**中没有配置合理临时目录的问题

```
<property>
<name>hive.exec.local.scratchdir</name>
<value>/home/hadoop/iotmp</value>
<description>Local scratch space for Hive jobs</description>
</property>
<property>
<name>hive.downloaded.resources.dir</name>
<value>/home/hadoop/iotmp</value>
<description>Temporary local directory for added resources in the remote file system.</description>
</property>
<property>
```

并且需要对目录进行权限设定

```
mkdir -p /home/hadoop/iotmp
chmod -R 775 /home/hadoop/iotmp
```

## 参考文档

- [出错信息解决](#)
- [配置参考](#)
- [配置参考](#)
- [出错信息解决](#)



## 2.Hive的数据导入导出

### Hive数据的导入

- 从本地文件系统中导入数据到Hive表；
- 从HDFS上导入数据到Hive表；
- 从别的表中查询出相应的数据并导入到Hive表中；
- 在创建表的时候通过从别的表中查询出相应的记录并插入到所创建的表中。

#### 1、从本地文件系统中导入数据到Hive表先在Hive里面创建好表

如下：

```
hive> create table wyp (id int, name string, age int, tel string)
      ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
```

这个表很简单，只有四个字段，具体含义我就不解释了。本地文件系统里面有个/home/wyp/wyp.txt文件，内容如下：

```
[wyp@master ~]$ cat wyp.txt
1 wyp 25 1318888888888
2 test 30 1388888888888
3 zs 34 899314121
```

wyp.txt文件中的数据列之间是使用\t分割的，可以通过下面的语句将这个文件里面的数据导入到wyp表里面，操作如下：

```
hive> load data local inpath 'wyp.txt' into table wyp;
Copying data from file:/home/wyp/wyp.txt
Copying file: file:/home/wyp/wyp.txt
Loading data to table default.wyp
Table default.wyp stats:
[num_partitions: 0, num_files: 1, num_rows: 0, total_size: 67]
OK
Time taken: 5.967 seconds
```

这样就将wyp.txt里面的内容导入到wyp表里面去了，可以到wyp表的数据目录下查看，如下命令：

```
hive> dfs -ls /user/hive/warehouse/wyp ;
Found 1 items
-rw-r--r-- 3 wyp supergroup 67 2014-02-19 18:23 /hive/warehouse/wyp/wyp.txt
```

需要注意的是：

和我们熟悉的关系型数据库不一样，Hive现在还不支持在insert语句里面直接给出一组记录的文字形式，也就是说，Hive并不支持INSERT INTO .... VALUES形式的语句。

## 2、HDFS上导入数据到Hive表

从本地文件系统中将数据导入到Hive表的过程中，其实是先将数据临时复制到HDFS的一个目录下（典型的情况是复制到上传用户的HDFS home目录下，比如/home/wyp/），然后再将数据从那个临时目录下移动（注意，这里说的是移动，不是复制！）到对应的Hive表的数据目录里面。

既然如此，那么Hive肯定支持将数据直接从HDFS上的一个目录移动到相应Hive表的数据目录下，假设有下面这个文件/home/wyp/add.txt，具体的操作如下：

```
[wyp@master /home/q/hadoop-2.2.0]$ bin/hadoop fs -cat /home/wyp/add.txt
5 wyp1 23 131212121212
6 wyp2 24 134535353535
7 wyp3 25 132453535353
8 wyp4 26 154243434355
```

上面是需要插入数据的内容，这个文件是存放在HDFS上/home/wyp目录（和一中提到的不同，一中提到的文件是存放在本地文件系统上）里面，我们可以通过下面的命令将这个文件里面的内容导入到Hive表中，具体操作如下：

```
hive> load data inpath '/home/wyp/add.txt' into table wyp;
Loading data to table default.wyp
Table default.wyp stats:
[num_partitions: 0, num_files: 2, num_rows: 0, total_size: 215]
OK
Time taken: 0.47 seconds

hive> select * from wyp;
OK
5 wyp1 23 131212121212
6 wyp2 24 134535353535
7 wyp3 25 132453535353
8 wyp4 26 154243434355
1 wyp 25 13188888888888
2 test 30 13888888888888
3 zs 34 899314121
Time taken: 0.096 seconds, Fetched: 7 row(s)
```

从上面的执行结果我们可以看到，数据的确导入到wyp表中了！请注意load data inpath '/home/wyp/add.txt' into table wyp;里面是没有local这个单词的，这个是和一中的区别。

### 3、从别的表中查询出相应的数据并导入到Hive表中

假设Hive中有test表，其建表语句如下所示：

```
hive> create table test(  
> id int, name string  
> ,tel string)  
> partitioned by  
> (age int)  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY '\t'  
> STORED AS TEXTFILE;  
OK  
Time taken: 0.261 seconds
```

下面语句就是将wyp表中的查询结果并插入到test表中：

```
hive> insert into table test  
> partition (age='25')  
> select id, name, tel  
> from wyp;  
#####  
####  
这里输出了一堆Mapreduce任务信息，这里省略  
#####  
####  
Total MapReduce CPU Time Spent: 1 seconds 310 msec  
OK  
Time taken: 19.125 seconds
```

```
hive> select * from test;  
OK  
5 wyp1 131212121212 25  
6 wyp2 134535353535 25  
7 wyp3 132453535353 25  
8 wyp4 154243434355 25  
1 wyp 13188888888888 25  
2 test 13888888888888 25  
3 zs 899314121 25  
Time taken: 0.126 seconds, Fetched: 7 row(s)
```

这里做一下说明：我们知道我们传统数据块的形式`insert into table values`（字段1，字段2），这种形式hive是不支持的。

通过上面的输出，我们可以看到从wyp表中查询出来的东西已经成功插入到test表中去了！如果目标表（test）中不存在分区字段，可以去掉`partition (age='25')`语句。当然，我们也可以在`select`语句里面通过使用分区值来动态指明分区：

```
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> insert into table test
> partition (age)
> select id, name,
> tel, age
> from wyp;
#####
#####
# here output了一堆Mapreduce任务信息，这里省略
#####
#####
Total MapReduce CPU Time Spent: 1 seconds 510 msec
OK
Time taken: 17.712 seconds

hive> select * from test;
OK
5 wyp1 131212121212 23
6 wyp2 134535353535 24
7 wyp3 132453535353 25
1 wyp 13188888888888 25
8 wyp4 154243434355 26
2 test 13888888888888 30
3 zs 899314121 34
Time taken: 0.399 seconds, Fetched: 7 row(s)
```

这种方法叫做动态分区插入，但是Hive中默认是关闭的，所以在使用前需要先把`hive.exec.dynamic.partition.mode`设置为`nonstrict`。当然，Hive也支持`insert overwrite`方式来插入数据，从字面我们就可以看出，`overwrite`是覆盖的意思，是的，执行完这条语句的时候，相应数据目录下的数据将会被覆盖！而`insert into`则不会，注意两者之间的区别。例子如下：

```
hive> insert overwrite table test
> PARTITION (age)
> select id, name, tel, age
> from wyp;
```

更可喜的是，Hive还支持多表插入，什么意思呢？在Hive中，我们可以把insert语句倒过来，把from放在最前面，它的执行效果和放在后面是一样的，如下：

```
hive> show create table test3;
OK
CREATE TABLE test3(
id int,
name string)
Time taken: 0.277 seconds, Fetched: 18 row(s)
```

```
hive> from wyp
> insert into table test
> partition(age)
> select id, name, tel, age
> insert into table test3
> select id, name
> where age>25;
```

```
hive> select * from test3;
OK
8 wyp4
2 test
3 zs
Time taken: 4.308 seconds, Fetched: 3 row(s)
```

可以在同一个查询中使用多个insert子句，这样的好处是我们只需要扫描一遍源表就可以生成多个不相交的输出。这个很酷吧！

## 4、在创建表的时候通过从别的表中查询出相应的记录并插入到所创建的表中

在实际情况中，表的输出结果可能太多，不适于显示在控制台上，这时候，将Hive的查询输出结果直接存在一个新的表中是非常方便的，我们称这种情况为CTAS (create table .. as select) 如下：

```
hive> create table test4
> as
> select id, name, tel
> from wyp;

hive> select * from test4;
OK
5 wyp1 131212121212
6 wyp2 134535353535
7 wyp3 132453535353
8 wyp4 154243434355
1 wyp 13188888888888
2 test 13888888888888
3 zs 899314121
Time taken: 0.089 seconds, Fetched: 7 row(s)
```

数据就插入到test4表中去了，CTAS操作是原子的，因此如果select查询由于某种原因失败，新表是不会创建的！

## Hive数据的导出

根据导出的地方不一样，将这些方式分为三种：

- 导出到本地文件系统；
- 导出到HDFS中；
- 导出到Hive的另一个表中。

### 1、导出到本地文件系统

```
hive> insert overwrite local directory '/home/wyp/wyp'  
> select * from wyp;
```

这条HQL的执行需要启用Mapreduce完成，运行完这条语句之后，将会在本地文件系统的/home/wyp/wyp目录下生成文件，这个文件是Reduce产生的结果（这里生成的文件名是000000\_0），我们可以看看这个文件的内容：

```
[wyp@master ~/wyp]$ vim 000000_0  
5^Awyp1^A23^A131212121212  
6^Awyp2^A24^A134535353535  
7^Awyp3^A25^A132453535353  
8^Awyp4^A26^A154243434355  
1^Awyp^A25^A13188888888888  
2^Atest^A30^A13888888888888  
3^Azs^A34^A899314121  
  
```
```

可以看出，这就是wyp表中的所有数据。数据中的列与列之间的分隔符是^A(ascii码是\00001)。

和导入数据到Hive不一样，不能用insert into来将数据导出：

#### ##2、导出到HDFS中

和导入数据到本地文件系统一样的简单，可以用下面的语句实现：

```
hive> insert overwrite directory '/home/wyp/hdfs'
```

select \* from wyp; ``` 将会在HDFS的/home/wyp/hdfs目录下保存导出来的数据。注意，和导出文件到本地文件系统的HQL少一个local，数据的存放路径就不一样了。

# Hive 数据类型和文件格式

## 一 概述

Hive支持关系型数据库中的大多数基本数据类型,同时也支持关系型数据库中很少出现的3种集合数据类型.

## 1 基本数据类型

Hive支持多种不同长度的整型和浮点型数据类型,支持布尔类型,也支持无长度限制的字符串类型.

| 数据类型      | 长度             | 例子                    |
|-----------|----------------|-----------------------|
| TINYINT   | 1byte 有符号整数    | 20                    |
| SMALLINT  | 2byte 有符号整数    | 20                    |
| INT       | 4byte 有符号整数    | 20                    |
| BIGINT    | 8byte 有符号整数    | 20                    |
| BOOLEAN   | 布尔类型true/false | TRUE                  |
| FLOAT     | 单精度浮点数         | 3.14159               |
| DOUBLE    | 双精度浮点数         | 3.14159               |
| STRING    | 字符序列           | 'now is time'         |
| TIMESTAMP | 整数             | '2012-02-03 12:34:56' |
| BINARY    | 字节数组           |                       |

## 2 集合数据类型

Hive中列支持使用struct map array集合数据类型.

| 数据类型   | 长度                   | 例子                               |
|--------|----------------------|----------------------------------|
| STRUCT | 和C语言中的struct类似       | struct('John','Doe')             |
| MAP    | MAP是一组键值对集合          | map('first','join','last','doe') |
| ARRAY  | 数组是一组具有相同类型和名称的变量的集合 | Array('John','Doe')              |

我们来看一个简单的集合类型的例子

```

0: jdbc:hive2://localhost:10000/default> create table employees(
...     . . . . . > name STRING,
...     . . . . . > salary FLOAT,
...     . . . . . > subordinates ARRAY<STRING>,
...     . . . . . > deductions MAP<STRING, FLOAT>,
...     . . . . . > address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>);
OK
No rows affected (1.228 seconds)
0: jdbc:hive2://localhost:10000/default>

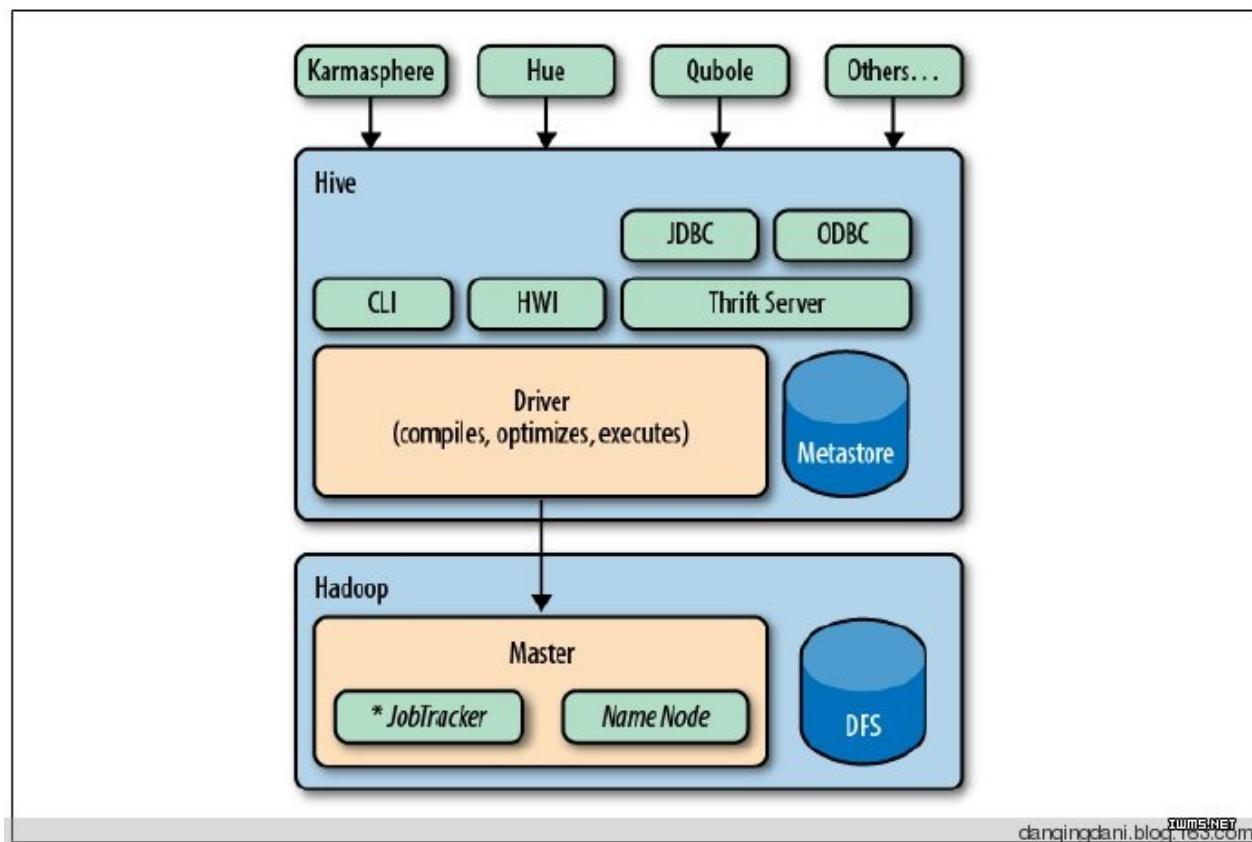
```

# Hive的存储架构与HQL语法

## 一 概述

当然，Hive和传统的关系型数据库有很大的区别，Hive将外部的任务解析成一个MapReduce可执行计划，而启动MapReduce是一个高延迟的一件事，每次提交任务和执行任务都需要消耗很多时间，这也就决定Hive只能处理一些高延迟的应用（如果你想处理低延迟的应用，你可以去考虑一下Hbase）。

同时，由于设计的目标不一样，Hive目前还不支持事务；不能对表数据进行修改（不能更新、删除、插入；只能通过文件追加数据、重新导入数据）；不能对列建立索引（但是Hive支持索引的建立，但是不能提高Hive的查询速度。如果你想提高Hive的查询速度，请学习Hive的分区、桶的应用）。





## 1 主要分为以下几个部分：

- 用户接口主要有三个：CLI，Client 和 WUI。其中最常用的是CLI，Cli启动的时候，会同时启动一个Hive副本。Client是Hive的客户端，用户连接至Hive Server。在启动 Client模式的时候，需要指出Hive Server所在节点，并且在该节点启动Hive Server。WUI是通过浏览器访问Hive。
- Hive将元数据存储在数据库中，如mysql、derby。Hive中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。
- 解释器、编译器、优化器完成HQL查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在HDFS中，并在随后有MapReduce调用执行。
- Hive的数据存储在HDFS中，大部分的查询、计算由MapReduce完成（包含的查询，比如`select from tbl`不会生成MapRedcue任务）。

## 2 元数据存储模式

Hive中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。由于Hive的元数据需要不断的更新、修改，而HDFS系统中的文件是多读少改的，这显然不能将Hive的元数据存储在HDFS中。目前Hive将元数据存储在数据库中，如Mysql、Derby中。我们可以通过以下的配置来修改Hive元数据的存储方式：配置mysql访问地址，用户名及密码

```
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://192.168.1.178:3306/hive?createDatabaseIfNo
tExist=true</value>
    <description>JDBC connect string for a JDBC metastore</descrip
tion>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>Driver class name for a JDBC metastore</descripti
on>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
    <description>username to use against metastore database</descr
iption>
</property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
    <description>password to use against metastore database</descr
iption>
</property>
```

hive会把所有的源数据存储在mysql上

```
chu888chu888@ubuntu-mysql:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 60
Server version: 5.5.44-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All righ
ts reserved.

Oracle is a registered trademark of Oracle Corporation and/or it
s
```

```
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current in  
put statement.
```

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| hive |  
| hive_hadoop |  
| hivetestdb |  
| mysql |  
| performance_schema |  
+-----+  
6 rows in set (0.03 sec)
```

```
mysql> use hive;  
Reading table information for completion of table and column nam  
es  
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
```

```
mysql> show tables;  
+-----+  
| Tables_in_hive |  
+-----+  
| BUCKETING_COLS |  
| CDS |  
| COLUMNS_V2 |  
| DATABASE_PARAMS |  
| DBS |  
| FUNCS |  
| FUNC_RU |  
| GLOBAL_PRIVS |  
| PARTITIONS |  
| PARTITION_KEYS |  
| PARTITION_KEY_VALS |
```

```

| PARTITION_PARAMS      |
| PART_COL_STATS        |
| ROLES                 |
| SDS                   |
| SD_PARAMS              |
| SEQUENCE_TABLE         |
| SERDES                |
| SERDE_PARAMS           |
| SKEWED_COL_NAMES       |
| SKEWED_COL_VALUE_LOC_MAP |
| SKEWED_STRING_LIST     |
| SKEWED_STRING_LIST_VALUES |
| SKEWED_VALUES          |
| SORT_COLS              |
| TABLE_PARAMS            |
| TAB_COL_STATS           |
| TBLS                   |
| VERSION                |
+-----+
29 rows in set (0.00 sec)

```

mysql>

## 二 数据存储

首先，Hive 没有专门的数据存储格式，也没有为数据建立索引，用户可以非常自由的组织 Hive 中的表，只需要在创建表的时候告诉 Hive 数据中的列分隔符和行分隔符，Hive 就可以解析数据。Hive 中所有的数据都存储在 HDFS 中，存储结构主要包括数据库、文件、表和视图。默认路径在 HDFS 的 /user/hive/warehouse 目录下。Hive 中包含以下数据模型：Table 内部表，External Table 外部表，Partition 分区，Bucket 桶。Hive 默认可以直接加载文本文件，还支持 sequence file 、 RCFile 。其次，Hive 中所有的数据都存储在 HDFS 中，Hive 中包含以下数据模型：Table，External Table，Partition，Bucket。

### 1) Hive 数据库

Hive中的数据库的概念本质上仅仅是表的一个目录或者命名空间.然而,对于具有很多组和用户的大集群来说,这是非常有用的,因为这样可以避免命名冲突.通常会使用数据库来将生产表组织成逻辑组.

如果用户没有显式指定数据库,那么将会使用默认的数据库**default**.

简单示例命令行

```
hive > create database test_database;
hive> show databases;
OK
default
test_database
```

如果数据库**finacials**已经存在的话,那么将会抛出一个错误信息,使用如下语句可以避免在这种情况下抛出错误信息:

```
0: jdbc:hive2://localhost:10000/default> create database if not
exists fincials;
```

如果数据库非常多的话,那么可以使用正则表达式匹配来筛选出需要的数据库表名

```
0: jdbc:hive2://localhost:10000/default> show databases like 'd.
*';
OK
+-----+---+
| database_name   |
+-----+---+
| default          |
+-----+---+
1 row selected (0.052 seconds)
```

Hive会为每个数据创建一个目录,数据库中的表将会以这个数据库目录的子目录形式存储.有一个例外就是**default**数据库中的表,因为这个数据库本身没有自己的目录.当我们创建数据库**financials**时,Hive将会对应地创建一个目录/`/user/hive/warehouse/financials.db`.这里请注意,数据库的文件目录名是以`.db`结尾的.

```
hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse
Found 4 items
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 17:25 /us
er/hive/warehouse/employees
drwxrwxr-x  - hadoop supergroup          0 2016-07-21 12:53 /us
er/hive/warehouse/fincials.db
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 09:50 /us
er/hive/warehouse/t_hive
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 09:54 /us
er/hive/warehouse/t_hive2
```

用户可以通过如下命令来修改这个默认的位置

```
0: jdbc:hive2://localhost:10000/default> create database finan
als2
. > location '/user/hive/wa
rehouse/fincials2.db';
OK
No rows affected (0.053 seconds)
```

基本的数据库操作

```

0: jdbc:hive2://localhost:10000/default> show databases;
OK
+-----+---+
| database_name   |
+-----+---+
| default         |
| financials2     |
| fincials        |
+-----+---+
3 rows selected (0.029 seconds)
0: jdbc:hive2://localhost:10000/default> use fincials;
OK
No rows affected (0.041 seconds)
0: jdbc:hive2://localhost:10000/default> drop database if exists
  financials2;
OK
No rows affected (0.238 seconds)
0: jdbc:hive2://localhost:10000/default>

```

## 2) 内部表

Hive的内部表与数据库中的Table在概念上是类似。每一个Table在Hive中都有一个相应的目录存储数据。

例如一个表pvs，它在HDFS中的路径为/warehouse/pvs，其中wh是在hive-site.xml中由\${hive.metastore.warehouse.dir}指定的数据仓库的目录，所有的Table数据（不包括External Table）都保存在这个目录中。删除表时，元数据与数据都会被删除。

内部表简单示例：

```

hive> create table test_inner_table (user_id int, cid string, c
  kid string, username string) row format delimited fields termina
  ted by '\t' lines terminated by '\n';

```

导入数据表的数据格式是：字段之间是tab键分割，行之间是断行。及要我们的文件内容格式：

```
100636 100890 c5c86f4cddc15eb7 yyyyvybtvt
100612 100865 97cc70d411c18b6f gyvcycy
100078 100087 ecd6026a15ffddf5 qa000100
```

首先在/tmp/目录下面建一个文件load.txt

```
sudo nano load.txt
hive> load data local inpath '/tmp/load.txt' into table test_inner_table;
hive> select * from test_inner_table;
hive> select count(*) from test_inner_table;
```

Hive 中的 Table 和数据库中的 Table 在概念上是类似的，每一个 Table 在 Hive 中都有一个相应的目录存储数据。例如，一个表 pvs，它在 HDFS 中的路径为：/warehouse/pvs，其中，warehouse 是在 hive-site.xml 中由 \${hive.metastore.warehouse.dir} 指定的数据仓库的目录，所有的 Table 数据（不包括 External Table）都保存在这个目录中。

### 3) 外部表

外部表指向已经在HDFS中存在的数据，可以创建Partition。它和内部表在元数据的组织上是相同的，而实际数据的存储则有较大的差异。

内部表的创建过程和数据加载过程这两个过程可以分别独立完成，也可以在同一个语句中完成，在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据的访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除。而外部表只有一个过程，加载数据和创建表同时完成（CREATE EXTERNAL TABLE .....LOCATION），实际数据是存储在LOCATION后面指定的 HDFS 路径中，并不会移动到数据仓库目录中。当删除一个External Table时，仅删除该链接。

外部表简单示例：

```
hive> create external table test_external_table (user_id int, c
id string, ckid string, username string) row format delimited fi
elds terminated by '\t' lines terminated by '\n';
```

导入数据表的数据格式是：字段之间是tab键分割，行之间是断行。及要我们的文件内容格式：

```
100636 100890 c5c86f4cddc15eb7 yyyyvybtvt  
100612 100865 97cc70d411c18b6f gyvcycy  
100078 100087 ecd6026a15ffddf5 qa000100
```

首先在/tmp/目录下面建一个文件load.txt

```
sudo nano load.txt  
hive> load data local inpath '/tmp/load.txt' into table test_external_table;  
hive> select * from test_external_table;  
hive> select count(*) from test_external_table;
```

### 4) 分区表

数据分区的概念存在已久.其可以有多种形式,但是通常使用分区来水平分散压力,将数据从物理上转移到和使用最频繁的用户更近的地方,以及实现其他目的.

Hive中有分区表的概念.我们可以看到分区表具有重要的性能优势,而且分区表还可以将数据以一种符合逻辑的方式进行组织,比如分层存储.

Partition对应于数据库中的Partition列的密集索引，但是Hive中Partition的组织方式和数据库中的很不相同。在Hive中，表中的一个Partition对应于表下的一个目录，所有的Partition的数据都存储在对应的目录中。

例如logs表中包含ds和city两个Partition，则对应于ds = 20090801, ctry = US 的HDFS子目录为/wh/pvs/ds=20090801/ctry=US；对应于 ds = 20090801, ctry = CA 的HDFS子目录为/wh/pvs/ds=20090801/ctry=CA。

分区表简单示例：

```
hive> create table logs(ts bigint,line string) partitioned by(dt
    string,country string) row format delimited fields terminated b
y '$' lines terminated by '\n';
```

data.txt 内容：

```
hadoop@hadoopmaster:/tmp$ more data1.txt
1$1
2$3
3$4
```

加载数据：

```
hive> load data local inpath '/tmp/data.txt' into table logs par
tition(dt='2015-01-01',country='zh');
hive> load data local inpath '/tmp/data.txt' into table logs par
tition(dt='2015-04-05',country='jp');
hive> load data local inpath '/tmp/data.txt' into table logs par
tition(dt='2015-04-05',country='zh');
```

查看数据：

```
0: jdbc:hive2://localhost:10000/default> select * from logs;
OK
+-----+-----+-----+-----+
| logs.ts | logs.line | logs.dt | logs.country |
+-----+-----+-----+-----+
| 1       | 1          | 2015-04-05 | jp        |
| 2       | 3          | 2015-04-05 | jp        |
| 3       | 4          | 2015-04-05 | jp        |
| NULL    | NULL       | 2015-04-05 | jp        |
| 1       | 1          | 2015-04-08 | cn        |
| 2       | 3          | 2015-04-08 | cn        |
| 3       | 4          | 2015-04-08 | cn        |
| NULL    | NULL       | 2015-04-08 | cn        |
+-----+-----+-----+-----+
8 rows selected (0.091 seconds)
```

Partition 对应于数据库中的 Partition 列的密集索引，但是 Hive 中 Partition 的组织方式和数据库中的很不相同。在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中。

例如：pvs 表中包含 ds 和 city 两个 Partition，则对应于  $ds = 20090801$ ,  $ctry = US$  的 HDFS 子目录为：`/wh/pvs/ds=20090801/ctry=US`；对应于  $ds = 20090801$ ,  $ctry = CA$  的 HDFS 子目录为：`/wh/pvs/ds=20090801/ctry=CA`, 我们看一下物理存储的结构

### Browse Directory

/user/hive/warehouse/logs						Go!
Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	dt=2015-01-01
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	dt=2015-04-05

### Browse Directory

/user/hive/warehouse/logs/dt=2015-04-05						Go!
Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	country=jp
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	country=zh

### Browse Directory

/user/hive/warehouse/logs/dt=2015-01-01/country=zh						Go!
Permission	Owner	Group	Size	Replication	Block Size	Name
-rwxr-xr-x	hadoop	supergroup	42 B	1	128 MB	data.txt

## 6) 外部分区表

外部表同样可以使用分区。事实上，用户可能会发现，这是管理大型生产型数据最常见的情况。这种结合给用户提供了一个可以和其他工具共享数据的方式，同时也可以优化查询性能。

因为用户可以自己定义目录结构，因此用户对于目录结构的使用具有更多的灵活性。

我们来举一个新例子，非常适合这种场景，即日志文件分析。对于日志文件信息，大多数的组织使用一种标准的格式，其中记录有时间戳，严重程度，也许还包含有服务器名称和进程ID，然后跟着一个可以为任何内容的文本信息。

```

create external table if not exists log_messages
(
    hms      INT,
    severity STRING,
    server    STRING,
    process_id INT,
    message   STRING
) partitioned by (year int,month int ,day int)
row format delimited fields terminated by '\t';

alter table log_message add partition(year=2012,month=1,day=2)
location 'hdfs://hadoopmaster/data/log/log_message/2012/01/02'

show partitions log_messages;
describe extended log_messages;

```

## 5) 桶

Buckets是将表的列通过Hash算法进一步分解成不同的文件存储。它对指定列计算hash，根据hash值切分数据，目的是为了并行，每一个Bucket对应一个文件。例如将user列分散至32个bucket，首先对user列的值计算hash，对应hash值为0的HDFS目录为/wh/pvs/ds=20090801/ctry=US/part-00000；hash值为20的HDFS目录为/wh/pvs/ds=20090801/ctry=US/part-00020。如果想应用很多的Map任务这样是不错的选择。

桶的简单示例：

创建数据文件：test\_bucket\_table.txt

创建表：按id划分4个桶

```
hive> CREATE TABLE bucketed_user (id INT, name STRING) CLUSTERED
    BY (id) INTO 4 BUCKETS row format delimited fields terminated b
y '\t' lines terminated by '\n';
```

加载数据：insert overwrite table bucketed\_user select \* from users ;

查看数据：select \* from bucket\_user; set hive.enforce.bucketing =
true;

Buckets 对指定列计算 hash，根据 hash 值切分数据，目的是为了并行，每一个 Bucket 对应一个文件。

将 user 列分散至 32 个 bucket，首先对 user 列的值计算 hash，对应 hash 值为 0 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00000；hash 值为 20 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00020

## 四 HQL DDL 语法

### 1 前言

Hive 是基于Hadoop 构建的一套数据仓库分析系统，它提供了丰富的SQL查询方式来分析存储在Hadoop 分布式文件系统中的数据，

可以将结构化的数据文件映射为一张数据库表，并提供完整的SQL查询功能，可以将SQL语句转换为MapReduce任务进行运行，通过自己的SQL 去查询分析需要的内容，这套SQL 简称Hive SQL，使不熟悉mapreduce 的用户很方便的利用SQL 语言查询，汇总，分析数据。

而mapreduce开发人员可以把已写的mapper 和reducer 作为插件来支持Hive 做更复杂的数据分析。它与关系型数据库的SQL 略有不同，但支持了绝大多数的语句如DDL、DML 以及常见的聚合函数、连接查询、条件查询。

HIVE不适合用于联机（online）事务处理，也不提供实时查询功能。它最适合应用在基于大量不可变数据的批处理作业。HIVE的特点：可伸缩（在Hadoop的集群上动态的添加设备），可扩展，容错，输入格式的松散耦合。

### 2 DDL操作

- 建表
- 删除表
- 修改表结构
- 创建/删除视图
- 显示命令

#### 建表

##### 1.建立一个简单的表

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

### 2. 创建表并创建索引字段ds

```
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

### 3. 复制一个空表

```
hive> CREATE TABLE empty_key_value_store LIKE testuser;
```

### 4. 建立一个需要导入数据的表

```
hive> create table user_info (user_id int, cid string, ckid string, username string) row format delimited fields terminated by '\t' lines terminated by '\n';
```

导入数据表的数据格式是：字段之间是tab键分割，行之间是断行。

及要我们的文件内容格式：

```
100636 100890 c5c86f4cddc15eb7 yyyyvybtvt  
100612 100865 97cc70d411c18b6f gyvcycy  
100078 100087 ecd6026a15ffddf5 qa000100
```

首先在/tmp/目录下面建一个文件load.txt

```
#sudo nano load.txt  
hive> load data local inpath '/tmp/load.txt' into table user_info;  
hive> select * from user_info;
```

## 5 显示所有表

```
hive> show tables;
```

## 6 按正则表达式显示表

```
hive> show tables '.*s';
```

## 7 给表增加一列

```
hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
OK
Time taken: 0.238 seconds
hive> desc pokes;
OK
foo          int
bar          string
new_col      int
Time taken: 0.275 seconds, Fetched: 3 row(s)
```

### 8 给表添加一列并添加字段解析

```
hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');
OK
Time taken: 0.151 seconds
hive> desc invites;
OK
foo          int
bar          string
new_col2    int          a comment
ds           string
# Partition Information
# col_name      data_type      comment
ds           string
Time taken: 0.163 seconds, Fetched: 9 row(s)
```

### 9 更改表名字

```
hive> ALTER TABLE invites RENAME TO 3koobecaf;
OK
Time taken: 0.189 seconds
hive> show tables;
OK
3koobecaf
empty_key_value_store
pokes
testuser
user_info
Time taken: 0.045 seconds, Fetched: 5 row(s)
```

### 10 删除表

```
hive> drop table pokes;
```

### 11 创建数据库

```
hive> CREATE DATABASE chu888chu888;
```

## 五 DML DQL操作

hive不支持用insert语句一条一条的进行插入操作，也不支持update操作。数据是以load的方式加载到建立好的表中。数据一旦导入就不可以修改。

### 1 DML包括：

- INSERT插入、UPDATE更新、DELETE删除
- 向数据表内加载文件
- 将查询结果插入到Hive表中
- 0.8新特性 insert into

向数据表内加载文件

- 相对路径，例如：project/data1
- 绝对路径，例如：/user/hive/project/data1
- 包含模式的完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1

例如：hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;

## 2 DQL操作

### 基本的Select 操作

- 使用ALL和DISTINCT选项区分对重复记录的处理。默认是ALL，表示查询所有记录。DISTINCT表示去掉重复的记录
- Where 条件
- 类似我们传统SQL的where 条件
- 目前支持 AND,OR ,0.9版本支持between
- IN, NOT IN
- 不支持EXIST ,NOT EXIST

### ORDER BY与SORT BY的不同

- ORDER BY 全局排序，只有一个Reduce任务
- SORT BY 只在本机做排序

### Limit

- Limit 可以限制查询的记录数 SELECT \* FROM t1 LIMIT 5
- 实现Top k 查询
- 下面的查询语句查询销售记录最大的 5 个销售代表。

```
SET mapred.reduce.tasks = 1
SELECT * FROM test SORT BY amount DESC LIMIT 5
```

### REGEX Column Specification

SELECT 语句可以使用正则表达式做列选择，下面的语句查询除了 ds 和 hr 之外的所有列：

```
SELECT `^(ds|hr)?+.+` FROM test
```

按先件查询

```
hive> SELECT a.foo FROM invites a WHERE a.ds='<DATE>' ;
```

将查询数据输出至目录：

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='<DATE>' ;
```

将查询结果输出至本地目录：

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;
```

选择所有列到本地目录：

```
hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;
```

```
hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;
```

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;
```

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_4' select a.invites, a.pokes FROM profiles a;
```

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(1) FROM invites a WHERE a.ds='<DATE>' ;
```

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;
```

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;
```

将一个表的统计结果插入另一个表中：

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(1) WHERE a.foo > 0 GROUP BY a.bar;
```

```
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(1) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;  
JOIN
```

```
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo, t2.foo;
```

将多表数据插入到同一表中：

```
FROM src
INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src
.key >= 100 and src.key < 200

INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12')
SELECT src.key WHERE src.key >= 200 and src.key < 300

INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.val
ue WHERE src.key >= 300;
```

将文件流直接插入文件：

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSF
ORM(a.foo, a.bar) AS (oof, rab) USING '/bin/cat' WHERE a.ds > '2
008-08-09';
```

### 3 基于Partition的查询

- 一般 SELECT 查询会扫描整个表，使用 PARTITIONED BY 子句建表，查询就可以利用分区剪枝（input pruning）的特性
- Hive 当前的实现是，只有分区断言出现在离 FROM 子句最近的那个 WHERE 子句中，才会启用分区剪枝

## Join

- Hive 只支持等值连接（equality joins）、外连接（outer joins）和（left semi joins）。Hive 不支持所有非等值的连接，因为非等值连接非常难转化到 map/reduce 任务
- LEFT, RIGHT 和 FULL OUTER 关键字用于处理 join 中空记录的情况
- LEFT SEMI JOIN 是 IN/EXISTS 子查询的一种更高效的实现
- join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统
- 实践中，应该把最大的那个表写在最后

join 查询时，需要注意几个关键点

- 只支持等值 join

- SELECT a.\* FROM a JOIN b ON (a.id = b.id)
- SELECT a.\* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)
- 可以 join 多于 2 个表，例如

```
SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c
ON (c.key = b.key2)
```

如果join中多个表的 join key 是同一个，则 join 会被转化为单个 map/reduce 任务  
 LEFT, RIGHT和FULL OUTER   SELECT a.val, b.val FROM a LEFT OUTER  
 JOIN b ON (a.key=b.key)

- 如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写
- 容易混淆的问题是表分区的情况

```
SELECT c.val, d.val FROM c LEFT OUTER JOIN d ON (c.key=d.key) WHERE
a.ds='2010-07-07' AND b.ds='2010-07-07'
```

如果 d 表中找不到对应 c 表的记录，d 表的所有列都会列出 NULL，包括 ds 列。也就是说，join 会过滤 d 表中不能找到匹配 c 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关.解决办法

```
SELECT c.val, d.val FROM c LEFT OUTER JOIN d ON (c.key=d.key AND
d.ds='2009-07-07' AND c.ds='2009-07-07')
```

## LEFT SEMI JOIN

- LEFT SEMI JOIN 的限制是，JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行

```
SELECT a.key, a.value
FROM a
WHERE a.key in
(SELECT b.key
FROM B);
可以被重写为：
SELECT a.key, a.val
FROM a LEFT SEMI JOIN b on (a.key = b.key)
```

## UNION ALL

- 用来合并多个select的查询结果，需要保证select中字段须

```
select_statement UNION ALL select_statement UNION ALL
select_statement
```

## 传统SQL与HQL的差别

Hive 视图与一般数据库视图 Hive视图与一般数据库视图作用角色相同，都是基于数据规模缩减或者基于安全机制下的某些条件查询下的数据子集。Hive视图只支持逻辑视图，不支持物化视图，即每次对视图的查询hive都将执行查询任务，因此视图不会带来性能上的提升。作为Hive查询优化的一部分，对视图的查询条件语句和视图的定义查询条件语句将会尽可能的合并成一个条件查询。

Hive索引与一般数据库索引 相比于传统数据库，Hive只提供有限的索引功能，通过在某些字段上建立索引来加速某些操作。通常当逻辑分区太多太细，partition无法满足时，可以考虑建立索引。Hive1.2.1版本目前支持的索引类型有 CompactIndexHandler和Bitmap。

**CompactIndexHandler** 压缩索引 通过将列中相同的值得字段进行压缩从而减小存储和加快访问时间。需要注意的是Hive创建压缩索引时会将索引数据也存储在Hive表中。对于表tb\_index (id int, name string) 而言，建立索引后的索引表中默认的三列依次为索引列 (id) 、hdfs文件地址(\_bucketname)、偏移量(offset)。特别注意，offset列类型为array。

**Bitmap** 位图索引 作为一种常见的索引，如果索引列只有固定的几个值，那么就可以采用位图索引来加速查询。利用位图索引可以方便的进行AND/OR/XOR等各类计算，Hive0.8版本开始引入位图索引，位图索引在大数据处理方面的应用广泛，比如可以利用bitmap来计算用户留存率（索引做与运算，效率远好于join的方式）。如果Bitmap索引很稀疏，那么就需要对索引压缩以节省存储空间和加快IO。Hive的Bitmap Handler采用的是EWAH (<https://github.com/lemire/javaewah>) 压缩方式。

## Hive不支持等值连接

- SQL中对两表内联可以写成：

```
select * from dual a,dual b where a.key = b.key;
```

- Hive中应为

```
select * from dual a join dual b on a.key = b.key;
```

而不是传统的格式：

```
SELECT t1.a1 as c1, t2.b1 as c2FROM t1, t2 WHERE t1.a2 = t2.b2
```

### 分号字符

- 分号是SQL语句结束标记，在HiveQL中也是，但是在HiveQL中，对分号的识别没有那么智慧，例如：

```
select concat(key,concat(';',key)) from dual;
```
- 但HiveQL在解析语句时提示：

```
FAILED: Parse Error: line 0:-1 mismatched input '<EOF>' expecting )  
in function specification
```

- 解决的办法是，使用分号的八进制的ASCII码进行转义，那么上述语句应写成  

```
select concat(key,concat('\073',key)) from dual;
```

### IS [NOT] NULL

SQL中null代表空值，值得警惕的是，在HiveQL中String类型的字段若是空(empty)字符串，即长度为0，那么对它进行IS NULL的判断结果是False.

Hive不支持将数据插入现有的表或分区中，

仅支持覆盖重写整个表，示例如下：

```
INSERT OVERWRITE TABLE t1  
SELECT * FROM t2; INSERT OVERWRITE TABLE t1SELECT * FROM t2;
```

### hive不支持INSERT INTO, UPDATE, DELETE操作

这样的话，就不要很复杂的锁机制来读写数据。INSERT INTO syntax is only available starting in version 0.8。INSERT INTO就是在表或分区中追加数据。

# Hive的模式设计

## 一 概述

Hive看上去以及实际行为都像一个关系型数据库。用户对如表和列这类术语比较熟悉,而且Hive提供的查询语言和用户之前使用过的SQL方言非常相似。不过Hive实现和使用的方式和传统的关系型数据库是非常不同的。通常,用户试图移植关系型数据库中的模式,而事实上Hive是反模式。

## 1 按天划分的表

按天划分表就是一种模式,其通常会在表中加入一个时间戳,例如表名为 supply\_2011\_01\_01等等。这种每天一张表的方式在数据库领域是反模式的一种方式,但是因为实际情况下数据集增长得很快,这种方式应用还是比较广泛的。

```
0: jdbc:hive2://hadoopmaster:10000/> CREATE TABLE supply_2011_01
_02(id int,part string,quantity int);
OK
No rows affected (1.279 seconds)
0: jdbc:hive2://hadoopmaster:10000/> CREATE TABLE supply_2011_01
_03(id int,part string,quantity int);
OK
No rows affected (0.055 seconds)
0: jdbc:hive2://hadoopmaster:10000/> CREATE TABLE supply_2011_01
_04(id int,part string,quantity int);
OK
No rows affected (0.056 seconds)
0: jdbc:hive2://hadoopmaster:10000/>

0: jdbc:hive2://hadoopmaster:10000/> select part,quantity supply
_2011_01_02 from supply_2011_01_02
. > union all
. > select part,quantity supply
_2011_01_02 from supply_2011_01_03
. > where quantity<4;
```

对于Hive,这种情况下应该使用分区表.Hive通过Where子句中的表达式来选择查询所需要的指定分区,这样的查询执行效率高,而且看起来清晰明了:

```
0: jdbc:hive2://hadoopmaster:10000/> CREATE TABLE supplybypartition
  (id int,part string,quantity int)
  . . . . . . . . . . . . . . . . . > partitioned by (day int);

0: jdbc:hive2://hadoopmaster:10000/> alter table supplybypartition
  add partition(day=20110102);
OK
No rows affected (0.088 seconds)
0: jdbc:hive2://hadoopmaster:10000/> alter table supplybypartition
  add partition(day=20110103);
OK
No rows affected (0.067 seconds)
0: jdbc:hive2://hadoopmaster:10000/> alter table supplybypartition
  add partition(day=20110104);
OK
No rows affected (0.083 seconds)

0: jdbc:hive2://hadoopmaster:10000/> select * from supplybypartition
  . . . . . . . . . . . . . . . . . > where day>=20110102 and day
<20110103 and quantity<4;
OK
+-----+-----+-----+
| supplybypartition.id | supplybypartition.part | supplybypartition.quantity | supplybypartition.day |
+-----+-----+-----+
|                               |                         |                         |                         |
+-----+-----+-----+
|                               |                         |                         |                         |
+-----+-----+-----+
|                               |                         |                         |                         |
+-----+-----+-----+
|                               |                         |                         |                         |
+-----+-----+-----+
No rows selected (0.162 seconds)
0: jdbc:hive2://hadoopmaster:10000/>
```

## 2 关于分区

Hive中分区的功能是非常有用的,这是因为Hive通常要对输入进行全盘扫描,来满足查询条件,通过创建很多的分区确定可以优化一些查询,但是同时可能会对其他一些重要的查询不利:

HDFS用于设计存储数百万的大文件,而非数十亿的小文件.使用过多分区可能导致的一个问题就是会创建大量的非必须的hadoop文件和文件夹.一个分区就对应着一个包含有多个文件的文件夹.如果指定的表存在数百个分区,那么可能每天都会创建好几万个文件.如果保持这样的表很多年,那么最终就会超出NameNode对系统云数据信息的处理能力.因为NameNode必须将所有系统文件的元数据保存在内存中.虽然每个文件只需要少量字节大小的元数据(大约是150字节/文件),但是这样也会限制一个HDFS实例所能管理的文件总数的上限.而其他的文件系统,比如MapR和Amazon S3就没有这个限制.

MapReduce会将一个任务(job)转换成多个任务(task).默认情况下,每个task都是一个新的JVM实例,都需要开启和销毁的开销.对于小文件,每个文件都会对应一个task.在一些情况下,JVM开启和销毁的时间中销毁可能会比实际处理数据的时间消耗要长

因此,一个理想的分区方案不应该导致产生太多的分区和文件夹目录,并且每个目录下的文件应该足够大,应该是文件系统中块大小的若干倍.

接时间范围进行分区的一个好的策略就是按照不同的时间颗粒度来确定合适大小的数据积累量,而且安装这个时间颗粒.随着时间的推移,分区数量的增长是均匀的,而且每个分区下包含的文件大小至少是文件系统中块或块大小的数倍.

如果用户找不到好的,大小相对合适的分区方式的话,我们可以考虑使用分桶表来解决问题

### 3 关于分桶表数据存储

## 二 事务

### 1 建表

```

hive> create table test_trancaction
  > (user_id Int, name String)
  > clustered by (user_id) into 3 buckets
  > stored as orc TBLPROPERTIES ('transactional'='true');
OK
Time taken: 0.813 seconds
hive> create table test_insert_test(id int, name string) row form
at delimited fields TERMINATED BY ',';
OK
Time taken: 0.11 seconds

```

## 2 导入数据

```

hive> insert into test_insert_test values(3, "ma");

hive> delete from test_insert_test where id=1;
FAILED: SemanticException [Error 10294]: Attempt to do update or
      delete using transaction manager that does not support these op
erations.

```

修改配置文件**hive-site.xml**

```

<!--start for trancaction -->

<property>

  <name>hive.support.concurrency</name>

  <value>true</value>

</property>

<property>

  <name>hive.enforce.bucketing</name>

  <value>true</value>

```

```
</property>

<property>
    <name>hive.exec.dynamic.partition.mode</name>
    <value>nonstrict</value>
</property>

<property>
    <name>hive.txn.manager</name>
    <value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>
</property>

<property>
    <name>hive.compactor.initiator.on</name>
    <value>true</value>
</property>

<property>
    <name>hive.compactor.worker.threads</name>
    <value>1</value>
</property>
```

### 查看分桶

```
hadoop@hadoopmaster:/usr/local/hive/conf$ hdfs dfs -ls /user/hiv
```

```
e/warehouse/test_insert_test
Found 3 items
-rwxrwxr-x  2 hadoop supergroup      6 2016-08-10 10:39 /us
er/hive/warehouse/test_insert_test/000000_0
-rwxrwxr-x  2 hadoop supergroup      5 2016-08-10 10:40 /us
er/hive/warehouse/test_insert_test/000000_0_copy_1
-rwxrwxr-x  2 hadoop supergroup      5 2016-08-10 10:40 /us
er/hive/warehouse/test_insert_test/000000_0_copy_2

hive> hadoop@hadoopmaster:/usr/local/hive/conf$ hdfs dfs -ls /us
er/hive/warehouse/test_trancaction
Found 3 items
drwxr-xr-x  - hadoop supergroup      0 2016-08-10 10:45 /us
er/hive/warehouse/test_trancaction/delta_0000001_0000001_0000
drwxr-xr-x  - hadoop supergroup      0 2016-08-10 10:46 /us
er/hive/warehouse/test_trancaction/delta_0000002_0000002_0000
drwxr-xr-x  - hadoop supergroup      0 2016-08-10 10:46 /us
er/hive/warehouse/test_trancaction/delta_0000003_0000003_0000

hive> delete from test_trancaction where user_id=1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be avail
able in the future versions. Consider using a different executio
n engine (i.e. tez, spark) or using Hive 1.X releases.
Query ID = hadoop_20160810104829_0e78e0cd-2bc9-4741-89c1-7a8d1f3
84682
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 3
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1470228460967_0010, Tracking URL = http://had
oopmaster:8088/proxy/application_1470228460967_0010/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_14702
28460967_0010
Hadoop job information for Stage-1: number of mappers: 3; number
```

```
of reducers: 3
2016-08-10 10:48:36,463 Stage-1 map = 0%,  reduce = 0%
2016-08-10 10:48:41,784 Stage-1 map = 33%,  reduce = 0%, Cumulative CPU 0.97 sec
2016-08-10 10:48:46,913 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 2.0 sec
2016-08-10 10:48:48,970 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.0 sec
2016-08-10 10:48:50,020 Stage-1 map = 100%,  reduce = 33%, Cumulative CPU 4.1 sec
2016-08-10 10:48:54,117 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.76 sec
MapReduce Total cumulative CPU time: 5 seconds 760 msec
Ended Job = job_1470228460967_0010
Loading data to table default.test_trancaction
MapReduce Jobs Launched:
Stage-Stage-1: Map: 3  Reduce: 3  Cumulative CPU: 5.76 sec  HD
FS Read: 32745 HDFS Write: 701 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 760 msec
OK
Time taken: 26.074 seconds
```

最后总结一下,做**Hive**的**Transaction**其实不合适,资源耗用量大,意义不大,本身**hive**做离线查询还是可以的.**ACID**支持你饶了我吧....二点 需要分桶表 需要修改**hive-site.xml**文件,剩余的还是很简单的.

## 参考链接

[ACID and Transactions in Hive](#)

# Hive综合案例实战

## 一 数据源的准备工作

首先我们去一个网站下载相关的数据，之后通过hive导入进行实验。<http://grouplens.org/>

**MovieLens**

GroupLens Research has collected and made available rating data sets from the MovieLens web site (<http://movielens.org>). The data sets were collected over various periods of time, depending on the size of the set. Before using these data sets, please review their README files for the usage licenses and other details.

Help our research lab: Please [take a short survey](#) about the MovieLens datasets

**MovieLens 100K Dataset**  
Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.  

- [README.txt](#)
- [ml-100k.zip](#) (size: 5 MB, [checksum](#))
- [Index of unzipped files](#)

Permalink: <http://grouplens.org/datasets/movielens/100k/>

**MovieLens 1M Dataset**  
Stable benchmark dataset. 1 million ratings from 6000 users on 4000 movies. Released 2/2003.  

- [README.txt](#)
- [ml-1m.zip](#) (size: 6 MB, [checksum](#))

Permalink: <http://grouplens.org/datasets/movielens/1m/>

**MovieLens 10M Dataset**

**Datasets**

- MovieLens**
- [HetRec 2011](#)
- [WikiLens](#)
- [Book-Crossing](#)
- [Jester](#)
- [EachMovie](#)

## 二 内部表

### 1 创建内部表并载入数据

```

hadoop@hadoopmaster:~$ beeline -u jdbc:hive2://hadoopmaster:1000
0/
Beeline version 2.1.0 by Apache Hive
0: jdbc:hive2://hadoopmaster:10000/> show databases;
OK
+-----+---+

```

```

| database_name |
+-----+---+
| default      |
| fincials     |
+-----+---+
2 rows selected (1.038 seconds)
0: jdbc:hive2://hadoopmaster:10000/> use default;
OK
No rows affected (0.034 seconds)
0: jdbc:hive2://hadoopmaster:10000/> create table u_data (userid
    INT, movieid INT, rating INT, unixtime STRING) row format delimited
    fields terminated by '\t' lines terminated by '\n';
OK
No rows affected (0.242 seconds)
0: jdbc:hive2://hadoopmaster:10000/> LOAD DATA LOCAL INPATH '/home/hadoop/u.data'
OVERWRITE INTO TABLE u_data;
Loading data to table default.u_data
OK
No rows affected (0.351 seconds)
0: jdbc:hive2://hadoopmaster:10000/> select * from u_data;
OK
+-----+-----+-----+-----+
| u_data.userid | u_data.movieid | u_data.rating | u_data.unixtime |
+-----+-----+-----+-----+
| 196          | 242          | 3           | 881250949
|
| 186          | 302          | 3           | 891717742
|
| 22           | 377          | 1           | 878887116
|
| 244          | 51           | 2           | 880606923
|
| 166          | 346          | 1           | 886397596
|
| 298          | 474          | 4           | 884182806
|
| 115          | 265          | 2           | 881171488

```

253	465	5	891628467
305	451	3	886324817
6	86	3	883603013
62	257	2	879372434
286	1014	5	879781125

## 2 查看占用的HDFS空间

```

hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse/u_data
Found 1 items
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:19 /user/hive/warehouse/u_data/u.data

```

## 3 写脚本反复导入100次

先查看以前有多少行

```

0: jdbc:hive2://hadoopmaster:10000/> select count(*) from u_data
;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
Query ID = hadoop_20160722102853_77aa1bc6-79c2-4916-9b07-a763d112ef41
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>

```

```
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1468978056881_0003, Tracking URL = http://had
oopmaster:8088/proxy/application_1468978056881_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_14689
78056881_0003
Hadoop job information for Stage-1: number of mappers: 1; number
of reducers: 1
2016-07-22 10:28:58,786 Stage-1 map = 0%, reduce = 0%
2016-07-22 10:29:03,890 Stage-1 map = 100%, reduce = 0%, Cumula
tive CPU 0.89 sec
2016-07-22 10:29:10,005 Stage-1 map = 100%, reduce = 100%, Cumu
lative CPU 1.71 sec
MapReduce Total cumulative CPU time: 1 seconds 710 msec
Ended Job = job_1468978056881_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.71 sec   HD
FS Read: 1987050 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 710 msec
OK
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be avail
able in the future versions. Consider using a different executio
n engine (i.e. tez, spark) or using Hive 1.X releases.
+-----+---+
|   c0   |
+-----+---+
| 100000  |
+-----+---+
1 row selected (17.757 seconds)
```

hive用Mapreduce引擎计算真心在速度上不行,10W用了17秒,比关系型数据库差不少,还是要用Spark呀

再我们需要了解如何用**hive**中的一次命令,我们可以这样用.

```

hadoop@hadoopmaster:~$ hive -e "LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE u_data;"

Loading data to table default.u_data
OK
Time taken: 1.239 seconds

```

最后写脚本

```

#!/bin/bash
for (( c=1; c<=10; c++ ))
do
    echo "正在写入第 $c 次数据..."
    hive -e "LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE u_data;"
    wait
done

```

插入完,检查查询成本

```

0: jdbc:hive2://hadoopmaster:10000/> select count(*) from u_data;
;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
Query ID = hadoop_20160722104633_18c3467d-9263-4785-8714-1570fc3bb9ae
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1468978056881_0009, Tracking URL = http://hadoopmaster:8088/proxy/application_1468978056881_0009/

```

```

Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_14689
78056881_0009
Hadoop job information for Stage-1: number of mappers: 1; number
of reducers: 1
2016-07-22 10:46:39,037 Stage-1 map = 0%, reduce = 0%
2016-07-22 10:46:46,190 Stage-1 map = 100%, reduce = 0%, Cumula
tive CPU 1.82 sec
2016-07-22 10:46:52,310 Stage-1 map = 100%, reduce = 100%, Cumu
lative CPU 2.67 sec
MapReduce Total cumulative CPU time: 2 seconds 670 msec
Ended Job = job_1468978056881_0009
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.67 sec HD
FS Read: 77198770 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 670 msec
OK
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be avail
able in the future versions. Consider using a different executio
n engine (i.e. tez, spark) or using Hive 1.X releases.
+-----+---+
|      c0      |
+-----+---+
| 3900000  |
+-----+---+
1 row selected (20.173 seconds)

```

用了20秒,看起来Mapreduce的启动成本确实有点高了

```

hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse/u_data
Found 39 items
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:37 /us
er/hive/warehouse/u_data/u.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:38 /us
er/hive/warehouse/u_data/u_copy_1.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:40 /us
er/hive/warehouse/u_data/u_copy_10.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:40 /us
er/hive/warehouse/u_data/u_copy_11.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:41 /us
er/hive/warehouse/u_data/u_copy_12.data

```

```
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:42 /user/hive/warehouse/u_data/u_copy_13.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:42 /user/hive/warehouse/u_data/u_copy_14.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:42 /user/hive/warehouse/u_data/u_copy_15.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:42 /user/hive/warehouse/u_data/u_copy_16.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_17.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_18.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_19.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_2.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_20.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_21.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_22.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:43 /user/hive/warehouse/u_data/u_copy_23.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_24.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_25.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_26.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_27.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_28.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:44 /user/hive/warehouse/u_data/u_copy_29.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_3.data
-rwxrwxr-x 2 hadoop supergroup 1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_30.data
```

```
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_31.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_32.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_33.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_34.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_35.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:45 /user/hive/warehouse/u_data/u_copy_36.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:46 /user/hive/warehouse/u_data/u_copy_37.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:46 /user/hive/warehouse/u_data/u_copy_38.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_4.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_5.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_6.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_7.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:39 /user/hive/warehouse/u_data/u_copy_8.data
-rwxrwxr-x  2 hadoop supergroup  1979173 2016-07-22 10:40 /user/hive/warehouse/u_data/u_copy_9.data
```

## 三 外部表

### 1 创建外部表并载入数据

```
0: jdbc:hive2://hadoopmaster:10000/> create external table u_da
ta_external_table (userid INT, movieid INT, rating INT, unixtime STRING) row format delimited fields terminated by '\t' lines terminated by '\n';
OK
No rows affected (0.047 seconds)

0: jdbc:hive2://hadoopmaster:10000/> show tables;
OK
+-----+---+
| tab_name |
+-----+---+
| employees |
| t_hive |
| t_hive2 |
| u_data |
| u_data_external_table |
+-----+---+
5 rows selected (0.036 seconds)
```

## 2 导入数据

```
hive -e "LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE
u_data;"
```

## 3 内部表与外部表区别

我用drop table 命令删除刚才创建的二张表，一个内表一个外表之后结果是。

```
hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse/
Found 5 items
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 17:25 /us
er/hive/warehouse/employees
drwxrwxr-x  - hadoop supergroup          0 2016-07-21 15:52 /us
er/hive/warehouse/fincials.db
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 09:50 /us
er/hive/warehouse/t_hive
drwxrwxr-x  - hadoop supergroup          0 2016-07-20 09:54 /us
er/hive/warehouse/t_hive2
drwxrwxr-x  - hadoop supergroup          0 2016-07-22 11:04 /us
er/hive/warehouse/u_data_external_table
```

内表的数据完全删除，而外表还有

最后归纳一下Hive中表与外部表的区别：

- 在导入数据到外部表，数据并没有移动到自己的数据仓库目录下，也就是说外部表中的数据并不是由它自己来管理的！而表则不一样；
- 在删除表的时候，Hive将会把属于表的元数据和数据全部删掉；而删除外部表的时候，Hive仅仅删除外部表的元数据，数据是不会删除的！那么，应该如何选择使用哪种表呢？在大多数情况没有太多的区别，因此选择只是个人喜好的问题。但是作为一个经验，如果所有处理都需要由Hive完成，那么你应该创建表，否则使用外部表！

## 四 分区表

```
0: jdbc:hive2://hadoopmaster:10000/> create table u_data_partitioned_table (userid INT, movieid INT, rating INT, unixtime STRING) partitioned by(day int) row format delimited fields terminated by '\t' lines terminated by '\n';
OK
No rows affected (0.256 seconds)
0: jdbc:hive2://hadoopmaster:10000/>
```

```

0: jdbc:hive2://hadoopmaster:10000/> LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE u_data_partitioned_table partition(day=20160101);
Loading data to table default.u_data_partitioned_table partition
(day=20160101)
OK
No rows affected (0.424 seconds)
0: jdbc:hive2://hadoopmaster:10000/>

100,000 rows selected (4.653 seconds)
0: jdbc:hive2://hadoopmaster:10000/> LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE u_data_partitioned_table partition(day=20160101);
Loading data to table default.u_data_partitioned_table partition
(day=20160101)
OK
No rows affected (0.424 seconds)
0: jdbc:hive2://hadoopmaster:10000/> LOAD DATA LOCAL INPATH '/home/hadoop/u.data' INTO TABLE u_data_partitioned_table partition(day=20160102);
Loading data to table default.u_data_partitioned_table partition
(day=20160102)
OK
No rows affected (0.499 seconds)
0: jdbc:hive2://hadoopmaster:10000/>

hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse/u_data_
partitioned_table
Found 2 items
drwxrwxr-x  - hadoop supergroup          0 2016-07-22 13:51 /us
er/hive/warehouse/u_data_partitioned_table/day=20160101
drwxrwxr-x  - hadoop supergroup          0 2016-07-22 13:51 /us
er/hive/warehouse/u_data_partitioned_table/day=20160102

```

## 五 分桶表

```

0: jdbc:hive2://hadoopmaster:10000/> CREATE TABLE bucketed_data_
user (userid INT, movieid INT, rating INT, unixtime STRING) CLUS

```

```

TERED BY (userid) INTO 4 BUCKETS row format delimited fields ter
minated by '\t' lines terminated by '\n';
OK
No rows affected (0.045 seconds)
0: jdbc:hive2://hadoopmaster:10000/>

0: jdbc:hive2://hadoopmaster:10000/> insert overwrite table buck
eted_data_user select userid,movieid,rating,unixtime from u_data
_partitioned_table;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be avail
able in the future versions. Consider using a different executio
n engine (i.e. tez, spark) or using Hive 1.X releases.
Query ID = hadoop_20160722140142_c272bc07-b74d-4b5b-9689-0bec2ce
71780
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1468978056881_0010, Tracking URL = http://had
oopmaster:8088/proxy/application_1468978056881_0010/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_14689
78056881_0010
Hadoop job information for Stage-1: number of mappers: 1; number
of reducers: 4
2016-07-22 14:01:48,774 Stage-1 map = 0%,  reduce = 0%
2016-07-22 14:01:55,978 Stage-1 map = 100%,  reduce = 0%, Cumula
tive CPU 1.89 sec
2016-07-22 14:02:06,236 Stage-1 map = 100%,  reduce = 50%, Cumul
ative CPU 5.66 sec
2016-07-22 14:02:07,272 Stage-1 map = 100%,  reduce = 100%, Cumu
lative CPU 9.43 sec
MapReduce Total cumulative CPU time: 9 seconds 430 msec
Ended Job = job_1468978056881_0010
Loading data to table default.bucketed_data_user
MapReduce Jobs Launched:
```

```

Stage-Stage-1: Map: 1 Reduce: 4 Cumulative CPU: 9.43 sec   HD
FS Read: 5959693 HDFS Write: 5937879 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 430 msec
OK
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
No rows affected (26.251 seconds)
0: jdbc:hive2://hadoopmaster:10000/>

0: jdbc:hive2://hadoopmaster:10000/> select count(*) from bucketed_data_user ;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
Query ID = hadoop_20160722141056_eaf582be-4107-403a-bacd-0a18f567f576
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1468978056881_0012, Tracking URL = http://hadoopmaster:8088/proxy/application_1468978056881_0012/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1468978056881_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-07-22 14:11:04,156 Stage-1 map = 0%,  reduce = 0%
2016-07-22 14:11:09,331 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2016-07-22 14:11:15,488 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.78 sec
MapReduce Total cumulative CPU time: 1 seconds 780 msec
Ended Job = job_1468978056881_0012
MapReduce Jobs Launched:
```

```
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 1.78 sec   HD
FS Read: 5945855 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 780 msec
OK
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
+-----+---+
|    c0    |
+-----+---+
| 300000  |
+-----+---+
1 row selected (20.397 seconds)
0: jdbc:hive2://hadoopmaster:10000/>
```

```
hadoop@hadoopmaster:~$ hdfs dfs -ls /user/hive/warehouse/bucketed_data_user
Found 4 items
-rwxrwxr-x  2 hadoop supergroup  1400994 2016-07-22 14:02 /user/hive/warehouse/bucketed_data_user/000000_0
-rwxrwxr-x  2 hadoop supergroup  1493856 2016-07-22 14:02 /user/hive/warehouse/bucketed_data_user/000001_0
-rwxrwxr-x  2 hadoop supergroup  1566738 2016-07-22 14:02 /user/hive/warehouse/bucketed_data_user/000002_0
-rwxrwxr-x  2 hadoop supergroup  1475931 2016-07-22 14:02 /user/hive/warehouse/bucketed_data_user/000003_0
```

# Hive的开发

## 一 Hive的Thrift服务

Hive具有一个可选的组件叫做HiveServer或者HiveThrift,其允许通过指定的端口访问Hive,Thrift是一个软件框架,其用于跨语言的服务开发.关于Thrift,可以通过链接<http://thrift.apache.org>获取更详细的介绍.Thrift允许客户端使用包括Java C++ Ruby和其他语言,通过编程的方式远程访问Hive.

访问Hive的最常用的方式就是通过CLI进行访问,不过CLI的设计使其不便于通过编程的方式进行访问.CLI是胖客户端,其需要本地具有所有的Hive组件,包括配置,同时还需要一个Hadoop客户端及其配置.同时,其可作为HDFS客户端,Mapreduce客户端以及JDBC客户端进行使用.

### 1 启动Thrift Server

如果想启动Hiveserver,可以在后台启动执行这个Hive服务:

```
hadoop@hadoopmaster:~$ hiveserver2 start &
```

检查HiveServer是否启动成功的最快捷方法就是使用netstat命令查看10000端口是否打开并监听连接:

```
$netstat -nl|grep 10000
```

正如前面所提到过的,HiveServer使用Thrift提供服务,Thrift提供了一个接口语言.通过这些接口,Thrift编译器可以产生创建网络RPC的多种语言的客户端的代码.

## 二 一个简单的链接例子

### 1 Maven地址

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
ttp://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hithinksoft.com</groupId>
    <artifactId>chu888chu888</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.apache.hive/hive
-jdbc -->
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-jdbc</artifactId>
            <version>2.1.0</version>
        </dependency>
    </dependencies>
    <repositories>
        <repository>
            <id>jboss</id>
            <url>http://maven.aliyun.com/nexus/content/groups/pu
blic/</url>
        </repository>
    </repositories>
</project>
```

例子

```
import java.sql.*;

/**
 * Created by chuguangming on 16/7/26.
 */
public class testHive {
    /**
     * @param args
     * @throws SQLException
     */
    public static void main(String[] args) throws ClassNotFoundException {
        Class.forName("org.apache.hive.jdbc.HiveDriver");
        try{
            Connection con = DriverManager.getConnection("jdbc:hive2://hadoopmaster:10000/default","hive","hive");
            PreparedStatement sta = con.prepareStatement("select * from u_data_partitioned_table");
            ResultSet result = sta.executeQuery();
            while(result.next()){
                System.out.println(result.getString(1));
            }
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

# Hive的安全

## 一 概述

在了解Hive的安全机制之前,我们需要首先清楚Hadoop的安全机制以及Hadoop的历史,Hadoop起源于Apache Nutch的子项目.在那个时代以及整个早期原型时代,功能性需要比安全性需求优先级要高.分布式系统的安全问题要比正常情况下更加复杂,因为不同机器上的多个组件需要相互进行通信.

Hadoop的安全性近期有了许多变化,其中主要是对Kerberos安全认证的支持,还包括其他一些问题的修复.Kerberos允许客户端和服务器端相互认证.客户端的每次请求中都会带有凭证(ticket)信息.在TaskTracker上执行的任务(task)都是由执行任务(job)的用户来执行的.用户无法通过设置hadoop.job.ugi属性的值来模拟其他人来执行任务.为了达到这个目的,所有的Hadoop组件从头到尾都要使用kerberos安全认证.

Hive在Hadoop引入Kerberos支持之前就已经存在了,而且Hive目前还没有完全和Hadoop的安全改变相融合.例如,Hive元数据存储链接可能是直接连接到一个JDBC数据库或者通过Thrift进行链接,这些都是要用用户身份进行各种操作.像HiveService这样的基于Thrift的组件还是要冒充他人来执行.Hadoop的文件用户权限模型(也就是对于一个文件分为用户组和其他3层权限)和很多其他数据库中用户权限模型具有很大的差异,数据库中通常是对使用字段级别进行授权和权限回收操作来进行权限控制的.

## 5 FAQ

### FAQ

#### 调试中出现的**Jline**版本过低的**FAQ**

```
Logging initialized using configuration in jar:file:/hive/apache-hive-1.1.0-bin/lib/hive-common-1.1.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/hadoop-2.5.2/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/hive/apache-hive-1.1.0-bin/lib/hive-jdbc-1.1.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[ERROR] Terminal initialization failed; falling back to unsupported
java.lang.IncompatibleClassChangeError: Found class jline.Terminal, but interface was expected
        at jline.TerminalFactory.create(TerminalFactory.java:101)
        at jline.TerminalFactory.get(TerminalFactory.java:158)
        at jline.console.ConsoleReader.<init>(ConsoleReader.java:229)
        at jline.console.ConsoleReader.<init>(ConsoleReader.java:221)
        at jline.console.ConsoleReader.<init>(ConsoleReader.java:209)
        at org.apache.hadoop.hive.cli.CliDriver.getConsoleReader(CliDriver.java:773)
        at org.apache.hadoop.hive.cli.CliDriver.executeDriver(CliDriver.java:715)
```

```

        at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:675)
        at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:615)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:212)

```

原因是hadoop目录下存在老版本jline：

```

/hadoop-2.5.2/share/hadoop/yarn/lib:
-rw-r--r-- 1 root root 87325 Mar 10 18:10 jline-0.9.94.jar

```

解决方法是：

将hive下的新版本jline的JAR包拷贝到hadoop下：

```
cp /hive/apache-hive-1.1.0-bin/lib/jline-2.12.jar ./
```

```
/hadoop-2.5.2/share/hadoop/yarn/lib:
```

```
-rw-r--r-- 1 root root 87325 Mar 10 18:10 jline-0.9.94.jar.bak
-rw-r--r-- 1 root root 213854 Mar 11 22:22 jline-2.12.jar
```

hive cli启动成功：

```
root@ubuntu:/hive# hive
```

```
Logging initialized using configuration in jar:file:/hive/apache-
```

```
-hive-1.1.0-bin/lib/hive-common-1.1.0.jar!/hive-log4j.properties
```

```
SLF4J: Class path contains multiple SLF4J bindings.
```

```
SLF4J: Found binding in [jar:file:/hadoop-2.5.2/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
```

```
SLF4J: Found binding in [jar:file:/hive/apache-hive-1.1.0-bin/lib/hive-jdbc-1.1.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
```

```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
```

```
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFact
```

```
ory]
hive>
```

## 调试中出现**java.io.tmpdir**目录的问题

异常详情如下：

```
Exception in thread "main" java.lang.RuntimeException: java.lang
.IllegalArgumentException: java.net.URISyntaxException: Relative
path in absolute URI: ${system:java.io.tmpdir%7D/$%7Bsystem:use
r.name%7D
        at org.apache.hadoop.hive.ql.session.SessionState.start(
SessionState.java:444)
        at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.ja
va:672)
        at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.j
ava:616)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native M
ethod)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMet
hodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Deleg
atingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:160)
Caused by: java.lang.IllegalArgumentException: java.net.URISyntaxException: Relative path in absolute URI: ${system:java.io.tmpd
ir%7D/$%7Bsystem:user.name%7D
        at org.apache.hadoop.fs.Path.initialize(Path.java:148)
        at org.apache.hadoop.fs.Path.<init>(Path.java:126)
        at org.apache.hadoop.hive.ql.session.SessionState.create
SessionDirs(SessionState.java:487)
        at org.apache.hadoop.hive.ql.session.SessionState.start(
SessionState.java:430)
        ... 7 more
Caused by: java.net.URISyntaxException: Relative path in absolut
e URI: ${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D
        at java.net.URI.checkPath(URI.java:1804)
        at java.net.URI.<init>(URI.java:752)
```

```
at org.apache.hadoop.fs.Path.initialize(Path.java:145)
... 10 more
```

解决方案如下：

1. 查看hive-site.xml配置，会看到配置值含有"system:java.io.tmpdir"的配置项
  2. 新建文件夹/home/grid/hive-0.14.0-bin/iotmp
  3. 将含有"system:java.io.tmpdir"的配置项的值修改为如上地址
- 启动hive，成功！

## hive内存不够用的问题

```
hive> select * from t_test where ds=20150323 limit 2;
OK
Exception in thread "main" java.lang.OutOfMemoryError: Java heap
space
```

问题原因： hive堆内存默认为256M

这个问题的解决方法为：

```
修改/usr/lib/hive/bin/hive-config.sh文件 中
# Default to use 256MB
export HADOOP_HEAPSIZE=${HADOOP_HEAPSIZE:-256}
将上面256调大就行
```

## 用户不对的错误

编写JDBC客户端程序连接hive时，出现报错：

```
org.apache.hive.service.cli.HiveSQLException: Failed to open new
session: java.lang.RuntimeException: org.apache.hadoop.ipc.Remo
teException(org.apache.hadoop.security.authorize.AuthorizationEx
ception): User: hadoop is not allowed to impersonate anonymous
```

具体出错信息

```
Exception in thread "main" org.apache.hive.service.cli.HiveSQLException: Failed to open new session: java.lang.RuntimeException: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.AuthorizationException): User: hadoop is not allowed to impersonate anonymous
    at org.apache.hive.jdbc.Utils.verifySuccess(Utils.java:258)
    at org.apache.hive.jdbc.Utils.verifySuccess(Utils.java:249)
    at org.apache.hive.jdbc.HiveConnection.openSession(HiveConnection.java:579)
    at org.apache.hive.jdbc.HiveConnection.<init>(HiveConnection.java:167)
    at org.apache.hive.jdbc.HiveDriver.connect(HiveDriver.java:107)
    at java.sql.DriverManager.getConnection(DriverManager.java:571)
    at java.sql.DriverManager.getConnection(DriverManager.java:215)
    at client.main(client.java:21)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:606)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:140)
Caused by: org.apache.hive.service.cli.HiveSQLException: Failed to open new session: java.lang.RuntimeException: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.AuthorizationException): User: hadoop is not allowed to impersonate anonymous
    at org.apache.hive.service.cli.session.SessionManager.openSession(SessionManager.java:324)
    at org.apache.hive.service.cli.CLIService.openSessionWithImpersonation(CLIService.java:187)
    at org.apache.hive.service.cli.thrift.ThriftCLIService.getSessionHandle(ThriftCLIService.java:424)
    at org.apache.hive.service.cli.thrift.ThriftCLIService.OpenSession(ThriftCLIService.java:318)
```

```
        at org.apache.hive.service.cli.thrift.TCLIService$Processor$OpenSession.getResult(TCLIService.java:1257)
        at org.apache.hive.service.cli.thrift.TCLIService$Processor$OpenSession.getResult(TCLIService.java:1242)
        at org.apache.thrift.ProcessFunction.process(ProcessFunction.java:39)
        at org.apache.thrift.TBaseProcessor.process(TBaseProcessor.java:39)
        at org.apache.hive.service.auth.TSetIpAddressProcessor.process(TSetIpAddressProcessor.java:56)
        at org.apache.thrift.server.TThreadPoolServer$WorkerProcess.run(TThreadPoolServer.java:286)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.lang.RuntimeException: java.lang.RuntimeException: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.AuthorizationException): User: hadoop is not allowed to impersonate anonymous
        at org.apache.hive.service.cli.session.HiveSessionProxy.invoke(HiveSessionProxy.java:89)
        at org.apache.hive.service.cli.session.HiveSessionProxy.access$000(HiveSessionProxy.java:36)
        at org.apache.hive.service.cli.session.HiveSessionProxy$1.run(HiveSessionProxy.java:63)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1657)
        at org.apache.hive.service.cli.session.HiveSessionProxy.invoke(HiveSessionProxy.java:59)
        at com.sun.proxy.$Proxy35.open(Unknown Source)
        at org.apache.hive.service.cli.session.SessionManager.openSession(SessionManager.java:315)
        ... 12 more
Caused by: java.lang.RuntimeException: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.AuthorizationE
```

```
xception): User: hadoop is not allowed to impersonate anonymous
      at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:554)
      at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:489)
      at org.apache.hive.service.cli.session.HiveSessionImpl.open(HiveSessionImpl.java:156)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccesso
rImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
gMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:497)
      at org.apache.hive.service.cli.session.HiveSessionProxy.invo
ke(HiveSessionProxy.java:78)
      ... 20 more
Caused by: java.lang.RuntimeException: org.apache.hadoop.ipc.Rem
oteException:User: hadoop is not allowed to impersonate anonymou
s
      at org.apache.hadoop.ipc.Client.call(Client.java:1476)
      at org.apache.hadoop.ipc.Client.call(Client.java:1407)
      at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.invoke(Pr
otobufRpcEngine.java:229)
      at com.sun.proxy.$Proxy30.getFileInfo(Unknown Source)
      at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolT
ranslatorPB.getFileInfo(ClientNamenodeProtocolTranslatorPB.java:
771)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodA
ccessorImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
gMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:497)
      at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeM
ethod(RetryInvocationHandler.java:187)
      at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(
RetryInvocationHandler.java:102)
      at com.sun.proxy.$Proxy31.getFileInfo(Unknown Source)
```

```

    at org.apache.hadoop.hdfs.DFSClient.getFileInfo(DFSClient.java:2116)
    at org.apache.hadoop.hdfs.DistributedFileSystem$22.doCall(DistributedFileSystem.java:1305)
    at org.apache.hadoop.hdfs.DistributedFileSystem$22.doCall(DistributedFileSystem.java:1301)
    at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
    at org.apache.hadoop.hdfs.DistributedFileSystem.getFileStatuses(DistributedFileSystem.java:1301)
    at org.apache.hadoop.fs.FileSystem.exists(FileSystem.java:1424)
    at org.apache.hadoop.hive.ql.session.SessionState.createRootHDFSDir(SessionState.java:639)
    at org.apache.hadoop.hive.ql.session.SessionState.createSessionDirs(SessionState.java:597)
    at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:526)
    ... 27 more

```

从最终的错误信息来看：User: hadoop is not allowed to impersonate anonymous，意思是用户hadoop不允许伪装成anonymous（hive的默认用户，默认配置可以查看）。

### 解决方案

```

<property>
    <name>hadoop.proxyuser.hadoop.groups</name>
    <value>hadoop</value>
    <description>Allow the superuser oozie to impersonate any members of the group group1 and group2</description>
</property>

<property>
    <name>hadoop.proxyuser.hadoop.hosts</name>
    <value>192.168.21.222,127.0.0.1,localhost</value>
    <description>The superuser can connect only from host1 and host2 to impersonate a user</description>
</property>

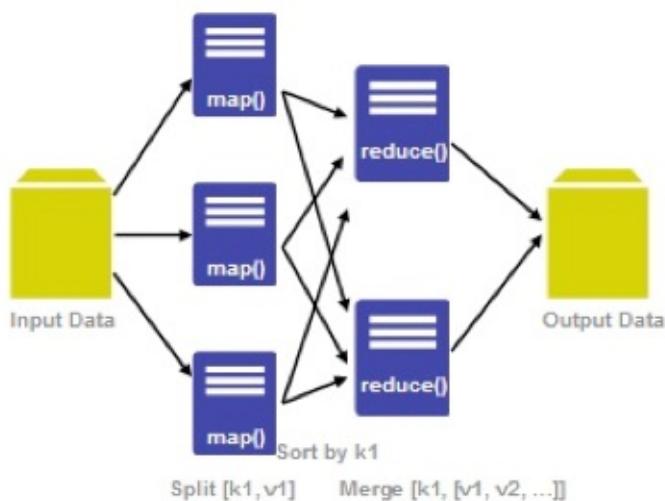
```



# Sqoop

Apache Sqoop (SQL-to-Hadoop) 项目旨在协助 RDBMS 与 Hadoop 之间进行高效的大数据交流。用户可以在 Sqoop 的帮助下，轻松地把关系型数据库的数据导入到 Hadoop 与其相关的系统 (如HBase和Hive)中；同时也可把数据从 Hadoop 系统里抽取并导出到关系型数据库里。除了这些主要的功能外，Sqoop 也提供了一些诸如查看数据库表等实用的小工具。

理论上，Sqoop 支持任何一款支持 JDBC 规范的数据库，如 DB2、MySQL 等。Sqoop 还能够将 DB2 数据库的数据导入到 HDFS 上，并保存为多种文件类型。常见的有定界文本类型，Avro 二进制类型以及 SequenceFiles 类型。在本文里，统一用定界文本类型。



Sqoop中一大亮点就是可以通过hadoop的mapreduce把数据从关系型数据库中导入数据到HDFS。Sqoop架构非常简单，其整合了Hive、Hbase和Oozie，通过map-reduce任务来传输数据，从而提供并发特性和容错。

## Sqoop1和Sqoop 2架构的变迁

首先这两个版本是完全不兼容的，其具体的版本号区别为

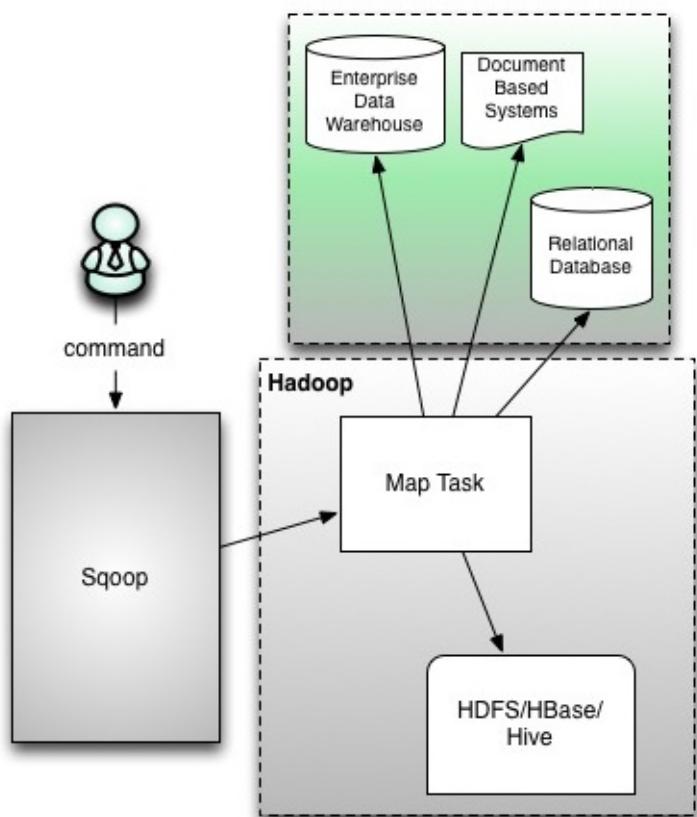
1.4.x为sqoop 1，1.99x为sqoop

sqoop1和sqoop2在架构和用法上已经完全不同。在架构上，sqoop1仅仅使用一个sqoop客户端，sqoop2引入了sqoopserver，对connector实现了集中的管理。

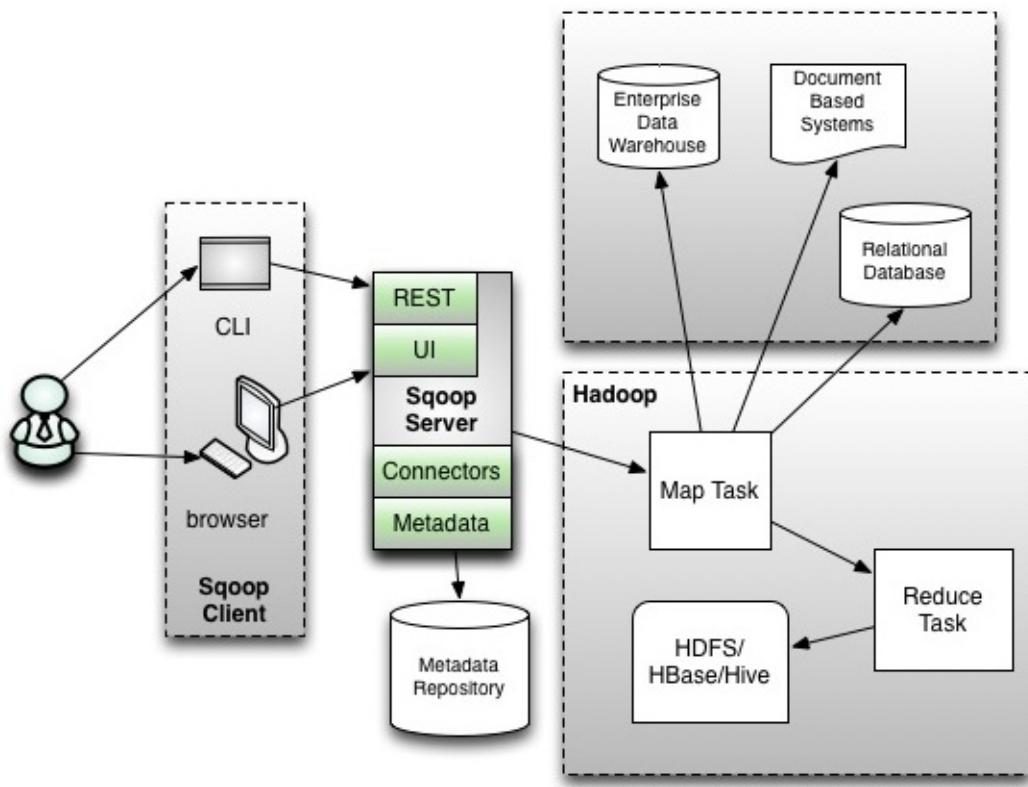
其访问方式也变得多样化了，其可以通过REST API、JAVA API、WEB UI以及CLI控制台方式进行访问。另外，其在安全性能方面也有一定的改善，在sqoop1中我们经常用脚本的方式将HDFS中的数据导入到mysql中，或者反过来将mysql数据导入到HDFS中，其中在脚本里边都要显示指定mysql数据库的用户名和密码的，安全性做的不是太完善。在sqoop2中，如果是通过CLI方式访问的话，会有一个交互过程界面，你输入的密码信息不被看到，同时Sqoop2引入基于角色的安全机制。

下图是sqoop1和sqoop2简单架构对比：

**Sqoop1**架构图：



**Sqoop2**架构图：



两个不同的版本，完全不兼容 版本号划分区别.

Apache版本： 1.4.x(Sqoop1); 1.99.x(Sqoop2)

CDH版本 : Sqoop-1.4.3-cdh4(Sqoop1) ; Sqoop2-1.99.2-cdh4.5.0

- sqoop1优点：架构部署简单
- sqoop1缺点：命令行方式容易出错，格式紧耦合，无法支持所有数据类型，安全机制不够完善，例如密码暴漏，安装需要root权限，connector必须符合 JDBC模型
- sqoop2优点：多种交互方式，命令行，web UI，rest API，connctor集中化管理，所有的链接安装在sqoop server上，完善权限管理机制，connector规范化，仅仅负责数据的读写
- sqoop2缺点：架构稍复杂，配置部署更繁琐

## Sqoop2的安装

### 1. 解压并安装

```
hadoop@Master:~$ sudo tar xvfz sqoop-1.99.6-bin-hadoop200.tar.gz  
hadoop@Master:~$ sudo mv sqoop-1.99.6-bin-hadoop200 /usr/local/sqoop  
hadoop@Master:~$ sudo chmod -R 775 /usr/local/sqoop  
hadoop@Master:~$ sudo chown -R hadoop:hadoop /usr/local/sqoop
```

### 2. 修改环境变量

```
hadoop@Master:~$ sudo nano /etc/profile  
  
#sqoop  
export SQOOP_HOME=/usr/local/sqoop  
export PATH=$SQOOP_HOME/bin:$PATH  
export CATALINA_BASE=$SQOOP_HOME/server  
export LOGDIR=$SQOOP_HOME/logs  
  
hadoop@Master:~$ source /etc/profile
```

### 3. 修改**sqoop**的环境变量

## 1.Sqoop2的安装

```
hadoop@Master:/$ sudo nano /usr/local/sqoop/conf/sqoop.properties

#修改指向我的hadoop安装目录
org.apache.sqoop.submission.engine.mapreduce.configuration.directory=/usr/local/hadoop/etc/hadoop

#catalina.properties 此文件不存在,需要自己建立
hadoop@Master:/$ sudo nano /usr/local/sqoop/conf/catalina.properties

common.loader=/usr/local/hadoop/share/hadoop/common/*.jar,/usr/local/hadoop/share/hadoop/common/lib/*.jar,/usr/local/hadoop/share/hadoop/hdfs/*.jar,/usr/local/hadoop/share/hadoop/hdfs/lib/*.jar,/usr/local/hadoop/share/hadoop/mapreduce/*.jar,/usr/local/hadoop/share/hadoop/mapreduce/lib/*.jar,/usr/local/hadoop/share/hadoop/tools/*.jar,/usr/local/hadoop/share/hadoop/tools/lib/*.jar,/usr/local/hadoop/share/hadoop/yarn/*.jar,/usr/local/hadoop/share/hadoop/yarn/lib/*.jar,/usr/local/hadoop/share/hadoop/httpfs/tomcat/lib/*.jar,
```

下载mysql驱动包，mysql-connector-java-5.1.27.jar

把jar包丢到到\$SQOOP\_HOME/server/lib下面

```
sudo cp mysql-connector-java-5.1.27.jar $SQOOP_HOME/server/lib
```

有时，启动sqoop时可能会遇到找不到JAVA\_HOME的情况，为了保险起见我们直接在配置文件中写入JAVA\_HOME

在/usr/local/sqoop/bin/sqoop.sh中，添加

```
export JAVA_HOME=/usr/lib/jvm/
HADOOP_COMMON_HOME=/usr/local/hadoop/share/hadoop/common
HADOOP_HDFS_HOME=/usr/local/hadoop/share/hadoop/hdfs
HADOOP_MAPRED_HOME=/usr/local/hadoop/share/hadoop/mapreduce
HADOOP_YARN_HOME=/usr/local/hadoop/share/hadoop/yarn
```

## 4. 启动sqoop

```
hadoop@Master:~/mysql-connector-java-5.0.8$ sqoop.sh server start
Sqoop home directory: /usr/local/sqoop
Setting SQOOP_HTTP_PORT:      12000
Setting SQOOP_ADMIN_PORT:     12001
Using CATALINA_OPTS:
Adding to CATALINA_OPTS:      -Dsqoop.http.port=12000 -Dsqoop.admin.port=12001
Using CATALINA_BASE:          /usr/local/sqoop/server
Using CATALINA_HOME:          /usr/local/sqoop/server
Using CATALINA_TMPDIR:        /usr/local/sqoop/server/temp
Using JRE_HOME:               /usr/lib/jvm//jre
Using CLASSPATH:              /usr/local/sqoop/server/bin/bootstrap.jar
```

## 5. 验证启动成功

```
./sqoop.sh server start      启动
./sqoop.sh server stop       停止
./sqoop.sh client           进入客户端
set server --host hadoopMaster --port 12000 --webapp sqoop   设置服务器，注意hadoopMaster为hdfs主机名
show connector --all         查看连接类型
create link --cid 1          创建连接，cid为连接类型id
show link                   查看连接
update link -l 1             修改id为1的连接
delete link -l 1             删除id为1的连接
create job -f 1 -t 2          创建从连接1到连接2的job
show job                     查看job
update job -jid 1            修改job
delete job -jid 1            删除job
status job -jid 1            看看job状态
stop job -jid    1           停止job
```

## 6.Hadoop的配置修改

## 1.Sqoop2的安装

---

需要在Hadoop的yarn-site.xml 这个配置文件中增加以下属性

```
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
```

## 7. 其他参考文档

一篇写的比我好的文档,我就不搬砖了,大家自己看:

## Sqoop1的安装

### 1 解压安装

```
hadoop@Master:~$ sudo tar xvfz sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz

hadoop@Master:~$ sudo mv sqoop-1.4.6.bin__hadoop-2.0.4-alpha /usr/local/sqoop1
hadoop@Master:~$ sudo chmod -R 775 /usr/local/sqoop1
hadoop@Master:~$ sudo chown -R hadoop:hadoop /usr/local/sqoop1
```

### 2 修改环境变量

```
hadoop@Master:~$ sudo nano /etc/profile
#sqoop
export SQOOP_HOME=/usr/local/sqoop1
export PATH=$SQOOP_HOME/bin:$PATH

hadoop@Master:~$ source /etc/profile
```

### 3 配置sqoop的环境变量

下载mysql驱动包，mysql-connector-java-5.1.27.jar  
把jar包丢到到\$SQOOP\_HOME/lib下面

```
hadoop@Master:/usr/local/sqoop1/conf$ cp sqoop-env-template.sh sqoop-env.sh

# 指定各环境变量的实际配置
# Set Hadoop-specific environment variables here.
```

## 2.Sqoop1的安装

```
#Set path to where bin/hadoop is available
#export HADOOP_COMMON_HOME=

#Set path to where hadoop-* core.jar is available
#export HADOOP_MAPRED_HOME=

#set the path to where bin/hbase is available
#export HBASE_HOME=

#Set the path to where bin/hive is available
#export HIVE_HOME=
```

但是一般情况下我们的/etc/profile已经配置相关的环境变量

```
export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:/usr/local/hive/lib
export PATH=${JAVA_HOME}/bin:$PATH

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin:/usr/local/hbase/bin

export JAVA_LIBRARY_PATH=/usr/local/hadoop/lib/native
```

```
export SCALA_HOME=/usr/lib/scala  
export PATH=$PATH:$SCALA_HOME/bin  
  
#sqoop  
export SQOOP_HOME=/usr/local/sqoop1  
export PATH=$SQOOP_HOME/bin:$PATH  
  
#HBASE  
export HBASE_HOME=/usr/local/hbase
```

## 4 开始测试

需要拷贝mysql的驱动到lib下面

```
hadoop@Master:~/mysql-connector-java-5.0.8$ sudo cp mysql-connector-java-5.0.8-bin.jar /usr/local/sqoop1/lib/
```

以mysql为例子

```
IP:192.168.1.178  
用户名:chu888chu888  
密码:skybar  
数据库:hivetestdb  
表:cdsgus
```

```
[root@hadoop01 ~]# sqoop help  
Available commands:  
  codegen          Generate code to interact with database records  
  create-hive-table Import a table definition into Hive  
  eval             Evaluate a SQL statement and display the results  
  export           Export an HDFS directory to a database table  
  help             List available commands  
  import           Import a table from a database to HDFS  
  import-all-tables Import tables from a database to HDFS  
  import-mainframe Import datasets from a mainframe server to HDFS  
  job              Work with saved jobs
```

## 2.Sqoop1的安装

list-databases	List available databases on a server
list-tables	List available tables in a database
merge	Merge results of incremental imports
metastore	Run a standalone Sqoop metastore
version	Display version information

列出所有的数据库

```
hadoop@Master:/usr/local/sqoop1/lib$ sqoop list-databases --connect jdbc:mysql://192.168.1.178 --username chu888chu888 --password skybar
```

列出数据库中所有的表

```
hadoop@Master:/usr/local/sqoop1/lib$ sqoop list-tables --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar
```

导出mysql表到hdfs上

```
hadoop@Master:$ hdfs dfs -mkdir /user/sqoop  
hadoop@Master:$ hdfs dfs -chown sqoop:hadoop /user/sqoop  
sqoop import --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar --table cdsgus --m 2 --target-dir /user/sqoop/cdsgus
```

```
hadoop@Master:~$ sqoop import --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar --table cdsgus --m 2 --target-dir /user/sqoop/cdsgus
```

Warning: /usr/local/sqoop1/.../hcatalog does not exist! HCatalog jobs will fail.

Please set \$HCAT\_HOME to the root of your HCatalog installation.

Warning: /usr/local/sqoop1/.../accumulo does not exist! Accumulo imports will fail.

Please set \$ACCUMULO\_HOME to the root of your Accumulo installation.

Warning: /usr/local/sqoop1/.../zookeeper does not exist! Accumulo imports will fail.

Please set \$ZOOKEEPER\_HOME to the root of your Zookeeper installation.

```
16/03/03 01:28:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
```

## 2.Sqoop1的安装

```
16/03/03 01:28:13 WARN tool.BaseSqoopTool: Setting your password  
on the command-line is insecure. Consider using -P instead.  
16/03/03 01:28:13 INFO manager.MySQLManager: Preparing to use a  
MySQL streaming resultset.  
16/03/03 01:28:13 INFO tool.CodeGenTool: Beginning code generati  
on  
16/03/03 01:28:14 INFO manager.SqlManager: Executing SQL stateme  
nt: SELECT t.* FROM `cdsgus` AS t LIMIT 1  
16/03/03 01:28:14 INFO manager.SqlManager: Executing SQL stateme  
nt: SELECT t.* FROM `cdsgus` AS t LIMIT 1  
16/03/03 01:28:14 INFO orm.CompilationManager: HADOOP_MAPRED_HOME  
is /usr/local/hadoop  
Note: /tmp/sqoop-hadoop/compile/7b9cf86a577c124c063ff5dc2242b3fb  
/cdsgus.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
16/03/03 01:28:17 INFO orm.CompilationManager: Writing jar file:  
/tmp/sqoop-hadoop/compile/7b9cf86a577c124c063ff5dc2242b3fb/cdsg  
us.jar  
16/03/03 01:28:17 WARN manager.MySQLManager: It looks like you a  
re importing from mysql.  
16/03/03 01:28:17 WARN manager.MySQLManager: This transfer can b  
e faster! Use the --direct  
16/03/03 01:28:17 WARN manager.MySQLManager: option to exercise  
a MySQL-specific fast path.  
16/03/03 01:28:17 INFO manager.MySQLManager: Setting zero DATETI  
ME behavior to convertToNull (mysql)  
16/03/03 01:28:17 INFO mapreduce.ImportJobBase: Beginning import  
of cdsgus  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop  
/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLogger  
Binder.class]  
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/slf4j-log  
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for  
an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFact  
ory]  
16/03/03 01:28:17 INFO Configuration.deprecation: mapred.jar is  
deprecated. Instead, use mapreduce.job.jar
```

## 2.Sqoop1的安装

```
16/03/03 01:28:18 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
16/03/03 01:28:18 INFO client.RMProxy: Connecting to ResourceManager at Master/192.168.1.80:8032
16/03/03 01:28:23 INFO db.DBInputFormat: Using read commited transaction isolation
16/03/03 01:28:23 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `cdsgus`
16/03/03 01:28:23 INFO mapreduce.JobSubmitter: number of splits: 2
16/03/03 01:28:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1456939431067_0002
16/03/03 01:28:24 INFO impl.YarnClientImpl: Submitted application application_1456939431067_0002
16/03/03 01:28:24 INFO mapreduce.Job: The url to track the job: http://Master:8088/proxy/application_1456939431067_0002/
16/03/03 01:28:24 INFO mapreduce.Job: Running job: job_1456939431067_0002
16/03/03 01:28:38 INFO mapreduce.Job: Job job_1456939431067_0002 running in uber mode : false
16/03/03 01:28:38 INFO mapreduce.Job: map 0% reduce 0%
16/03/03 01:32:11 INFO mapreduce.Job: map 50% reduce 0%
16/03/03 01:32:13 INFO mapreduce.Job: map 100% reduce 0%
16/03/03 01:32:14 INFO mapreduce.Job: Job job_1456939431067_0002 completed successfully
16/03/03 01:32:14 INFO mapreduce.Job: Counters: 31
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=247110
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=218
        HDFS: Number of bytes written=3130492684
        HDFS: Number of read operations=8
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=4
    Job Counters
        Killed map tasks=1
        Launched map tasks=3
```

## 2.Sqoop1的安装

```
Other local map tasks=3
Total time spent by all maps in occupied slots (ms)=4228
21
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=422821
Total vcore-seconds taken by all map tasks=422821
Total megabyte-seconds taken by all map tasks=432968704
Map-Reduce Framework
  Map input records=20050144
  Map output records=20050144
  Input split bytes=218
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=19391
  CPU time spent (ms)=206680
  Physical memory (bytes) snapshot=313565184
  Virtual memory (bytes) snapshot=3757293568
  Total committed heap usage (bytes)=65142784
File Input Format Counters
  Bytes Read=0
File Output Format Counters
  Bytes Written=3130492684
16/03/03 01:32:14 INFO mapreduce.ImportJobBase: Transferred 2.91
55 GB in 235.5966 seconds (12.672 MB/sec)
16/03/03 01:32:14 INFO mapreduce.ImportJobBase: Retrieved 200501
44 records.
```

 **RUNNING Applications** Logged in as: dr.who

**Cluster Metrics**

	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	1	0	3	4 Gb	16 Gb	0.8	3	16	0	2	0	0	0	0	0

Show 20+ entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1456939431067_0002	hadoop	cdsgus.jar	MAPREDUCE	default	Wed, 02 Mar 2016 17:28:23 GMT	N/A	RUNNING	UNDEFINED		ApplicationMaster

Showing 1 to 1 of 1 entries First Previous Next Last

```
导出mysql表全部数据到hive
hive> create database test_sqoop;
OK
Time taken: 0.81 seconds
```

## 2.Sqoop1的安装

```
hive> show databases;
OK
chu888chu888
default
test_sqoop
Time taken: 0.247 seconds, Fetched: 3 row(s)
hive>
```

使用sqoop创建表并导入表

```
sqoop import --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar --table cdsgus --hive-import --hive-table test_sqoop.cdsgus
```

将数据从hive导入mysql

```
mysql> use hivetestdb;
Database changed
mysql> show tables;
+-----+
| Tables_in_test_sqoop |
+-----+
| cdsgus                |
+-----+
1 row in set (0.00 sec)
mysql> truncate cdsgus;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from cdsgus;
Empty set (0.00 sec)
```

```
sqoop --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar --table cdsgus --export-dir /user/hive/warehouse/test_sqoop.db/cdsgus/ --input-fields-terminated-by '\0001'
```

增量导入

```
sqoop import --connect jdbc:mysql://192.168.1.178/hivetestdb --username chu888chu888 --password skybar --table cdsgus --hive-import --hive-table test_sqoop.cdsgus --check-column id --incremental append --last-value 2
```

### HBASE导入

```
sqoop import --connect jdbc:mysql://192.168.1.178/hive_hadoop  
--username chu888chu888 --password skybar --table TBLS --hbase-table  
TBLS --hbase-create-table --hbase-row-key TBL_ID --column-family SD_ID
```

## 错误阻力

```
hadoop@Master:/ $ sqoop import --connect jdbc:mysql://192.168.1.1  
78/hivetestdb --username chu888chu888 --password skybar --table  
cdsgus  
Warning: /usr/local/sqoop1/..../hcatalog does not exist! HCatalog  
jobs will fail.  
Please set $HCAT_HOME to the root of your HCatalog installation.  
Warning: /usr/local/sqoop1/..../accumulo does not exist! Accumulo  
imports will fail.  
Please set $ACCUMULO_HOME to the root of your Accumulo installat  
ion.  
Warning: /usr/local/sqoop1/..../zookeeper does not exist! Accumulo  
imports will fail.  
Please set $ZOOKEEPER_HOME to the root of your Zookeeper install  
ation.  
16/03/03 00:32:16 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6  
16/03/03 00:32:16 WARN tool.BaseSqoopTool: Setting your password  
on the command-line is insecure. Consider using -P instead.  
16/03/03 00:32:16 INFO manager.MySQLManager: Preparing to use a  
MySQL streaming resultset.  
16/03/03 00:32:16 INFO tool.CodeGenTool: Beginning code generati  
on  
16/03/03 00:32:16 INFO manager.SqlManager: Executing SQL stateme  
nt: SELECT t.* FROM `cdsgus` AS t LIMIT 1  
16/03/03 00:32:16 ERROR manager.SqlManager: Error reading from d  
atabase: java.sql.SQLException: Streaming result set com.mysql.j  
dbc.RowDataDynamic@654f0d9c is still active. No statements may b  
e issued when any streaming result sets are open and in use on a  
given connection. Ensure that you have called .close() on any a  
ctive streaming result sets before attempting more queries.  
java.sql.SQLException: Streaming result set com.mysql.jdbc.RowDa
```

```
taDynamic@654f0d9c is still active. No statements may be issued
when any streaming result sets are open and in use on a given co
nnection. Ensure that you have called .close() on any active str
eaming result sets before attempting more queries.

        at com.mysql.jdbc.SQLSyntax.createSQLException(SQLSyntax.java:
914)
        at com.mysql.jdbc.MysqlIO.checkForOutstandingStreamingData(M
ySQLIO.java:2181)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1542)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1723)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3277)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3206)
        at com.mysql.jdbc.Statement.executeQuery(Statement.java:1232
)
        at com.mysql.jdbc.Connection.getMaxBytesPerChar(Connection.j
ava:3673)
        at com.mysql.jdbc.Field.getMaxBytesPerCharacter(Field.java:4
82)
        at com.mysql.jdbc.ResultSetMetaData.getPrecision(ResultSetMe
taData.java:443)
        at org.apache.sqoop.manager.SqlManager.getColumnInfoForRawQu
ery(SqlManager.java:286)
        at org.apache.sqoop.manager.SqlManager.getColumnTypesForRawQ
uery(SqlManager.java:241)
        at org.apache.sqoop.manager.SqlManager.getColumnTypes(SqlMan
ager.java:227)
        at org.apache.sqoop.manager.ConnManager.getColumnTypes(ConnM
anager.java:295)
        at org.apache.sqoop.orm.ClassWriter.getColumnTypes(ClassWrit
er.java:1833)
        at org.apache.sqoop.orm.ClassWriter.generate(ClassWriter.jav
a:1645)
        at org.apache.sqoop.toolCodeGenTool.generateORM(CodeGenTool
.java:107)
        at org.apache.sqoop.tool.ImportTool.importTable(ImportTool.j
ava:478)
        at org.apache.sqoop.tool.ImportTool.run(ImportTool.java:605)
        at org.apache.sqoop.Sqoop.run(Sqoop.java:143)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:70)
        at org.apache.sqoop.Sqoop.runSqoop(Sqoop.java:179)
```

```
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:218)
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:227)
at org.apache.sqoop.main(Sqoop.java:236)
16/03/03 00:32:17 ERROR tool.ImportTool: Encountered IOException
running import job: java.io.IOException: No columns to generate
for ClassWriter
at org.apache.sqoop.orm.ClassWriter.generate(ClassWriter.java:1651)
at org.apache.sqoop.toolCodeGenTool.generateORM(CodeGenTool.java:107)
at org.apache.sqoop.tool.ImportTool.importTable(ImportTool.java:478)
at org.apache.sqoop.tool.ImportTool.run(ImportTool.java:605)
at org.apache.sqoop.Sqoop.run(Sqoop.java:143)
at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:70)
at org.apache.sqoop.Sqoop.runSqoop(Sqoop.java:179)
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:218)
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:227)
at org.apache.sqoop.main(Sqoop.java:236)
```

如果出现上面的错误，请更新/usr/lib/sqoop/mysql-java-connector.jar文件。

ISSUE: <https://issues.apache.org/jira/browse/SQOOP-1400>

这里面还有一种可能就是你在hadoop/common这个目录也有一个mysql的驱动包，这个包也许版本很古老！

```
hadoop@Master:/usr/local/hadoop/share/hadoop/common$ ls
hadoop-common-2.6.0.jar  hadoop-common-2.6.0-tests.jar  hadoop-n
fs-2.6.0.jar  jdiff  lib  mysql-connector-java-5.0.8-bin.jar  so
urces  templates
hadoop@Master:/usr/local/hadoop/share/hadoop/common$ sudo rm -rf
mysql-connector-java-5.0.8-bin.jar
```

## FAQ

出现这二段警告,我们怎么解决,需要修改脚本

## 2.Sqoop1的安装

修改hadoop@hadoopmaster:/usr/local/sqoop/bin\$ nano configure-sqoop

```
hadoop@hadoopmaster:/usr/local/hive/lib$ sqoop list-databases --  
connect jdbc:mysql://hadoopslave2 --username hive --password hiv  
e  
Warning: /usr/local/sqoop/.../hcatalog does not exist! HCatalog j  
obs will fail.  
Please set $HCAT_HOME to the root of your HCatalog installation.  
Warning: /usr/local/sqoop/.../accumulo does not exist! Accumulo i  
mports will fail.  
Please set $ACCUMULO_HOME to the root of your Accumulo installat  
ion.  
Warning: /usr/local/sqoop/.../zookeeper does not exist! Accumulo  
imports will fail.  
Please set $ZOOKEEPER_HOME to the root of your Zookeeper install  
ation.
```

# 一 数据库与表

## 数据库基本操作命令

### 1 选择数据库命令

- Mysql:

登录方式：

```
#直接本地登录 root:123456
#mysql -u root -p
#远程登录 192.168.1.178 chu888chu888:skybar
#mysql -h 192.168.1.178 -u chu888chu888 -p
```

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| Northwind    |
| Pubs          |
| ReportServer |
| hive          |
| hive_hadoop   |
| hivetestdb    |
| mysql         |
| performance_schema |
+-----+
9 rows in set (0.10 sec)
```

```
mysql> use Pubs;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
```

### 3.Sqoop综合案例

```
mysql> show tables;
+-----+
| Tables_in_Pubs |
+-----+
| authors      |
| discounts    |
| employee     |
| jobs         |
| pub_info     |
| publishers   |
| roysched     |
| sales        |
| stores       |
| titleauthor |
| titles       |
+-----+
11 rows in set (0.00 sec)
```

```
mysql> show columns from jobs;
+-----+-----+-----+-----+-----+
| Field | Type           | Null | Key | Default
|       | Extra          |      |     |          |
+-----+-----+-----+-----+-----+
| job_id | smallint(6)    | NO  | PRI | NULL
|       | auto_increment |      |     |          |
| job_desc | varchar(50) | NO  |     | New Position - t
| title not formalized yet |      |
| min_lvl | tinyint(3) unsigned | NO  |     | NULL
|          |                  |      |     |
| max_lvl | tinyint(3) unsigned | NO  |     | NULL
|          |                  |      |     |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

```
mysql> show status;
+-----+-----+
| Variable_name | Value |
```

Aborted_clients	0	
Aborted_connects	0	
Binlog_cache_disk_use	0	
Binlog_cache_use	0	
Binlog_stmt_cache_disk_use	0	
Binlog_stmt_cache_use	0	

- Oracle: 等待补充
- DB2: 等待补充
- Inceptor:

```
[root@dhc-1 ~]# beeline -u jdbc:hive2://192.168.1.70:10000/
scan complete in 2ms
Connecting to jdbc:hive2://192.168.1.70:10000/
2016-03-22 08:33:48,094 INFO jdbc.Utils: Supplied authorities: 1
92.168.1.70:10000
2016-03-22 08:33:48,094 INFO jdbc.Utils: Resolved authority: 192
.168.1.70:10000
Connected to: Apache Hive (version 0.12.0-transwarp-tdh40)
Driver: Hive JDBC (version 0.12.0-transwarp-tdh40)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 0.12.0-transwarp-tdh40 by Apache Hive
0: jdbc:hive2://192.168.1.70:10000/> show databases;
+-----+
| database_name   |
+-----+
| default         |
+-----+
1 row selected (2.282 seconds)
0: jdbc:hive2://192.168.1.70:10000/>

1 row selected (2.282 seconds)
0: jdbc:hive2://192.168.1.70:10000/> use default;
No rows affected (0.068 seconds)
0: jdbc:hive2://192.168.1.70:10000/> show tables;
+-----+
| tab_name   |
+-----+
+-----+
No rows selected (0.08 seconds)
0: jdbc:hive2://192.168.1.70:10000/>
```

## 实验准备数据一 Pubs数据库

为了能在Inceptor中实现兼容性测试,我们必须去移植一下我们样例数据库中的数据(来之微软的Pubs数据库)

### 1 载入Inceptor

```
--登录Inceptor server节点  
beeline -u jdbc:hive2://192.168.1.70:10000/
```

## 2 使用Sqoop将MySQL数据库导入HDFS

1. 在Inceptor metastore节点服务器上安装sqoop服务

```
yum install sqoop
```

2. 由于Inceptor-SQL中metastore中已经安装了mysql，就不需要安装mysql了

3. 将mysql-connector-java-5.1.38tar.gz驱动包先解压

```
tar -zxvf mysql-connector-java-5.1.38tar.gz
```

4. cd进刚刚解压后的目录，将里面的mysql-connector-java-5.1.38-bin.jar包copy到/usr/lib/sqoop/lib本地目录下

5. 从mysql——>HDFS上（import，将mysql中的db1数据库里面的表导入到/user/datadir，这里的datadir目录一定不要事先创建，不然会报错，语句执行的时候会自动创建目录的！最后一行的-m表示map成4个文件）

```
sqoop import \  
--username chu888chu888 \  
--password skybar \  
--connect jdbc:mysql://192.168.1.178:3306/Pubs \  
--table titleauthor \  
--target-dir /user/chu888chu888/data/titleauthor -m 4  
  
sqoop import \  
--username chu888chu888 \  
--password skybar \  
--connect jdbc:mysql://192.168.1.178:3306/Pubs \  
--table authors \  
--target-dir /user/chu888chu888/data/authors -m 4  
  
sqoop import \  

```

```
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table authors \
--target-dir /user/chu888chu888/data/employee -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table discounts \
--target-dir /user/chu888chu888/data/discounts -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table jobs \
--target-dir /user/chu888chu888/data/jobs -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table pub_info \
--target-dir /user/chu888chu888/data/pub_info -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table publishers \
--target-dir /user/chu888chu888/data/publishers -m 4
```

有一个问题如果表没有主键的话，就会导入不了。

```
alter table roysched add roysched_id int unsigned not Null auto_
increment primary key;
```

```
sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table roysched \
--target-dir /user/chu888chu888/data/roysched -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table sales \
--target-dir /user/chu888chu888/data/sales -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table stores \
--target-dir /user/chu888chu888/data/stores -m 4

sqoop import \
--username chu888chu888 \
--password skybar \
--connect jdbc:mysql://192.168.1.178:3306/Pubs \
--table titles \
--target-dir /user/chu888chu888/data/titles -m 4
```

## 6 SQL SERVER导入的问题

```
sqoop import \
--table address \
--connect "jdbc:sqlserver://192.168.1.139:1433;database=Adventureworks" \
--username=sa \
--password=123456 \
--hive-drop-import-delims \
--null-string '\\N' \
--null-non-string '\\N' \
--fields-terminated-by '\001' \
--target-dir /user/test/address1 -m 1
```

### 3 在Inceptor中建立外表结构

```
mysql> desc authors;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| au_id | varchar(11) | NO | PRI | NULL | |
| au_lname | varchar(40) | NO | MUL | NULL | |
| au_fname | varchar(20) | NO | | NULL | |
| phone | varchar(12) | NO | | UNKNOWN | |
| address | varchar(40) | YES | | NULL | |
| city | varchar(20) | YES | | NULL | |
| state | varchar(2) | YES | | NULL | |
| zip | varchar(5) | YES | | NULL | |
| contract | bit(1) | NO | | NULL | |
+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

create external table authors
(
    au_id          STRING,
    au_lname       STRING,
    au_fname       STRING,
    phone          STRING,
    address        STRING,
    city           STRING,
    state          STRING,
```

### 3.Sqoop综合案例

```
zip           STRING,  
contract      STRING  
)row format delimited fields terminated by ',' location '/user/c  
hu888chu888/data/authors';
```

```
mysql> desc discounts;  
+-----+-----+-----+-----+-----+  
| Field | Type | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+  
| discounttype | varchar(40) | NO | | NULL | |  
| stor_id | varchar(4) | YES | | NULL | |  
| lowqty | smallint(6) | YES | | NULL | |  
| highqty | smallint(6) | YES | | NULL | |  
| discount | decimal(6,2) | NO | | NULL | |  
+-----+-----+-----+-----+-----+
```

```
5 rows in set (0.00 sec)  
create external table discounts  
(
```

```
    discounttype      STRING,  
    stor_id          STRING,  
    lowqty           STRING,  
    highqty          STRING,  
    discount          STRING,  
    discount_id       STRING  
)row format delimited fields terminated by ',' location '/user/c  
hu888chu888/data/discounts';
```

```
mysql> desc employee;  
+-----+-----+-----+-----+  
| Field | Type | Null | Key | Default | Extra |  
|  
+-----+-----+-----+-----+  
| emp_id | varchar(9) | NO | PRI | NULL |  
|  
| fname | varchar(20) | NO | | NULL |  
|
```

### 3.Sqoop综合案例

```
| minit      | varchar(1)          | YES |       | NULL  |
|           |
| lname      | varchar(30)         | NO  | MUL   | NULL  |
|           |
| job_id     | smallint(6)         | NO  |       | 1      |
|           |
| job_lvl    | tinyint(3) unsigned | YES |       | 10     |
|           |
| pub_id     | varchar(4)          | NO  |       | 9952   |
|           |
| hire_date  | date               | YES |       | NULL  |
|
+-----+-----+-----+-----+-----+
-+
8 rows in set (0.00 sec)
```

```
create external table employee
(
    emp_id      STRING,
    fname        STRING,
    minit        STRING,
    lname        STRING,
    job_id       STRING,
    job_lvl      STRING,
    pub_id       STRING,
    hire_date    STRING
)row format delimited fields terminated by ',' location '/user/chu888chu888/data/employee';
```

```
mysql> desc jobs;
+-----+-----+-----+-----+
| Field  | Type           | Null | Key  | Default |
|        | Extra          |      |       |
+-----+-----+-----+-----+
| job_id | smallint(6)    | NO  | PRI  | NULL    |
|        | auto_increment |      |       |
| job_desc | varchar(50)  | NO  |       | New Position - t
```

### 3.Sqoop综合案例

```
title not formalized yet | |
| min_lvl | tinyint(3) unsigned | NO | | NULL
| |
| max_lvl | tinyint(3) unsigned | NO | | NULL
| |
+-----+-----+-----+-----+
-----+-----+
4 rows in set (0.00 sec)
create external table jobs
(
    job_id      STRING,
    job_desc     STRING,
    min_lvl      STRING,
    max_lvl      STRING
)row format delimited fields terminated by ',' location '/user/c
hu888chu888/data/jobs';
```

```
mysql> desc pub_info;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| pub_id | varchar(4) | NO   | PRI | NULL    |       |
| logo    | longblob   | YES  |     | NULL    |       |
| pr_info | longtext   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

create external table pub_info
(
    pub_id      STRING,
    logo        STRING,
    pr_info     STRING
)row format delimited fields terminated by ',' location '/user/c
hu888chu888/data/pub_info';
```

```
mysql> desc publishers;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
```

### 3.Sqoop综合案例

```
| pub_id | varchar(4) | NO  | PRI | NULL   |      |
| pub_name | varchar(40) | YES |      | NULL   |      |
| city    | varchar(20) | YES |      | NULL   |      |
| state   | varchar(2)  | YES |      | NULL   |      |
| country | varchar(30) | YES |      | USA    |      |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
create external table publishers
(
    pub_id      STRING,
    pub_name    STRING,
    city        STRING,
    state       STRING,
    country     STRING
)row format delimited fields terminated by ',' location '/user/c
hu888chu888/data/publishers';

mysql> desc roysched;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| title_id   | varchar(6) | NO  | MUL | NULL   |      |
| lorange    | int(11)   | YES |      | NULL   |      |
| hirange    | int(11)   | YES |      | NULL   |      |
| royalty    | int(11)   | YES |      | NULL   |      |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
create external table roysched
(
    title_id    STRING,
    lorange     STRING,
    hirange     STRING,
    royalty     STRING
)row format delimited fields terminated by ',' location '/user/c
hu888chu888/data/roysched';

mysql> desc sales;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |

```

### 3.Sqoop综合案例

```
+-----+-----+-----+-----+-----+
| stor_id | varchar(4) | NO   | PRI  | NULL   |       |
| ord_num | varchar(20) | NO   | PRI  | NULL   |       |
| ord_date | date      | YES  |       | NULL   |       |
| qty      | smallint(6) | NO   |       | NULL   |       |
| payterms | varchar(12) | NO   |       | NULL   |       |
| title_id | varchar(6)  | NO   | PRI  | NULL   |       |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
create external table sales
(
    stor_id      STRING,
    ord_num      STRING,
    ord_date     STRING,
    qty          STRING,
    title_id     STRING
)row format delimited fields terminated by ',' location '/user/chu888chu888/data/sales';
```

```
mysql> desc stores;
```

```
+-----+-----+-----+-----+-----+
| Field        | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| stor_id      | varchar(4) | NO   | PRI  | NULL   |       |
| stor_name    | varchar(40) | YES  |       | NULL   |       |
| stor_address | varchar(40) | YES  |       | NULL   |       |
| city         | varchar(20) | YES  |       | NULL   |       |
| state        | varchar(2)  | YES  |       | NULL   |       |
| zip          | varchar(5)  | YES  |       | NULL   |       |
+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

```
create external table stores
(
    stor_id      STRING,
    stor_name    STRING,
    stor_address STRING,
    city         STRING,
    zip          STRING
```

### 3.Sqoop综合案例

```
)row format delimited fields terminated by ',' location '/user/c  
hu888chu888/data/stores';

mysql> desc titleauthor;
+-----+-----+-----+-----+-----+
--+
| Field      | Type           | Null | Key | Default | Extr
a |
+-----+-----+-----+-----+-----+
--+
| au_id       | varchar(11)    | NO   | PRI | NULL    | 
|
| title_id    | varchar(6)     | NO   | PRI | NULL    | 
|
| au_ord      | tinyint(3) unsigned | YES  |      | NULL    | 
|
| royaltyper  | int(11)        | YES  |      | NULL    | 
|
+-----+-----+-----+-----+-----+
--+
4 rows in set (0.00 sec)

create external table titleauthor
(
    au_id STRING,
    title_id STRING,
    au_ord TinyInt,
    royaltyper INT
)row format delimited fields terminated by ',' location '/user/c  
hu888chu888/data/titleauthor';

mysql> desc titles;
+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| title_id    | varchar(6)     | NO   | PRI | NULL    | 
|
| title       | varchar(80)    | NO   | MUL | NULL    | 
|
| type        | varchar(12)    | NO   |      | UNDECIDED | 
|
| pub_id      | varchar(4)     | YES  |      | NULL    | 
|
+-----+-----+-----+-----+-----+
```

```
| price      | decimal(19,4) | YES   |       | NULL    |       |
| advance    | decimal(19,4) | YES   |       | NULL    |       |
| royalty    | int(11)       | YES   |       | NULL    |       |
| ytd_sales  | int(11)       | YES   |       | NULL    |       |
| notes      | varchar(200)  | YES   |       | NULL    |       |
| pubdate    | datetime      | NO    |       | NULL    |       |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

create external table titles
(
    title_id STRING,
    title    STRING,
    type    STRING,
    pub_id   STRING,
    price   STRING,
    advance  STRING,
    royalty  STRING,
    ytd_sales STRING,
    notes   STRING,
    pubdate  STRING
)row format delimited fields terminated by ',' location '/user/c
hu888chu888/data/titles';
```

## 一 SQOOP2的实验

### 数据库基本操作命令

#### 1 选择数据库命令

- Mysql:

现在我用一个例子讲解sqoop2的具体使用方法,数据准备,有一个mysql的表叫worker，里面有三条数据，我们要将其导入hadoop,这是建表语句

登录方式：

```
#直接本地登录 root:123456
```

```
#mysql -u root -p
```

```
mysql> show databases;
```

```
+-----+  
| Database      |  
+-----+  
| information_schema |  
| hive          |  
| mysql          |  
| performance_schema |  
+-----+  
4 rows in set (0.02 sec)
```

```
mysql> create database chu888chu888;
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> use chu888chu888;
```

```
Database changed
```

```
mysql> CREATE TABLE `workers` (  
    ->   `id` int(11) NOT NULL AUTO_INCREMENT,  
    ->   `name` varchar(20) NOT NULL,  
    ->   PRIMARY KEY (`id`)  
    -> ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> insert into workers (name) values ('jack');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into workers (name) values ('vicky');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into workers (name) values ('martin');
```

```
Query OK, 1 row affected (0.00 sec)
```

## 2. 导入数据

```
$sqoop2-shell
sqoop:000> show connector
16/08/09 13:35:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
+-----+-----+-----+
|      Name          | Version |          C
lass
| Supported Directions |
+-----+-----+-----+
| oracle-jdbc-connector | 1.99.7 | org.apache.sqoop.connector.
jdbc.oracle.OracleJdbcConnector | FROM/TO           |
| sftp-connector       | 1.99.7 | org.apache.sqoop.connector.
sftp.SftpConnector     | TO            |
| kafka-connector     | 1.99.7 | org.apache.sqoop.connector.
kafka.KafkaConnector   | TO            |
| kite-connector       | 1.99.7 | org.apache.sqoop.connector.
kite.KiteConnector     | FROM/TO         |
| ftp-connector        | 1.99.7 | org.apache.sqoop.connector.
ftp.FtpConnector       | TO            |
| hdfs-connector       | 1.99.7 | org.apache.sqoop.connector.
hdfs.HdfsConnector     | FROM/TO         |
| generic-jdbc-connector | 1.99.7 | org.apache.sqoop.connector.
jdbc.GenericJdbcConnector | FROM/TO         |
+-----+-----+-----+
|-----+-----+-----+
```

## **sqoop**主要特点

- 1、可以将关系型数据库中的数据导入hdfs、hive或者hbase等hadoop组件中，也可将hadoop组件中的数据导入到关系型数据库中；
- 2、sqoop在导入导出数据时，充分采用了map-reduce计算框架，根据输入条件生成一个map-reduce作业，在hadoop集群中运行。采用map-reduce框架同时在多个节点进行import或者export操作，速度比单节点运行多个并行导入导出效率高，同时提供了良好的并发性和容错性；
- 3、支持insert、update模式，可以选择参数，若内容存在就更新，若不存在就插入；
- 4、对国外的主流关系型数据库支持性更好。

## **datax**主要特点：

- 1、异构数据库和文件系统之间的数据交换；
- 2、采用Framework + plugin架构构建，Framework处理了缓冲，流控，并发，上下文加载等高速数据交换的大部分技术问题，提供了简单的接口与插件交互，插件仅需实现对数据处理系统的访问；
- 3、数据传输过程在单进程内完成，全内存操作，不读写磁盘，也没有IPC；
- 4、开放式的框架，开发者可以在极短的时间开发一个新插件以快速支持新的数据库/文件系统。

## **sqoop**和**datax**的区别：

- 1、sqoop采用map-reduce计算框架进行导入导出，而datax仅仅在运行datax的单台机器上进行数据的抽取和加载，速度比sqoop慢了许多；
- 2、sqoop只可以在关系型数据库和hadoop组件之间进行数据迁移，而在hadoop相关组件之间，比如hive和hbase之间就无法使用sqoop互相导入导出数据，同时在关系型数据库之间，比如mysql和oracle之间也无法通过sqoop导入导出数据。与之

相反，datax能够分别实现关系型数据库和hadoop组件之间、关系型数据库之间、hadoop组件之间的数据迁移；

3、sqoop是专门为hadoop而生，对hadoop支持度好，而datax可能会出现不支持高版本hadoop的现象；

4、sqoop只支持官方提供的指定几种关系型数据库和hadoop组件之间的数据交换，而在datax中，用户只需根据自身需求修改文件，生成相应rpm包，自行安装之后就可以使用自己定制的插件；

## 性能比较

### 1、mysql->hdfs

在mysql中生成50,000,000条数据，将这些数据分别使用datax和sqoop导入到hdfs中，分别比较它们的性能参数：

sqoop:

属性	值
CPU时间(ms)	325500
读取物理内存快照大小(byte)	3045625856
读取虚拟内存快照大小(byte)	10975498240
平均速率(MB/s)	20.0809
总时间(s)	99.2047

```

2016-09-08 17:25:39,121 INFO mapreduce.Job: Job job_1473141398464_0012 completed successfully
2016-09-08 17:25:39,513 INFO mapreduce.Job: Counters: 30
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=441700
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=442
    HDFS: Number of bytes written=2088888897
    HDFS: Number of read operations=16
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=8
  Job Counters
    Launched map tasks=4
    Other local map tasks=4
    Total time spent by all maps in occupied slots (ms)=335257
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=335257
    Total vcore-seconds taken by all map tasks=335257
    Total megabyte-seconds taken by all map tasks=343303168
  Map-Reduce Framework
    Map input records=50000000
    Map output records=50000000
    Input split bytes=442
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=3138
    CPU time spent (ms)=325500
    Physical memory (bytes) snapshot=3045625856
    Virtual memory (bytes) snapshot=10975498240
    Total committed heap usage (bytes)=3025141760
  File Input Format Counters
    Bytes Read=0
  File Output Format Counters
    Bytes Written=2088888897
2016-09-08 17:25:39,520 INFO mapreduce.ImportJobBase: Transferred 1.9454 GB in 99.2047 seconds
(20.0809 MB/sec)
2016-09-08 17:25:39,524 INFO mapreduce.ImportJobBase: Retrieved 50000000 records.
[root@tdhl bin]# 

```

datax:

属性	值
CPU平均占用率(%)	21.99
平均速率(MB/s)	4.95
总时间(s)	202

## 5.DataX性能对比

```
2016-09-08 17:30:41.481 [job-0] INFO JobContainer - DataX jobId [0] completed successfully.
2016-09-08 17:30:41.482 [job-0] INFO HookInvoker - No hook invoked, because base dir not exist
s or is a file: /root/datax/hook
2016-09-08 17:30:41.584 [job-0] INFO JobContainer -
    [total cpu info] =>
        averageCpu | maxDeltaCpu | minDeltaCpu
        21.99%    | 21.99%   | 21.99%
    [total gc info] =>
        NAME          | totalGCCount | maxDeltaGCCCount | minDeltaGCCou
nt      | totalGCTime     | maxDeltaGCTime   | minDeltaGCTime
        PS MarkSweep | 0           | 0             | 0
        | 0.000s       | 0.000s       | 0.000s       |
        PS Scavenge   | 664         | 664          | 664
        | 1.280s       | 1.280s       | 1.280s
[root@tdhl bin]#
```

## 2、oracle->hdfs

在oracle中生成50,000,000条数据，将这些数据分别使用datax和sqoop导入到hdfs中，分别比较它们的性能参数：

sqoop：

属性	值
CPU时间	86510毫秒
读取物理内存快照大小	2865557504
读取虚拟内存快照大小	10937077760
平均速率	6.4137MB/s
总时间	94.9979s

## 5.DataX性能对比

```
FILE: Number of bytes written=442924
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=439
HDFS: Number of bytes written=638888897
HDFS: Number of read operations=16
HDFS: Number of large read operations=0
HDFS: Number of write operations=8
Job Counters
    Launched map tasks=4
    Other local map tasks=4
    Total time spent by all maps in occupied slots (ms)=223035
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=223035
    Total vcore-seconds taken by all map tasks=223035
    Total megabyte-seconds taken by all map tasks=228387840
Map-Reduce Framework
    Map input records=50000000
    Map output records=50000000
    Input split bytes=439
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=572
    CPU time spent (ms)=86510
    Physical memory (bytes) snapshot=2865557504
    Virtual memory (bytes) snapshot=10937077760
    Total committed heap usage (bytes)=1811415040
File Input Format Counters
    Bytes Read=0
File Output Format Counters
    Bytes Written=638888897
2016-09-11 14:37:25,433 INFO mapreduce.ImportJobBase: Transferred 609.2919 MB in 94.9979 seconds (6.4137 MB/sec)
2016-09-11 14:37:25,439 INFO mapreduce.ImportJobBase: Retrieved 50000000 records.
[root@tdhl bin]# 
```

datax:

属性	值
CPU平均占用率	15.50%
平均速率	5.14MB/s
总时间	122s

```
2016-09-11 14:00:15.706 [job-0] INFO JobContainer -  
    [total cpu info] =>  
        averageCpu | maxDeltaCpu | minDeltaCpu  
        15.50%     | 15.50%   | 15.50%  
  
        [total gc info] =>  
            NAME          | totalGCCount    | maxDeltaGCCount | minDeltaGCCou  
nt      | totalGCTime    | maxDeltaGCTime | minDeltaGCTime | nt  
| PS MarkSweep | 0           | 0             | 0             |  
| 0.000s       | 0.000s       | 0.000s       | 0             |  
| PS Scavenge  | 359          | 359          | 359          |  
| 0.744s       | 0.744s       | 0.744s       | 359          |  
  
2016-09-11 14:00:15.706 [job-0] INFO JobContainer - PerfTrace not enable!  
2016-09-11 14:00:15.707 [job-0] INFO StandAloneJobContainerCommunicator - Total 50000000 records, 538888897 bytes | Speed 5.14MB/s, 500000 records/s | Error 0 records, 0 bytes | All Task WaitWriterTime 7.264s | All Task WaitReaderTime 19.788s | Percentage 100.00%  
2016-09-11 14:00:15.708 [job-0] INFO JobContainer -  
任务启动时刻 : 2016-09-11 13:58:13  
任务结束时刻 : 2016-09-11 14:00:15  
任务总计耗时 : 122s  
任务平均流量 : 5.14MB/s  
记录写入速度 : 500000rec/s  
读出记录总数 : 500000000  
读写失败总数 : 0
```

## 与TDH的兼容性

1、与TDH中的hadoop版本兼容，能够将关系型数据库中数据导入TDH中的hdfs中；

2、datax拥有一个sqoop没有的功能，就是将数据从hdfs导入到hbase，但是该功能目前仅仅支持的hbase版本为：0.94.x和1.1.x两个。而TDH中hyperbase的hbase版本为0.98.6，所以也不支持TDH的Hyperbase。

# 第十一章 HBASE

## 一 引子

在说Hbase是个啥家伙之前，首先我们来看看两个概念，面向行存储和面向列存储。面向行存储，我相信大伙儿应该都清楚，我们熟悉的RDBMS就是此种类型的，面向行存储的数据库主要适合于事务性要求严格场合，或者说面向行存储的存储系统适合OLTP，但是根据CAP理论(参考:[CAP理论参考](#))，传统的RDBMS，为了实现强一致性，通过严格的ACID事务来进行同步，这就造成了系统的可用性和伸缩性方面大大折扣，而目前的很多NoSQL产品，包括Hbase，它们都是一种最终一致性的系统，它们为了高的可用性牺牲了一部分的一致性。好像，我上面说了面向列存储，那么到底什么是面向列存储呢？Hbase,Cassandra,Bigtable都属于面向列存储的分布式存储系统。看到这里，如果您不明白Hbase是个啥东东，不要紧，我再总结一下下：

Hbase是一个面向列存储的分布式存储系统，它的优点在于可以实现高性能的并发读写操作，同时Hbase还会对数据进行透明的切分，这样就使得存储本身具有了水平伸缩性。

## 二 Hbase是个啥东东？

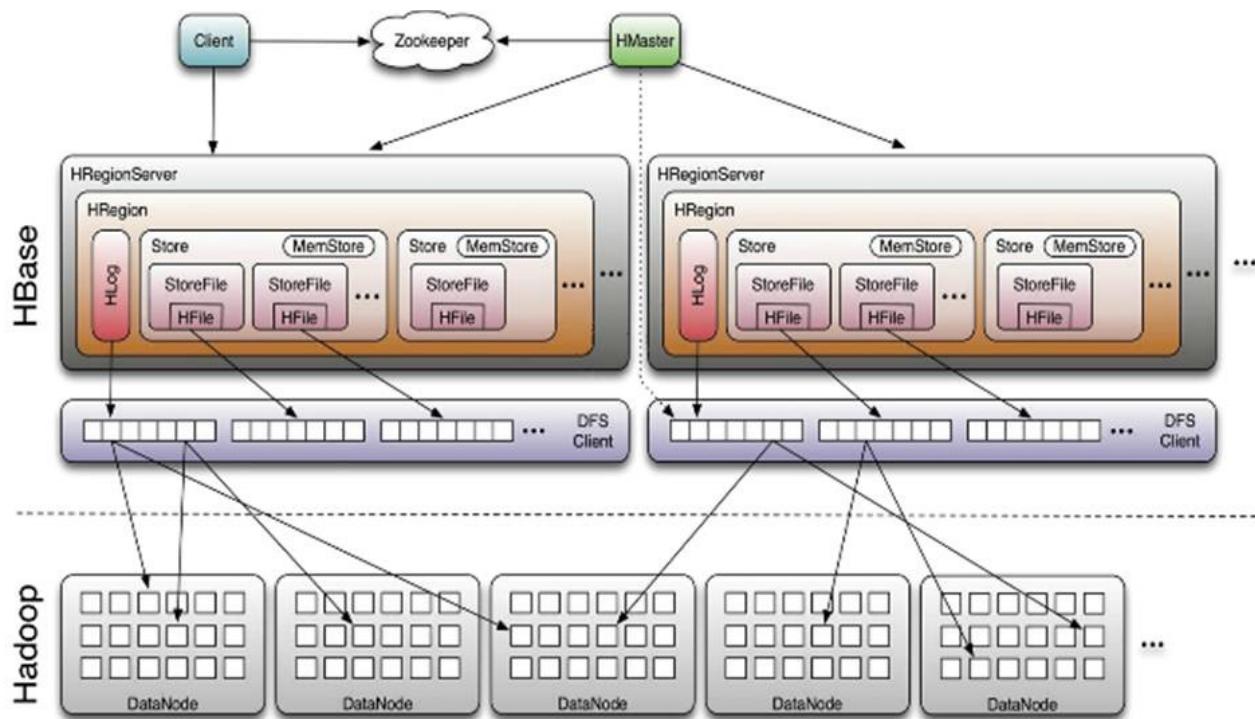
HBase建立在HDFS之上，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据仓库系统。它介于NoSQL和RDBMS之间，仅能通过行键(row key)和行键序列来检索数据，仅支持单行事务(可通过Hive支持来实现多表联合等复杂操作)。主要用来存储非结构化和半结构化的松散数据。与Hadoop一样，HBase目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

**HBase**表一般有这样的特点：

- 大：一个表可以有上亿行，上百万列
- 面向列：面向列(族)的存储和权限控制，列(族)独立检索。
- 稀疏：对于为空(null)的列，并不占用存储空间，因此，表可以设计的非常稀疏。

### 三 HBase 体系架构

HBase的服务器体系结构遵循简单的主从服务器架构。它由HRegion Server和HMaster组成，HMaster负责管理所有的HRegion Server，HBase中所有的服务器都通过ZooKeeper来协调。HBase的体系结构如下图所示。



### 四 Hbase 的二类数据模型

HBASE的二类数据模型是指从逻辑模型与物理模型来了解Hbase的数据模型,表是HBase表达数据的逻辑组织方式,而基于列的存储则是数据在底层的组织方式.

#### 1 逻辑模型

HBase,Cassandra的数据模型非常类似，他们的思想都是来源于Google的Bigtable，因此这三者的数据模型非常类似，唯一不同的就是Cassandra具有Super column family的概念，而Hbase目前我没发现。

在Hbase里面有以下两个主要的概念，Row key,Column Family，我们首先来看看Column family,Column family中文又名“列族”，Column family是在系统启动之前预先定义好的，每一个Column Family都可以根据“限定符”有多个column.下面我们来举个例子就会非常的清晰了。

假如系统中有一个User表，如果按照传统的RDBMS的话，User表中的列是固定的，比如schema 定义了name,age,sex等属性，User的属性是不能动态增加的。但是如果采用列存储系统，比如Hbase，那么我们可以定义User表，然后定义info 列族，User的数据可以分为：info:name = zhangsan,info:age=30,info:sex=male等，如果后来你又想增加另外的属性，这样很方便只需要info:newProperty就可以了。

也许前面的这个例子还不够清晰，我们再举个例子来解释一下，熟悉SNS的朋友，应该都知道有好友Feed，一般设计Feed，我们都是按照“某人在某时做了标题为某某的事情”，但是同时一般我们也会预留一下关键字，比如有时候feed也许需要url，feed需要image属性等，这样来说，feed本身的属性是不确定的，因此如果采用传统的关系数据库将非常麻烦，况且关系数据库会造成一些为null的单元浪费，而列存储就不会出现这个问题，在Hbase里，如果每一个column 单元没有值，那么是占用空间的。下面我们通过两张图来形象的表示这种关系：

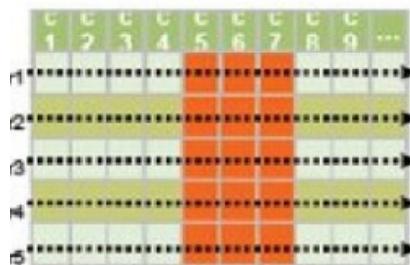
userId	feedId	title	content	image	url	feedDate
1	1	李四加张三为好友	null	null	null	2010.02.10
2	2	张三上传了照片	null	/1.jpeg/photos/zhangsan		2010.02.09

上图是传统的RDBMS设计的Feed表，我们可以看出feed有多少列是固定的，不能增加，并且为null的列浪费了空间。但是我们再看看下图，下图为Hbase，Cassandra,Bigtable的数据模型图，从下图可以看出，Feed表的列可以动态的增加，并且为空的列是不存储的，这就大大节约了空间，关键是Feed这东西随着系统的运行，各种各样的Feed会出现，我们事先没办法预测有多少种Feed，那么我们也就没有办法确定Feed表有多少列，因此Hbase,Cassandra,Bigtable的基于列存储的数据模型就非常适合此场景。说到这里，采用Hbase的这种方式，还有一个非常重要的好处就是Feed会自动切分，当Feed表中的数据超过某一个阈值以后，Hbase会自动为我们切分数据，这样的话，查询就具有了伸缩性，而再加上Hbase的弱事务性的特性，对Hbase的写入操作也将变得非常快。

RowKey	column-family-1		column-family-2			column-family-3
	column-A	column-B	column-A	column-B	column-C	column-A
key001	t2:hk t1:jy		t4:ipad t3:ipod t1:iphone			
key002	t3:style t1:wk		t3:smile t2:jk	t2:wtf	t1:hw	
key003		t1:high				t4:powerful

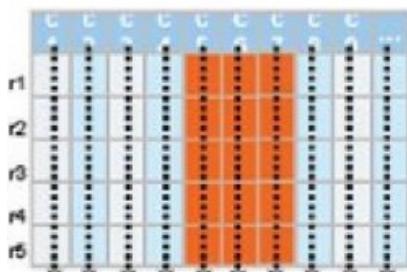
上面说了Column family，那么我之前说的Row key是啥东东，其实你可以理解row key为RDBMS中的某一个行的主键，但是因为Hbase不支持条件查询以及Order by等查询，因此Row key的设计就要根据你系统的查询需求来设计了额。我还拿刚才那个Feed的例子来说，我们一般是查询某个人最新的一些Feed，因此我们Feed的Row key可以有以下三个部分构成，这样以来当我们要查询某个人的最新的Feed就可以指定Start Rowkey为<0><0>，End Rowkey为来查询了，同时因为Hbase中的记录是按照rowkey来排序的，这样就使得查询变得非常快。

### 传统行式数据库



- 数据是按行存储的
- 没有索引的查询使用大量I/O
- 建立索引和物化视图需要花费大量时间和资源
- 面向查询的需求，数据库必须被大量膨胀才能满足性能要求

### 列式数据库



- 数据是按列存储-每一列单独存放
- 数据即是索引
- 指访问查询涉及的列-大量降低系统I/O
- 每一列由一个线索来处理-查询的并发处理
- 数据类型一致，数据特征相似-高效压缩

## Blog 表示例

Row key	Timestamp	article	author
1	1317179000001	article:content= HBase is the <a href="#">Hadoop</a> database. Use it when you need random, realtime read/write access to your Big Data.	
	1317179253210	article:tags= Hadoop,HBase,NoSQL	
	1317179028807	article:title=Head First HBase	
	1317179321857		author:name=hujinjun
	1317180718830		author:nickname=yedu
	1317180070811		author:nickname=一叶凌江
10	...	...	...
100	...	...	...
11	...	...	...
2	...	...	...

## 2 物理模型

虽然在逻辑

## 五 HBase and Hive ?

## 1 HBase 实例及代码解释

要想使用HBase存取数据必须要有两个步骤：

1、建立HBase表

```
create 'test', 'info'
put 'test1', '101', 'info:name', 'wang'
put 'test1', '101', 'info:sex', 'female'

put 'test2', '102', 'info:name', 'zhang'
put 'test2', '102', 'info:sex', 'male'

get 'test', '101'
```

上面创建了一个HBase的test表，用于HBase和数据库做映射使用，同时往这个表里put了两行数据，分别是101和102（row key），info代表列簇，包含了name和sex两列的值

### 2、建立HBase外表

```
create external table hbase_test(id string, name string, sex string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties('hbase.columns.mapping'=':key,info:name,info:sex')
tblproperties('hbase.table.name'='test');
```

上述建立了一张外表，stored by制定HBase的存储格式，with后面是序列化和反序列化，作用是进行map映射，从上面的语句可以看出，将id映射成了key、将name、和sex映射成了info（列簇）

# HBASE的伪分布安装与分布式安装

## 一 伪分布式安装

### 1. 下载解压给权限

可以从官方下载地址下载 HBase 最新版本，推荐 stable 目录下的二进制版本。我下载的是 hbase-1.1.3-bin.tar.gz。确保你下载的版本与你现存的 Hadoop 版本兼容（兼容列表）以及支持的JDK版本（从HBase 1.0.x 已经不支持 JDK 6 了）。

兼容列表:

```
tar -zxvf hbase-1.1.3-bin.tar.gz
sudo mv hbase-1.1.3 /usr/local/hbase
cd /usr/local/
sudo chmod -R 775 hbase
sudo chown -R hadoop:hadoop hbase
```

### 2. 修改HBASE的JDK环境变量

```
sudo nano /usr/local/hbase/conf/hbase-env.sh
export JAVA_HOME=/usr/lib/jvm/
```

### 3. 修改hbase-site.xml

```
hadoop@Master:/usr/local/hbase/conf$ sudo nano hbase-site.xml

<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://Master:9000/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
</configuration>
```

## 4.启动验证

- 启动hbase

```
start-hbase.sh
```

- 进入hbase shell

## 1.HBASE的伪分布安装与分布式安装

```
hadoop@Master:/usr/local/hbase/bin$ ./hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands

.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.1.3, r72bc50f5fafeb105b2139e42bbe3d61ca724989, Sat Jan 16 18:29:00 PST 2016

hbase(main):001:0>

hadoop@Master:/usr/local/hbase/bin$ jps
1601 ResourceManager
1430 SecondaryNameNode
2374 HRegionServer
1883 JobHistoryServer
3037 Jps
2253 HMaster
1246 NameNode
2190 HQuorumPeer
hadoop@Master:/usr/local/hbase/bin$
```

The screenshot shows the Apache HBase Master interface running on port 192.168.1.80. The top navigation bar includes links for Home, Table Details, Local Logs, Log Level, Debug Dump, Metrics Dump, and HBase Configuration. The main content area is divided into several sections:

- Region Servers**: A table showing two servers: master\_16201,1455905014114 (Start time: Sat Feb 20 02:03:34 CST 2016, Requests Per Second: 0, Num. Regions: 2) and Total:1 (Requests Per Second: 0, Num. Regions: 2).
- Backup Masters**: A table showing one backup master.
- Tables**: A table showing User Tables, System Tables, and Snapshots.
- Tasks**: A section showing monitored tasks, non-RPC tasks, RPC handler tasks, active RPC calls, client operations, and a View as JSON link. It notes that no tasks are currently running on this node.

## 5. 尝试一下**Thrift**通信服务

这里我们尝试使用HBase 的 Thrift API，用Python和HBase进行简单交互。首先启动HBase的Thrift服务：

```
hadoop@Master:/usr/local/hbase/bin$ ./hbase-daemon.sh start thrift
starting thrift, logging to /usr/local/hbase/bin/../logs/hbase-hadoop-thrift-Master.out
```

然后安装Python的happybase模块，HBase是对 HBase的Thrift接口的一个简单包装：

```
chuguaningdeMBP:~ chuguangming$ sudo pip install happybase
```

然后启动ipython，如果没有ipython，请通过pip安装：

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
Type "copyright", "credits" or "license" for more information.

IPython 3.2.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
```

```
In [1]: import happybase
```

```
In [2]: connection = happybase.Connection('localhost')
```

```
In [3]: connection.tables()
```

```
Out[3]: []
```

```
In [4]: families = {'basic':dict(max_versions=3), 'detail':dict(max_versions=1000), 'comment':dict(max_versions=1000), 'answer':dict(max_versions=1000), 'follower':dict(max_versions=1000)}
```

```
In [5]: connection.create_table('question', families)
```

```
In [6]: connection.tables()
```

```
Out[6]: ['question']
```

```
In [7]:
```

## 二 分布式安装

### 1 软件环境

OS:Linux Master 3.19.0-25-generic #26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86\_64 x86\_64 x86\_64 GNU/Linux

Java:java version "1.8.0\_65" Java(TM) SE Runtime Environment (build 1.8.0\_65-b17) Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

Hadoop:Hadoop 2.6.0

Hbase:hbase-1.1.3

### 2 集群部署机器:

IP	HostName	Master	RegionServer
192.168.1.80	Master	yes	no
192.168.1.82	Slave1	no	yes
192.168.1.84	Slave2	no	yes

### 3 准备:

假设你已经安装部署好了 Hadoop 集群和 Java，可以参考以前的部署文章。

### 4 Master的安装:

#### 1基本安装

```
tar -zxvf hbase-1.0.0-bin.tar.gz
sudo mv hbase-1.0.0 /usr/local/hbase
cd /usr/local/
sudo chmod -R 775 hbase
sudo chown -R hadoop:hadoop: hbase
```

#### 2修改\$JAVA\_HOME为jdk安装目录

```
usr/local/hbase/conf$ sudo nano hbase-env.sh  
export JAVA_HOME=/usr/lib/jvm/
```

### 3修改hbase-site.xml

```
/usr/local/hbase/conf/hbase-site.xml  
  
<configuration>  
  <property>  
    <name>hbase.rootdir</name>  
    <value>hdfs://Master:9000/hbase</value>  
  </property>  
  <property>  
    <name>hbase.cluster.distributed</name>  
    <value>true</value>  
  </property>  
  <property>  
    <name>hbase.zookeeper.quorum</name>  
    <value>Master,Slave1,Slave2</value>  
  </property>  
  <property>  
    <name>hbase.zookeeper.property.dataDir</name>  
    <value>/home/hadoop</value>  
  </property>  
</configuration>
```

其中第一个属性指定本机的hbase的存储目录，必须与Hadoop集群的core-site.xml文件配置保持一致；第二个属性指定hbase的运行模式，true代表全分布模式；第三个属性指定Zookeeper管理的机器，一般为奇数个；第四个属性是数据存放的路径。这里我使用的默认的HBase自带的Zookeeper。

### 4配置regionservers

```
Slave1  
Slave2
```

`regionservers`文件列出了所有运行`hbase`的机器（即`HRegionServer`）。此文件的配置和Hadoop中的`slaves`文件十分相似，每行指定一台机器的主机名。当`HBase`启动的时候，会将此文件中列出的所有机器启动。关闭时亦如此。

### 5修改 `ulimit` 限制

`HBase`会在同一时间打开大量的文件句柄和进程，超过 Linux 的默认限制，导致可能会出现错误。所以编辑`/etc/security/limits.conf`文件，添加以下两行，提高能打开的句柄数量和进程数量。注意将`hadoop`改成你运行 `HBase` 的用户名。

```
hadoop -      nfile  65535  
hadoop -      nproc   32000
```

还需要在`/etc/pam.d/common-session`加上这一行：

```
session required pam_limits.so
```

否则在`/etc/security/limits.conf`上的配置不会生效。

最后还要注销（`logout`或者`exit`）后再登录，这些配置才能生效！使用`ulimit -n -u`命令查看最大文件和进程数量是否改变了。记得在每台安装 `HBase` 的机器上运行哦。

## 5 Slave上面的操作

基本上把以上步骤重复一下就可以了。

在 Master 节点上执行：

```
cd /usr/local  
hadoop@hadoopmaster:/usr/local$ sudo tar cvfz ~/hbase.tar.gz ./h  
base  
scp ~/hbase.tar.gz hadoop@hadoopslave1:/home/hadoop/  
scp ~/hbase.tar.gz hadoop@hadoopslave2:/home/hadoop/
```

在 Slave 节点上执行：

```
hadoop@hadoopslave1:/usr/local$ sudo tar xvfz ~/hbase.tar.gz -C /usr/local/
hadoop@hadoopslave1:/usr/local$ sudo chown -R hadoop:hadoop /usr/local/hbase/
hadoop@hadoopslave1:/usr/local$ sudo chmod -R 775 /usr/local/hbase/
```

## 6 运行 HBase

在master上运行

```
start-dfs.sh
start-yarn.sh
mr-jobhistory-daemon.sh start historyserver
start-hbase.sh
```

## 7 验证 HBase 成功安装

在 master 运行 jps 应该会有HMaster进程。在各个 slave 上运行jps 应该会有 HQuorumPeer,HRegionServer两个进程。在浏览器中输入 <http://Master:16010> 可以看到 HBase Web UI 。



### Master Master

#### Region Servers

Base Stats	Memory	Requests	Storefiles	Compactions	
ServerName		Start time		Requests Per Second	Num. Regions
slave1,16020,1456151341240		Mon Feb 22 22:29:01 CST 2016		0	0
slave2,16020,1456151342618		Mon Feb 22 22:29:02 CST 2016		0	1
Total:2				0	1

#### Dead Region Servers

ServerName	Stop time
master,16201,1455904625948	Mon Feb 22 22:29:18 CST 2016
master,16201,1455905014114	Mon Feb 22 22:29:18 CST 2016
master,16201,1455906782787	Mon Feb 22 22:29:18 CST 2016
Total: servers: 3	



# HBASE常用的Shell命令

## 一 表的管理

### 1进入hbase shell console

```
$HBASE_HOME/bin/hbase shell
```

如果有kerberos认证，需要事先使用相应的keytab进行一下认证（使用kinit命令），认证成功之后再使用hbase shell进入可以使用whoami命令可查看当前用户

```
hbase(main)> whoami
```

### 2 查看有哪些表

```
hbase(main)> list
```

### 3 创建表

语法：create <table>, {NAME => <family>, VERSIONS => <VERSIONS>}  
例如：创建表t1，有两个family name：f1，f2，且版本数均为2

```
hbase(main)> create 't1',{NAME => 'f1', VERSIONS => 2},{NAME => 'f2', VERSIONS => 2}
```

创建表t1,列族为f1,列族版本号为5,命令如下

```
hbase(main):002:0> create 't1',{NAME=>'f1',VERSIONS=>5}
```

或者使用如下等价的命令

```
hbase(main):003:0> create 't2','f1','f2','f3'
```

创建表**t1**,将表依据分割算法**HexStringSplit**分布在**15个Region**里,命令如下:

```
hbase(main):006:0* create 't3','f1',{NUMREGIONS=>15,SPLITALGO=>'  
HexStringSplit'}
```

创建表**t1**,指定切分点,命令如下:

```
hbase(main):007:0> create 't4','f1',{SPLITS=>['10','20','30','40'  
}]
```

**put**:向表/行/列指定的单元格添加数据

向表**t1**中行**row1**,列**f1:1**,添加数据**value1**,时间戳为**142222222222**,命令如下:

```
hbase(main):009:0> put 't1','row1','f1:1','value1',55555434344
```

## 4 删除表

分两步：首先**disable**，然后**drop**,例如：删除表**t1**

```
hbase(main)> disable 't1'  
hbase(main)> drop 't1'
```

## 5 查看表的结构

语法：**describe <table>**

例如：查看表**t1**的结构

```
hbase(main)> describe 't1'
```

## 6 修改表结构

修改表结构必须先disable

语法：alter 't1', {NAME => 'f1'}, {NAME => 'f2', METHOD => 'delete'}

例如：修改表test1的cf的TTL为180天

```
hbase(main)> disable 'test1'
```

```
hbase(main)> alter 'test1',{NAME=>'body',TTL=>'15552000'},{NAME=>'meta', TTL=>'15552000'}
```

```
hbase(main)> enable 'test1'
```

## 二 权限管理

### 1 分配权限

语法 : grant <user> <permissions> <table> <column family> <column qualifier>

参数后面用逗号分隔权限用五个字母表示：“RWXCA”。

READ('R'), WRITE('W'), EXEC('X'), CREATE('C'), ADMIN('A')

例如，给用户'test'分配对表t1有读写的权限，

```
hbase(main)> grant 'test','RW','t1'
```

### 2 查看权限

语法 : user\_permission <table>

例如，查看表t1的权限列表

```
hbase(main)> user_permission 't1'
```

### 3 收回权限

与分配权限类似，语法：revoke <user> <table> <column family> <column qualifier>

例如，收回test用户在表t1上的权限

```
hbase(main)> revoke 'test','t1'
```

## 三表数据的增删改查

### 1 添加数据

语法：put <table>,<rowkey>,<family:column>,<value>,<timestamp>  
例如：给表t1的添加一行记录：rowkey是rowkey001，family name : f1，column name : col1，value : value01，timestamp : 系统默认  
hbase(main)> put 't1','rowkey001','f1:col1','value01'  
用法比较单一。

### 2 查询某行记录

语法：get <table>,<rowkey>,[<family:column>,....]  
例如：查询表t1，rowkey001中的f1下的col1的值  
hbase(main)> get 't1','rowkey001','f1:col1'  
或者：  
hbase(main)> get 't1','rowkey001',{COLUMN=>'f1:col1'}  
查询表t1，rowkey002中的f1下的所有列值  
hbase(main)> get 't1','rowkey001'

### 3 扫描表

语法：scan <table>,{COLUMNS => [ <family:column>,.... ], LIMIT => num}  
另外，还可以添加STARTROW、TIMERANGE和FILER等高级功能  
例如：扫描表t1的前5条数据  
hbase(main)> scan 't1',{LIMIT=>5}

### 4 查询表中的数据行数

语法：count <table>, {INTERVAL => intervalNum, CACHE => cacheNum}  
INTERVAL设置多少行显示一次及对应的rowkey，默认1000；CACHE每次去取的缓存区大小，默认是10，调整该参数可提高查询速度  
例如，查询表t1中的行数，每100条显示一次，缓存区为500  
hbase(main)> count 't1', {INTERVAL => 100, CACHE => 500}

### 三 删除数据

#### 1 删除行中的某个列值

语法：delete <table>, <rowkey>, <family:column> , <timestamp>, 必须指定列名  
例如：删除表t1，rowkey001中的f1:col1的数据  
hbase(main)> delete 't1', 'rowkey001', 'f1:col1'  
注：将删除改行f1:col1列所有版本的数据

#### 2 删除行

语法：deleteall <table>, <rowkey>, <family:column> , <timestamp>  
，可以不指定列名，删除整行数据  
例如：删除表t1，rowk001的数据  
hbase(main)> deleteall 't1', 'rowkey001'

#### 3 删除表中的所有数据

语法： truncate <table>  
其具体过程是：disable table -> drop table -> create table  
例如：删除表t1的所有数据  
hbase(main)> truncate 't1'

### 四 Region管理

## 1 移动region

语法：move 'encodeRegionName', 'ServerName'

encodeRegionName指的regionName后面的编码，ServerName指的是master-stat us的Region Servers列表

示例

```
hbase(main)>move '4343995a58be8e5bbc739af1e91cd72d', 'db-41.xxx.xxx.org,60020,1390274516739'
```

## 2 开启/关闭region

语法：balance\_switch true|false

```
hbase(main)> balance_switch
```

## 3 手动split

语法：split 'regionName', 'splitKey'

## 4 手动触发major compaction

语法：

Compact all regions in a table:

```
hbase> major_compact 't1'
```

Compact an entire region:

```
hbase> major_compact 'r1'
```

Compact a single column family within a region:

```
hbase> major_compact 'r1', 'c1'
```

Compact a single column family within a table:

```
hbase> major_compact 't1', 'c1'
```

### 3. 基于**HBASE**的**Java**开发

## 4. 基于**HBASE**的**Python**开发

## 5.HBASE与传统数据库的区别

### 1 主要区别

1. Hbase适合大量插入同时又有读的情况
2. Hbase的瓶颈是硬盘传输速度，Oracle的瓶颈是硬盘寻道时间。

Hbase本质上只有一种操作，就是插入，其更新操作是插入一个带有新的时间戳的行，而删除是插入一个带有插入标记的行。其主要操作是收集内存中一批数据，然后批量的写入硬盘，所以其写入的速度主要取决于硬盘传输的速度。Oracle则不同，因为他经常要随机读写，这样硬盘磁头需要不断的寻找数据所在，所以瓶颈在于硬盘寻道时间。

3. Hbase很适合寻找按照时间排序top n的场景
4. 索引不同造成行为的差异。
5. Oracle 这样的传统数据库既可以做OLTP又可以做OLAP，但在某种极端的情况下(负荷十分之大)，就不适合了。

### 2 Hbase的局限：

1. 只能做简单的Key value查询，复杂的sql统计做不到。
2. 只能在row key上做快速查询。

### 3 传统数据库的行式存储



在数据分析的场景里面，我们经常是以某个列作为查询条件，返回的结果经常也只是某些列，不是全部的列。行式数据库在这种情况下的I/O性能会很差，以Oracle为例，Oracle会有一个很大的数据文件，在这个数据文件中，划分了很多block，然后在每个block中放入行，行是一行一行放进去，挤在一起，然后把block塞满，当然也会预留一些空间，用于将来update。这种结构的缺点是：当我们读某个列的时候，比如我们只需要读红色标记的列的时候，不能只读这部分数据，我必须把整个

block读取到内存中，然后再把这些列的数据取出来，换句话说，我为了读表中某些列的数据，我必须把整个列的行读完，才可以读到这些列。如果这些列的数据很少，比如1T的数据中只占了100M,为了读100M数据却要读取1TB的数据到内存中去，则显然是不划算。

## 3 B+索引

Oracle中采用的数据访问技术主要是B数索引：



从树的跟节点出发，可以找到叶子节点，其记录了key值对应的那行的位置。对B树的操作：

B树插入——分裂节点

B数删除——合并节点

## 4 列式存储



同一个列的数据会挤在一起，比如挤在block里，当我需要读某个列的时候，值需要把相关的文件或块读到内存中去，整个列就会被读出来，这样I/O会少很多。

同一个列的数据的格式比较类似，这样可以做大幅度的压缩。这样节省了存储空间，也节省了I/O,因为数据被压缩了，这样读的数据量随之也少了。

行式数据库适合OLTP，反倒列式数据库不适合OLTP。

具体百度知道参考:[OLTP OLAP](#)

当今的数据处理大致可以分成两大类：联机事务处理OLTP（on-line transaction processing）、联机分析处理OLAP（On-Line Analytical Processing）。OLTP是传统的关系型数据库的主要应用，主要是基本的、日常的事务处理，例如银行交易。OLAP是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。

### OLTP:

也称为面向交易的处理系统，其基本特征是顾客的原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。

这样做的最大优点是可以即时地处理输入的数据，及时地回答。也称为实时系统(Real time System)。衡量联机事务处理系统的一个重要性能指标是系统性能，具体体现为实时响应时间(Response Time)，即用户在终端上送入数据之后，到计算机对这个请求给出答复所需要的时间。OLTP是由数据库引擎负责完成的。

OLTP数据库旨在使事务应用程序仅写入所需的数据，以便尽快处理单个事务。

### **OLAP:**

简写为OLAP,随着数据库技术的发展和应用，数据库存储的数据量从20世纪80年代的兆(M)字节及千兆(G)字节过渡到现在的兆兆(T)字节和千兆兆(P)字节，同时，用户的查询需求也越来越复杂，涉及的已不仅是查询或操纵一张关系表中的一条或几条记录，而且要对多张表中千万条记录的数据进行数据分析和信息综合，关系数据库系统已不能全部满足这一要求。在国外，不少软件厂商采取了发展其前端产品来弥补关系数据库管理系统支持的不足，力图统一分散的公共应用逻辑，在短时间内响应非数据处理专业人员的复杂查询要求。

联机分析处理(OLAP)系统是数据仓库系统最主要的应用，专门设计用于支持复杂的分析操作，侧重对决策人员和高层管理人员的决策支持，可以根据分析人员的要求快速、灵活地进行大数据量的复杂查询处理，并且以一种直观而易懂的形式将查询结果提供给决策人员，以便他们准确掌握企业(公司)的经营状况，了解对象的需求，制定正确的方案。

## **4 BigTable的LSM (Log Struct Merge) 索引**



在Hbase中日志即数据，数据就是日志，他们是一体化的。为什么这么说了，因为Hbase的更新时插入一行，删除也是插入一行，然后打上删除标记，则不就是日志吗？

在Hbase中，有Memory Store,还有Store File，其实每个Memory Store和每个Store File就是对每个列族附加上一个B+树(有点像Oracle的索引组织表，数据和索引是一体化的)，也就是图的下面是列族，上面是B+树，当进行数据的查询时，首先会在内存中memory store的B+树中查找，如果找不到，再到Store File中去找。

如果找的行的数据分散在好几个列族中，那怎么把行的数据找全呢？那就需要找好几个B+树，这样效率就比较低了。所以尽量让每次insert的一行的列族都是稀疏的，只在某一个列族上有值，其他列族没有值，

## 6.HBASE安装疑难杂症

### 错误表现-SLF4J: Class path contains multiple SLF4J bindings.

错误表现：

```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/hbase/lib/slf4j-log4j12-1  
.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/hadoop/share/hadoop/commo  
n/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder  
.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for  
an explanation.
```

含义为：

发生jar包冲突了：

分别为：

```
file:/usr/hbase/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/impl/Stat  
icLoggerBinder.class  
file:/usr/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar  
!/org/slf4j/impl/StaticLoggerBinder.class
```

移除其中一个jar包即可

解决方案：

使用下面命令：

```
/usr/hbase/lib rm slf4j-log4j12-1.6.4.jar
```

问题解决

## 7.Hive与Hbase的区别

### 1 两者分别是什么？

Apache Hive是一个构建在Hadoop基础设施之上的数据仓库。通过Hive可以使用HQL语言查询存放在HDFS上的数据。HQL是一种类SQL语言，这种语言最终被转化为Map/Reduce。虽然Hive提供了SQL查询功能，但是Hive不能够进行交互查询--因为它只能够在Hadoop上批量的执行Hadoop。

Apache HBase是一种Key/Value系统，它运行在HDFS之上。和Hive不一样，Hbase能够在它的数据库上实时运行，而不是运行MapReduce任务。Hbase被分区为表格，表格又被进一步分割为列簇。列簇必须使用schema定义，列簇将某一类型列集合起来（列不要求schema定义）。例如，“message”列簇可能包含：“to”，“from” “date”，“subject”，和“body”。每一个 key/value 对在 Hbase 中被定义为一个 cell，每一个key由row-key，列簇、列和时间戳。在Hbase中，行是key/value映射的集合，这个映射通过row-key来唯一标识。Hbase利用Hadoop的基础设施，可以利用通用的设备进行水平的扩展。

### 2 两者的特点

Hive帮助熟悉SQL的人运行MapReduce任务。因为它是JDBC兼容的，同时，它也能够和现存的SQL工具整合在一起。运行Hive查询会花费很长时间，因为它会默认遍历表中所有的数据。虽然有这样的缺点，一次遍历的数据量可以通过Hive的分区机制来控制。分区允许在数据集上运行过滤查询，这些数据集存储在不同的文件夹内，查询的时候只遍历指定文件夹（分区）中的数据。这种机制可以用来，例如，只处理在某一个时间范围内的文件，只要这些文件名中包括了时间格式。

HBase通过存储key/value来工作。它支持四种主要的操作：增加或者更新行，查看一个范围内的cell，获取指定的行，删除指定的行、列或者是列的版本。版本信息用来获取历史数据（每一行的历史数据可以被删除，然后通过Hbase compactions就可以释放出空间）。虽然HBase包括表格，但是schema仅仅被表格和列簇所要求，列不需要schema。Hbase的表格包括增加/计数功能。

### 3 限制

Hive目前不支持更新操作。另外，由于hive在hadoop上运行批量操作，它需要花费很长的时间，通常是几分钟到几个小时才可以获取到查询的结果。Hive必须提供预先定义好的schema将文件和目录映射到列，并且Hive与ACID不兼容。

HBase查询是通过特定的语言来编写的，这种语言需要重新学习。类SQL的功能可以通过Apache Phoenix实现，但这是以必须提供schema为代价的。另外，Hbase也并不是兼容所有的ACID特性，虽然它支持某些特性。最后但不是最重要的--为了运行Hbase，Zookeeper是必须的，zookeeper是一个用来进行分布式协调的服务，这些服务包括配置服务，维护元信息和命名空间服务。

## 4 应用场景

Hive适合用来对一段时间内的数据进行分析查询，例如，用来计算趋势或者网站的日志。Hive不应该用来进行实时的查询。因为它需要很长时间才可以返回结果。

Hbase非常适合用来进行大数据的实时查询。Facebook用Hbase进行消息和实时的分析。它也可以用来统计Facebook的连接数。

## 总结

Hive和Hbase是两种基于Hadoop的不同技术--Hive是一种类SQL的引擎，并且运行MapReduce任务，Hbase是一种在Hadoop之上的NoSQL的Key/vale数据库。当然，这两种工具是可以同时使用的。就像用Google来搜索，用FaceBook进行社交一样，Hive可以用来进行统计查询，HBase可以用来进行实时查询，数据也可以从Hive写到Hbase，设置再从Hbase写回Hive。

## 第十二章 **HBASE** 实战

# 1.HBASE基于Java开发

## 使用Java操作HBASE(增删查改)

```
package com.chu;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.Delete;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.HTablePool;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.filter.Filter;
import org.apache.hadoop.hbase.filter.FilterList;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.util.Bytes;

public class CreateHBaseTableDemo
{
    public static Configuration configuration;
```

```
static
{
    configuration = HBaseConfiguration.create();
    configuration.set("hbase.zookeeper.property.clientPort",
"2181");
    configuration.set("hbase.zookeeper.quorum", "192.168.1.1
59");
    configuration.set("hbase.master", "192.168.1.159:600000"
);
}

public static void main(String[] args)
{
    //createTable("HBaseDemoTable1");
    //insertData("HBaseDemoTable1");
    //QueryAll("HBaseDemoTable1");
    //QueryByCondition1("HBaseDemoTable1");
    //QueryByCondition2("HBaseDemoTable1");
    QueryByCondition3("HBaseDemoTable1");
    //deleteRow("wujintao", "abcdef");
    //deleteByCondition("wujintao", "abcdef");
}

/**
 * 创建表
 *
 * @param tableName
 */
public static void createTable(String tableName)
{
    System.out.println("start create table .....");
    try
    {
        HBaseAdmin hBaseAdmin = new HBaseAdmin(configuration
);
        if (hBaseAdmin.tableExists(tableName))
        {
            // 如果存在要创建的表，那么先删除，再创建
            hBaseAdmin.disableTable(tableName);
        }
    }
}
```

```
        hBaseAdmin.deleteTable(tableName);
        System.out.println(tableName + " is exist, delete
....");
    }
    HTableDescriptor tableDescriptor = new HTableDescriptor(tableName);
    tableDescriptor.addFamily(new HColumnDescriptor("col
umn1"));
    tableDescriptor.addFamily(new HColumnDescriptor("col
umn2"));
    tableDescriptor.addFamily(new HColumnDescriptor("col
umn3"));
    hBaseAdmin.createTable(tableDescriptor);
} catch (MasterNotRunningException e)
{
    e.printStackTrace();
} catch (ZooKeeperConnectionException e)
{
    e.printStackTrace();
} catch (IOException e)
{
    e.printStackTrace();
}
System.out.println("end create table .....");
}

/**
 * 插入数据
 *
 * @param tableName
 */
public static void insertData(String tableName)  {

    System.out.println("start insert data .....");
    try
    {
        HTable table = new HTable(configuration, tableName);
    }
}
```

```
        Put put = new Put(Bytes.toBytes("45678hyhyju"));
        put.add(Bytes.toBytes("column1"),
                Bytes.toBytes("email"),
                Bytes.toBytes("5211486@qq.com"));
        table.put(put);// 放入表
        table.close();// 释放资源
    } catch (IOException e)
    {
        e.printStackTrace();
    }
    System.out.println("end insert data .....");

}

/***
 * 删除一张表
 *
 * @param tableName
 */
public static void dropTable(String tableName) {
    try {
        HBaseAdmin admin = new HBaseAdmin(configuration);
        admin.disableTable(tableName);
        admin.deleteTable(tableName);
    } catch (MasterNotRunningException e) {
        e.printStackTrace();
    } catch (ZooKeeperConnectionException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/***
 * 根据 rowkey删除一条记录
 *
 * @param tablename

```

```

    * @param rowkey
    */
    public static void deleteRow(String tablename, String rowkey
) {
    try {
        HTable table = new HTable(configuration, tablename);
        List list = new ArrayList();
        Delete d1 = new Delete(rowkey.getBytes());
        list.add(d1);

        table.delete(list);
        System.out.println("删除行成功!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * 组合条件删除
 *
 * @param tablename
 * @param rowkey
 */
public static void deleteByCondition(String tablename, String rowkey) {
    //目前还没有发现有效的API能够实现 根据非rowkey的条件删除 这个功能
能，还有清空表全部数据的API操作
}

/**
 * 查询所有数据
 *
 * @param tableName
 */
public static void QueryAll(String tableName) {

```

```

try
{
    HTable table = new HTable(configuration, tableName);
    ResultScanner rs = table.getScanner(new Scan());
    for (Result r : rs)
    {
        System.out.println("获得到rowkey:" + new String(r
.getRow()));
        for (KeyValue keyValue : r.raw())
        {
            System.out.println("列：" + new String(keyVal
ue.getFamily()))
                + "====值：" + new String(keyValue.get
Value()));
        }
    }
}
catch (IOException e)
{
    e.printStackTrace();
}

/**
 * 单条件查询,根据rowkey查询唯一一条记录
 *
 * @param tableName
 */
public static void QueryByCondition1(String tableName) {

    try {
        HTable table = new HTable(configuration, tableName);
        Get scan = new Get("45678hyhyjju".getBytes());// 根据
rowkey查询
        Result r = table.get(scan);
        System.out.println("获得到rowkey:" + new String(r.get
Row()));
    }
}

```

```

        for (KeyValue keyValue : r.raw()) {
            System.out.println("列：" + new String(keyValue.getFamily())
                + "====值：" + new String(keyValue.getValue()));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * 单条件按查询，查询多条记录
 *
 * @param tableName
 */
public static void QueryByCondition2(String tableName) {

    try {
        HTable table = new HTable(configuration, tableName);
        Filter filter = new SingleColumnValueFilter(Bytes
            .toBytes("column1"), null, CompareOp.EQUAL,
        Bytes
            .toBytes("123456")); // 当列column1的值为aaa时
        进行查询

        Scan s = new Scan();
        s.setFilter(filter);
        ResultScanner rs = table.getScanner(s);
        for (Result r : rs) {
            System.out.println("获得到rowkey：" + new String(r
                .getRow()));
            for (KeyValue keyValue : r.raw()) {
                System.out.println("列：" + new String(keyValue.getFamily())
                    + "====值：" + new String(keyValue.getValue()));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```
    }

}

/***
 * 组合条件查询
 *
 * @param tableName
 */
public static void QueryByCondition3(String tableName) {

    try {
        HTable table = new HTable(configuration, tableName);

        List<Filter> filters = new ArrayList<Filter>();

        Filter filter1 = new SingleColumnValueFilter(Bytes
            .toBytes("column1"), null, CompareOp.EQUAL,
        Bytes
            .toBytes("123456"));
        filters.add(filter1);

        Filter filter2 = new SingleColumnValueFilter(Bytes
            .toBytes("column2"), null, CompareOp.EQUAL,
        Bytes
            .toBytes("123456"));
        filters.add(filter2);

        Filter filter3 = new SingleColumnValueFilter(Bytes
            .toBytes("column3"), null, CompareOp.EQUAL,
        Bytes
            .toBytes("c2g111"));
        filters.add(filter3);

        FilterList filterList1 = new FilterList(filters);

        Scan scan = new Scan();
        scan.setFilter(filterList1);
        ResultScanner rs = table.getScanner(scan);
        for (Result r : rs) {
```

```
        System.out.println("获得到rowkey:" + new String(r
.getRow()));
        for (KeyValue keyValue : r.raw()) {
            System.out.println("列：" + new String(keyVal
ue.getFamily())
                + "====值：" + new String(keyValue.ge
tValue()));
        }
    }
    rs.close();

} catch (Exception e) {
    e.printStackTrace();
}

}

}
```

### MAVEN库

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
tt://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.chu</groupId>
    <artifactId>HBaseDemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.apache.hive/
hive-jdbc -->
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-jdbc</artifactId>
            <version>2.1.0</version>
        </dependency>
    
```

```
<!-- https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hibernate-jpa-2.0-api -->
<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.0-api</artifactId>
    <version>1.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-exec</artifactId>
    <version>2.1.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-client -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>1.1.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase</artifactId>
    <version>1.1.5</version>
    <type>pom</type>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-common -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-common</artifactId>
    <version>1.1.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-server -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-server</artifactId>
    <version>1.1.5</version>
```

```
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase
/hbase-protocol -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-protocol</artifactId>
    <version>1.1.5</version>
</dependency>

</dependencies>
<repositories>
    <repository>
        <id>jboss</id>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </repository>
</repositories>
</project>
```

# 一 整合SQL引擎层

1. NOSQL(Not only SQL 非关系型数据库)的特性之一是不使用SQL作为查询语言,本节简单介绍NOSQL定义,为何NOSQL上定义SQL引擎,以及现有基于HBASE的SQL引擎的具体实现
2. NOSQL是不同于传统关系型数据库的数据库系统的统称.两者有很多显著的不同点,其中最重要的是NOSQL不使用SQL作为查询语言.其数据存储可以不需要固定的表格模式,也经常会避免使用SQL的JOIN操作,一般具备水平扩展的特征.NOSQL的实现具有两个特征:使用硬盘或者把随机存储器作为存储媒体

## 1 把SQL整合到HBASE的原因

现有的SQL解决方案通常都不是水平可伸缩的,因此当数据量变大时会遇到阻力.我们已经知道NOSQL区别于关系型数据库的一点就是NOSQL不使用SQL作为查询语言,至于为何在NOSQL数据存储HBASE上提供SQL接口,有如下三个原因:

1. 使用诸如SQL这样易于理解的语言,使人们能够更加轻松地使用HBASE
2. 使用诸如SQL这样更高层次的语言来编写,减少了编码的代码量.
3. 执行查询时,在数据访问与运行时执行之间加上SQL这样一层抽象可以进行大量优化.

## 2 基于HBASE的SQL引擎实现

现阶段业内有一些HBASE SQL引擎层的尝试

### Hive 整合HBASE

Hive与HBASE的整合功能从Hive0.6版本开始出现,利用两者对外的API接口互相通信,通信主要依赖hive\_hbase-handler.jar工具包(Hive Storage Handlers).由于HBASE有一次比较大的版本变动,所以并不是每个版本的Hive都能和现有的HBASE版本进行整合,因此对版本的范围要求比较严格.

### Phoenix

Phoenix由Salesforce.com开源,是构建在Apache Hbase之上一个SQL中间层,可以让开发者在HBASE上执行SQL查询.Phoenix完全使用Java开发,并且提供一个客户端可嵌入的JDBC驱动.对于简单的低延迟查询,其量级为毫秒,对于百万级别的行数来说,其量级为秒.Phoenix并不像HBase那样用于map-reduce job,而是通过标准化的语言来访问HBASE数据.根据项目创建者所述,对于10万到100万行的简单查询来说,Phoenix要胜过Hive.对于使用了HBASE API,协同处理器及自定义过滤器的Impala与OpenTSDB来说,进行相似的查询Phoenix的速度也要更快一些.

## Impala

Cloudera发布实时查询开源项目Impala,经多款产品实测表明,比原来基于MapReduce的Hive SQL查询速度提升了3-90倍,Impala是google dremel的模仿,但在SQL功能上青出于蓝.

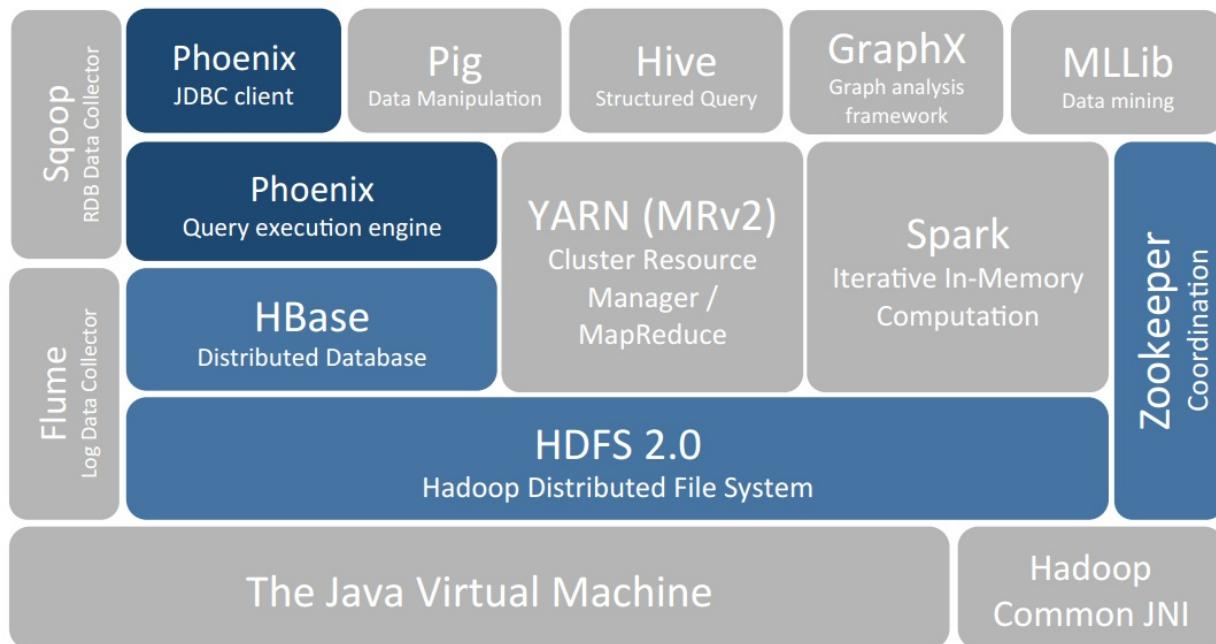
Impala采用与Hive相同的元数据,SQL语法 ODBC驱动和用户接口,但是对于平台有一定的局限性在CDH上表现良好,在其他平台上兼容性不好.

## 二 Phoenix的安装与使用

### 1 概述

Apache Phoenix是构建在HBase之上关系型数据库层,作为内嵌的客户端JDBC驱动用以对HBase中的数据进行低延迟访问。Apache Phoenix会将用户编写的sql查询编译为一系列的scan操作,最终产生通用的JDBC结果集返回给客户端。数据表的元数据存储在HBase的表中被会标记版本号,所以进行查询的时候会自动选择

正确的schema。直接使用HBase的API，结合协处理器（coprocessor）和自定义的过滤器的话，小范围的查询在毫秒级响应，千万数据的话响应速度为秒级。



## 2 Phoenix的特点

1. 嵌入式的JDBC驱动,实现了大部分的Java.sql接口,包括元数据API
2. 可以通过多行键/值单元对列进行建模
3. 完善的查询支持,优化过scan
4. DDL支持:通过CREATE TABLE DROP TABLE ALTER TABLE实现
5. 版本化的模式仓库:当写入数据时,快照查询会使用适当的模式
6. DML支持:用于逐行插入的UPSERT VALUES ,用于相同或不同表之间大量数据传输的UPSERT SELECT,用于删除行的DELETE
7. 通过客户端的批处理实现的有限事务支持
8. 紧跟ANSI SQL标准

不同于Hive On HBASE的方式,Phoenix将Query Plan直接使用HBASE API实现,规避MapReduce框架,减少查询的时间延迟.Phoenix中SQL Query Plan的执行,基本上是通过构建一系列HBASE Scan来完成的.

目前支持简单的表创建,修改,数据删除过滤查询等SQL语句,从语法上看,不支持多表操作,由于不支持多表联合类的操作如各种Join等,所以在Where部分也就不能做多表比较.基于HBASE的timestamp和不限制修饰符等等特性,实现了一些有趣的功能,比如动态列,嵌套数据结构,schema演化等.

由于协处理器和过滤器自身能力的限制,如果完全不依赖MapReduce框架,只通过HBASE客户端API想要实现复杂查询操作,如多表联合操作,相对比较困难,或者大量工作需要在客户端代码中实现,性能上可能无法满足需求.

### (1) 下载

访问Phoenix的官方网站进行下载. 下载地

址:<http://www.apache.org/dyn/closer.lua/phoenix/>

The screenshot shows the Apache Phoenix homepage. At the top, there's a dark header bar with the Apache logo and the word "PHOENIX" followed by the tagline "We put the SQL back in NoSQL". To the right of the header are links for "About", "Using", "Features", and "Reference". The main title "APACHE PHOENIX" is prominently displayed with a stylized orange flame icon between "PHOENIX". Below it, the subtitle "OLTP and operational analytics for Hadoop" is visible. The page features several sections with icons: a download icon for artifacts, a JIRAs icon for browsing issues, a Source {code} icon for building from source, and a search bar. A yellow banner at the bottom contains news about transaction support and a follow link for @ApachePhoenix. The footer includes logos for various companies like Sears, salesforce, intuit, DELL, HUAWEI, Alibaba.com, Intel, and PubMatic.

### (2) 安装

在**Master**上面的操作

```
#sudo tar xvzf phoenix-4.7.0-HBase-1.1-bin.tar.gz  
#sudo chown -R hadoop:hadoop phoenix-4.7.0-HBase-1.1  
#sudo chmod -R 775 phoenix-4.7.0-HBase-1.1  
#sudo mv phoenix-4.7.0-HBase-1.1 /usr/local/phoenix  
#sudo cp /usr/local/phoenix/*.jar /usr/local/hbase/lib/  
#sudo cp /usr/local/hbase/conf/hbase-site.xml /usr/local/phoenix  
/bin/  
hadoop@hadoopmaster:/usr/local$ sudo tar cvfz ~/phoenix2016.tar.  
gz phoenix/  
scp /home/hadoop/phoenix2016.tar.gz hadoop@hadoopslove1:/home/ha  
doop/  
scp /home/hadoop/phoenix2016.tar.gz hadoop@hadoopslove2:/home/ha  
doop/
```

在**Slave**上面的操作

```
hadoop@hadoopslove1:~$ tar xvzf phoenix2016.tar.gz  
hadoop@hadoopslove1:~$ sudo mv phoenix /usr/local/phoenix  
hadoop@hadoopslove1:~$ sudo chown -R hadoop:hadoop /usr/local/ph  
oenix/  
hadoop@hadoopslove1:~$ sudo chmod -R 775 /usr/local/phoenix/  
hadoop@hadoopslove1:/usr/local/phoenix$ cp *.jar /usr/local/hbas  
e/lib/
```

完成以上步骤,基本上Phoenix就安装成功了.

### (3)基本使用

#### 验证操作

```
hadoop@Master:/usr/local/phoenix/bin$ ./sqlline.py localhost  
执行SQL语句  
CREATE TABLE IF NOT EXISTS us_population2 (  
    state CHAR(2) NOT NULL,  
    city VARCHAR NOT NULL,  
    population BIGINT
```

```
CONSTRAINT my_pk2 PRIMARY KEY (state, city));
```

## 插入操作

```
upsert into us_population2(state, city, population) values('NY',
'New York', 8143197);
upsert into us_population2(state, city, population) values('CA',
'Chicago', 2842518);
upsert into us_population2(state, city, population) values('TX',
'Houston', 2016582);
upsert into us_population2(state, city, population) values('TX',
'San Antonio', 1256509);
upsert into us_population2(state, city, population) values('CA',
'San Jose', 912332);
```

## 执行操作

```
SELECT state as "State",count(city) as "City Count",sum(population) as "Population Sum"
FROM us_population2
GROUP BY state
ORDER BY sum(population) DESC;
```

```
0: jdbc:phoenix:localhost> SELECT state as "State",count(city) as "City Count",sum(population) as "Population Sum"
. . . . . . . . . . . . . > FROM us_population2
. . . . . . . . . . . . . > GROUP BY state
. . . . . . . . . . . . . > ORDER BY sum(population) DESC;
+-----+-----+-----+
| State | City Count | Population Sum |
+-----+-----+-----+
| NY    | 1          | 8143197      |
| CA    | 2          | 3754850      |
| TX    | 2          | 3273091      |
+-----+-----+-----+
3 rows selected (0.087 seconds)
0: jdbc:phoenix:localhost>
```

<http://phoenix.apache.org/language/index.html>

## 使用JDBC访问Phoenix

```
package com.chu;

/**
 * Created by chuguangming on 16/9/23.
 */

import java.sql.*;

class BaseDB {

    /**
     * name : getConnection
     * time : 2015年5月6日 下午2:07:06
     * description : get JDBC connection
     *
     * @return connection
     */
    public static Connection getConnection() {
        try {
            // load driver
            Class.forName("org.apache.phoenix.jdbc.PhoenixDriver");
        }

        // get connection
        // jdbc 的 url 类似为 jdbc:phoenix [ :<zookeeper quorum> [ :<port number> ] [ :<root node> ] ],
        // 需要引用三个参数：hbase.zookeeper.quorum、hbase.zookeeper.property.clientPort、and zookeeper.znode.parent，
        // 这些参数可以缺省不填而在 hbase-site.xml 中定义。
        return DriverManager.getConnection("jdbc:phoenix:hadoopmaster,hadoopslave1,hadoopslave2");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

```
    }

}

public class HBaseSQLDriverTest {
    public static void main(String[] args) throws SQLException {
        //Simpletest();
        //create();
        //upsert();
        //query();
        //delete();
    }

    /**
     * name : delete
     * time : 2015年5月4日 下午4:03:11
     * description : delete data
     */
    public static void delete() {

        Connection conn = null;
        try {
            // get connection
            conn = BaseDB.getConnection();

            // check connection
            if (conn == null) {
                System.out.println("conn is null...");
                return;
            }

            // create sql
            String sql = "delete from user88888 where id='001'";

            PreparedStatement ps = conn.prepareStatement(sql);

            // execute upsert
        }
    }
}
```

```
        String msg = ps.executeUpdate() > 0 ? "delete success..." : "delete fail...";

        // you must commit
        conn.commit();
        System.out.println(msg);

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

}

public static void query() {

    Connection conn = null;
    try {
        // get connection
        conn = BaseDB.getConnection();

        // check connection
        if (conn == null) {
            System.out.println("conn is null...");
            return;
        }

        // create sql
        String sql = "select * from user88888";
        PreparedStatement ps = conn.prepareStatement(sql);
    }
}
```

```

        ResultSet rs = ps.executeQuery();

        System.out.println("id" + "\t" + "account" + "\t" +
"passwd");
        System.out.println("====");

        if (rs != null) {
            while (rs.next()) {
                System.out.print(rs.getString("id") + "\t");
                System.out.print(rs.getString("account") + "
\t");
                System.out.println(rs.getString("passwd"));
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

}

/***
 * name : insert
 * time : 2015年5月4日 下午2:59:11
 * description :
 */
public static void upsert() {

    Connection conn = null;
    try {
        // get connection
        conn = BaseDB.getConnection();

```

```

        // check connection
        if (conn == null) {
            System.out.println("conn is null...");
            return;
        }

        // create sql
        String sql = "upsert into user8888(id, INFO.account
, INFO.passwd) values('001', 'admin', 'admin')";

        PreparedStatement ps = conn.prepareStatement(sql);

        // execute upsert
        String msg = ps.executeUpdate() > 0 ? "insert succes
s..."
                : "insert fail...";

        // you must commit
        conn.commit();
        System.out.println(msg);

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

public static void create() {
    Connection conn = null;
    try {
        // get connection
        conn = BaseDB.getConnection();
    }
}

```

```

        // check connection
        if (conn == null) {
            System.out.println("conn is null...");
            return;
        }

        // check if the table exist
        ResultSet rs = conn.getMetaData().getTables(null, null, "USER",
   null);
        if (rs.next()) {
            System.out.println("table user is exist...");
            return;
        }
        // create sql
        String sql = "CREATE TABLE user88888 (id varchar PRIMARY KEY, INFO.account varchar ,INFO.passwd varchar)";

        PreparedStatement ps = conn.prepareStatement(sql);

        // execute
        ps.execute();
        System.out.println("create success...");

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

private static void Simpletest() throws SQLException {

```

```

        Statement stmt = null;
        ResultSet rs = null;
        String viewName = "\"US_POPULATION2\""; // 这是对HBase表"
food:products"创建的Phoenix view

        System.err.println("\n[viewName = " + viewName + "]\n");

        /* ecs1.njzd.com:2181是zookeeper的某一个节点的ip:port
           即使集群中的ZooKeeper存在多个节点，这里也只需要写出一个节点的i
p:port就可以了*/
        // 如果是Scala，还需要这一句
        //Class.forName("org.apache.phoenix.jdbc.PhoenixDriver")
;

        Connection conn = DriverManager.getConnection("jdbc:phoe
nix:hadoopmaster,hadoopslave1,hadoopslave2");

        /* 在Phoenix中，如果table name/view name、column name等字符
串不加上双引号就会被认为是大写。所以，这里的brand_name要加上双引号 */
        PreparedStatement pstmt = conn.prepareStatement("select
* from " + viewName);
        rs = pstmt.executeQuery();

        while (rs.next()) {

            System.err.println(rs.getString("STATE"));

            System.err.println("=====");
            =====");
        }
        /* 关闭资源*/
        rs.close();
        pstmt.close();
    }
}

```

pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"

```

```

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.chu</groupId>
    <artifactId>HBaseDemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc -->
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-jdbc</artifactId>
            <version>2.1.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hibernate-jpa-2.0-api -->
        <dependency>
            <groupId>org.hibernate.javax.persistence</groupId>
            <artifactId>hibernate-jpa-2.0-api</artifactId>
            <version>1.0.1.Final</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-exec</artifactId>
            <version>2.1.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-client -->
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-client</artifactId>
            <version>1.1.5</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase -->
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase</artifactId>

```

```

        <version>1.1.5</version>
        <type>pom</type>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hbase
/hbase-common -->
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-common</artifactId>
        <version>1.1.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hbase
/hbase-server -->
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-server</artifactId>
        <version>1.1.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hbase
/hbase-protocol -->
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-protocol</artifactId>
        <version>1.1.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.phoenix
/phoenix-core -->
    <dependency>
        <groupId>org.apache.phoenix</groupId>
        <artifactId>phoenix-core</artifactId>
        <version>4.7.0-HBase-1.1</version>
    </dependency>

</dependencies>
<repositories>
    <repository>
        <id>jboss</id>
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
    </repository>
</repositories>
```

```
</project>
```

## 3. HBASE的数据迁移

### 概述

将数据移到Hbase的方法有以下几种：

- 使用Hbase的Put API
- 使用HBase的批量加载工具
- 使用自定义的MapReduce方法

使用HBase的Put API是最直接的方法。这种方法的使用并不难学，但大多数情况下，它并非总是最有效的方法。特别是在有一大批数据需要移入Hbase并且对移入都是问题又有限定的情况下，这种方法的效率并不高。我们需要处理的数据通常都有很大的数据量，这可能也是我们使用Hbase而不是其他数据库的原因。你必须在HBase的项目的开始阶段就仔细考虑如何将所有数据转入Hbase，否则你将会遇到一些严重的性能问题。

Hbase提供了批量加载的功能来支持高效地将大量数据加载到HBase中。批量加载功能使用了一个MapReduce任务将数据加载到一个特定的Hbase表中。它会生成一些HBase内部的HFile数据格式的文件，然后再将这些数据文件直接加载到正在运行的集群中。使用批量加载功能最简单的方法是使用importtsv工具。Importtsv是一个可将数据从TSV文件加载到HBase中的内置工具。它会运行一些MapReduce任务来读取TSV文件中的数据，然后将其输出直接写入HBASE表或HBASE内部数据格式的文件中。

虽然importtsv工具在将文本数据导入HBASE的时候非常有用，但是在有些情况下，比如导入一些其他格式的数据时，你可能必须以编程的方式来生成数据。MapReduce是处理海量数据最有效方法。它可能也是将海量数据加载到Hbase中唯一可行的方法。当然我们也可以使用Mapreduce将数据导入到HBASE中。然而当数据量非常大的时候，MapReduce任务的负载也可能非常重，如果不正确对待，重负载mapreduce任务运行进的吞吐量也可能很差的。

数据迁移是HBASE上一项写密集的任务，除非我们能先生成好一些内部数据文件然后再把它们直接加载到HBASE中去。尽管HBASE的写操作总是非常快，但是如果配置不正确，在迁移的过程中也会经常出现写操作堵塞的情况。写密集任务的另一个问题

是：所有写操作可能针对的都是同一台区域服务器，尤其是在将大量数据加载到一个新的HBase安装的时候，这种情况更容易发生。因为所有负载都集中在一台服务器上，不能均衡分配给集群中的各个服务器，所以写入速度也会明显减慢。

## 方案的选择

HBase本身提供了很多种数据导入的方式，通常有两种常用方式：

1. 使用HBase提供的TableOutputFormat，原理是通过一个Mapreduce作业将数据导入HBase
2. 另一种方式就是使用HBase原生Client API

这两种方式由于需要频繁的与数据所存储的RegionServer通信，一次性入库大量数据时，特别占用资源，所以都不是最有效的。了解过HBase底层原理的应该都知道，HBase在HDFS中是以HFile文件结构存储的，一个比较高效便捷的方法就是使用“Bulk Loading”方法直接生成HFile，即HBase提供的HFileOutputFormat类。

## (1) MapReduce的方式来实现数据导入

我们采用hbase自带的importtsv工具来导入数据，首先要把数据文件上传到hdfs上，然后导入hbase表，该方法只能导入tsv格式的数据，需要先将txt格式转换为tsv格式。

### 1. 下载数据

本文中使用“美国国家海洋和大气管理局 气候平均值”的公共数据集合。访问<http://www1.ncdc.noaa.gov/pub/data/normals/1981-2010/>下载。在目录products | hourly下的小时温度数据。下载hly-temp-10pctl.txt文件。此数据下载巨慢，可以从内网获取，我已放到FTP服务器Hadoop目录中的TestData目录中。

### 2. 转换数据

用python脚本将txt文件转换为tsv格式文件，生成之后上传到虚拟机linux下的/home/hadoop/data下，python脚本[下载链接](#)

### 3. 基本数据迁移

```
python to_tsv_hly.py -f hly-temp-10pctl.txt -t hly-temp-10pctl.tsv
```

### 3. 在上创建数据存放目录

```
hadoop fs -mkdir /input
```

### 4. 将数据库**copy**到数据存放目录中

```
hadoop@Master:~$ hadoop fs -ls
Found 7 items
drwxr-xr-x  - hadoop supergroup          0 2016-02-25 18:41 .sp
arkStaging
drwxr-xr-x  - hadoop supergroup          0 2016-03-03 02:24 cds
gus
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 21:32 chu
888chu888
drwxrwxr-x  - hadoop supergroup          0 2016-01-28 18:54 hiv
e
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 22:15 inp
ut
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 22:19 out
put
drwxrwxr-x  - hadoop supergroup          0 2016-01-28 18:53 tmp
hadoop@Master:~$ hadoop fs -copyFromLocal /home/hadoop/hly-temp-
10pctl.tsv /input
```

### 5. 在中创建要导入数据的表

### 3. 基本数据迁移

```
hbase(main):001:0> create 'hly_temp', {NAME => 't', VERSIONS =>
1}
0 row(s) in 1.6740 seconds

=> Hbase::Table - hly_temp
hbase(main):002:0>
```

## 6. 执行数据导入

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.c
olumns=HBASE_ROW_KEY,t:v01,t:v02,t:v03,t:v04,t:v05,t:v06,t:v07,t
:v08,t:v09,t:v10,t:v11,t:v12,t:v13,t:v14,t:v15,t:v16,t:v17,t:v18
,t:v19,t:v20,t:v21,t:v22,t:v23,t:v24 hly_temp /input
```



## 7. 导入后检查数据

```
hbase(main):002:0> count 'hly_temp'
Current count: 1000, row: FMW000405040927

Current count: 2000, row: GQW000414150624

Current count: 3000, row: RMW000407100321
.....
..
Current count: 166000, row: USW000949101017

166805 row(s) in 35.6090 seconds
=> 166805

hbase(main):004:0> scan 'hly_temp', {COLUMNS => 't', LIMIT => 2}
ROW                                COLUMN+CELL
```

AQW000617050101 24204703, value=759P	column=t:v01, timestamp=14509
AQW000617050101 24204703, value=766C	column=t:v02, timestamp=14509
AQW000617050101 24204703, value=759C	column=t:v03, timestamp=14509
AQW000617050101 24204703, value=759C	column=t:v04, timestamp=14509
AQW000617050101 24204703, value=759C	column=t:v05, timestamp=14509
AQW000617050101 24204703, value=759C	column=t:v06, timestamp=14509
AQW000617050101 24204703, value=752C	column=t:v07, timestamp=14509
AQW000617050101 24204703, value=775C	column=t:v08, timestamp=14509
AQW000617050101 24204703, value=801C	column=t:v09, timestamp=14509
AQW000617050101 24204703, value=810C	column=t:v10, timestamp=14509
AQW000617050101 24204703, value=810C	column=t:v11, timestamp=14509
AQW000617050101 24204703, value=810C	column=t:v12, timestamp=14509
AQW000617050101 24204703, value=810C	column=t:v13, timestamp=14509

### 3. 基本数据迁移

AQW000617050101 24204703, value=808C	column=t:v14, timestamp=14509
AQW000617050101 24204703, value=806C	column=t:v15, timestamp=14509
AQW000617050101 24204703, value=810C	column=t:v16, timestamp=14509
AQW000617050101 24204703, value=808C	column=t:v17, timestamp=14509
AQW000617050101 24204703, value=801C	column=t:v18, timestamp=14509
AQW000617050101 24204703, value=801C	column=t:v19, timestamp=14509
AQW000617050101 24204703, value=790C	column=t:v20, timestamp=14509
AQW000617050101 24204703, value=781C	column=t:v21, timestamp=14509
AQW000617050101 24204703, value=781C	column=t:v22, timestamp=14509
AQW000617050101 24204703, value=770C	column=t:v23, timestamp=14509
AQW000617050101 24204703, value=770C	column=t:v24, timestamp=14509
AQW000617050102 24204703, value=768P	column=t:v01, timestamp=14509
AQW000617050102 24204703, value=766C	column=t:v02, timestamp=14509
AQW000617050102	column=t:v03, timestamp=14509

### 3. 基本数据迁移

24204703, value=759C	
AQW000617050102	column=t:v04, timestamp=14509
24204703, value=759C	
AQW000617050102	column=t:v05, timestamp=14509
24204703, value=759C	
AQW000617050102	column=t:v06, timestamp=14509
24204703, value=759C	
AQW000617050102	column=t:v07, timestamp=14509
24204703, value=757C	
AQW000617050102	column=t:v08, timestamp=14509
24204703, value=775C	
AQW000617050102	column=t:v09, timestamp=14509
24204703, value=801C	
AQW000617050102	column=t:v10, timestamp=14509
24204703, value=810C	
AQW000617050102	column=t:v11, timestamp=14509
24204703, value=810C	
AQW000617050102	column=t:v12, timestamp=14509
24204703, value=810C	
AQW000617050102	column=t:v13, timestamp=14509
24204703, value=811C	
AQW000617050102	column=t:v14, timestamp=14509
24204703, value=810C	
AQW000617050102	column=t:v15, timestamp=14509
24204703, value=810C	
AQW000617050102	column=t:v16, timestamp=14509
24204703, value=808C	

```

AQW000617050102          column=t:v17, timestamp=14509
24204703, value=808C

AQW000617050102          column=t:v18, timestamp=14509
24204703, value=801C

AQW000617050102          column=t:v19, timestamp=14509
24204703, value=795C

AQW000617050102          column=t:v20, timestamp=14509
24204703, value=790C

AQW000617050102          column=t:v21, timestamp=14509
24204703, value=781C

AQW000617050102          column=t:v22, timestamp=14509
24204703, value=781C

AQW000617050102          column=t:v23, timestamp=14509
24204703, value=774C

AQW000617050102          column=t:v24, timestamp=14509
24204703, value=770C

2 row(s) in 0.4910 seconds

至此，导入成功！

```

## (2) Put API 的方法实现导入

数据迁移最常见的情况可能就是从现有的RDBMS将数据导入到HBASE中了.对于这类任务,可能最简单也最直接的方法就是:用一个客户端来读取数据,然后通过HBASE的PUT API把数据送到HBASE中去.如果需要传输的数据不太多,这种方法非常合适.

### 1 准备数据

### 3. 基本数据迁移

本文中使用“美国国家海洋和大气管理局 气候平均值”的公共数据集合。访问 <http://www1.ncdc.noaa.gov/pub/data/normals/1981-2010/> 下载。在目录 products | hourly 下的小时温度数据。下载 hly-temp-10pctl.txt 文件。此数据下载巨慢,可以从内网获取,我已放到 FTP 服务器 Hadoop 目录中的 TestData 目录中。

在 MySQL 数据库中创建一个表 hly\_temp\_normal

```
create table hly_temp_normal
(
    id int not null auto_increment primary key,
    stnid char(11),
    month tinyint,
    day tinyint,
    value1 VARCHAR(5),
    value2 VARCHAR(5),
    value3 VARCHAR(5),
    value4 VARCHAR(5),
    value5 VARCHAR(5),
    value6 VARCHAR(5),
    value7 VARCHAR(5),
    value8 VARCHAR(5),
    value9 VARCHAR(5),
    value10 VARCHAR(5),
    value11 VARCHAR(5),
    value12 VARCHAR(5),
    value13 VARCHAR(5),
    value14 VARCHAR(5),
    value15 VARCHAR(5),
    value16 VARCHAR(5),
    value17 VARCHAR(5),
    value18 VARCHAR(5),
    value19 VARCHAR(5),
    value20 VARCHAR(5),
    value21 VARCHAR(5),
    value22 VARCHAR(5),
    value23 VARCHAR(5),
    value24 VARCHAR(5)
);
```

### 3. 基本数据迁移

使用insert\_gly.py脚本将数据载入到MYSQL数据库中.需要修改insert\_gly.py这个脚本中的主机名用户名密码和数据库名称 [附带脚本下载](#)

```
python insert_hly.py -f hly-temp-normal.txt -t hly_temp_normal
```

## 4. 通过PutAPI的方法来导出数据

### 概述

HBase本身提供了很多种数据导入的方式，通常有两种常用方式：

1. 使用HBase提供的TableOutputFormat，原理是通过一个Mapreduce作业将数据导入HBase
2. 另一种方式就是使用HBase原生Client API

这两种方式由于需要频繁的与数据所存储的RegionServer通信，一次性入库大量数据时，特别占用资源，所以都不是最有效的。了解过HBase底层原理的应该都知道，HBase在HDFS中是以HFile文件结构存储的，一个比较高效便捷的方法就是使用“Bulk Loading”方法直接生成HFile，即HBase提供的HFileOutputFormat类。

## Bulk Load基本原理

Bulk Load处理由两个主要步骤组成

1. 准备数据文件 Bulk Load的第一步，会运行一个Mapreduce作业，其中使用到了HFileOutputFormat输出HBase数据文件：StoreFile。HFileOutputFormat的作用在于使得输出的HFile文件可以适应单个region，使用TotalOrderPartitioner类将map输出结果分区到各个不同的key区间中，每个key区间都对应着HBase表的region。
2. 导入HBase表 第二步使用completebulkload工具将第一步的结果文件依次交给负责文件对应region的RegionServer，并将文件move到region在HDFS上的存储目录中，一旦完成，将数据开放给clients。如果在bulk load准备导入或在准备导入与完成导入的临界点上发现region的边界已经改变，completebulkload工具会自动split数据文件到新的边界上，但是这个过程并不是最佳实践，所以用户在使用时需要最小化准备导入与导入集群间的延时，特别是当其他client在同时使用其他工具向同一张表导入数据。

## 5. 使用管理工具

每个人都希望自己的HBASE管理员能够让集群运行流畅,存储大量的数据,并且能同时,迅速和可靠地处理几百万的并发请求.对于管理员来说,让HBASE中海量数据一直保持可存取,易管理和便于查询是一项至关重要的任务.

除了对于你运行的集群要有扎实的了解之外,你所使用的工具也同样重要.HBASE自带了一些管理工具,它可以使管理员的工作变得轻松一些.HBASE带有一个基于WEB的管理页面.在此页面中可以查看集群的状态.

### 1. HBASE主WEB界面

HBASE主WEB界面是一个简单而又有用的工具,它可以为我们提供集群当前的状态的概况.从该页面中可以获取所运行HBASE版本/HBASE的配置(包括HDFS根路径和ZOOKeeper仲裁地址)/集群的平均负载以及表/区域/区域服务器的列表.

HBASE主界面的默认端口是60010,请确保你的网络防火墙已经对你的客户端打开了该端口.

```
http://192.168.1.80:16010/master-status
```

HBASE主WEB界面如下

## 5. 使用管理工具

APACHE HBASE

Home Table Details Local Logs Log Level Debug Dump Metrics Dump HBase Con

### Master Master

#### Region Servers

ServerName	Start time	Requests Per Sec
slave1,16020,1460641908755	Thu Apr 14 21:51:48 CST 2016	0
slave2,16020,1460641905097	Thu Apr 14 21:51:45 CST 2016	0
Total:2		0

#### Dead Region Servers

Servername	Port	Start time
Total:0		

#### Tables

User Tables	System Tables	Snapshots					
18 table(s) in set. [Details]							
Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	MIANTIAO_USERNAME	1	0	0	0	0	'MIANTIAO_USERNAME\$1 => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.phoenix.iache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexcoprocessor\$6 => {o.localIndexSplitter[805306366]}}, {NAME => '0', DATA_BLOCK_ENCODING = 'FAST_DIFF'}}}
default	PERFORMANCE_10000	9	0	0	0	0	'PERFORMANCE_10000\$1 => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.phoenix.iache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexcoprocessor\$6 => {o.localIndexSplitter[805306366]}}, {NAME => 'STATS', DATA_BLOCK_ENCODING = 'FAST_DIFF'}}}
default	SYSTEM.SEQUENCE	1	0	0	0	0	'SYSTEM.SEQUENCE\$1 => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.phoenix.iache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexcoprocessor\$6 => {o.localIndexSplitter[805306366]}}, {NAME => '0', VERSIONS => '1000', KEEPF_DELETED_CELG => 'FAST_DIFF'}}}
default	SYSTEM.STATS	1	0	0	0	0	'SYSTEM.STATS', {TA => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], cc[org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.hadoop.805306366], METADATA => {'SPLIT_POLICY' => 'org.apache.phoenix.scherKEEP_DELETED_CELG => 'FAST_DIFF'}}}}
default	US_POPULATION	1	0	0	0	0	'US_POPULATION', \${1 => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.phoenix.iache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexcoprocessor\$6 => {o.localIndexSplitter[805306366]}}, {NAME => '0', DATA_BLOCK_ENCODING = 'FAST_DIFF'}}}}
default	US_POPULATION2	1	0	0	0	0	'US_POPULATION2', \${1 => {org.apache.phoenix.coprocessor.ScanRegionObserver[805306366], org.apache.phoenix.io[805306366], coprocessor\$3 => {org.apache.phoenix.coprocessor[805306366], org.apache.phoenix.io[805306366], coprocessor\$5 => {org.apache.phoenix.iache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexcoprocessor\$6 => {o.localIndexSplitter[805306366]}}, {NAME => '0', DATA_BLOCK_ENCODING = 'FAST_DIFF'}}}}
default	lxw1234	1	0	0	0	0	'lxw1234', {NAME =>
default	question	1	0	0	0	0	'question', {NAME => VERSIONS => '1000', BLOCKCACHE => 'false'}, {NAME => 'basic', BLOOM => 'comment', BLOOMFILTER => 'false'}, {NAME => 'detail', BLOOMFILTER => 'NON_BLOOMFILTER => 'NACHE => 'false')}
default	ssss	1	0	0	0	0	'ssss', {NAME => '11', RSIONS => '2'}
default	t1	1	0	0	0	0	't1', {NAME => '11', VI}
default	t2	1	0	0	0	0	't2', {NAME => '11'}, {
default	t3	15	0	0	0	0	't3', {NAME => '11'}
default	t4	5	0	0	0	0	't4', {NAME => '11'}

#### Tasks

Show All Monitored Tasks	Show non-RPC Tasks	Show All RPC Handler Tasks	Show Active RPC Calls	Show Client (
No tasks currently running on this node.				

#### Software Attributes

Attribute	Value	Description
Hadoop Version	2.6.1, revision=2e18d1/9e4a80bb6a9f29cf2de945189126bcc	Hadoop version ar
Hadoop Compiled	2014-09-05T23:05Z, kasha	When Hadoop ver
Hadoop Source Checksum	6424fcab95bfff8337780a181ad7c78	Hadoop source MI
ZooKeeper Client Version	3.4.6, revision=1569965	ZooKeeper client v
ZooKeeper Client Compiled	02/20/2014 09:09 GMT	When ZooKeeper
Zookeeper Quorum	Master:2181 Slave1:2181 Slave2:2181	Addresses of all re dump.
Zookeeper Base Path	/hbase	Root node of this c
HBase Root Directory	hdfs://Master:9000/hbase	Location of HBase

正如你所看到的那样,在"Attributes区中显示的信息有":HBASE和Hadoop的版本/HBASE在HDFS上的根目录/集群的平均负载以及Zookeeper仲裁地址.HBASE根目录和Zookeeper服务器地址是HBASE配置文件hbase-site.xml中设置的值.平均负载是每个区域服务器所负责区域的平均值.一个区域服务器的负载显示在Region Server区中.

在Catalog Table(目录表)区中你可以看到两张表:-ROOT和.META.这是两张HBASE的系统表,-ROOT表里保存的是部署有.META表所有区域服务器的引用,而.META表中保存的是所有用户表区域的引用.

## 使用**HBASE SHELL**管理表

我们现在通过下列步骤来展示如何使用DDL命令来管理HBASE表.

1. 在客户机节点上执行以下命令启动HBASE SHELL

```
#hbase shell
```

2. 在HBASE SHELL中使用create命令创建一张带有一个列族(f1)的表(t1)

```
hbase(main):001:0> create 't1', 'f1'
```

3. 使用list命令列出所有表的列表

```
hbase(main):008:0> list
```

```
TABLE
```

```
MIANTIAO_USERNAME
```

```
PERFORMANCE_10000
```

```
SYSTEM.CATALOG
```

SYSTEM.FUNCTION

SYSTEM.SEQUENCE

SYSTEM.STATS

US\_POPULATION

US\_POPULATION2

UUU

UUU33

hly\_temp

lxw1234

question

ssss

t1

t2

t3

```
t4
```

```
18 row(s) in 0.0370 seconds
```

```
=> ["MIANTIAO_USERNAME", "PERFORMANCE_10000", "SYSTEM.CATALOG",
"SYSTEM.FUNCTION", "SYSTEM.SEQUENCE", "SYSTEM.STATS", "US_POPULA-
TION", "US_POPULATION2", "UUU", "UUU33", "hly_temp", "lxw1234",
"question", "ssss", "t1", "t2", "t3", "t4"]
hbase(main):009:0>
```

4. 使用describe命令来显示该表的属性

```
hbase(main):009:0> describe 't1'
Table t1 is ENABLED
```

```
t1
```

```
COLUMN FAMILIES DESCRIPTION
```

```
{NAME => 'f1', BLOOMFILTER => 'ROW', VERSIONS => '5', IN_MEMORY
=> 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING =
> '
NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS =>
'0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE
=> '0'}
```

```
1 row(s) in 0.1840 seconds
```

5. 使用disable 命令来禁用该表

## 5. 使用管理工具

```
hbase(main):013:0> disable 't1'  
0 row(s) in 0.0170 seconds  
  
hbase(main):014:0> is_enabled 't1'  
false  
  
0 row(s) in 0.0160 seconds
```

6. 使用alter命令来修改表的属性。下面这段代码会先将f1修改为只有一个版本，然后再添加了一个新列族的f2

```
hbase(main):015:0> alter 't1', {NAME=>'f1', VERSIONS=>'1'}, {NAME=>  
'f2'}  
Updating all regions with the new schema...  
1/1 regions updated.  
Done.  
Updating all regions with the new schema...  
1/1 regions updated.  
Done.  
0 row(s) in 3.8100 seconds
```

7. 使用enable命令来启用该表

```
hbase(main):016:0> enable 't1'  
0 row(s) in 1.4190 seconds  
  
hbase(main):017:0> is_enabled 't1'  
true  
  
0 row(s) in 0.0090 seconds
```

8. 输入以下命令来再次禁用该表。然后将期删除

## 5. 使用管理工具

```
hbase(main):018:0> disable 't1'  
0 row(s) in 2.2440 seconds
```

```
hbase(main):019:0> drop 't1'  
0 row(s) in 1.2650 seconds
```

9. 使用put命令将下列数据插入到该表中

```
hbase(main):021:0> put 't1','row1','f1:c1','value1'  
0 row(s) in 0.1030 seconds
```

```
hbase(main):022:0> put 't1','row1','f1:c2','value2'  
0 row(s) in 0.0130 seconds
```

```
hbase(main):023:0> put 't1','row2','f1:c1','value3'  
0 row(s) in 0.0230 seconds
```

put命令使用表名(t1)/行键(row1)/列族和限定符(f1:c1)以及要赋予的值(value1)为参数,别外它还有一个可选的时间戳参数.在f1:c1中.冒号(:)是列族(f1)与限定符(c1)之前的间隔符.在HBASE SHELL中,如果想要把一个值当成字符串,你就用单引号包括起来就行.

10. 执行count命令来获取该表的行数

```
hbase(main):024:0> count 't1'  
2 row(s) in 0.0930 seconds
```

```
=> 2
```

count命令可以统计一张表中记录数,它会以"行键"为基础进行统计,所以为2.

11. 使用scan命令来扫描数据.在对记录很多的表进行scan时,不要忘记指定LIMIT属性

```
hbase(main):029:0> scan 't1',{LIMIT=>10}  
ROW  
COLUMN+CELL
```

```
row1  
4326819, value=value1  
column=f1:c1, timestamp=146064
```

## 5. 使用管理工具

```
row1                                column=f1:c2, timestamp=146064
4335627, value=value2

row2                                column=f1:c1, timestamp=146064
4754895, value=value7

row3                                column=f1:c1, timestamp=146064
4762929, value=value7

3 row(s) in 0.0600 seconds
```

### 12. 使用get命令来读取某一行

```
hbase(main):030:0> get 't1','row1'
COLUMN          CELL
f1:c1           timestamp=1460644326819, value
=value1

f1:c2           timestamp=1460644335627, value
=value2

2 row(s) in 0.0200 seconds
```

### 13. 使用delete命令来删除某一指定单元格

```
hbase(main):032:0> delete 't1','row1','f1:c1'
0 row(s) in 0.0250 seconds
```

```
hbase(main):033:0> scan 't1',{LIMIT=>10}
ROW          COLUMN+CELL
row1         column=f1:c2, timestamp=146064
4335627, value=value2
```

## 5. 使用管理工具

```
row2                                column=f1:c1, timestamp=146064
4754895, value=value7

row3                                column=f1:c1, timestamp=146064
4762929, value=value7

3 row(s) in 0.0240 seconds
```

14. 使用deleteall命令删除某一指定行的所有单元格

```
hbase(main):002:0> delete 't1', 'row1', 'f1:c1'
0 row(s) in 0.0610 seconds
```

15. 再次执行get命令；你会发现整个row1行都已被从表中删除了。

```
hbase(main):001:0> get 't1', 'row1'
COLUMN          CELL
f1:c2          timestamp=1460644335627, va
lue=value2

1 row(s) in 0.3090 seconds
```

16. 使用incr命令让计数器(row1:f1:c1)的值加1

```
hbase(main):003:0> scan 't1'
ROW          COLUMN+CELL
row1          column=f1:c2, timestamp=146
0644335627, value=value2

row2          column=f1:c1, timestamp=146
0644754895, value=value7

row3          column=f1:c1, timestamp=146
0644762929, value=value7
```

## 5. 使用管理工具

```
3 row(s) in 0.1120 seconds

hbase(main):004:0> incr 't1','row1','f1:c1',1
COUNTER VALUE = 1
0 row(s) in 0.0280 seconds

hbase(main):005:0> scan 't1'
ROW                                     COLUMN+CELL

row1                               column=f1:c1, timestamp=146
0801878627, value=\x00\x00\x00\x00\x00\x00\x00\x01

row1                               column=f1:c2, timestamp=146
0644335627, value=value2

row2                               column=f1:c1, timestamp=146
0644754895, value=value7

row3                               column=f1:c1, timestamp=146
0644762929, value=value7

3 row(s) in 0.0310 seconds

17. 再使该计数器加10
hbase(main):006:0> incr 't1','row1','f1:c1',10
COUNTER VALUE = 11
0 row(s) in 0.0160 seconds

hbase(main):007:0> scan 't1'
ROW                                     COLUMN+CELL

row1                               column=f1:c1, timestamp=146
0802073632, value=\x00\x00\x00\x00\x00\x00\x00\x0B

row1                               column=f1:c2, timestamp=146
0644335627, value=value2
```

```
row2                                column=f1:c1, timestamp=146  
0644754895, value=value7  
  
row3                                column=f1:c1, timestamp=146  
0644762929, value=value7  
  
3 row(s) in 0.0300 seconds
```

### 18. 使用get\_counter命令来读取该计数器的新值

```
hbase(main):008:0> get_counter 't1', 'row1', 'f1:c1'  
COUNTER VALUE = 11
```

### 19. 执行truncate命令来截断该表

```
truncate命令会进行一系列的操作，对表进行禁用/删除/然后再重建  
hbase(main):009:0> truncate 't1'  
Truncating 't1' table (it may take a while):  
- Disabling table...  
- Truncating table...  
0 row(s) in 3.4570 seconds
```

## 使用**HBASE SHELL**管理集群

### 在**HBASE SHELL**中执行**JAVA**方法

### **WAL**工具 手动分割和转储**WAL**

### **HFile**工具 以文本方式查看**HFile**的内容

### **HBASE hbck**检查**HBASE**集群的一致性

查看hbasesmeta情况

```
hbase hbck
```

1. 重新修复hbase meta表（根据hdfs上的regioninfo文件，生成meta表）

```
hbase hbck -fixMeta
```

2. 重新将hbase meta表分给regionserver (根据meta表，将meta表上的region分给regionserver)

```
hbase hbck -fixAssignments
```

新版本的 hbck

1) 缺失hbase.version文件

加上选项 -fixVersionFile 解决

2) 如果一个region即不在META表中，又不在hdfs上面，但是在regionserver的online region集合中

加上选项 -fixAssignments 解决

3) 如果一个region在META表中，并且在regionserver的online region集合中，但是在hdfs上面没有

加上选项 -fixAssignments -fixMeta 解决，(-fixAssignments告诉regionserver close region)，(-fixMeta删除META表中region的记录)

4) 如果一个region在META表中没有记录，没有被regionserver服务，但是在hdfs上面有

加上选项 -fixMeta -fixAssignments 解决，(-fixAssignments 用于assign region)，(-fixMeta用于在META表中添加region的记录)

5) 如果一个region在META表中没有记录，在hdfs上面有，被regionserver服务了

加上选项 -fixMeta 解决，在META表中添加这个region的记录，先undeploy region，后assign

6) 如果一个region在META表中有记录，但是在hdfs上面没有，并且没有被regionserver服务

加上选项 -fixMeta 解决，删除META表中的记录

7) 如果一个region在META表中有记录，在hdfs上面也有，table不是disabled的，但是这个region没有被服务

加上选项 -fixAssignments 解决，assign这个region

8) 如果一个region在META表中有记录，在hdfs上面也有，table是disabled的，但是这个region被某个regionserver服务了加上选项 -fixAssignments 解决，undeploy这个region

9) 如果一个region在META表中有记录，在hdfs上面也有，table不是disabled的，但是这个region被多个regionserver服务了 加上选项 -fixAssignments 解决，通知所有regionserver close region，然后assign region

10) 如果一个region在META表中，在hdfs上面也有，也应该被服务，但是META表中记录的regionserver和实际所在的regionserver不相符 加上选项 -fixAssignments 解决

11) region holes

需要加上 -fixHdfsHoles ，创建一个新的空region，填补空洞，但是不assign这个region，也不在META表中添加这个region的相关信息

12) region在hdfs上面没有.regioninfo文件

-fixHdfsOrphans 解决

13) region overlaps

需要加上 -fixHdfsOverlaps

说明：

(1) 修复region holes时，-fixHdfsHoles 选项只是创建了一个新的空region，填补上了这个区间，还需要加上-fixAssignments -fixMeta 来解决问题，(-fixAssignments 用于assign region)，(-fixMeta 用于在META表中添加region的记录)，所以有了组合拳 -repairHoles 修复region holes，相当于-fixAssignments -fixMeta -fixHdfsHoles -fixHdfsOrphans

(2) -fixAssignments，用于修复region没有assign、不应该assign、assign了多次的问题

(3) -fixMeta，如果hdfs上面没有，那么从META表中删除相应的记录，如果hdfs上面有，在META表中添加上相应的记录信息

(4) -repair 打开所有的修复选项，相当于-fixAssignments -fixMeta -fixHdfsHoles -fixHdfsOrphans -fixHdfsOverlaps -fixVersionFile -sidelineBigOverlaps

新版本的hbck从(1) hdfs目录(2) META(3) RegionServer这三处获得region的Table和Region的相关信息，根据这些信息判断并repair

## HBASE HIVE使用类SQL语言查询HBASE的数据

# 6 Hbase 数据备份及恢复

## 1 简介

若在生产环境中使用HBase,必须了解备份HBase的各种可选方案和操作方法.备份HBase时的难点是其待备份的数据集可能非常巨大,因此备份方案必须有很高的效果.HBase备份方案必须即能够伸缩至对数百TB的存储容量进行备份,又能够在合理的时间范围内完成数据恢复的工作.

备份HBase有两种策略

1. 关闭集群后进行全备份
2. 在线对集群进行备份

在进行关机全备份时,必须首先关闭HBase(或禁用所有表),然后使用Hadoop的distcp命令将HBase目录中的内容复制到同一(或另一)HDFS中的其他目录中.在使用关机全备份进行恢复时,只要使用distcp命令将所备份的文件复制回原HBase目录就可以了.

在线备份集群也有以下几种方法.

1. 使用copytable工具来将一张表的数据复制到另一张表中
2. 将HBase表导出为HDFS文件,然后再将文件导入到HBase中
3. HBase集群复制

CopyTable实用程序可以用来在一张表与本集群(或其他集群)中的另一张表之间复制数据,Export实用程序可将某张表中的数据转储到本集群的HDFS中.作为Export的搭档,Import实用程序可用来恢复转储文件的数据.

上述每种方法都有其自身的优缺点.关机全备份的优点是在备份过程中集群不会有数据写入,因此它能够确保备份的一致性.其不足之处也很显而易见,那就是要关闭集群.对于在线备份集群的方式来说,因为集群处于运行状态,所以有在备份过程中丢失某些数据修改的风险.此外,HBASE的数据修改只是一个行级的原子操作,如果你的表是彼此依赖的,而且在执行Export或CopyTable的过程中表数据发生了修改,那么所生成的备份就可能有不一致的问题.在当前Apache发布的版本中,HBASE还不支持对表建立快照.

HBASE支持集群复制的功能,这是一种在不同HBASE部署之间复制数据的方法.集群复制可以视为一种HBASE级的灾难恢复解决方案.

除了表以外,你可能还要对HDFS元数据和HBASE区域的开始键进行备份.HDFS元数据包括HDFS文件系统的映像和提交日志.一处元数据的损坏有可能毁掉整个HDFS元数据,因此建议经常对元数据进行备份.区域开始键代表了的数据在HBASE中分布情况.如果备份了区域的开始键,那么在恢复时不仅可以恢复数据,而且还可以对数据分布情况进行恢复,如果能预先使用均匀分布的区域开始键对表进行分割,那么使用CopyTable或Import恢复数据的速度就会有很大的提高.

## 2 使用distcp进行关机全备份

distcp(distributed copy 分布式复制)是由Hadoop提供的一个用于在同一HDFS集群或者不同HDFS集群之间复制大型数据集,它使用MapReduce来并行复制文件/处理错误并进行恢复/报告任务运行状态.

HBASE的所有文件(包括系统文件)存储在HDFS上,因此只要使用distcp将HBASE目录复制到同一HDFS或者其他HDFS的另一个目录中,就可以完成对源HBASE集群的备份工作.

HBASE请注意,这是一种关机情况下的全备份方案,我们可以使用distcp工具来进行备份的原因是HBASE集群已被关闭(或所有表都已被禁用),因此在备份过程中不会有对文件的修改操作.不要在运行中的HBASE集群上使用distcp.因此这种解决方案适合那种允许对HBASE集群进行周期性关闭的环境.例如一个仅用于后端批处理业务而不对前端请求进行响应的集群.

### 1 首先启动HBASE,查看文件存放位置

```

hadoop@Master:~$ start-dfs.sh
Starting namenodes on [Master]
Master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-namenode-Master.out
Slave2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-datanode-Slave2.out
Slave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-datanode-Slave1.out
Starting secondary namenodes [Master]
Master: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-secondarynamenode-Master.out

```

```
hadoop@Master:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn
-hadoop-resourcemanager-Master.out
Slave2: starting nodemanager, logging to /usr/local/hadoop/logs/
yarn-hadoop-nodemanager-Slave2.out
Slave1: starting nodemanager, logging to /usr/local/hadoop/logs/
yarn-hadoop-nodemanager-Slave1.out
hadoop@Master:~$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /usr/local/hadoop/logs/mapred
-hadoop-historyserver-Master.out
hadoop@Master:~$ start-hbase.sh
Slave2: starting zookeeper, logging to /usr/local/hbase/bin/..1
ogs/hbase-hadoop-zookeeper-Slave2.out
Slave1: starting zookeeper, logging to /usr/local/hbase/bin/..1
ogs/hbase-hadoop-zookeeper-Slave1.out
Master: starting zookeeper, logging to /usr/local/hbase/bin/..1
ogs/hbase-hadoop-zookeeper-Master.out
starting master, logging to /usr/local/hbase/logs/hbase-hadoop-m
aster-Master.out
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermS
ize=128m; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPe
rmSize=128m; support was removed in 8.0
Slave1: starting regionserver, logging to /usr/local/hbase/bin/..
./logs/hbase-hadoop-regionserver-Slave1.out
Slave2: starting regionserver, logging to /usr/local/hbase/bin/..
./logs/hbase-hadoop-regionserver-Slave2.out
Slave1: Java HotSpot(TM) 64-Bit Server VM warning: ignoring opti
on PermSize=128m; support was removed in 8.0
Slave1: Java HotSpot(TM) 64-Bit Server VM warning: ignoring opti
on MaxPermSize=128m; support was removed in 8.0
Slave2: Java HotSpot(TM) 64-Bit Server VM warning: ignoring opti
on PermSize=128m; support was removed in 8.0
Slave2: Java HotSpot(TM) 64-Bit Server VM warning: ignoring opti
on MaxPermSize=128m; support was removed in 8.0
hadoop@Master:~$
```

2 如果源HBASE集群和备份HBASE集群已经启动了,请将其关闭,检查HMASTER守护进程是否已启动,以确认源集群上的HBASE已被关闭.我们把HBASE目录备份到集群上的/backup目录中.请在备份集群的Hadoop客户端上使用如下命令预先创建好该目录.

```

hadoop@Master:~$ hadoop fs -mkdir /backup
hadoop@Master:~$ hadoop fs -ls
Found 7 items
drwxr-xr-x  - hadoop supergroup          0 2016-02-25 18:41 .sp
arkStaging
drwxr-xr-x  - hadoop supergroup          0 2016-03-03 02:24 cds
gus
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 21:32 chu
888chu888
drwxrwxr-x  - hadoop supergroup          0 2016-01-28 18:54 hiv
e
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 22:15 inp
ut
drwxr-xr-x  - hadoop supergroup          0 2016-01-27 22:19 out
put
drwxrwxr-x  - hadoop supergroup          0 2016-01-28 18:53 tmp
hadoop@Master:~$ hadoop fs -ls /
Found 5 items
drwxr-xr-x  - hadoop supergroup          0 2016-05-07 00:44 /ba
ckup
drwxr-xr-x  - hadoop supergroup          0 2016-04-30 00:23 /hb
ase
drwxr-xr-x  - hadoop supergroup          0 2016-04-03 18:43 /in
put
drwxrwx---  - hadoop supergroup          0 2016-01-28 19:16 /tm
p
drwxr-xr-x  - hadoop supergroup          0 2016-03-03 01:00 /us
er

hadoop@Master:~$ stop-hbase.sh

hadoop@Master:~$ jps
1984 JobHistoryServer
1553 SecondaryNameNode

```

```
2290 Jps
1698 ResourceManager
1336 NameNode
hadoop@Master:~$
```

请确认上述输出结果中未列出HMaster守护进程

3 使用distcp命令从源集群将HBASE根目录复制到备份集群中.HBASE的根目录由HBASE配置文件(hbase-site.xml)中的hbase.rootdir属性所指定.

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://Master:9000/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>Master,Slave1,Slave2</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/hadoop</value>
  </property>
</configuration>
```

```
hadoop@Master:/$ hadoop distcp hdfs://Master:9000/hbase hdfs://Master:9000/backup
16/05/07 01:02:37 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=false, syncFolder=false, deleteMissing=false, ignoreFailures=false, maxMaps=20, sslConfigurationFile='null', copyStrategy='uniformsize', sourceFileListing=null, sourcePaths=[hdfs://Master:9000/hbase], targetPath=hdfs://Master:9000/backup, targetPathExists=true, preserveRawXattrs=false}
16/05/07 01:02:37 INFO client.RMProxy: Connecting to ResourceManager at Master/192.168.1.80:8032
```

```
16/05/07 01:02:38 INFO Configuration.deprecation: io.sort.mb is
deprecated. Instead, use mapreduce.task.io.sort.mb
16/05/07 01:02:38 INFO Configuration.deprecation: io.sort.factor
is deprecated. Instead, use mapreduce.task.io.sort.factor
16/05/07 01:02:38 INFO client.RMProxy: Connecting to ResourceMan
ager at Master/192.168.1.80:8032
16/05/07 01:02:39 INFO mapreduce.JobSubmitter: number of splits:
5
16/05/07 01:02:39 INFO mapreduce.JobSubmitter: Submitting tokens
for job: job_1462550883601_0002
16/05/07 01:02:39 INFO impl.YarnClientImpl: Submitted applicatio
n application_1462550883601_0002
16/05/07 01:02:39 INFO mapreduce.Job: The url to track the job:
http://Master:8088/proxy/application_1462550883601_0002/
16/05/07 01:02:39 INFO tools.DistCp: DistCp job-id: job_14625508
83601_0002
16/05/07 01:02:39 INFO mapreduce.Job: Running job: job_146255088
3601_0002
16/05/07 01:02:44 INFO mapreduce.Job: Job job_1462550883601_0002
running in uber mode : false
16/05/07 01:02:44 INFO mapreduce.Job: map 0% reduce 0%
16/05/07 01:02:55 INFO mapreduce.Job: map 40% reduce 0%
16/05/07 01:02:56 INFO mapreduce.Job: map 60% reduce 0%
16/05/07 01:02:57 INFO mapreduce.Job: map 76% reduce 0%
16/05/07 01:02:58 INFO mapreduce.Job: map 80% reduce 0%
16/05/07 01:03:01 INFO mapreduce.Job: map 100% reduce 0%
16/05/07 01:03:02 INFO mapreduce.Job: Job job_1462550883601_0002
completed successfully
16/05/07 01:03:03 INFO mapreduce.Job: Counters: 33
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=539820
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=345874159
    HDFS: Number of bytes written=345774967
    HDFS: Number of read operations=2275
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=587
```

```

Job Counters
  Launched map tasks=5
  Other local map tasks=5
  Total time spent by all maps in occupied slots (ms)=4940
9
  Total time spent by all reduces in occupied slots (ms)=0
  Total time spent by all map tasks (ms)=49409
  Total vcore-seconds taken by all map tasks=49409
  Total megabyte-seconds taken by all map tasks=50594816
Map-Reduce Framework
  Map input records=404
  Map output records=0
  Input split bytes=680
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=996
  CPU time spent (ms)=8070
  Physical memory (bytes) snapshot=582979584
  Virtual memory (bytes) snapshot=9369141248
  Total committed heap usage (bytes)=162856960
File Input Format Counters
  Bytes Read=98512
File Output Format Counters
  Bytes Written=0
org.apache.hadoop.tools.mapred.CopyMapper$Counter
  BYTESCOPIED=345774967
  BYTESEXPECTED=345774967
  COPY=404
hadoop@Master:/$

```

4 如果想把数据恢复直接拷贝回去即可

### 3 使用**CopyTable**在表间复制数据

**CopyTable**是一个实用程序,它可将一张表中的数据复制到同一个集群或其他HBASE集群的另一张表中.你可以仅将数据复制到同一集群的另一张表中,但如果还有其他可做为备份的集群,你可能更愿意以一种联机备份的方式用**CopyTable**来将表中的数

据复制到备份集群中.

CopyTable还可以带开始和结束时间两个参数.如果指定开始时间和结束时间,该命令就会只对那些时间戳在指定时间范围内的数据进行复制.这一特性使该命令在某些情况下可以对HBASE表进行增量备份.

### 4 将**HBase** 表导出为**HDFS**上的转储文件

### 5 通过从**HDFS**导入转储文件来恢复**HBASE**数据

## 7 监控与诊断

# 副录-HBase资源收集

## 1. Apache HBase™ 参考指南

## 第十三章 Spark

# 安装

## 基于**YARN**的部署方案

### 1. 软件环境：

```
Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)
Hadoop: 2.6.0
Spark: 1.6.0
```

### 2. 环境准备

#### 修改主机名

我们将搭建1个Master，2个Slave的集群方案。首先修改主机名nano  
`/etc/hostname`，在Master上修改为Master，其中一个Slave上修改为Slave1，另一个同理。

#### 配置**hosts**

在每台主机上修改host文件

```
nano /etc/hosts

192.168.1.80      Master
192.168.1.82      Slave1
192.168.1.84      Slave2
```

配置之后ping一下用户名看是否生效

```
ping Slave1
ping Slave2
```

### 3. **SSH** 免密码登录

## 1. 基YARN安装

### 安装OpenSSH server

```
sudo apt-get install openssh-server
```

在所有机器上都生成私钥和公钥

```
ssh-keygen -t rsa #一路回车
```

需要让机器间都能相互访问，就把每个机子上的id\_rsa.pub发给Master节点，传输公钥可以用scp来传输。

```
scp ~/.ssh/id_rsa.pub spark@Master:~/.ssh/id_rsa.pub.Slave1
```

在Master上，将所有公钥加到用于认证的公钥文件authorized\_keys中

```
cat ~/.ssh/id_rsa.pub* >> ~/.ssh/authorized_keys
```

将公钥文件authorized\_keys分发给每台Slave

```
scp ~/.ssh/authorized_keys spark@Master:~/.ssh/
```

在每台机子上验证SSH无密码通信

```
ssh Master  
ssh Slave1  
ssh Slave2
```

如果登陆测试不成功，则可能需要修改文件authorized\_keys的权限（权限的设置非常重要，因为不安全的设置安全设置，会让你不能使用RSA功能）

```
chmod 600 ~/.ssh/authorized_keys
```

## 4. 安装 Java

从官网下载最新版Java就可以，Spark官方说明Java只要是6以上的版本都可以，我下的是jdk-7u75-linux-x64.gz在~/workspace目录下直接解压

```
tar -zxvf jdk-7u75-linux-x64.gz
```

修改环境变量sudo vi /etc/profile，添加下列内容，注意将home路径替换成你的：

## 1. 基YARN安装

```
export JAVA_HOME=/usr/lib/jvm/
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:/usr/local/hive/lib
export PATH=${JAVA_HOME}/bin:$PATH

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin:/usr/local/hbase/bin

export JAVA_LIBRARY_PATH=/usr/local/hadoop/lib/native

export SCALA_HOME=/usr/lib/scala
export PATH=$PATH:$SCALA_HOME/bin
```

然后使环境变量生效，并验证 Java 是否安装成功

```
$ source /etc/profile      #生效环境变量
$ java -version          #如果打印出如下版本信息，则说明安装成功
java version "1.7.0_75"
Java(TM) SE Runtime Environment (build 1.7.0_75-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.75-b04, mixed mode)
```

## 5. 安装 Scala

## 1. 基YARN安装

Spark官方要求 Scala 版本为 2.10.x，注意不要下错版本，我这里下了 2.10.6

```
tar -zxvf scala-2.10.4.tgz
```

再次修改环境变量 `sudo vi /etc/profile`，添加以下内容：

```
export SCALA_HOME=/usr/lib/scala
```

```
export PATH=$PATH:$SCALA_HOME/bin
```

同样的方法使环境变量生效，并验证 `scala` 是否安装成功

```
$ source /etc/profile #生效环境变量
```

```
$ scala -version #如果打印出如下版本信息，则说明安装成功
```

```
Scala code runner version 2.10.6 -- Copyright 2002-2013, LAMP/EP  
FL
```

## 6. 安装配置 Hadoop YARN

此处参考以前的安装过程

## 7. Spark安装

下载解压进入官方下载地址下载最新版 Spark。我下载的是 `spark-1.6.0-bin-hadoop2.6.tgz`。

## 1. 基YARN安装

```
tar -zxvf spark-1.6.0-bin-hadoop2.6.tgz  
mv spark-1.6.0-bin-hadoop2.4 /usr/local/spark
```

配置 Spark

```
chmod -R 775 /usr/local/spark  
chown -R hadoop:hadoop /usr/local/spark  
cd /usr/local/spark/conf      #进入spark配置目录  
cp spark-env.sh.template spark-env.sh    #从配置模板复制  
vi spark-env.sh      #添加配置内容  
  
export SCALA_HOME=/usr/lib/scala  
export JAVA_HOME=/usr/lib/jvm/  
export HADOOP_HOME=/usr/local/hadoop  
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop  
SPARK_MASTER_IP=Master  
SPARK_LOCAL_DIRS=/usr/local/spark  
SPARK_DRIVER_MEMORY=1G
```

注：在设置Worker进程的CPU个数和内存大小，要注意机器的实际硬件条件，如果配置的超过当前Worker节点的硬件条件，Worker进程会启动失败。

## 1. 基YARN安装

nano Slaves 在 slaves 文件下填上 Slave 主机名：

Slave1

Slave2

将配置好的 spark-1.6.0 文件夹分发给所有 Slaves 吧

启动 Spark

sbin/start-all.sh

验证 Spark 是否安装成功

用 jps 检查，在 master 上应该有以下几个进程：

```
$ jps  
7949 Jps  
7328 SecondaryNameNode  
7805 Master  
7137 NameNode  
7475 ResourceManager
```

在 slave 上应该有以下几个进程：

```
$jps  
3132 DataNode  
3759 Worker  
3858 Jps  
3231 NodeManager
```

进入 Spark 的 Web 管理页面： <http://192.168.1.80:8080>

## 8. 运行示例

本地模式两线程运行

./bin/run-example SparkPi 10 --master local[2]

**Spark Standalone** 集群模式运行

## 1. 基YARN安装

```
./spark-submit --class org.apache.spark.examples.SparkPi --master spark://Master:7077 /usr/local/spark/lib/spark-examples-1.6.0-hadoop2.6.0.jar 100
```

### Spark on YARN 集群上 **yarn-cluster** 模式运行

```
./spark-submit --class org.apache.spark.examples.SparkPi --master yarn-cluster /usr/local/spark/lib/spark-examples-1.6.0-hadoop2.6.0.jar 10
```

Spark 1.6.0 Spark Master at spark://Master:7077

URL: spark://Master:7077  
REST URL: spark://Master:6066 (cluster mode)  
Alive Workers: 2  
Cores in use: 2 Total, 0 Used  
Memory in use: 2.0 GB Total, 0.0 B Used  
Applications: 0 Running, 1 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20160225183543-192.168.1.82-42697	192.168.1.82:42697	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20160225183543-192.168.1.84-45676	192.168.1.84:45676	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

**Running Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160225184313-0000	Spark Pi	2	1024.0 MB	2016/02/25 18:43:13	hadoop	FINISHED	16 s

**Completed Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160225184313-0000	Spark Pi	2	1024.0 MB	2016/02/25 18:43:13	hadoop	FINISHED	16 s

注意 Spark on YARN 支持两种运行模式，分别为**yarn-cluster**和**yarn-client**，具体的区别可以看这篇博文，从广义上讲，**yarn-cluster**适用于生产环境；而**yarn-client**适用于交互和调试，也就是希望快速地看到application的输出。

## 3.Hadoop与Spark的区别

### 概述

谈到大数据，相信大家对Hadoop和Apache Spark这两个名字并不陌生。但我们往往对它们的理解只是停留在字面上，并没有对它们进行深入的思考，下面不妨跟我一块看下它们究竟有什么异同。

### 解决问题的层面不一样

首先，Hadoop和Apache Spark两者都是大数据框架，但是各自存在的目的不尽相同。Hadoop实质上更多是一个分布式数据基础设施：它将巨大的数据集分派到一个由普通计算机组成的集群中的多个节点进行存储，意味着您不需要购买和维护昂贵的服务器硬件。

同时，Hadoop还会索引和跟踪这些数据，让大数据处理和分析效率达到前所未有的高度。Spark，则是那么一个专门用来对那些分布式存储的大数据进行处理的工具，它并不会进行分布式数据的存储。

### 两者可合可分

Hadoop除了提供为大家所共识的HDFS分布式数据存储功能之外，还提供了叫做MapReduce的数据处理功能。所以这里我们完全可以抛开Spark，使用Hadoop自身的MapReduce来完成数据的处理。

相反，Spark也不是非要依附在Hadoop身上才能生存。但如上所述，毕竟它没有提供文件管理系统，所以，它必须和其他的分布式文件系统进行集成才能运作。这里我们可以选择Hadoop的HDFS，也可以选择其他的基于云的数据系统平台。但Spark默认来说还是被用在Hadoop上面的，毕竟，大家都认为它们的结合是最好的。

以下是天地会珠海分舵从网上摘录的对MapReduce的最简洁明了的解析：

我们要数图书馆中的所有书。你数1号书架，我数2号书架。这就是“Map”。我们人越多，数书就更快。

现在我们到一起，把所有人的统计数加在一起。这就是“Reduce”。

## Spark数据处理速度秒杀MapReduce

Spark因为其处理数据的方式不一样，会比MapReduce快上很多。MapReduce是分步对数据进行处理的：“从集群中读取数据，进行一次处理，将结果写到集群，从集群中读取更新后的数据，进行下一次的处理，将结果写到集群，等等...”Booz Allen Hamilton的数据科学家Kirk Borne如此解析。

反观Spark，它会在内存中以接近“实时”的时间完成所有的数据分析：“从集群中读取数据，完成所有必须的分析处理，将结果写回集群，完成，”Born说道。Spark的批处理速度比MapReduce快近10倍，内存中的数据分析速度则快近100倍。

如果需要处理的数据和结果需求大部分情况下是静态的，且你也有耐心等待批处理的完成的话，MapReduce的处理方式也是完全可以接受的。

但如果你需要对流数据进行分析，比如那些来自于工厂的传感器收集回来的数据，又或者说你的应用是需要多重数据处理的，那么你也许更应该使用Spark进行处理。

大部分机器学习算法都是需要多重数据处理的。此外，通常会用到Spark的应用场景有以下方面：实时的市场活动，在线产品推荐，网络安全分析，机器日记监控等。

## 灾难恢复

两者的灾难恢复方式迥异，但是都很不错。因为Hadoop将每次处理后的数据都写入到磁盘上，所以其天生就能很有弹性的对系统错误进行处理。

Spark的数据对象存储在分布于数据集群中的叫做弹性分布式数据集(RDD: Resilient Distributed Dataset)中。“这些数据对象既可以放在内存，也可以放在磁盘，所以RDD同样也可以提供完成的灾难恢复功能，”Borne指出。

## 第十五章 CDH的发行版本

# 离线安装Cloudera Manager 5和CDH5(5.2.3)

CDH (Cloudera's Distribution, including Apache Hadoop)，是Hadoop众多分支中的一种，由Cloudera维护，基于稳定版本的Apache Hadoop构建，并集成了很多补丁，可直接用于生产环境。

Cloudera Manager则是为了便于在集群中进行Hadoop等大数据处理相关的服务安装和监控管理的组件，对集群中主机、Hadoop、Hive、Spark等服务的安装配置管理做了极大简化。

## 一 系统环境

- 实验环境：Mac下VMware虚拟机
- 操作系统：CentOS 6.5 x64 (至少内存2G以上，这里内存不够的同学建议还是整几台真机配置比较好，将CDH的所有组件全部安装会占用很多内存，我已开始设置的虚拟机内存是1G，安装过程中直接卡死了)
- Cloudera Manager : 5.2.3
- CDH: 5.2.3

## 二 安装说明

### 1 官方参考文档：

官方安装文档:[地址](#)

官方共给出了3中安装方式：

第一种方法必须要求所有机器都能连网，由于最近各种国外的网站被墙的厉害，我尝试了几次各种超时错误，巨耽误时间不说，一旦失败，重装非常痛苦。

第二种方法下载很多包。

第三种方法对系统侵入性最小，最大优点可实现全离线安装，而且重装什么的都非常方便。后期的集群统一包升级也非常好。这也是我之所以选择离线安装的原因。

## 2 Cloudera Manager下载地址：

[下载地址](#)

## 3 Cloudera Manager安装要求信息：

[浏览地址](#)

## 4 CDH安装包下载地址：

<http://archive.cloudera.com/cdh5/parcels/latest/>，

由于我们的操作系统为CentOS6.5，需要下载以下文件：

CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel

CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel.sha1

manifest.json

## 5 注意：

与CDH4的不同，原来安装CDH4的时候还需要下载IMPALA、Cloudera Search(SOLR)，CDH5中将他们包含在一起了，所以只需要下载一个CDH5的包就可以了。

而在咱们公司内部可以在内部的ftp服务器（192.168.1.110）都能找到所需要的文件，在下面我将会把这些文件的拷贝方法一同写入到里面。

## 三 准备工作：系统环境搭建

以下操作均用root用户操作。

### 1. 网络配置(所有节点)

```
vi /etc/sysconfig/network 修改hostname：
```

```
NETWORKING=yes  
HOSTNAME=dhc-4
```

通过 `service network restart` 重启网络服务生效。

`vi /etc/hosts` ,修改ip与主机名的对应关系

```
192.168.1.213 dhc-4
192.168.1.214 dhc-5
192.168.1.215 dhc-6
```

## 2 注意：

这里需要将每台机器的ip及主机名对应关系都写进去，本机的也要写进去，否则启动Agent的时候会提示hostname解析错误。

## 四 打通SSH，设置ssh无密码登陆（所有节点）

在所有节点上执行 `ssh-keygen -t rsa` 一路回车，生成无密码的密钥对。

将主节点公钥添加到认证文件中：`cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`，并设置`authorized_keys`的访问权限：`chmod 600 ~/.ssh/authorized_keys`。

分别使用以下命令scp文件到所有datenode节点：

```
scp ~/.ssh/authorized_keys root@dhc-5:~/.ssh/
scp ~/.ssh/authorized_keys root@dhc-6:~/.ssh/
```

## 1 测试：

在主节点上ssh n2，正常情况下，不需要密码就能直接登陆进去了。

## 五. 安装Oracle的Java（所有节点）

CentOS，自带OpenJdk，不过运行CDH5需要使用Oracle的Jdk，需要Java 7的支持。

卸载自带的OpenJdk，使用 `rpm -qa | grep java` 查询java相关的包，使用 `rpm -e --nodeps 包名` 卸载之。

## 1. 基于Centos的安装

去Oracle的官网下载jdk的rpm安装包，并使用 `rpm -ivh` 包名 安装之。  
而咱们公司中的jdk1.7放在：

```
/home/hadoop/Hadoop/CDH/CDHCentOS6/jdk-7u80-linux-x64.rpm
```

可以使用命令

`scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/jdk-7u80-linux-x64.rpm /usr/local` 将jdk1.7放在`/usr/local`目录下，然后进入到此目录下，使用命令 `rpm -ivh jdk-7u80-linux-x64.rpm` 安装jdk1.7。  
由于是rpm包并不需要我们来配置环境变量，我们只需要配置一个全局的 `JAVA_HOME` 变量即可，执行命令：

```
echo "JAVA_HOME=/usr/java/latest/" >> /etc/environment
```

## 六. 安装配置 MySql (主节点)

在安装mysql数据库之前，先使用命令

查看：`rpm -qa | grep mysql`

卸载：`rpm -e --nodeps mysql-libs-5.1.71-1.el6.x86_64`

将本机自带的mysql数据库卸载。

在线安装可以通过 `yum install mysql-server` 本例中，在本地ftp服务器上下载了一个MySQL的安装包，可以使用命令：

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/MySQL-5.5.49-1.el6.x86_64.rpm-bundle.tar /usr/local
```

将MySQL的安装包放在`/usr/local/`目录下，进入到此目录下，执行命令：

```
tar -xvf MySQL-5.5.49-1.el6.x86_64.rpm-bundle.tar
```

进行解压，解压完成之后会出现几个以.rpm为结尾的安装包，我们只需要安装 MySQL-server 和 MySQL-client 这两个包即可，可以使用命令：`rpm -ivh +包名` 即可安装。

## 1. 基于Centos的安装

安装完成以后使用命令 `service mysql start` 打开MySQL服务，并根据提示设置root的初试密码：

```
mysqladmin -u root password 'xxxx' 。
```

```
mysql -uroot -p123456 进入mysql命令行，创建以下数据库：
```

```
#hive
create database hive DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
#activity monitor
create database amon DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
```

设置root授权访问以上所有的数据库：

```
#授权root用户在主节点拥有所有数据库的访问权限
#grant all privileges on *.* to 'root'@'dhc-4' identified by '12
3456' with grant option;
#flush privileges;
```

## 官方**MySql**配置文档：

[http://www.cloudera.com/content/cloudera/en/documentation/cloudera-manager/v5-latest/Cloudera-Manager-Installation-Guide/cm5ig\\_mysql.html#cmig\\_topic\\_5\\_5](http://www.cloudera.com/content/cloudera/en/documentation/cloudera-manager/v5-latest/Cloudera-Manager-Installation-Guide/cm5ig_mysql.html#cmig_topic_5_5)

## 七. 关闭防火墙和**SELinux**

### 注意：

需要在所有的节点上执行，因为涉及到的端口太多了，临时关闭防火墙是为了安装起来更方便，安装完毕后可以根据需要设置防火墙策略，保证集群安全。

关闭防火墙：

```
service iptables stop (临时关闭)
chkconfig iptables off (重启后生效)
```

关闭SELINUX（实际安装过程中发现没有关闭也是可以的，不知道会不会有问题，还需进一步进行验证）：

```
setenforce 0 (临时生效)
修改 /etc/selinux/config 下的 SELINUX=disabled (重启后永久生效)
```

## 八. 所有节点配置NTP服务（参照：补充说明1）

集群中所有主机必须保持时间同步，如果时间相差较大会引起各种问题。具体思路如下：

master节点作为ntp服务器与外界对时中心同步时间，随后对所有datanode节点提供时间同步服务。

所有datanode节点以master节点为基础同步时间。

所有节点安装相关组件：`yum install ntp`。完成后，配置开机启动：  
`chkconfig ntpd on`，检查是否设置成功：`chkconfig --list ntpd` 其中2-5为on状态就代表成功。

### 1 主节点配置

在配置之前，先使用`ntpdate`手动同步一下时间，免得本机与对时中心时间差距太大，使得`ntp`不能正常同步。这里选用65.55.56.206作为对时中心，`ntpdate -u 65.55.56.206`。

`ntp`服务只有一个配置文件，配置好了就OK。这里只给出有用的配置，不需要的配置都用#注掉，这里就不在给出：

```
driftfile /var/lib/ntp/drift
restrict 127.0.0.1
restrict -6 ::1
restrict default nomodify notrap
server 65.55.56.206 prefer
includefile /etc/ntp/crypto/pw
keys /etc/ntp/keys
```

配置文件完成，保存退出，启动服务，执行如下命令：`service ntpd start`

检查是否成功，用`ntpq -p`命令查看同步状态，出现以下状态代表启动成功：

```
synchronised to NTP server () at stratum 2
time correct to within 74 ms
polling server every 128 s
```

如果出现异常请等待几分钟，一般等待5-10分钟才能同步。

## 2 配置ntp客户端（所有datanode节点）

```
driftfile /var/lib/ntp/drift
restrict 127.0.0.1
restrict -6 ::1
restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery
#这里是主节点的主机名或者ip
server n1
includefile /etc/ntp/crypto/pw
keys /etc/ntp/keys
```

ok保存退出，请求服务器前，请先使用`ntpdate -u n1`（主节点ntp服务器）手动同步一下时间：

这里可能出现同步失败的情况，请不要着急，一般是本地的ntp服务器还没有正常启动，一般需要等待5-10分钟才可以正常同步。启动服务：`service ntpd start`

因为是连接内网，这次启动等待的时间会比master节点快一些，但是也需要耐心等待一会儿。

补充说明1：由于咱们使用的服务器没有联网，因此这一步可省略，但是你要检查一下你的三台主机之间的时间是否差异太大，如果太大将会对下面的安装造成不必要的麻烦，可以用 `date` 命令查看当前主机的时间，然后如果发现差异太大的话，使用命令：`date -s + 时间点 (主节点上面的时间点)` 进行时间同步。

## 九 安装Cloudera Manager Server 和 Agent

### 1 主节点解压安装

咱们公司内部的Cloudera manager存放在ftp服务器上，可以使用命令：

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/cdh5.3.2/cloudera-manager-el6-cm5.3.2_x86_64.tar.gz /opt
```

cloudera manager的目录默认位置在/opt下，解压：`tar xzvf cloudera-manager*.tar.gz` 将解压后的cm-5.2.3和cloudera目录放到/opt目录下。

### 2 为Cloudera Manager 5建立数据库

首先需要去MySql的官网下载JDBC驱动，

<http://dev.mysql.com/downloads/connector/j/>，解压后，找到mysql-connector-java-5.1.33-bin.jar，放到/opt/cm-5.1.3/share/cmflib/中。

本公司内部可以使用命令：

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/mysql-connector-java-5.1.36.tar.gz /usr/local
```

将数据库的JDBC驱动放在/usr/local/目录下，进入到此目录下，进行解压：`tar -zxvf mysql-connector-java-5.1.36.tar.gz`。解压后，找到mysql-connector-java-5.1.36-bin.jar，放到/opt/cm-5.3.2/share/cmflib/中。

在主节点初始化CM5的数据库：

```
/opt/cm-5.3.2/share/cmf/schema/scm_prepare_database.sh mysql cm  
-hlocalhost -uroot -p123456 --scm-host localhost scm scm scm
```

## 十 Agent配置

修改/opt/cm-5.3.2/etc/cloudera-scm-agent/config.ini中的server\_host为主节点的主机名。

### 1 同步Agent到其他节点

```
scp -r /opt/cm-5.3.2 root@dhc-5:/opt/  
scp -r /opt/cm-5.3.2 root@dhc-6:/opt/
```

### 2 在所有节点创建cloudera-scm用户

```
useradd --system --home=/opt/cm-5.3.2/run/cloudera-scm-server/ -  
-no-create-home --shell=/bin/false --comment "Cloudera SCM User"  
cloudera-scm
```

### 3 准备Parcels，用以安装CDH5

将CHD5相关的Parcel包放到主节点的/opt/cloudera/parcel-repo/目录中，相关的文件如下：

```
CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel  
CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel.sha1  
manifest.json
```

本公司内部的这些文件可以使用命令：

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/cdh5  
.3.2/CDH-5.3.2-1.cdh5.3.2.p0.10-el6.parcel /opt/cloudera/parcel-  
repo/
```

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/cdh5  
.3.2/CDH-5.3.2-1.cdh5.3.2.p0.10-el6.parcel.sha1 /opt/cloudera/par-  
cel-repo/
```

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDHCentOS6/cdh5  
.3.2/manifest.json /opt/cloudera/parcel-repo/
```

最后将CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel.sha1，重命名为CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel.sha，这点必须注意，否则，系统会重新下载CDH-5.1.3-1.cdh5.1.3.p0.12-el6.parcel文件。

## 4 相关启动脚本

通过 `/opt/cm-5.3.2/etc/init.d/cloudera-scm-server start` 启动服务端。  
(只需主节点启动这项服务即可)

通过 `/opt/cm-5.3.2/etc/init.d/cloudera-scm-agent start` 启动Agent服务。  
(每个节点都要启动这项服务)

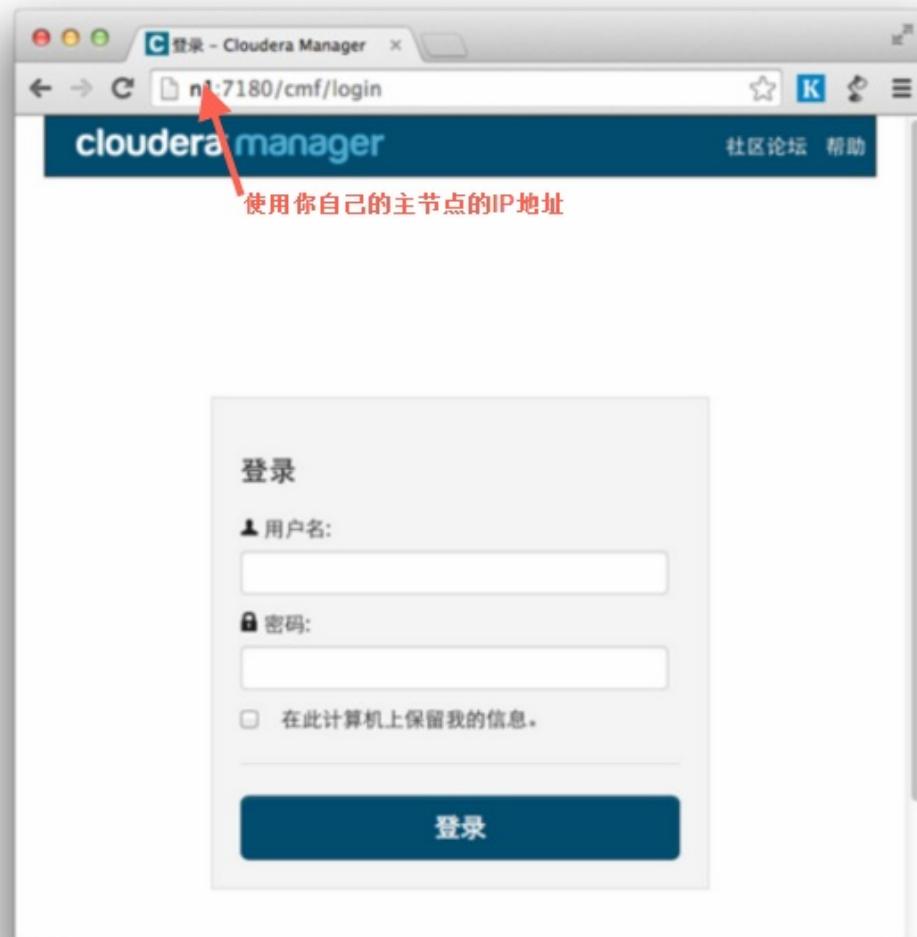
我们启动的其实是个service脚本，需要停止服务将以上的start参数改为stop就可以了，重启是restart。

## 十一 CDH5的安装配置

Cloudera Manager Server和Agent都启动以后，就可以进行CDH5的安装配置了。

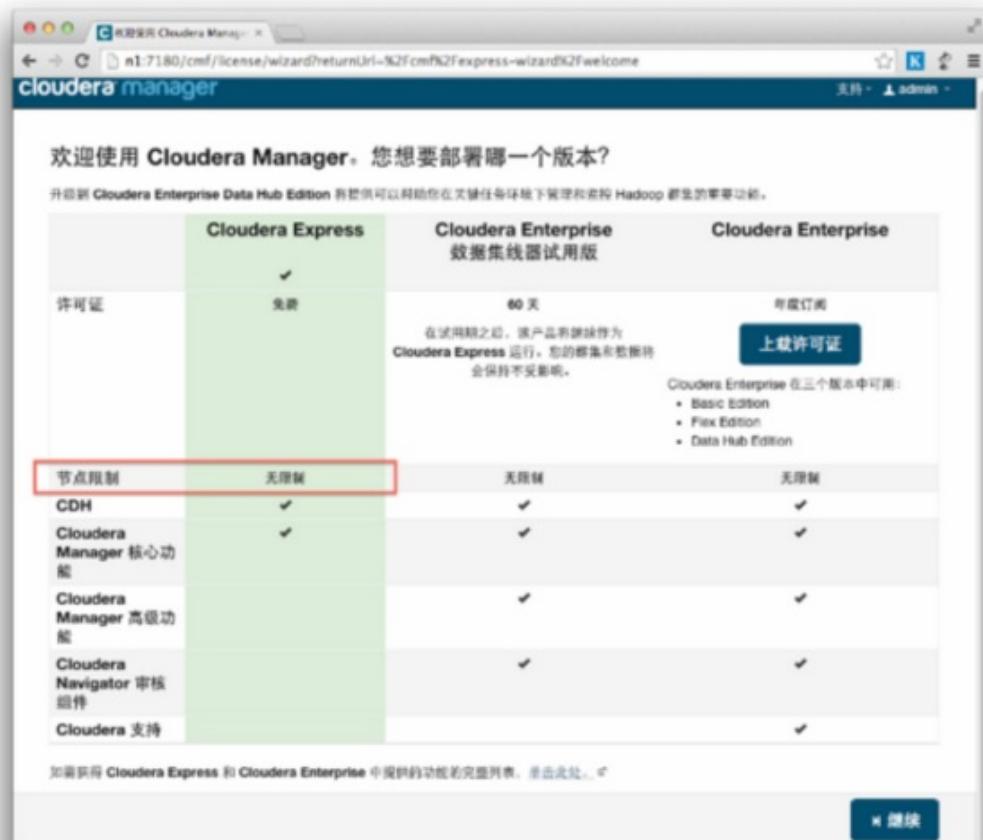
## 1. 基于Centos的安装

这时可以通过浏览器访问主节点的7180端口测试一下了（由于CM Server的启动需要花点时间，这里可能要等待一会才能访问），默认的用户名和密码均为admin：

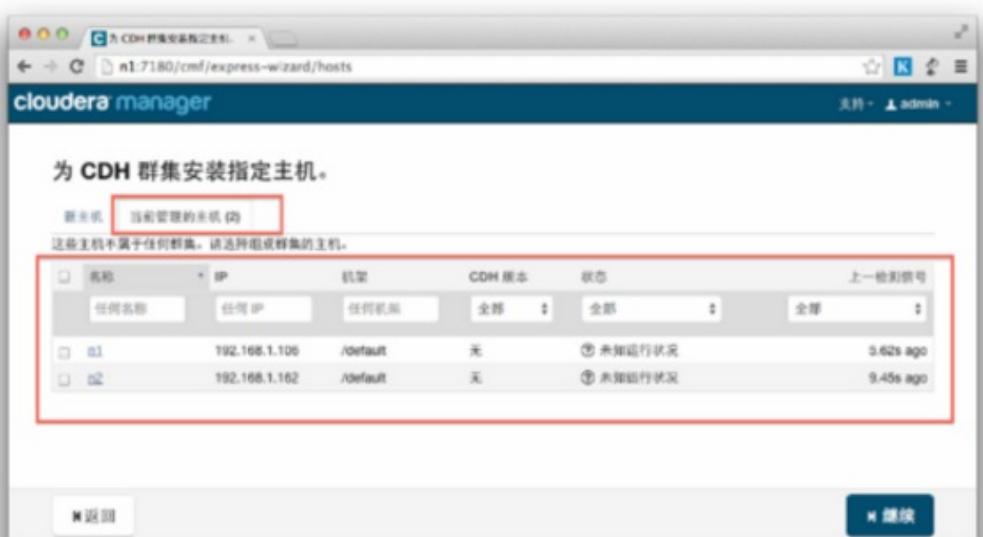


可以在下图中看到，免费版本的CM5已经没有原来50个节点数量的限制了。

## 1. 基于Centos的安装



各个Agent节点正常启动后，可以在当前管理的主机列表中看到对应的节点。选择要安装的节点，点继续。



接下来，出现以下包名，说明本地Parcel包配置无误，直接点继续就可以了。

## 1. 基于Centos的安装



点击，继续，如果配置本地Parcel包无误，那么下图中的已下载，应该是瞬间就完成了，然后就是耐心等待分配过程就行了，大约10多分钟吧，取决于内网网速。



接下来是服务器检查，可能会遇到以下问题：

Cloudera 建议将 /proc/sys/vm/swappiness 设置为 0。当前设置为 60。使用 sysctl 命令在运行时更改该设置并编辑 /etc/sysctl.conf 以在重启后保存该设置。您可以继续进行安装，但可能会遇到问题，Cloudera Manager 报告您的主机由于交换运行

## 1. 基于Centos的安装

状况不佳。以下主机受到影响：通过 `echo 0 > /proc/sys/vm/swappiness` (在主每个点的主机上运行) 即可解决。下面还有一个问题，是hive数据库缺少MySQL的jdbc的jar包，可以使用下面的命令：

```
cp /opt/cm-5.3.2/share/cmf/lib/mysql-connector-java-5.1.36-bin.jar /opt/cloudera/parcels/CDH-5.3.2-1.cdh5.3.2.p0.10/lib/hive/lib/
```

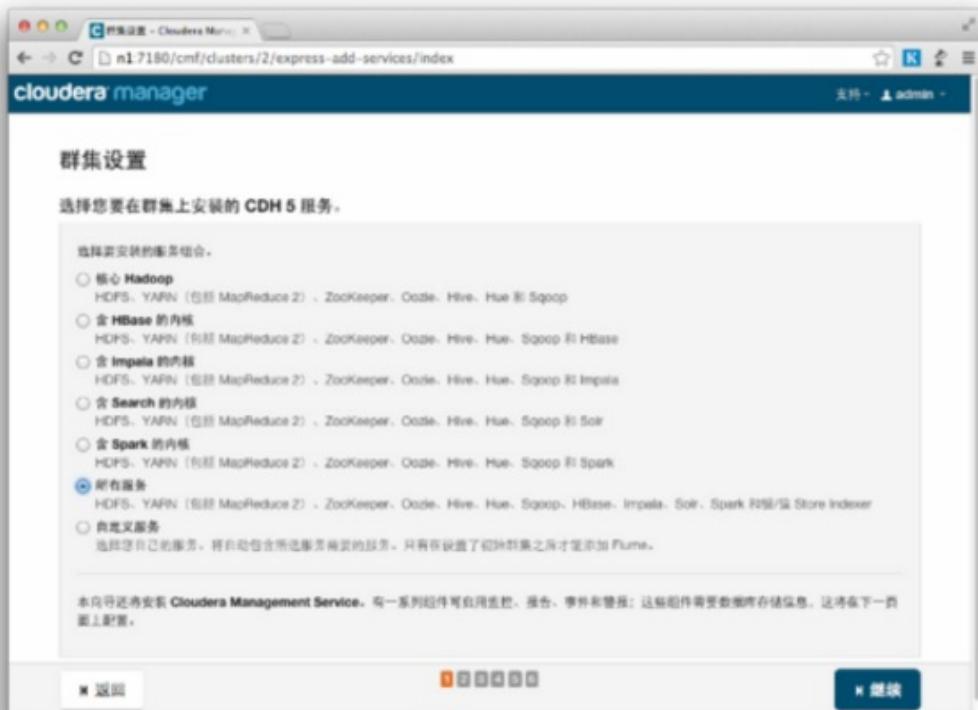
运行完成以后，点击继续。



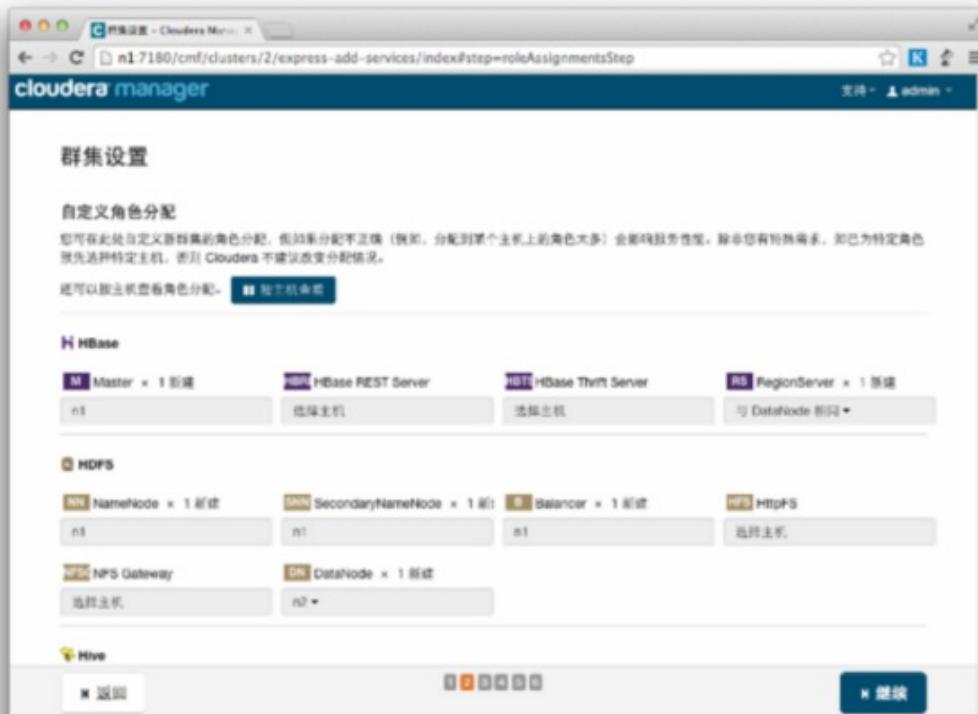
## 1. 基于Centos的安装

只有当所有选项前面都有对号之后才是正确的。

接下来是选择安装服务：

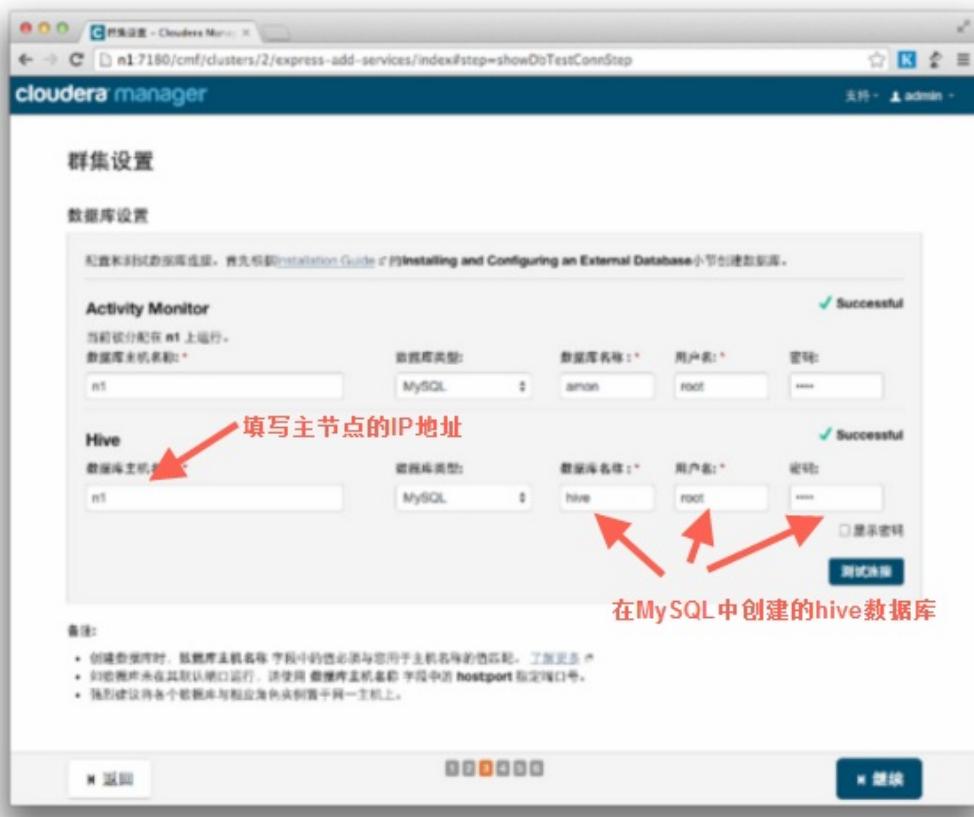


服务配置，一般情况下保持默认就可以了（Cloudera Manager会根据机器的配置自动进行配置，如果需要特殊调整，自行进行设置就可以了）：

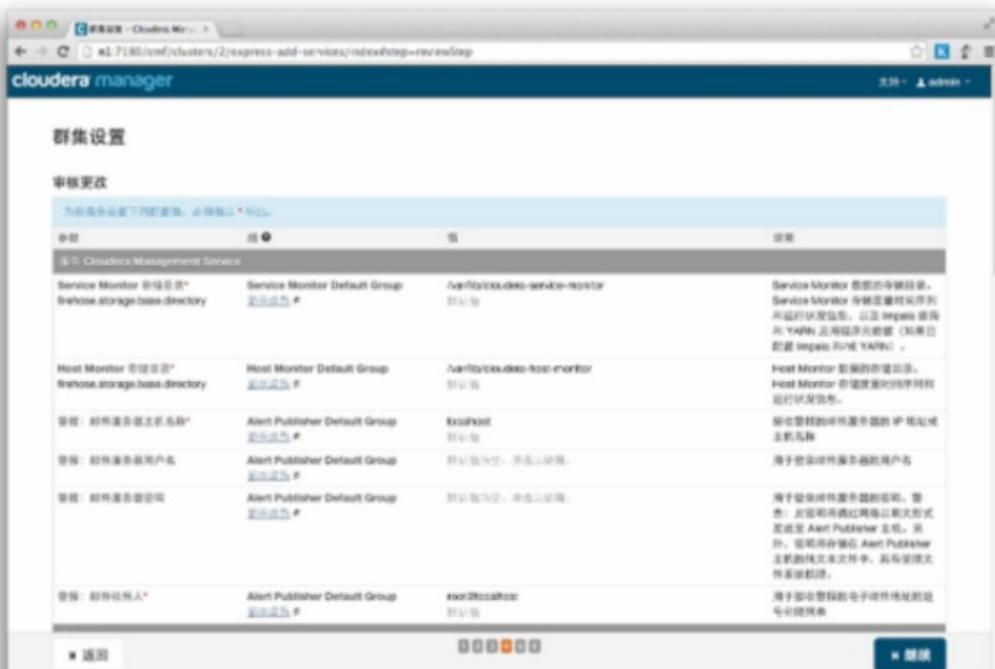


接下来是数据库的设置，检查通过后就可以进行下一步的操作了：

## 1. 基于Centos的安装



下面是集群设置的审查页面，我这里都是保持默认配置的：



终于到安装各个服务的地方了，注意，这里安装Hive的时候可能会报错，因为我们使用了MySQL作为hive的元数据存储，hive默认没有带mysql的驱动，通过以下命令拷贝一个就行了：

## 1. 基于Centos的安装

```
cp /opt/cm-5.3.2/share/cmflib/mysql-connector-java-5.1.36-bin.jar /opt/cloudera/parcels/CDH-5.3.2-1.cdh5.2.3.p0.12/lib/hive/lib/
``]
```



服务的安装过程大约半小时内就可以完成：



安装完成后，就可以进入集群界面看一下集群的当前状况了。

这里可能会出现 无法发出查询：对 Service Monitor 的请求超时 的错误提示，如果各个组件安装没有问题，一般是因为服务器比较卡导致的，过一会刷新一下页面就好了：



在每台主机上面修改一下ntp的配置就可：



还有一个要修改的地方，就是到最后的话会在HDFS上面出错，可以使用下面的命令进行配置：

```
sudo -u oozie bash hadoop fs -setrep -R 1 / ``
```

还有另外一个配置的地方只有等安装完成以后出错了才可以修改！

其他资料可参考：<http://www.tuicool.com/articles/ENjmeaY/>

## ubuntu14.04下关于CDH5离线安装教程

在安装一系列的安装包之前，首先要解决的是ubuntu14.04的无密码传输的root用户的权限问题。

### 1. 修改 root 密码

```
sudo passwd root
```

1.1如果使用普通用户安装时，首先创建用户和用户组：

```
sudo addgroup cdh
sudo adduser -ingroup cdh cdh
赋予用户 sudo 权限：
$ sudo nano /etc/sudoers #也可以使用visudo编辑
# User privilege specification
root    ALL=(ALL:ALL) ALL
cdh     ALL=(ALL:ALL) ALL
```

### 2. 以其他账户登录，通过 **sudo vim** 修改 **/etc/ssh/sshd\_config**：

```
xxx@ubuntu14:~$ su - root
Password:
root@ubuntu14:~# vi /etc/ssh/sshd_config
```

### 3. 注释掉 **#PermitRootLogin without-password**，添加 **PermitRootLogin yes**

```
# Authentication:  
LoginGraceTime 120  
#PermitRootLogin without-password  
PermitRootLogin yes  
StrictModes yes
```

### 4. 重启 ssh 服务

```
root@ubuntu14:~# sudo service ssh restart  
ssh stop/waiting  
ssh start/running, process 1499  
root@ubuntu14:~#
```

然后切换至root用户下：

```
#####1. 设置Host(所有节点)  
127.0.0.1      localhost  
#127.0.1.1      ubuntu1  
192.168.1.190    ubuntu1.cdh  
192.168.1.135    ubuntu2.cdh  
192.168.1.145    ubuntu3.cdh  
  
# The following lines are desirable for IPv6 capable hosts  
::1      localhost ip6-localhost ip6-loopback  
fe00::0 ip6-localnet  
ff00::0 ip6-mcastprefix  
ff02::1 ip6-allnodes  
ff02::2 ip6-allrouters
```

执行命令 `sudo vi /etc/hostname`  
在文件里写入自己修改后的名字，与上一步ip对应的名字一样。

执行命令 `shutdown -r now` 使配置生效。

### 2. 打通SSH，设置ssh无密码登陆（所有节点）

注意如果是以普通用户安装，配置的SSH免密码登录的对象是普通用户而不是root。在主节点上执行 `ssh-keygen -t rsa` 一路回车，生成无密码的密钥对。将公钥添加到认证文件中：`cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`，并设置`authorized_keys`的访问权限：`chmod 600 ~/.ssh/authorized_keys`。在每个节点上都执行以上命令，然后scp文件到所有datenode节点：

```
scp ~/.ssh/id_rsa.pub root@ubuntu2:/usr/local  
cat /usr/local/id_rsa.pub>>~/.ssh/authorized_keys  
chmod 600 ~/.ssh/authorized_keys  
测试： 在主节点上ssh ubuntu2.cdh，正常情况下，不需要密码就能直接登陆进去了。  
。
```

### 3关闭防火墙

```
root@m1:~# ufw disable
```

### 安装jdk

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/  
CDH/CDHUbuntu14.04/jdk-7u79-linux-x64.gz /usr/  
local
```

执行上述命令，获取jdk安装包

```
tar -zxvf jdk-7u79-linux-x64.gz  
mv jdk1.7.0_79/ jdk
```

解压jdk压缩包并改名（注意如果是普通用户安装的话不要改名了，而且最好安装路径为`/usr/java/jdk1.7.0_79`，而且配置环境变量要在此用户的`.bashrc`文件里配一下，应为即使使用`sudo`他也是先到`.bashrc`找环境配置）

```
vim ~/.bashrc
```

在打开的文件的末尾添加

```
export JAVA_HOME=/usr/local/jdk
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:
${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

保存退出，然后输入下面的命令来使之生效

```
source ~/.bashrc
```

## 安装配置**MySql**（主节点）

```
1. scp hadoop@192.168.1.110:/home/hadoop/
Hadoop/CDH/CDHUbuntu14.04/mysql-
server_5.7.12-1ubuntu14.04_amd64.deb-
bundle.tar /usr/local/
```

```
2. scp hadoop@192.168.1.110:/home/hadoop/
Hadoop/CDH/CDHCentOS6/mysql-connector-
java-5.1.36.tar.gz /usr/local/
```

执行上述两行命令，分别获取MySQL安装包和MySQL的连接jar包，并且存放  
在/usr/local目录下

```
1. scp hadoop@192.168.1.110:/home/hadoop/
Hadoop/CDH/CDHUbuntu14.04/libaio1_0.3.107-3ubuntu2_amd64.deb
/usr/local/
2. scp hadoop@192.168.1.110:/home/hadoop/
Hadoop/CDH/CDHUbuntu14.04/libmecab2_0.996-1.1_amd64.deb
/usr/local/
```

上述两条命令分别获取后面安装数据库需要用到的libaio1包和libmecab2包

```
tar -xvf mysql-server_5.7.12-1ubuntu14.04_amd64.deb-bundle.tar
```

执行上述命令解压缩包

解压开来后，一共有11个deb包，用`sudo dpkg -i [包名]`命令逐个安装，因为包与包中间存在依赖关系，这里安装有个先后顺序。我的安装的顺序是：

1. `mysql-common_5.7.12-1ubuntu14.04_amd64.deb`

2. `libmysqlclient20_5.7.12-1ubuntu14.04_amd64.deb`

3. `libmysqlclient-dev_5.7.12-1ubuntu14.04_amd64.deb`

4. `libmysqld-dev_5.7.12-1ubuntu14.04_amd64.deb`

5. 而后需要安装依赖包`libaio1`, 即 `libaio1_0.3.107-3ubuntu2_amd64.deb`

而后继续：

5. `mysql-community-client_5.7.12-1ubuntu14.04_amd64.deb`

6. `mysql-client_5.7.12-1ubuntu14.04_amd64.deb`

7. `mysql-community-source_5.7.12-1ubuntu14.04_amd64.deb`

6. 这里需要再安装一个依赖包叫`libmecab2`, 即`libmecab2_0.996-1.1_amd64.deb`

安装好后，继续安装最后一个：

8. `mysql-community-server_5.7.12-1ubuntu14.04_amd64.deb`

安装过程中需要设置数据库密码。

使用命令：`mysql -uroot -p123456`登录mysql，进入mysql命令行，创建以下数据库：

```
#hive  
create database hive DEFAULT CHARSET utf8 COLLATE utf8_general_ci;  
#activity monitor  
create database amon DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
```

设置root授权访问以上所有的数据库：

```
#授权root用户在主节点拥有所有数据库的访问权限  
grant all privileges on *.* to 'root'@'ubuntu1.cdh' identified by '123456' with grant option;  
flush privileges;
```

### 配置**MySql**的监听地址

```
root@m1:~# cp /etc/mysql/my.cnf /etc/mysql/my.cnf.bak  
root@m1:~# vi /etc/mysql/my.cnf  
#bind-address = 127.0.0.1  
bind-address = 0.0.0.0
```

这一步是必须的，如果不配置，后面再web上面安装的时候将会连接不到数据库  
然后执行命令

```
sudo service mysql restart
```

关闭防火墙

```
ufw disable
```

设置每台机器时间与主节点相同，即执行以下命令

```
date -s 主节点时间
```

### 安装 Cloudera Manager Server 和 Agents

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/cloudera-manage  
r-trusty-cm5.3.9_amd64.tar.gz /opt/
```

将cloudera-manager的安装包放在/opt

```
#
```

进入/opt目录下，对**cloudera-manager**安装包进行解压，使用命令：

```
tar -zxvf cloudera-manager-trusty-cm5.3.9_amd64.tar.gz
```

解压完成之后会出现一个目录 cm-5.3.9

在所有节点创建cloudera-scm用户

```
useradd --system --home=/opt/cm-5.3.9/run/cloudera-scm-server/ -  
-no-create-home --shell=/bin/false --comment "Cloudera SCM User"  
cloudera-scm
```

### 为**Cloudera Manager 5**建立数据库

将刚才下载的MySQL的连接jar包进行解压，文件位置是在：/usr/local,解压完成之后，使用命令：

```
mv /usr/local/mysql-connector-java-5.1.36/mysql-connector-java-5  
.1.36-bin.jar /opt/cm-5.3.9/share/cmflib/
```

将jar包放在/opt/cm-5.3.9/share/cmflib/目录下

在主节点初始化CM5的数据库：

```
/opt/cm-5.3.9/share/cmflschema/scm_prepare_database.sh mysql cm  
-hlocalhost  
-uroot -p123456 --scm-host localhost scm scm scm
```

修改**/opt/cm-5.3.9/etc/cloudera-scm-agent/config.ini**中的**server\_host**为主节点的主机名，例如本机主机名为**server-host=ubuntu1.cdh**

执行以下命令，把cm-5.3.9拷贝到其他机器的/opt文件夹

```
scp -r /opt/cm-5.3.9 root@ubuntu2.cdh:/opt/
```

## 准备**Packages**，用以安装**CDH5**

将CDH5相关的Parcel包放到主节点的**/opt/cloudera/parcel-repo/**目录中（**cloudera/parcel-repo**需要手动创建）。

```
CDH-5.3.9-1.cdh5.3.9.p0.8-trusty.parcel  
CDH-5.3.9-1.cdh5.3.9.p0.8-trusty.parcel.sha1  
manifest.json
```

使用命令：

```
scp hadoop@192.168.1.110:/home/hadoop/Hadoop/CDH/CDH-5.3.9-1.cdh  
5.3.9.p0.8-trusty.parcel /opt/cloudera/parcel-repo/
```

将ftp服务器中的**CDH-5.3.9-1.cdh5.3.9.p0.8-trusty.parcel**放到**/opt/cloudera/parcel-repo/**目录下。然后执行以下命令把**CDH-5.3.9-1.cdh5.3.9.p0.8-trusty.parcel**.改名

```
mv CDH-5.3.9-1.cdh5.3.9.p0.8-trusty.parcel.sha1 CDH-5.3.9-1.cdh5  
.3.9.p0.8-trusty.parcel.sha
```

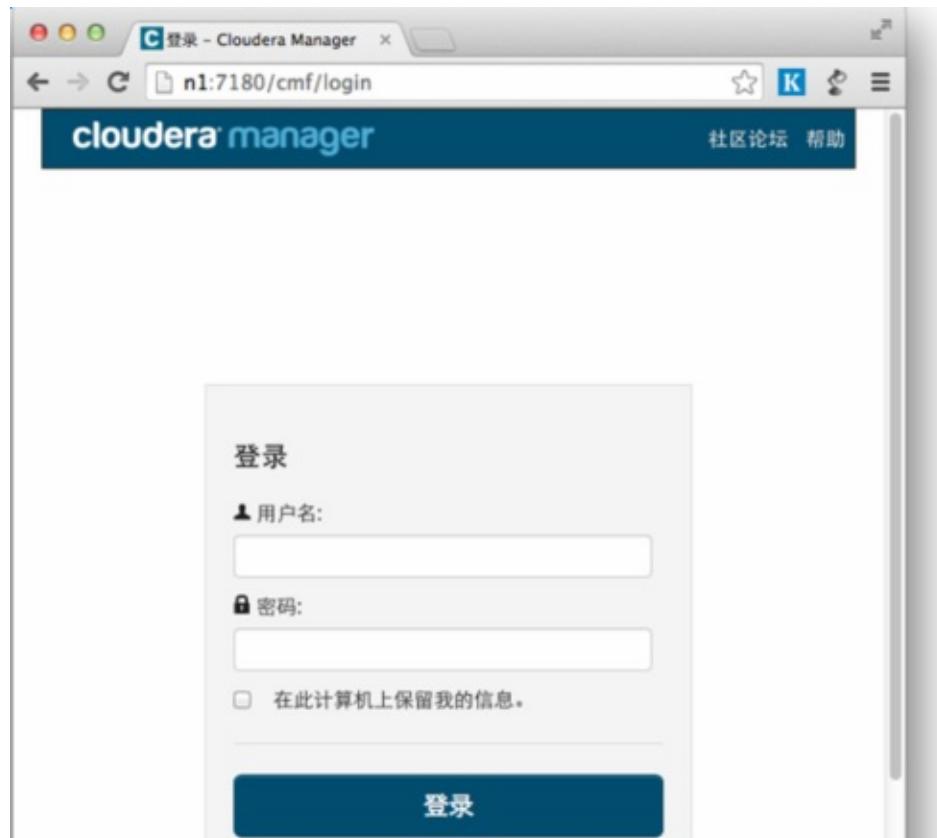
## 相关启动脚本

通过**/opt/cm-5.3.9/etc/init.d/cloudera-scm-server start**启动服务端。

通过**/opt/cm-5.3.9/etc/init.d/cloudera-scm-agent start**启动Agent服务。（注意普通用户安装时，从机上的Agent服务要用root用户启动，不然会有一个inceptor的服务起不了，用sudo也不好使的，主机的要用cdh用户启动，因为之前配置的ssh配的就是cdh的不然又找不到其它主机）

## 1. 第一步

输入<http://192.168.1.190:7180>(ip为自己的主机ip)



## 2. 第二步

## 2. 基于Ubuntu的安装

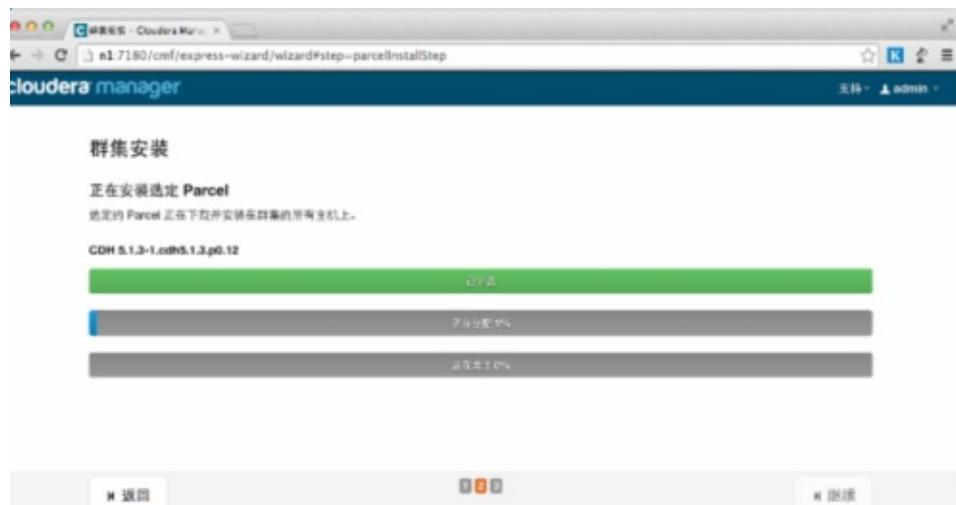
The screenshot shows the Cloudera Manager license selection interface. It compares two editions: Cloudera Express (free, 60-day trial) and Cloudera Enterprise (annual subscription). A red box highlights the '节点限制' (Node Limit) column, which is '无限制' (Unlimited) for Express and '受限' (Limited) for Enterprise. Other columns include '许可证' (License), 'CDH' (checkmark), 'Cloudera Manager 核心功能' (checkmark), 'Cloudera Manager 高级功能' (checkmark), 'Cloudera Navigator 审核组件' (checkmark), and 'Cloudera 支持' (checkmark). A large blue button labeled '继续' (Continue) is at the bottom right.

## 3. 第三步

The screenshot shows the Cloudera Manager host selection interface for a CDH cluster. It lists hosts 'a1' and 'a2' with their IP addresses, rack assignments, and status. A red box highlights the '当前管理的主机 (2)' (Managed hosts (2)) tab. A message at the top says '这些主机不属于任何群集。请选择将组成群集的主机。' (These hosts are not part of any cluster. Please select hosts to form a cluster.) A large blue button labeled '继续' (Continue) is at the bottom right.

## 4. 第四步

## 2. 基于Ubuntu的安装



## 5. 第五步

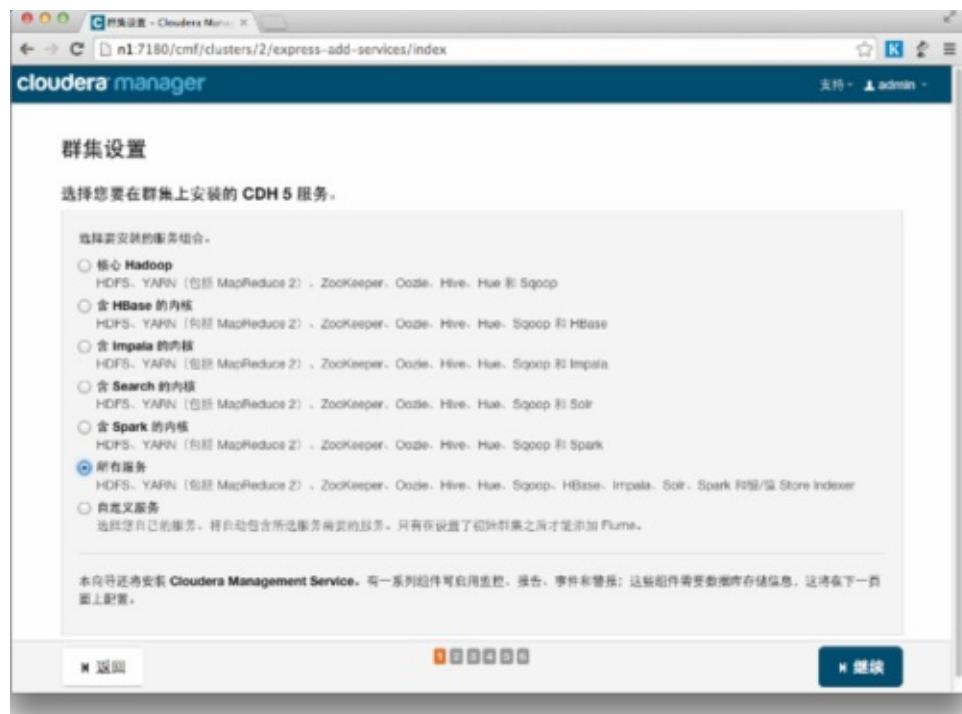
这个是检查主机是的常见错误，我们只需要执行以下命令

```
echo 0 > /proc/sys/vm/swappiness  
#貌似这句只能由root执行
```

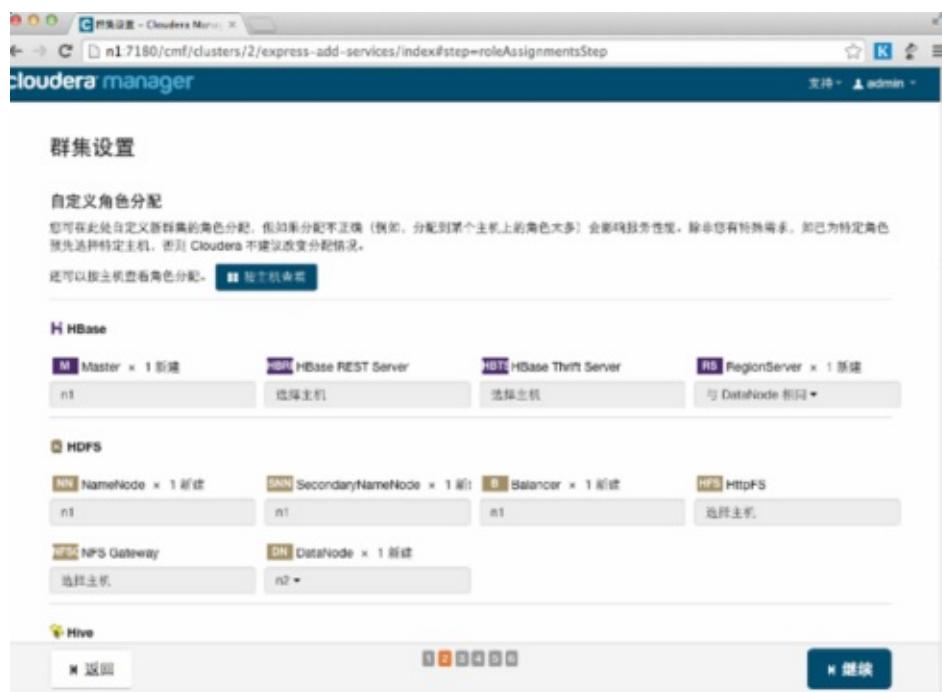


## 6. 第六步

## 2. 基于Ubuntu的安装



## 7. 第七步



## 8. 第八步

## 2. 基于Ubuntu的安装

The screenshot shows the 'Database Settings' section of the Cloudera Manager interface. It displays two successful connection tests: one for 'Activity Monitor' (MySQL database type, host n1, user root) and one for 'Hive' (MySQL database type, host n1, user root). Both tests show a green checkmark and the word 'Successful'. Below the tables, there is a note about specifying hostnames and port numbers correctly. At the bottom right, there is a 'Test All' button.

## 9. 第九步

The screenshot shows the 'Service Configuration' page for the 'CloudBees Management Service'. It lists several configuration items with their current values and descriptions:

参数	组	值	说明
Service Monitor 目录位置	Service Monitor Default Group	/var/lib/cloudera-service-monitor	Service Monitor 监控的存储目录。Service Monitor 会按递增时间序列存储的快照信息，以及 Impala 读写和 YARN 应用提交元数据（如果已配置 Impala 和或 YARN）。
租户存储基础目录	Host Monitor Default Group	/var/lib/cloudera-host-monitor	Host Monitor 监控的存储目录。Host Monitor 会按递增时间序列和运行环境信息。
警报：邮件通知主机组名	Alert Publisher Default Group	localhost	警报警报的邮件发送器的 IP 地址或主机名称。
警报：邮件通知用户名	Alert Publisher Default Group	root@localhost	用于发送邮件警报的用户名。
警报：邮件收件人	Alert Publisher Default Group	root@localhost	用于发送邮件警报的电子邮件地址的逗号分隔列表。

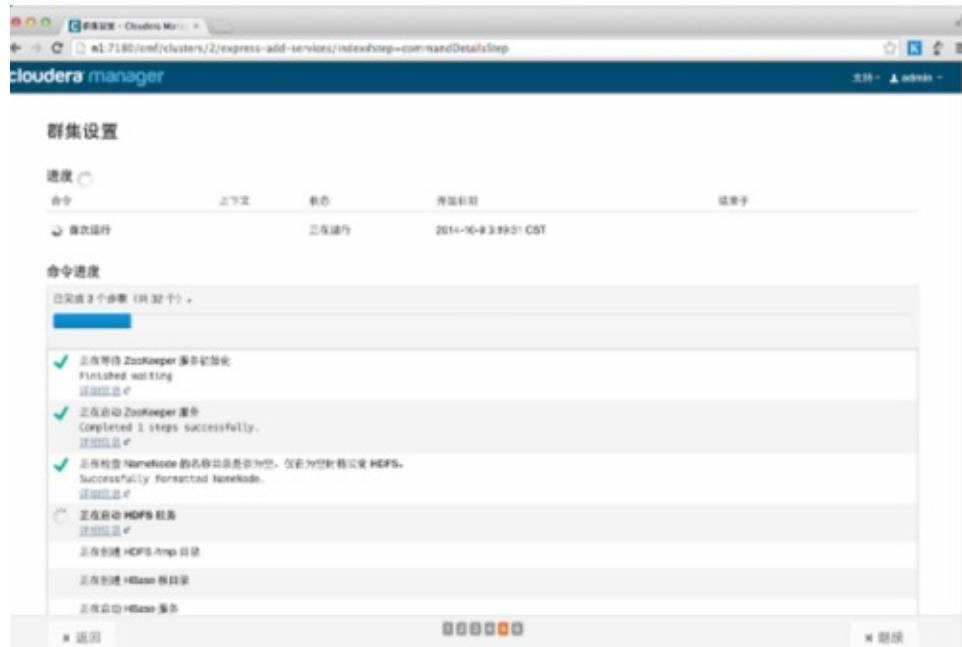
## 10. 第十步

终于到安装各个服务的地方了，注意，这里安装Hive的时候可能会报错，因为我们使用了MySQL作为hive的元数据存储，hive默认没有带mysql的驱动，通过以下命令拷贝一个就行了：

```
cp /opt/cm-5.3.9/share/cmf/lib/mysql-connector-java-5.1.36-bin.jar /opt/cloudera/parcels/CDH-5.3.9-1.cdh5.3.9.p0.8/lib/hive/lib/
```

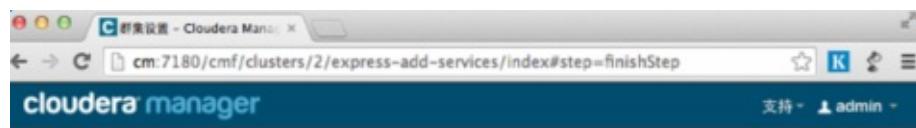
另外hue数据库server启动会失败，查看日志是因为snappy导入失败造成的，这是因为snappy是用Python写成的，我们这里缺少了一个Python-libxslt1包，逐步执行以下命令即可

```
aptitude install python  
aptitude update  
apt-get install Python-libxslt1
```



## 11.第十一步

## 2.基于Ubuntu的安装



### 群集设置

恭喜您！

服务已安装、配置并在群集中运行。

## 12.第十二步

如果hdfs上有感叹好，提示运行不良有不足的块，可在主机上执行：

```
sudo -u oozie bash  
hadoop fs -setrep -R 1 /  
执行完稍等一会就好了。
```

## 第十六章 TDH发行版本

# 第一章：Transwarp Manager的安装

## 安装前准备

- 修改/etc/hosts文件内容

在/etc/hosts文件中添加主机名，添加在最后一行，如192.168.1.200 dhc-1(注意hostname不支持使用'\_,::')，配置完成后可以互相ping下，如果ping不通，请检查/etc/hosts文件和静态IP的设置

- 关闭防火墙

使用chkconfig iptables off关闭防火墙

- 安装目录的创建-非必选

在/mnt目录中创建disk1目录（若配有SSD固态硬盘还需创建randisk目录）

- 时间设定-非必选

设置系统时间为NTP网络时间(如date -s '2016-1-19 9:00:00')

- /etc/sysconfig/network文件修改

```
NETWORKING=yes  
HOSTNAME=dhc-1
```

- ip地址修改

## 1. 安装

```
vi /etc/sysconfig/network-scripts/ifcfg-eth0

#描述网卡对应的设备别名，例如ifcfg-eth0的文件中它为eth0
DEVICE=eth0
#设置网卡获得ip地址的方式，可能的选项为static，dhcp或bootp
BOOTPROTO=static
BROADCAST=192.168.0.255 #对应的子网广播地址
HWADDR=00:07:E9:05:E8:B4 #对应的网卡物理地址
#如果设置网卡获得 ip地址的方式为静态指定，此字段就指定了网卡对应的ip地址
IPADDR=12.168.0.33
NETMASK=255.255.255.0 #网卡对应的网络掩码
NETWORK=192.168.0.0 #网卡对应的网络地址
```

- 网关地址修改

```
vi /etc/sysconfig/network

#表示系统是否使用网络，一般设置为yes。如果设为no，
#则不能使用网络，而且很多系统服务程序将无法启动
NETWORKING=yes
#设置本机的主机名，这里设置的主机名要和/etc/hosts中设置的主机名对应
HOSTNAME=centos
#设置本机连接的网关的IP地址
GATEWAY=192.168.0.1
```

- DNS修改

```
vi /etc/resolv.conf
```

- 网络服务重启

```
service network restart
```

## 安装步骤：

### 一、进入/mnt/disk1目录

## 1. 安装

```
[root@dhc-1 disk1]# pwd  
/mnt/disk1  
[root@dhc-1 disk1]# ls -la  
total 6724372  
drwxr-xr-x. 3 root root 4096 Jan 19 17:50 .  
drwxr-xr-x. 4 root root 4096 Jan 19 18:16 ..  
-rw-r--r--. 1 root root 4467982336 Jan 10 21:02 CentOS-6.5-x86_64-bin-DVD1.iso  
drwxr-xr-x. 6 root root 4096 Nov 11 23:42 transwarp  
-rw-r--r--. 1 root root 2417751281 Jan 12 13:48 transwarp-4.2.2-19029-zh.el6.x86_64.tar.gz  
[root@dhc-1 disk1]#
```

## 二、使用root用户解压其中的transwarp安装包并安装

```
>tar -zxvf transwarp-4.2.2-19029-zh.el6.x86_64.tar.gz  
>cd transwarp  
>./install
```

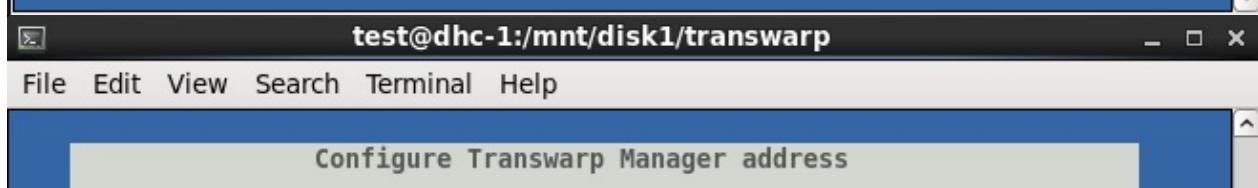
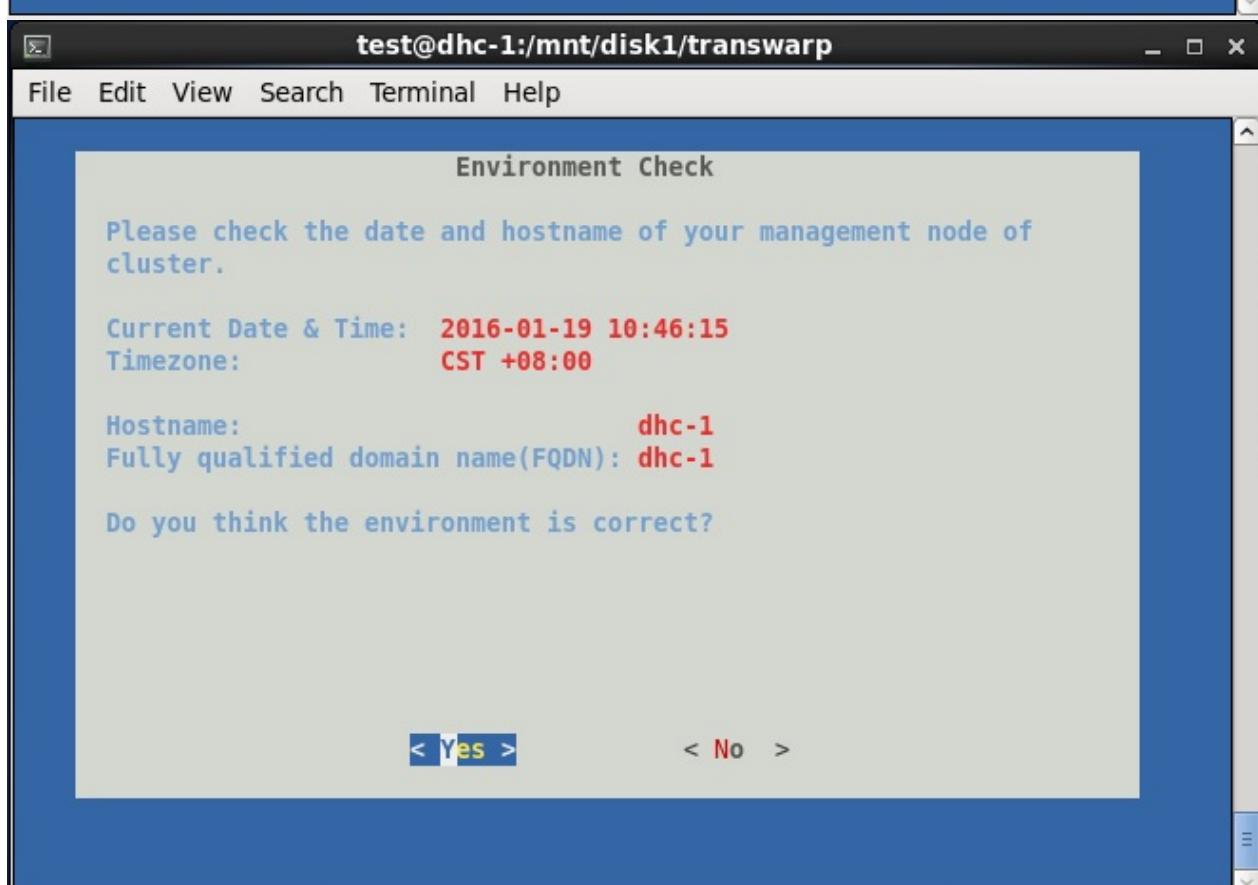
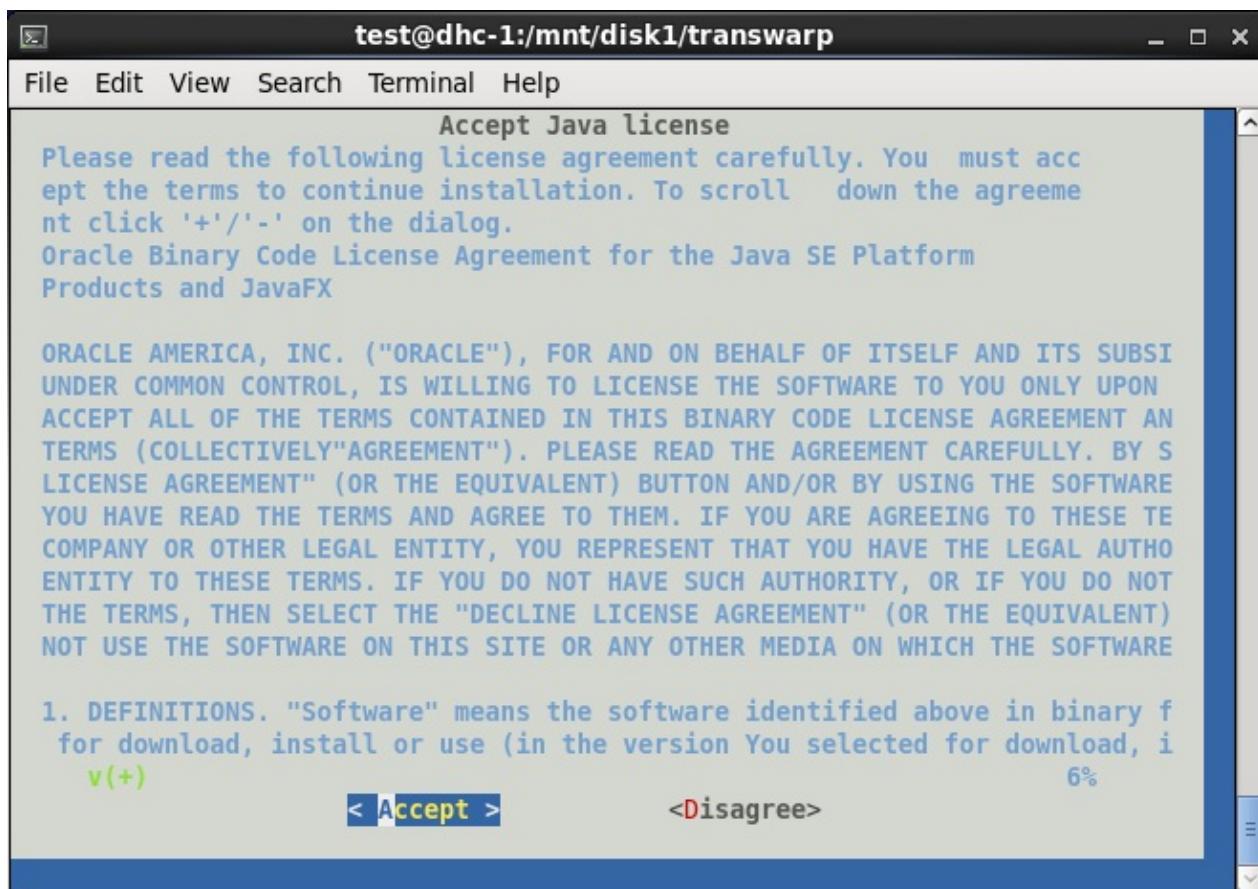
```
[root@dhc-1 disk1]# cd transwarp  
[root@dhc-1 transwarp]# ls -la  
total 684  
drwxr-xr-x. 6 root root 4096 Nov 11 23:42 .  
drwxr-xr-x. 3 root root 4096 Jan 19 17:50 ..  
-rwxr-xr-x. 1 root root 91 Nov 11 23:42 install  
-rw-r--r--. 1 root root 19346 Nov 11 23:42 java_license  
-rwxr-xr-x. 1 root root 488984 Nov 11 23:42 libdialog.so  
drwxr-xr-x. 3 root root 4096 Nov 11 23:42 manager  
-rwxr-xr-x. 1 root root 103 Nov 11 23:42 process-ui  
drwxr-xr-x. 2 root root 4096 Nov 11 23:42 script  
drwxr-xr-x. 3 root root 4096 Nov 11 23:42 support  
drwxr-xr-x. 4 root root 4096 Nov 11 23:42 transwarp  
-rwxr-xr-x. 1 root root 135568 Nov 11 23:42 ui-window  
-rwxr-xr-x. 1 root root 2956 Nov 11 23:42 uninstall.sh  
-rw-r--r--. 1 root root 1529 Nov 11 23:42 update_package_pdsh.sh  
-rw-r--r--. 1 root root 1562 Nov 11 23:42 update_package.sh  
-rw-r--r--. 1 root root 6 Nov 11 23:42 VERSION  
-rw-r--r--. 1 root root 0 Nov 11 23:42 versionlist  
[root@dhc-1 transwarp]# ./install
```

## 三、安装完成后。

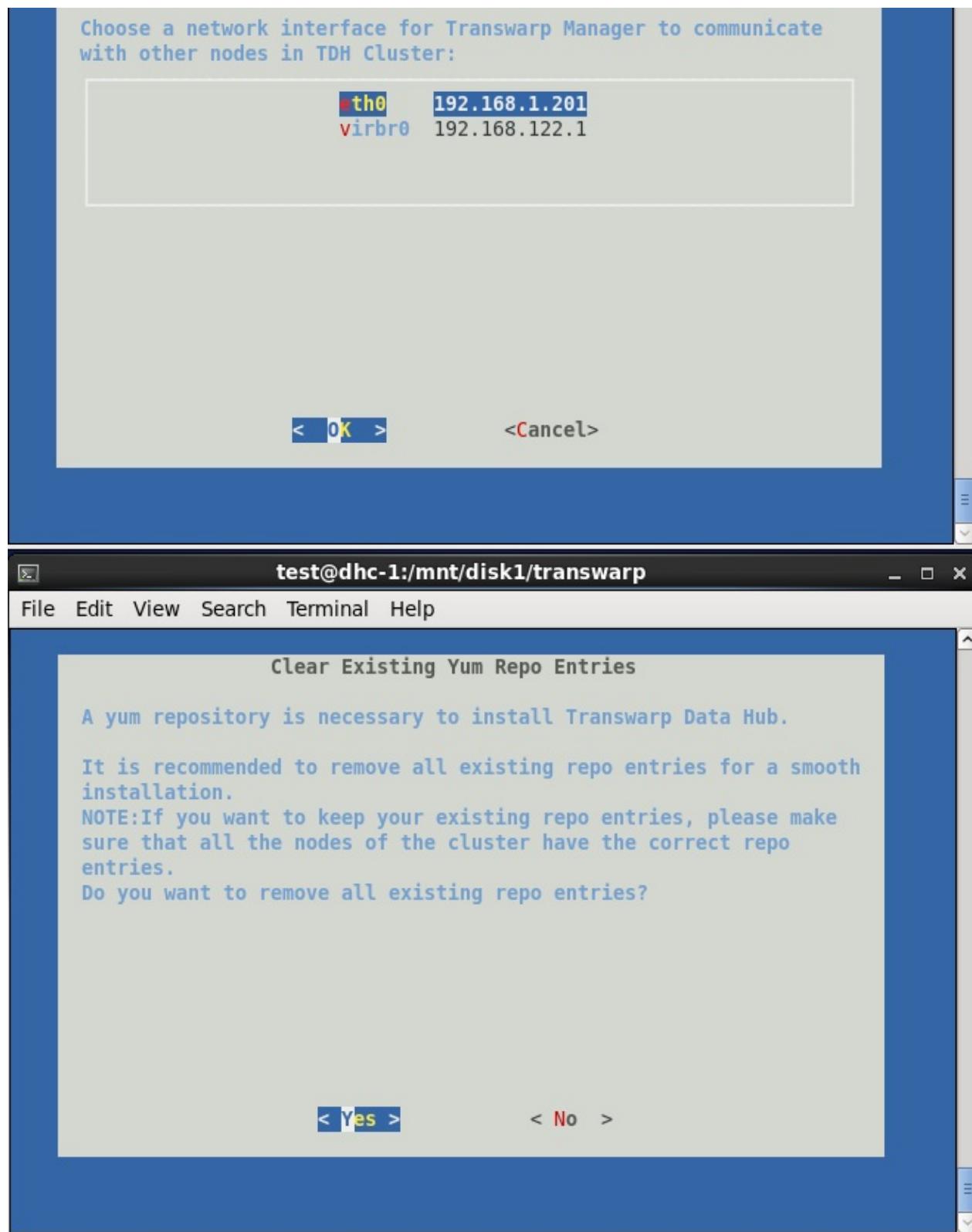
会自动弹出界面，依次选择Accept→选择网卡→默认端口8180→删除已有yum资源库→create new repository→Use ISO File→选择/mnt/disk1中的CentOS6.5安装包



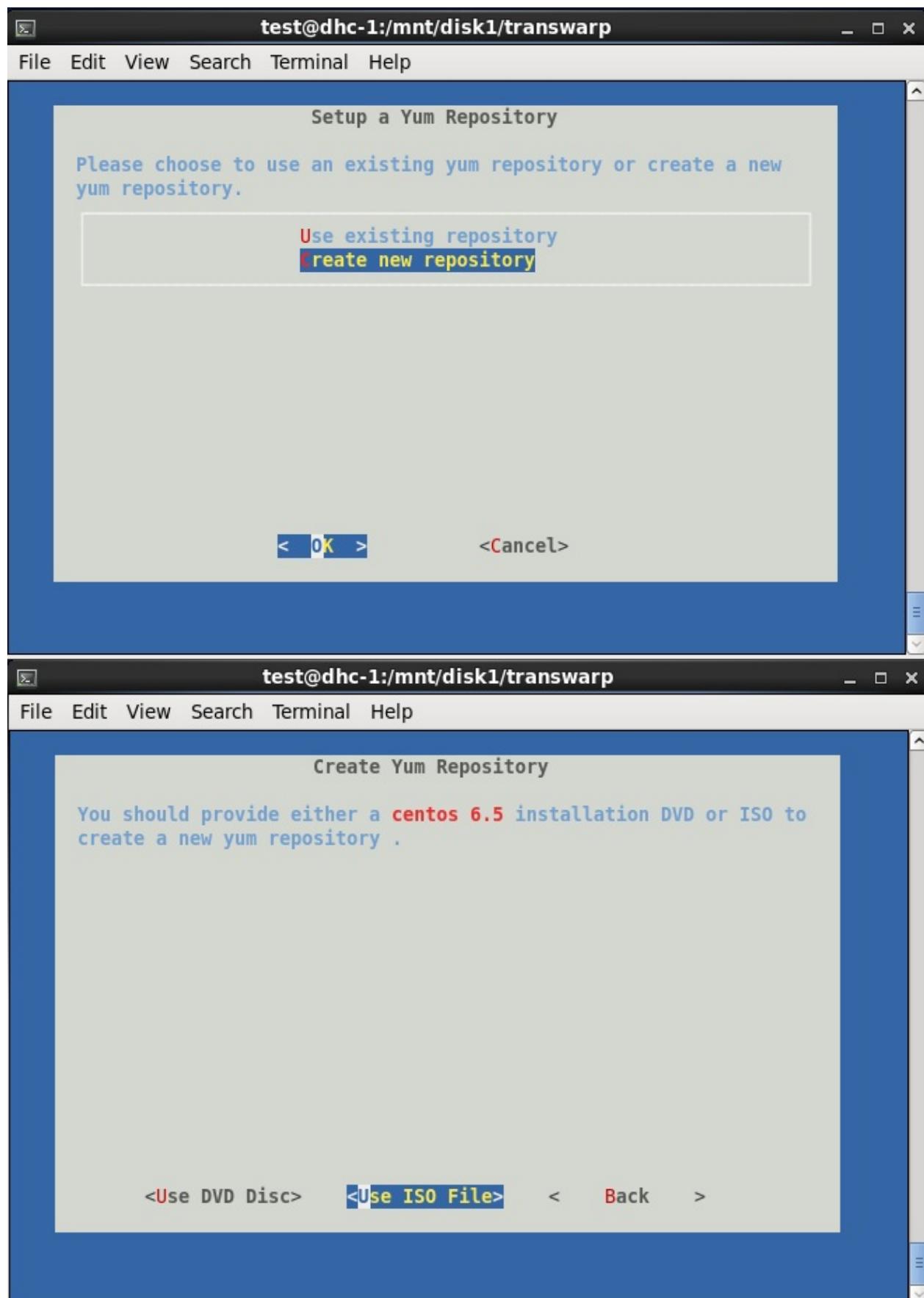
## 1. 安装



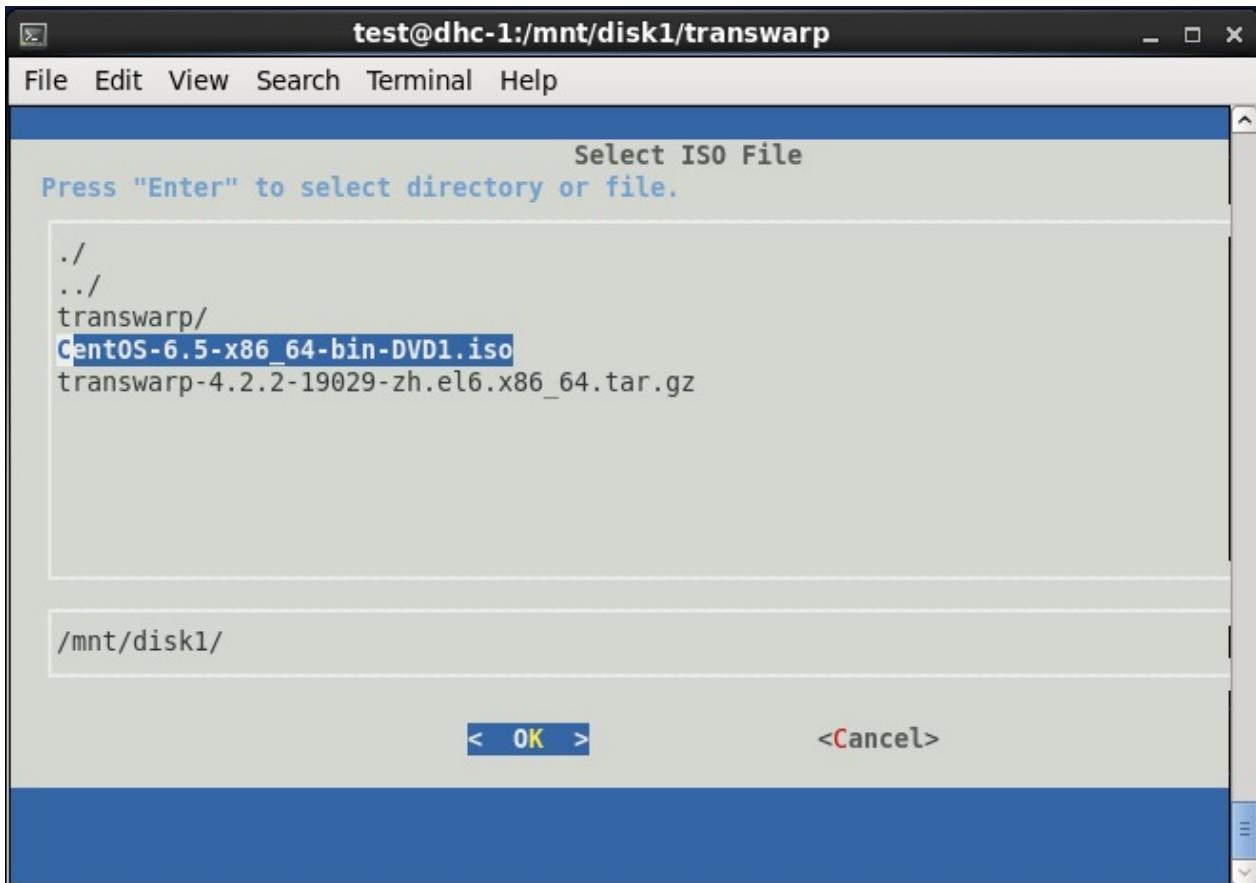
## 1.安装



## 1.安装



## 1. 安装



## 四、安装之后的设置

安装好Centos6.5以后，打开chrome浏览器，输入安装Manager的本地节点ip地址加端口号8180，如192.168.1.200：8180，进行如下步骤操作：

1. 输入admin、admin进入界面,为了方便多人对Mananger的操作，可以新建多个隶属于admin组的账户，同样可以操作服务器集群，避免了登入登出时被别的用户挤出。
2. 填写集群名称（随意取名）
3. 添加机柜（使用/rack1，/rack2.....）指定
4. 添加节点（可以使用〔〕来批量添加，如172.16.2.〔68-70〕）
5. 输入root账号和密码进行确认设定
6. 分配机柜，将刚刚的第一个节点分配到/rack1中，其他两个节点分配到/rack2中
7. 选择需要/etc/hosts来确认网络解析
8. 为了负载均衡，将YARN分配到/rack1中，Inceptor—server分配到/rack2中

## 五、安装组件和服务

## 1. 安装

按照左侧栏提示分别需要安装Zookeeper、HDFS、YARN、Hyperbase、Inceptor-SQL，其他可以暂时不用安装

1. Zookeeper：将全部节点都添加上（一定要为奇数），其他默认
2. HDFS：记住两个重要目录即可，分别为dfs.namenode.name.dir和dfs.datanode.data.dir，分别在/home/hadoop节点下的hdfs\_image和data目录下。另外需要特别注意的是，在安装HDFS过程中可能会遇到formatnamenode失败的现象，查看界面上的操作日志，可以看到报以下这个错误：
  - 在所有的JournalNode上，删除/hadoop/journal中所有的内容，然后执行service hadoop-hdfs-journalnode-hdfs1 restart
  - 在所有NameNode节点上，清空dfs.namenode.name.dir配置的相应目录的所有内容
  - 在所有DataNode节点上，清空dfs.datanode.data.dir配置的相应目录的所有内容
1. YARN：基础参数中配置yarn.nodemanager.resource.cpu-vcores的CPU核数，配置yarn.nodemanager.resource.memory-mb的内存大小，推荐配置为YARN的核数全给，内存给一半

若CPU若不知道分几个核数，可以在命令行中执行

```
cat /proc/cpuinfo | grep processor | wc -l
```

1. HyperBase：配置master.memory内存大小，（若内存大小为8G，那么这里就应该是8G-YARN的yarn.nodemanager.resource.memory-mb内存大小），Mastermemory相当于NN，Region server类似于DN，一般Master memory不耗费内存，主要Region server比较耗费内存Inceptor-SQL(SQL on spark)：高级参数里面可以设置安全护栏，即hive.server.enable，值为FALSE不开启，值为TRUE后面服务就需要安装kerberos认证了，这项看具体实际需求。

另外在资源分配选项中，executor有Fixed（同构机器，每台机器配置差不多）和Ratio（异构机器，每台机器配置相差很大）两种，一般选择Fixed，下面的内核和内存千万不能超过YARN所设置的内核数和内存大小的值，因为Inceptor-SQL是从YARN那里申请资源的！推荐配置为内核数：内存=1:2（1个内核配置2GB），Inceptor server节点和Inceptor metastore节点需要安装在同一节点上，若跨节点对

## 1. 安装

表的操作会延迟会很高，Inceptor metastore存储的是表的信息，记住Inceptor metastore节点的IP地址（即Inceptor server地址）因为使用sqoop服务要在metastore节点上操作mysql数据库（操作之前还需添加mysql的驱动）

### 六、确认安装后，登陆**Inceptor**的命令：

```
beeline -u jdbc:hive2://<Inceptor ip>:10000/  
beeline -u jdbc:hive2://192.168.1.70:10000
```

hive1登陆命令为：transwarp -t -h [Inceptor ip]

### 七、操作数据库

```
>show databases;  
>use database;  
>show tables;  
>create table country(id int, name string);
```

### 安装常见错误汇总：

1. format namenode出错，造成format namenode失败的原因是因为原TDH没有删除干净，在hadoop/namenode-dir/current里面有个锁，删除后可以使用命令etc/init.d/hadoop-hdfs-namenode start来启动namenode节点
2. 如果HBase安装不成功，region server报红，首先cd /usr/lib/zookeeper/bin目录执行zkcli.sh -service [任意一台zookeeper主机ip]，再使用rmr /hyperbase命令将里面的hyperbase目录删除，要注意的是里面的原数据会丢失
3. Inceptor报黄就去YARN中检查

### 常用命令：

1. Manager的相关命令可以使用：service --status-all来查看
2. 显示hdfs集群的命令是：hadoop dfsadmin -report  
或者sudo -u hdfs hdfs dfsadmin -reports

## 1. 安装

---

3. 使HDFS中数据平衡的命令是：`sudo -u hdfs hdfs balancer`

# Inceptor-SQL 使用

## 简介

Inceptor是一种交互式分析引擎，本质是一种SQL翻译器。Inceptor中一共可以操作五种类型的表结构：

1. 普通文本表（**TXT**表）
2. 分区表（分单值分区和范围分区）
3. 分桶表
4. **ORC**表（**Hive ORC**格式）
5. **ORC**事务表（可进行增删改查操作，必须建立分桶表和外表，且两个表的表格式要和源数据字段一一对应起来）

注意：一般来说分区和分桶是结合起来使用的，例如先将一个大表按照时间进行分区，再对每个分区进行分桶。

## 一、普通表导入数据

1. 从HDFS导入数据
  - i. 创建HDFS数据目录，在本地创建一个存放数据的文件夹，为了区分不同用户和不同数据源，建立以下两个目录

```
hadoop fs -mkdir -p /user/user1/data/inceptor
hadoop fs -mkdir -p /user/user1/data/hyperbase
```

1. 首先将本地path存放的数据文件put到HDFS目录中，数据可以存放在集群中的任意一台机器中（注意本步操作可能会报load数据没有权限，HDFS上的数据和表的权限不一致 使用：`(sudo -u hdfs hadoop fs -chown -R hive /user/*)` 命令进行owner的修改，hive为owner名字）或者使用`sudo -u hdfs hadoop fs -chmod -R 777 /user/*`

```
hadoop fs -put <path>/data.txt /user/user1/data/inceptor
```

- 将上传进HDFS的文件load到Inceptor事先建立好的s3表中，在Inceptor中输入如下命令：

```
load data inpath '/user/user1/data/inceptor/data.txt' into table s3;
```

- 从其他表导入

- 将t3的表结构复制给t4，注意不复制数据

```
create table t4 like t3;
```

- 查看

```
select * from t4;
```

- 将t3表中的数据插入到t4表中

```
insert into table t4 select * from t3;
```

## 二、分区表

- 创建单值分区

- 创建单值分区表（每创建一个单值分区表就会产生一个小文件，这里只有一个name值）

```
create table single_tbl(name string) partitioned by(level string);
```

(注意后面的partition分区键和文本是无关的！文本只导入name！分区键是通过load语句中的level具体标识来指定的)

1. 把本地包含单列数据的txt文件put到HDFS中的/user/datadir目录中

```
hadoop fs -put /tmp/a.txt /user/datadir
```

1. 将HDFS中的a.txt文件load到single\_tbl单值分区表，即将a这个文档都设置成A标签

```
load data inpath 'user/datadir/a.txt' single_tbl partition(level = 'A');
```

1. 创建范围分区表（用于避免全表扫描，快速检索，导入数据的方法也很少，只能通过从另一个表插入到范围表中，其产生原因是为了规避单值分区每创建一个表就会产生一个小文件，而范围分区则是每个分区存储一个文件）

- i. 创建范围分区表rangeprt

```
create table rangeprt(name string)partitioned by range(age int)
(
    partition values less than(30),
    partition values less than(40),
    partition values less than(50),
    partition values less than(MAXVALUE)
);
```

(注意分区表为左闭右开区间)

1. 将本地文件或文件夹put到HDFS的user/datadir的目录中

```
hadoop fs -put /tmp/b.txt user/datadir
```

1. 创建外表，来将HDFS中的文件进行导入进来(外表是用来指定导入数据格式的，且drop外表时，HDFS上的数据还存在)

```
create external table userinfo(name string,age int) row format d
elimited fields terminated by ',' location 'user/datadir';
```

- 将外表的数据插入到建立好的rangeprt表中

```
insert into table rangeprt select * from userinfo;
```

- 查看插入分区表里的数据分布

```
show partitions rangeprt;
```

### 三、分桶表

(必须创建外表，只支持从外表导入数据，在分桶表中经常做聚合和join操作，速度非常快。另外分桶规则主要分为1、int型，按照数值取模，分几个桶就模几2、string型，按照hash表来分桶)

- 创建分桶表bucket\_tbl(这里分桶的大小是用表的总数据大小除以200M，经实际优化测试，每个桶的数据为200M处理速度最优)

```
create table bucket_tbl(id int, name string) clustered by (id) into 3 buckets;
```

- 创建外表bucket\_info,bucket\_info表会自动将HDFS目录/user/datadir中的数据自动load进表里，这和普通表需要手动进行load不一样

```
create external table bucket_info(id int, name string) row format  
delimited fields terminated by ',' location '/user/datadir';
```

- 将从本地txt文件put到HDFS中的表（如普通表），再load进外表中

```
load data inpath '/user/tdh/data/bucket-data' into table bucket_info;
```

- 设置分桶开关

```
set hive.enforce.bucketing=true;
```

### 1. 插入数据（按照取模大小顺序排列）

```
insert into table bucket_tbl select *from bucket_info;
```

## 四、holodesk表

(holodesk表既可以基于内存也可以基于ssd存储和查询，holodesk会存两份，一份存在内存或者ssd中，一份存在HDFS中，这样可能在查询的性能上有所延迟)  
holodesk最擅长处理groupby的SQL查询语句

说明：建立holodesk表之前最好先建立cube，cube一般为3-5列，表很小，在Inceptor中建立cube内表，取的速度很快，遍历会很快，cube不能将所有的数据都放入内存，所以建内表时，将部分需要的数据放在内存中，因为cube只有3-4列，大大简化了原ssd中的大数据集，查询速度会很快，所以说一般holodesk是和cube配合使用的。内存表创建有两种方式：第一种通过CTAS建表，建表时数据即填入，这种情况下，内存表不能分区或者分桶。第二种通过创建空表，此时内存表可以分区或者分桶，之后可以通过Insert into select插入数据。

### (1) 通过CTAS (create table...as select) 建表

```
create table table_name tblproperties("cache"="cache_medium", "cache.checkpoint"="true|false", ["filters"="filter_type"]) as select id, name, sex, date from user_info;
```

### (2) 通过创建空表建表，再插入数据

```
CREATE TABLE table_name (column_name data_type, column_name data_type...) PARTITIONED BY (partition_key data_type, partition_key data_type) CLUSTERED BY (cluster_key, cluster_key, ...) INTO n BUCKETS TBLPROPERTIES ("cache" = "cache_medium", "cache.checkpoint"="true|false", ["filters"="filter_type"])
```

说明：

"cache"="cache\_medium"指定计算缓存的介质。可以选择ram,SSD和memeory三种。只有当服务器上配置有SSD时,才可以选择SSD作为缓存,Inceptor会自动利用SSD为计算加速。"

cache.checkpoint"="true|false"指定是否设置checkpoint。如果设置checkpoint,查询结果会被同步放入HDFS中,在存储了内存表的机器当机时,内存表中的数据可以从HDFS中直接读取恢复而不需要重新进行查询计算。 "filters"="filter\_type"为可选项,它指定一个过滤器。利用过滤器可以为某些查询进行优化

## 五、建立**ORC**格式表，如下三种方式

(1)

```
create table country (id int, country string) stored as orc;
```

(2)

```
create external table ex_tbl(id int, country string)
  row format delimited fields terminated by ','
  stored as textfile
  location '/user/tdh/externaltbl';
```

(3)

```
insert into country select * from ex_tbl;
```

## 六、建立**ORC**格式事务表（必须要分桶，既可以单值插入，又可以通过外表插入）

(1)

```
create table orc_tbl(id int, country string) clustered by (id)
  into 3 buckets stored as orc
  tblproperties("transactional" = "true");
```

(2) (创建外表需要注意的是，一定要指定分隔符，不然当external表自动加载HDFS中的/user/datadir时不知道以什么分隔数据，造成查询出的数据全部都是null值)

```
create external table external_tbl(id int, country string) row format delimited fields terminated by ',' location '/user/datadir' ;
```

(3) 设置分桶开关

```
set hive.enforce.bucketing=true;
```

(4)

```
insert into orc_tbl select * from external_tbl;
```

(注意：ORC只是一种表的格式类型，建表时指定了transactional" = "true"，则表明这是一个事务表，必须要分桶，若没有指定则只是普通的ORC表，不需要进行分桶操作)

### 注意事项：

- HDFS不能直接load到Inceptor中的ORC事务表中，(只能load到普通表和ORC表中)要想在ORC事务表里插入数据有两种方法：a.建立一张外表，再将HDFS load进外表上，再insert into select \* from external table b.由于ORC事务表支持增删改查，也可以使用单值插入语句insert into table country values(101,japan)
- 查看分区表的命令是show partitions [table名]
- 查看每个表的创建时语句命令是show create table [table名]
- 使用命令hdfs dfs -ls /user/country (或者使用hadoop fs -ls /user/country命令)

- 默认数据库存放位置 hdfs : //nameservice/inceptorsql1/user/hive/warehouse/ 在Inceptor创建数据库时一般使用它的default默认数据库，若自己建立数据库请不要指定location，还有自己建立的数据库可能会因为权限不够而造成一些操作失败报错。可以使用hadoop fs -ls /inceptorsql1/user/hive/warehouse查看默认目录下存储的数据，eg：

(1)

```
create database ccc location '/user/ccc';  
  
create table ccc1;
```

上述语句建立的数据库位置为user/ccc/hive/ccc1

(2)

```
create table ccc2(a int) location 'user/ccc2';  
上述语句建立表的位置在user/ccc2
```

- 外表的作用是load导数据使用的，起到的是媒介作用，而ORC表则是做具体的操作的，外表一般是和ORC事务表配合使用的
- 分区表中的单值插入数据必须指定level
- 分桶中的桶大小，即一个文件大小一般为200M，处理效率最优，拿总文件大小除以200M就大概预估出分几个桶了
- 从HDFS中向Mysql中导入数据规定必须先在Mysql中创建临时表，先从HDFS的location目录下导入到tmp表中，再从tmp表导入到Mysql真正的表中
- Flume需要先使用yum install flume命令安装，Flume的默认存放位置为/usr/lib/flume/conf/flume.conf，vi进去后进行相应的修改，有两个位置需要注意，第一个是spoolDir后跟log所在HDFS中的文件夹名！切记，不是跟具体的log文件或者txt文件！（如：spoolDir=/tmp/flume/），第二个是path后面是Active NameNode的HDFS路径（如：  
path=hdfs://172.16.2.77:8020/user/datadir），在flume.conf配置中默认指定缓冲区积攒到1k就写入HDFS中
- 养成在Inceptor中使用命令desc formatted ;来查看各个表的底层结构和属性

- **hadoop fs**：命令使用面最广，可以操作任何文件系统。

**hdfs dfs**：命令只能操作HDFS文件系统相关。

## 附录（示例代码）

```
--登录Inceptor server节点
beeline -u jdbc:hive2://172.16.2.75:10000/

--DDL
--创建数据库(location 中文件的权限需和数据库owner一致)
--没有指定location，则数据库放在默认位置
--额外属性
create database db4
comment 'this is a database'
location '/user/test/'
with dbproperties('owner' = 'transwarp','time'='2016');

--在默认位置创建数据库，使用desc 查看数据库
create database db2;

--删除数据库(数据库需为空)
drop database dbname

--描述数据库
describe database dbname;

--修改数据库
alter database db1 set dbproperties('owner'='hehe');

-----创建简单text表-----
--
--创建普通text表
create table t1(col1 string, col2 int);
create table t3(col1 int, col2 string)row format delimited field
s terminated by ',' location '/user/datadir';
create table t2 like t1;
create table t4 as select col1 from t3;
```

```
--导入数据
--从hdfs导入数据(注意owner)
load data inpath '/user/datadir/data' into table t3;

--从其他表中插入
insert into table t4 select * from t3;

--创建分区表
--单值分区
create table part_t6(name string) partitioned by (level string);
--分区表导入数据，一次导入一个分区。分区键是从sql语句中定义，并不是从文件中
导入。
--在本句中，如果文件有两列数据，则第二列是无效的。
load data inpath '/user/datadir/part-data1' into table part_t6 p
artition(level='A')//level变成A

--创建范围分区表(不支持从文件导入分区表，支持insert into select from)
create table rangepart_t7 (name string)partitioned by range(age
int)(
partition values less than(30),
partition values less than(40),
partition values less than(50),
partition values less than(MAXVALUE)
);
--普通数据表
create external table user_info(name string, age int)row format
delimited fields terminated by ',' location '/user/datadir';
load data inpath '/user/tdh/data/range-part1' into table user_in
fo;
--通过普通数据表向范围分区表插入数据
insert into table rangepart_t7 select * from user_info;
select * from range_part_t6;
--查看分区信息
show partitions rangepart_t7;
--查看底层文件
```

```

-----
-- 分桶表创建(数据只能通过insert插入，load无效)
-- 创建分桶表
-- 在windows下创建的文本文件是一\r\n换行，在unix下以\n换行，
-- 导入数据应该是unix格式的。
--- 以数字为bucket id
drop table if exists bucket_tbl;
create table bucket_tbl(id int, name string)clustered by (id) in
to 3 buckets;
-- 创建普通数据表
drop table if exists bucket_info;
create external table bucket_info(id int, name string)row format
delimited fields terminated by ',' location '/user/datadir';
-- 导入数据到数据表中
load data inpath '/user/tdh/data/bucket-data' into table bucket_
info;
-- 插入数据到分桶表中
set hive.enforce.bucketing = true;
insert into table bucket_tbl select *from bucket_info;

--- 以字符串为bucket id,
drop table if exists bucket_tbl;
create table bucket_tbl(id int, name string)clustered by (name)
into 3 buckets;
-- 创建普通数据表
drop table if exists bucket_info;
create external table bucket_info(id int, name string)row format
delimited fields terminated by ',' location '/user/datadir';
-- 导入数据到数据表中
load data inpath '/user/tdh/data/bucket-data' into table bucket_
info;
-- 插入数据到分桶表中
set hive.enforce.bucketing = true;
insert into table bucket_tbl select *from bucket_info;

---- 创建外表

```

```
drop table if exists ;
create external table ex_tbl(id int,country string)
row format delimited fields terminated by ','
stored as textfile
location '/user/tdh/externaltbl';
-----
-----
----建立ORC表
drop table if exists country;
create table country (id int, country string)stored as orc;

----建立ORC事务表
drop table if exists country;
create table country(id int, country string) clustered by (id)into 3 buckets stored as orc tblproperties("transactional" = "true");
insert into table country select * from ex_tbl;
insert into table country values(100,'japan');
insert into table country values(101,'isis');

--创建分区分桶ORC表，分区字段不能用date类型，
drop table if exists country;
create table country (id int, country string) partitioned by(level string) clustered by (id)into 3 buckets stored as orc tblproperties("transactional" = "true");
--从外表插入数据
insert into country partition (level='A') select * from ex_tbl where id<5;
--单条插入
insert into table country partition (level='C') values(101,'isis');
```

## 3. 使用JDBC、ODBC工具连接Inceptor

### 一、squirrel工具

介绍：是一个用Java编写的开源数据库工具，可以用来查看/编辑数据库的内容、发出SQL命令。它可以支持兼容JDBC的数据库，可以使用统一的界面处理不同的数据库，本例为使用小松鼠连接Inceptor

安装步骤：

1、确认windows系统的电脑上安装了jdk1.7或以上

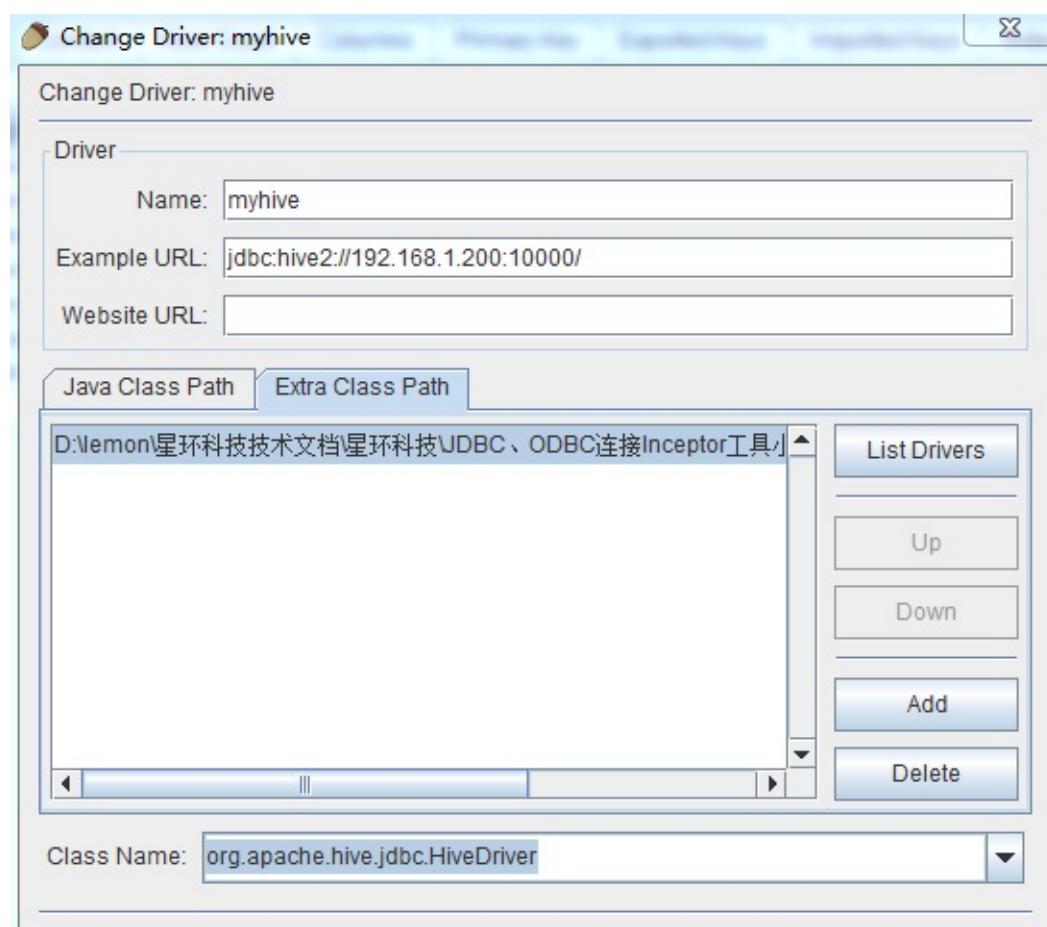
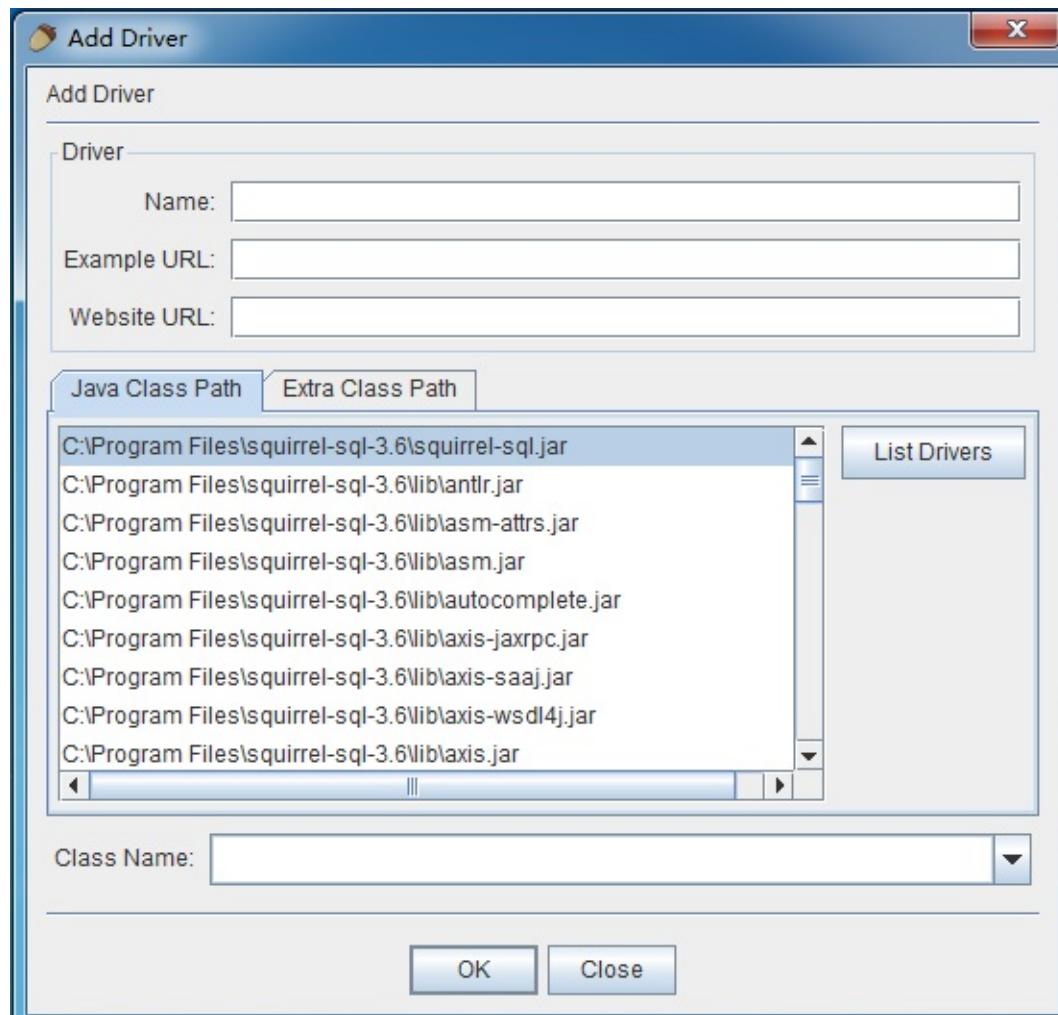
2、打开CMD，输入

```
java -jar squirrel-sql-3.6-standard.jar
```

(待安装好后进入windows安装目录下运行squirrel-sql.bat)

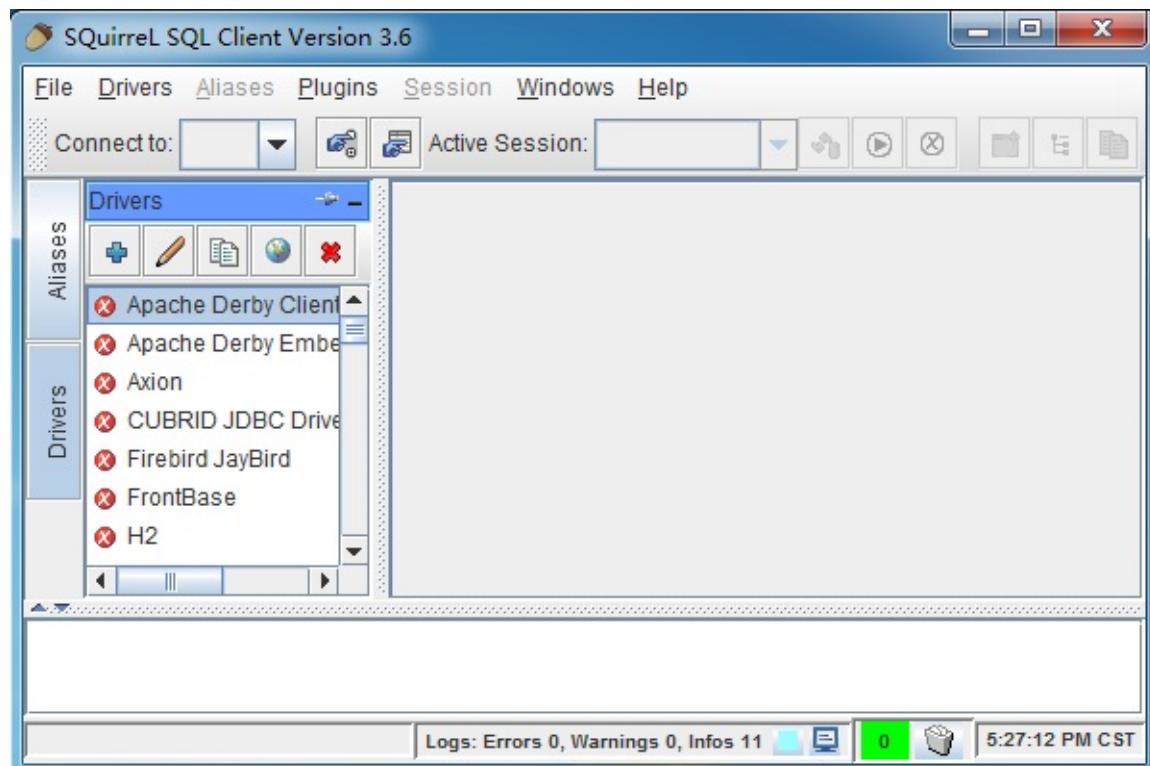
3、连接数据库需要填写驱动信息，Name：myhive（随意起名） URL：  
jdbc:hive2://[Inceptor server ip]:10000/ (/后面不写数据库名代表默认的default数据库)最后在Extra Class Path里添加外部JDBC的jar包，将inceptor-driver-4.2.2.jar添加进去，点击List Drivers，下面的Class Name就会出现默认的org.apache.hive.jdbc.HiveDriver，选中即可进行下一步

### 3. 使用 JDBC、ODBC 工具连接 Inceptor





#### 4、完成安装，添加别名，进行查询



## 二、Tableau工具

#### 1、在官网上下载Tableau软件



### 3. 使用JDBC、ODBC工具连接Inceptor

安装完成后需要先注册，再使用datestopper工具进行破解，具体教程请参考

<http://jingyan.baidu.com/article/3ea51489fbbca152e61bbad2.html>

## 2、安装ODBC驱动程序

根据自己的机器安装64位/32位的Hive2驱动包

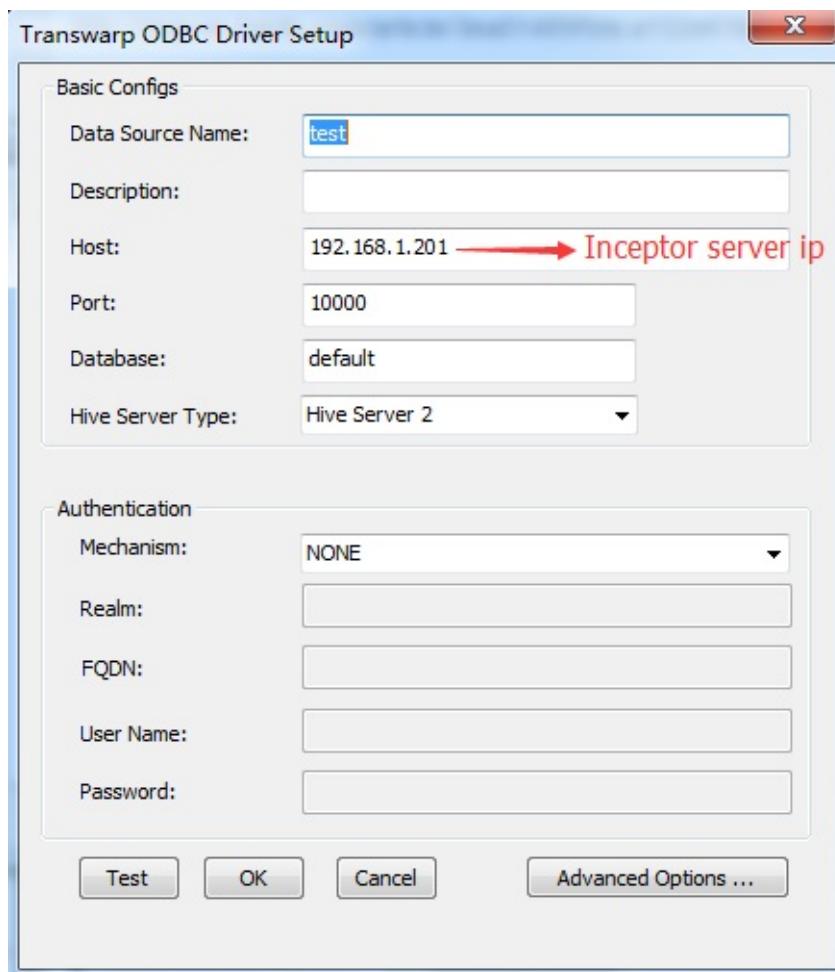
	TranswarpODBCDriver_Hive2_x64.exe	2016/2/24 15:38	应用程序	3,968 KB
	TranswarpODBCDriver_Hive2_x86.exe	2016/2/24 15:38	应用程序	3,318 KB

## 3、添加ODBC驱动程序

在CMD中输入odbc，单击添加，找到刚刚安装好的驱动并填写连接参数



### 3. 使用JDBC、ODBC工具连接Inceptor



点击Test，测试成功后点击OK即可

## 4 Sqoop的使用

### sqoop操作步骤：

1. 在Inceptor metastore节点服务器上安装sqoop服务 yum install sqoop
2. 由于Inceptor-SQL中metastore中已经安装了mysql，就不需要安装mysql了
3. 将mysql-connector-java-5.1.38tar.gz驱动包先解压

```
tar -zxvf mysql-connector-java-5.1.38tar.gz
```

1. cd进刚刚解压后的目录，将里面的mysql-connector-java-5.1.38-bin.jar包copy到/usr/lib/sqoop/lib本地目录下
2. 在Inceptor Server节点上输入mysql -u [用户名] -p连接数据库，此时提示输入密码，通过后输入mysql命令，再执行Grant操作：  
----add user to mysql (username=tdh, password=123456)，授权可以访问所有数据库

```
GRANT ALL PRIVILEGES ON *.* TO 'tdh'@'%' IDENTIFIED BY '123456'  
WITH GRANT OPTION;
```

(若仅授权db1数据库里所有表，则可以这样指定)：

```
GRANT ALL PRIVILEGES ON db1.* TO 'tdh'@'%' IDENTIFIED BY '123456'  
' WITH GRANT OPTION;)
```

1. 浏览mysql数据库

```
sqoop list-databases \  
--username tdh \  
--password 123456 \  
--connect jdbc:mysql://<mysql IP>:3306/
```

### 1. 浏览SQL SERVER数据库

```
浏览SQL SERVER
sqoop list-databases --connect "jdbc:sqlserver://192.168.1.139:1433;username=sa;password=123456"
```

### 1. 浏览mysql数据库中的表，db1为mysql中的一个数据库，可以使用describe [table名]命令查看mysql中表的信息

```
sqoop list-tables \
--username tdh \
--password 123456 \
--connect jdbc:mysql://<mysql IP>:3306/db1
```

### 1. 浏览SQL SERVER中的表并且导入到HDFS中

```
sqoop list-databases --connect "jdbc:sqlserver://192.168.1.139:1433;username=sa;password=123456"

sqoop list-tables --connect "jdbc:sqlserver://192.168.1.139:1433;username=sa;password=123456;database=pubs"

sqoop import --connect "jdbc:sqlserver://192.168.1.139:1433;username=sa;password=123456;database=pubs" --table sales --target-dir /user/lm/sqoop/data/sales -m 1
```

### 1. 从mysql——>HDFS上（import，将mysql中的db1数据库里面的表导入到/user/datadir，这里的datadir目录一定不要事先创建，不然会报错，语句执行的时候会自动创建目录的！最后一行的-m表示map成4个文件）

```
sqoop import \
--username tdh \
--password 123456 \
--connect jdbc:mysql://<mysql IP>:3306/db1 \
--table country \
--target-dir /user/user1/data/sqoop -m 4
```

### 1. 从HDFS——>mysql表上 (export)

```
sqoop export \
--username tdh \
--password 123456 \
--connect jdbc:mysql://<mysql IP>:3306/db1 \
--table cc \
--export-dir /user/testdir \
--staging-table tmptable
```

### 1. 查看sqoop导入进来的文件

```
hadoop fs -cat /user/user1/data/inceptor/part-m-00000 | more
```

## 第1件重要的事：



在SQOOP抽取数据的过程中，强烈建议用以下范本操作，能避免几乎所有多数的错误。

下面四个选项一定要设置

```
--fields-terminated-by "\01" \
--hive-drop-import-delims \
--null-string '\\N' \
--null-non-string '\\N'

--fields-terminated-by "\01" ---> 设置分割符号 ^A
--hive-drop-import-delims ---> 把所有列里面的 \n, ^A等Hive占用符 转成 ""
--null-string '\\N'       ---> 把所有的 String类型的空值 转换成 hive的NULL值 "\N"
--null-non-string '\\N'   ---> 把非String类型的空值 转换成 hive的NULL 值 "\N"
```

SQOOP在这种配置下，第一步会把所有字段中的特殊字符干掉，尤其是 ^A 和\n，同时下一步设定列分隔符为 ^A，充分保证了列分割的准确性。 ---> 这是一个绝不会犯错的选择

## 第2件重要的事：



1. SQOOP 不要直接将RDB里面的数据直接导入到 Hive中，正确的做法是先导入到HDFS 然后再建立外表
2. SQOOP 中，尽量按照我上一封邮件发出的模板操作，用--query的形式导数据，有几个好处：
  - a. 很多时候RDB表太大，需要用where条件分区域导数，非--query的方式几乎不可用。
  - b. SQOOP里面select 的字段可以 直接用到后期的Hyperbase 外表和 Txt 外表。
  - c. 你永远知道自己取出了哪几个字段，而不管RDB的表是否变化过。

### 合理划分SQOOP 任务：按分区来抽取数据

对于大数据Oracle表的SQOOP导入，可以按Oracle分区导入，一个分区一个线程（MAP）。这样会带来两个好处，效率超高，流量可控。  
这次在交管局用到的SQOOP脚本见附件（他们是一天一个分区，90天记录90个分区，约10亿条数据300G大小。单个线程速度在5~6MB/s,性能可线性累加）

```
--query "select $QUERY from $DB2_DATABASE.$TABLENAME partition ($Partition) where \$CONDITIONS" \
```

3/24/2016

www.transwarp.io

3

## 注意事项

- 在执行导入导出数据时，可能由于yarn资源不足或者其他进程的占用，而一直停留在job作业等待处理中，此时可以通过浏览器进入YARN中Resource Manager节点中的8088端口查看被占用的Application ID号，里面描述为Application master为常驻进程，不用杀掉，再在shell中输入命令

```
yarn -application -kill <Application ID>
```

来杀死卡掉的进程，再运行上面的import、export语句。

原因很简单，Inceptor-sql的常驻进程ApplicationMaster跑的是spark任务，非常消耗内存使用量，约为7-8G，所以在没有用到Inceptor-SQL的操作场景的时候就应该关闭该服务。

## 5 使用JDBC、ODBC工具连接Inceptor

### 一、squirrel工具

介绍：是一个用Java编写的开源数据库工具，可以用来查看/编辑数据库的内容、发出SQL命令。它可以支持兼容JDBC的数据库，可以使用统一的界面处理不同的数据库，本例为使用小松鼠连接Inceptor

安装步骤：

1、确认windows系统的电脑上安装了jdk1.7或以上

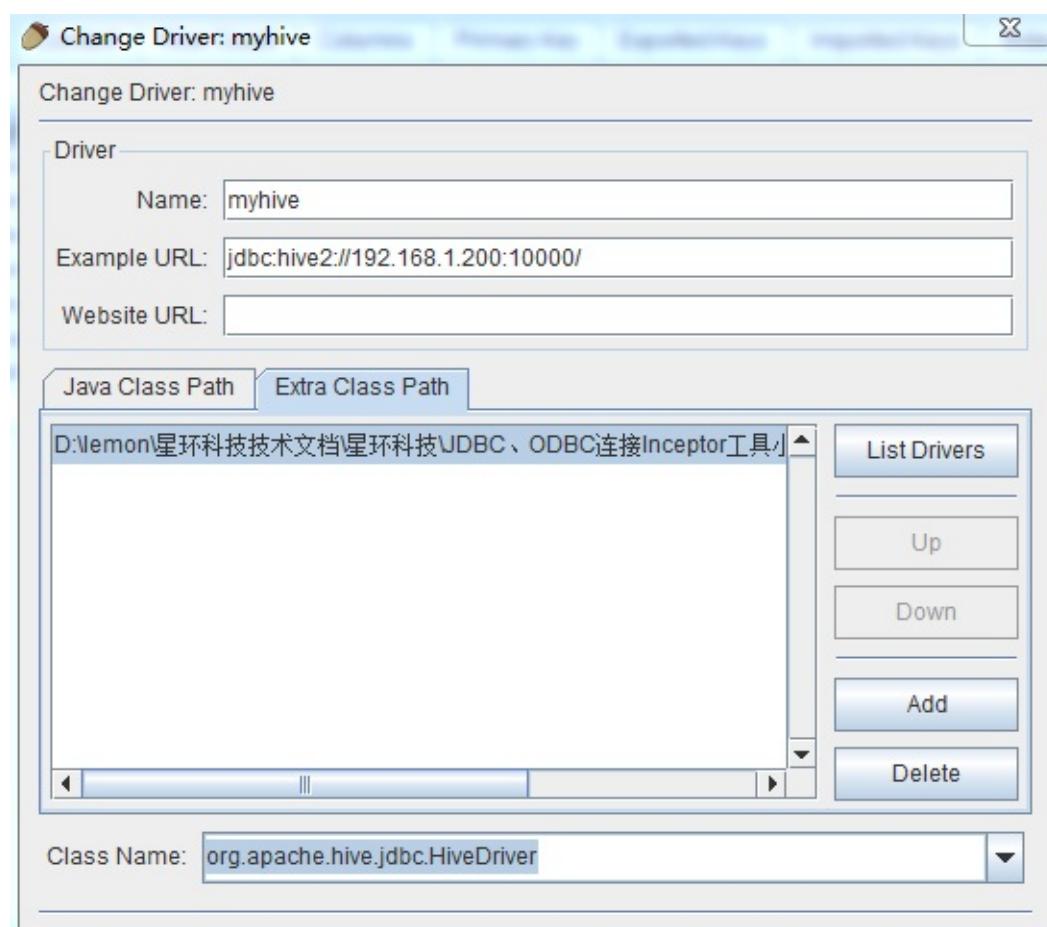
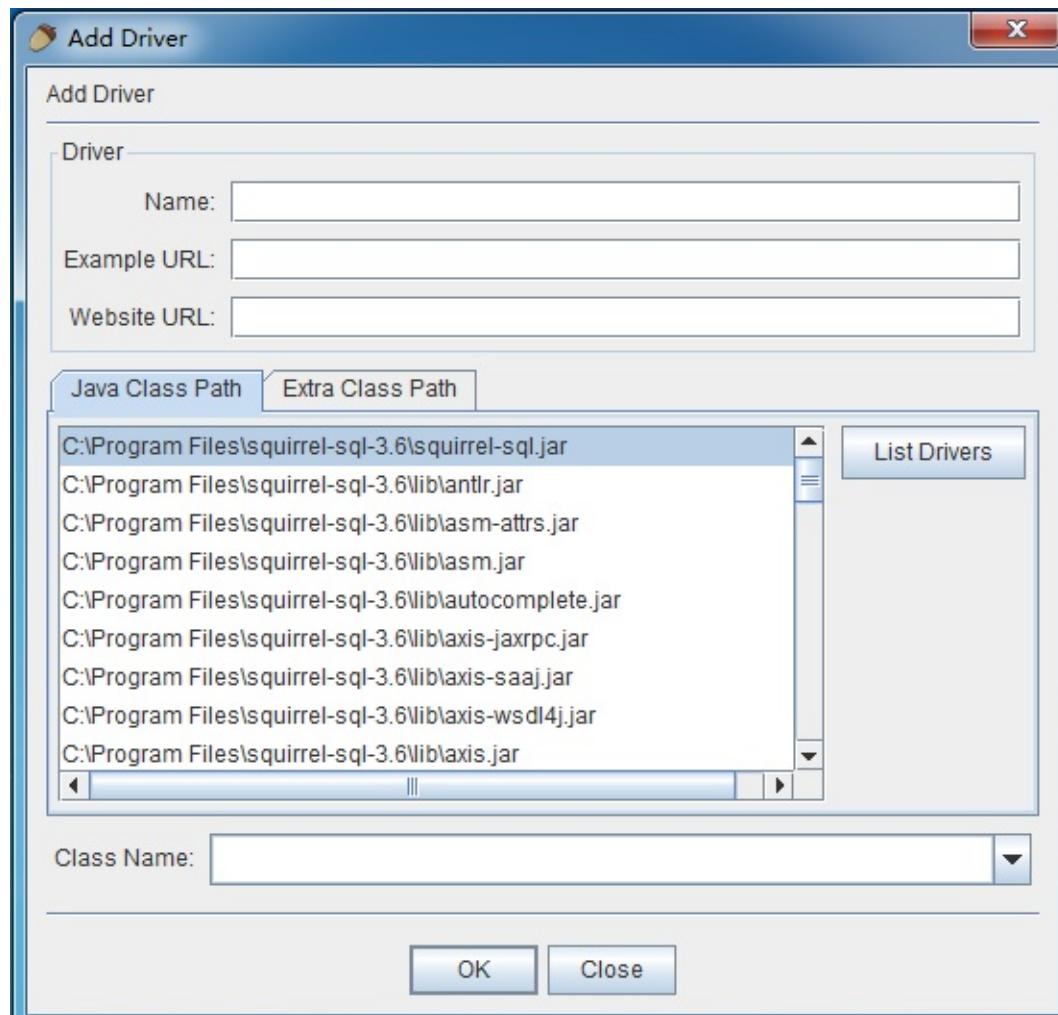
2、打开CMD，输入

```
java -jar squirrel-sql-3.6-standard.jar
```

(待安装好后进入windows安装目录下运行squirrel-sql.bat)

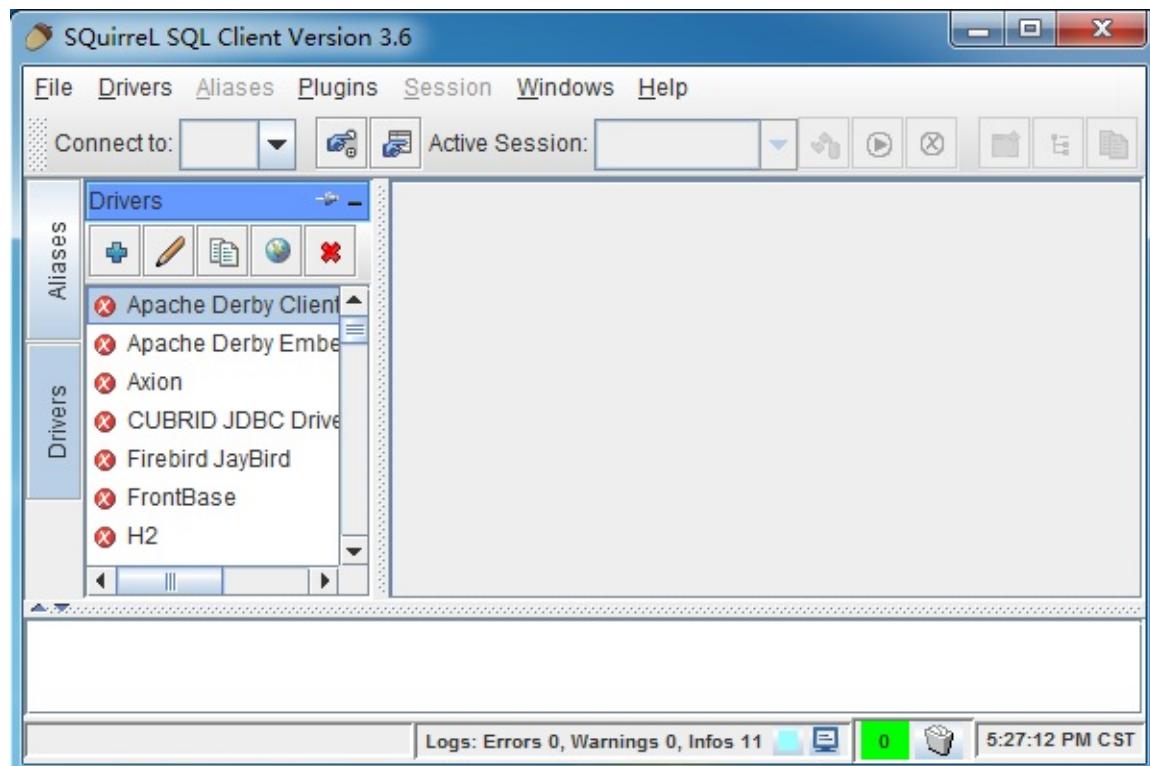
3、连接数据库需要填写驱动信息，Name：myhive（随意起名） URL：  
jdbc:hive2://[Inceptor server ip]:10000/ (/后面不写数据库名代表默认的default数据库)最后在Extra Class Path里添加外部JDBC的jar包，将inceptor-driver-4.2.2.jar添加进去，点击List Drivers，下面的Class Name就会出现默认的org.apache.hive.jdbc.HiveDriver，选中即可进行下一步

## 5 使用JDBC、ODBC工具连接Inceptor





### 4、完成安装，添加别名，进行查询



## 二、Tableau工具

### 1、在官网上下载Tableau软件



安装完成后需要先注册，再使用datestopper工具进行破解，具体教程请参考

<http://jingyan.baidu.com/article/3ea51489fbbca152e61bbad2.html>

### 2、安装ODBC驱动程序

根据自己的机器安装64位/32位的Hive2驱动包

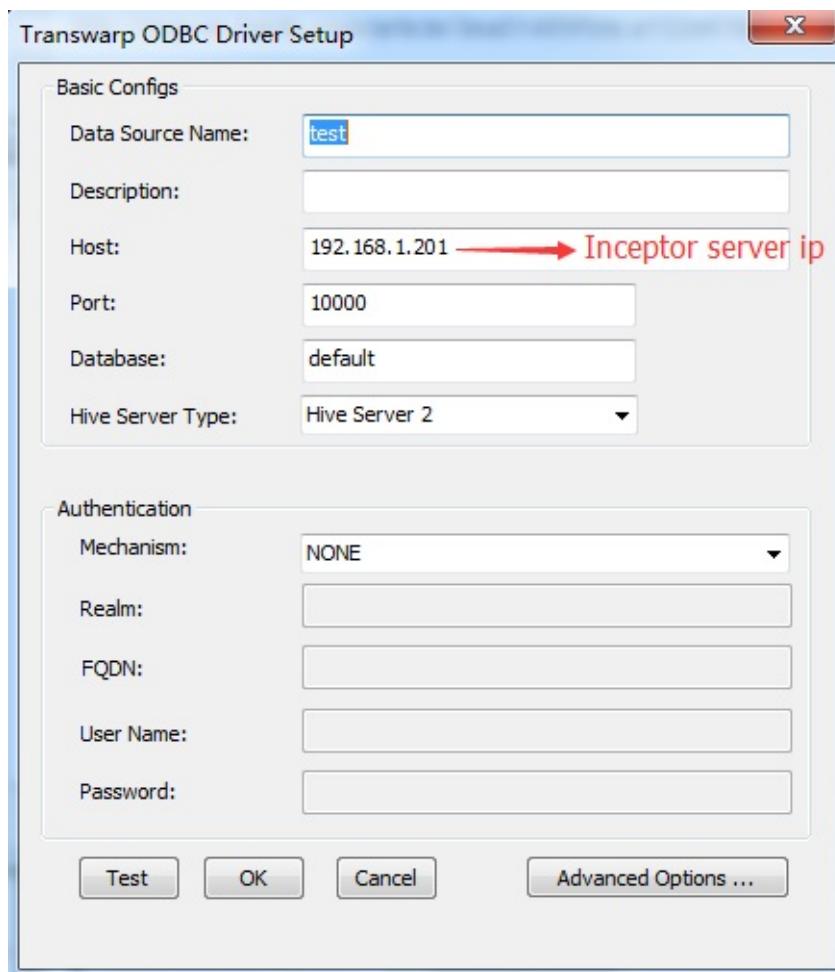
	TranswarpODBCDriver_Hive2_x64.exe	2016/2/24 15:38	应用程序	3,968 KB
	TranswarpODBCDriver_Hive2_x86.exe	2016/2/24 15:38	应用程序	3,318 KB

### 3、添加ODBC驱动程序

在CMD中输入odbc，单击添加，找到刚刚安装好的驱动并填写连接参数



## 5 使用JDBC、ODBC工具连接Inceptor



点击Test，测试成功后点击OK即可

## 第二章 进入SQL兼容性测试

### 查询数据

#### 1 DISTINCT关键字

Mysql:

```
mysql> select DISTINCT title_id from titles;
+-----+
| title_id |
+-----+
| BU1032   |
| BU1111   |
| BU2075   |
| BU7832   |
| MC2222   |
| MC3021   |
| MC3026   |
| PC1035   |
| PC8888   |
| PC9999   |
| PS1372   |
| PS2091   |
| PS2106   |
| PS3333   |
| PS7777   |
| TC3218   |
| TC4203   |
| TC7777   |
+-----+
18 rows in set (0.00 sec)
```

Inceptor:

```
0: jdbc:hive2://192.168.1.70:10000/> select DISTINCT title_id fr  
om titles;  
+-----+  
| title_id |  
+-----+  
| BU7832   |  
| BU2075   |  
| MC3026   |  
| MC2222   |  
| PS7777   |  
| PS1372   |  
| PC8888   |  
| TC7777   |  
| BU1032   |  
| MC3021   |  
| PC9999   |  
| TC3218   |  
| PC1035   |  
| BU1111   |  
| PS2091   |  
| PS3333   |  
| PS2106   |  
| TC4203   |  
+-----+  
18 rows selected (7.386 seconds)
```

## 2 限制结果

Mysql:

```
mysql> select title from titles limit 5;
+-----+
| title
| |
+-----+
| But Is It User Friendly?
| |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |
| Cooking with Computers: Surreptitious Balance Sheets
| |
| Emotional Security: A New Algorithm
| |
| Fifty Years in Buckingham Palace Kitchens
| |
+-----+
--+
5 rows in set (0.00 sec)
```

Inceptor:

```
0: jdbc:hive2://192.168.1.70:10000> select title from titles li
mit 5;
+-----+
|          title          |
+-----+
| The Busy Executive's Database Guide      |
| Cooking with Computers: Surreptitious Balance Sheets |
| You Can Combat Computer Stress!        |
| Straight Talk About Computers          |
| Silicon Valley Gastronomic Treats     |
+-----+
5 rows selected (1.918 seconds)
```

## 排序查询数据

## 1 Order语句:

Mysql:

```
mysql> select title,price from titles order by price;
+-----+-----+
| title | price |
+-----+-----+
| Net Etiquette | NULL |
| The Psychology of Computer Cooking | NULL |
| You Can Combat Computer Stress! | 2.9900 |
| The Gourmet Microwave | 2.9900 |
| Life Without Fear | 7.0000 |
| Emotional Security: A New Algorithm | 7.9900 |
| Is Anger the Enemy? | 10.9500 |
| Cooking with Computers: Surreptitious Balance Sheets | 11.9500 |
| Fifty Years in Buckingham Palace Kitchens | 11.9500 |
| Sushi, Anyone? | 14.9900 |
| Prolonged Data Deprivation: Four Case Studies | 19.9900 |
| Silicon Valley Gastronomic Treats | 19.9900 |
| Straight Talk About Computers | 19.9900 |
| The Busy Executive's Database Guide | 19.9900 |
| Secrets of Silicon Valley |
```

```
| 20.0000 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
n | 20.9500 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | 21.5900 |
| But Is It User Friendly?
| 22.9500 |
+-----+
--+
18 rows in set (0.00 sec)
```

```
mysql> select title,price from titles order by price desc;
+-----+
--+
| title
| price   |
+-----+
--+
| But Is It User Friendly?
| 22.9500 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | 21.5900 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
n | 20.9500 |
| Secrets of Silicon Valley
| 20.0000 |
| The Busy Executive's Database Guide
| 19.9900 |
| Prolonged Data Deprivation: Four Case Studies
| 19.9900 |
| Straight Talk About Computers
| 19.9900 |
| Silicon Valley Gastronomic Treats
| 19.9900 |
| Sushi, Anyone?
| 14.9900 |
| Cooking with Computers: Surreptitious Balance Sheets
| 11.9500 |
| Fifty Years in Buckingham Palace Kitchens
| 11.9500 |
```

```

| Is Anger the Enemy?
| 10.9500 |
| Emotional Security: A New Algorithm
| 7.9900 |
| Life Without Fear
| 7.0000 |
| The Gourmet Microwave
| 2.9900 |
| You Can Combat Computer Stress!
| 2.9900 |
| The Psychology of Computer Cooking
|      NULL |
| Net Etiquette
|      NULL |
+-----+
--+
18 rows in set (0.00 sec)

```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000/> select title,price from titles order by price;
+-----+
--+
|          title
|      price |
+-----+
--+
| Sushi
| 0877      |
| Is Anger the Enemy?
| 10.9500    |
| Cooking with Computers: Surreptitious Balance Sheets
| 11.9500    |
| Fifty Years in Buckingham Palace Kitchens
| 11.9500    |
| The Busy Executive's Database Guide
| 19.9900    |
| Straight Talk About Computers

```

```

| 19.9900      |
| Silicon Valley Gastronomic Treats
| 19.9900      |
| Prolonged Data Deprivation: Four Case Studies
| 19.9900      |
| You Can Combat Computer Stress!
| 2.9900      |
| The Gourmet Microwave
| 2.9900      |
| Secrets of Silicon Valley
| 20.0000      |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | 21.5900      |
| But Is It User Friendly?
| 22.9500      |
| Life Without Fear
| 7.0000      |
| Emotional Security: A New Algorithm
| 7.9900      |
| The Psychology of Computer Cooking
| null        |
| Net Etiquette
| null        |
| Onions
| trad_cook    |
+-----+
---+-----+
18 rows selected (1.993 seconds)

```

## 过滤数据

Mysql:

```

mysql> select title from titles where price>=14;
+-----+
--+
| title
|

```

```

+-----+
--+
| The Busy Executive's Database Guide
|
| Straight Talk About Computers
|
| Silicon Valley Gastronomic Treats
|
| But Is It User Friendly?
|
| Secrets of Silicon Valley
|
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |
| Prolonged Data Deprivation: Four Case Studies
|
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
n |
| Sushi, Anyone?
|
+-----+
--+
9 rows in set (0.00 sec)

```

```

mysql> select title from titles where price between 5 and 10;
+-----+
| title
+-----+
| Life Without Fear
| Emotional Security: A New Algorithm
+-----+
2 rows in set (0.00 sec)

```

```

mysql> select title from titles where price IS NULL;
+-----+
| title
+-----+
| The Psychology of Computer Cooking
| Net Etiquette
+-----+

```

```
2 rows in set (0.00 sec)
```

inceptor:

```
0: jdbc:hive2://192.168.1.70:10000/> select title from titles where price>=14;
+-----+
---+          title
| |
+-----+
---+
| The Busy Executive's Database Guide
|
| Straight Talk About Computers
|
| Silicon Valley Gastronomic Treats
|
| But Is It User Friendly?
|
| Secrets of Silicon Valley
|
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |
| Prolonged Data Deprivation: Four Case Studies
|
| Sushi
|
+-----+
---+
8 rows selected (7.862 seconds)
```

```
0: jdbc:hive2://192.168.1.70:10000/> select title from titles where price between 5 and 10;
+-----+
|          title          |
+-----+
| Life Without Fear      |
| Emotional Security: A New Algorithm |
```

```
+-----+
2 rows selected (1.413 seconds)

0: jdbc:hive2://192.168.1.70:10000/> select title,price from titles where price='null' ;
+-----+-----+
|          title          | price |
+-----+-----+
| The Psychology of Computer Cooking | null  |
| Net Etiquette                  | null  |
+-----+-----+
2 rows selected (1.162 seconds)
```

## AND操作符

MYSQL:

```
mysql> select title,title_id,price,pubdate from titles
-> where title_id='BU1032' and price>10;
+-----+-----+-----+-----+
| title           | title_id | price   | pub
date           |
+-----+-----+-----+-----+
| The Busy Executive's Database Guide | BU1032  | 19.9900 | 199
1-06-12 00:00:00 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Inceptor:

```
0: jdbc:hive2://192.168.1.70:10000/> select title,title_id,price  
,pubdate from titles  
.....> where title_id='BU1032' and  
price>10;  
+-----+-----+-----+  
|       title          | title_id | price  |  
| pubdate           |  
+-----+-----+-----+  
| The Busy Executive's Database Guide | BU1032   | 19.9900 |  
1991-06-12 00:00:00.0 |  
+-----+-----+-----+  
1 row selected (1.153 seconds)
```

## OR操作符

MYSQL:

```
mysql> select title,title_id,price from titles where title_id='BU1032' or price>10;
+-----+-----+
| title | title_id | price |
+-----+-----+
| The Busy Executive's Database Guide | BU1032 | 19.9900 |
| Cooking with Computers: Surreptitious Balance Sheets | BU1111 | 11.9500 |
| Straight Talk About Computers | BU7832 | 19.9900 |
| Silicon Valley Gastronomic Treats | MC2222 | 19.9900 |
| But Is It User Friendly? | PC1035 | 22.9500 |
| Secrets of Silicon Valley | PC8888 | 20.0000 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation | PS1372 | 21.5900 |
| Is Anger the Enemy? | PS2091 | 10.9500 |
| Prolonged Data Deprivation: Four Case Studies | PS3333 | 19.9900 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | TC3218 | 20.9500 |
| Fifty Years in Buckingham Palace Kitchens | TC4203 | 11.9500 |
| Sushi, Anyone? | TC7777 | 14.9900 |
+-----+
--+-----+-----+
12 rows in set (0.01 sec)
```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000/> select title,title_id,price
  from titles where title_id='BU1032' or price>10;
+-----+
---+-----+-----+
|           title
|   | title_id   |   price   |
+-----+
---+-----+-----+
| The Busy Executive's Database Guide
|   | BU1032     | 19.9900  |
| Cooking with Computers: Surreptitious Balance Sheets
|   | BU1111     | 11.9500  |
| Straight Talk About Computers
|   | BU7832     | 19.9900  |
| Silicon Valley Gastronomic Treats
|   | MC2222     | 19.9900  |
| But Is It User Friendly?
|   | PC1035     | 22.9500  |
| Secrets of Silicon Valley
|   | PC8888     | 20.0000  |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | PS1372     | 21.5900  |
| Is Anger the Enemy?
|   | PS2091     | 10.9500  |
| Prolonged Data Deprivation: Four Case Studies
|   | PS3333     | 19.9900  |
| Fifty Years in Buckingham Palace Kitchens
|   | TC4203     | 11.9500  |
| Sushi
|   | TC7777     | 0877      |
+-----+
---+-----+-----+
11 rows selected (1.023 seconds)

```

## IN操作符

Mysql:

```
mysql> select title,title_id,price from titles where title_id in
 ('BU1032','BU1111');
+-----+-----+
| title | title_i
d | price |
+-----+-----+
| The Busy Executive's Database Guide | BU1032
| 19.9900 |
| Cooking with Computers: Surreptitious Balance Sheets | BU1111
| 11.9500 |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> select title,title_id,price from titles where title_id no
t in ('BU1032','BU1111');
+-----+-----+
| title
| title_id | price |
+-----+-----+
| You Can Combat Computer Stress!
| BU2075 | 2.9900 |
| Straight Talk About Computers
| BU7832 | 19.9900 |
| Silicon Valley Gastronomic Treats
| MC2222 | 19.9900 |
| The Gourmet Microwave
| MC3021 | 2.9900 |
| The Psychology of Computer Cooking
| MC3026 | NULL |
| But Is It User Friendly?
| PC1035 | 22.9500 |
| Secrets of Silicon Valley
| PC8888 | 20.0000 |
| Net Etiquette
```

```

| PC9999    |      NULL |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | PS1372    | 21.5900 |
| Is Anger the Enemy?
| PS2091    | 10.9500 |
| Life Without Fear
| PS2106    | 7.0000 |
| Prolonged Data Deprivation: Four Case Studies
| PS3333    | 19.9900 |
| Emotional Security: A New Algorithm
| PS7777    | 7.9900 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
n | TC3218    | 20.9500 |
| Fifty Years in Buckingham Palace Kitchens
| TC4203    | 11.9500 |
| Sushi, Anyone?
| TC7777    | 14.9900 |
+-----+
--+
16 rows in set (0.00 sec)

```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000/> select title,title_id,price
   from titles where title_id in ('BU1032','BU1111');
+-----+
--+
|          title          | title_
id  | price   |
+-----+
--+
| The Busy Executive's Database Guide | BU1032
| 19.9900 |
| Cooking with Computers: Surreptitious Balance Sheets | BU1111
| 11.9500 |
+-----+
--+
2 rows selected (1.017 seconds)

```

```
0: jdbc:hive2://192.168.1.70:10000> select title,title_id,price
  from titles where title_id not in ('BU1032','BU1111');
+-----+
|-----+-----+
|          title
|  title_id |      price   |
+-----+-----+
| You Can Combat Computer Stress!
| BU2075    | 2.9900    |
| Straight Talk About Computers
| BU7832    | 19.9900   |
| Silicon Valley Gastronomic Treats
| MC2222    | 19.9900   |
| The Gourmet Microwave
| MC3021    | 2.9900    |
| The Psychology of Computer Cooking
| MC3026    | null       |
| But Is It User Friendly?
| PC1035    | 22.9500   |
| Secrets of Silicon Valley
| PC8888    | 20.0000   |
| Net Etiquette
| PC9999    | null       |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
| PS1372    | 21.5900   |
| Is Anger the Enemy?
| PS2091    | 10.9500   |
| Life Without Fear
| PS2106    | 7.0000    |
| Prolonged Data Deprivation: Four Case Studies
| PS3333    | 19.9900   |
| Emotional Security: A New Algorithm
| PS7777    | 7.9900    |
| Onions
| TC3218    | trad_cook  |
| Fifty Years in Buckingham Palace Kitchens
| TC4203    | 11.9500   |
| Sushi
| TC7777    | 0877      |
```

```
+-----+  
---+-----+-----+  
16 rows selected (1.121 seconds)
```

## Like通配符

Mysql:

```
mysql> select title,title_id,price from titles where title like  
'You %';  
+-----+-----+-----+  
| title | title_id | price |  
+-----+-----+-----+  
| You Can Combat Computer Stress! | BU2075 | 2.9900 |  
+-----+-----+-----+  
1 row in set (0.01 sec)  
  
mysql> select title,title_id,price from titles where title like  
'I_ %';  
+-----+-----+-----+  
| title | title_id | price |  
+-----+-----+-----+  
| Is Anger the Enemy? | PS2091 | 10.9500 |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000> select title,title_id,price
  from titles where title like 'You %';
+-----+-----+-----+
|       title          | title_id | price  |
+-----+-----+-----+
| You Can Combat Computer Stress! | BU2075   | 2.9900 |
+-----+-----+-----+
1 row selected (1.057 seconds)

0: jdbc:hive2://192.168.1.70:10000> select title,title_id,price
  from titles where title like 'I_ %';
+-----+-----+-----+
|       title          | title_id | price  |
+-----+-----+-----+
| Is Anger the Enemy? | PS2091   | 10.9500|
+-----+-----+-----+
1 row selected (1.472 seconds)

```

## 使用数据处理函数

函数没有SQL的可移植性强,能运行在多个系统上的代码称为可移植的,相对来说,多数SQL语句是可移植的,在SQL实现上有一定的差异,这些差异通常不那么难处理.而函数的可移植性却并不强.几乎每种主要的DBMS的实现都支持其他实现不支持的函数,而且有时差异还很大.

为了代码的可移植性,请保证做好代码的注释,以便以后你能确切地知道所编写SQL代码的含义.

MYSQL:

```

mysql> select upper(title) from titles;
+-----+
| upper(title) |
+-----+
| BUT IS IT USER FRIENDLY? |
+-----+

```

```
|  
| COMPUTER PHOBIC AND NON-PHOBIC INDIVIDUALS: BEHAVIOR VARIATION  
S |  
| COOKING WITH COMPUTERS: SURREPTITIOUS BALANCE SHEETS  
|  
| EMOTIONAL SECURITY: A NEW ALGORITHM  
|  
| FIFTY YEARS IN BUCKINGHAM PALACE KITCHENS  
|  
| IS ANGER THE ENEMY?  
|  
| LIFE WITHOUT FEAR  
|  
| NET ETIQUETTE  
|  
| ONIONS, LEEKS, AND GARLIC: COOKING SECRETS OF THE MEDITERRANEAN  
N |  
| PROLONGED DATA DEPRIVATION: FOUR CASE STUDIES  
|  
| SECRETS OF SILICON VALLEY  
|  
| SILICON VALLEY GASTRONOMIC TREATS  
|  
| STRAIGHT TALK ABOUT COMPUTERS  
|  
| SUSHI, ANYONE?  
|  
| THE BUSY EXECUTIVE'S DATABASE GUIDE  
|  
| THE GOURMET MICROWAVE  
|  
| THE PSYCHOLOGY OF COMPUTER COOKING  
|  
| YOU CAN COMBAT COMPUTER STRESS!  
|  
+-----  
--+  
18 rows in set (0.00 sec)
```

Inceptor:

见TDH官方手册上的函数支持

## 聚集函数

MYSQL:

```
mysql> select avg(price) from titles;
+-----+
| avg(price) |
+-----+
| 14.76625000 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> select count(price) from titles;
+-----+
| count(price) |
+-----+
|          16 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> select max(price) from titles;
+-----+
| max(price) |
+-----+
|    22.9500 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> select min(price) from titles;
+-----+
| min(price) |
+-----+
|    2.9900 |
+-----+
1 row in set (0.00 sec)
```

```

mysql> select sum(price) from titles;
+-----+
| sum(price) |
+-----+
| 236.2600 |
+-----+
1 row in set (0.00 sec)

mysql> select avg( distinct price) from titles;
+-----+
| avg( distinct price) |
+-----+
| 14.66818182 |
+-----+
1 row in set (0.03 sec)

mysql> select count(*) as title_items,
    -> min(price) as price_min,
    -> max(price) as price_max,
    -> avg(price) as price_avg
    -> from titles;
+-----+-----+-----+-----+
| title_items | price_min | price_max | price_avg   |
+-----+-----+-----+-----+
| 18 | 2.9900 | 22.9500 | 14.76625000 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000/> select avg(price) as price_
avg from titles;
+-----+
| price_avg      |
+-----+
| 71.8213333333334 |
+-----+
1 row selected (1.033 seconds)

```

```
0: jdbc:hive2://192.168.1.70:10000/> select count(price) from titles;
+-----+
| _c0   |
+-----+
| 18    |
+-----+
1 row selected (1.339 seconds)

0: jdbc:hive2://192.168.1.70:10000/> select avg( distinct price)
  from titles;
+-----+
|      _c0      |
+-----+
| 100.24100000000001 |
+-----+

0: jdbc:hive2://192.168.1.70:10000/> select count(*) as title_items,min(price) as price_min,max(price) as price_max,avg(price) as price_avg from titles;
+-----+-----+-----+-----+
| title_items | price_min | price_max | price_avg
+-----+-----+-----+
| 18          | 0877     | trad_cook  | 71.8213333333334
+-----+-----+-----+
1 row selected (0.899 seconds)
```

## 数据分组

Mysql:

```
mysql> select type from titles group by type;
+-----+
| type      |
+-----+
| business   |
| mod_cook    |
| popular_comp |
| psychology  |
| trad_cook   |
| UNDECIDED   |
+-----+
6 rows in set (0.01 sec)
```

```
mysql> select type,avg(price)as book_avg_price from titles group
by type having book_avg_price>10;
+-----+-----+
| type      | book_avg_price |
+-----+-----+
| business   | 13.73000000 |
| mod_cook    | 11.49000000 |
| popular_comp | 21.47500000 |
| psychology  | 13.50400000 |
| trad_cook   | 15.96333333 |
+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select title_id,sum(qty) as book_sum_qty from sales where
title_id in (select title_id from titles where title='Sushi, An
yone?');
+-----+
| title_id | book_sum_qty |
+-----+
| TC7777   |        20 |
+-----+
1 row in set (0.00 sec)
```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000> select type from titles group by type;
+-----+
|     type      |
+-----+
| psychology   |
| popular_comp |
| trad_cook    |
| business     |
| Leeks         |
| UNDECIDED    |
| mod_cook     |
| Anyone?      |
+-----+

0: jdbc:hive2://192.168.1.70:10000> select type,avg(price)as book_avg_price from titles group by type having book_avg_price>10;
+-----+-----+
|     type      | book_avg_price   |
+-----+-----+
| psychology   | 13.504          |
| popular_comp | 21.475          |
| trad_cook    | 11.95           |
| business     | 13.73            |
| mod_cook     | 11.48999999999998 |
| Anyone?      | 877.0            |
+-----+-----+
6 rows selected (2.829 seconds)

select s.title_id,sum(s.qty) as book_sum_qty from sales as s where s.title_id in (select t.title_id from titles t where t.title= 'Sushi, Anyone?') group by s.title_id;
+-----+-----+
| title_id | book_sum_qty  |
+-----+-----+
+-----+-----+
No rows selected (2.285 seconds)

```

## 表连接

MYSQL:

```
mysql> desc titles;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| title_id | varchar(6) | NO   | PRI | NULL    |       |
| title    | varchar(80) | NO   | MUL | NULL    |       |
| type     | varchar(12) | NO   |      | UNDECIDED |       |
| pub_id   | varchar(4)  | YES  |      | NULL    |       |
| price    | decimal(19,4) | YES  |      | NULL    |       |
| advance  | decimal(19,4) | YES  |      | NULL    |       |
| royalty  | int(11)    | YES  |      | NULL    |       |
| ytd_sales | int(11)    | YES  |      | NULL    |       |
| notes    | varchar(200) | YES  |      | NULL    |       |
| pubdate  | datetime   | NO   |      | NULL    |       |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> desc titleauthor;
+-----+-----+-----+-----+
--+
| Field | Type      | Null | Key | Default | Extr
a |
+-----+-----+-----+-----+
--+
| au_id    | varchar(11) | NO   | PRI | NULL    |       |
|          |             |      |     |          |       |
| title_id | varchar(6)  | NO   | PRI | NULL    |       |
|          |             |      |     |          |       |
| au_ord   | tinyint(3) unsigned | YES  |      | NULL    |       |
|          |             |      |     |          |       |
| royaltyper | int(11)    | YES  |      | NULL    |       |
|          |             |      |     |          |       |
+-----+-----+-----+-----+
--+
4 rows in set (0.00 sec)
```

```
mysql> desc authors;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| au_id  | varchar(11) | NO   | PRI | NULL    |       |
| au_lname | varchar(40) | NO   | MUL | NULL    |       |
| au_fname | varchar(20) | NO   |      | NULL    |       |
| phone   | varchar(12)  | NO   |      | UNKNOWN |       |
| address | varchar(40) | YES  |      | NULL    |       |
| city    | varchar(20)  | YES  |      | NULL    |       |
| state   | varchar(2)   | YES  |      | NULL    |       |
| zip     | varchar(5)   | YES  |      | NULL    |       |
| contract | bit(1)    | NO   |      | NULL    |       |
+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

```
mysql> select title,au_ord from titles inner join titleauthor on
titles.title_id=titleauthor.title_id;
+-----+
| title
| au_ord |
+-----+
--+
| But Is It User Friendly?
|      1 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      2 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      1 |
| Cooking with Computers: Surreptitious Balance Sheets
|      2 |
| Cooking with Computers: Surreptitious Balance Sheets
|      1 |
| Emotional Security: A New Algorithm
|      1 |
| Fifty Years in Buckingham Palace Kitchens
|      1 |
```

```
| Is Anger the Enemy?  
|     2 |  
| Is Anger the Enemy?  
|     1 |  
| Life Without Fear  
|     1 |  
| Net Etiquette  
|     1 |  
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea  
n |     1 |  
| Prolonged Data Deprivation: Four Case Studies  
|     1 |  
| Secrets of Silicon Valley  
|     1 |  
| Secrets of Silicon Valley  
|     2 |  
| Silicon Valley Gastronomic Treats  
|     1 |  
| Straight Talk About Computers  
|     1 |  
| Sushi, Anyone?  
|     2 |  
| Sushi, Anyone?  
|     3 |  
| Sushi, Anyone?  
|     1 |  
| The Busy Executive's Database Guide  
|     2 |  
| The Busy Executive's Database Guide  
|     1 |  
| The Gourmet Microwave  
|     1 |  
| The Gourmet Microwave  
|     2 |  
| You Can Combat Computer Stress!  
|     1 |  
+-----+  
--+-----+  
25 rows in set (0.00 sec)
```

```
mysql> select title,au_ord from titles inner join titleauthor on
      titles.title_id=titleauthor.title_id  inner join authors on aut
      hors.au_id=titleauthor.au_id;
+-----+
| title
| au_ord |
+-----+
| But Is It User Friendly?
|     1 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
|     2 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
|     1 |
| Cooking with Computers: Surreptitious Balance Sheets
|     2 |
| Cooking with Computers: Surreptitious Balance Sheets
|     1 |
| Emotional Security: A New Algorithm
|     1 |
| Fifty Years in Buckingham Palace Kitchens
|     1 |
| Is Anger the Enemy?
|     2 |
| Is Anger the Enemy?
|     1 |
| Life Without Fear
|     1 |
| Net Etiquette
|     1 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
|     1 |
| Prolonged Data Deprivation: Four Case Studies
|     1 |
| Secrets of Silicon Valley
|     1 |
| Secrets of Silicon Valley
|     2 |
| Silicon Valley Gastronomic Treats
```

```

|      1 |
| Straight Talk About Computers
|      1 |
| Sushi, Anyone?
|      2 |
| Sushi, Anyone?
|      3 |
| Sushi, Anyone?
|      1 |
| The Busy Executive's Database Guide
|      2 |
| The Busy Executive's Database Guide
|      1 |
| The Gourmet Microwave
|      1 |
| The Gourmet Microwave
|      2 |
| You Can Combat Computer Stress!
|      1 |
+-----+
--+
25 rows in set (0.00 sec)

```

```

mysql> select title,au_ord from titles left join titleauthor on
titles.title_id=titleauthor.title_id;
+-----+
--+
| title
| au_ord |
+-----+
--+
| But Is It User Friendly?
|      1 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      2 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      1 |
| Cooking with Computers: Surreptitious Balance Sheets
|      2 |
| Cooking with Computers: Surreptitious Balance Sheets

```

```
|      1 |
| Emotional Security: A New Algorithm
|      1 |
| Fifty Years in Buckingham Palace Kitchens
|      1 |
| Is Anger the Enemy?
|      2 |
| Is Anger the Enemy?
|      1 |
| Life Without Fear
|      1 |
| Net Etiquette
|      1 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
n |      1 |
| Prolonged Data Deprivation: Four Case Studies
|      1 |
| Secrets of Silicon Valley
|      1 |
| Secrets of Silicon Valley
|      2 |
| Silicon Valley Gastronomic Treats
|      1 |
| Straight Talk About Computers
|      1 |
| Sushi, Anyone?
|      2 |
| Sushi, Anyone?
|      3 |
| Sushi, Anyone?
|      1 |
| The Busy Executive's Database Guide
|      2 |
| The Busy Executive's Database Guide
|      1 |
| The Gourmet Microwave
|      1 |
| The Gourmet Microwave
|      2 |
| The Psychology of Computer Cooking
```

```
|    NULL |
| You Can Combat Computer Stress!
|      1 |
+-----+
--+
26 rows in set (0.00 sec)

mysql> select title,au_ord from titles right join titleauthor on
titles.title_id=titleauthor.title_id;
+-----+
--+
| title
| au_ord |
+-----+
--+
| Prolonged Data Deprivation: Four Case Studies
|      1 |
| The Busy Executive's Database Guide
|      2 |
| You Can Combat Computer Stress!
|      1 |
| But Is It User Friendly?
|      1 |
| Cooking with Computers: Surreptitious Balance Sheets
|      2 |
| Sushi, Anyone?
|      2 |
| Straight Talk About Computers
|      1 |
| The Busy Executive's Database Guide
|      1 |
| Secrets of Silicon Valley
|      1 |
| Sushi, Anyone?
|      3 |
| Net Etiquette
|      1 |
| Emotional Security: A New Algorithm
|      1 |
| Fifty Years in Buckingham Palace Kitchens
```

```

|      1 |
| Sushi, Anyone?
|      1 |
| Silicon Valley Gastronomic Treats
|      1 |
| The Gourmet Microwave
|      1 |
| Cooking with Computers: Surreptitious Balance Sheets
|      1 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      2 |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s |      1 |
| Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranea
n |      1 |
| Secrets of Silicon Valley
|      2 |
| The Gourmet Microwave
|      2 |
| Is Anger the Enemy?
|      2 |
| Is Anger the Enemy?
|      1 |
| Life Without Fear
|      1 |
+-----+
--+
25 rows in set (0.00 sec)

```

Inceptor:

```

0: jdbc:hive2://192.168.1.70:10000> select title,au_ord from ti
tles inner join titleauthor on titles.title_id=titleauthor.title
_id;
+-----+
--+
|          title
|  au_ord  |
+-----+

```

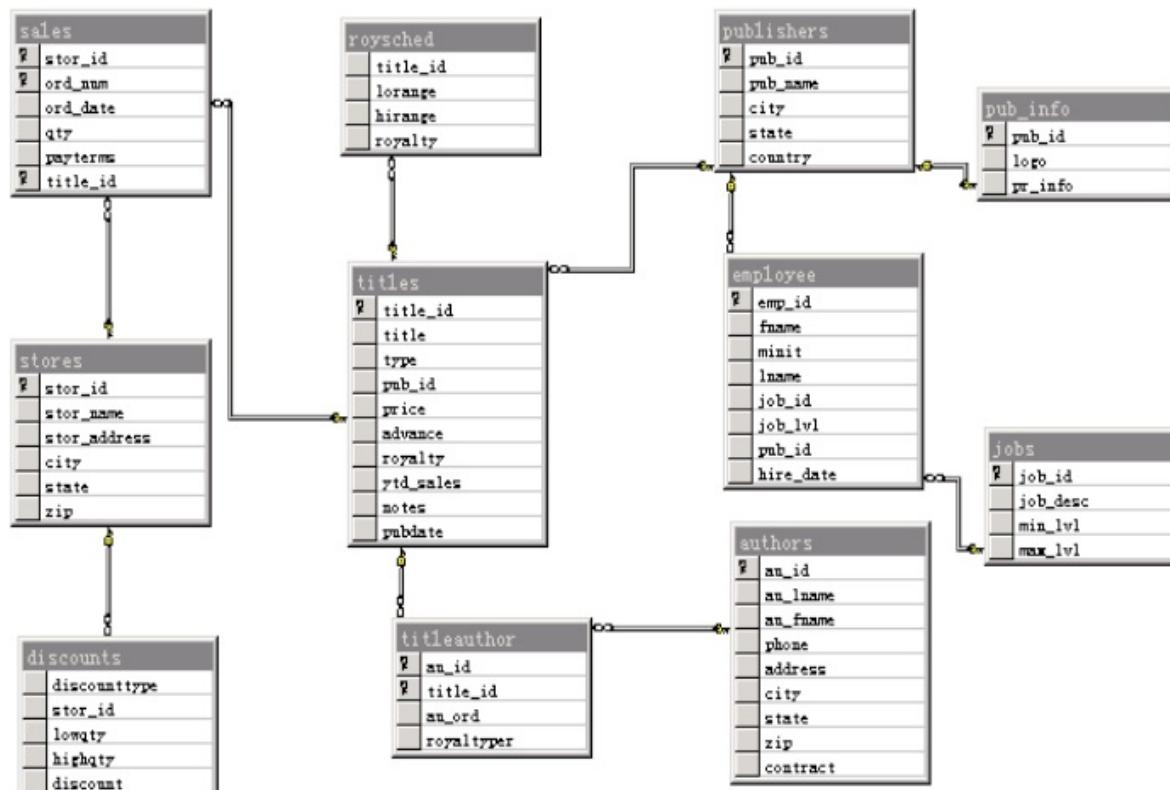
```
----+-----+
| The Busy Executive's Database Guide
|   | 2      |
| The Busy Executive's Database Guide
|   | 1      |
| Cooking with Computers: Surreptitious Balance Sheets
|   | 2      |
| Cooking with Computers: Surreptitious Balance Sheets
|   | 1      |
| You Can Combat Computer Stress!
|   | 1      |
| Straight Talk About Computers
|   | 1      |
| Silicon Valley Gastronomic Treats
|   | 1      |
| The Gourmet Microwave
|   | 1      |
| The Gourmet Microwave
|   | 2      |
| But Is It User Friendly?
|   | 1      |
| Secrets of Silicon Valley
|   | 1      |
| Secrets of Silicon Valley
|   | 2      |
| Net Etiquette
|   | 1      |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | 2      |
| Computer Phobic AND Non-Phobic Individuals: Behavior Variation
s | 1      |
| Is Anger the Enemy?
|   | 2      |
| Is Anger the Enemy?
|   | 1      |
| Life Without Fear
|   | 1      |
| Prolonged Data Deprivation: Four Case Studies
|   | 1      |
| Emotional Security: A New Algorithm
```

```
| 1      |
| Onions
| 1      |
| Fifty Years in Buckingham Palace Kitchens
| 1      |
| Sushi
| 2      |
| Sushi
| 3      |
| Sushi
| 1      |
+-----+
---+-----+
25 rows selected (1.858 seconds)
```

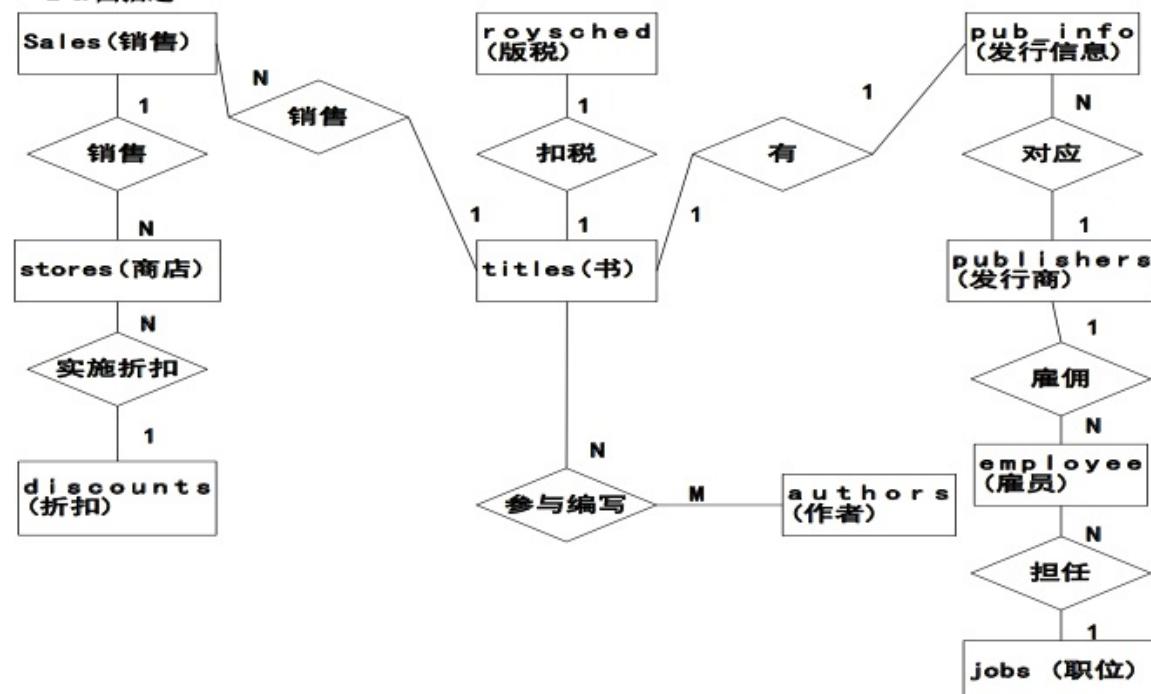
# 第四章 SQL语句练习

该实验以 SQL Server 2000 系统自带的 pubs 数据库为例，以一个图书出版公司为模型。

## ● SQL SERVER200 实体关系图描述



## ● E-R 图描述



**Authors:**

属性名	数据类型	含义说明	可为空	检查	键/索引
au_id	Id	作者编号	否	是 <sup>1</sup>	主键
au_lname	varchar(40)	作者姓	否		
au_fname	varchar(20)	作者名	否		
phone	char(12)	电话	否		
address	varchar(40)	地址	是		
city	varchar(20)	所在城市	是		
state	char(2)	所在州	是		
zip	char(5)	邮编	是	是 <sup>2</sup>	
contract	Bit	是否签约	否		

1 **au\_id** CHECK 约束定义为 (**au\_id** LIKE '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]' )。

2 **zip** CHECK 约束定义为 (**zip** LIKE '[0-9][0-9][0-9][0-9][0-9]' )。

**discounts**

属性名	数据类型	含义说明	可为空	检查	键/索引
discounttype	varchar(40)	折扣类型	否		
stor_id	char(4)	商店编号	是		外键 stores(stor_id)
lowqty	Smallint	数量下限	是		
highqty	Smallint	数量上限	是		
discount	Float	折扣	否		

**Employee**

属性名	数据类型	含义说明	可为空	默认值	检查	键/索引
emp_id	Empid	职工编号	否		是 <sup>1</sup>	主键
fname	varchar(20)	职工名	否			
minit	char(1)		是			
lname	varchar(30)	职工姓	否			
job_id	Smallint	工作编号	否	1		外键 jobs(job_id)
job_lvl	Tinyint		否	10		
pub_id	char(4)	出版社编号	否	'9952'		外键 publishers(pub_id)
Hire_date	Datetime	工作日期	否	GETDATE()		

CHECK 约束定义为：

(**emp\_id** LIKE '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]') OR  
 (**emp\_id** LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][0-9][FM]' )。

**Jobs**

属性名	数据类型	含义说明	可为空	检查	键/索引
job_id	Smallint	工作编号	否		主键
job_desc	varchar(50)	工作描述	否		
min_lvl	Tinyint		否	是 <sup>1</sup>	
max_lvl	Tinyint		否	是 <sup>2</sup>	

(1) min\_lvl CHECK 约束定义为 (min\_lvl >= 10)。

(2) max\_lvl CHECK 约束定义为 (max\_lvl <= 250)。

---

**pub\_info**

属性名	数据类型	含义说明	可为空	检查	键/索引
pub_id	char(4)	出版社编号	否		主键, 外键 publishers(pub_id)
logo	Image	标志图	是		
pr_info	Text	出版信息	是		

---

**Publishers**

属性名	数据类型	含义说明	可为空	检查	键/索引
pub_id	char(4)	出版社编号	否	是 <sup>1</sup>	主键
pub_name	varchar(40)	出版社名称	是		
city	varchar(20)	所在城市	是		
state	char(2)	所在州	是		
country	varchar(30)	所在国家	是		

1 pub\_id CHECK 约束定义为

(pub\_id = '1756' OR (pub\_id = '1622' OR (pub\_id = '0877' OR (pub\_id = '0736' OR (pub\_id = '1389')))) OR (pub\_id LIKE '99[0-9][0-0]'))。

1 pub\_id CHECK 约束定义为

(pub\_id = '1756' OR (pub\_id = '1622' OR (pub\_id = '0877' OR (pub\_id = '0736' OR (pub\_id = '1389')))) OR (pub\_id LIKE '99[0-9][0-0]')).

---

### roysched

属性名	数据类型	含义说明	可为空	检查	键/索引
title_id	Tid	书编号	否		外键 titles(title_id)
lorange	Int	低	是		
hirange	Int	高	是		
royalty	Int	版权	是		

---

### Sales

属性名	数据类型	含义说明	可为空	键/索引
stor_id	char(4)	商店编号	否	组合主键, 聚集索引, 外键 stores(stor_id)

ord_num	varchar(20)	订单编码	否	组合主键, 聚集索引
ord_date	Datetime	订购日期	否	
qty	Smallint	数量	否	
payterms	varchar(12)	付款方式	否	
title_id	Tid	书编号	否	组合主键, 聚集索引, 外键 titles(title_id)

---

### titles

属性名	数据类型	含义说明	可为空	检查	键/索引
title_id	Tid	书编号	否		主键
title	varchar(80)	书名	否		
type	char(12)	类型	否		
pub_id	char(4)	出版社编号	是		外键 publishers (pub_id)
price	Money	价格	是		
advance	Money	预付款	是		
royalty	Int	版税	是		
Ytd_sales	Int	年销售量	是		
notes	varchar(200)	简介	是		
pubdate	Datetime	出版日期	是		

**Stores**

属性名	数据类型	含义说明	可为空	检查	键/索引
stor_id	char(4)	商店编号	否		主键
stor_name	varchar(40)	商店名称	是		
stor_address	varchar(40)	商店地址	是		
city	varchar(20)	所在城市	是		
state	char(2)	所在州	是		
zip	char(5)	邮编	是		

**titleauthor**

属性名	数据类型	含义说明	可为空	检查	键/索引
au_id	id	作者编号	否		组合主键, 聚集索引, 外键 authors(au_id)
title_id	tid	书编号	否		组合主键, 聚集索引, 外键 titles(title_id)
au_ord	tinyint		是		

**titleauthor**

属性名	数据类型	含义说明	可为空	检查	键/索引
au_id	id	作者编号	否		组合主键, 聚集索引, 外键 authors(au_id)
title_id	tid	书编号	否		组合主键, 聚集索引, 外键 titles(title_id)
au_ord	tinyint		是		

royaltyper	int	版权百分比	是		
------------	-----	-------	---	--	--

## SQL 练习

### 目的1：

1. 加深对表间关系的理解。
2. 理解数据库中数据的查询方法和应用。
3. 学会各种查询的异同及相互之间的转换方法。

### 内容1：

1. 查询所有作者的作者号、姓名信息
2. 查询所有作者的姓名、作者号信息，并在每个作者的作者号前面显示字符串“身份证号：”，表明显示的信息是身份证件信息
3. 查询在CA州的作者姓名和城市
4. 查询出版日期在1992.1.1-2000.12.31之间的书名和出版日期(查询1991年出版的书)
5. 查询每个出版社出版的书
6. 查询某店销售某书的数量
7. 查询有销售记录的所有书信息，包括书的编号、书名、类型和价格
8. 查询已销售书的信息
9. 显示所有的书名（无销售记录的书也包括在内）
10. 查询已销售书的信息（书号、书名、作者等）
11. 查询所有出版商业（business）书籍的出版社的名称

### 目的2：

1. 理解数据库中数据的其他查询方法和应用；
2. 学会各种查询要求的实现。

### 内容2：

在实验1的基础上，练习查询语句的使用，包括计算列、求和、最大、最小值、各类选择条件、字符匹配、分组和排序，体会各种查询的执行过程，为简单综合应用打下良好的基础。

1. 查询书名以T开头或者出版社号为0877，而且价格大于16的书的信息。

2. 按照类型的升序和价格的降序（在类型相同时）显示书的信息（书名、作者、出版社、类型、价格）
3. 查询销售量大于30的书名及销售数量
4. 查询在2002.1.1到2002.10.31间，每本书的销售总额
5. 查询所有作者的所在城市和州名，要求没有重复信息
6. 计算多少种书已被订价
7. 查询每本书的书名、作者及它的售书总量
8. 计算所有书的平均价格
9. 查询价格最高的书的书名、作者及价格

### 目的3：

1. 加深对数据库相关性质的理解；
2. 各种约束性理解；
3. 学会数据库中数据的更新的方法。

### 内容3：

1. 参照以上各表给出的主键、外键设置的设置要求，在自己创建的表中进行相应的设置。
2. 向authors表中插入一行作者信息（具体值自定）
3. 数量超过100的商店增加10%的折扣
4. 删掉2001.10.3的订单
5. 删掉1中所建立的索引
6. 建立CA州作者所著书的视图（包括作者号、姓名、所在州、书名、价格、出版日期）
7. 建立付款方式是现金（cash）的订单视图
8. 建立CA州的所有商店的视图

### 目的4：

1. 在查询分析器中，练习使用IN、比较符、ANY或ALL等操作符进行查询。
2. 练习使用EXISTS操作符进行嵌套查询操作

### 内容4：

1. 在pubs数据库的titleauthor和中，用IN谓词查询来自‘CA’州(在authors表中)的作家的全部作品 (title\_id) 和作家的代号(au\_id)。
2. 在pubs数据库中，用比较运算符引出的子查询找出在名称为“Algodata Infosystems”的出版社所在城市中的作者的姓名 (au\_lname, au\_fname)
3. 在pubs数据库中的titles表中，查询价格大于所有类型(TYPE)为“business”的图书价格的书名 (title) 和价格(price)
4. 在pubs数据库的sales表中查找所有销售量大于所有图书平均销售量 avg (qty) ) 的书的代号(title\_id)及销售量 (qty) 。
5. 用带有IN的嵌套查询，查询来自城市 (city) 为“London”的客户所订的订单信息 (customers和orders表) 。
6. 用带有IN的嵌套查询，查询Northwind数据库中的产品表 (Products) 中来自国家为“Germany” (在供应商表 (Suppliers) 表中) 的供货商供应的产品信息 (包括Productid , Productname , categoryid, unitprice) 。
7. 使用EXISTS子查询在Pubs数据库titles 表及publishers表中查询New Moon Books出版社所出版的图书名称 (title)

## 目的5：

1. 分类汇总。

## 内容5：

1. 找出pubs数据库titles表中计算机类图书中价格最高的图书的价格。
2. 查询titles表中有几类图书。
3. 按照州进行分类，查找每个州有几名作者。
4. 要求按照出版商id进行分类，查找每个出版商的书到目前为止的销售额总和 (ytd\_sales) 。
5. 在pubs数据库的titles表中，找出平均价格大于18美元的书的种类。
6. 在pubs数据库的titles表中，找出最高价大于20美元的书的种类。
7. 找出title\_id和pub\_name的对应关系。
8. 找出title\_id, title和pub\_name的对应关系。
9. 查询每个作者的编号，姓名，所出的书的编号，并对结果排序。
10. 从authors表中选择state,city列，从publisher表中选择state,city列，并把两个查询的结果合并为一个结果集，并对结果集按city列、state列进行排序。
11. 对上面的查询语句作修改，保留所有重复的记录。
12. 显示所有来自CA州的作家的全部作品和作家代号。 (使用IN，和连接两种方

法)

13. 查找由位于以字母 B 开头的城市中的任一出版商出版的书名：(使用exists和in两种方法)

## SQL具体操作练习

### 简单查询学生选课数据问题

1. 列出全部学生的信息。
2. 列出信息系全部学生的学号及姓名。
3. 列出所有已被选修的选修课的课号。
4. 求c01号课成绩大于80分的学生的学号及成绩，并按成绩由高到低列出。
5. 列出非信息系学生的名单。
6. 查询成绩在70~80分之间的学生选课得分情况
7. 列出选修c01号课或c03号课的全体学生的学号和成绩。
8. 列出所有95级学生的学生成绩情况。
9. 列出成绩为空值(或不为空值)的学生的学号和课号。
10. 求出所有学生的总成绩。
11. 列出每个学生的平均成绩。
12. 列出各科的平均成绩、最高成绩、最低成绩和选课人数。

### 简单查询学生选课数据问题答案

1. SELECT \* FROM 学生
2. SELECT 学号,姓名 FROM 学生 WHERE 专业='信息系'
3. SELECT DISTINCT 课号 FROM 选修课
4. SELECT 学号,成绩 FROM 选课 WHERE 课号='01' AND 成绩>80 ORDER BY 成绩 DESC
5. 方法一：SELECT 姓名 FROM 学生 WHERE 专业<>'信息系'  
方法二：SELECT 姓名 FROM 学生 WHERE NOT 专业='信息系'  
方法三：SELECT 姓名 FROM 学生 WHERE 专业!= '信息系'
6. 方法一：SELECT \* FROM 选课 WHERE 成绩>=70 AND 成绩<=80  
方法二：SELECT \* FROM 选课 WHERE 成绩 BETWEEN 70 AND 80  
不在此范围内的查询：（注意写出和以下语句等价的语句）  
SELECT \* FROM 选课 WHERE 成绩 NOT BETWEEN 70 AND 80
7. 方法一：SELECT 学号,成绩 FROM 选课 WHERE 课号='c01' OR 课号='c03'  
方法二：SELECT 学号,成绩 FROM 选课 WHERE 课号 IN ('c01', 'c03')  
相反条件查询：SELECT 学号,成绩 FROM 选课 WHERE 课号 NOT IN ('c01', 'c03')
8. SELECT \* FROM 选课 WHERE 学号 LIKE '95%'  
SELECT \* FROM 选课 WHERE 学号 LIKE '95\_\_\_\_\_'  
相反条件查询：SELECT \* FROM 选课 WHERE 学号 NOT LIKE '98%'
9. 答案一：SELECT 学号,课号 FROM 选课 WHERE 成绩 IS NULL  
答案二：SELECT 学号,课号 FROM 选课 WHERE 成绩 IS NOT NULL
10. SELECT SUM(成绩) AS 总成绩 FROM 选课
11. SELECT 学号,AVG(成绩) AS 平均成绩 FROM 选课 GROUP BY 学号
12. SELECT 课号,AVG(成绩) AS 平均成绩,MAX(成绩) AS 最高分,  
MIN(成绩) AS 最低分,COUNT(学号) AS 选课人数 FROM 选课 GROUP BY 课号

## 目的4：

1. 在查询分析器中，练习使用IN、比较符、ANY或ALL等操作符进行查询。
2. 练习使用EXISTS操作符进行嵌套查询操作

## 请完成以下习题：

1. 在pubs数据库的titleauthor表中，用IN谓词查询来自‘CA’州(在authors表中)的作家的全部作品 (title\_id) 和作家的代号(au\_id)。

```
select title_id,au_id from titleauthor
```

```

        where au_id in (select au_id from authors
where state='CA')

```

在pubs数据库中，用比较运算符引出的子查询找出在名称为“Algodata Infosystems”的出版社所在城市中的作者的姓名 (au\_lname, au\_fname)

```

select au_lname, au_fname
from authors
where city= (select city
      from publishers
      where pub_name='Algodata Infosystems')

```

在pubs数据库中的titles表中，查询价格大于所有类型(TYPE)为“business”的图书价格的书名 (title) 和价格(price)

```

select title,price
from titles
where price>all (select price
      from titles
      where type='business')

```

在pubs数据库的sales表中查找所有销售量大于所有图书平均销售量avg (qty) ) 的书的代号(title\_id)及销售量 (qty) 。

```

select title_id ,qty
from sales
where qty>all(select avg(qty)
from sales
)

```

用带有IN的嵌套查询，查询来自城市 (city) 为“London”的客户所订的订单信息 (customers和orders表) 。

```

select *
from orders
where customerID in (select customerID
      from customers
      where city='london')

```

用带有IN的嵌套查询，查询Northwind数据库中的产品表 (Products) 中来自国家为“Germany” (在供应商表 (Suppliers) 表中) 的供货商供应的产品信息 (包括Productid, Productname, categoryid, unitprice) 。

```

SELECT Productid,Productname, categoryid,unitprice
from Products
where Productid in(select supplierID
from Suppliers
where country='Germany')

```

## 提高操作实验

练习使用 EXISTS 操作符进行嵌套查询操作。请完成以下习题：

使用 EXISTS 子查询在 Pubs 数据库 titles 表及 publishers 表中查询 New Moon Books 出版社所出版的图书名称 (title)

```

select title
      from titles
     where exists (select *
                   from publishers
                  where pub_name='New Moon Books')

```

### T-SQL 高级查询课堂练习及答案

--练习1

--找出pubs数据库titles表中计算机类图书中价格最高的图书的价格。

USE pubs

GO

SELECT max(price) FROM titles

where type='popular\_comp'

GO

--练习2

--查询titles表中有几类图书。

USE pubs

GO

SELECT count(distinct type) FROM titles

GO

--练习3

--按照州进行分类，查找每个州有几名作者。

USE pubs

```

GO
SELECT state, count(*) FROM authors
group by state
order by 1
GO

```

--练习4

--要求按照出版商id进行分类，查找每个出版商的书到目前为止的销售额总和(ytd\_sales)。

```

USE pubs
GO
SELECT pub_id, sum(ytd_sales) FROM titles
group by pub_id
order by 1
GO

```

--练习5

--在pubs数据库的titles表中，找出平均价格大于18美元的书的种类。

```

USE pubs
GO
SELECT pub_id, avg(price) '平均价格' FROM titles
GROUP BY pub_id
HAVING avg(price) > 18
GO

```

--练习6

--在pubs数据库的titles表中，找出最高价大于20美元的书的种类。

```

USE pubs
GO
SELECT type, max(price) '平均价格' FROM titles
GROUP BY type
HAVING max(price) > 20
GO

```

--练习7

--

找出title\_id和pub\_name的对应关系。

```
Use pubs
```

```

go
Select titles.title_id, publishers.pub_name
From titles JOIN publishers
ON titles.pub_id=publishers.pub_id
Go

```

--练习8

--找出title\_id, title和pub\_name的对应关系。

```
Use pubs
```

```
go
```

```

Select titles.title_id, titles.title, publishers.pub_name
From titles JOIN publishers
ON titles.pub_id=publishers.pub_id
Go

```

--练习9

--查询每个作者的编号，姓名，所出的书的编号，并对结果排序。

```
Use pubs
```

```
go
```

```

Select authors.au_id,
       authors.au_fname + '.' + authors.au_lname 'name',
       titleauthor.title_id
From authors JOIN titleauthor
ON authors.au_id=titleauthor.au_id
order by authors.au_id
go

```

--练习10

--从authors表中选择state,city列，从publisher表中选择state,city列，并把两个查询的结果合并为一个结果集，并对结果集按city列、state列进行排序。

```
use pubs
```

```
go
```

```

select state,city from publishers
union
select state,city from authors
order by 1,2

```

--练习11

-- 对上面的查询语句作修改，保留所有重复的记录。

```
use pubs
go
select state,city from publishers
union all
select state,city from authors
order by 1,2
```

--练习12

--显示所有来自CA州的作家的全部作品和作家代号。（使用IN，和连接两种方法）

```
use pubs
go
select title_id,au_id
from titleauthor
where au_id in
( select au_id from authors
where state = 'CA')
order by title_id
go
```

```
use pubs
go
select t.title_id,t.au_id
from titleauthor t join authors a
on t.au_id = a.au_id
where a.state = 'CA'
order by title_id
go
```

--练习13

--查找由位于以字母 B 开头的城市中的任一出版商出版的书名：(使用exists和in两种方法)

```
USE pubs
GO
SELECT title
FROM titles
WHERE EXISTS
  (SELECT *
```

```
FROM publishers
WHERE pub_id = titles.pub_id
AND city LIKE 'B%')
GO
```

```
USE pubs
GO
SELECT title
FROM titles
WHERE pub_id IN
(SELECT pub_id
FROM publishers
WHERE city LIKE 'B%')
GO
```

## 附录一：POC实施前准备

### 第一章 OS & Linux

#### 1、检查操作系统及其版本

查看方法：

```
cat /etc/issue
```

#### 2、检查hostname,FQDN,DNS

查看方法：

```
hostname
```

```
hostname -f
```

```
cat /etc/resolv.conf
```

更改方法：

```
vim /etc/sysconfig/network
```

```
vim /etc/hosts
```

□ hostname只能是以字母和数字的组合(中间允许'-'')，不能有"," / "." / "\_" 等特殊字符

```
vim /etc/resolv.conf
```

#### 3、检查机器硬件配置

查看方法：

```
磁盘：df -h  
内存：free -g (free -m)  
网络：ethtool eth0 \ ifconfig  
CPU:cat /proc/cpuinfo
```

### 4、检查机器时间

查看方法：`date`

更改方法：`date -s '2016-3-3 9:00:00'`

### 5、需要了解的Linux命令：

- 文件／文件夹操作类：

**cd**、**mkdir**、**rm**、**cp**、**mv**、**touch**、**du -h --max-depth=1**

- 查看文本：**cat**、**less**、**tail**、**vim**、**vi**
- 查找类：**grep**、**find**
- 压缩解压缩：**tar**、**gzip**、**zip**、**unzip**
- 帮助类：**man**
- 时间类：**date**
- IO类：**\*iostat -mx 3**
- 权限相关类：**sudo -u**、**chown**、**chmod**、**chgrp**
- 端口连接类：**netstat -nlp**、**ping IP**、**telnet IP 端口**
- 查看文件占用：**lsof**
- 启停服务：**etc/init.d/mysql [start|stop|restart]**
- 网页类：**elinks http://192.168.1.210:60010**
- 挂载：**mount**、**umount**

## 第十七章 HUE安装与配置

### Hue安装

#### 环境说明

操作系统：Ubuntu 14.04

集群节点：

- Master
- slave1
- slave2

hadoop用户为：root

这里我们将hue安装在Slave2节点上

#### 安装编译hue需要的相关依赖

```
sudo apt-get install ant gcc g++ libkrb5-dev libffi-dev libmysqlclient-dev libssl-dev libsasl2-dev libsasl2-modules-gssapi-mit libsqlite3-dev libtidy-0.99-0 libxml2-dev libxslt-dev make liblucene2-dev maven python-dev python-setuptools libgmp3-dev
```

#### 下载解压并移动

到官网[下载](#)对应tar包

```
root@slave2:~$ sudo tar zxvf hue-3.10.0.tgz
root@slave2:~$ sudo cp -R hue-3.10.0 /usr/local/hue
```

#### 编译

```
root@slave2:~$ cd /usr/local/hue
root@slave2:/usr/local/hue# sudo make apps
```

### 添加**hue**用户并赋权

```
root@slave2:/usr/local/hue# sudo adduser hue
root@slave2: sudo chmod -R 775 /usr/local/hue
root@slave2: sudo chown -R hue:hue /usr/local/hue
```

### 启动**hue**

```
root@slave2:/usr/local/hue# ./build/env/bin/supervisor
```

打开slave2:8888查看到hue界面，代表hue安装成功。

下一步就是配置**hue**，使它能够管理hdfs、hive、hbase，并能使用Oozie、Pig等，将在下面的文章中给大家介绍。

## Hue配置

### 配置集群的访问权限

由于**hue**的启动用户是**hue**，所以需要为**hue**添加集群的访问权限，在各节点的/usr/local/hadoop/etc/hadoop/core-site.xml，添加如下参数：

```
<property>
    <name>hadoop.proxyuser.hue.hosts</name>
    <value>*</value>
</property>
<property>
    <name>hadoop.proxyuser.hue.groups</name>
    <value>*</value>
</property>
```

配置完，记得重启hadoop集群

## 配置hdfs

配置/usr/local/hue/desktop/conf/hue.ini

### 1) 配置hdfs的超级用户

```
# This should be the hadoop cluster admin
default_hdfs_superuser=root
```

### 2) hdfs相关配置

这里主要配置三项：fs\_defaultfs、webhdfs\_url、hadoop\_conf\_dir；

其中，webhdfs\_url默认本身就是开启的，不需要在hadoop中特别开启。

```
[[hdfs_clusters]]
# HA support by using HttpFs
[[[default]]]

# Enter the filesystem uri
fs_defaultfs=hdfs://Master:8020

# NameNode logical name.
## logical_name=

# Use WebHdfs/HttpFs as the communication mechanism.
# Domain should be the NameNode or HttpFs host.
# Default port is 14000 for HttpFs.
webhdfs_url=http://Master:50070/webhdfs/v1

# Change this if your HDFS cluster is Kerberos-secured
## security_enabled=false

# In secure mode (HTTPS), if SSL certificates from YARN Re
st APIs
# have to be verified against certificate authority
## ssl_cert_ca_verify=True

# Directory of the Hadoop configuration
hadoop_conf_dir=/usr/local/hadoop/etc/hadoop
```

## 配置yarn

配置/usr/local/hue/desktop/conf/hue.ini：

主要配置四个地方：resourcemanager\_host、resourcemanager\_api\_url、proxy\_api\_url、history\_server\_api\_url。

```
[[yarn_clusters]]  
  
[[[default]]]  
# Enter the host on which you are running the ResourceManager  
resourcemanager_host=Master  
  
# The port where the ResourceManager IPC listens on  
## resourcemanager_port=8032  
  
# Whether to submit jobs to this cluster  
submit_to=True  
  
# Resource Manager logical name (required for HA)  
## logical_name=  
  
# Change this if your YARN cluster is Kerberos-secured  
## security_enabled=false  
  
# URL of the ResourceManager API  
resourcemanager_api_url=http://Master:8088  
  
# URL of the ProxyServer API  
proxy_api_url=http://Master:8088  
  
# URL of the HistoryServer API  
history_server_api_url=http://Master:19888  
  
# URL of the Spark History Server  
## spark_history_server_url=http://localhost:18088  
  
# In secure mode (HTTPS), if SSL certificates from YARN Rest APIs  
# have to be verified against certificate authority  
## ssl_cert_ca_verify=True
```

## 配置hive

### 1) 首先配置hue.ini

主要配置两个地方：hive\_server\_host、hive\_conf\_dir。

```
[beeswax]

# Host where HiveServer2 is running.
# If Kerberos security is enabled, use fully-qualified domain
name (FQDN).
hive_server_host=Master

# Port where HiveServer2 Thrift server runs on.
## hive_server_port=10000

# Hive configuration directory, where hive-site.xml is located
hive_conf_dir=/usr/local/hive/conf

# Timeout in seconds for thrift calls to Hive service
## server_conn_timeout=120
```

### 2) 启动hive2

```
root@Master:/usr/local/hive/bin# hive --service hiveserver2 &
```

## 配置hbase

### 1) 首先配置hue.ini

主要配置两个地方：hbase\_clusters、hbase\_conf\_dir。

```
[hbase]
# Comma-separated list of HBase Thrift servers for clusters in
the format of '(name|host:port)'.
# Use full hostname with security.
# If using Kerberos we assume GSSAPI SASL, not PLAIN.
hbase_clusters=(Cluster|Master:9090)

# HBase configuration directory, where hbase-site.xml is located.
hbase_conf_dir=/usr/local/hbase/conf

# Hard limit of rows or columns per row fetched before truncating.
## truncate_limit = 500

# 'buffered' is the default of the HBase Thrift Server and supports security.
# 'framed' can be used to chunk up responses,
# which is useful when used in conjunction with the nonblocking server in Thrift.
## thrift_transport=buffered
```

## 2) 启动thrift

```
root@Master:/usr/local/hbase/bin# hbase-daemon.sh start thrift
```

特别注意：这里的thrift必须是1，而不是thrift2

## 启动hue

```
root@slave2:/usr/local/hue# ./build/env/bin/supervisor
```

打开slave2:8888/about/查看到hue界面，如果页面中没有报hdfs、yarn、hbase、hive相关的警告则代表配置成功，之后就能在hue中使用相关的功能。

但是，我们可能会看到如下警告：

SQLITE_NOT_FOR_PRODUCTION_USE	SQLite is only recommended for small development environments with a few users.
Impala	No available Impalad to send queries to.
Oozie Editor/Dashboard	The app won't work without a running Oozie server
Pig Editor	The app won't work without a running Oozie server
Spark	The app won't work without a running Livy Spark Server

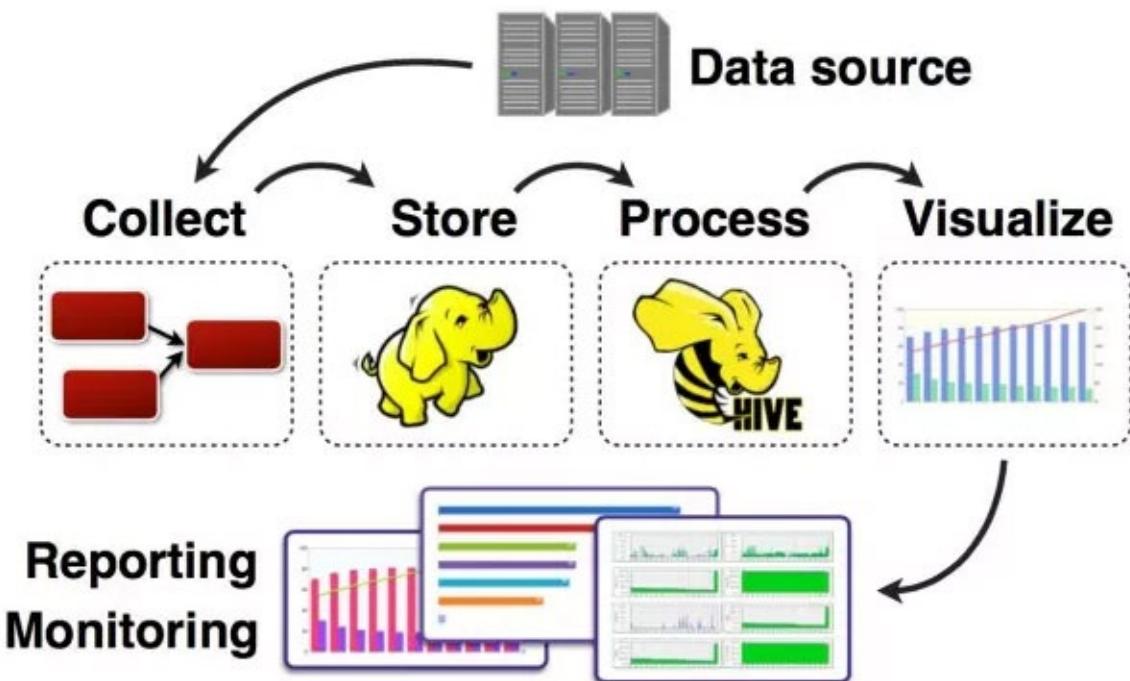
那是由于我们没有安装和配置相应功能，该块内容，将在后续文章中补充。

# 第十八章 数据采集与爬虫

## 一 数据采集概念

任何完整的大数据平台，一般包括以下几个过程：

- 数据采集
- 数据存储
- 数据处理
- 数据展现（可视化，报表和监控）



其中，数据采集是所有数据系统必不可少的，随着大数据越来越被重视，数据采集的挑战也变的尤为突出。这其中包括：

- 数据源多种多样
- 数据量大，变化快
- 如何保证数据采集的可靠性的性能
- 如何避免重复数据
- 如何保证数据的质量

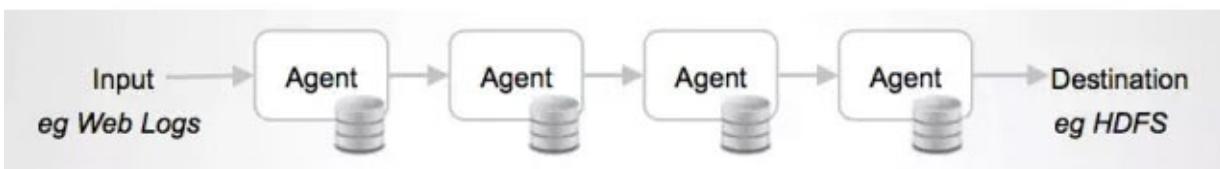
我们今天就来看看当前可用的六款数据采集的产品，重点关注它们是如何做到高可靠，高性能和高扩展。

# 1 Apache Flume

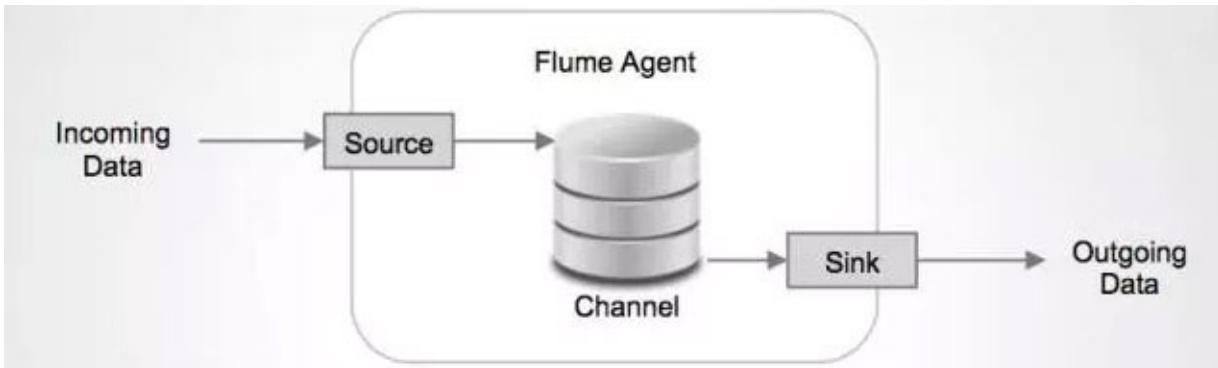
官网：<https://flume.apache.org/>

Flume 是Apache旗下的一款开源、高可靠、高扩展、容易管理、支持客户扩展的数据采集系统。Flume 使用JRuby来构建，所以依赖Java运行环境。

Flume 最初是由Cloudera的工程师设计用于合并日志数据的系统，后来逐渐发展用于处理流数据事件。



Flume 设计成一个分布式的管道架构，可以看作在数据源和目的地之间有一个Agent的网络，支持数据路由。



每一个agent都由Source，Channel和Sink组成。

## Source

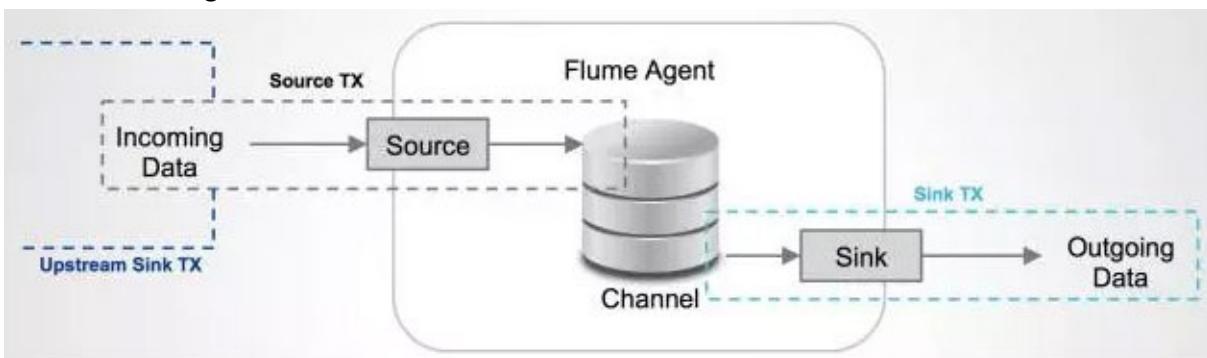
Source 负责接收输入数据，并将数据写入管道。Flume 的Source 支持HTTP，JMS，RPC，NetCat，Exec，Spooling Directory。其中 Spooling 支持监视一个目录或者文件，解析其中新生成的事件。

## Channel

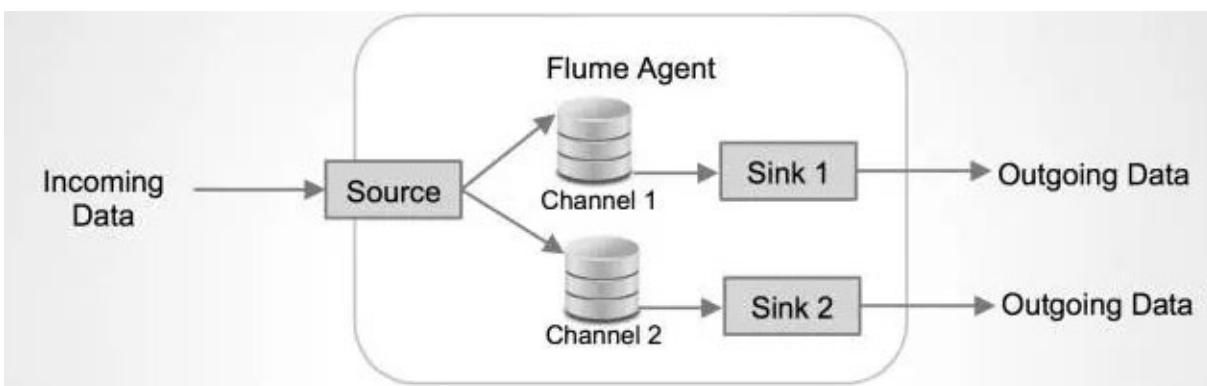
Channel 存储，缓存从source到Sink的中间数据。可使用不同的配置来做 Channel，例如内存，文件，JDBC等。使用内存性能高但不持久，有可能丢数据。使用文件更可靠，但性能不如内存。

## Sink

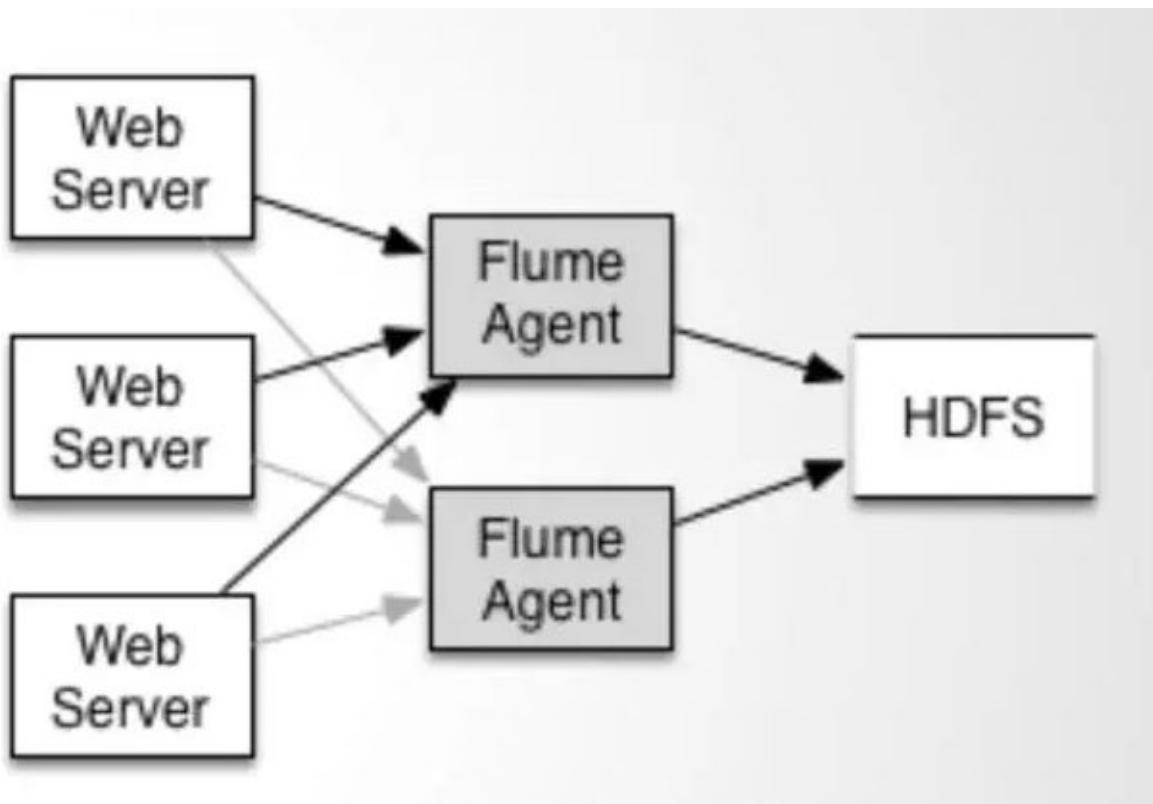
Sink负责从管道中读出数据并发给下一个Agent或者最终的目的地。Sink支持的不同目的地种类包括：HDFS，HBASE，Solr，ElasticSearch，File，Logger或者其它的Flume Agent。



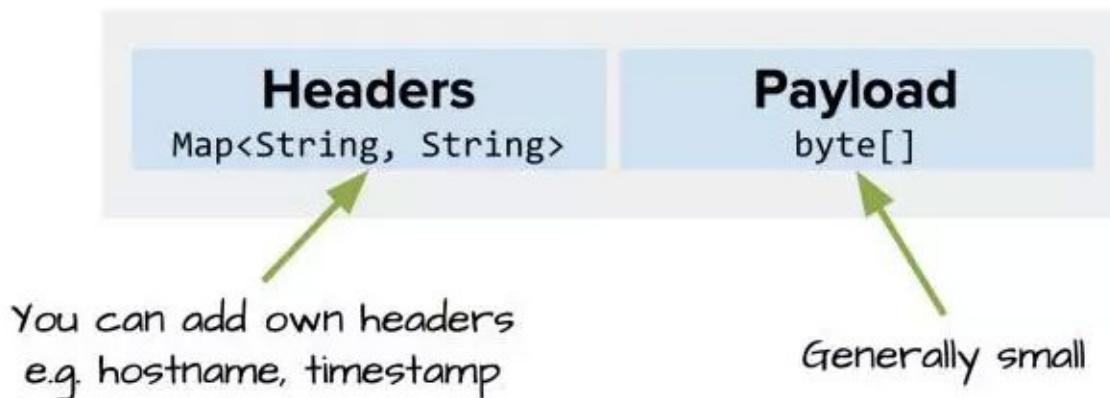
Flume在source和sink端都使用了transaction机制保证在数据传输中没有数据丢失。



Source上的数据可以复制到不同的通道上。每一个Channel也可以连接不同数量的Sink。这样连接不同配置的Agent就可以组成一个复杂的数据收集网络。通过对agent的配置，可以组成一个路由复杂的数据传输网络。



配置如上图所示的agent结构，Flume支持设置sink的Failover和Load Balance，这样就可以保证即使有一个agent失效的情况下，整个系统仍能正常收集数据。



Flume中传输的内容定义为事件（Event），事件由Headers（包含元数据，Meta Data）和Payload组成。

Flume提供SDK，可以支持用户定制开发：

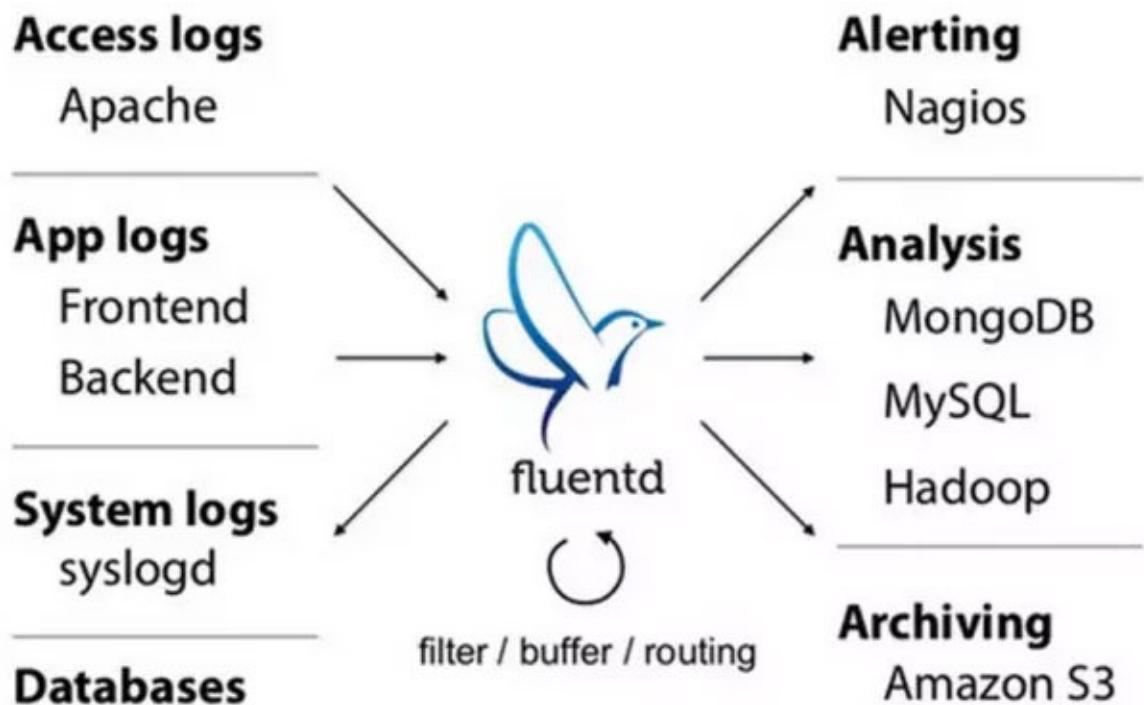
Flume客户端负责在事件产生的源头把事件发送给Flume的Agent。客户端通常和产生数据源的应用在同一个进程空间。常见的Flume客户端有Avro，log4J，syslog和HTTP Post。另外ExecSource支持指定一个本地进程的输出作为Flume的输入。当然很有可能，以上的这些客户端都不能满足需求，用户可以定制的客户端，和已有的Flume的Source进行通信，或者定制实现一种新的Source类型。

同时，用户可以使用Flume的SDK定制Source和Sink。似乎不支持定制的Channel。

## 2、Fluentd

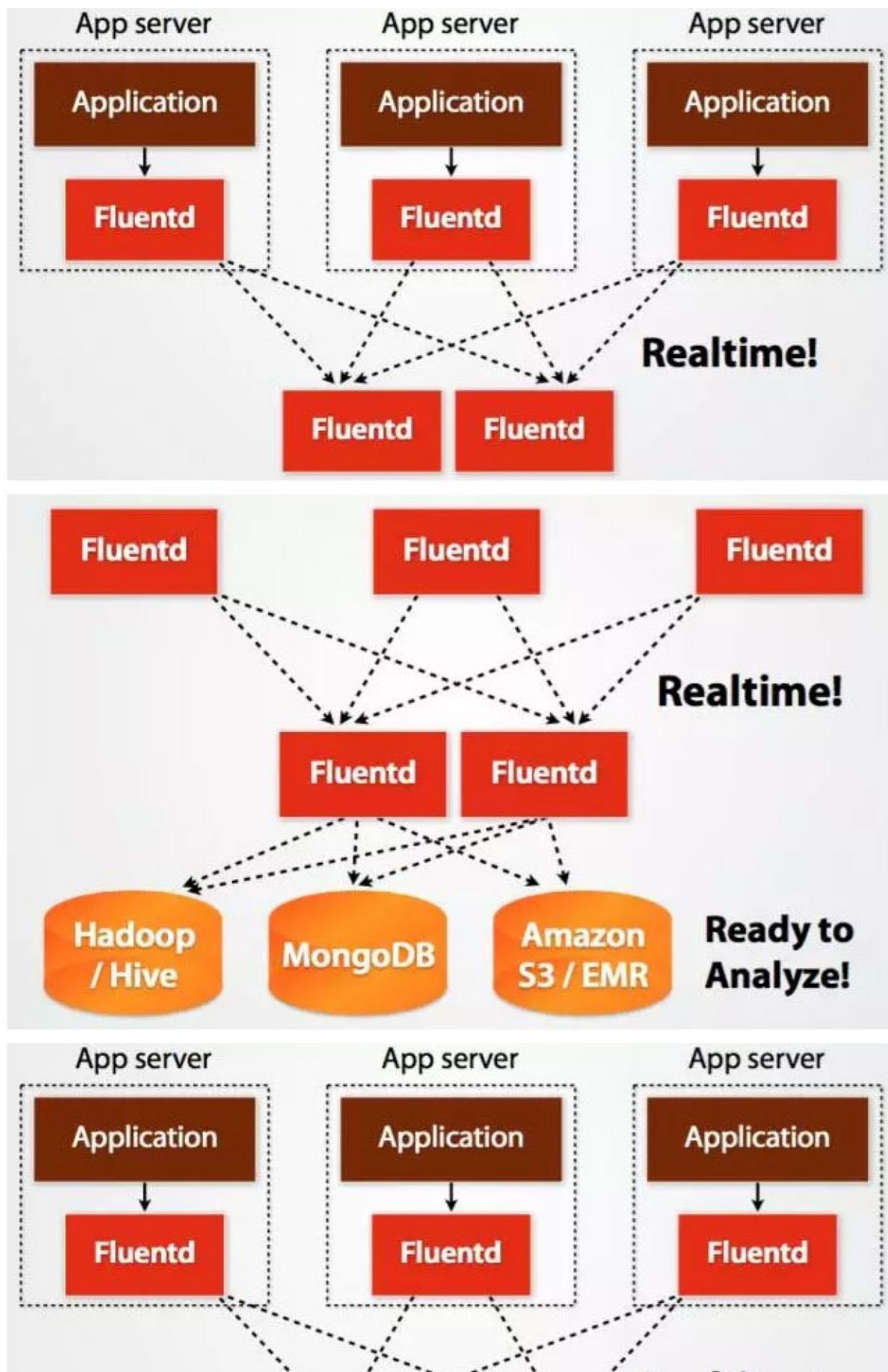
官网：<http://docs.fluentd.org/articles/quickstart>

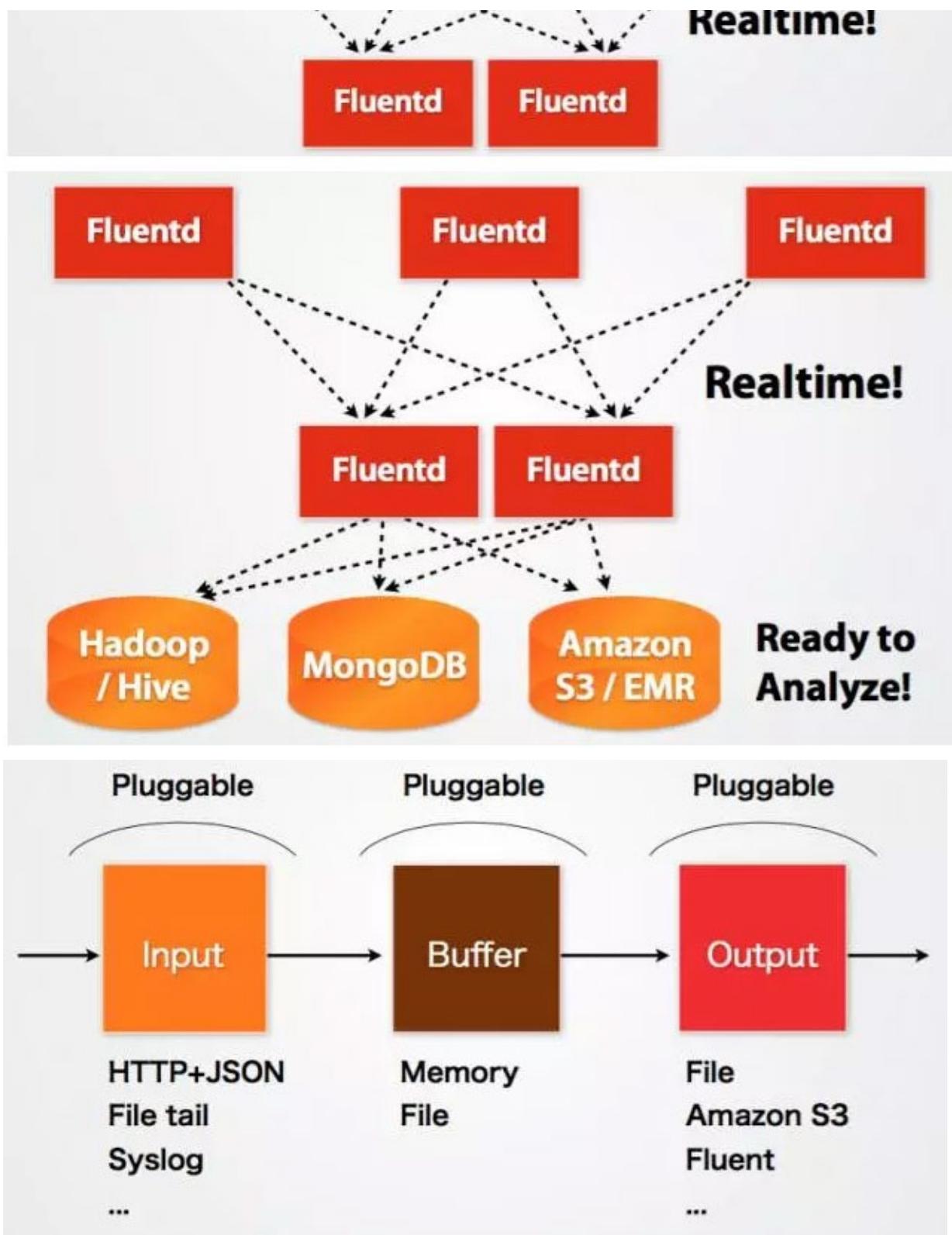
Fluentd是另一个开源的数据收集框架。Fluentd使用C/Ruby开发，使用JSON文件来统一日志数据。它的可插拔架构，支持各种不同种类和格式的数据源和数据输出。最后它也同时提供了高可靠和很好的扩展性。Treasure Data, Inc 对该产品提供支持和维护。





Fluentd的部署和Flume非常相似：





Fluentd的Input／Buffer／Output非常类似于Flume的Source／Channel／Sink。

## Input

Input负责接收数据或者主动抓取数据。支持syslog，http，file tail等。

## Buffer

Buffer负责数据获取的性能和可靠性，也有文件或内存等不同类型的Buffer可以配置。

## Output

Output负责输出数据到目的地例如文件，AWS S3或者其它的Fluentd。

Fluentd的配置非常方便，如下图：

```
# read logs from a file          # forward other logs to servers
<source>                         # (load-balancing + fail-over)
  type tail                         <match **>
  path /var/log/httpd.log           type forward
  format apache                      <server>
  tag apache.access                  host 192.168.0.11
</source>                           weight 20
                                      </server>
# save access logs to MongoDB      <server>
<match apache.access>              host 192.168.0.12
  type mongo                        weight 60
  host 127.0.0.1                   </server>
</match>                            </match>
```

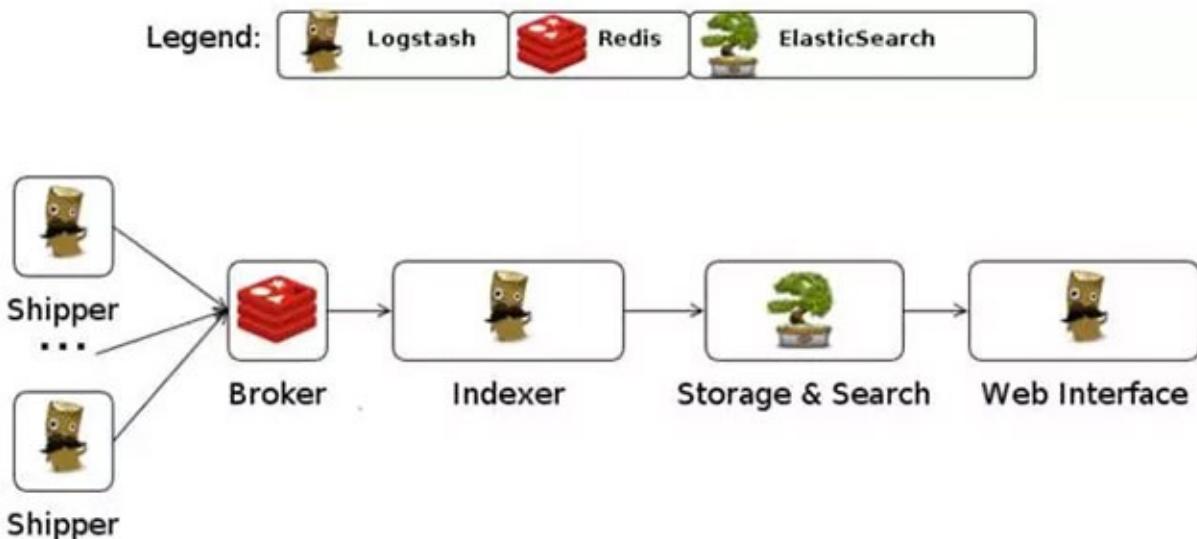
## 3 Logstash

官方网站：<https://github.com/elastic/logstash>

Logstash是著名的开源数据栈ELK（ElasticSearch, Logstash, Kibana）中的那个L。

Logstash用JRuby开发，所有运行时依赖JVM。

Logstash的部署架构如下图，当然这只是一种部署的选项。



一个典型的Logstash的配置如下，包括了Input，filter的Output的设置。

```

1 input {
2   file {
3     type => "apache-access"
4     path => "/var/log/apache2/other_vhosts_access.log"
5   }
6
7   file {
8     type => "apache-error"
9     path => "/var/log/apache2/error.log"
10  }
11 }
12
13 filter {
14   grok {
15     match => { "message" => "%{COMBINEDAPACHELOG}" }
16   }
17   date {
18     match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
19   }
20 }
21
22 output {
23   stdout { }
24   redis {
25     host => "192.168.1.200"
26     data_type => "list"
27     key => "logstash"
28   }
29 }
```

几乎在大部分的情况下ELK作为一个栈是被同时使用的。所有当你的数据系统使用ElasticSearch的情况下，logstash是首选。

## 4 数据采集总结

我们简单讨论了几种流行的数据收集平台，它们大都提供高可靠和高扩展的数据收集。大多平台都抽象出了输入，输出和中间的缓冲的架构。利用分布式的网络连接，大多数平台都能实现一定程度的扩展性和高可靠性。

其中Flume，Fluentd是两个被使用较多的产品。如果你用ElasticSearch，Logstash也许是首选，因为ELK栈提供了很好的集成。Chukwa和Scribe由于项目的不活跃，不推荐使用。

Splunk作为一个优秀的商业产品，它的数据采集还存在一定的限制，相信Splunk很快会开发出更好的数据收集的解决方案。

## 二 爬虫技术方案选择

开发网络爬虫应该选择Nutch、Crawler4j、WebMagic、scrapy、WebCollector还是其他的？这里按照我的经验随便扯淡一下：上面说的爬虫，基本可以分3类：

1. 分布式爬虫：Nutch
2. JAVA单机爬虫：Crawler4j、WebMagic、WebCollector
3. 非JAVA单机爬虫：scrapy

## 1 分布式爬虫

爬虫使用分布式，主要是解决两个问题：

1. 海量URL管理
2. 网速

现在比较流行的分布式爬虫，是Apache的Nutch。但是对于大多数用户来说，Nutch是这几类爬虫里，最不好的选择，理由如下：

1. Nutch是为搜索引擎设计的爬虫，大多数用户是需要一个做精准数据爬取（精抽取）的爬虫。Nutch运行的一套流程里，有三分之二是为了搜索引擎而设计的。对精抽取没有太大的意义。也就是说，用Nutch做数据抽取，会浪费很多的时间在不必要的计算上。而且如果你试图通过对Nutch进行二次开发，来使得它适用于精抽取的业务，基本上就要破坏Nutch的框架，把Nutch改的面目全非，有修改Nutch的能力，真的不如自己重新写一个分布式爬虫框架了。

2. Nutch依赖hadoop运行，hadoop本身会消耗很多的时间。如果集群机器数量较少，爬取速度反而不如单机爬虫快。
3. Nutch虽然有一套插件机制，而且作为亮点宣传。可以看到一些开源的Nutch插件，提供精抽取的功能。但是开发过Nutch插件的人都知道，Nutch的插件系统有多蹩脚。利用反射的机制来加载和调用插件，使得程序的编写和调试都变得异常困难，更别说在上面开发一套复杂的精抽取系统了。而且Nutch并没有为精抽取提供相应的插件挂载点。Nutch的插件有只有五六个挂载点，而这五六个挂载点都是为了搜索引擎服务的，并没有为精抽取提供挂载点。大多数Nutch的精抽取插件，都是挂载在“页面解析”(parser)这个挂载点的，这个挂载点其实是为了了解析链接（为后续爬取提供URL），以及为搜索引擎提供一些易抽取的网页信息(网页的meta信息、text文本)。
4. 用Nutch进行爬虫的二次开发，爬虫的编写和调试所需的时间，往往是单机爬虫所需的十倍时间不止。了解Nutch源码的学习成本很高，何况是要让一个团队的人都读懂Nutch源码。调试过程中会出现除程序本身之外的各种问题(hadoop的问题、hbase的问题)。
5. 很多人说Nutch2有gora，可以持久化数据到avro文件、hbase、mysql等。很多人其实理解错了，这里说的持久化数据，是指将URL信息（URL管理所需要的数据）存放到avro、hbase、mysql。并不是你要抽取的结构化数据。其实对大多数人来说，URL信息存在哪里无所谓。
6. Nutch2的版本目前并不适合开发。官方现在稳定的Nutch版本是nutch2.2.1，但是这个版本绑定了gora-0.3。如果想用hbase配合nutch（大多数人用nutch2就是为了用hbase），只能使用0.90版本左右的hbase，相应的就要将hadoop版本降到hadoop 0.2左右。而且nutch2的官方教程比较有误导作用，Nutch2的教程有两个，分别是Nutch1.x和Nutch2.x，这个Nutch2.x官网上写的是可以支持到hbase 0.94。但是实际上，这个Nutch2.x的意思是Nutch2.3之前、Nutch2.2.1之后的一个版本，这个版本在官方的SVN中不断更新。而且非常不稳定（一直在修改）。所以，如果你不是要做搜索引擎，尽量不要选择Nutch作为爬虫。有些团队就喜欢跟风，非要选择Nutch来开发精抽取的爬虫，其实是冲着Nutch的名气（Nutch作者是Doug Cutting），当然最后的结果往往是项目延期完成。

如果你是要做搜索引擎，Nutch1.x是一个非常好的选择。Nutch1.x和solr或者es配合，就可以构成一套非常强大的搜索引擎了。如果非要用Nutch2的话，建议等到Nutch2.3发布再看。目前的Nutch2是一个非常不稳定的版本。

## 2 JAVA单机爬虫

其实开源网络爬虫（框架）的开发非常简单，难题和复杂的问题都被以前的人解决了（比如DOM树解析和定位、字符集检测、海量URL去重），可以说是毫无技术含量。包括Nutch，其实Nutch的技术难点是开发hadoop，本身代码非常简单。网络爬虫从某种意义来说，类似遍历本机的文件，查找文件中的信息。没有任何难度可言。之所以选择开源爬虫框架，就是为了省事。比如爬虫的URL管理、线程池之类的模块，谁都能做，但是要做稳定也是需要一段时间的调试和修改的。

对于爬虫的功能来说。用户比较关心的问题往往是：

1. 爬虫支持多线程么、爬虫能用代理么、爬虫会爬取重复数据么、爬虫能爬取JS生成的信息么？

不支持多线程、不支持代理、不能过滤重复URL的，那都不叫开源爬虫，那叫循环执行http请求。能不能爬js生成的信息和爬虫本身没有太大关系。爬虫主要是负责遍历网站和下载页面。爬js生成的信息和网页信息抽取模块有关，往往需要通过模拟浏览器(htmlunit,selenium)来完成。这些模拟浏览器，往往需要耗费很多的时间来处理一个页面。所以一种策略就是，使用这些爬虫来遍历网站，遇到需要解析的页面，就将网页的相关信息提交给模拟浏览器，来完成JS生成信息的抽取。

1. 爬虫可以爬取ajax信息么？

网页上有一些异步加载的数据，爬取这些数据有两种方法：使用模拟浏览器（问题1中描述过了），或者分析ajax的http请求，自己生成ajax请求的url，获取返回的数据。如果是自己生成ajax请求，使用开源爬虫的意义在哪里？其实是要用开源爬虫的线程池和URL管理功能（比如断点爬取）。

如果我已经可以生成我所需要的ajax请求（列表），如何用这些爬虫来对这些请求进行爬取？爬虫往往都是设计成广度遍历或者深度遍历的模式，去遍历静态或者动态页面。爬取ajax信息属于deep web（深网）的范畴，虽然大多数爬虫都不直接支持。但是也可以通过一些方法来完成。比如WebCollector使用广度遍历来遍历网站。爬虫的第一轮爬取就是爬取种子集合(seeds)中的所有url。简单来说，就是将生成的ajax请求作为种子，放入爬虫。用爬虫对这些种子，进行深度为1的广度遍历（默认就是广度遍历）。

1. 爬虫怎么爬取要登陆的网站？

这些开源爬虫都支持在爬取时指定cookies，模拟登陆主要是靠cookies。至于cookies怎么获取，不是爬虫管的事情。你可以手动获取、用http请求模拟登陆或者用模拟浏览器自动登陆获取cookie。

### 1. 爬虫怎么抽取网页的信息？

开源爬虫一般都会集成网页抽取工具。主要支持两种规范：CSS SELECTOR和XPATH。至于哪个好，这里不评价。

### 1. 爬虫怎么保存网页的信息？

有一些爬虫，自带一个模块负责持久化。比如webmagic，有一个模块叫pipeline。通过简单地配置，可以将爬虫抽取到的信息，持久化到文件、数据库等。还有一些爬虫，并没有直接给用户提供数据持久化的模块。比如crawler4j和webcollector。让用户自己在网页处理模块中添加提交数据库的操作。至于使用pipeline这种模块好不好，就和操作数据库使用ORM好不好这个问题类似，取决于你的业务。

### 1. 爬虫被网站封了怎么办？

爬虫被网站封了，一般用多代理（随机代理）就可以解决。但是这些开源爬虫一般没有直接支持随机代理的切换。所以用户往往都需要自己将获取的代理，放到一个全局数组中，自己写一个代理随机获取（从数组中）的代码。

### 1. 网页可以调用爬虫么？

爬虫的调用是在Web的服务端调用的，平时怎么用就怎么用，这些爬虫都可以使用。

### 1. 爬虫速度怎么样？

单机开源爬虫的速度，基本都可以讲本机的网速用到极限。爬虫的速度慢，往往是因为用户把线程数开少了、网速慢，或者在数据持久化时，和数据库的交互速度慢。而这些东西，往往都是用户的机器和二次开发的代码决定的。这些开源爬虫的速度，都很可以。

### 1. 明明代码写对了，爬不到数据，是不是爬虫有问题，换个爬虫能解决么？

如果代码写对了，又爬不到数据，换其他爬虫也是一样爬不到。遇到这种情况，要么是网站把你封了，要么是你爬的数据是javascript生成的。爬不到数据通过换爬虫是不能解决的。

### 1. 哪个爬虫可以判断网站是否爬完、那个爬虫可以根据主题进行爬取？

爬虫无法判断网站是否爬完，只能尽可能覆盖。

至于根据主题爬取，爬虫之后把内容爬下来才知道是什么主题。所以一般都是整个爬下来，然后再去筛选内容。如果嫌爬的太泛，可以通过限制URL正则等方式，来缩小一下范围。

### 1. 哪个爬虫的设计模式和构架比较好？

设计模式纯属扯淡。说软件设计模式好的，都是软件开发完，然后总结出几个设计模式。设计模式对软件开发没有指导性作用。用设计模式来设计爬虫，只会使得爬虫的设计更加臃肿。

至于构架，开源爬虫目前主要是细节的数据结构的设计，比如爬取线程池、任务队列，这些大家都能控制好。爬虫的业务太简单，谈不上什么构架。

所以对于JAVA开源爬虫，我觉得，随便找一个用的顺手的就可以。如果业务复杂，拿哪个爬虫来，都是要经过复杂的二次开发，才可以满足需求。

## 3 非JAVA单机爬虫

在非JAVA语言编写的爬虫中，有很多优秀的爬虫。这里单独提取出来作为一类，并不是针对爬虫本身的质量进行讨论，而是针对larbin、scrapy这类爬虫，对开发成本的影响。

先说python爬虫，python可以用30行代码，完成JAVA 50行代码干的任务。python写代码的确快，但是在调试代码的阶段，python代码的调试往往耗费远远多于编码阶段省下的时间。使用python开发，要保证程序的正确性和稳定性，就需要写更多的测试模块。当然如果爬取规模不大、爬取业务不复杂，使用scrapy这种爬虫也是蛮不错的，可以轻松完成爬取任务。

对于C++爬虫来说，学习成本会比较大。而且不能只计算一个人的学习成本，如果软件需要团队开发或者交接，那就是很多人的学习成本了。软件的调试也不是那么容易。

还有一些ruby、php的爬虫，这里不多评价。的确有一些非常小型的数据采集任务，用ruby或者php很方便。但是选择这些语言的开源爬虫，一方面要调研一下相关的生态圈，还有就是，这些开源爬虫可能会出一些你搜不到的BUG（用的人少、资料也少）

### 三 基于Python的爬虫库

#### 1 通用

1. `urllib` - 网络库(`stdlib`)。
2. `requests` - 网络库。
3. `grab` – 网络库（基于`pycurl`）。
4. `pycurl` – 网络库（绑定`libcurl`）。
5. `urllib3` – Python HTTP库，安全连接池、支持文件post、可用性高。
6. `httpplib2` – 网络库。
7. `RoboBrowser` – 一个简单的、极具Python风格的Python库，无需独立的浏览器即可浏览网页。
8. `MechanicalSoup` -一个与网站自动交互Python库。
9. `mechanize` -有状态、可编程的Web浏览库。
10. `socket` – 底层网络接口(`stdlib`)。
11. `Unirest for Python` – Unirest是一套可用于多种语言的轻量级的HTTP库。
12. `hyper` – Python的HTTP/2客户端。
13. `PySocks` – SocksPy更新并积极维护的版本，包括错误修复和一些其他的特征。作为`socket`模块的直接替换。

#### 2 异步

1. `treq` – 类似于`requests`的API（基于`twisted`）。
2. `aiohttp` – `asyncio`的HTTP客户端/服务器(PEP-3156)。

#### 3 功能齐全的爬虫

1. `grab` – 网络爬虫框架（基于`pycurl/multicur`）。
2. `scrapy` – 网络爬虫框架（基于`twisted`），不支持Python3。
3. `pyspider` – 一个强大的爬虫系统。
4. `cola` – 一个分布式爬虫框架。

#### 4 其他

1. `portia` – 基于Scrapy的可视化爬虫。

2. `restkit` – Python的HTTP资源工具包。它可以帮助你轻松地访问HTTP资源，并围绕它建立的对象。
3. `demiurge` – 基于`PyQuery`的爬虫微框架。

## 5 HTML/XML解析器

1. `Ixml` – C语言编写高效HTML/ XML处理库。支持XPath。
2. `cssselect` – 解析DOM树和CSS选择器。
3. `pyquery` – 解析DOM树和jQuery选择器。
4. `BeautifulSoup` – 低效HTML/ XML处理库，纯Python实现。
5. `html5lib` – 根据WHATWG规范生成HTML/
6. XML文档的DOM。该规范被用在现在所有的浏览器上。
7. `feedparser` – 解析RSS/ATOM feeds。
8. `MarkupSafe` – 为XML/HTML/XHTML提供了安全转义的字符串。
9. `xmltodict` – 一个可以让你在处理XML时感觉像在处理JSON一样的Python模块。
10. `xhtml2pdf` – 将HTML/CSS转换为PDF。
11. `untangle` – 轻松实现将XML文件转换为Python对象。

## 6 清理

1. `Bleach` – 清理HTML（需要`html5lib`）。
2. `sanitize` – 为混乱的数据世界带来清明。

## 7 解析和操作简单文本的库。

1. `difflib` – (Python标准库)帮助进行差异化比较。
2. `Levenshtein` – 快速计算Levenshtein距离和字符串相似度。
3. `fuzzywuzzy` – 模糊字符串匹配。
4. `esmre` – 正则表达式加速器。
5. `ftfy` – 自动整理Unicode文本，减少碎片化。

## 8 转换

1. `unidecode` – 将Unicode文本转为ASCII。

## 9 字符编码

1. `uniout` – 打印可读字符，而不是被转义的字符串。
2. `chardet` – 兼容 Python 的 2/3 的字符编码器。
3. `xpinyin` – 一个将中国汉字转为拼音的库。
4. `pangu.py` – 格式化文本中 CJK 和字母数字的间距。

## 10 Slug化

1. `awesome-slugify` – 一个可以保留 unicode 的 Python `slugify` 库。
2. `python-slugify` – 一个可以将 Unicode 转为 ASCII 的 Python `slugify` 库。
3. `unicode-slugify` – 一个可以将生成 Unicode slugs 的工具。
4. `pytils` – 处理俄语字符串的简单工具（包括 `pytils.translit.slugify`）。

## 11 通用解析器

1. `PLY` – `lex` 和 `yacc` 解析工具的 Python 实现。
2. `pyparsing` – 一个通用框架的生成语法分析器。

## 12 人的名字

1. `python-nameparser` – 解析人的名字的组件。

## 13 电话号码

1. `phonenumbers` – 解析，格式化，存储和验证国际电话号码。

## 14 用户代理字符串

1. `python-user-agents` – 浏览器用户代理的解析器。
2. `HTTP Agent Parser` – Python 的 HTTP 代理分析器。

## 15 解析和处理特定文本格式的库。

1. `tablib` – 一个把数据导出为 XLS、CSV、JSON、YAML 等格式的模块。
2. `textract` – 从各种文件中提取文本，比如 Word、PowerPoint、PDF 等。

3. `messytables` – 解析混乱的表格数据的工具。
4. `rows` – 一个常用数据接口，支持的格式很多（目前支持CSV，HTML，XLS，TXT – 将来还会提供更多！）。

## 16 Office

1. `python-docx` – 读取，查询和修改的Microsoft Word2007/2008的`docx`文件。
2. `xlwt / xlrd` – 从Excel文件读取写入数据和格式信息。
3. `XlsxWriter` – 一个创建Excel`.xlsx`文件的Python模块。
4. `xlwings` – 一个BSD许可的库，可以很容易地在Excel中调用Python，反之亦然。
5. `openpyxl` – 一个用于读取和写入的Excel2010 XLSX/ XLSM/ xltx/ XLTM文件的库。
6. `Marmir` – 提取Python数据结构并将其转换为电子表格。

## 17 PDF

1. `PDFMiner` – 一个从PDF文档中提取信息的工具。
2. `PyPDF2` – 一个能够分割、合并和转换PDF页面的库。
3. `ReportLab` – 允许快速创建丰富的PDF文档。
4. `pdftables` – 直接从PDF文件中提取表格。

## 18 Markdown

1. `Python-Markdown` – 一个用Python实现的John Gruber的Markdown。
2. `Mistune` – 速度最快，功能全面的Markdown纯Python解析器。
3. `markdown2` – 一个完全用Python实现的快速的Markdown。

## 19 YAML

1. `PyYAML` – 一个Python的YAML解析器。

## 20 CSS

1. `cssutils` – 一个Python的CSS库。

## 21 ATOM/RSS

1. `feedparser` – 通用的feed解析器。

## 22 SQL

1. `sqlparse` – 一个非验证的SQL语句分析器。

## 23 HTTP

1. `http-parser` – C语言实现的HTTP请求/响应消息解析器。

## 24 微格式

1. `opengraph` – 一个用来解析Open Graph协议标签的Python模块。

## 25 可移植的执行体

1. `pefile` – 一个多平台的用于解析和处理可移植执行体（即PE）文件的模块。

## 26 PSD

1. `psd-tools` – 将Adobe Photoshop PSD（即PE）文件读取到Python数据结构。

## 27 自然语言处理

1. `NLTK` - 编写Python程序来处理人类语言数据的最好平台。
2. `Pattern` – Python的网络挖掘模块。他有自然语言处理工具，机器学习以及其它。
3. `TextBlob` – 为深入自然语言处理任务提供了一致的API。是基于NLTK以及Pattern的巨人之肩上发展的。
4. `jieba` – 中文分词工具。
5. `SnowNLP` – 中文文本处理库。
6. `loso` – 另一个中文分词库。
7. `genius` – 基于条件随机域的中文分词。
8. `langid.py` – 独立的语言识别系统。

9. Korean – 一个韩文形态库。
10. pymorphy2 – 俄语形态分析器（词性标注+词形变化引擎）。
11. PyPLN – 用Python编写的分布式自然语言处理通道。这个项目的目标是创建一种简单的方法使用NLTK通过网络接口处理大语言库。

## 28 浏览器自动化与仿真

1. selenium – 自动化真正的浏览器（Chrome浏览器，火狐浏览器，Opera浏览器，IE浏览器）。
2. Ghost.py – 对PyQt的webkit的封装（需要PyQT）。
3. Spynner – 对PyQt的webkit的封装（需要PyQT）。
4. Splinter – 通用API浏览器模拟器（selenium web驱动，Django客户端，Zope）。

## 29 多重处理

1. threading – Python标准库的线程运行。对于I/O密集型任务很有效。对于CPU绑定的任务没用，因为python GIL。
2. multiprocessing – 标准的Python库运行多进程。
3. celery – 基于分布式消息传递的异步任务队列/作业队列。concurrent-futures – concurrent-futures 模块为调用异步执行提供了一个高层次的接口。

## 30 异步网络编程库

1. asyncio – （在Python 3.4 +版本以上的 Python标准库）异步I/O，时间循环，协同程序和任务。
2. Twisted – 基于事件驱动的网络引擎框架。
3. Tornado – 一个网络框架和异步网络库。
4. pulsar – Python事件驱动的并发框架。
5. diesel – Python的基于绿色事件的I/O框架。
6. gevent – 一个使用greenlet 的基于协程的Python网络库。
7. eventlet – 有WSGI支持的异步框架。
8. Tomorrow – 异步代码的奇妙的修饰语法。队列
9. celery – 基于分布式消息传递的异步任务队列/作业队列。
10. huey – 小型多线程任务队列。

11. mrq – Mr. Queue – 使用redis & Gevent 的Python分布式工作任务队列。
12. RQ – 基于Redis的轻量级任务队列管理器。
13. simpleq – 一个简单的，可无限扩展，基于Amazon SQS的队列。
14. python-gearman – Gearman的Python API。

## 31 云计算

1. picloud – 云端执行Python代码。
2. dominoup.com – 云端执行R，Python和matlab代码。

## 32 电子邮件

1. flanker – 电子邮件地址和Mime解析库。
2. Talon – Mailgun库用于提取消息的报价和签名。

## 33 网址和网络地址操作

1. furl – 一个小的Python库，使得操纵URL简单化。
2. purl – 一个简单的不可改变的URL以及一个干净的用于调试和操作的API。
3. urllib.parse – 用于打破统一资源定位器（URL）的字符串在组件（寻址方案，网络位置，路径等）之间的隔断，为了结合组件到一个URL字符串，并将“相对URL”转化为一个绝对URL，称之为“基本URL”。
4. tldextract – 从URL的注册域和子域中准确分离TLD，使用公共后缀列表。
5. netaddr – 用于显示和操纵网络地址的Python库。

## 34 提取网页内容的库。

1. newspaper – 用Python进行新闻提取、文章提取和内容策展。
2. html2text – 将HTML转为Markdown格式文本。
3. python-goose – HTML内容/文章提取器。
4. lassie – 人性化的网页内容检索工具
5. micawber – 一个从网址中提取丰富内容的小库。
6. sumy -一个自动汇总文本文件和HTML网页的模块
7. Haul – 一个可扩展的图像爬虫。
8. python-readability – arc90 readability工具的快速Python接口。
9. scrapely – 从HTML网页中提取结构化数据的库。给出了一些Web页面和数据

提取的示例，scrapely为所有类似的网页构建一个分析器。

## 35 视频

1. youtube-dl – 一个从YouTube下载视频的小命令行程序。
2. you-get – Python3的YouTube、优酷/ Niconico视频下载器。

## 36 维基

1. WikiTeam – 下载和保存wikis的工具。

## 37 用于**WebSocket**的库。

1. Crossbar – 开源的应用消息传递路由器（Python实现的用于Autobahn的WebSocket和WAMP）。
2. AutobahnPython – 提供了WebSocket协议和WAMP协议的Python实现并且开源。
3. WebSocket-for-Python – Python 2和3以及PyPy的WebSocket客户端和服务器库。

## 38 DNS解析

1. dnsyo – 在全球超过1500个的DNS服务器上检查你的DNS。
2. pycares – c-ares的接口。c-ares是进行DNS请求和异步名称决议的C语言库。

## 39 计算机视觉

1. SimpleCV – 用于照相机、图像处理、特征提取、格式转换的简介，可读性强的接口（基于OpenCV）。
2. mahotas – 快速计算机图像处理算法（完全使用 C++ 实现），完全基于 numpy 的数组作为它的数据类型。

## 40 代理服务器

1. shadowsocks – 一个快速隧道代理，可帮你穿透防火墙（支持TCP和UDP），

TFO，多用户和平滑重启，目的IP黑名单）。

2. tproxy – tproxy是一个简单的TCP路由代理（第7层），基于Gevent，用Python进行配置。其他Python工具列表

## 41 awesome-python

1. pycrumbs
2. python-github-projects
3. python\_reference
4. pythonidae

## 第十九章 Hadoop相关资源

### Hadoop查询引擎

#### 一、Phoenix

贡献者：：Salesforce

简介：这是一个Java中间层，可以让开发者在Apache HBase上执行SQL查询。Phoenix完全使用Java编写，代码位于GitHub上，并且提供了一个客户端可嵌入的JDBC驱动。

Phoenix查询引擎会将SQL查询转换为一个或多个HBase scan，并编排执行以生成标准的JDBC结果集。直接使用HBase API、协同处理器与自定义过滤器，对于简单查询来说，其性能量级是毫秒，对于百万级别的行数来说，其性能量级是秒。

Phoenix最值得关注的一些特性有：

①嵌入式的JDBC驱动，实现了大部分的java.sql接口，包括元数据API ②可以通过多部行键或是键/值单元对列进行建模 ③完善的查询支持，可以使用多个谓词以及优化的扫描键 ④DDL支持：通过CREATE TABLE、DROP TABLE及ALTER TABLE来添加/删除列 ⑤版本化的模式仓库：当写入数据时，快照查询会使用恰当的模式 ⑥DML支持：用于逐行插入的UPSERT VALUES、用于相同或不同表之间大量数据传输的UPSERT ⑦SELECT、用于删除行的DELETE ⑧通过客户端的批处理实现的有限的事务支持 ⑨单表——还没有连接，同时二级索引也在开发当中 ⑩紧跟ANSI SQL标准

#### 二、Stinger

贡献者：：Hortonworks

简介：原叫Tez，下一代Hive,Hortonworks主导开发，运行在YARN上的DAG计算框架。

某些测试下，Stinger能提升10倍左右的性能，同时会让Hive支持更多的SQL，其主要优点包括：

①让用户在Hadoop获得更多的查询匹配。其中包括类似OVER的字句分析功能，支持WHERE查询，让Hive的样式系统更符合SQL模型。

②优化了Hive请求执行计划，优化后请求时间减少90%。改动了Hive执行引擎，增加单Hive任务的被秒处理记录数。

③在Hive社区中引入了新的列式文件格式（如ORC文件），提供一种更现代、高效和高性能的方式来储存Hive数据。

④引入了新的运行时框架——Tez，旨在消除Hive的延时和吞吐量限制。Tez通过消除不必要的task、障碍同步和对HDFS的读写作业来优化Hive job。这将优化Hadoop内部的执行链，彻底加速Hive负载处理。

Stinger官方网站>>>

### 三、Presto

贡献者：Facebook

简介：Facebook开源的数据查询引擎Presto，可对250PB以上的数据进行快速地交互式分析。该项目始于2012年秋季开始开发，目前该项目已经在超过1000名Facebook雇员中使用，运行超过30000个查询，每日数据在1PB级别。

Facebook称Presto的性能比诸如Hive和MapReduce要好上10倍有多。

Presto当前支持ANSI SQL的大多数特效，包括联合查询、左右联接、子查询以及一些聚合和计算函数；支持近似截然不同的计数(DISTINCT COUNT)等。

### 四、Shark

简介：Shark即Hive on Spark，本质上是通过Hive的HQL解析，把HQL翻译成Spark上的RDD操作，然后通过Hive的metadata获取数据库里的表信息，实际HDFS上的数据和文件，会由Shark获取并放到Spark上运算。Shark的特点就是快，完全兼容Hive，且可以在shell模式下使用rdd2sql()这样的API，把HQL得到的结果集，继续在scala环境下运算，支持自己编写简单的机器学习或简单分析处理函数，对HQL结果进一步分析计算。

①Shark速度快的原因除了Spark平台提供的基于内存迭代计算外，在设计上还存在对Spark上进行了一定的改造，主要有

②partial DAG execution：对join优化，调节并行粒度，因为Spark本身的宽依赖和窄依赖会影响并行计算和速度

基于列的压缩和存储：把HQL表数据按列存，每列是一个array，存在JVM上，避免了JVM GC低效，而压缩和解压相关的技术是Yahoo!提供的。

结来说，Shark是一个插件式的东西，在我现有的Spark和Hive及hadoop-client之间，在这两套都可用的情况下，Shark只要获取Hive的配置（还有metastore和exec等关键包），Spark的路径，Shark就能利用Hive和Spark，把HQL解析成RDD的转换，把数据取到Spark上运算和分析。在SQL on Hadoop这块，Shark有别于Impala，Stringer，而这些系统各有自己的设计思路，相对于对MR进行优化和改进的思路，Shark的思路更加简单明了些。

[Shark官方网站>>>](#)

### 五、Pig

简介：Pig是一种编程语言，它简化了Hadoop常见的工作任务。Pig可加载数据、表达转换数据以及存储最终结果。Pig内置的操作使得半结构化数据变得有意义（如日志文件）。同时Pig可扩展使用Java中添加的自定义数据类型并支持数据转换。

Pig最大的作用就是对mapreduce算法(框架)实现了一套shell脚本，类似我们通常熟悉的SQL语句，在Pig中称之为Pig Latin，在这套脚本中我们可以对加载出来的数据进行排序、过滤、求和、分组(group by)、关联(Joining)，Pig也可以由用户自定义一些函数对数据集进行操作，也就是传说中的UDF(user-defined functions)。

[Pig官方网站>>>](#)

### 六、Cloudera Impala

贡献者：[:Cloudera](#)

简介：Cloudera Impala 可以直接为存储在HDFS或HBase中的Hadoop数据提供快速，交互式的SQL查询。除了使用相同的存储平台外，Impala和Apache Hive一样也使用了相同的元数据，SQL语法（Hive SQL），ODBC驱动和用户接口（Hue Beeswax），这就很方便的为用户提供了一个相似并且统一的平台来进行批量或实时查询。

Cloudera Impala 是用来进行大数据查询的补充工具。Impala 并没有取代像Hive这样基于MapReduce的分布式处理框架。Hive和其它基于MapReduce的计算框架非常适合长时间运行的批处理作业，例如那些涉及到批量 Extract、Transform、Load，即需要进行ETL作业。

Impala 提供了：

- ① 数据科学家或数据分析师已经熟知的SQL接口
- ② 能够在Apache Hadoop 的大数据中进行交互式数据查询

③ Single system for big data processing and analytics so customers can avoid costly modeling and ETL just for analytics

Cloudera Impala[官方网站>>>](#)

## 七、Apache Drill

贡献者：：MapR

简介：Apache Drill是一个能够对大数据进行交互分析、开源的分布式系统，且基于Google Dremel实现，它能够运行在上千个节点的服务器集群上，且能在几秒内处理PB级或者万亿条的数据记录。Drill能够帮助企业用户快速、高效地进行Hadoop数据查询和企业级大数据分析。Drill于2012年8月份由Apache推出。

从Drill官方对其架构的介绍中得知，其具有适于实时的分析和快速的应用开发、适于半结构化/嵌套数据的分析、兼容现有的SQL环境和Apache Hive等特征。另外，Drill的核心模块是Drillbit服务，该服务模块包括远程访问子模块、SQL解析器、查询优化器、任务计划执行引擎、存储插件接口（DFS、HBase、Hive等的接口）、分布式缓存模块等几部分，如下图所示：

## 36大数据

Apache Drill[官方网站>>>](#)

## 八、Apache Tajo

简介：Apache Tajo项目的目的是在HDFS之上构建一个先进的数据仓库系统。Tajo将自己标榜为一个“大数据仓库”，但是它好像和之前介绍的那些低延迟查询引擎类似。虽然它支持外部表和Hive数据集（通过HCatalog），但是它的重点是数据管理，提供低延迟的数据访问，以及为更传统的ETL提供工具。它也需要在数据节点上部署Tajo特定的工作进程。

Tajo的功能包括：

①ANSI SQL兼容 ②JDBC 驱动 ③集成Hive metastore能够访问Hive数据集 ④一个命令行客户端 ⑤一个自定义函数API

Apache Tajo[官方网站>>>](#)

## 九、Hive

简介：hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive官方网站>>>

## 流式计算

### 一、Facebook Puma

贡献者：Facebook

简介：实时数据流分析

### 二、Twitter Rainbird

贡献者：Twitter

简介：Rainbird一款基于Zookeeper, Cassandra, Scribe, Thrift的分布式实时统计系统，这些基础组件的基本功能如下：

- ① Zookeeper，Hadoop子项目中的一款分布式协调系统，用于控制分布式系统中各个组件中的一致性。
- ②Cassandra，NoSQL中一款非常出色的产品，集合了Dynamo和Bigtable特性的分布式存储系统，用于存储需要进行统计的数据，统计数据，并且提供客户端进行统计数据的查询。（需要使用分布式Counter补丁CASSANDRA-1072）
- ③ Scribe，Facebook开源的一款分布式日志收集系统，用于在系统中将各个需要统计的数据源收集到Cassandra中。
- ④ Thrift，Facebook开源的一款跨语言C/S网络通信框架，开发人员基于这个框架可以轻易地开发C/S应用。

用处

Rainbird可以用于实时数据的统计：

- ①统计网站中每一个页面，域名的点击次数
- ②内部系统的运行监控（统计被监控服务器的运行状态）

### ③记录最大值和最小值

## 三、Yahoo S4

贡献者：Yahoo

简介：S4（Simple Scalable Streaming System）最初是Yahoo!为提高搜索广告有效点击率的问题而开发的一个平台，通过统计分析用户对广告的点击率，排除相关度低的广告，提升点击率。目前该项目刚启动不久，所以也可以理解为是他们提出的一个分布式流计算（Distributed Stream Computing）的模型。

S4的设计目标是：

- 提供一种简单的编程接口来处理数据流
- 设计一个可以在普通硬件之上可扩展的高可用集群。
- 通过在每个处理节点使用本地内存，避免磁盘I/O瓶颈达到最小化延迟
- 使用一个去中心的，对等架构；所有节点提供相同的功能和职责。没有担负特殊责任的中心节点。这大大简化了部署和维护。
- 使用可插拔的架构，使设计尽可能的即通用又可定制化。
- 友好的设计理念，易于编程，具有灵活的弹性

Yahoo S4官方网站>>>

## 四、Twitter Storm

贡献者：Twitter

简介：Storm是Twitter开源的一个类似于Hadoop的实时数据处理框架，它原来是由BackType开发，后BackType被Twitter收购，将Storm作为Twitter的实时数据分析系统。

实时数据处理的应用场景很广泛，例如商品推荐，广告投放，它能根据当前情景上下文（用户偏好，地理位置，已发生的查询和点击等）来估计用户点击的可能性并实时做出调整。

storm的三大作用领域：

1. 信息流处理（Stream Processing）

Storm可以用来实时处理新数据和更新数据库，兼具容错性和可扩展性，它可以用来处理源源不断的的消息，并将处理之后的结果保存到持久化介质中。

### 2. 连续计算 (Continuous Computation)

Storm可以进行连续查询并把结果即时反馈给客户，比如将Twitter上的热门话题发送到客户端。

### 3. 分布式远程过程调用 (Distributed RPC)

除此之外，Storm也被广泛用于以下方面：

精确的广告推送 实时日志的处理 Twitter Storm官方网站>>>

## 离线计算

### 一、Hadoop MapReduce

简介：MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。概念”Map（映射）”和”Reduce（归约）”，和它们的主要思想，都是从函数式编程语言里借来的，还有从矢量编程语言里借来的特性。它极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。当前的软件实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

Hadoop MapReduce官方网站>>>

### 二、Berkeley Spark

简介：Spark是UC Berkeley AMP lab所开源的类Hadoop MapReduce的通用的并行，Spark，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的map reduce的算法。

### 三、DataTorrent

简介：DataTorrent基于Hadoop 2.x构建，是一个实时的、有容错能力的数据流式处理和分析平台，它使用本地Hadoop应用程序，而这些应用程序可以与执行其它任务，如批处理，的应用程序共存。该平台的架构如下图所示：

## DataTorrent

相关文章：DataTorrent 1.0每秒处理超过10亿个实时事件

DataTorrent 将数据分析速度从“实时”提升至“现在时”

# 键值存储

## 一、LevelDB

### LevelDB

贡献者：Google

简介：Leveldb是一个google实现的非常高效的kv数据库，目前的版本1.2能够支持billion级别的数据量了。在这个数量级别下还有着非常的性能，主要归功于它的良好的设计。特别是LMS算法。

LevelDB 是单进程的服务，性能非常之高，在一台4核Q6600的CPU机器上，每秒钟写数据超过40w，而随机读的性能每秒钟超过10w。

此处随机读是完全命中内存的速度，如果是不命中 速度大大下降。

LevelDB官方网站>>>

## 二、RocksDB

贡献者：facebook

简介：RocksDB虽然在代码层面上是在LevelDB原有的代码上进行开发的，但却借鉴了Apache HBase的一些好的idea。在云计算横行的年代，开口不离Hadoop，RocksDB也开始支持HDFS，允许从HDFS读取数据。RocksDB支持一次获取多个K-V，还支持Key范围查找。LevelDB只能获取单个Key。

RocksDB除了简单的Put、Delete操作，还提供了一个Merge操作，说是为了对多个Put操作进行合并。

RocksDB提供一些方便的工具，这些工具包含解析sst文件中的K-V记录、解析MANIFEST文件的内容等。RocksDB支持多线程合并，而LevelDB是单线程合并的。

RocksDB官方网站>>>

### 三、HyperDex

贡献者：Facebook

#### HyperDex

HyperDex是一个分布式、可搜索的键值存储系统，特性如下：

分布式KV存储，系统性能能够随节点数目线性扩展 吞吐和延时都能秒杀现在风头正劲的MongoDB，吞吐甚至强于Redis 使用了hyperspace hashing技术，使得对存储的K-V的任意属性进行查询成为可能 官网：<http://hyperdex.org/>

### 四、TokyoCabinet

大数据

日本人Mikio Hirabayashi（平林千雄）开发的一款DBM数据库。Tokyo Cabinet 是一个DBM的实现。这里的数据库由一系列key-value对的记录构成。key和value都可以是任意长度的字节序列，既可以是二进制也可以是字符串。这里没有数据类型和数据表的概念。当做为Hash表数据库使用时，每个key必须是不同的，因此无法存储两个key相同的值。提供了以下访问方法：提供key,value参数来存储，按 key删除记录，按key来读取记录，另外，遍历key也被支持，虽然顺序是任意的不能被保证。这些方法跟Unix标准的DBM,例如GDBM,NDBM 等等是相同的，但是比它们的性能要好得多（因此可以替代它们）。下一代KV存储系统，支持strings、integers、floats、lists、maps和sets等丰富的数据类型。TokyoCabinet官方网站  
>>> 五、Voldemort 大数据 Voldemort是一个分布式键值存储系统，是Amazon's Dynamo的一个开源克隆。特性如下：支持自动复制数据到多个服务器上。支持数据自动分割所以每个服务器只包含总数据的一个子集。提供服务器故障透明处理功能。支持可拔插的序化支持，以实现复杂的键-值存储，它能够很好的集成常用的序化框架如：Protocol Buffers、Thrift、Avro和Java Serialization。数据项都被标识版本能够在发生故障时尽量保持数据的完整性而不会影响系统的可用性。每个节点相互独立，互不影响。支持可插拔的数据放置策略 官网：<http://project-voldemort.com/>

六、Amazon Dynamo 贡献者：亚马逊 简介：Amazon Dynamo 是一个经典的分布式Key-Value 存储系统，具备去中心化，高可用性，高扩展性的特点，但是为了达到这个目标在很多场景中牺牲了一致性。Dynamo在Amazon中得到了成功地应用，能够跨数据中心部署于上万个结点上提供服务，它的设计思想也被后续的许多分布式系统借鉴。如近来火热的Cassandra，实际上就是基本照搬了Dynamo的P2P架构，同时融合了BigTable的数据模型及存储算法。Amazon Dynamo官方网站>>>

七、Tair 贡献者：淘宝 简介：tair 是淘宝自己开发的一个分布式 key/value 存储引擎。tair 分为持久化和非持久化两种使用方式。非持久化的 tair 可以看成是一个分布式缓存。持久化的 tair 将数据存放于磁盘中。为了解决磁盘损坏导致数据丢失，tair 可以配置数据的备份数目，tair 自动将一份数据的不同备份放到不同的主机上，当有主机发生异常，无法正常提供服务的时候，其他的备份会继续提供服务。tair 的总体结构 Tair。tair 作为一个分布式系统，是由一个中心控制节点和一系列的服务节点组成。我们称中心控制节点为 config server。服务节点是 data server。config server 负责管理所有的 data server，维护 data server 的状态信息。data server 对外提供各种数据服务，并以心跳的形式将自身状况汇报给 config server。config server 是控制点，而且是单点，目前采用一主一备的形式来保证其可靠性。所有的 data server 地位都是等价的。

八、Apache Accumulo Apache Accumulo Apache Accumulo 是一个可靠的、可伸缩的、高性能的排序分布式的 Key-Value 存储解决方案，基于单元访问控制以及可定制的服务器端处理。Accumulo 使用 Google BigTable 设计思路，基于 Apache Hadoop、Zookeeper 和 Thrift 构建。

官网：<http://accumulo.apache.org/>

### 九、Redis

Redis 是一个高性能的 key-value 存储系统，和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string（字符串）、list（链表）、set（集合）和 zset（有序集合）。与 memcached 一样，为了保证效率，数据都是缓存在内存中，区别的是 Redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了主从同步。

Redis 的出现，很大程度补偿了 memcached 这类 key/value 存储的不足，在部分场合可以对关系数据库起到很好的补充作用。它提供了 Python、Ruby、Erlang、PHP 客户端，使用很方便。

官网：<http://redis.io/>

## 表格存储

### 一、OceanBase

贡献者：阿里巴巴

相关文章：26页PPT解密支撑支付宝交易的分布式数据库系统——OceanBase

简介：OceanBase是一个支持海量数据的高性能分布式数据库系统，实现了数千亿条记录、数百TB数据上的跨行跨表事务，由淘宝核心系统研发部、运维、DBA、广告、应用研发等部门共同完成。在设计和实现OceanBase的时候暂时摒弃了不紧急的DBMS的功能，例如临时表，视图(view)，研发团队把有限的资源集中到关键点上，当前 OceanBase主要解决数据更新一致性、高性能的跨表读事务、范围查询、join、数据全量及增量dump、批量数据导入。

目前OceanBase已经应用于淘宝收藏夹，用于存储淘宝用户收藏条目和具体的商品、店铺信息，每天支持4~5千万的更新操作。等待上线的应用还包括CTU、SNS等，每天更新超过20亿，更新数据量超过2.5TB，并会逐步在淘宝内部推广。

OceanBase 0.3.1在Github开源，开源版本为Revision:12336。

官网：<http://alibaba.github.io/oceanbase/>

### 二、Amazon SimpleDB

贡献者：亚马逊

Amazon SimpleDB是一个分散式数据库，以Erlang撰写。同与Amazon EC2和亚马逊的S3一样作为一项Web服务，属于亚马逊网络服务的一部分。

### Amazon SimpleDB

正如EC2和S3，SimpleDB的按照存储量，在互联网上的传输量和吞吐量收取费用。在2008年12月1日，亚马逊推出了新的定价策略，提供了免费1 GB的数据和25机器小时的自由层(Free Tire)。将其中的数据转移到其他亚马逊网络服务是免费的。

它是一个可大规模伸缩、用 Erlang 编写的高可用数据存储。

官网：<http://aws.amazon.com/cn/simpledb/>

### 三、Vertica

贡献者：惠普

简介：惠普2011年2月份起始3月21号完成收购Vertica。Vertica基于列存储。基于列存储的设计相比传统面向行存储的数据库具有巨大的优势。同时Vertica支持MPP（massively parallel processing）等技术，查询数据时Vertica只需取得需要的列，而不是被选择行的所有数据，其平均性能可提高50x-1000x倍。（查询性能高速度快）

Vertica的设计者多次表示他们的产品围绕着高性能和高可用性设计。由于对MPP技术的支持，可提供对粒度，可伸缩性和可用性的优势。每个节点完全独立运作，完全无共享架构，降低对共享资源的系统竞争。

Vertica的数据库使用标准的SQL查询，同时Vertica的架构非常适合云计算，包括虚拟化，分布式多节点运行等，并且可以和Hadoop/MapReduce进行集成。

Vertica官网：<http://www.vertica.com/>

### 四、Cassandra

贡献者：facebook

相关文章：开源分布式NoSQL数据库系统——Cassandra Cassandra与HBase的大数据对决 谁是胜者？

简介：Cassandra是一套开源分布式NoSQL数据库系统。它最初由Facebook开发，用于储存收件箱等简单格式数据，集GoogleBigTable的数据模型与Amazon Dynamo的完全分布式的架构于一身Facebook于2008将 Cassandra 开源，此后，由于Cassandra良好的可扩放性，被Digg、Twitter等知名Web 2.0网站所采纳，成为了一种流行的分布式结构化数据存储方案。

### Cassandra

Cassandra是一个混合型的非关系的数据库，类似于Google的BigTable。其主要功能比Dynamo（分布式的Key-Value存储系统）更丰富，但支持度却不如文档存储MongoDB（介于关系数据库和非关系数据库之间的开源产品，是非关系数据库当中功能最丰富，最像关系数据库的。支持的数据结构非常松散，是类似json的bjson格式，因此可以存储比较复杂的数据类型）。Cassandra最初由Facebook开发，后转变成了开源项目。它是一个网络社交云计算方面理想的数据库。以Amazon专有的完全分布式的Dynamo为基础，结合了Google BigTable基于列族（Column Family）的数据模型。P2P去中心化的存储。很多方面都可以称之为Dynamo 2.0。

Cassandra官网：<http://cassandra.apache.org/>

### 五、HyperTable

#### Hypertable

简介：Hypertable是一个开源、高性能、可伸缩的数据库，它采用与Google的Bigtable相似的模型。在过去数年中，Google为在PC集群上运行的可伸缩计算基础设施设计建造了三个关键部分。

## Hypertable

第一个关键的基础设施是Google File System（GFS），这是一个高可用的文件系统，提供了一个全局的命名空间。它通过跨机器（和跨机架）的文件数据复制来达到高可用性，并因此免受传统文件存储系统无法避免的许多失败的影响，比如电源、内存和网络端口等失败。第二个基础设施是名为Map-Reduce的计算框架，它与GFS紧密协作，帮助处理收集到的海量数据。第三个基础设施是Bigtable，它是传统数据库的替代。Bigtable让你可以通过一些主键来组织海量数据，并实现高效的查询。Hypertable是Bigtable的一个开源实现，并且根据我们的想法进行了一些改进。

HyperTable官网：<http://hypertable.org/>

## 六、FoundationDB

简介：支持ACID事务处理的NoSQL数据库，提供非常好的性能、数据一致性和操作弹性。

2015年1月2日，FoundationDB已经发布了其key-value数据库的3.0版本，主要专注于可伸缩性和性能上的改善。FoundationDB的CEO David Rosenthal在一篇博客上宣布了新的版本，其中展示了FoundationDB 3.0在可伸缩性方面的数据，它可以在一个32位的c3.8xlarge EC2实例上每秒写入1440万次；这在性能上是之前版本的36倍。

除了性能和可伸缩性的改善之外，FoundationDB 3.0还包含了对监控支持的改善。这种监控机制不仅仅是简单的机器检查，它添加了对多种潜在的硬件瓶颈的诊断，并且把那些高层级的信息整合到现有监控基础架构中。

官网：<https://foundationdb.com/>

## 七：HBase

贡献者：Fay Chang 所撰写的“Bigtable

## HBase

简介：HBase是一个分布式的、面向列的开源数据库，该技术来源于Fay Chang所撰写的Google论文“Bigtable：一个结构化数据的分布式存储系统”。就像Bigtable利用了Google文件系统（File System）所提供的分布式数据存储一样，HBase在

Hadoop之上提供了类似于Bigtable的能力。HBase是Apache的Hadoop项目的子项目。HBase不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据仓库。另一个不同的是HBase基于列的而不是基于行的模式。

官网：<http://hbase.apache.org/>

## 文件存储

### 一、CouchDB

简介：CouchDB是用Erlang开发的面向文档的数据库系统，最近刚刚发布了1.0版本（2010年7月14日）。CouchDB不是一个传统的关系数据库，而是面向文档的数据库，其数据存储方式有点类似lucene的index文件格式，CouchDB最大的意义在于它是一个面向web应用的新一代存储系统，事实上，CouchDB的口号就是：下一代的Web应用存储系统。

#### CouchDB

特点：

一、CouchDB是分布式的数据库，他可以把存储系统分布到n台物理的节点上面，并且很好的协调和同步节点之间的数据读写一致性。这当然也得靠Erlang无与伦比的并发特性才能做到。对于基于web的大规模应用文档应用，分布式可以让它不必像传统的关系数据库那样分库拆表，在应用代码层进行大量的改动。

二、CouchDB是面向文档的数据库，存储半结构化的数据，比较类似lucene的index结构，特别适合存储文档，因此很适合CMS，电话本，地址本等应用，在这些应用场景，文档数据库要比关系数据库更加方便，性能更好。

三、CouchDB支持REST API，可以让用户使用JavaScript来操作CouchDB数据库，也可以用JavaScript编写查询语句，我们可以想像一下，用AJAX技术结合CouchDB开发出来的CMS系统会是多么的简单和方便。

其实CouchDB只是Erlang应用的冰山一角，在最近几年，基于Erlang的应用也得到的蓬勃的发展，特别是在基于web的大规模，分布式应用领域，几乎都是Erlang的优势项目。

官网：<http://couchdb.apache.org/>

### 二、MongoDB

简介：MongoDB 是一个基于分布式文件存储的数据库。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

相关文章：MongoDB的基本特性与内部构造 大数据吃香 创业公司MongoDB估值达16亿美元

mongodb

特点

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

\*面向集合存储，易存储对象类型的数据。

mongodb集群参考

mongodb集群参考

\*模式自由。

\*支持动态查询。

\*支持完全索引，包含内部对象。

\*支持查询。

\*支持复制和故障恢复。

\*使用高效的二进制数据存储，包括大型对象（如视频等）。

\*自动处理碎片，以支持云计算层次的扩展性。

\*支持RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。

\*文件存储格式为BSON（一种JSON的扩展）。

\*可通过网络访问。

官网：<https://www.mongodb.org/>

### 三、Tachyon

贡献者：Haoyuan Li（李浩源）

李浩源

简介：Tachyon是一个分布式内存文件系统，可以在集群里以访问内存的速度来访问存在tachyon里的文件。把Tachyon是架构在最底层的分布式文件存储和上层的各种计算框架之间的一种中间件。主要职责是将那些不需要落地到DFS里的文件，落地到分布式内存文件系统中，来达到共享内存，从而提高效率。同时可以减少内存冗余，GC时间等。

Tachyon

Tachyon架构

Tachyon的架构是传统的Master—slave架构，这里和Hadoop类似，TachyonMaster里WorkflowManager是 Master进程，因为是为了防止单点问题，通过Zookeeper做了HA，可以部署多台Standby Master。Slave是由Worker Daemon和Ramdisk构成。这里个人理解只有Worker Daemon是基于JVM的，Ramdisk是一个off heap memory。Master和Worker直接的通讯协议是Thrift。

下图来自Tachyon的作者Haoyuan Li：

Tachyon

下载地址：<https://github.com/amplab/tachyon>

### 四、KFS

简介：GFS的C++开源版本，Kosmos distributed file system (KFS)是一个专门为数据密集型应用（搜索引擎，数据挖掘等）而设计的存储系统，类似于Google的GFS和Hadoop的HDFS分布式文件系统。KFS使用C++实现，支持的客户端包括C++，Java和Python。KFS系统由三部分组成，分别是metaserver、chunkserver和client library。

官网：<http://code.google.com/p/kosmosfs/>

### 五、HDFS

大数据

简介：Hadoop分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS放宽了一部分POSIX约束，来实现流式读取文件系统数据的目的。HDFS在最开始是作为Apache Nutch搜索引擎项目的基础架构而开发的。HDFS是Apache Hadoop Core项目的一部分。

官网：<http://hadoop.apache.org/>

## 资源管理

### 一、Twitter Mesos

开发者：Twitter研发人员John Oskasson

#### Twitter Mesos

简介：Apache Mesos是由加州大学伯克利分校的AMPLab首先开发的一款开源群集管理软件，支持Hadoop、ElasticSearch、Spark、Storm 和Kafka等架构，由于其开源性质越来越受到一些大型云计算公司的青睐，例如Twitter、Facebook等。

参考文章：Mesos渐入主流,Twitter模式有望“无限复制”-CSDN.NET

官网：<http://mesos.apache.org/>

### 二、Hadoop Yarn

Hadoop 新 MapReduce 框架 Yarn。为从根本上解决旧 MapReduce 框架的性能瓶颈，促进 Hadoop 框架的更长远发展，从 0.23.0 版本开始，Hadoop 的 MapReduce 框架完全重构，发生了根本的变化。新的 Hadoop MapReduce 框架命名为 MapReduceV2 或者叫 Yarn，其架构图如下图所示：

#### Hadoop Yarn

Yarn 框架相对于老的 MapReduce 框架什么优势呢？我们可以看到：

1、这个设计大大减小了 JobTracker（也就是现在的 ResourceManager）的资源消耗，并且让监测每一个 Job 子任务 (tasks) 状态的程序分布式化了，更安全、更优美。

2、在新的 Yarn 中，ApplicationMaster 是一个可变更的部分，用户可以对不同的编程模型写自己的 AppMst，让更多类型的编程模型能够跑在 Hadoop 集群中，可以参考 hadoop Yarn 官方配置模板中的 mapred-site.xml 配置。

3、对于资源的表示以内存为单位(在目前版本的 Yarn 中，没有考虑 cpu 的占用)，比之前以剩余 slot 数目更合理。

4、老的框架中，JobTracker 一个很大的负担就是监控 job 下的 tasks 的运行状况，现在，这个部分就扔给 ApplicationMaster 做了，而 ResourceManager 中有一个模块叫做 ApplicationsMasters( 注意不是 ApplicationMaster)，它是监测 ApplicationMaster 的行状况，如果出问题，会将其在其他机器上重启。

5、Container 是 Yarn 为了将来作资源隔离而提出的一个框架。这一点应该借鉴了 Mesos 的工作，目前是一个框架，仅仅提供 java 虚拟机内存的隔离，hadoop 团队的设计思路应该后续能支持更多的资源调度和控制，既然资源表示成内存量，那就没有了之前的 map slot/reduce slot 分开造成集群资源闲置的尴尬情况。

官网：<http://hadoop.apache.org/>

## 日志收集系统

### 一、Facebook Scribe

scribe

贡献者：Facebook

简介：Scribe是Facebook开源的日志收集系统，在Facebook内部已经得到大量的应用。它能够从各种日志源上收集日志，存储到一个中央存储系统（可以是NFS，分布式文件系统等）上，以便于进行集中统计分析处理。它为日志的“分布式收集，统一处理”提供了一个可扩展的，高容错的方案。当中央存储系统的网络或者机器出现故障时，scribe会将日志转存到本地或者另一个位置，当中央存储系统恢复后，scribe会将转存的日志重新传输给中央存储系统。其通常与Hadoop结合使用，scribe用于向HDFS中push日志，而Hadoop通过MapReduce作业进行定期处理。

Scribe的系统架构

scribe

代码托管：<https://github.com/facebook/scribe>

## 二、Cloudera Flume

贡献者：Cloudera

简介：**Flume**是**Cloudera**提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，**Flume**支持在日志系统中定制各类数据发送方，用于收集数据；同时，**Flume**提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

**Flume**提供了从**console**（控制台）、**RPC**（Thrift-RPC）、**text**（文件）、**tail**（UNIX tail）、**syslog**（syslog日志系统，支持TCP和UDP等2种模式），**exec**（命令执行）等数据源上收集数据的能力。

当前**Flume**有两个版本**Flume 0.9X**版本的统称**Flume-og**，**Flume1.X**版本的统称**Flume-ng**。由于**Flume-ng**经过重大重构，与**Flume-og**有很大不同，使用时请注意区分。

**Cloudera Flume**构架：

**Cloudera Flume**

官网：<http://flume.apache.org/>

## 三、logstash

简介：**logstash** 是一个应用程序日志、事件的传输、处理、管理和搜索的平台。你可以用它来统一对应用程序日志进行收集管理，提供 Web 接口用于查询和统计。他可以对你的日志进行收集、分析，并将其存储供以后使用（如，搜索），您可以使用它。说到搜索，**logstash**带有一个web界面，搜索和展示所有日志。

**logstash**

官网：<http://www.logstash.net/>

## 四、kibana

简介：**Kibana** 是一个为 **Logstash** 和 **ElasticSearch** 提供的日志分析的 Web 接口。可使用它对日志进行高效的搜索、可视化、分析等各种操作。**kibana** 也是一个开源和免费的工具，他可以帮助您汇总、分析和搜索重要数据日志并提供友好的web界面。他可以为 **Logstash** 和 **ElasticSearch** 提供的日志分析的 Web 界面。

主页：<http://kibana.org/>

代码托管：<https://github.com/rashidkpc/Kibana/downloads>

## 消息系统

### 一、StormMQ

简介：MQMessageQueue消息队列产品 StormMQ，是一种服务程序。

官网：<http://stormmq.com/>

### 二、ZeroMQ

简介：这是个类似于Socket的一系列接口，他跟Socket的区别是：普通的socket是端到端的（1:1的关系），而ZMQ却是可以N：M 的关系，人们对BSD套接字的了解较多的是点对点的连接，点对点连接需要显式地建立连接、销毁连接、选择协议（TCP/UDP）和处理错误等，而ZMQ屏蔽了这些细节，让你的网络编程更为简单。ZMQ用于node与node间的通信，node可以是主机或者是进程。

引用官方的说法：“ZMQ(以下ZeroMQ简称ZMQ)是一个简单好用的传输层，像框架一样的一个socket library，他使得Socket编程更加简单、简洁和性能更高。是一个消息处理队列库，可在多个线程、内核和主机盒之间弹性伸缩。ZMQ的明确目标是“成为标准网络协议栈的一部分，之后进入Linux内核”。现在还未看到它们的成功。但是，它无疑是极具前景的、并且是人们更加需要的“传统”BSD套接字之上的一层封装。ZMQ让编写高性能网络应用程序极为简单和有趣。”

官网：<http://zeromq.org/>

### 三、RabbitMQ

简介：RabbitMQ是一个受欢迎的消息代理，通常用于应用程序之间或者程序的不同组件之间通过消息来进行集成。本文简单介绍了如何使用 RabbitMQ，假定你已经配置好了rabbitmq服务器。

#### RabbitMQ

RabbitMQ是用Erlang，对于主要的编程语言都有驱动或者客户端。我们这里要用的是Java，所以先要获得Java客户端。

像RabbitMQ这样的消息代理可用来模拟不同的场景，例如点对点的消息分发或者订阅/推送。我们的程序足够简单，有两个基本的组件，一个生产者用于产生消息，还有一个消费者用来使用产生的消息。

官网：<https://www.rabbitmq.com/>

#### 四、Apache ActiveMQ

简介：ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现，尽管JMS规范出台已经很久的事情了，但是JMS在当今的J2EE应用中间仍然扮演着特殊的地位。

#### Apache ActiveMQ

特性：

1. 多种语言和协议编写客户端。语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议：OpenWire,Stomp REST,WS Notification,XMPP,AMQP
2. 完全支持JMS1.1和J2EE 1.4规范（持久化，XA消息，事务）
3. 对Spring的支持，ActiveMQ可以很容易内嵌到使用Spring的系统里面去，而且也支持Spring2.0的特性
4. 通过了常见J2EE服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过JCA 1.5 resource adaptors的配置，可以让ActiveMQ可以自动的部署到任何兼容J2EE 1.4 商业服务器上
5. 支持多种传送协议：in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA
6. 支持通过JDBC和journal提供高速的消息持久化
7. 从设计上保证了高性能的集群，客户端-服务器，点对点
8. 支持Ajax
9. 支持与Axis的整合
10. 可以很容易得调用内嵌JMS provider，进行测试

官网：<http://activemq.apache.org/>

#### 五、Jafka

贡献者：LinkedIn

简介：Jafka 是一个开源的、高性能的、跨语言分布式消息系统，使用 GitHub 托管。Jafka 最早是由 Apache 孵化的 Kafka（由 LinkedIn 捐助给 Apache）克隆而来。由于是一个开放式的数据传输协议，因此除了 Java 开发语言受到支持，Python、Ruby、C、C++ 等其他语言也能够很好的得到支持。

特性：

1、消息持久化非常快，服务端存储消息的开销为  $O(1)$ ，并且基于文件系统，能够持久化 TB 级的消息而不损失性能。

2、吞吐量取决于网络带宽。

3、完全的分布式系统，broker、producer、consumer 都原生自动支持分布式。自动实现复杂均衡。

4、内核非常小，整个系统（包括服务端和客户端）只有一个 272KB 的 jar 包，内部机制也不复杂，适合进行内嵌或者二次开发。整个服务端加上依赖组件共 3.5MB。

5、消息格式以及通信机制非常简单，适合进行跨语言开发。目前自带的 Python 3.x 的客户端支持发送消息和接收消息。

官网：<http://kafka.apache.org/>

### 六、Apache Kafka

贡献者：LinkedIn

简介：Apache Kafka 是由 Apache 软件基金会开发的一个开源消息系统项目，由 Scala 写成。Kafka 最初是由 LinkedIn 开发，并于 2011 年初开源。2012 年 10 月从 Apache Incubator 毕业。该项目的目标是为处理实时数据提供一个统一、高通量、低等待的平台。

Kafka 是一个分布式的、分区的、多副本的日志提交服务。它通过一种独一无二的设计提供了一个消息系统的功能。

Kafka 集群可以在一个指定的时间内保持所有发布上来的消息，不管这些消息有没有被消费。打个比方，如果这个时间设置为两天，那么在消息发布的两天以内，这条消息都是可以被消费的，但是在两天后，这条消息就会被系统丢弃以释放空间。Kafka 的性能不会受数据量的大小影响，因此保持大量的数据不是一个问题。

官网：<http://kafka.apache.org/>

# 分布式服务

## 一、ZooKeeper

贡献者：Google

简介：ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、名字服务、分布式同步、组服务等。

ZooKeeper是以Fast Paxos算法为基础的，paxos算法存在活锁的问题，即当有多个proposer交错提交时，有可能互相排斥导致没有一个proposer能提交成功，而Fast Paxos作了一些优化，通过选举产生一个leader，只有leader才能提交propose，具体算法可见Fast Paxos。因此，要想弄懂ZooKeeper首先得对Fast Paxos有所了解。

架构：

zookeeper

官网：<http://zookeeper.apache.org/>

RPC（Remote Procedure Call Protocol）——远程过程调用协议

## 一、Apache Avro

简介：Apache Avro是Hadoop下的一个子项目。它本身既是一个序列化框架，同时也实现了RPC的功能。Avro官网描述Avro的特性和功能如下：

丰富的数据结构类型；快速可压缩的二进制数据形式；存储持久数据的文件容器；提供远程过程调用RPC；简单的动态语言结合功能。相比于Apache Thrift和Google的Protocol Buffers，Apache Avro具有以下特点：

支持动态模式。Avro不需要生成代码，这有利于搭建通用的数据处理系统，同时避免了代码入侵。数据无须加标签。读取数据前，Avro能够获取模式定义，这使得Avro在数据编码时只需要保留更少的类型信息，有利于减少序列化后的数据大小。  
官网：<http://avro.apache.org/>

## 二、Facebook Thrift

贡献者：Facebook

简介：Thrift源于大名鼎鼎的facebook之手，在2007年facebook提交Apache基金会将Thrift作为一个开源项目，对于当时的facebook来说创造thrift是为了解决facebook系统中各系统间大数据量的传输通信以及系统之间语言环境不同需要跨平台的特性。

thrift可以支持多种程序语言，例如: C++, C#, Cocoa, Erlang, Haskell, Java, Ocaml, Perl, PHP, Python, Ruby, Smalltalk. 在多种不同的语言之间通信thrift可以作为二进制的高性能的通讯中间件，支持数据(对象)序列化和多种类型的RPC服务。

Thrift适用于程序对程序静态的数据交换，需要先确定好他的数据结构，他是完全静态化的，当数据结构发生变化时，必须重新编辑IDL文件，代码生成，再编译载入的流程，跟其他IDL工具相比较可以视为是Thrift的弱项，Thrift适用于搭建大型数据交换及存储的通用工具，对于大型系统中的内部数据传输相对于JSON和xml无论在性能、传输大小上有明显的优势。

Thrift 主要由5个部分组成：

- 类型系统以及 IDL 编译器：负责由用户给定的 IDL 文件生成相应语言的接口代码
- TProtocol：实现 RPC 的协议层，可以选择多种不同的对象串行化方式，如 JSON, Binary。
- TTransport：实现 RPC 的传输层，同样可以选择不同的传输层实现，如socket, 非阻塞的 socket, MemoryBuffer 等。
- TProcessor：作为协议层和用户提供的服务实现之间的纽带，负责调用服务实现的接口。
- TServer：聚合 TProtocol, TTransport 和 TProcessor 几个对象。

上述的这5个部件都是在 Thrift 的源代码中通过为不同语言提供库来实现的，这些库的代码在 Thrift 源码目录的 lib 目录下面，在使用 Thrift 之前需要先熟悉与自己的语言对应的库提供的接口。

Facebook Thrift构架：

Thrift

官网：<http://thrift.apache.org/>

集群管理 一、Nagios

**简介：**Nagios是一款开源的免费网络监视工具，能有效监控Windows、Linux和Unix的主机状态，交换机路由器等网络设置，打印机等。在系统或服务状态异常时发出邮件或短信报警第一时间通知网站运维人员，在状态恢复后发出正常的邮件或短信通知。

Nagios可运行在Linux/Unix平台之上，同时提供一个可选的基于浏览器的WEB界面以方便系统管理人员查看网络状态，各种系统问题，以及日志等等。

官网：<http://www.nagios.org/>

### 二、Ganglia

**简介：**Ganglia是UC Berkeley发起的一个开源集群监视项目，设计用于测量数以千计的节点。Ganglia的核心包含gmond、gmetad以及一个Web前端。主要是用来监控系统性能，如：cpu、mem、硬盘利用率，I/O负载、网络流量情况等，通过曲线很容易见到每个节点的工作状态，对合理调整、分配系统资源，提高系统整体性能起到重要作用。

### Ganglia

官网：<http://ganglia.sourceforge.net/>

### 三、Apache Ambari

**简介：**Apache Ambari是一种基于Web的工具，支持Apache Hadoop集群的供应、管理和监控。Ambari目前已支持大多数Hadoop组件，包括HDFS、MapReduce、Hive、Pig、Hbase、Zookeeper、Sqoop和Hcatalog等。

Apache Ambari 支持HDFS、MapReduce、Hive、Pig、Hbase、Zookeeper、Sqoop和Hcatalog等的集中管理。也是5个顶级hadoop管理工具之一。

### Apache Ambari

Ambari主要取得了以下成绩：

通过一步一步的安装向导简化了集群供应。预先配置好关键的运维指标（metrics），可以直接查看Hadoop Core（HDFS和MapReduce）及相关项目（如HBase、Hive和HCatalog）是否健康。支持作业与任务执行的可视化与分析，能够更好地查看依赖和性能。通过一个完整的RESTful API把监控信息暴露出来，集成了现有的运维工具。用户界面非常直观，用户可以轻松有效地查看信息并控制集

群。Ambari使用Ganglia收集度量指标，用Nagios支持系统报警，当需要引起管理员的关注时（比如，节点停机或磁盘剩余空间不足等问题），系统将向其发送邮件。

此外，Ambari能够安装安全的（基于Kerberos）Hadoop集群，以此实现了对Hadoop安全的支持，提供了基于角色的用户认证、授权和审计功能，并为用户管理集成了LDAP和Active Directory。

官网：<http://ambari.apache.org/>

### 基础设施一、LevelDB

贡献者：Jeff Dean和Sanjay Ghemawat

简介：Leveldb是一个google实现的非常高效的kv数据库，目前的版本1.2能够支持billion级别的数据量了。在这个数量级别下还有着非常的性能，主要归功于它的良好的设计。特别是LMS算法。LevelDB是单进程的服务，性能非常之高，在一台4核Q6600的CPU机器上，每秒钟写数据超过40w，而随机读的性能每秒钟超过10w。

Leveldb框架：

Leveldb

官网：<http://code.google.com/p/leveldb/>

### 二、SSTable

简介：如果说Protocol Buffer是谷歌独立数据记录的通用语言，那么有序字符串表(SSTable，Sorted String Table)则是用于存储，处理和数据集交换的最流行的数据输出格式。正如它的名字本身，SSTable是有效存储大量键-值对的简单抽象，对高吞吐量顺序读/写进行了优化。

SSTable是Bigtable中至关重要的一块，对于LevelDB来说也是如此。

### 三、RecordIO

贡献者：Google

简介：我们大家都在用文件来存储数据。文件是存储在磁盘上的。如果在一些不稳定的介质上，文件很容易损坏。即时文件某个位置出现一点小小的问题，整个文件就废了。

下面我来介绍Google的一个做法，可以比较好的解决这个问题。那就是recordio文件格式。recordio的存储单元是一个一个record。这个record可以根据业务的需要自行定义。但Google有一种建议的处理方式就是使用protobuf。

recordio底层的格式其实很简单。一个record由四部分组成：

MagicNumber (32 bits) Uncompressed data payload size (64 bits) Compressed data payload size (64 bits), or 0 if the data is not compressed Payload, possibly compressed. 详细格式如下图所示：

## RecordIO

到这里，大家可能已经知道，recordio之所以能对付坏数据，其实就是在那个MagicNumber（校验值）。

## 四、Flat Buffers

贡献者：Google

简介：谷歌开源高效、跨平台的序列化库FlatBuffers。

该库的构建是专门为游戏开发人员的性能需求提供支持，它将序列化数据存储在缓存中，这些数据既可以存储在文件中，又可以通过网络原样传输，而不需要任何解析开销。

FlatBuffers有如下一些关键特性——

访问序列化数据不需要打包/拆包 节省内存而且访问速度快——缓存只占用访问数据所需要的内存；不需要任何额外的内存。 灵活性——通过可选字段向前向后兼容代码规模小 强类型——错误在编译时捕获，而不是在运行时 便利性——生成的C++头文件代码简洁。如果需要，有一项可选功能可以用来在运行时高效解析Schema和JSON-like格式的文本。 跨平台——使用C++编写，不依赖STL之外的库，因此可以用于任何有C++编辑器的平台。当前，该项目包含构建方法和在Android、Linux、Windows和OSX等操作系统上使用该库的示例。与Protocol Buffers或JSON Parsing这样的可选方案相比，FlatBuffers的优势在于开销更小，这主要是由于它没有解析过程。

代码托管：<https://github.com/google/flatbuffers>

## 五、Protocol Buffers

贡献者：Google

简介：Protocol Buffers是Google公司开发的一种数据描述语言，类似于XML能够将结构化数据序列化，可用于数据存储、通信协议等方面。它不依赖于语言和平台并且可扩展性极强。现阶段官方支持C++、JAVA、Python等三种编程语言，但可以找到大量的几乎涵盖所有语言的第三方拓展包。

通过它，你可以定义你的数据的结构，并生成基于各种语言的代码。这些你定义的数据流可以轻松地在传递并不破坏你已有的程序。并且你也可以更新这些数据而现有的程序也不会受到任何的影响。

Protocol Buffers经常被简称为protobuf。

官网：<http://code.google.com/p/protobuf/>

### 六、Consistent Hashing（哈希算法）

简介：一致性哈希算法在1997年由麻省理工学院提出的一种分布式哈希（DHT）实现算法，设计目标是为了解决因特网中的热点(Hot spot)问题，初衷和CARP十分类似。一致性哈希修正了CARP使用的简单哈希算法带来的问题，使得分布式哈希(DHT)可以在P2P环境中真正得到应用。

#### Consistent Hashing

一致性hash算法提出了在动态变化的Cache环境中，判定哈希算法好坏的四个定义：

1、平衡性(Balance)：平衡性是指哈希的结果能够尽可能分布到所有的缓冲中去，这样可以使得所有的缓冲空间都得到利用。很多哈希算法都能够满足这一条件。

2、单调性(Monotonicity)：单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到原有的或者新的缓冲中去，而不会被映射到旧的缓冲集合中的其他缓冲区。

3、分散性(Spread)：在分布式环境中，终端有可能看不到所有的缓冲，而是只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上时，由于不同终端所见的缓冲范围有可能不同，从而导致哈希的结果不一致，最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。分散性的定义就是上述情况发生的严重程度。好的哈希算法应能够尽量避免不一致的情况发生，也就是尽量降低分散性。

4、负载(Load)：负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射为不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。

在分布式集群中，对机器的添加删除，或者机器故障后自动脱离集群这些操作是分布式集群管理最基本的功能。如果采用常用的 $\text{hash}(\text{object}) \% N$ 算法，那么在有机器添加或者删除后，很多原有的数据就无法找到了，这样严重的违反了单调性原则。

## 七、Netty

贡献者：JBoss

简介：Netty是由JBoss提供的一个java开源框架。Netty提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠的网络服务器和客户端程序。

### Netty

也就是说，Netty 是一个基于NIO的客户，服务器端编程框架，使用Netty 可以确保你快速和简单的开发出一个网络应用，例如实现了某种协议的客户，服务端应用。Netty相当简化和流线化了网络应用的编程开发过程，例如，TCP和UDP的socket服务开发。

“快速”和“简单”并不意味着会让你的最终应用产生维护性或性能上的问题。Netty 是一个吸收了多种协议的实现经验，这些协议包括FTP,SMTP,HTTP，各种二进制，文本协议，并经过相当精心设计的项目，最终，Netty 成功的找到了一种方式，在保证易于开发的同时还保证了其应用的性能，稳定性和伸缩性。

官网：<http://netty.io/>

## 八、BloomFilter

简介：Bloom filter 是由 Howard Bloom 在 1970 年提出的二进制向量数据结构，它具有很好的空间和时间效率，被用来检测一个元素是不是集合中的一个成员。如果检测结果为是，该元素不一定在集合中；但如果检测结果为否，该元素一定不在集合中。因此Bloom filter具有100%的召回率。这样每个检测请求返回有“在集合内（可能错误）”和“不在集合内（绝对不在集合内）”两种情况，可见 Bloom filter 是牺牲了正确率和时间以节省空间。

Bloom filter 优点就是它的插入和查询时间都是常数，另外它查询元素却不保存元素本身，具有良好的安全性。

## 搜索引擎

### 一、Nutch

简介：Nutch 是一个开源Java 实现的搜索引擎。它提供了我们运行自己的搜索引擎所需的全部工具。包括全文搜索和Web爬虫。

尽管Web搜索是漫游Internet的基本要求，但是现有web搜索引擎的数目却在下降。并且这很有可能进一步演变成为一个公司垄断了几乎所有的web搜索为其谋取商业利益。这显然不利于广大Internet用户。

#### Nutch

Nutch为我们提供了这样一个不同的选择。相对于那些商用的搜索引擎，Nutch作为开放源代码 搜索引擎将会更加透明，从而更值得大家信赖。现在所有主要的搜索引擎都采用私有的排序算法，而不会解释为什么一个网页会排在一个特定的位置。除此之外，有的搜索引擎依照网站所付的 费用，而不是根据它们本身的价值进行排序。与它们不同，Nutch没有什么需要隐瞒，也没有 动机去扭曲搜索的结果。Nutch将尽自己最大的努力为用户提供最好的搜索结果。

Nutch目前最新的版本为version v2.2.1。

官网：<https://nutch.apache.org/>

### 二、Lucene

开发者：Doug Cutting（Hadoop之父，你懂的）

简介：Lucene是apache软件基金会4 jakarta项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，即它不是一个完整的全文检索引擎，而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（英文与德文两种西方语言）。Lucene的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

#### Lucene

官网：<http://lucene.apache.org/>

### 三、SolrCloud

简介：SolrCloud是Solr4.0版本以后基于Solr和Zookeeper的分布式搜索方案。

SolrCloud是Solr的基于Zookeeper一种部署方式。Solr可以以多种方式部署，例如单机方式，多机Master-Slaver方式。

原理图：

SolrCloud

SolrCloud有几个特色功能：

集中式的配置信息使用ZK进行集中配置。启动时可以指定把Solr的相关配置文件上传

Zookeeper，多机器公用。这些ZK中的配置不会再拿到本地缓存，Solr直接读取ZK中的配置信息。配置文件的变动，所有机器都可以感知到。另外，Solr的一些任务也是通过ZK作为媒介发布的。目的是为了容错。接收到任务，但在执行任务时崩溃的机器，在重启后，或者集群选出候选者时，可以再次执行这个未完成的任务。

自动容错SolrCloud对索引分片，并对每个分片创建多个Replication。每个Replication都可以对外提供服务。一个Replication挂掉不会影响索引服务。更强大的是，它还能自动的在其它机器上帮你把失败机器上的索引Replication重建并投入使用。

近实时搜索立即推送式的replication（也支持慢推送）。可以在秒内检索到新加入索引。

查询时自动负载均衡SolrCloud索引的多个Replication可以分布在多台机器上，均衡查询压力。如果查询压力大，可以通过扩展机器，增加Replication来减缓。

自动分发的索引和索引分片发送文档到任何节点，它都会转发到正确节点。

事务日志事务日志确保更新无丢失，即使文档没有索引到磁盘。

### 四、Solr

简介：Solr是一个独立的企业级搜索应用服务器，它对外提供类似于Web-service的API接口。用户可以通过http请求，向搜索引擎服务器提交一定格式的XML文件，生成索引；也可以通过Http Get操作提出查找请求，并得到XML格式的返回结果。

Solr

Solr是一个高性能，采用Java5开发，基于Lucene的全文搜索服务器。同时对其进行扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。

官网：<https://lucene.apache.org/solr/>

### 五、ElasticSearch

简介：ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是第二最流行的企业搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

官网：<http://www.elasticsearch.org/>

### 六、Sphinx

简介：Sphinx是一个基于SQL的全文检索引擎，可以结合MySQL,PostgreSQL做全文搜索，它可以提供比数据库本身更专业的搜索功能，使得应用程序更容易实现专业化的全文检索。Sphinx特别为一些脚本语言设计搜索API接口，如PHP,Python,Perl,Ruby等，同时为MySQL也设计了一个存储引擎插件。

Sphinx单一索引最大可包含1亿条记录，在1千万条记录情况下的查询速度为0.x秒（毫秒级）。Sphinx创建索引的速度为：创建100万条记录的索引只需3~4分钟，创建1000万条记录的索引可以在50分钟内完成，而只包含最新10万条记录的增量索引，重建一次只需几十秒。

官网：<http://sphinxsearch.com>

### 七、SenseiDB

贡献者：linkedin

简介：SenseiDB是一个NoSQL数据库，它专注于高更新率以及复杂半结构化搜索查询。熟悉Lucene和Solr的用户会发现，SenseiDB背后有许多似曾相识的概念。SenseiDB部署在多节点集群中，其中每个节点可以包括N块数据片。Apache Zookeeper用于管理节点，它能够保持现有配置，并可以将任意改动（如拓扑修改）传输到整个节点群中。SenseiDB集群还需要一种模式用于定义将要使用的数据模型。

从SenseiDB集群中获取数据的唯一方法是通过Gateways（它没有“INSERT”方法）。每个集群都连接到一个单一gateway。你需要了解很重要的一点是，由于SenseiDB本身没法处理原子性（Atomicity）和隔离性（Isolation），因此只能通过外部在gateway层进行限制。另外，gateway必须确保数据流按照预期的方式运作。内置的gateway有以下几种形式：

来自文件 来自JMS队列 通过JDBC 来自Apache Kafka 官网：<http://senseidb.com>

## 数据挖掘

### 一、Mahout

#### Mahout

简介：Apache Mahout 是 Apache Software Foundation (ASF) 开发的一个全新的开源项目，其主要目标是创建一些可伸缩的机器学习算法，供开发人员在 Apache 在许可下免费使用。该项目已经发展到了它的第二个年头，目前只有一个公开发行版。Mahout 包含许多实现，包括集群、分类、CP 和进化程序。此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中。

虽然在开源领域中相对较为年轻，但 Mahout 已经提供了大量功能，特别是在集群和 CF 方面。Mahout 的主要特性包括：

Taste CF。Taste 是 Sean Owen 在 SourceForge 上发起的一个针对 CF 的开源项目，并在 2008 年被赠予 Mahout。一些支持 Map-Reduce 的集群实现包括 k-Means、模糊 k-Means、Canopy、Dirichlet 和 Mean-Shift。Distributed Naive Bayes 和 Complementary Naive Bayes 分类实现。针对进化编程的分布式适用性功能。Matrix 和矢量库。上述算法的示例。官网：<http://mahout.apache.org/>

## IaaS

IaaS（Infrastructure as a Service），即基础设施即服务。

### 一、OpenStack

简介：OpenStack是一个由NASA（美国国家航空航天局）和Rackspace合作研发并发起的，以Apache许可证授权的自由软件和开放源代码项目。

OpenStack是一个开源的云计算管理平台项目，由几个主要的组件组合起来完成具体工作。OpenStack支持几乎所有类型的云环境，项目目标是提供实施简单、可大规模扩展、丰富、标准统一的云计算管理平台。OpenStack通过各种互补的服务提供了基础设施即服务（IaaS）的解决方案，每个服务提供API以进行集成。

## OpenStack

6个核心项目：Nova（计算，Compute），Swift（对象存储，Object），Glance（镜像，Image），Keystone（身份，Identity），Horizon（自助门户，Dashboard），Quantum & Melange（网络&地址管理），另外还有若干社区项目，如Rackspace（负载均衡）、Rackspace（关系型数据库）。

## 二、Docker

贡献者：dotCloud

### Docker

简介：Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）。几乎没有性能开销，可以很容易地在机器和数据中心中运行。最重要的是，他们不依赖于任何语言、框架或包括系统。

官网：<http://www.docker.io/>

## 三、Kubernetes

贡献者：Google

简介：Kubernetes是Google开源的容器集群管理系统。它构建在Docker技术之上，为容器化的应用提供资源调度、部署运行、服务发现、扩容缩容等整一套功能，本质上可看作是基于容器技术的mini-PaaS平台。

Kubernetes从另一个角度对资源进行抽象，它让开发人员和管理人员共同着眼于服务的行为和性能的提升，而不是仅仅关注对单一的组件或者是基础资源。

那么Kubernetes集群到底提供了哪些单一容器所没有功能？它主要关注的是对服务级别的控制而并非仅仅是对容器级别的控制，Kubernetes提供了一种“机智”的管理方式，它将服务看成一个整体。在Kubernetes的解决方案中，一个服务甚至可以自我扩展，自我诊断，并且容易升级。例如，在Google中，我们使用机器学习技术来保证每个运行的服务的当前状态都是最高效的。

代码托管：<https://github.com/GoogleCloudPlatform/kubernetes/>

#### 四、Imctfy

贡献者：Google

简介：Google开源了自己所用Linux容器系统的开源版本Imctfy，读音为lem-kut-fee。包括一个C++库（使用了C++11，文档可以参考头文件）和命令行界面。目前的版本是0.1，只提供了CPU与内存隔离。项目还在密集开发中。

mctfy本身是针对某些特定使用场景设计和实现的，目前拥有一台机器上所有容器时运行情况最好，不推荐与LXC和其他容器系统一起使用（虽然也可行）。已在Ubuntu 12.04+和Ubuntu 3.3与3.8内核上测试。

代码托管：<https://github.com/google/Imctfy/>

#### 监控管理 一、Dapper

贡献者：Google

简介：Dapper是一个轻量的ORM(对象关系映射（英语：Object Relational Mapping，简称ORM，或O/RM，或O/R mapping）。并不单纯的是一个DBHelper。因为在Dapper中数据其实就是一个对象。Dapper扩展与 IDbConnection上，所以事实上它的侵入性很低。我用了StructureMap。如果不喜欢单独使用，或者自己实现下。

代码就一个SqlMapper.cs文件，主要是IDbConnection的扩展方法，编译后就40K的一个很小的dll。

特性：

Dapper很快。Dapper的速度接近与IDataReader。Dapper支持主流数据库Mysql,SQLite,Mssql2000,Mssql2005,Oracle等一系列的数据库 支持多表并联的对象。支持一对多 多对多的关系，并且没侵入性。原理通过Emit反射IDataReader的序列队列，来快速的得到和产生对象 Dapper语法十分简单。并且无须迁就数据库的设计 官方站点 <http://code.google.com/p/dapper-dot-net/>

代码托管：<http://bigbully.github.io/Dapper-translation/>

#### 二、Zipkin

贡献者：Twitter

简介：Zipkin（分布式跟踪系统）是Twitter的一个开源项目，允许开发者收集Twitter各个服务上的监控数据，并提供查询接口。该系统让开发者可通过一个Web前端轻松的收集和分析数据，例如用户每次请求服务的处理时间等，可方便的监测系统中存在的瓶颈。

### Zipkin

官方网站：<http://twitter.github.io/zipkin/>

代码托管：<https://github.com/twitter/zipkin>

## 数据交易

代表：数据堂

数据堂成立于2011年，总部位于北京，它其实最开始是由科研和人工智能大数据服务商发展而来的，是国内首家专注于互联网综合数据交易和服务的公司。提供数据定制、数据云服务、数据交易等服务，致力于融合和盘活各类大数据资源，实现数据价值最大化，推动相关技术、应用和产业的创新。

目前，数据堂的业务领域拓展到包括到智慧交通、健康医疗、金融征信、政府大数据运营等诸多领域，逐渐形成多业务多模式立体化的集团式发展。已成功为国内外多家企业提供数据定制服务，包括百度、腾讯、阿里巴巴、奇虎360、联想、Microsoft、NEC、Canon、Intel、Samsung、Nuance、Fujitsu等。

产品分析：

数据堂的产品可以概括为两个方面，一是面向B端客户提供定制化数据源服务，二是大数据交易平台。数据堂的B端客户包括百度、腾讯、阿里巴巴等，主要业务有代采集、处理和制作数据或出售和租赁数据。数据堂的C端客户是需要数据的个体，可以通过数据交易平台购买和租赁数据。

其中大数据交易平台的目标是打通数据拥有方和需求方的需求，通过数据拥有方合作，积累了征信、交通、健康、医疗等数十个领域的数据集。第二步数据加工处理分析，把分析结果放到云上，当前服务BAT在内的一千家企业以上。

## 金融大数据

代表：BBD(数联铭品)

BBD(数联铭品)成立于2013年，至今才2年多时间，但发展势头迅速。作为大数据金融风险管理专家，这家公司还是商业大数据行业标准COSR的制定者。除了位于成都的总部，还在北京、上海、深圳和杭州设有分支机构，并在香港和新加坡设有子公司。2015年，BBD被评为国家高新技术企业，还拿到中国人民银行审批通过的《企业征信业务经营备案证》。

在大数据领域，有“南周北孙”的说法。“南周”就是指BBD创始人兼首席科学家周涛。这位27岁时就当上教授的明星科学家，刚刚当选央视年度科技创新人物，与屠呦呦一同获奖。

截至目前，BBD已覆盖2300万家企业法人主体的基本数据，建立了最全面的工商、诉讼、专利、招聘、社交、招中标数据库，尤其是对诉讼数据进行了深度挖掘，所采集的数据全部来自于公开数据。

服务对象包括银行、会计师事务所、律所、投资机构、征信评级机构、金融信息终端、媒体和咨询机构等，典型客户有毕马威、普华永道、长沙银行、重庆银行、中证信用、中经社、《财富》(中文版)、财新传媒、四川省旅游局等。

产品分析：

目前国际上领先的金融服务机构缺少标准的方法论和模型，对轻资产高研发高成长的新经济企业进行评级和分析。BBD提供可靠且可验证的金融大数据和技术方案，基于大数据框架和国际标准建立中国特色的Fin-Tech，建设新经济框架下，中小企业信用体系尤其是轻资产高研发高成长企业信用体系。

数据科学家和金融科学家组成的团队，研究了一套新经济企业的行为模型，建立了一个企业行为数据库，模型分为七个维度：企业的基本信息;行业数据信息;法人治理结构信息;关联方信息;财务和非财务KPI;社交媒体信息;无形资产和资产质押数据。

BBD旗下的明星产品HIGGS Credit是一款企业全息画像搜索引擎，依托于全面的企业数据库，可以提供企业DNA全息画像、企业行为KPI数据与行业比对、实时动态的企业尽职调查数据、关联方异动数据监测等数据服务。公司还有BBD Finance，BBD Index，BBD Anti-Fraud, BBD Innovation、BBD Credit等针对不同领域的多条产品线。

## 工业大数据

代表：美林数据

如果要给美林贴上一个标签的话，应该是“工业大数据”。这家公司位于西安，旗下有数据分析产品、行业大数据解决方案、数据运营服务三大核心业务。公司深耕大数据行业应用，面向电力、军工制造、金融领域提供定制化大数据落地解决方案。深度参与国家“中国制造2025”、“一带一路”、“大数据”战略计划，并成立国内首家“一带一路大数据交易所”，主要客户群为电力、军工制造、金融等相关领域，是国内首家军工制造业大数据落地方案提供商。

产品分析：

去年5月，美林数据发布国内首款集数据可视化探索与数据深度发掘功能于一体的大数据分析平台(Tempo-DataAnalysis)，面向企业不同领域和不同层级的数据分析、价值应用人员，提供数据可视化探索、数据深度分析和数据应用开发的一体化服务，持续为用户打造“专业、敏捷、易用”的产品体验。

该平台的特点在于可视化(分析过程可视化、分析结果可视化、高维数据可视化)、智能化(数据模型自动识别、数图智能匹配)、增值化(发掘数据规律、获取商业洞察力、创造商业价值)、大数据支持(分布式并行计算、内存计算)、多数据源接入支持(文件数据格式、关系型数据库、HDFS、H)等。

美林的大数据产品线还包括数据挖掘平台、统计分析平台、企业门户平台、数据资源管理平台等。

## 营销大数据

代表：华院数据

华院数据位于上海，提供基于数据挖掘的面向营销分析和管理、客户关系管理和决策支持的应用软件和咨询解决方案。

这家成立于2002年的企业，一直为传统运营商、银行、保险、航空等具有海量数据的机构提供数据挖掘和分析解决方案。几年前，随着数据采集、存储和传输技术的发展，华院开始了以数据分析为核心的多元化尝试，将足迹延伸到电商、制造业、医疗、安防、物流等更多的新兴行业。

华院也着力于产业大数据生态孵化，培育了数云、数创、数真等十余家围绕垂直行业的大数据企业。

产品分析：

包括大数据运营视窗系统、精准营销系统、智能推荐引擎、标签管理系统等。其中大数据视窗系统通过对企业数据及业务规则的深度挖掘分析，将企业的业务流、信息流、数据流进行全面整合分析，借助最新的数字可视化技术，为企事业管理层建立统一的运营视窗，快速洞察企业运营中的风险和瓶颈。

值得一提的是，华院旗下的数云，现已成为淘宝上卖得最贵的电商CRM，数云从2011年起步，现已拓展到为上千家零售企业提供大数据营销服务，帮助品牌管理的会员超过1.5亿，从品牌层面帮助零售企业构建以客户为核心的大数据管理、分析、营销应用平台，收集和挖掘大数据的价值，拓宽流量渠道，帮助企业实现数据化营销、智能化管理。

## 一 **github**相关资源收集

### Hadoop

- Apache Tez – 它是一个针对Hadoop数据处理应用程序的新分布式执行框架，该框架基于YARN；
- SpatialHadoop – SpatialHadoop是Apache Hadoop的MapReduce扩展，专门用于处理空间数据；
- GIS Tools for Hadoop – 用于Hadoop框架的大数据空间分析；
- Elasticsearch Hadoop – Elasticsearch与Hadoop深度集成，可用于实时搜索和分析，支持Map/Reduce、Cascading、Apache Hive和Apache Pig；
- dumbo - Python模块，使Hadoop程序的编写和运行更为容易；
- hadoopy – 用Cython写的Python MapReduce库；
- mrjob - mrjob是一个Python2.5+程序包，可以帮助编写和运行Hadoop工作流；
- pydoop - 为Hadoop提供Python API的程序包；
- hdfs-du - Hadoop分布式文件系统（HDFS）的交互可视化；
- White Elephant - Hadoop的日志聚合器和仪表板；
- Genie - Genie提供REST-ful API，以便运行Hadoop、Hive和Pig jobs，还管理多个Hadoop资源，并在它们之间进行作业提交；
- Apache Kylin – 最初来自eBay公司的开源分布式分析引擎，能提供Hadoop之上的SQL查询接口及多维分析（OLAP），以支持超大规模数据集；
- Crunch - 基于Go的工具包，用于在Hadoop上的ETL和特征提取；
- Apache Ignite - 分布式内存平台。

### YARN

- Apache Slider - Apache Slider是Apache软件基金会的孵化项目，旨在能够轻松地实现现有应用程序到YARN集群的部署；
- Apache Twill - Apache Twill是Apache Hadoop® YARN的抽象层，降低了开发分布式应用程序的复杂度，让开发者更专注于自己的应用逻辑；
- mpich2-yarn – 在YARN上运行MPICH2。

## NoSQL

- Apache HBase - Apache HBase；
- Apache Phoenix – Hbase的SQL驱动，支持辅助索引；
- happybase -一个开发者友好型的Python库，用于Apache HBase的交互；
- Hannibal –用于监测和维护HBase 集群的工具；
- Haeinsa –用于HBase的线性可扩展多行多表交易库；
- hindex – Hbase的辅助索引；
- Apache Accumulo - Apache Accumulo可排序分布式键/值存储，是一个强大的、可扩展高性能数据存储和检索
- OpenTSDB -可扩展时间序列数据库；
- Apache Cassandra

## Hadoop 中的SQL

- Apache Hive
- Apache Phoenix - Hbase的SQL驱动，支持辅助索引；
- Pivotal HAWQ – Hadoop上的并行数据库；
- Lingual -用于级联的SQL接口（MR / TEZ工作发生器）；
- Cloudera Impala
- Presto –用于大数据的分布式SQL查询引擎，该查询引擎由Facebook开发，现已开源；
- Apache Tajo - Apache Hadoop的数据仓库系统；
- Apache Drill

## 数据管理

- Apache Calcite -动态数据管理框架；
- Apache Atlas -用于元数据标记及类群捕获，支持复杂的商业数据分类。

## 工作流，生命周期及管理

- Apache Oozie - Apache Oozie；
- Azkaban
- Apache Falcon -数据管理与处理平台；

- Apache NiFi - 数据流系统；
- AirFlow – AirFlow是以编程方式建立、调度和监控数据管道的平台；
- Luigi - Python包，用于构建批处理作业的复杂管道。

## 数据提取及整合

- Apache Flume - Apache Flume；
- Suro - Netflix分布式数据管道；
- Apache Sqoop - Apache Sqoop；Apache Kafka - Apache Kafka；Gobblin from LinkedIn – Hadoop的通用数据提取框架；

## DSL

- Apache Pig - Apache Pig
- Apache DataFu – Hadoop中用于处理大规模数据的库的集合；
- vahara – 基于Apache Pig的机器学习和自然语言处理；
- packetpig - 用于开源大数据安全性分析；
- akela – Mozilla的实用工具库，用于Hadoop、HBase、Pig等等；
- seqpig - Hadoop中用于大型定序数据集的简单可扩展脚本（bioinfomation除外）；
- Lipstick – Pig工作流程可视化工具；A(pache)的Lipstick简介；
- PigPen - PigPen 是Clojure或分布式Clojure的Map-reduce，能够编译Apache Pig，但是不需要过多了解Pig也可以使用PigPen。

## 库和工具

- Kite Software Development Kit – 一组库、工具、示例和文档；
- gohadoop - Apache Hadoop YARN的本地Go客户端；
- Hue – 用Apache Hadoop分析数据的Web界面；
- Apache Zeppelin - 基于Web的笔记，可进行交互式数据分析；
- Jumbune - Jumbune是为分析Hadoop集群和MapReduce作业而构建的开源产品；
- Apache Thrift
- Apache Avro - Apache Avro是一个数据序列化系统；
- Elephant Bird – Twitter中LZO、缓冲协议相关的Hadoop、Pig、Hive和HBase

代码的集合；

- Spring for Apache Hadoop
- hdfs - A native go client for HDFS
- Oozie Eclipse Plugin - Eclipse中用于编辑Apache Oozie工作流的图形编辑器。

## 实时数据处理

- Apache Storm
- Apache Samza
- Apache Spark
- Apache Flink - Apache Flink是高效的分布式通用数据处理的平台，用于精准的流处理。

## 分布式计算和编程

### Apache Spark

- Spark Packages - Apache Spark中程序包的community（社区）索引；
- SparkHub - Apache Spark的社区；
- Apache Crunch
- Cascading - Cascading是在Hadoop上构建数据应用的成熟的应用开发平台；
  - Apache Flink - Apache Flink是高效的分布式通用数据处理的平台；
  - Apache Apex (incubating) -企业级的统一流处理和批处理引擎。

## 包装，配置与监测

- Apache Bigtop - 用于Apache Hadoop生态系统的包装和测试；
- Apache Ambari - Apache Ambari
- Ganglia Monitoring System
- ankush -一个大数据集群管理工具，用于创建和管理不同的技术集群；
- Apache Zookeeper - Apache Zookeeper
- Apache Curator - 用于ZooKeeper的客户端简化包装和丰富ZooKeeper框架；
- Buildoop - Hadoop生态系统生成器；
- Deploop - Hadoop的部署系统；
- Jumbune -一个用于开源MapReduce分析，MapReduce流程调试，HDFS数据

- 质量校验和Hadoop集群监测的工具；
- inviso - Inviso是一个轻量级的工具，它提供搜索Hadoop作业，可视化性能，查看集群利用率的能力。

## 搜索

- ElasticSearch
- SenseiDB
- Apache Solr -开源、分布式、实时、半结构化的数据库；
- Banana - Apache Solr的Kibana端口。

## 搜索引擎框架=

- Apache Nutch –Apache Nutch是一个高度可扩展的，可伸缩的开源网络爬虫软件项目。

## 安全性

- Apache Ranger - Ranger是一个框架，能够跨Hadoop平台启用、监控和全面管理数据安全性；
- Apache Sentry - Hadoop的一个授权模块；
- Apache Knox Gateway –用于与Hadoop集群交互的REST API网关。

## 基准

- Big Data Benchmark
- HiBench
- Big-Bench
- hive-benchmarks
- hive-testbench –一个测试平台，用于进行任何规模数据的Apache Hive实验；
- YCSB -雅虎云服务基准（YCSB）是一个开源规范和程序套件，用于评估计算机程序的检索和维护功能；它常被用于比较NoSQL数据库管理系统的相对性能。

## 机器学习和大数据分析

- Apache Mahout
- Oryx 2 –基于Spark、Kafka的Lambda架构，用于实时大规模的机器学习；
- MLlib - MLlib是Apache Spark的可扩展机器学习库；
- R - R是用于统计计算和图形的自由软件环境；
- RHadoop -包括RHDFS、RHBase、RMR2和plyrnr；
- RHive –用于从R中开始Hive查询；
- Apache Lens

## Hive Plugins

```
http://nexr.github.io/hive-udf/
https://github.com/edwardcapriolo/hive_cassandra_udfs
https://github.com/livingsocial/HiveSwarm
https://github.com/ThinkBigAnalytics/Hive-Extensions-from-Thin
k-Big-Analytics
https://github.com/karthkk/udfs
https://github.com/twitter/elephant-bird - Twitter
https://github.com/lovelysystems/ls-hive
https://github.com/stewi2/hive-udfs
https://github.com/klout/brickhouse
https://github.com/markgrover/hive-translate (PostgreSQL trans
late())
https://github.com/deanwampler/HiveUDFs
https://github.com/myui/hivemall (Machine Learning UDF/UDAF/UD
TF)
https://github.com/edwardcapriolo/hive-geoip (GeoIP UDF)
https://github.com/Netflix/Surus
```

## Storage Handler

```
https://github.com/dvasilen/Hive-Cassandra  
https://github.com/yc-huang/Hive-mongo  
https://github.com/balshor/gdata-storagehandler  
https://github.com/karthkk/hive-hbase-json  
https://github.com/sunsuk7tp/hive-hbase-integration  
https://bitbucket.org/rodrigopr/redisstoragehandler  
https://github.com/zhuGuangBin/HiveJDBCStorageHanlder  
https://github.com/chimpler/hive-solr  
https://github.com/bfemiano/accumulo-hive-storage-manager  
https://github.com/rcongiu/Hive-JSON-Serde  
https://github.com/mochi/hive-json-serde  
https://github.com/ogrodnek/csv-serde  
https://github.com/parag/HiveJsonSerde  
https://github.com/johanoskarsson/hive-json-serde  
https://github.com/electrum/hive-serde - JSON  
https://github.com/karthkk/hive-hbase-json
```

## Libraries and tools

```
https://github.com/forward3d/rbhive  
https://github.com/synctree/activerecord-hive-adapter  
https://github.com/hrp/sequel-hive-adapter  
https://github.com/forward/node-hive  
https://github.com/recruitcojp/WebHive  
shib - WebUI for query engines: Hive and Presto  
clive - Clojure library for interacting with Hive via Thrift  
https://github.com/anjuke/hwi  
https://code.google.com/a/apache-extras.org/p/hipy/  
https://github.com/dmorel/Thrift-API-HiveClient2 (Perl - HiveServer2)  
PyHive - Python interface to Hive and Presto  
https://github.com/recruitcojp/OdbcHive
```

## 第二十章 Hadoop100问

Q：大数据只代表那些量很大的数据吗？

A: 虽然从名字上看是这样，但是实际上我们用“大数据”来形容因为某种原因无法适应传统数据库软件工具的数据，而这些软件工具在过去的数十年间一直被用于分析和商业智能。举个例子，大数据也许无法完全适应关系型数据库(例如图像的像素数据)，或者需要经过特别的处理才能和其他数据共同使用(例如从机器设备获得的时间序列数据)。

Q: 我们在油气行业不是一直都在用大数据吗？

A: 是的！地震探测和历史学家储存的传感器数据就是两个很好的例子。早期，由于这些数据量很大而难以处理，在典型的数据库工具中表现并不出色，所以我们就将它限制在了预定义的工作流和应用单元当中。结果是我们不知不觉地限制了自己寻求关键业务问题的准确答案的能力。现在的大数据运动都是为了实现以全新的方式去应用这些棘手的、对运营提出了挑战的数据，从而获取更多问题的答案。

Q: 当前的大数据运动究竟在做什么呢？

A: 都是为了实现所有数据的自由支配——不管它是图像、视频、音频、自然语言文本、机器可读文本、传感器数据还是平常的数据库中的老式关系型数据，不管数据量是兆字节还是兆兆字节，不管信息来源是实时的快照还是不断流入的数据流。

Q: 但是要怎么实现呢？关键是这些数据很难管理啊

A: 相比“传统的”数据而言，我们可以采取各种不同的IT解决方案来管理并查询这些数据。我们可以从Yahoo、Google、eBay等互联网企业身上学到很多，他们都是新型工具和技术的领导者。他们每天用到的数据和油气行业一直以来卖力管理的数据非常相似。他们每天都要检查兆兆字节的网络服务器日志，加深对客户交互的理解；还对社交媒体内容应用了自然语言处理和情感倾向分析；物联网的发展带入了更多FitBit和苹果智能手表这样的“可穿戴设备”，所以传感器数据也是他们的重点关注之一。

Q: 我们为什么要做什么大数据？

A: 为什么不呢？我们都知道，油气行业的风险非常高，因为一次油气开采的花费可以高达70亿美元，所以必须根据数据进行商业决策，不能凭直觉拍脑袋。在交通运输行业中，传感器数据（一种大数据的来源）可以检测引擎行为，并且可以结合引擎性能和引擎或车辆的主数据，例如修理历史、服务和利用历史等数据（在大数据出现

之前他们拥有的所有数据来源)，方便运营商准确预测引擎故障的时间。对于火车、航空、快递公司而言，这意味着他们可以组织故障车辆进行预防性维护，而不是坐以待毙，让车辆在路上发生故障，使旅客、运货发生滞留。

## 第二十一章 大数据相关公司收集

# 1. 电商网站架构案例

## 第二十二章 运维

## 1.运维常用工具

### Bootstrapping：

Kickstart、Cobbler、rpmbuild/xen、kvm、lxc、Openstack、Cloudstack、Opennebula、Eucalyplus、RHEV

### 配置类工具：

Capistrano、Chef、puppet、func、salstack、Ansible、rundeck、CFengine、Rudder

### 自动化构建和测试：

Ant、Maven、Selenium、PyUnit、QUnit、JMeter、Gradle、PHPUnit

### 监控类工具：

Cacti、Nagios(Icinga)、Zabbix、基于时间监控前端Grafana、Mtop、MRTG(网络流量监控图形工具)、Monit、Diamond+Graphite+Grafana

### 微服务平台：

OpenShift、Cloud Foundry、Kubernetes、Mesosphere

### 性能监控工具：

dstat(多类型资源统计)、atop(htop/top)、nmon(类Unix系统性能监控)、slabtop(内核slab缓存信息)、sar(性能监控和瓶颈检查)、sysdig(系统进程高级视图)、tcpdump(网络抓包)、iftop(类似top的网络连接工具)、iperf(网络性能工具)、smem(高级内存报表工具)、collectl(性能监控工具)、TCP优化监控工具tcpdive

### 免费APM工具：

mmtrix(见过的最全面的分析工具)、alibench、JAVA性能监控pinpoint

**进程监控:**

monit、Supervisor、frigga、StrongLoop Process Manager

**日志系统:**

Logstash、Scribe

**绘图工具:**

RRDtool、Gnuplot

**流控系统:**

Panabit、在线数据包分析工具Pcap Analyzer

**安全检查:**

chrootkit、rkhunter

**PaaS :**

Cloudify、Cloudfoundry、Openshift、Deis（Docker、CoreOS、Atomic、ubuntu core/Snappy、RancherOS）

**Troubleshooting:**

Sysdig、Systemtap、Perf

**服务发现：**

SmartStack、etcd

## 持续集成:

Go、Jenkins、Gitlab、facebook代码审查工具phabricator、spinnaker

## APP CD:

fastlane

## 磁盘压测:

fio、iozone、IOMeter(win) Memcache      Mcrouter(scaling memcached)  
Redis      Dynomite、Twemproxy、codis/SSDB/Aerospike、Redis Cluster

## MySQL 监控:

mytop、orzdba、Percona-toolkit、Maatkit、innotop、myawr、SQL级监控  
mysqlpcap、拓扑可视化工具

## MySQL 基准测试:

mysqlsla、sql-bench、Super Smack、Percona's TPCC-MYSQL Tool、sysbench

## MySQL Proxy:

SOHU-DBProxy、Mycat、Altas、cobar、58同城Oceanus、kingshard

## MySQL 逻辑备份工具:

mysqldump、mysqlhotcopy、mydumper、MySQLDumper 、mk-parallel-dump/mk-parallel-restore

## MySQL 物理备份工具:

Xtrabackup、LVM Snapshot

## MongoDB压测：

iibench&sysbench

## 第二十三章 机器学习入门

### 入门文章

一文读懂机器学习，大数据/自然语言处理/算法全有了

## 第二十四章 **Centos**下的**Oracle 11g**安装

### **Oracle**安装文档

本次安装是在自己所建的虚拟机上安装的，用到了视图链接工具VNC

#### 1.本次所用安装包：

linux.x64\_11gR2\_database\_1of2.zip、linux.x64\_11gR2\_database\_2of2.zip

#### 2.参考链接如下：

Oracle安装参考链接：<http://www.linuxidc.com/Linux/2015-02/113222.html>

VNC安装参考链接：<http://www.linuxidc.com/Linux/2015-01/112326.html> Oracle

11G CentOS 6.5: <http://www.aiplaypc.com/229.html>

<http://blog.csdn.net/cafardhaibin/article/details/25071249>

#### 3.安装前的配置：

##### (1)安装前的基本准备工作：

1. 安装VMware Workstation
2. 安装CentOS，主机命名为：oracledb
3. 磁盘需要大于30G（经验值）
4. 内存必须大于1G（官方要求）
5. 操作系统swap分区大于2G（如果物理内存小于2G，则需要设置，设置值为物理内存的1-2倍，如果物理内存大于2G，则无需设置。）
6. 虚拟机网络连接方式：桥接模式(B)直接连接物理网络
7. 安装完成后设置虚拟机网络(ipv4)为固定IP地址(system-config-network)
8. 进行网络测试OK，则操作系统环境准备完毕
9. 安装虚拟机时一定要选择：先创建虚拟机后安装操作系统
10. 为了安装Oracle，故选择安装类型为：桌面版本。

## 11. 安装SSH Secure Shell Client并连接主机

### (2) 基本的主机配置

1. 以下步骤中的命令太长的可通过：SSH Secure Shell Client 直接复制进行
2. vi基本命令：i--编辑状态 退出编辑并保存时先按ESC键，再按符合“:wq”或者":x"即可
3. 注意每个步骤时的当前用户，是root还是oracle

step-1#修改主机名

```
[root@oracledb ~]# sed -i "s/HOSTNAME=localhost.localdomain/HOSTNAME=oracledb/" /etc/sysconfig/network  
[root@oracledb ~]# hostname oracledb
```

step-2#添加主机名与IP对应记录

```
[root@oracledb ~]# vi /etc/hosts  
192.168.1.8      oracledb
```

step-3#关闭防火墙Selinux

```
[root@oracledb ~]# sed -i "s/SELINUX=enforcing/SELINUX=disabled/" /etc/selinux/config  
[root@oracledb ~]# setenforce 0
```

修改网络配置

1. 修改网卡配置 编辑：vi /etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE=eth0 #描述网卡对应的设备别名，例如ifcfg-eth0的文件中它为eth0  
#设置网卡获得ip地址的方式，可能的选项为static，dhcp或bootp，分别对应静态  
指定的 ip地址，通过dhcp协议获得的ip地址，通过bootp协议获得的ip地址  
BOOTPROTO=static  
BROADCAST=192.168.0.255 #对应的子网广播地址  
HWADDR=00:07:E9:05:E8:B4 #对应的网卡物理地址  
IPADDR=192.168.1.147 #如果设置网卡获得 ip地址的方式为静态指定，此字段就  
指定了网卡对应的ip地址  
NETMASK=255.255.255.0 #网卡对应的网络掩码  
NETWORK=192.168.1.0 #网卡对应的网络地址
```

2. 修改网关配置

编辑：vi /etc/sysconfig/network 修改后如下：

```
NETWORKING=yes(表示系统是否使用网络，一般设置为yes。如果设为no，则不能使  
用网络，而且很多系统服务程序将无法启动)  
HOSTNAME=centos(设置本机的主机名，这里设置的主机名要和/etc/hosts中设置的  
主机名对应)  
GATEWAY=192.168.0.1(设置本机连接的网关的IP地址。)
```

我在修改这里打开编辑时前三项已经默认有了所以只增加了GATEWAY

3. 修改DNS 配置

编辑：vi /etc/resolv.conf 修改后如下：

nameserver 即是DNS服务器 IP地址，第一个是首选，第二个是备用。

4. 重启网络服务

执行命令：

service network restart 或 /etc/init.d/network restart

(3)安装以下**RPM**软件包（加**32bit**括号注解的是**32位**版本的软件包，对同名未加注解的则是**64位**版本的该软件包。在**64位**版本平台上，两种版本都要安装）

```
binutils-2.17.50.0.6
compat-libstdc++-33-3.2.3
compat-libstdc++-33-3.2.3 (32 bit)
elfutils-libelf-0.125
elfutils-libelf-devel-0.125
gcc-4.1.2
gcc-c++-4.1.2
glibc-2.5-24
glibc-2.5-24 (32 bit)
glibc-common-2.5
glibc-devel-2.5
glibc-devel-2.5 (32 bit)
glibc-headers-2.5
ksh-20060214
libaio-0.3.106
libaio-0.3.106 (32 bit)
libaio-devel-0.3.106
libaio-devel-0.3.106 (32 bit)
libgcc-4.1.2
libgcc-4.1.2 (32 bit)
libstdc++-4.1.2
libstdc++-4.1.2 (32 bit)
libstdc++-devel 4.1.2
make-3.81
sysstat-7.0.2
```

直接执行以下的**yum**安装命令，若已安装会有提示

```
yum install -y binutils  
yum install -y compat-libstdc  
yum install -y elfutils-libelf  
yum install -y gcc  
yum install -y glibc  
yum install -y ksh  
yum install -y libaio  
yum install -y libgcc  
yum install -y libstdc  
yum install -y make  
yum install -y sysstat  
yum install libXp -y  
yum install -y glibc-kernheaders
```

或者通过以下方法更加快一些：

```
yum install gcc libaio libaio-devel libstdc++ libstdc++-devel li  
bgcc elfutils-libelf-devel glibc-devel glibc-devel gcc-c++ compa  
t-libstdc++-33 unixODBC unixODBC-devel
```

有一个rpm包需要独立下载pdksh-5.2.14-37.el5\_8.1.x86\_64，然后rpm -ivh安装即可。下载地址:[pdksh-5.2.14-37.el5\\_8.1.x86\\_64](#)

## (4) 调整内核参数及用户限制

以**root**用户进行配置文件的编辑

I. 编辑**/etc/sysctl.conf**文件，设置相关参数的系统默认值。如果该文件中已有相关参数的设置，则确保参数值不小于如下对应值；如果没有相关参数的设置，则按照如下格式添加相应的参数设置行

```
[root@oracledb ~]#vim /etc/sysctl.conf

fs.aio-max-nr = 1048576
fs.file-max = 6815744
kernel.shmall = 2097152
kernel.shmmmax = 536870912
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
```

[root@oracledb ~]# sysctl -p (备注：用于输出配置后的结果，如果有错误会提示)

## II. 编辑/etc/security/limits.conf文件，修改操作系统对**oracle**用户资源的限制。在该文件中添加如下行

```
[root@localhost ~]#vim /etc/security/limits.conf

oracle      soft    nproc  2047
oracle      hard    nproc  16384
oracle      soft    nofile 1024
oracle      hard    nofile 65536
oracle      hard    stack  10240
```

## (5) 目录结构及空间规划可查看参考链接

<http://www.linuxidc.com/Linux/2015-02/113222.htm>

## (6) 数据库安装用户和组的创建

使用**root**用户，进行如下操作：

创建**oinstall**组

```
[root@localhost ~]#groupadd -g 5000 oinstall
```

创建dba组

```
[root@localhost ~]#groupadd -g 501 dba
```

创建oracle用户

```
[root@localhost ~]#useradd -g oinstall -G dba oracle (执行后会在/home目录下创建oracle次目录)
```

## (7) 创建相应的文件系统（或安装目录）并改变相应的权限（记住路径，方便之后的查找）

```
[root@localhost ~]# cd /home/oracle  
[root@localhost oracle]# mkdir -p /home/oracle/app/oracle  
[root@localhost oracle]# chown -R oracle:oinstall /home/oracle/*  
[root@localhost oracle]# chmod -R 775 /home/oracle/*  
[root@localhost oracle]#
```

（执行命令后将Oracle安装包用Xftp导入到/home/oracle目录下解压）

## (8) 数据库安装用户的**profile**文件的设置

使用**oracle**用户进行如下操作

假设数据库（实例）名为**powerdes**。编

辑/home/oracle/.bash\_profile（./bash\_profile文件是解压包解压后生成），插入以下内容（数据库实例名可以自己设置，但是要记住。.bash\_profile文件是主要配置环境变量的文件，如果出现执行文件没有找到的情况，很有可能就是里边的path写错了）。

```
export ORACLE_BASE=/home/oracle/app/oracle #这个路径为（5）中的你创建的目录
export ORACLE_HOME=/home/oracle/app/oracle/product/11.2.0/dbhome_1
export ORACLE_SID=powerdes #数据库实例名。实例名必须记住，创数据库时会用到
export PATH=$ORACLE_HOME/bin:$PATH #非常重要！涉及的所有的可执行的文件都在PATH下边
export ORACLE_TERM=xterm
export TNS_ADMIN=$ORACLE_HOME/network/admin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/oracm/lib:$ORACLE_HOME/lib
export CLASSPATH=$CLASSPATH:$ORACLE_HOME/rdbms/jlib:$ORACLE_HOME/jlib:$ORACLE_HOME/network/lib
export LANG=en_US.gbk
export NLS_LANG=american_america.ZHS16GBK
export EDITOR=vi
```

## （9）准备VNC远程连接linux桌面

参考链接：<http://www.linuxidc.com/Linux/2015-01/112326p2.html>

### I.在Windows上安装VNC

### II.在linux上安装VNC

先检查一下服务器是否已经安装了VNC服务。检查服务器是否安装VNC的命令如下：

```
[root@localhost ~]# rpm -qa | grep vnc
[root@localhost ~]#
```

如果没有安装vnc可以使用下面命令进行安装：

```
yum install -y tigervnc tigervnc-server
```

顺利安装完，检查一下

```
[root@localhost ~]# rpm -qa|grep vnc
tigervnc-1.1.0-16.el6.CentOS.x86_64
tigervnc-server-1.1.0-16.el6.centos.x86_64
libvncserver-0.9.7-4.el6.x86_64
[root@localhost ~]#
```

### III. 配置VNC

编辑/etc/sysconfig/vncservers配置文件（这是设置VNC用户的文件），在该文件里编辑以下内容：

```
vim /etc/sysconfig/vncservers
VNCSEVERS="1:root" #配置远程桌面登录的用户名，如果两个用户，则使用VNCSE
RVERS="1:user1 2:user2"(VNCSEVER= "usernumber:myusername")
VNCSEVERARGS[1]="-geometry 800x600" #设置分辨率，连接上服务器后也可以
更改
```

### IV. 设置VNC密码

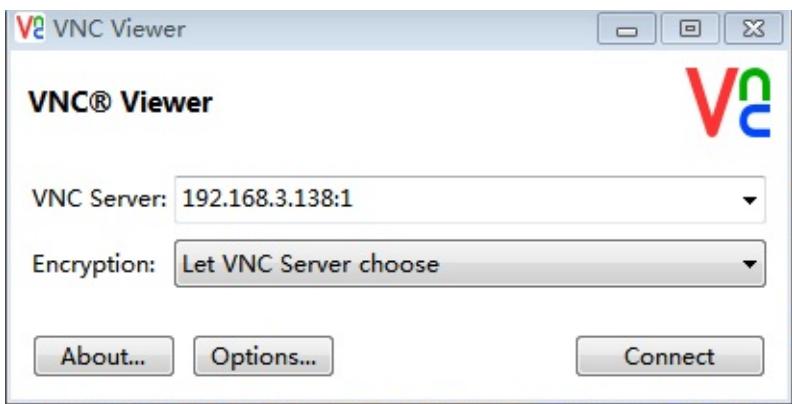
使用下面的命令为VNC设置密码（需要使用su命令切换到需要设置VNC密码的用户上，例如：su - oracle，其中oracle就是在上面配置文件内写的帐号）

```
[root@localhost ~]# vncpasswd
Password:
Password must be at least 6 characters - try again
Password:
Verify:
[root@localhost ~]#
```

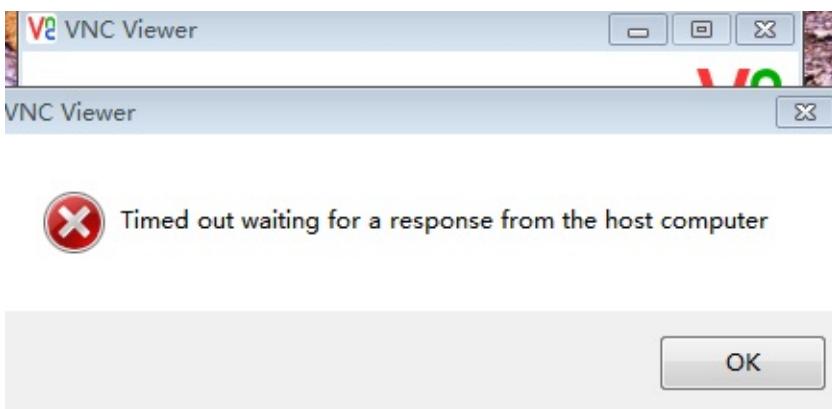
注意：密码至少为6个字符

### V. 建立VNC链接

配置VNC完成后，需要执行命令service vncserver restart重新启动VNC。然后打开安装在Windows中的VNC，输入要连接的服务器IP。注意冒号后边对应的是你设置的VNC用户对应的用户名编号。如下图所示



注意：直接连接会发现连接超时。如下图所示：



解决方法：开启防火墙VNCServer端口

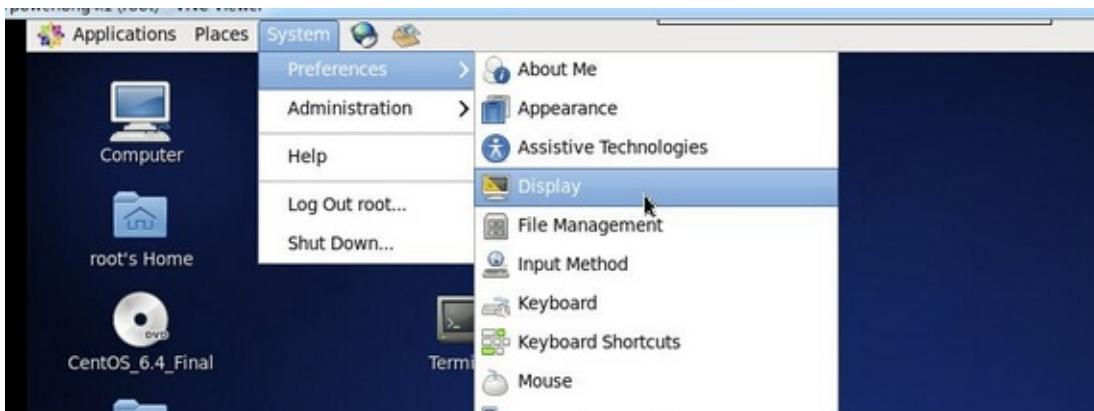
首先编辑/etc/sysconfig/iptables文件

```
#vim /etc/sysconfig/iptables
```

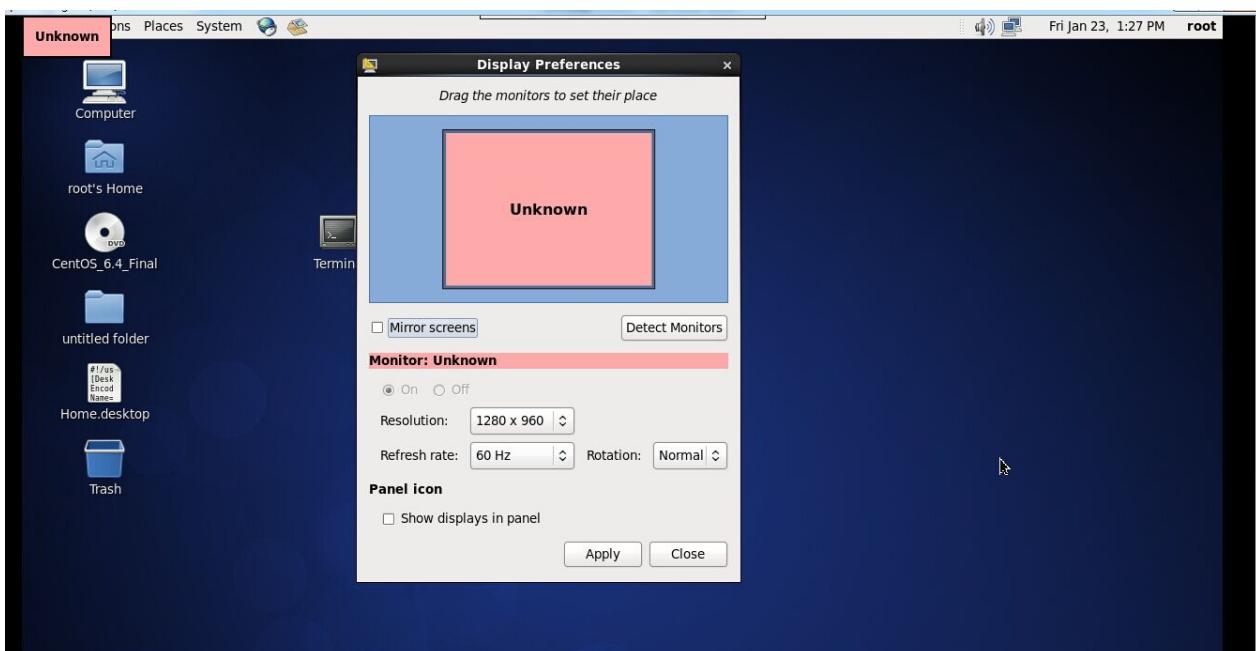
在“-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT”下面添加一行“-A INPUT -m state --state NEW -m tcp -p tcp --dport 5901 -j ACCEPT”,然后重启iptables服务。

```
#/etc/init.d/iptables restart
```

连接成功后会显示出虚拟机的桌面。然后点击system下的preferences下的display



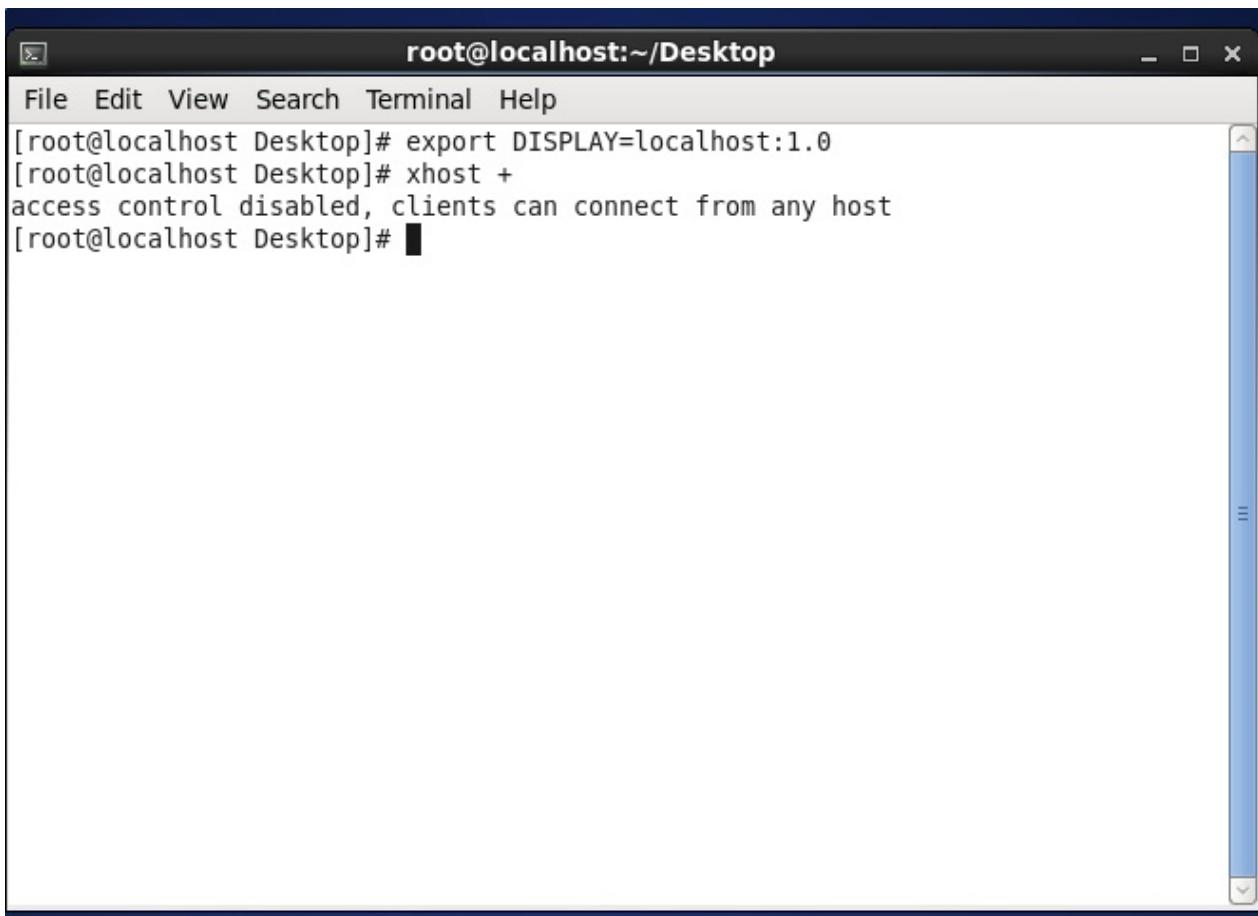
点击display后会弹出如下图所示的窗口，根据自己的需求更改分辨率



然后新打开一个终端，在命令行中执行 export DISPLAY=localhost:1.0 (要连接服务器IP : vnc用户对应的编号) 命令与xhost +命令

```
[root@powerlong4 rlwrap-0.37]# xhost +
access control disabled, clients can connect from any host
[root@powerlong4 rlwrap-0.37]#
```

表示linux下视窗环境已经准备OK，可以进行oracle安装。如下图



A screenshot of a terminal window titled "root@localhost:~/Desktop". The window has a dark blue header bar with the title and a light gray body. The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the following command sequence:

```
[root@localhost Desktop]# export DISPLAY=localhost:1.0
[root@localhost Desktop]# xhost +
access control disabled, clients can connect from any host
[root@localhost Desktop]#
```

## 4. 安装过程

### FAQ问题--用oracle用户安装报错

#### 1. 用oracle用户安装报错

```
Checking monitor: must be configured to display at least 256 col
ors      Failed <<<
>>> Could not execute auto check for display colors using co
mmand /usr/X11R6/bin/xdpinfo. Check if the DISPLAY variable is
set.
```

解决方法：

1、root 下先执行#xhost +

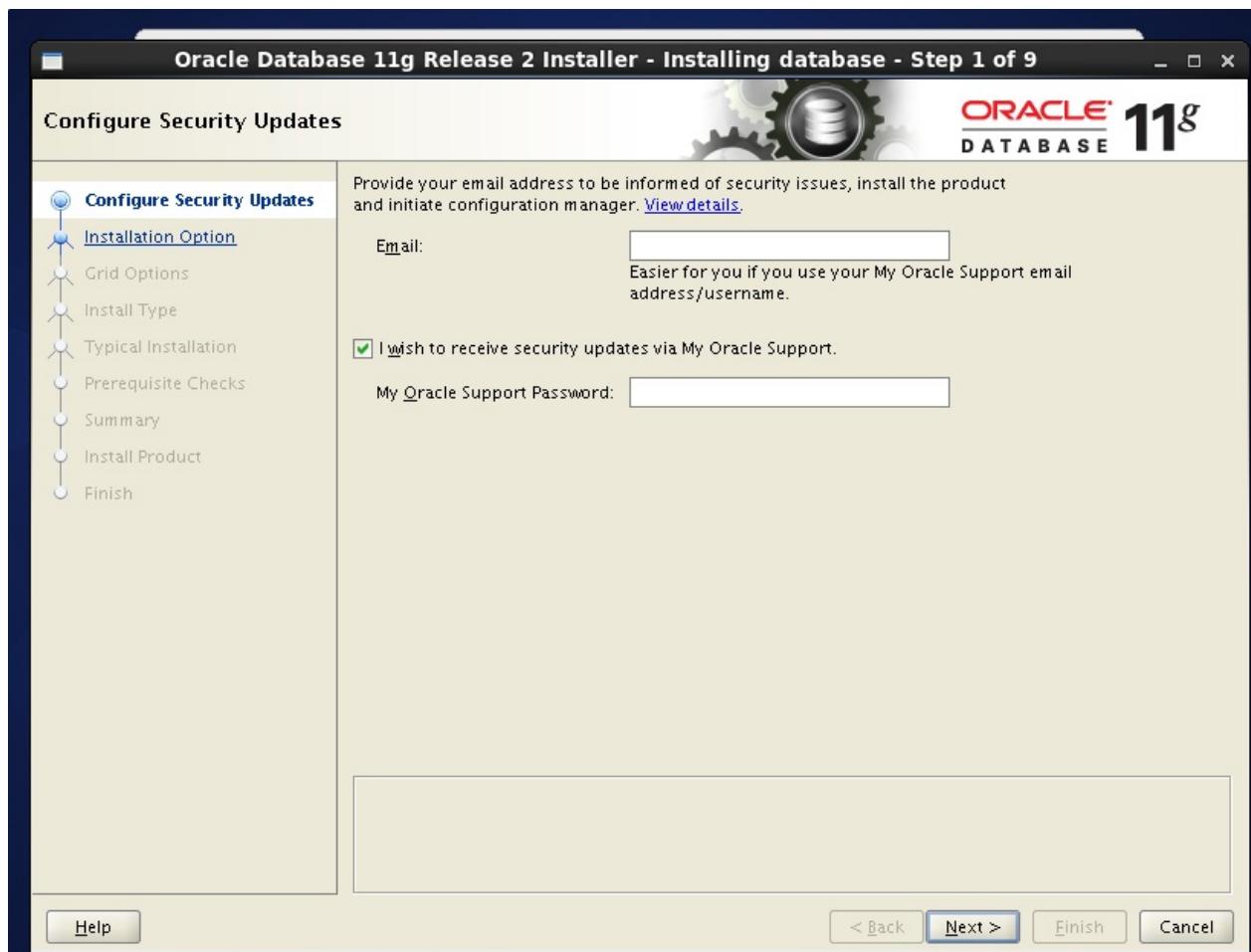
2、su oracle

3、export DISPLAY=:0.0

## (1) 开始安装oracle

切换oracle用户然后cd到你解压完安装包的database下，执行./runInstaller。然后vnc就会弹出安装界面窗口。过程如下各图：

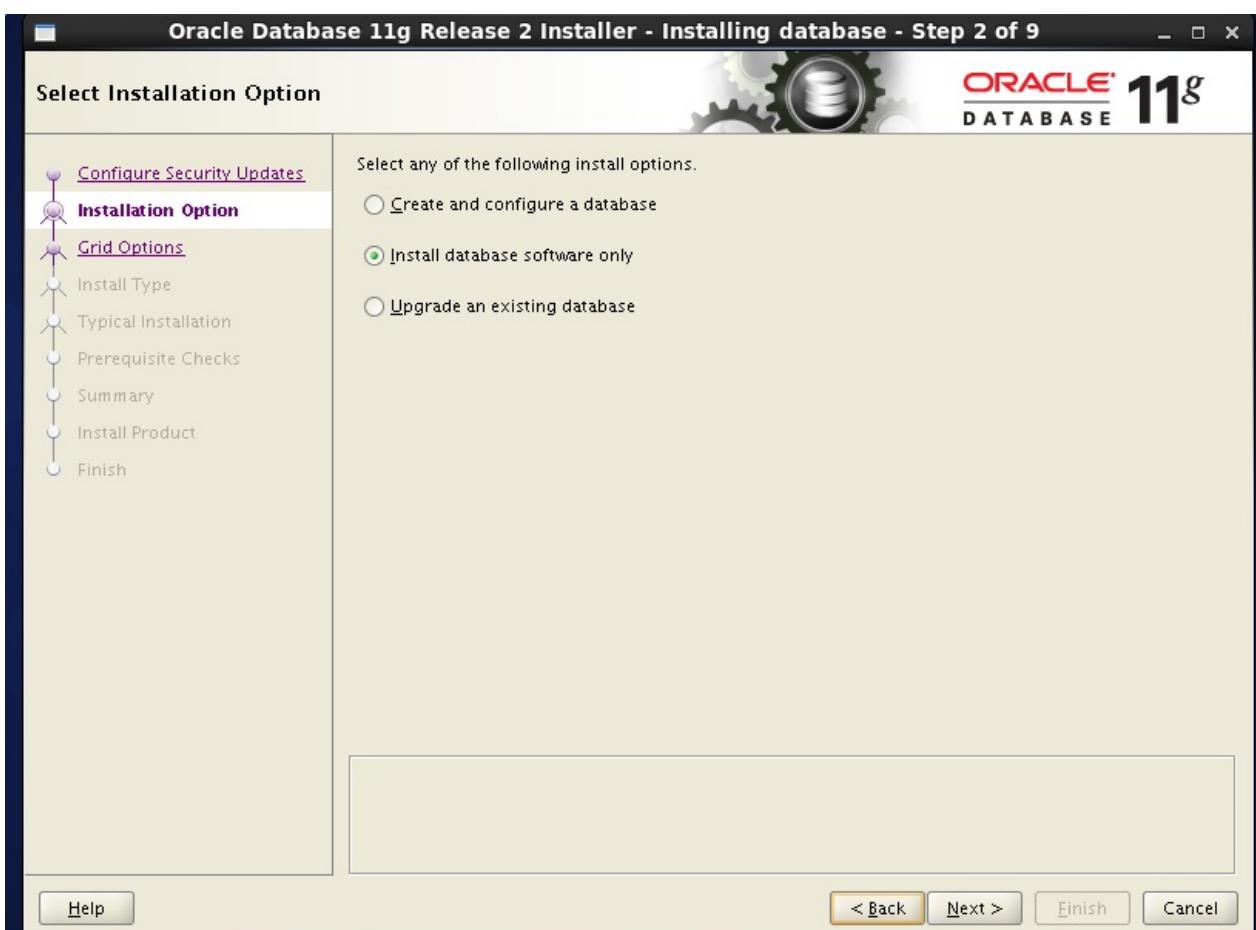
不建议填写email，下边的创建密码选项可以不选，点击next(填写有可能卡死在这个地方)



选择i want to remain.....，点击continue



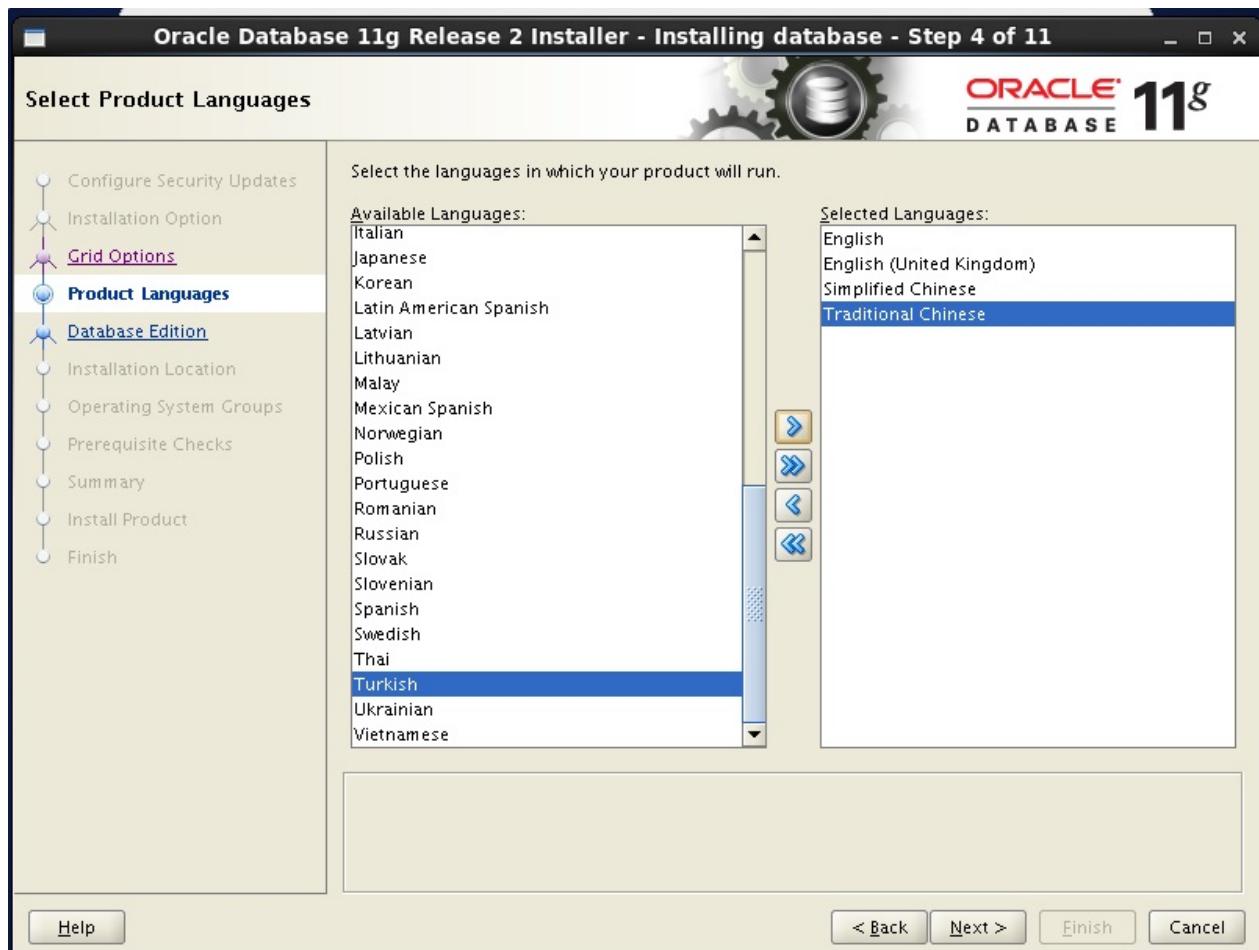
选择第二个仅安装数据库软件，点击next



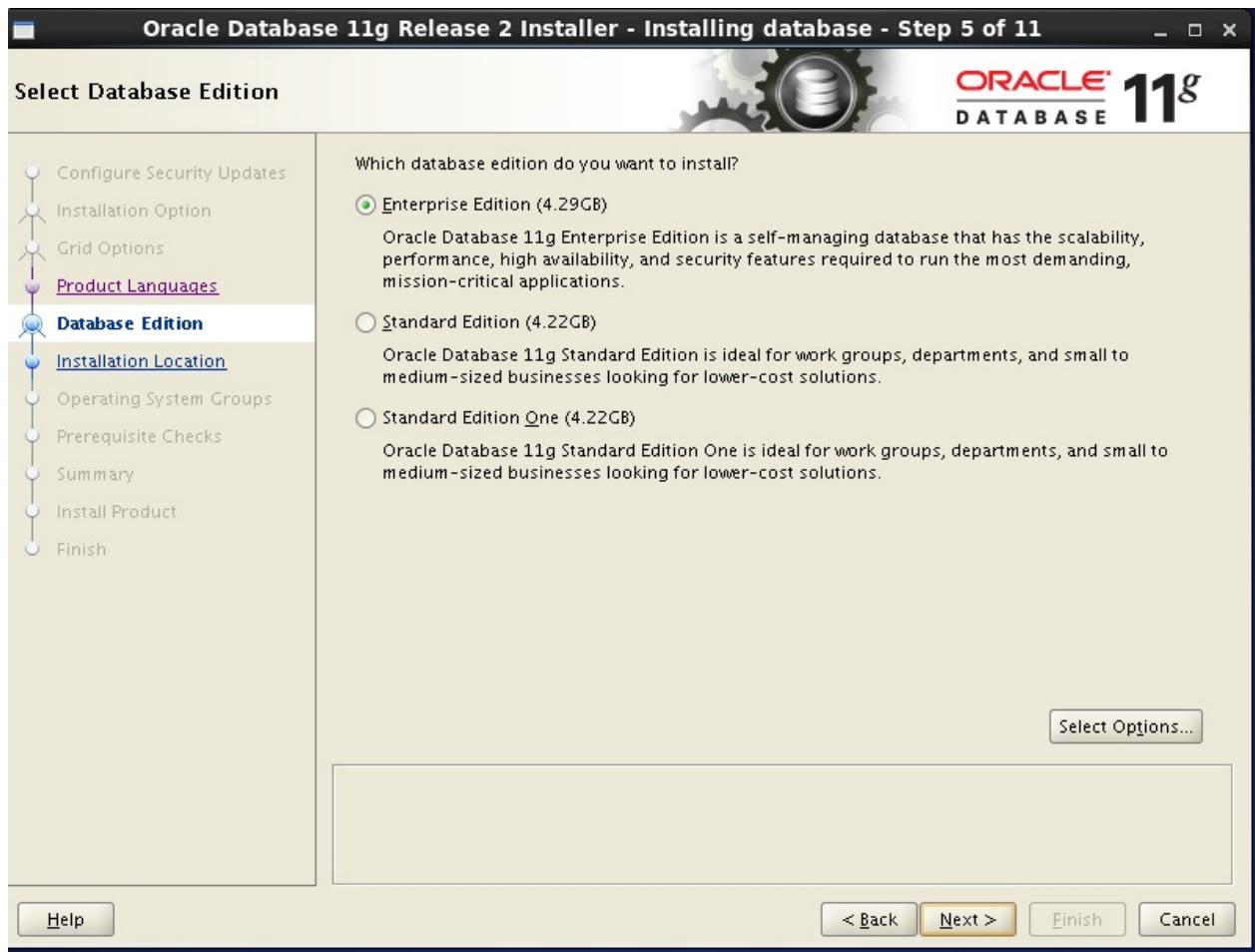
选择第一个，点击next



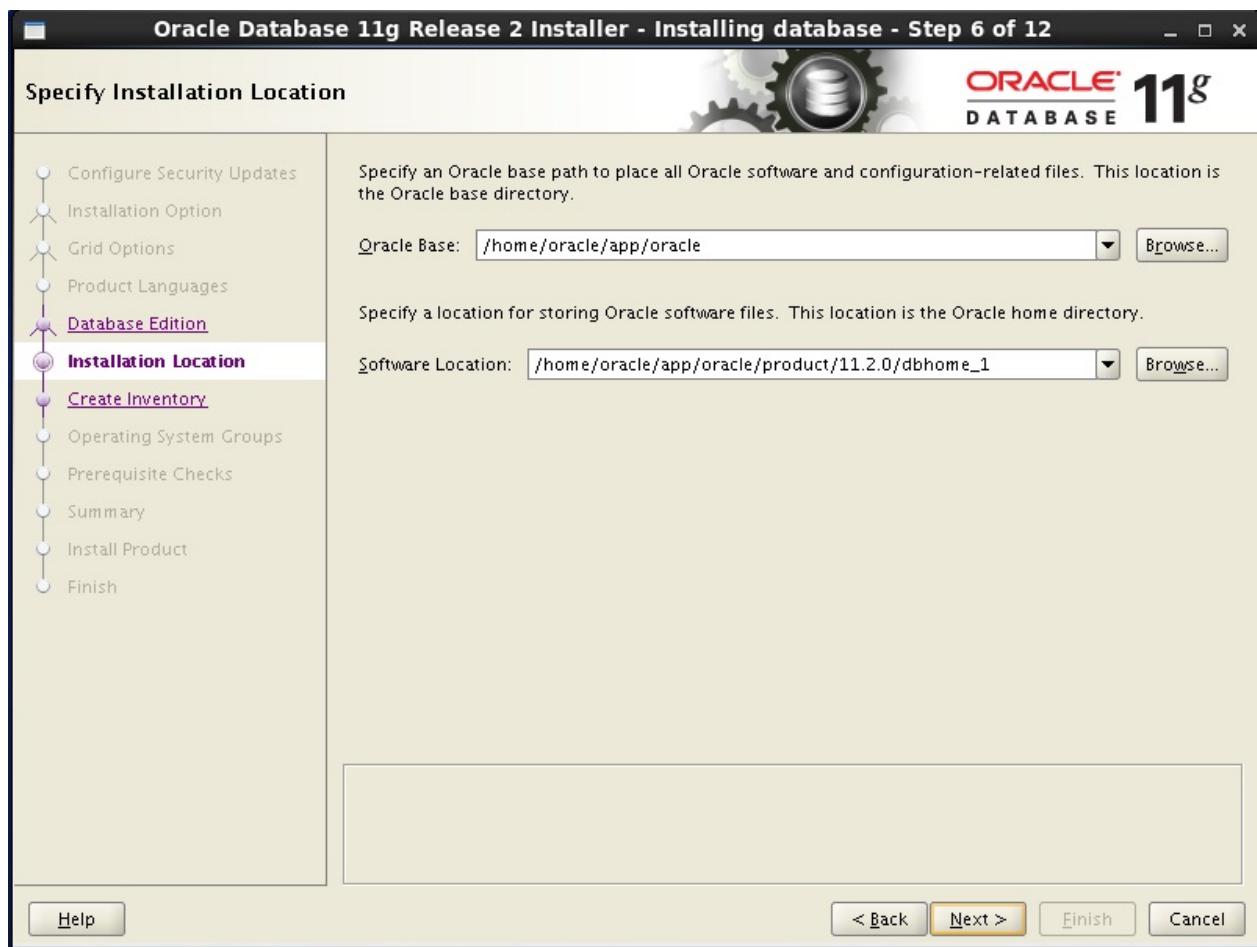
选择如下图所示的languages，点击next



选择第一个，点击next



默认点击next



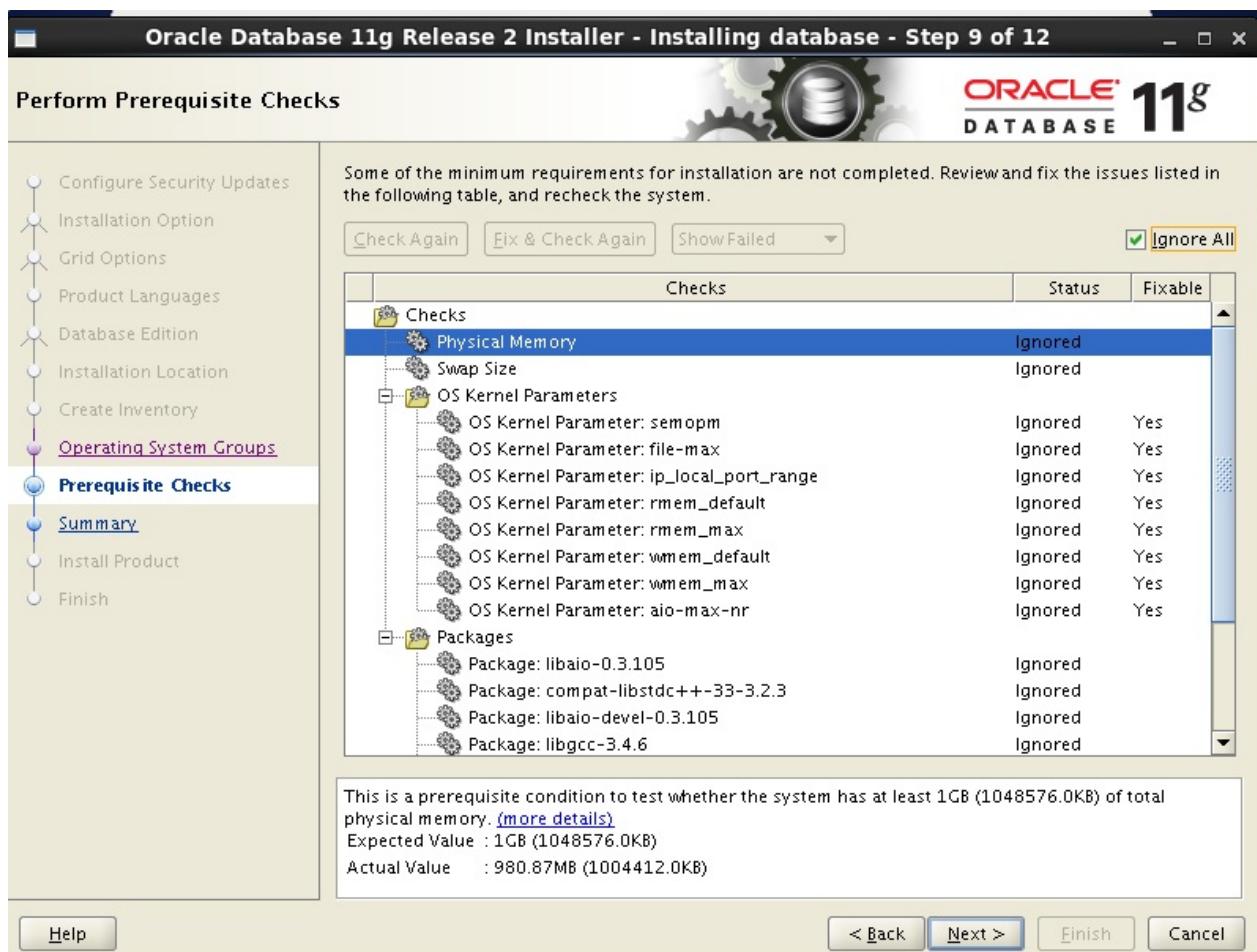
默认点击next



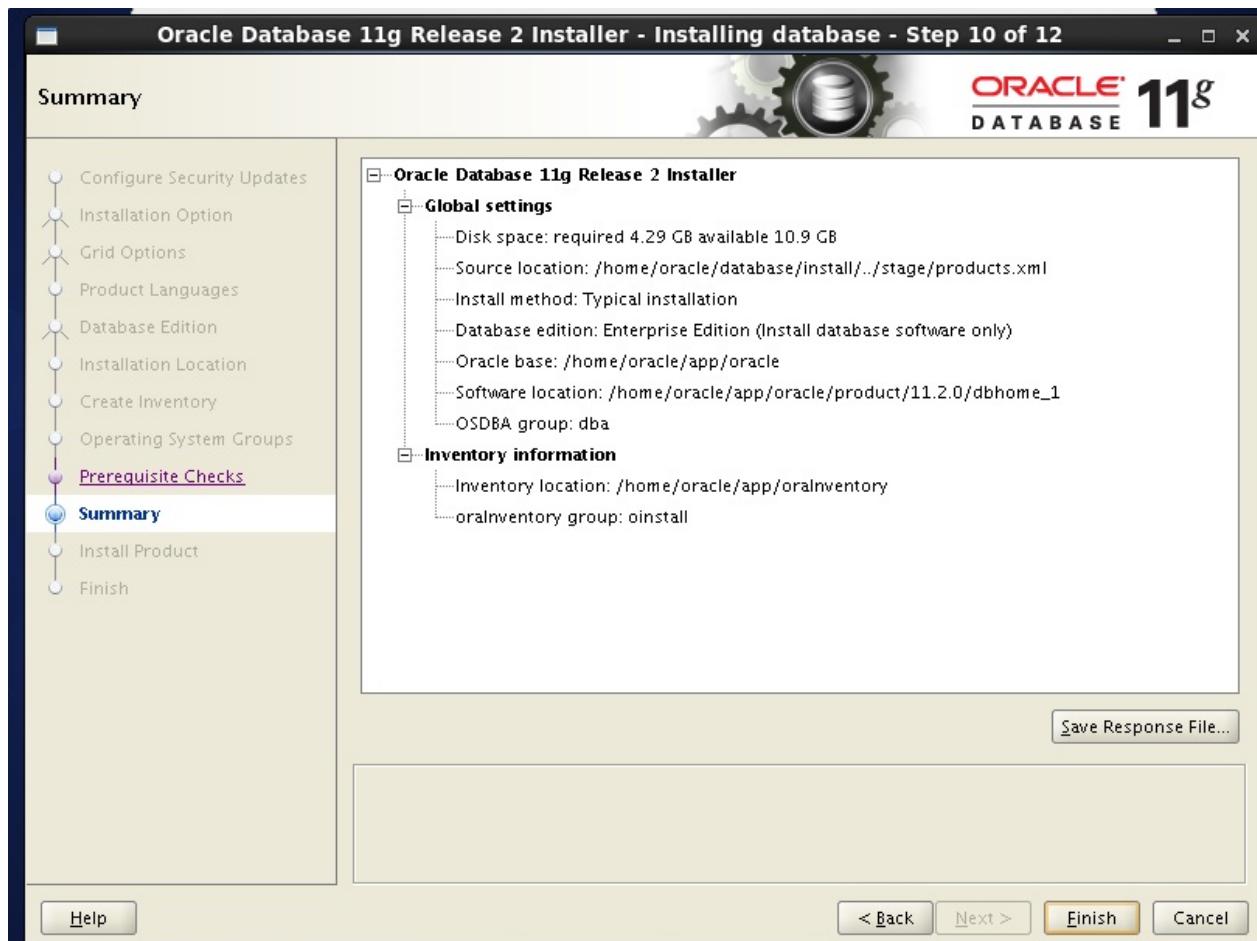
按下图选择组名，点击next



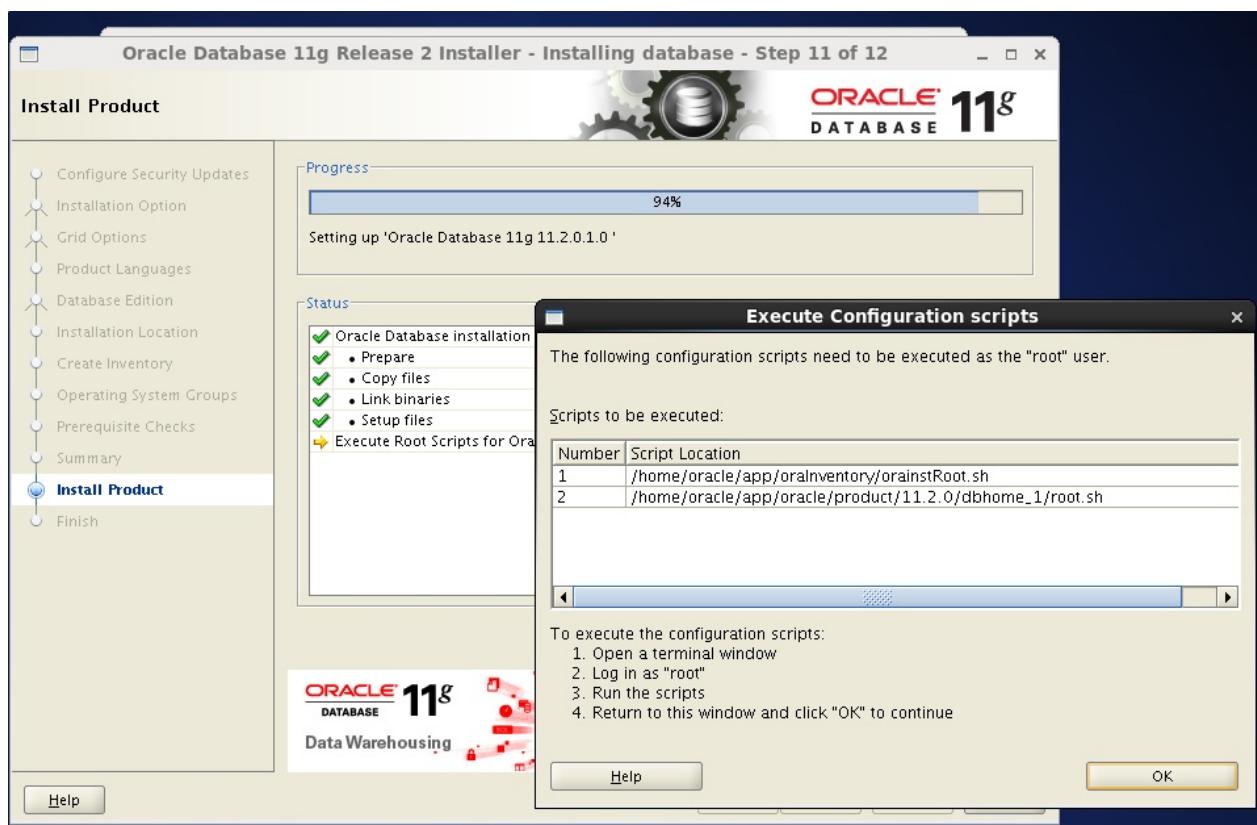
由于CentOS版本较高，所以Oracle11g在check的时候不识别高版本lib包，选择Ignore All，点击next



点击finish开始安装



安装快结束时，会弹出一个窗口通知要执行两个脚本，点击OK，再点击close结束安装



执行之前提示的两个脚本，分别cd进入相应的路径下，执行sh oraInstRoot.sh和sh root.sh如下图

```
root@localhost ~]# cd /home/oracle/app
root@localhost app]# ls -la
total 16
drwxrwxr-x. 4 oracle oinstall 4096 Apr  5 15:07 .
drwx----- 6 oracle oinstall 4096 Apr  5 13:58 ..
drwxrwxr-x. 4 oracle oinstall 4096 Apr  5 15:23 oracle
drwxrwx--- 5 oracle oinstall 4096 Apr  5 15:27 oraInventory
root@localhost app]# cd oraInventory/
root@localhost oraInventory]# ls -la
total 36
drwxrwx--- 5 oracle oinstall 4096 Apr  5 15:27 .
drwxrwxr-x 4 oracle oinstall 4096 Apr  5 15:07 ..
drwxrwx--- 2 oracle oinstall 4096 Apr  5 15:27 ContentsXML
rw-rw---- 1 oracle oinstall   37 Apr  5 15:27 install.platform
drwxrwx--- 2 oracle oinstall 4096 Apr  5 15:16 logs
rw-rw---- 1 oracle oinstall  309 Apr  5 15:16 oraInstaller.properties
rw-rw---- 1 oracle oinstall   64 Apr  5 15:27 oraInst.loc
drwxrwx--- 1 oracle oinstall 1695 Apr  5 15:27 oraInstRoot.sh
drwxrwx--- 2 oracle oinstall 4096 Apr  5 15:16 oui
root@localhost oraInventory]# sh oraInstRoot.sh
Changing permissions of /home/oracle/app/oraInventory.
Adding read,write permissions for group.
Removing read,write,execute permissions for world.

Changing groupname of /home/oracle/app/oraInventory to oinstall.
The execution of the script is complete.
root@localhost oraInventory]# 
```

```
drwxr-xr-x. 7 oracle oinstall 4096 Apr  5 15:17 xds
[root@localhost dbhome_1]# sh root.sh
Running Oracle 11g root.sh script...

The following environment variables are set as:
  ORACLE_OWNER= oracle
  ORACLE_HOME= /home/oracle/app/oracle/product/11.2.0/dbhome_1

Enter the full pathname of the local bin directory: [/usr/local/bin]:
  Copying dbhome to /usr/local/bin ...
  Copying oraenv to /usr/local/bin ...
  Copying coraenv to /usr/local/bin ...

Creating /etc/oratab file...
Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root.sh script.
Now product-specific root actions will be performed.
Finished product-specific root actions.
[root@localhost dbhome_1]# 
```

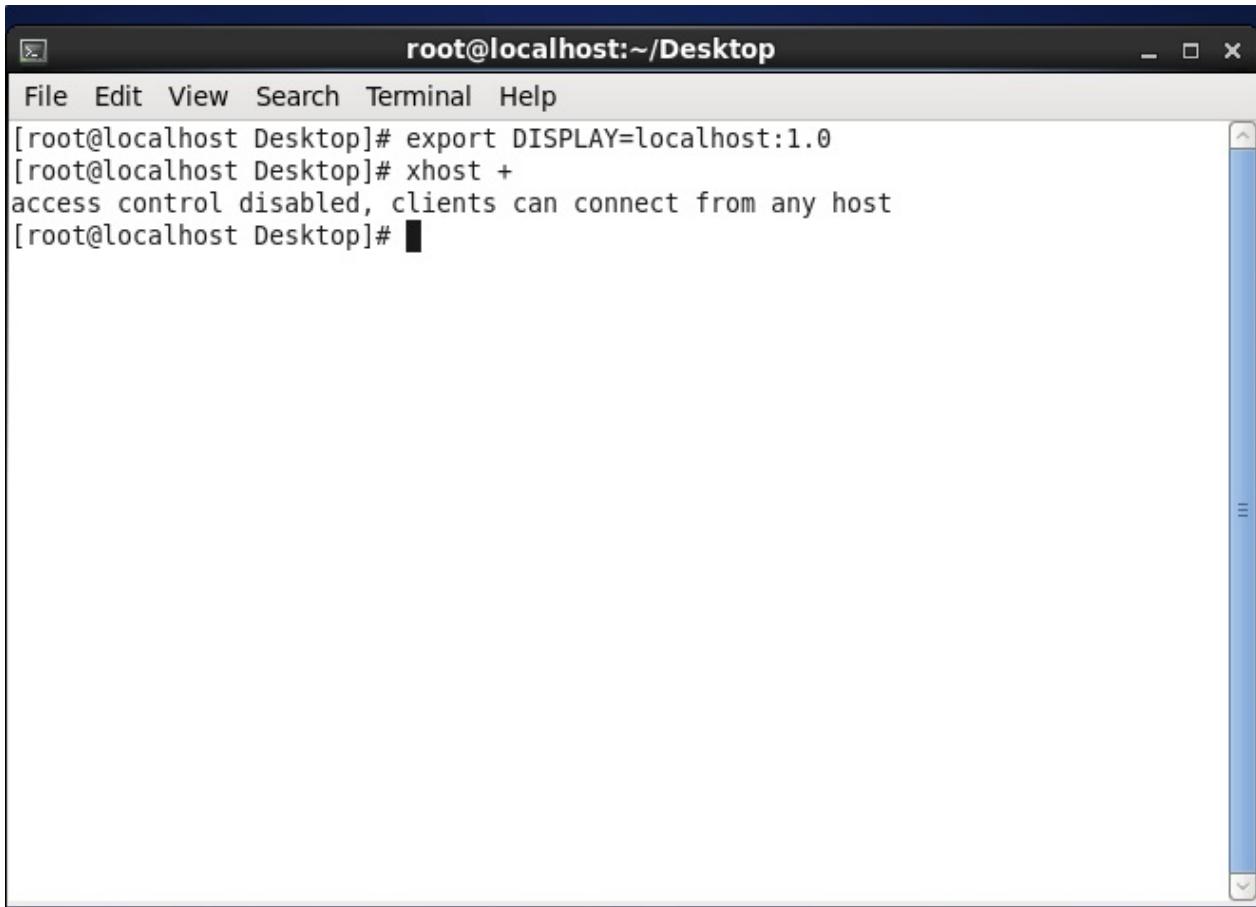
至此oracle安装已经完成

## (2) 数据库建库

注意：以下命令都是在**oracle**用户下执行的。在执行以下所需的命令时，如果不能弹出相应的设置窗口，可以执行命令**service vncserver restart**将VNC重新启动一下，然后再将视图环境重新设置一遍：执行

```
export DISPLAY=localhost:1.0  
xhost +
```

显示如下图：



The screenshot shows a terminal window with the title bar 'root@localhost:~/Desktop'. The window contains the following text:

```
File Edit View Search Terminal Help  
[root@localhost Desktop]# export DISPLAY=localhost:1.0  
[root@localhost Desktop]# xhost +  
access control disabled, clients can connect from any host  
[root@localhost Desktop]# █
```

然后再执行相应的执行命令。

在建库前首先创建监听器，切换用户**oracle**，**cd**到安装时配置环境变量中的**export ORACLE\_HOME**下的bin中，执行**./netca**，会在vnc上弹出添加监听器的窗口。需要注意的是在选定监听程序的协议的时候，选择TPC协议，选择标准端口1521（可以参考下列各图）

选择**Listener configuration**，点击**next**



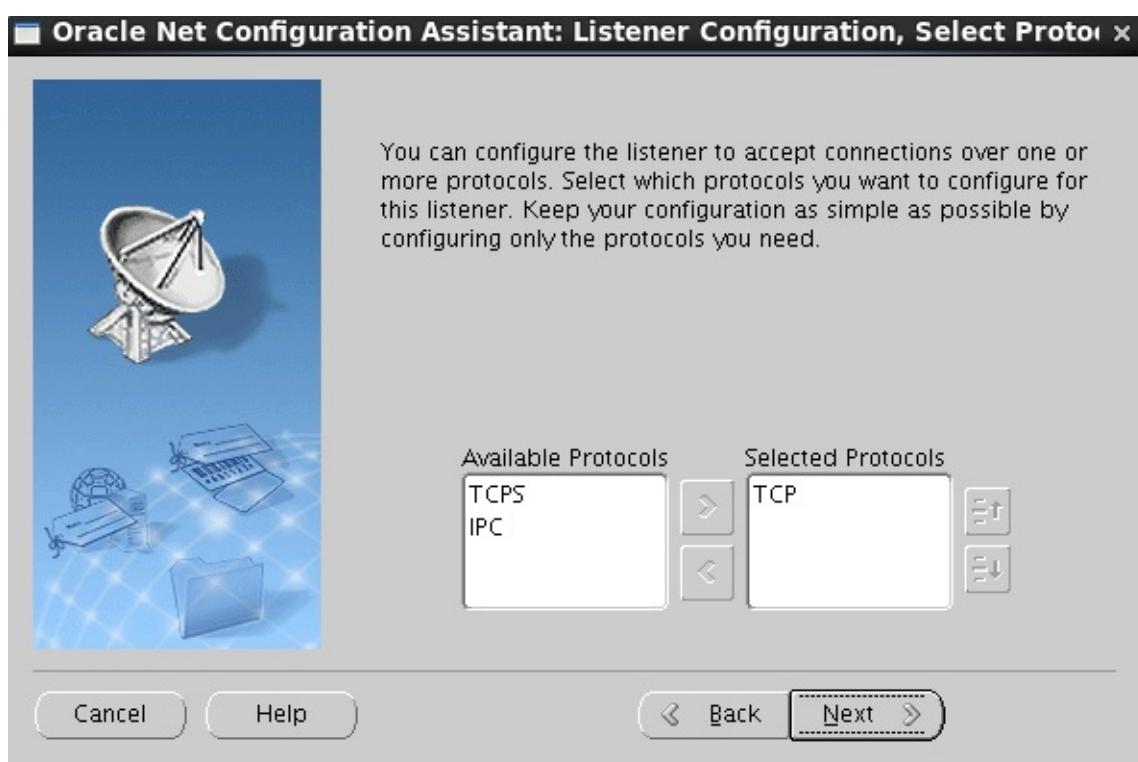
默认选择Add，点击next



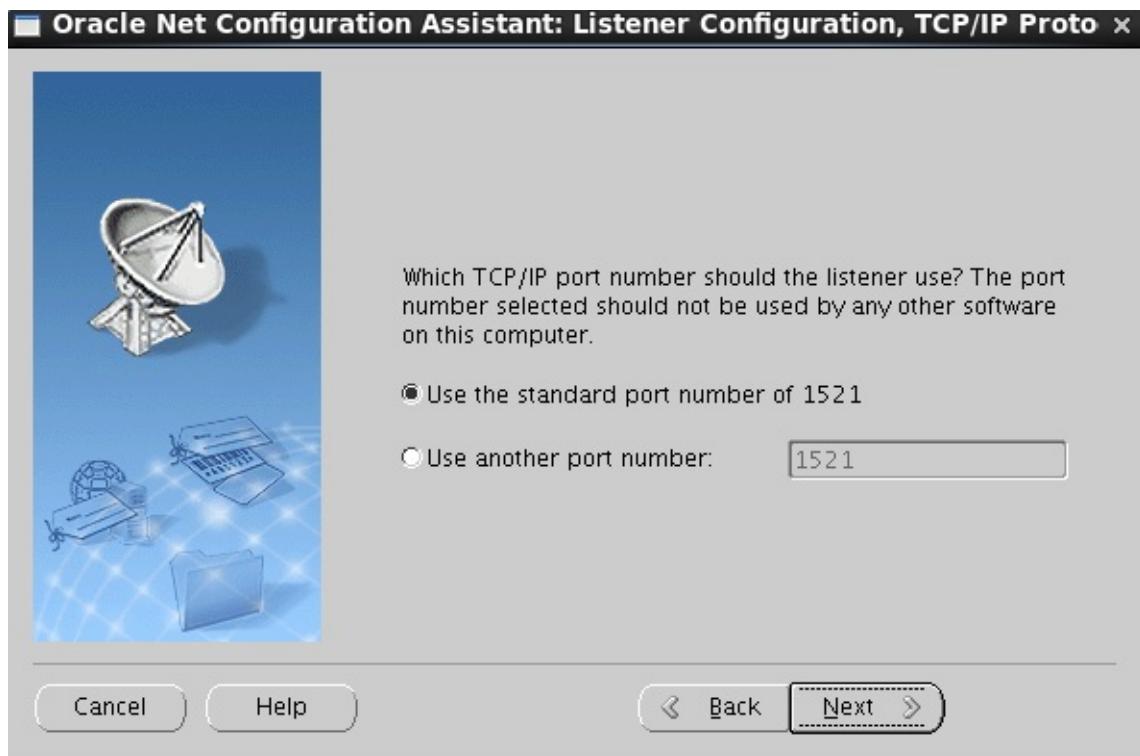
设置监听器名称，点击next



默认选择TCP，点击next



默认选择Use the standard port number of 1521,点击next



默认选择No，点击next



点击next

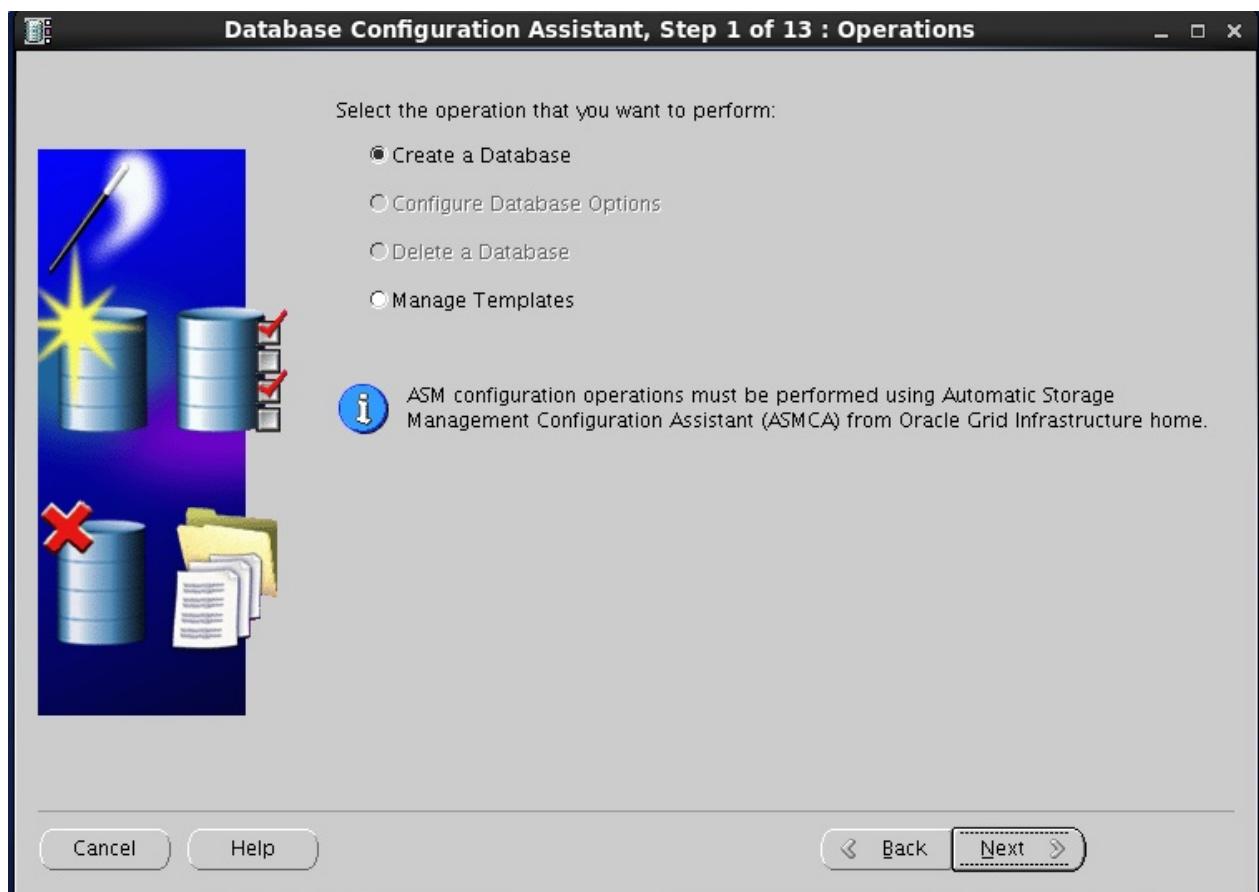


点击next后点击finish完成监听器创建。创建完监听器后执行./dbca文件创建数据库。

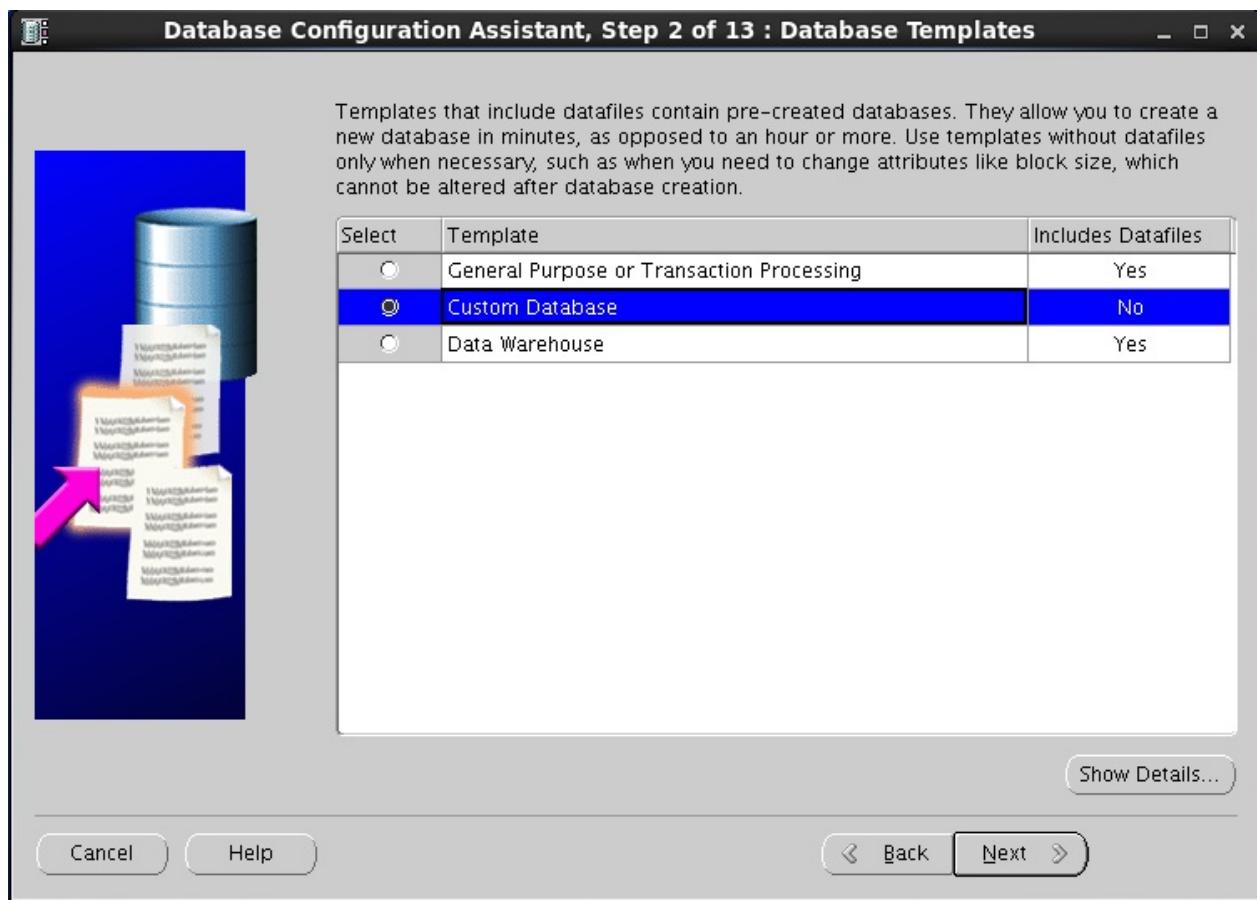
点击Next



选择Create a Database，点击Next



选择Custom Database，点击Next

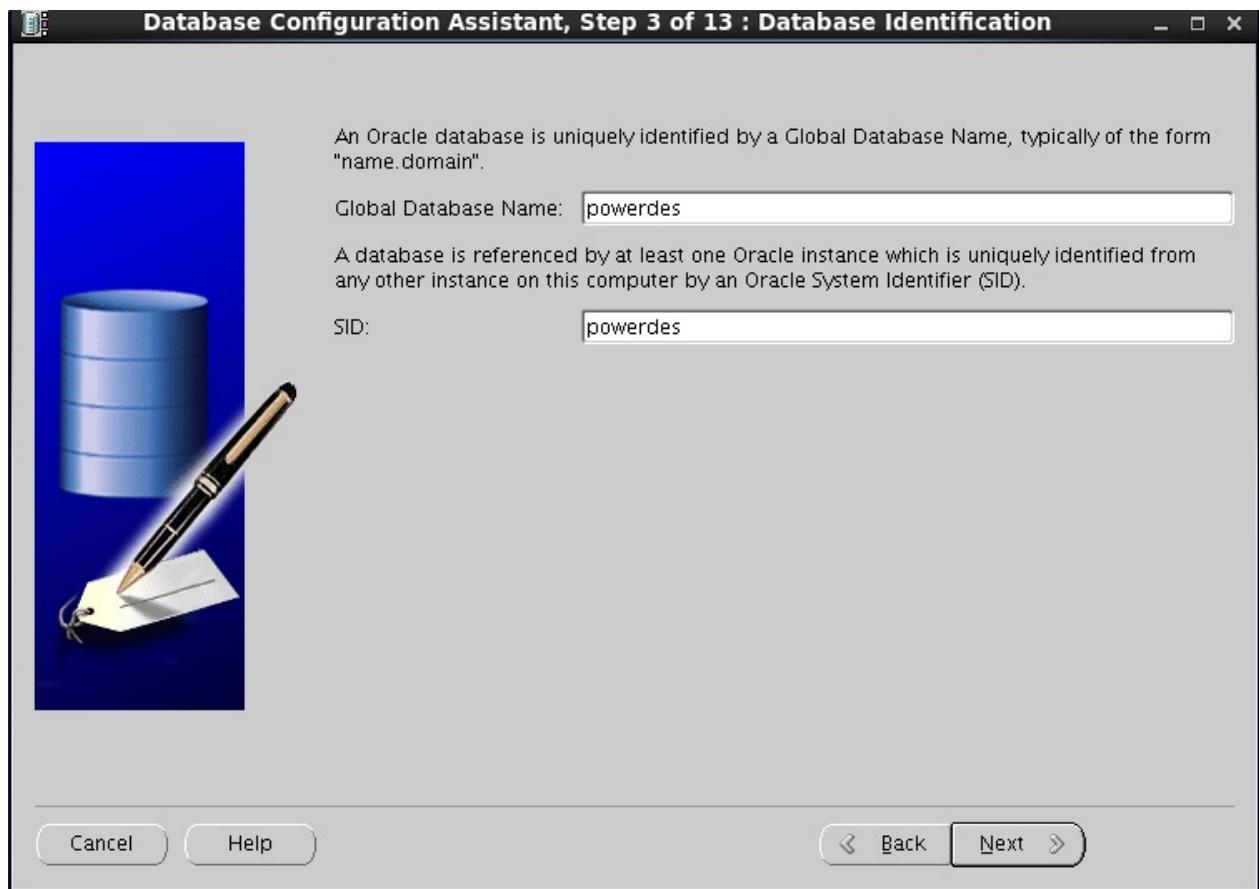


Global Database Name框：输入前面设定的数据库名

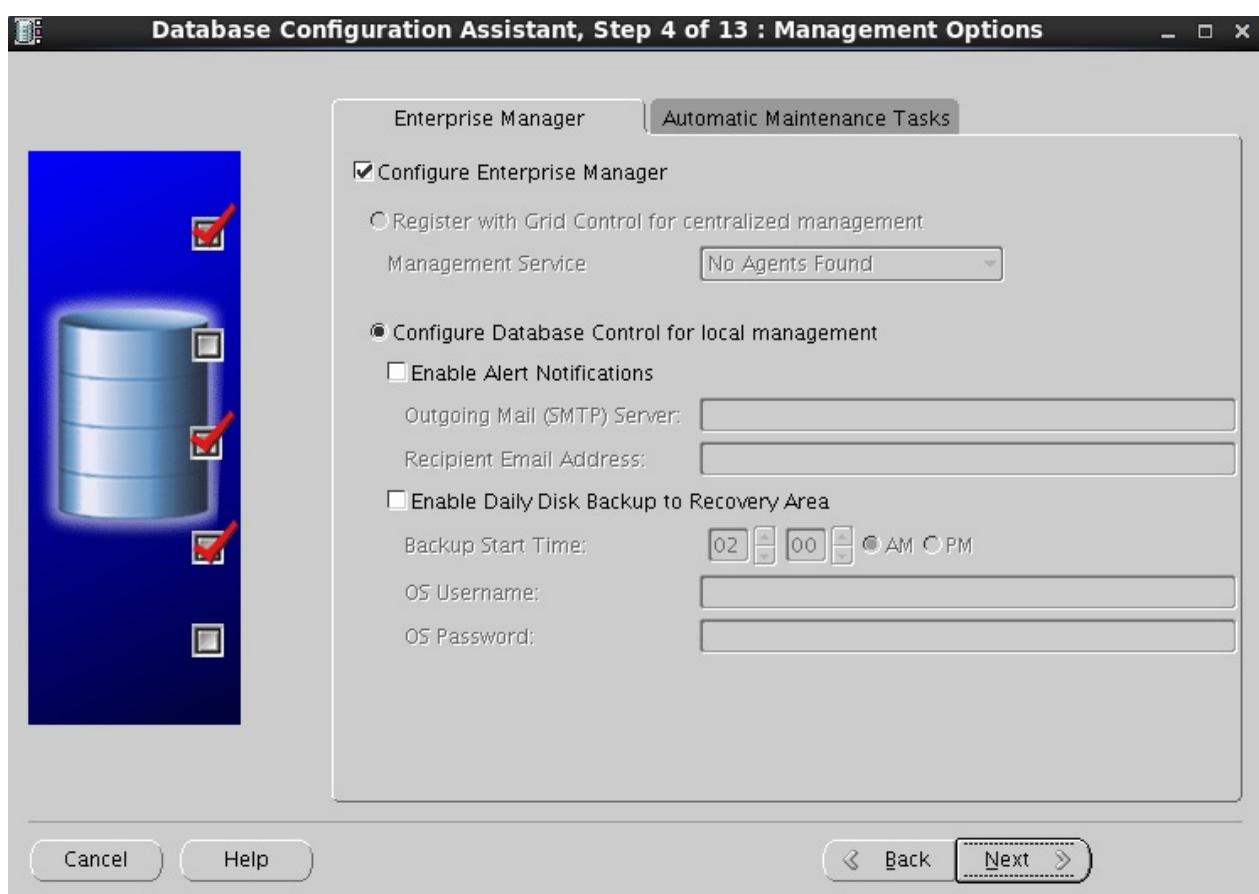
SID框：自动出现和数据库名相同的内容作为数据库实例名，单实例情况下不作改动

点击Next

注意：下图的global database name必须跟之前.bash\_profile里边设的ORACLE\_SID的名字相同。

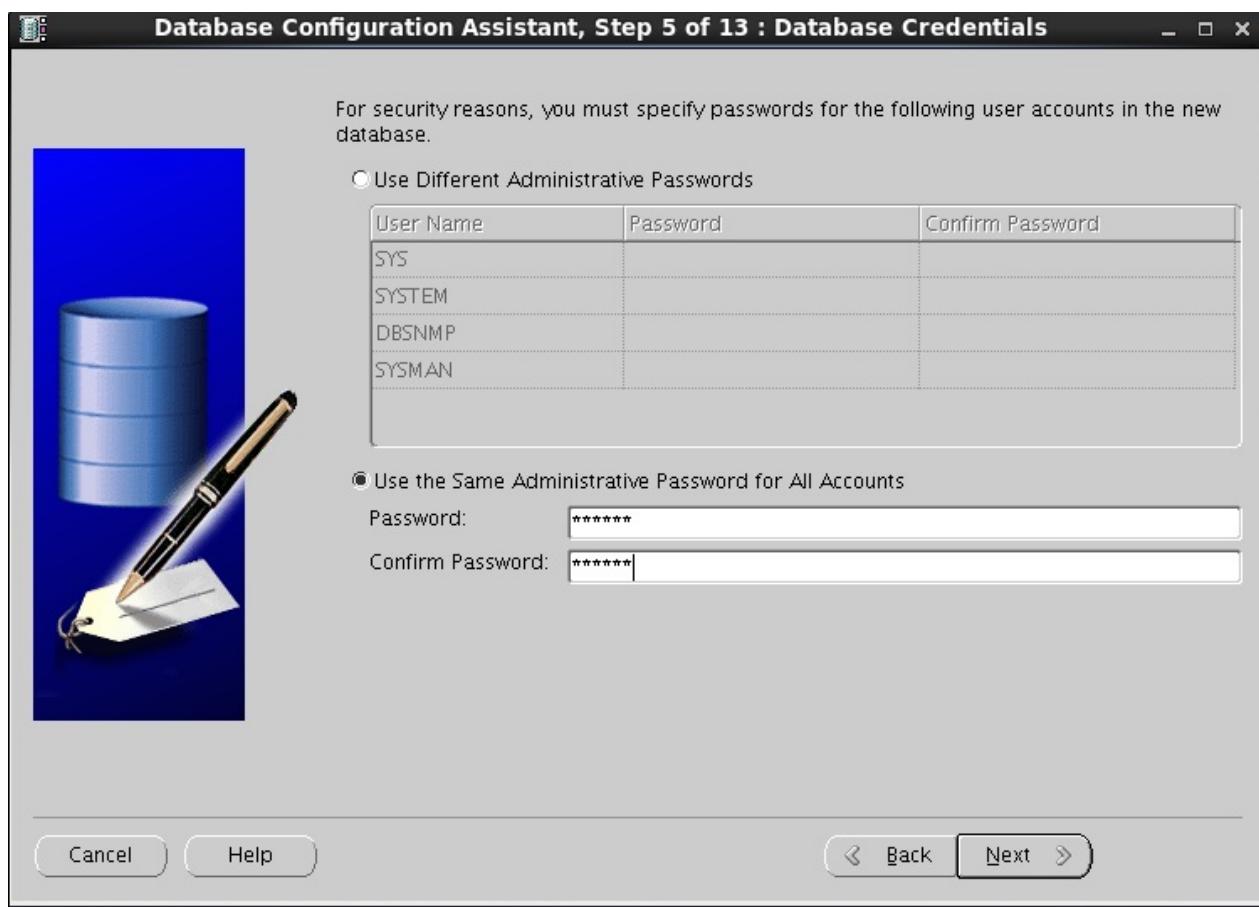


默认所选，点击Next

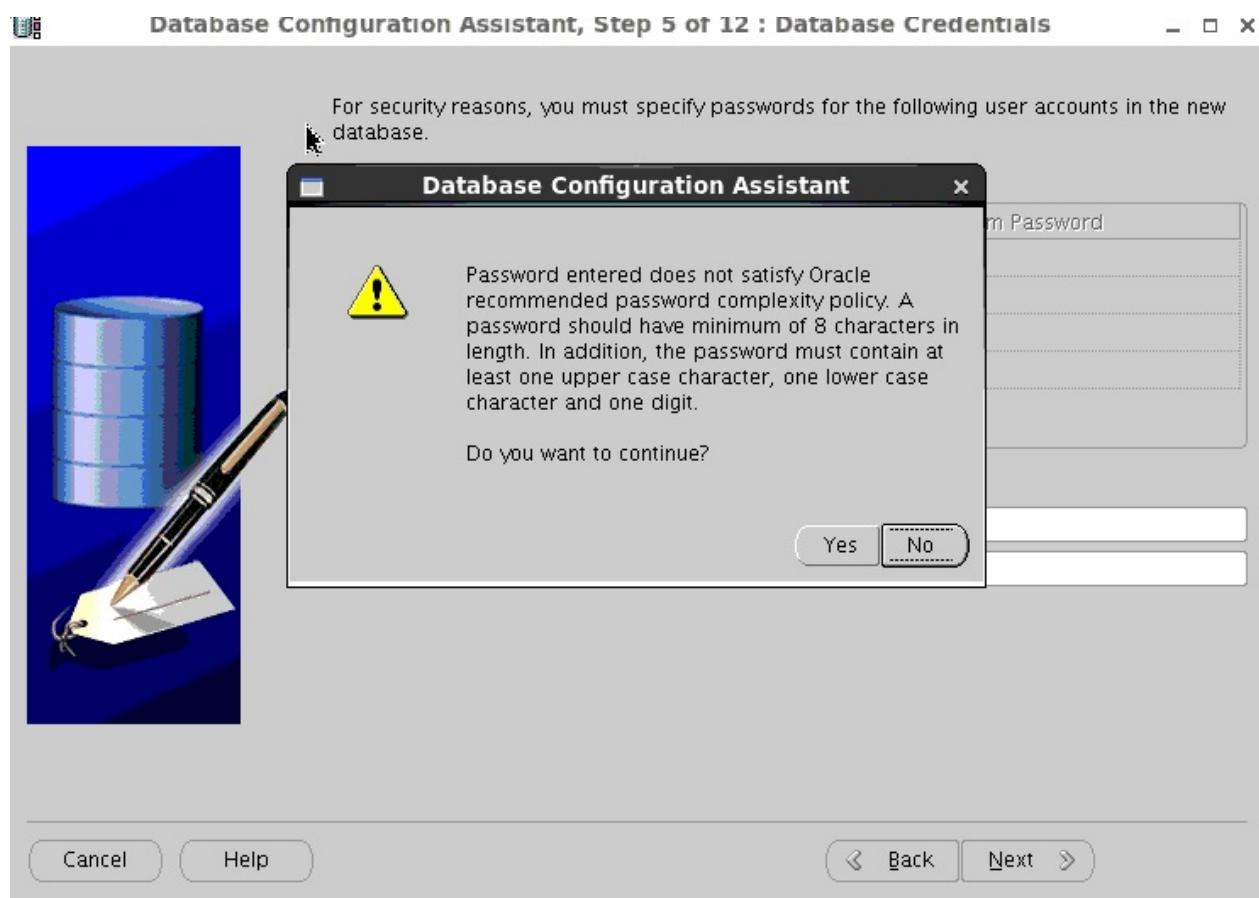


Use Different Administrative Passwords 表格的 Password 和 Confirm Password 列中分别为 User Name 列 SYS 、 SYSTEM 、 DBSNMP 和 SYSMAN 用户输入口令并重复一次输入（如密码设置过于简单，下一步前会有弹出窗口提示确认接受安全风险） sys 和 system 密码设置可以在数据库建立后修改。

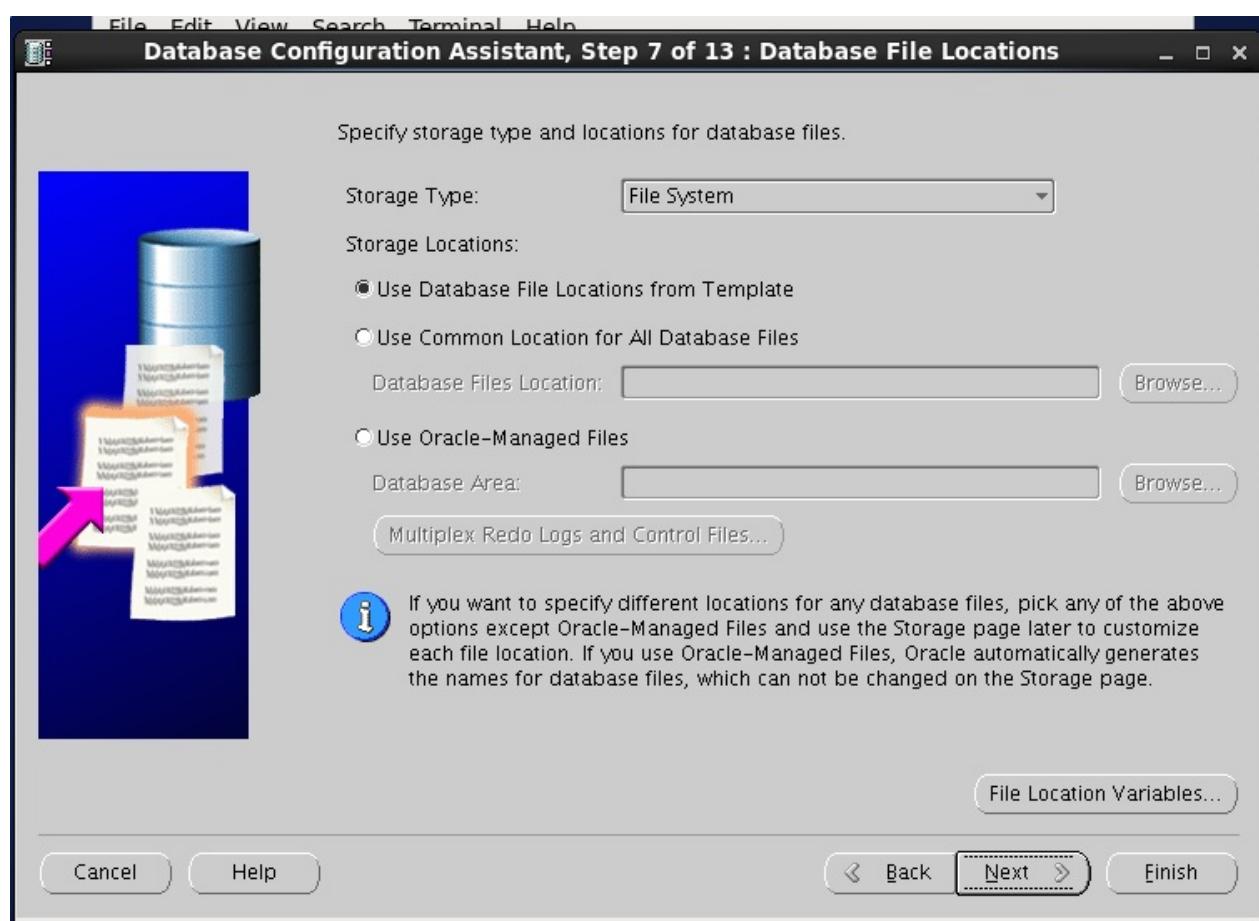
这里选择使用相同密码。点击 Next



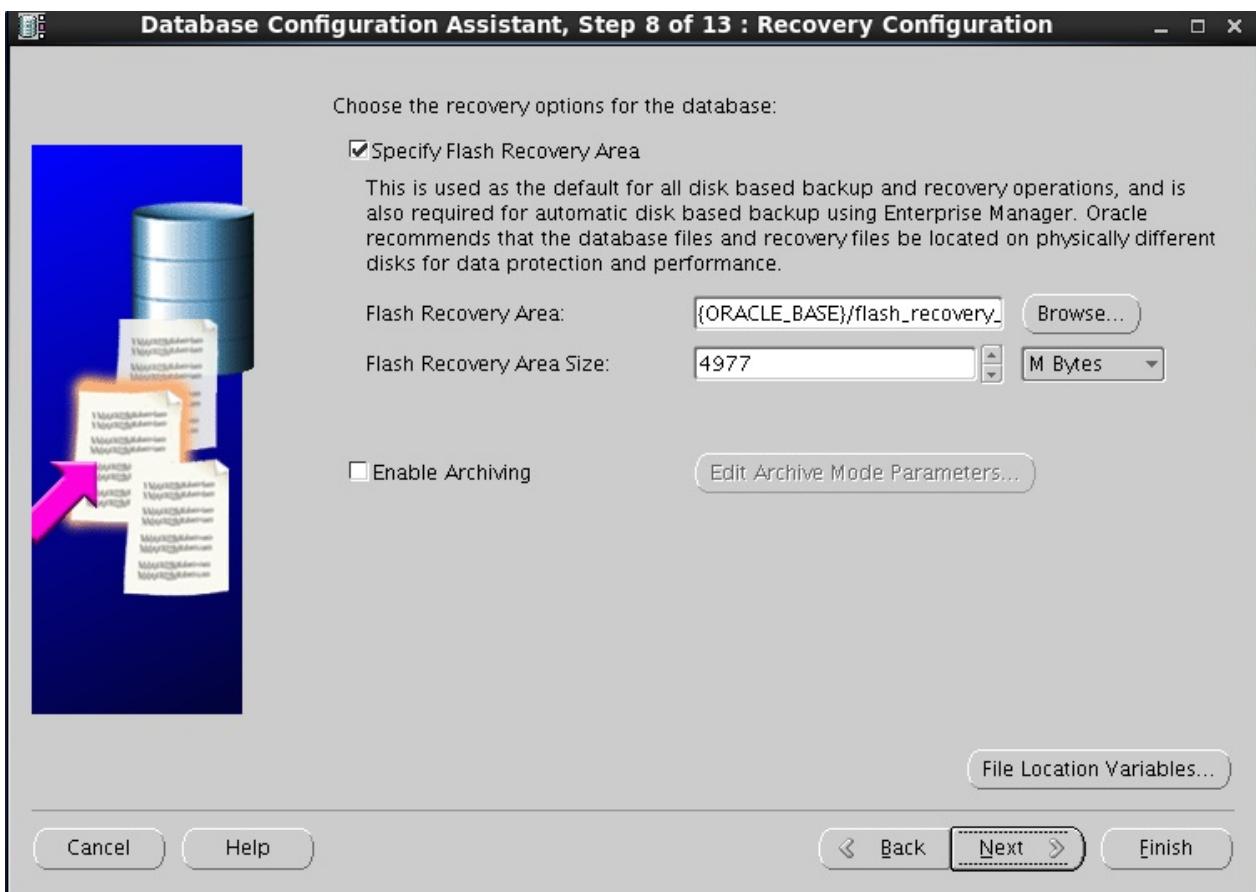
提示设置的密码过于简单存在安全风险是否继续



按照图示选择，点击Next



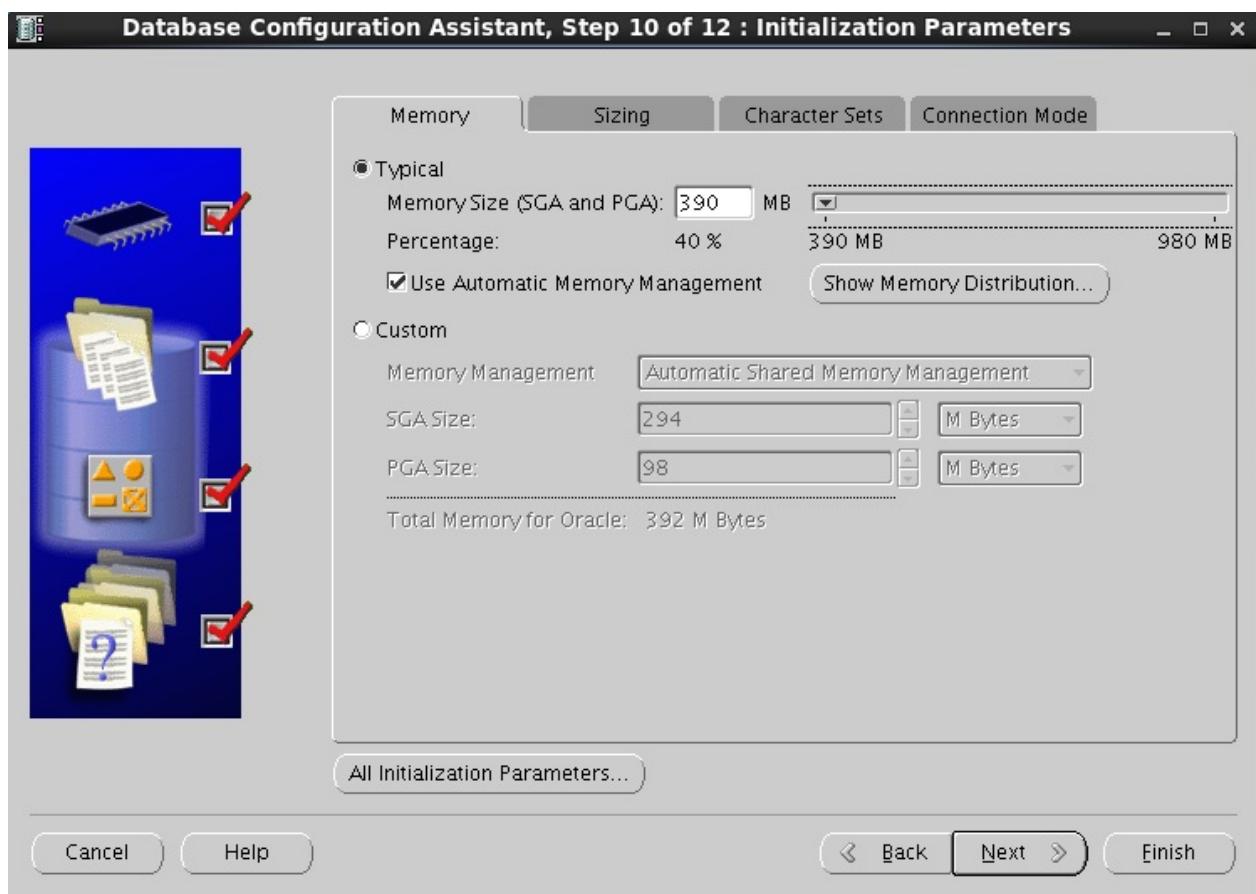
默认所选，点击Next



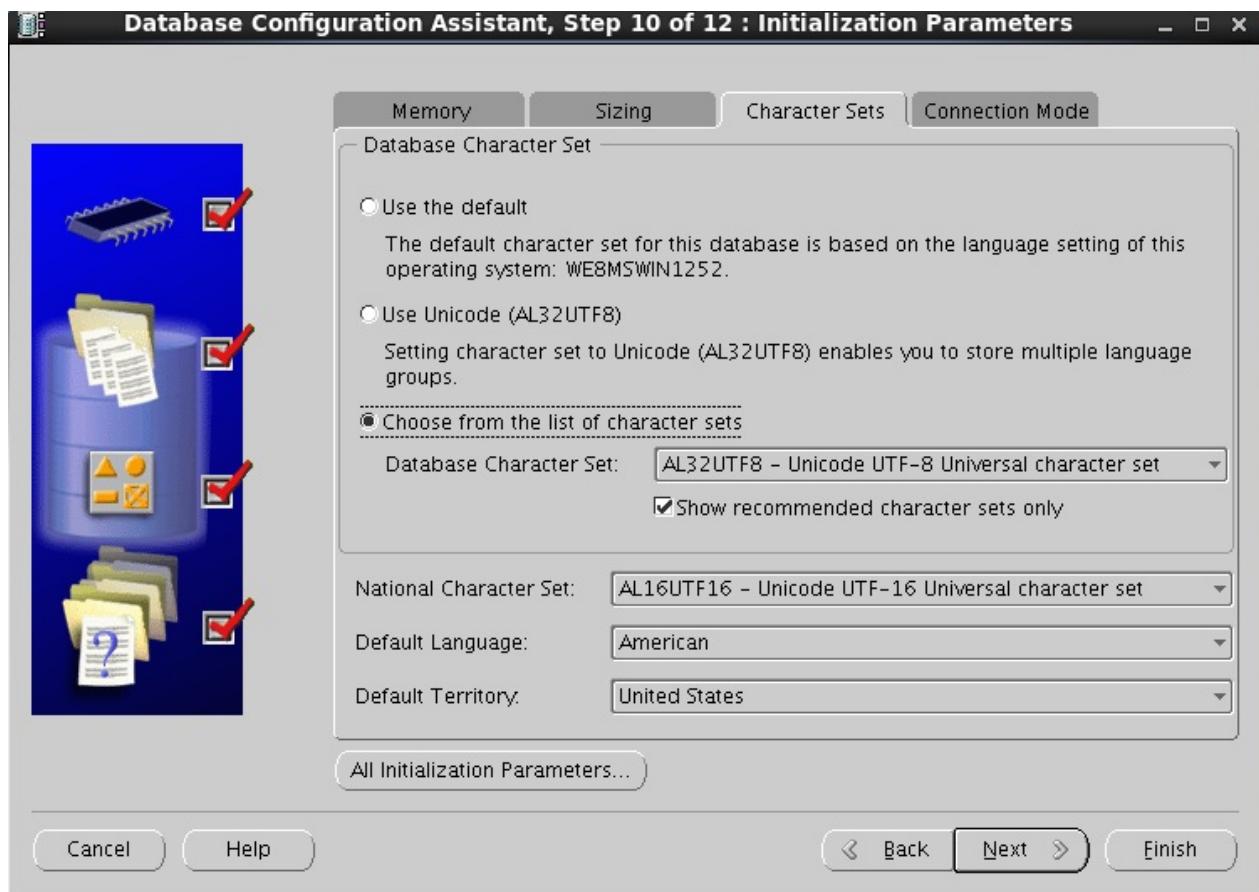
默认所选，点击Next



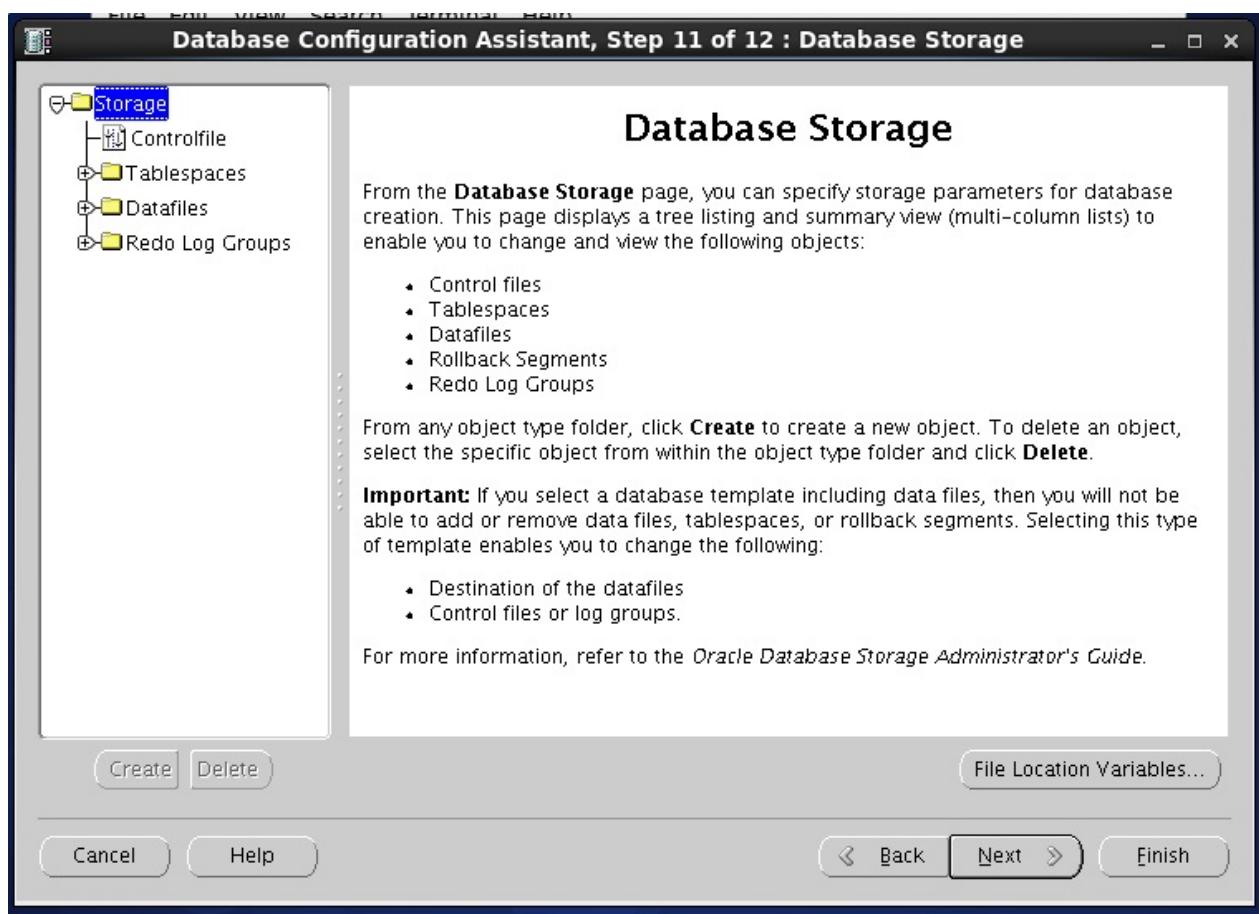
默认所选，点击Character Sets



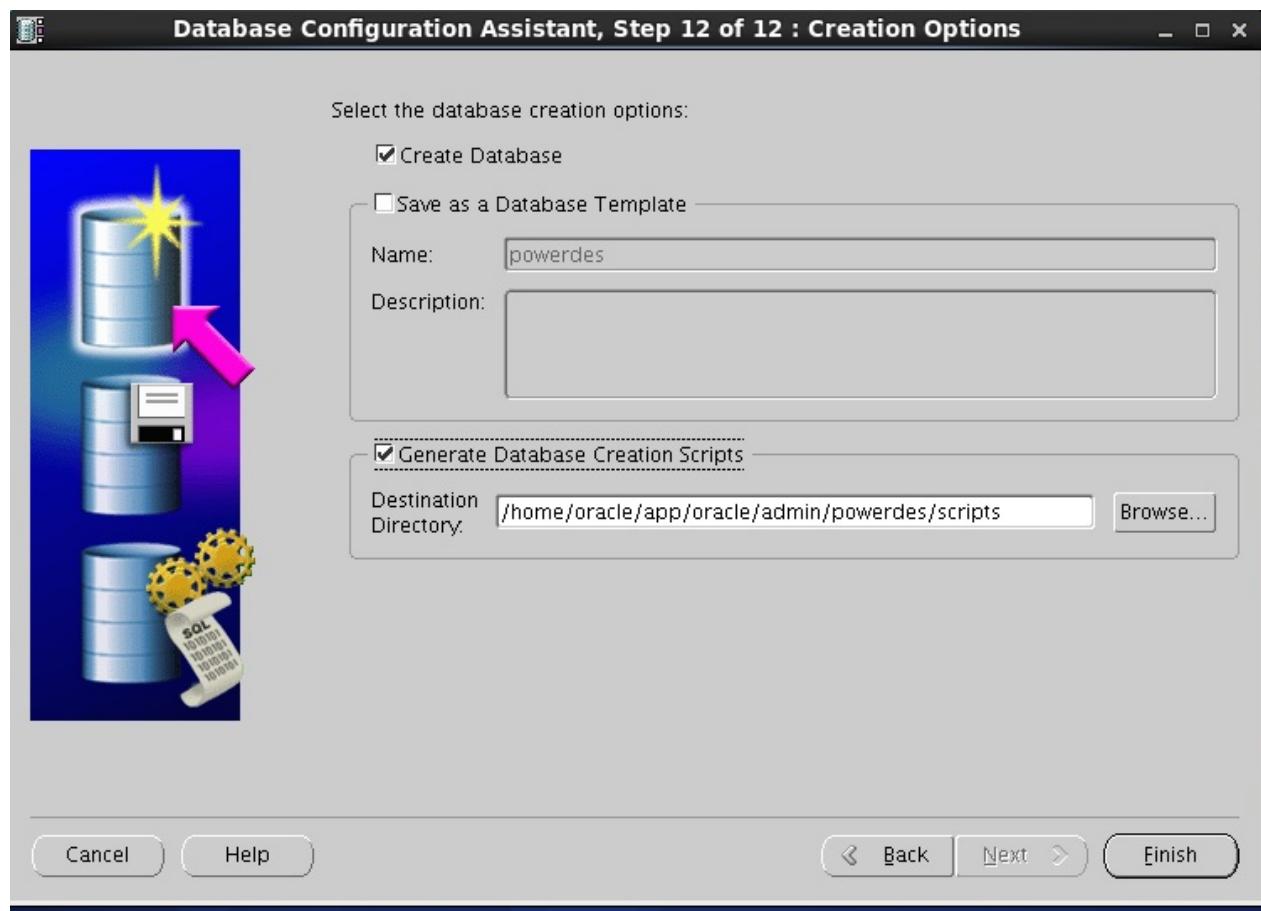
按照图示选择兼容utf8的选项，点击Next



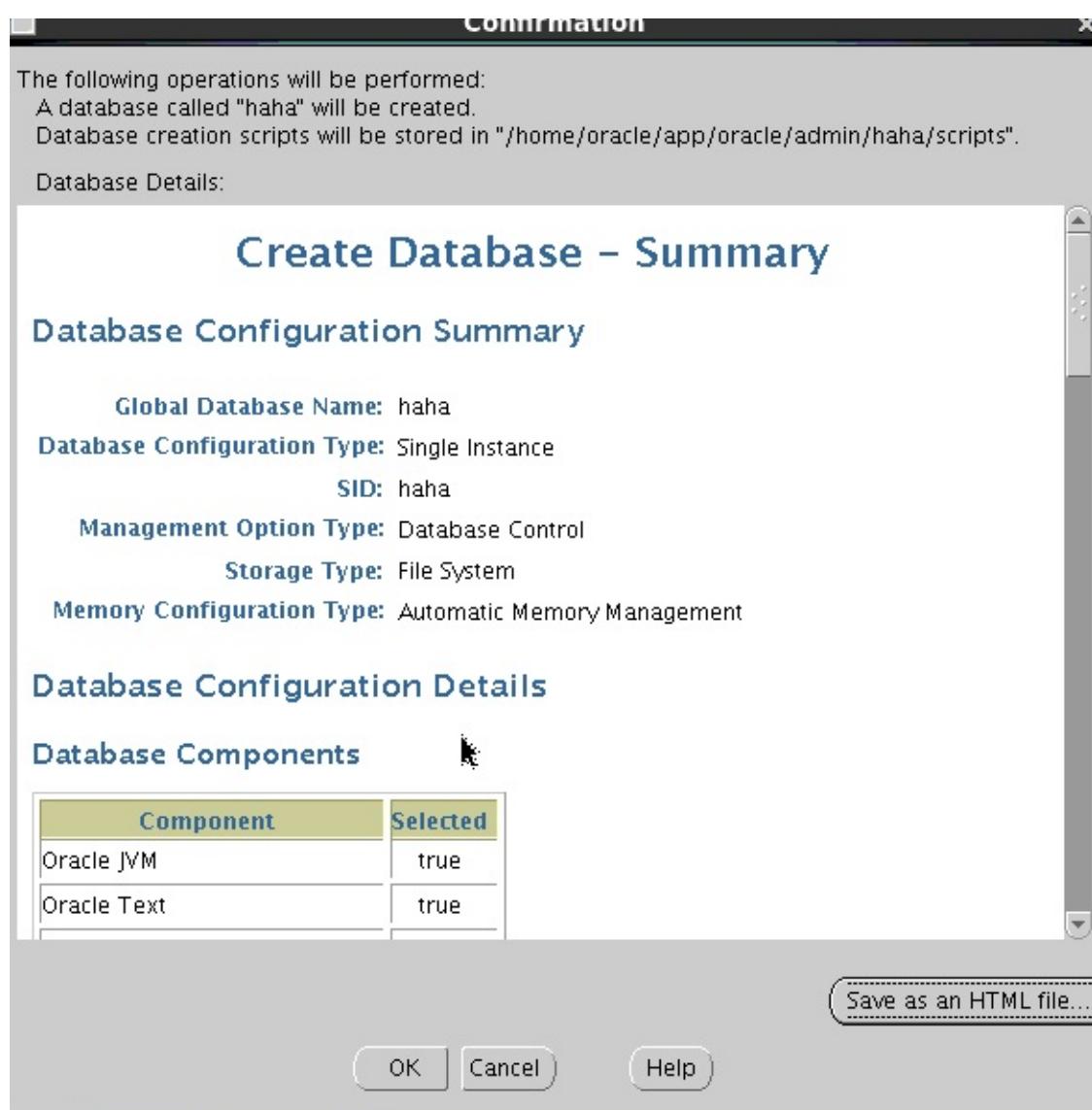
在File Location Variables里可以看到一些基础参数信息。点击Next



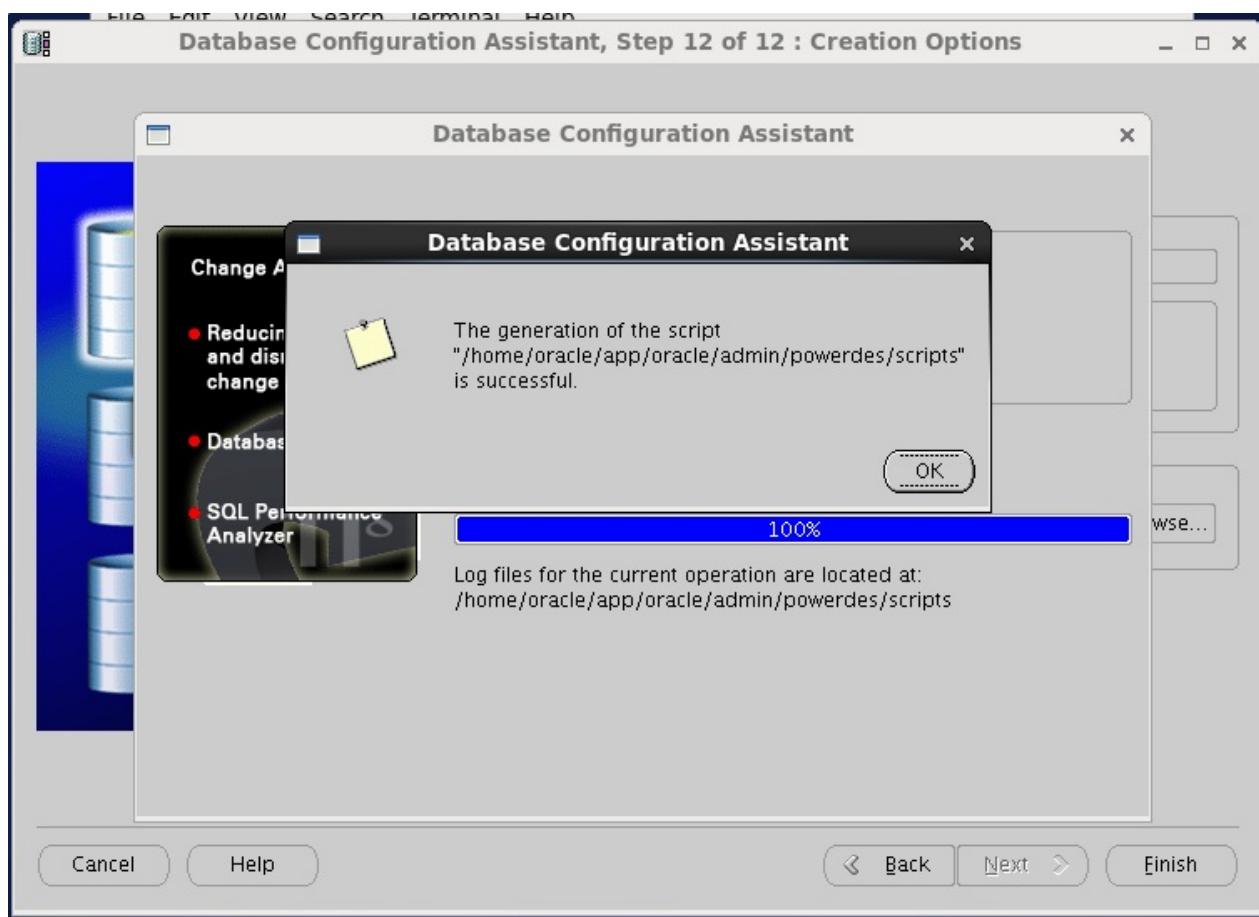
选择Generate Database Creation Scripts，默认  
为/home/oracle/app/oracle/admin/powerdes/scripts，点击Finish



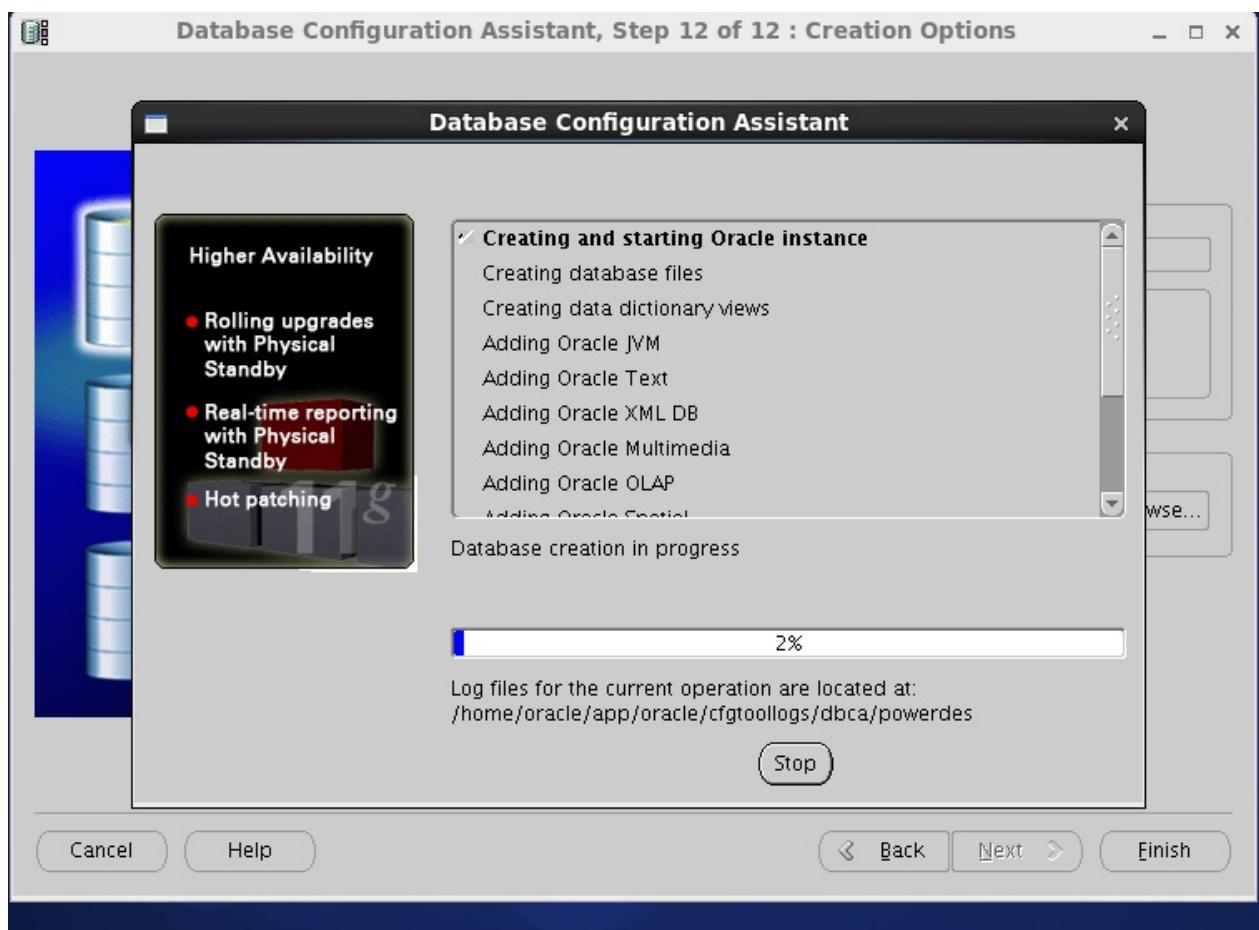
点击OK



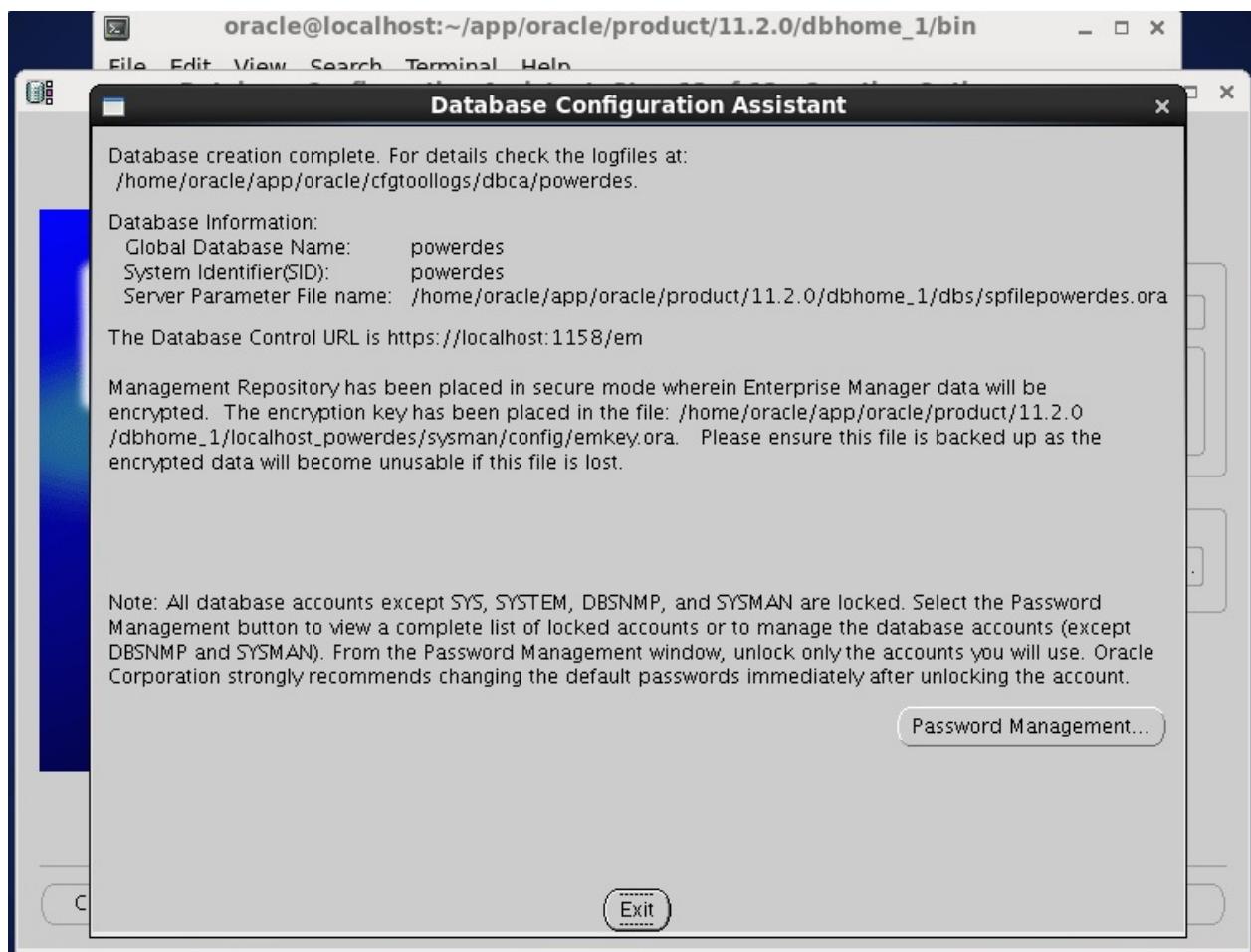
点击OK



等待创建



点击Exit完成创建



创建完数据库后启动数据库

如果在切换用户进入数据库时，执行lsnrctl start或是执行sqlplus "/ as sysdba"报出如下错误

```
bash:lsnrctl:command not found.  
bash:sqlplus:command not found.
```

可能是因为在切换oracle用户时没有加“-”，如#su - oracle。

注：su 和 su - 的区别

su是不更改环境变量的，而su -是要更改环境变量的。也就是说su只是获得了oracle的权限，su -是切换到oracle并获得oracle的环境变量及执行权限。

### (3)Linux下操作数据库的步骤：

- I.以oracle用户登录
- II.执行：lsnrctl start (启动监听)
- III.执行：sqlplus /nolog
- IV.执行：sql>conn /as sysdba
- V.执行：sql>startup
- VI.进行数据库操作
- VII.执行：sql>shutdown
- VIII.执行：sql>quit
- IX.执行：lsnrctl stop (关闭监听)

注意：如果执行**sql>conn /as sysdba**时出现**Connected to an idle instance.**那么先执行**sql>startup**，后执行**sql>conn /as sysdba**

### (4)创建Oracle的启动脚本

复制以上脚本内容，在/etc/init.d目录下创建一个名为oracle的文件，然后黏贴进去，保存退出，然后chmod +x 给它执行权限，别忘了chkconfig添加开机启动以及防火墙规则的添加，到此数据库就安装完成了。

chkconfig参考：<http://www.cnblogs.com/panjun-Donet/archive/2010/08/10/1796873.html>

```
#!/bin/bash
# chkconfig: 2345 90 10
export ORACLE_BASE=/home/oracle/app/oracle
export ORACLE_HOME=/home/oracle/app/oracle/product/11.2.0/dbhome_1
export ORACLE_SID=powerdes
export PATH=$PATH:$ORACLE_HOME/bin
ORCL_OWN="oracle"
# if the executables do not exist -- display error
if [ ! -f $ORACLE_HOME/bin/dbstart -o ! -d $ORACLE_HOME ]
then
    echo "Oracle startup: cannot start"
    exit 1
fi
# depending on parameter -- start, stop, restart
# of the instance and listener or usage display
```

```
case "$1" in
start)
# Oracle listener and instance startup
echo -n "Starting Oracle: "
su - $ORCL_OWN -c "$ORACLE_HOME/bin/dbstart"
touch /var/lock/subsys/oradb
su - $ORCL_OWN -c "$ORACLE_HOME/bin/emctl start dbconsole"
echo "OK"
;;
stop)
# Oracle listener and instance shutdown
echo -n "Shutdown Oracle: "
su - $ORCL_OWN -c "$ORACLE_HOME/bin/emctl stop dbconsole"
su - $ORCL_OWN -c "$ORACLE_HOME/bin/dbshut"
rm -f /var/lock/subsys/oradb
echo "OK"
;;
reload|restart)
$0 stop
$1 start
;;
*)
echo "Usage: 'basename $0' start|stop|restart|reload"
exit 1
esac
exit 0
```

# JFinal 开发环境配置

## 一、安装 Maven

Maven是一个项目构建和管理的工具，提供了帮助管理构建、文档、报告、依赖、scms、发布、分发的方法。可以方便的编译代码、进行依赖管理、管理二进制库等等。

Maven的好处在于可以将项目过程规范化、自动化、高效化以及强大的可扩展，利用 Maven自身及其插件还可以获得代码检查报告、单元测试覆盖率、实现持续集成等等。

### 1. 下载 Maven

<http://maven.apache.org/download.cgi> (Maven官方下载地址)

点击该链接下载	Link	Checksum	Signature
Binary tar.gz archive	<a href="#">apache-maven-3.3.9-bin.tar.gz</a>	apache-maven-3.3.9-bin.tar.gz.md5	apache-maven-3.3.9-bin.tar.gz.asc
Binary zip archive	<a href="#">apache-maven-3.3.9-bin.zip</a>	apache-maven-3.3.9-bin.zip.md5	apache-maven-3.3.9-bin.zip.asc
Source tar.gz archive	<a href="#">apache-maven-3.3.9-src.tar.gz</a>	apache-maven-3.3.9-src.tar.gz.md5	apache-maven-3.3.9-src.tar.gz.asc
Source zip archive	<a href="#">apache-maven-3.3.9-src.zip</a>	apache-maven-3.3.9-src.zip.md5	apache-maven-3.3.9-src.zip.asc

### 2. 将下载后的压缩文件解压到任意目录，这里选择解压到 **D:\Program Files** 目录下，并配置其环境变量（参考配置 JDK 的环境变量）

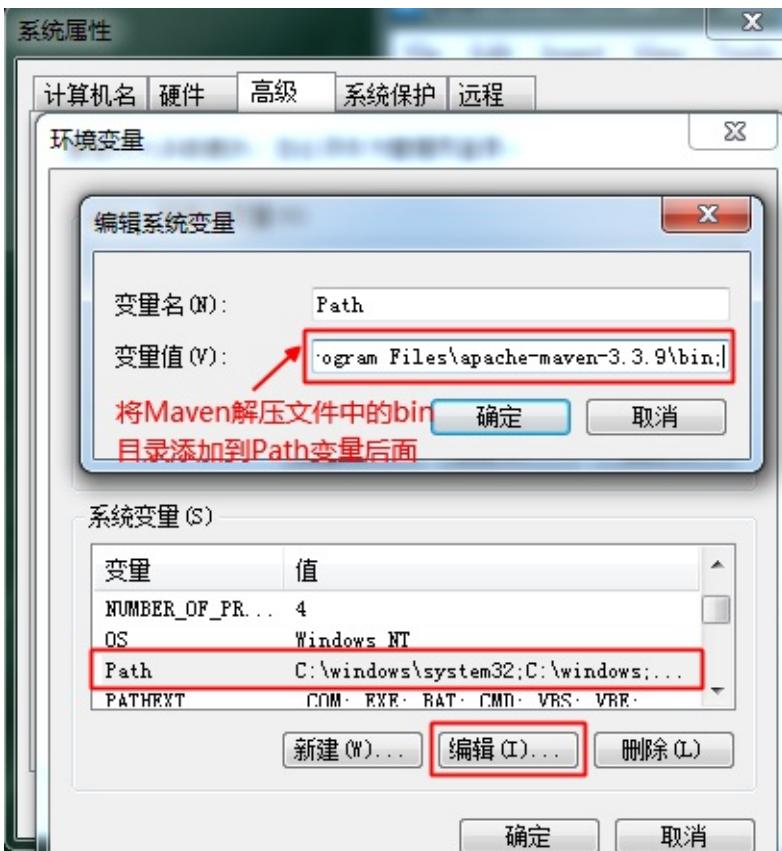
配置 Maven 环境变量前，先确认“JAVA\_HOME”这个环境变量是对应于 JDK 的安装目录。



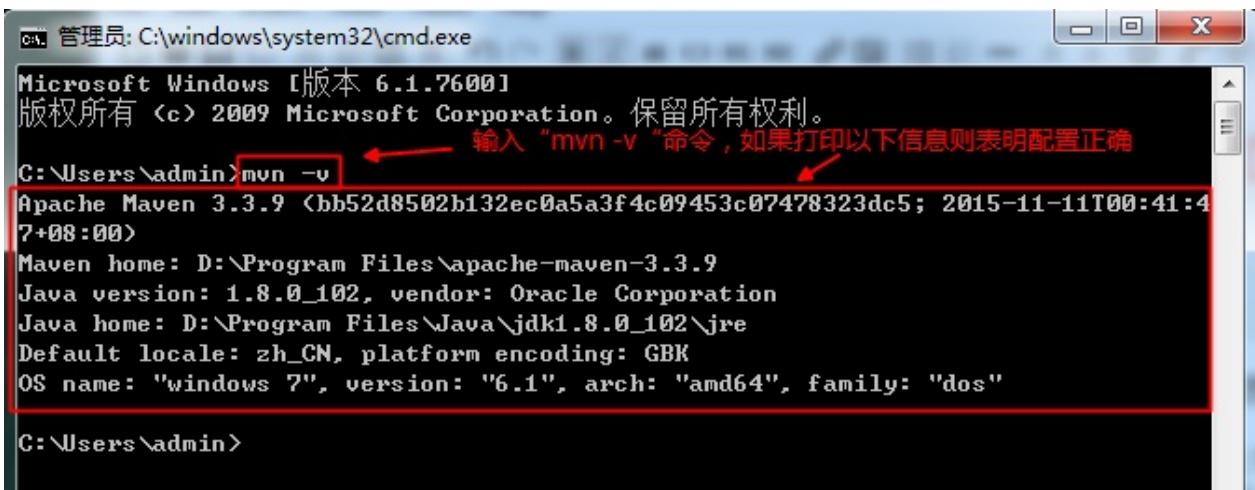
```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7600]
版权所有 © 2009 Microsoft Corporation。保留所有权利。
C:\Users\admin>echo %JAVA_HOME%
D:\Program Files\Java\jdk1.8.0_102
C:\Users\admin>
```

查看“JAVA\_HOME”环境变量的值

配置 Maven 的系统变量。



验证配置是否正确。



修改默认仓库路径。到 D:\Program Files\apache-maven-3.3.9\conf 目录下打开 settings.xml 文件，修改默认的仓库路径。

```

46 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
47   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
48   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
49   <!-- localRepository
50   | The path to the local repository maven will use to store artifacts.
51   |
52   | Default: ${user.home}/.m2/repository-->
53   <localRepository>D:\Program Files\apache-maven-3.3.9\repository</localRepository>
54 
```

自定义仓库存储路径

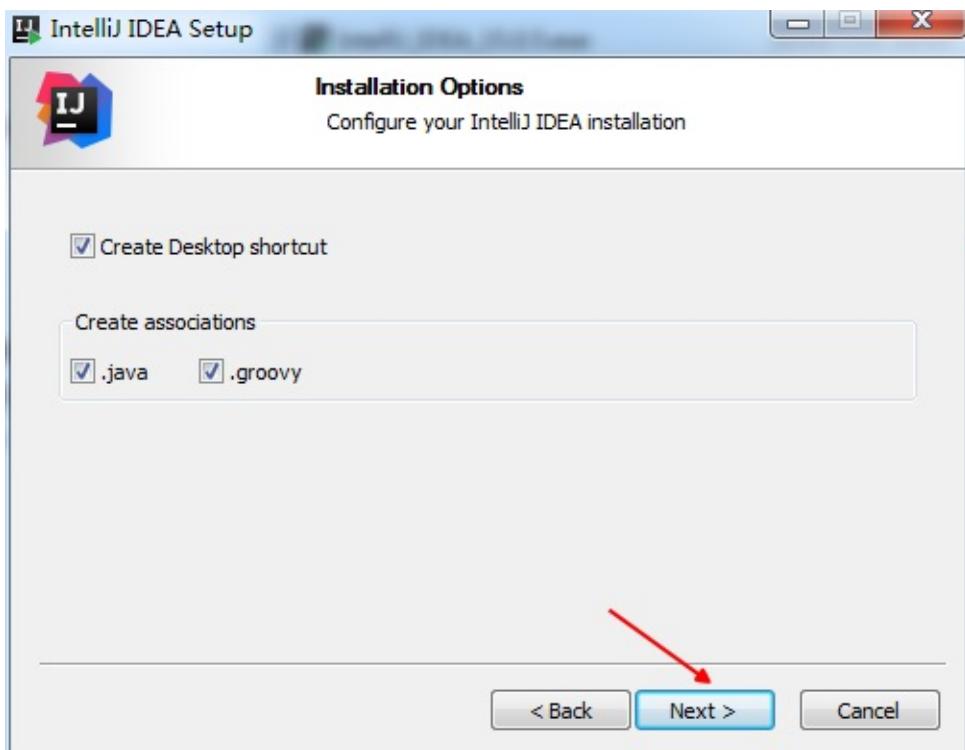
## 二、安装 IDEA

## 1. 下载 IDEA14或者 IDEA15

<http://pan.baidu.com/s/1pLEixej>

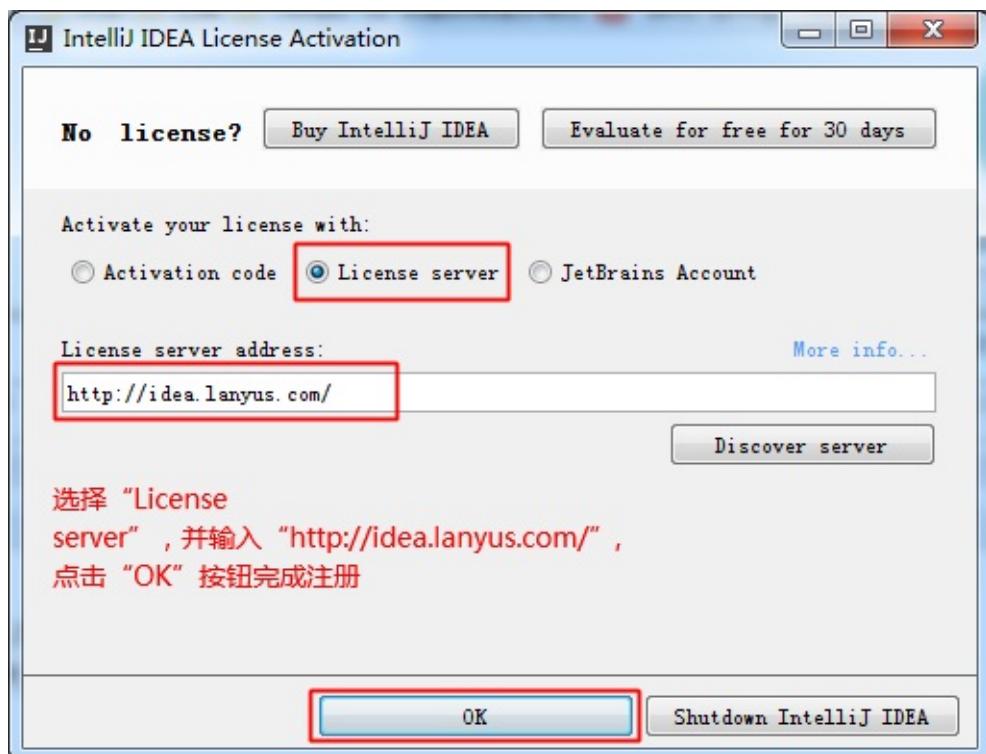
## 2. 鼠标双击安装程序开始安装





### 3. 安装完成后打开IDEA，并完成激活

```
License server address :  
http://idea.lanyus.com/  
  
备用的License server address :  
http://0.idea.lanyus.com  
http://1.idea.lanyus.com  
http://2.idea.lanyus.com
```

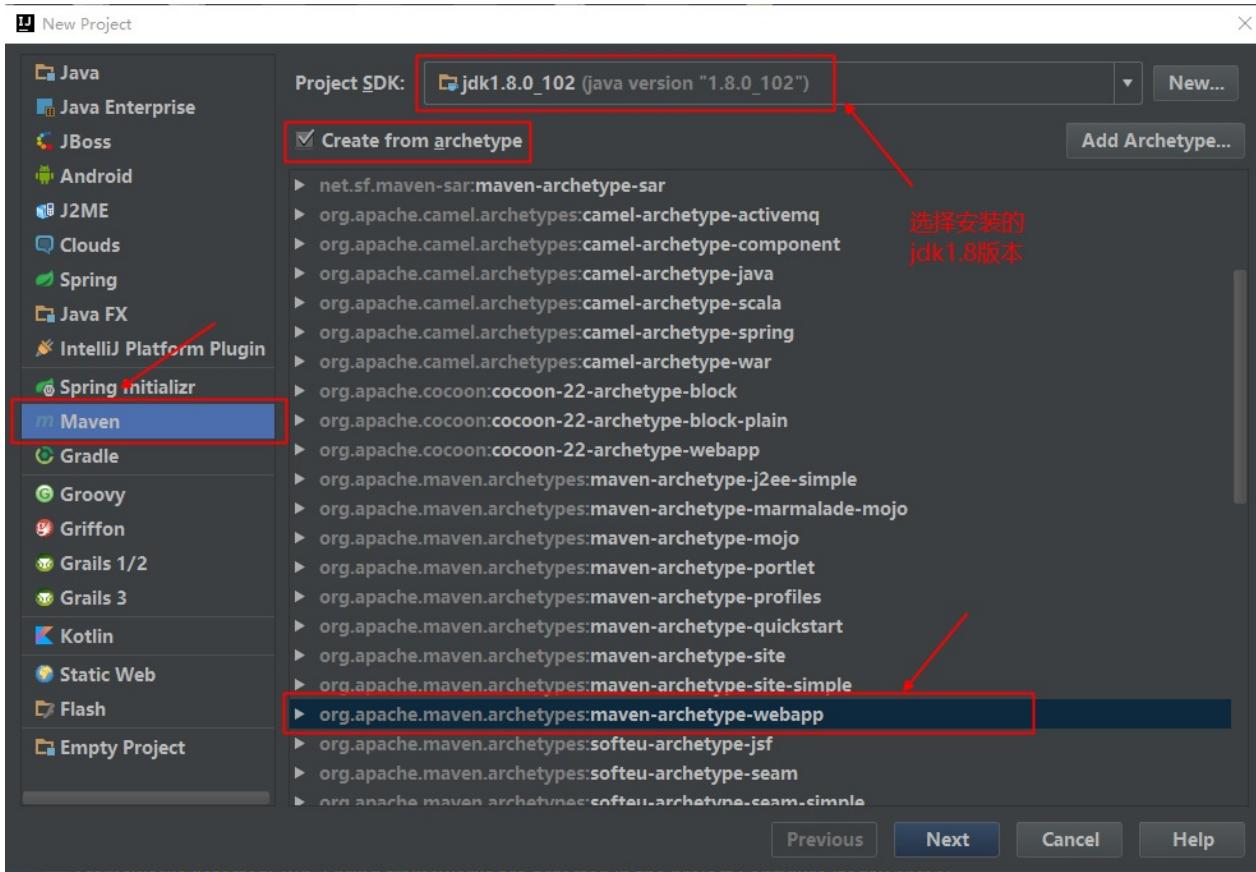


选择“License server”，并输入“<http://idea.lanyus.com/>”，点击“OK”按钮完成注册

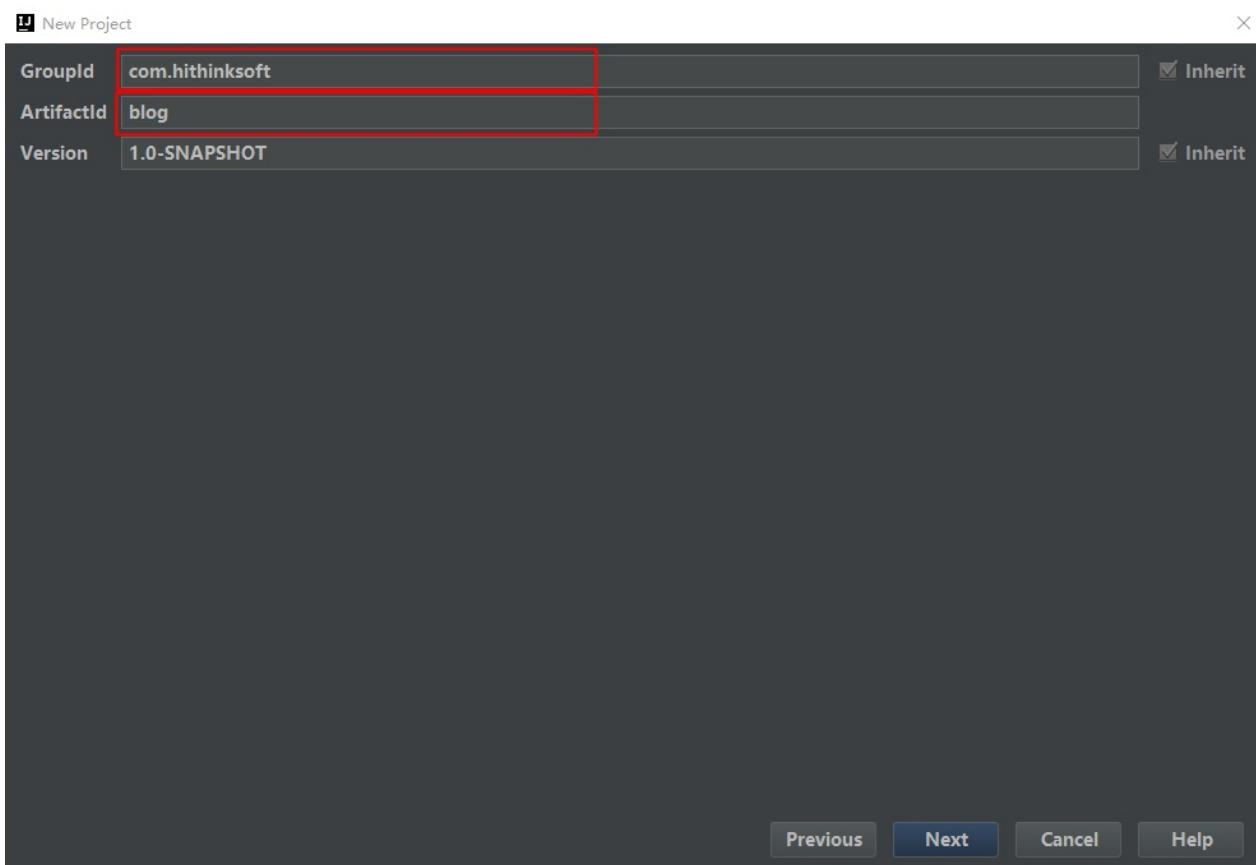
### 三、创建一个 JavaWeb 工程项目

#### 1. 新建项目

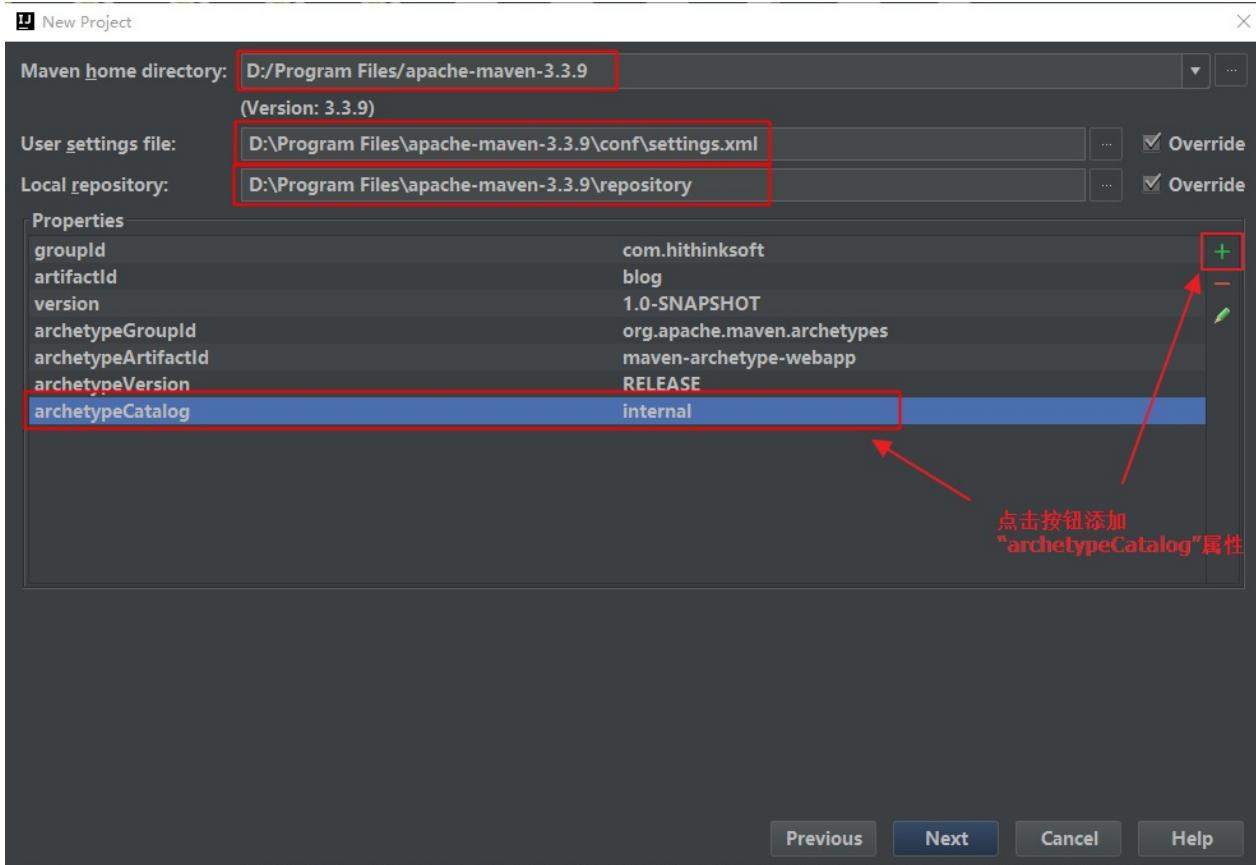
点击 IDEA 菜单栏左上角的“File”选项，在出现的下拉列表中选择第一项“New”，在“New”选项的右边再选择“Project”选项打开“New Project”对话框。



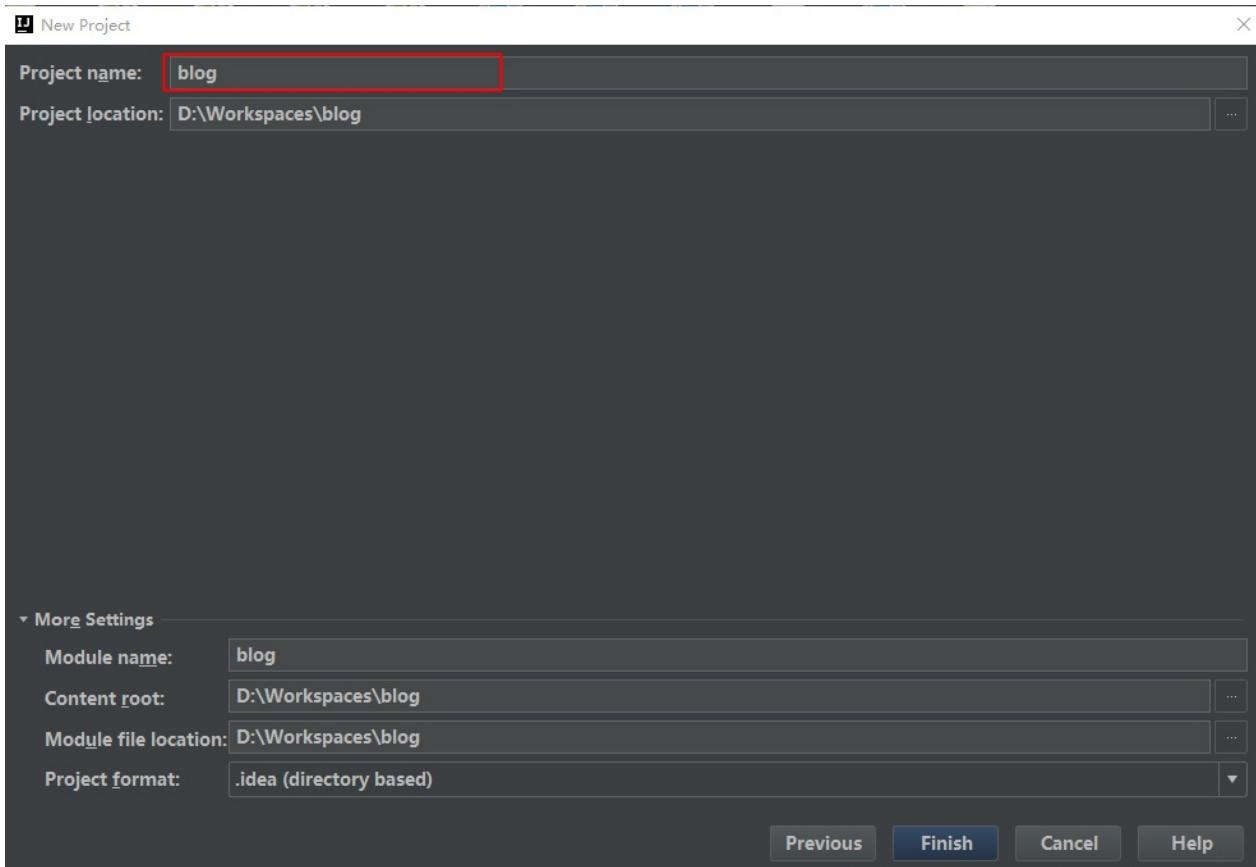
设置 GroupId 和 ArtifactId。



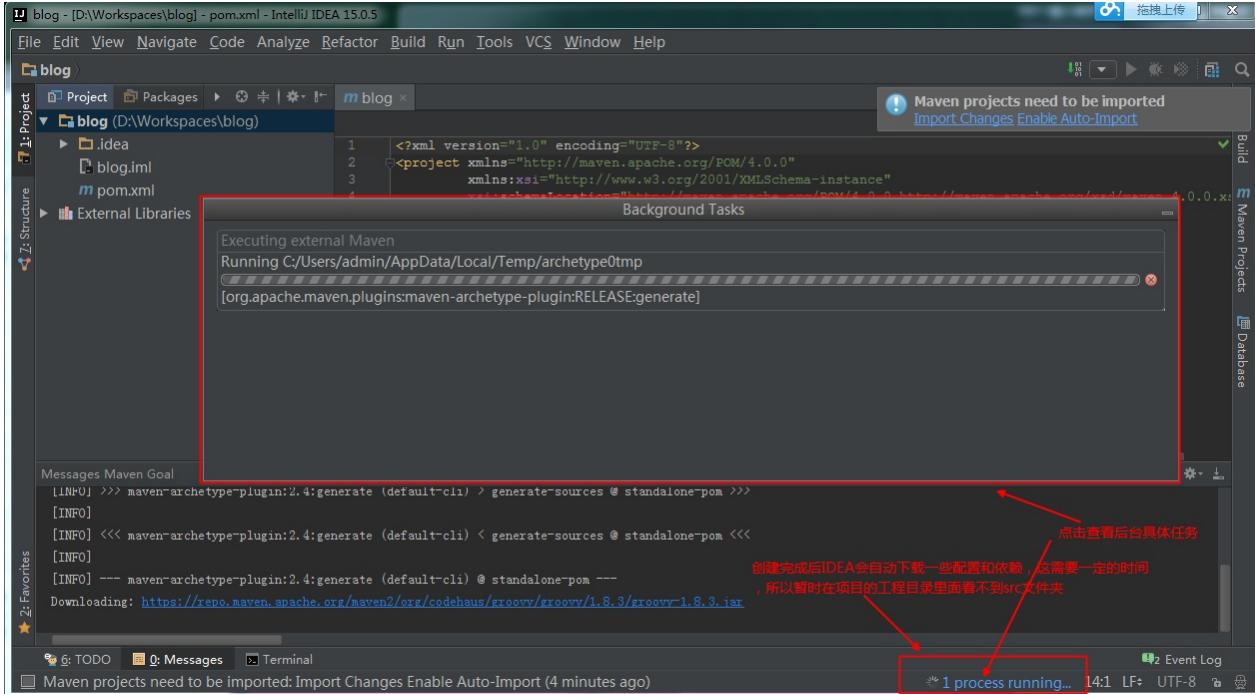
设置 Maven 的安装目录、`settings.xml` 的文件位置和本地仓库的位置等信息。



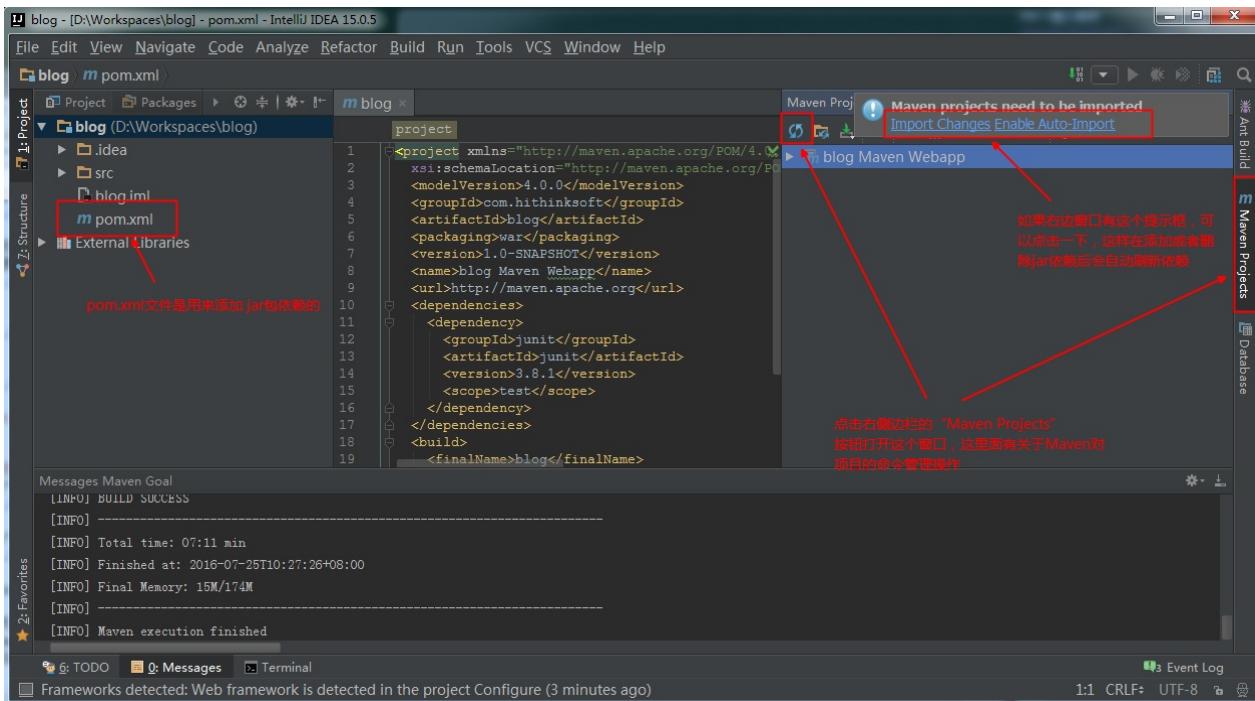
设置项目名称。



项目创建完成。



## 2. 在 pom.xml 文件中添加 jar 包依赖



pom.xml文件说明：

project：pom.xml文件中的顶层元素；

modelVersion：指明 POM使用的对象模型的版本。这个值很少改动。

groupId：指明创建项目的组织或者小组的唯一标识。GroupId是项目的关键标识，典型的，此标识以组织的完全限定名来定义。比如，org.apache.maven.plugins是所有Maven插件项目指定的 groupId。

artifactId：指明此项目产生的主要产品的基本名称。项目的主要产品通常为一个 JAR文件。第二，象源代码包通常使用 artifactId作为最后名称的一部分。典型的产品名称使用这个格式： - . (比如：myapp-1.0.jar)。

version：项目产品的版本号。Maven帮助你管理版本，可以经常看到 SNAPSHOT这个版本，表明项目处于开发阶段。

name：项目的显示名称，通常用于 maven产生的文档中。

url：指定项目站点，通常用于 maven产生的文档中。

description：描述此项目，通常用于 maven产生的文档中。

要添加依赖我们要先到 maven仓库里面去查询。

[Maven仓库地址：http://mvnrepository.com/](http://mvnrepository.com/)

首先我们要添加一些基本的依赖关系如：

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

Maven Repository: javax... mvnrepository.com/search?q=javax.servlet.api

**Found 26 results**

输入依赖的groupId或者artifactId等信息进行查询

1. Java Servlet API  
javaservlet > javax.servlet-api  
Java Servlet API      3,466 usages  
GPL | CDDL

2. JavaServlet(TM) Specification  
javaservlet > servlet-api  
JavaServlet(TM) Specification      5,621 usages  
GPL | CDDL

3. JavaServer Pages(TM) API  
javaservlet.jsp > jsp-api  
JavaServer Pages(TM) API      678 usages  
GPL | CDDL

Popular Categories: Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, Bytecode Libraries, Command Line Parsers.

C Maven Repository: javax... mvnrepository.com/artifact/javaservlet/javax.servlet-api

**Java Servlet API**

Java Servlet API

License	CDDL 2   GPL		
Categories	Java Specifications		
Tags	api   javax   servlet   specs   standard		
Usages	3,466		
短信通知、语音通知、验证码  [免费]发送222条短信			
java.servlet.api的所有版本都在这里，我们选择一个合适的版本，点击进入			
Version	Usages	Type	Date
4.0.x	4.0.0-b01	3	beta (Oct, 2015)
	3.1.0	1,575	release (Apr, 2013)
	3.1-b09	0	beta (Apr, 2013)
	3.1-b08	3	beta (Apr, 2013)
	3.1-b07	4	beta (Mar, 2013)
	3.1-b06	3	beta (Feb, 2013)

等待 pagead2.googlesyndication.com...

**Note:** There is a new version for this artifact  
New Version 4.0.0-b01

**Java Servlet API > 3.1.0**

<b>License</b>	CDDL 2   GPL
<b>Categories</b>	Java Specifications
<b>Organization</b>	GlassFish Community
<b>HomePage</b>	<a href="http://servlet-spec.java.net">http://servlet-spec.java.net</a>
<b>Date</b>	(Apr 25, 2013)
<b>Repository</b>	central
<b>Usages</b>	3,466

**Maven** (selected) | Gradle | SBT | Ivy | Gape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

Include comment with link to declaration

选择 Maven 复制依赖关系，将它粘贴到 pom.xml 文件的 <dependencies></dependencies> 标签中

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

blog - [D:\Workspaces\blog] - blog - IntelliJ IDEA 15.0.5

blog pom.xml

Project Structure

Project: blog (D:\Workspaces\blog)

Dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.hithinksoft</groupId>
    <artifactId>blog</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>blog Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>unit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <finalName>blog</finalName>
    </build>
</project>
```

Maven Projects

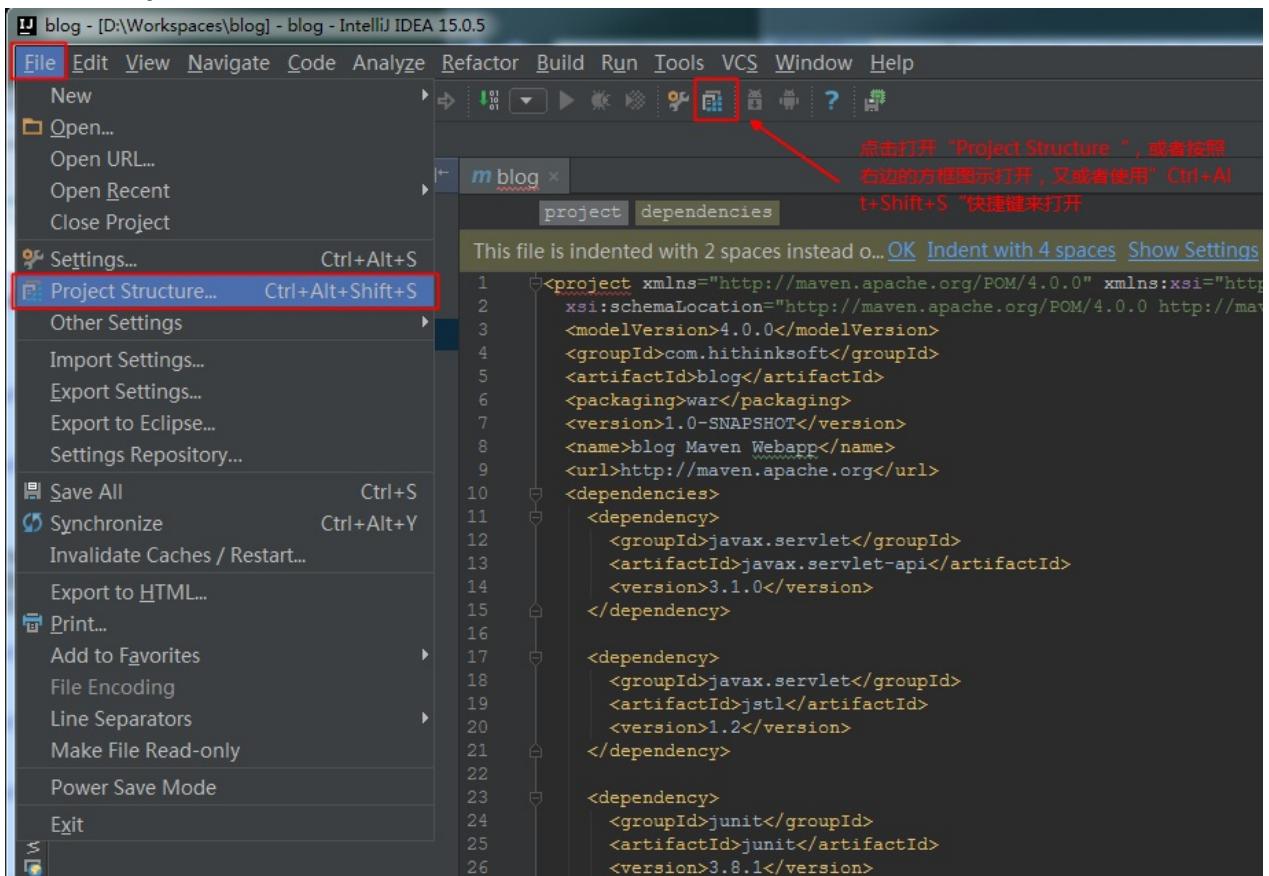
blog Maven Webapp

- Lifecycle
- Plugins
- Dependencies
  - javax.servlet:javax.servlet-api:3.1.0
  - junit:junit:3.8.1 (test)

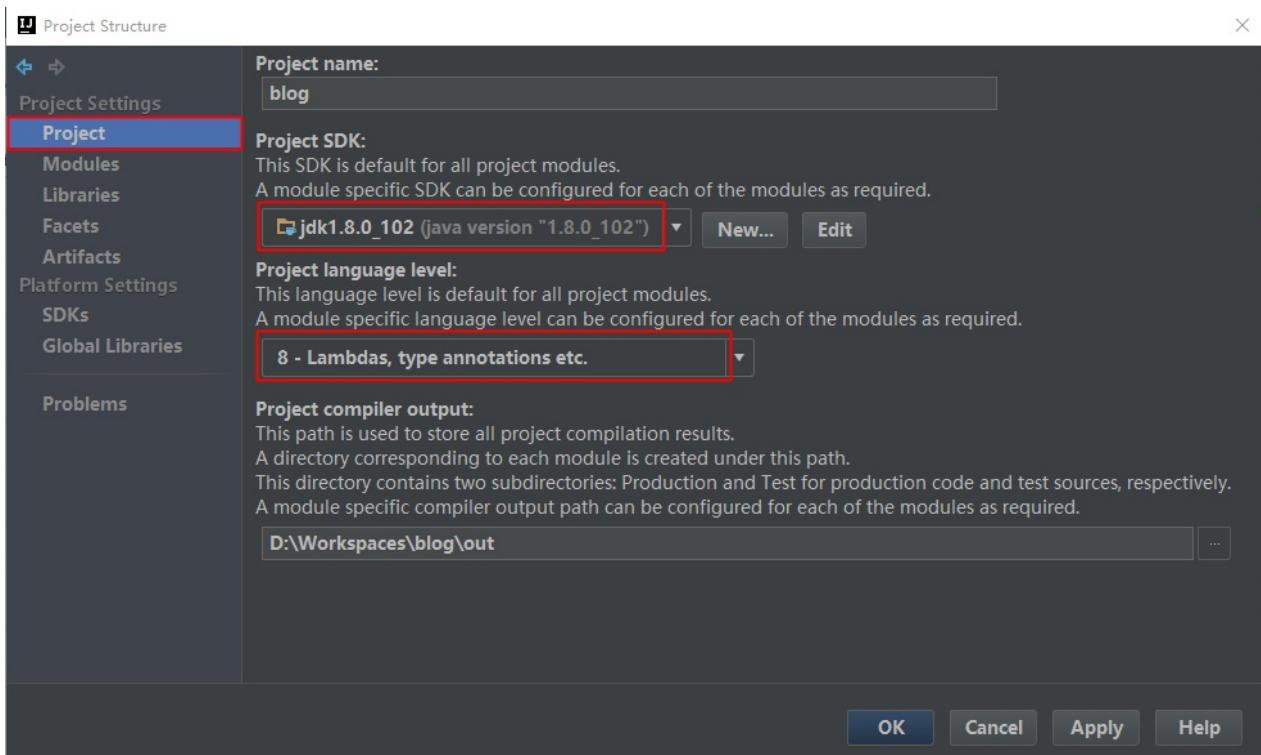
粘贴依赖关系后，IDEA会根据其中的groupId、artifactId和version来唯一匹配javax.servlet-api-3.1.0.jar这个jar包，并自动从远程的Maven仓库中下载这个jar包。

### 3. 配置项目

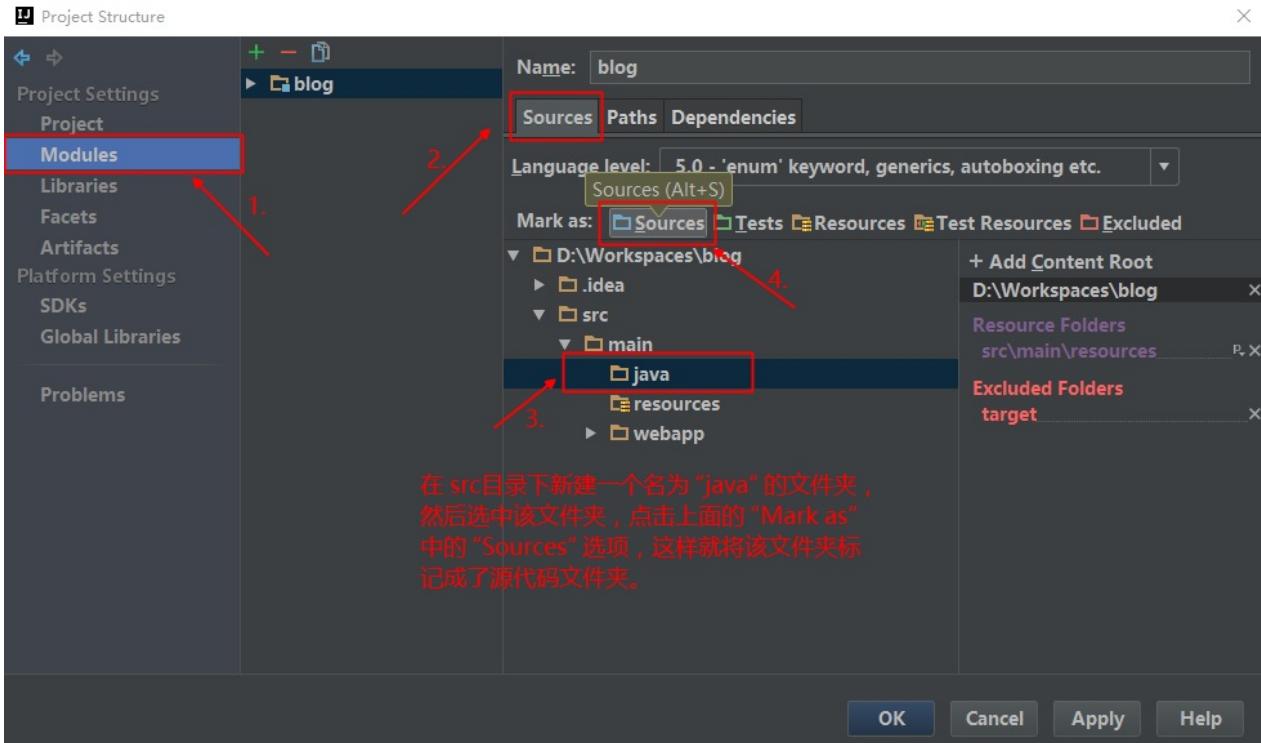
## 打开“Project Structure”。



## 配置“Project”。

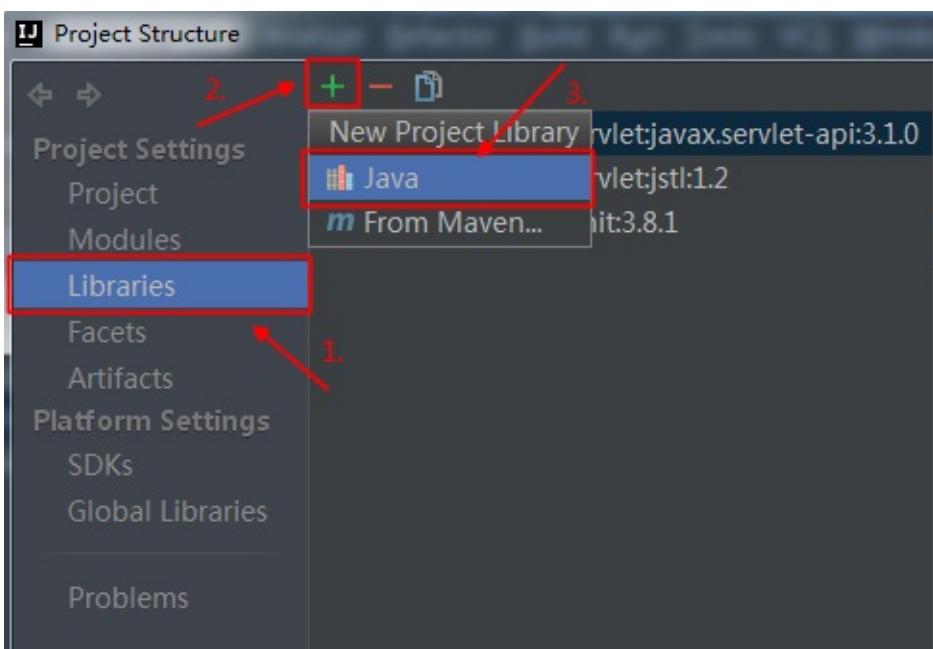


## 配置“Modules”。

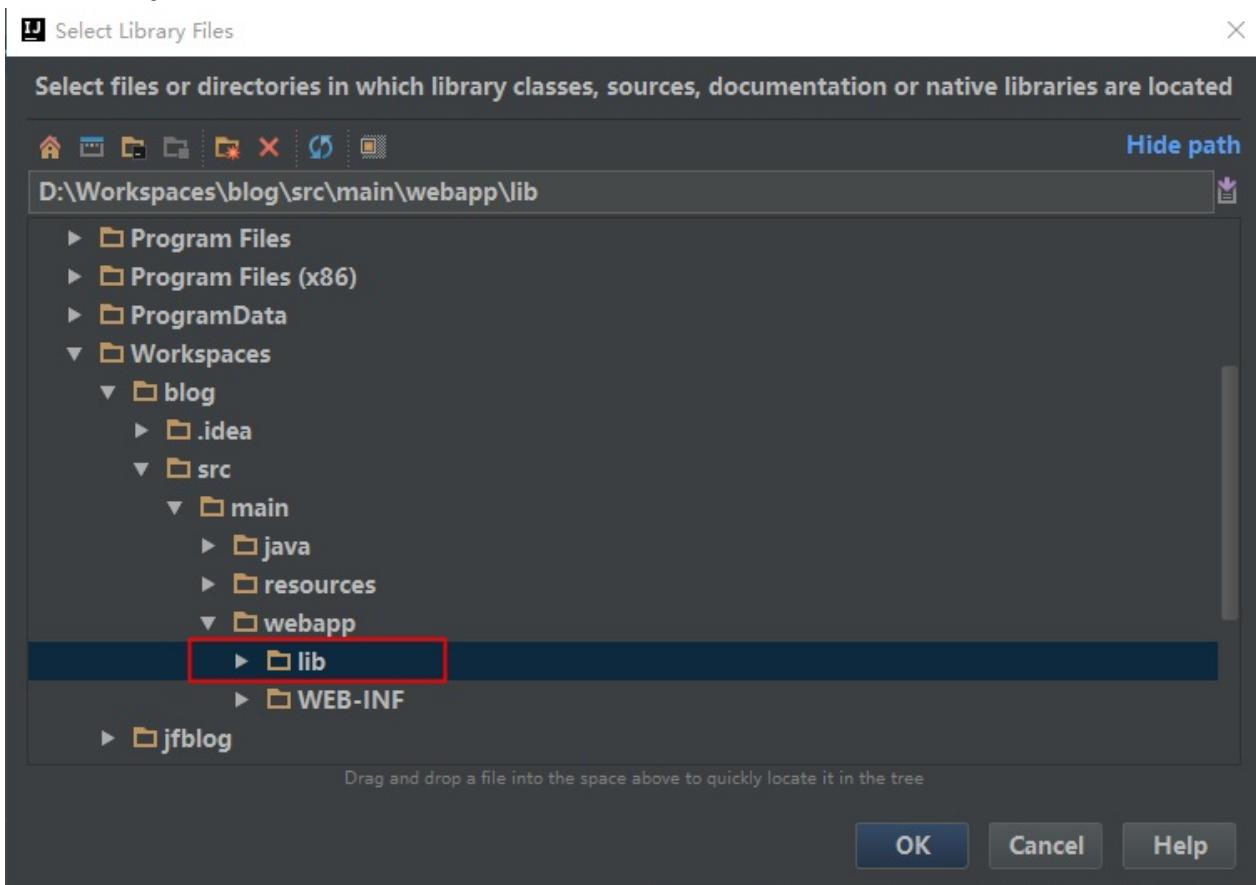


## 配置“Libraries”。

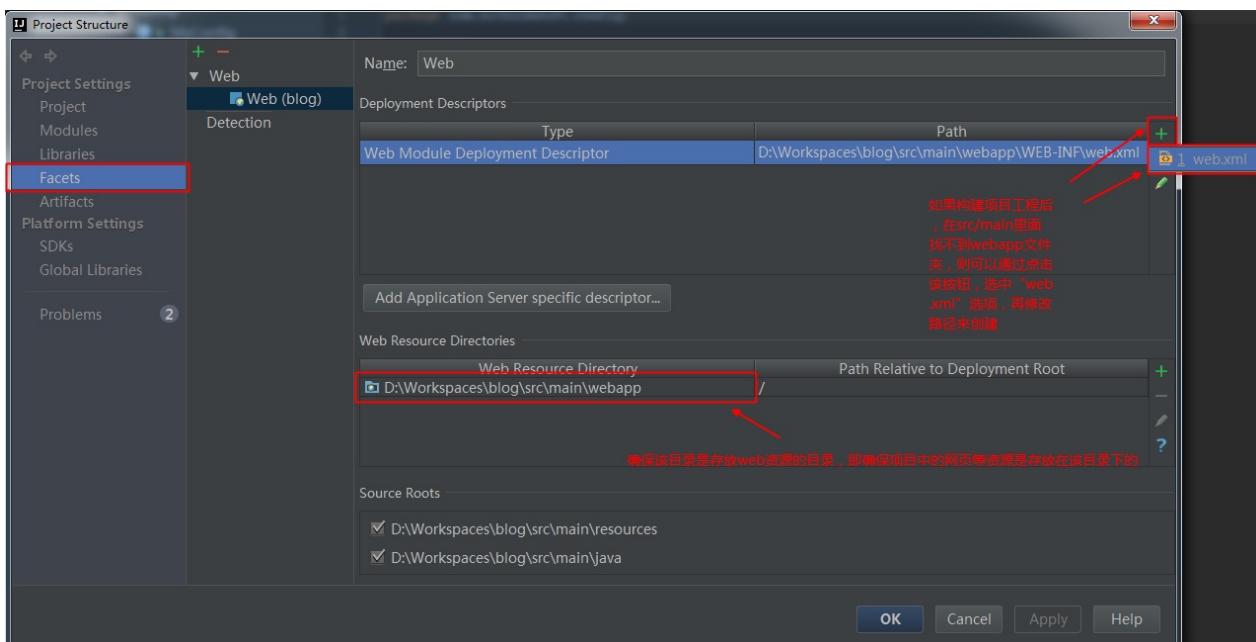
添加外部依赖的jar包。有时候将一些本地已存在的jar包添加到了src/main/webapp/WEB-INF/lib（该目录要手动创建）这个目录里，那么可以在那里设置将这些jar包导进项目中而不需要通过在pom.xml中去添加这些jar包的依赖关系来导入。



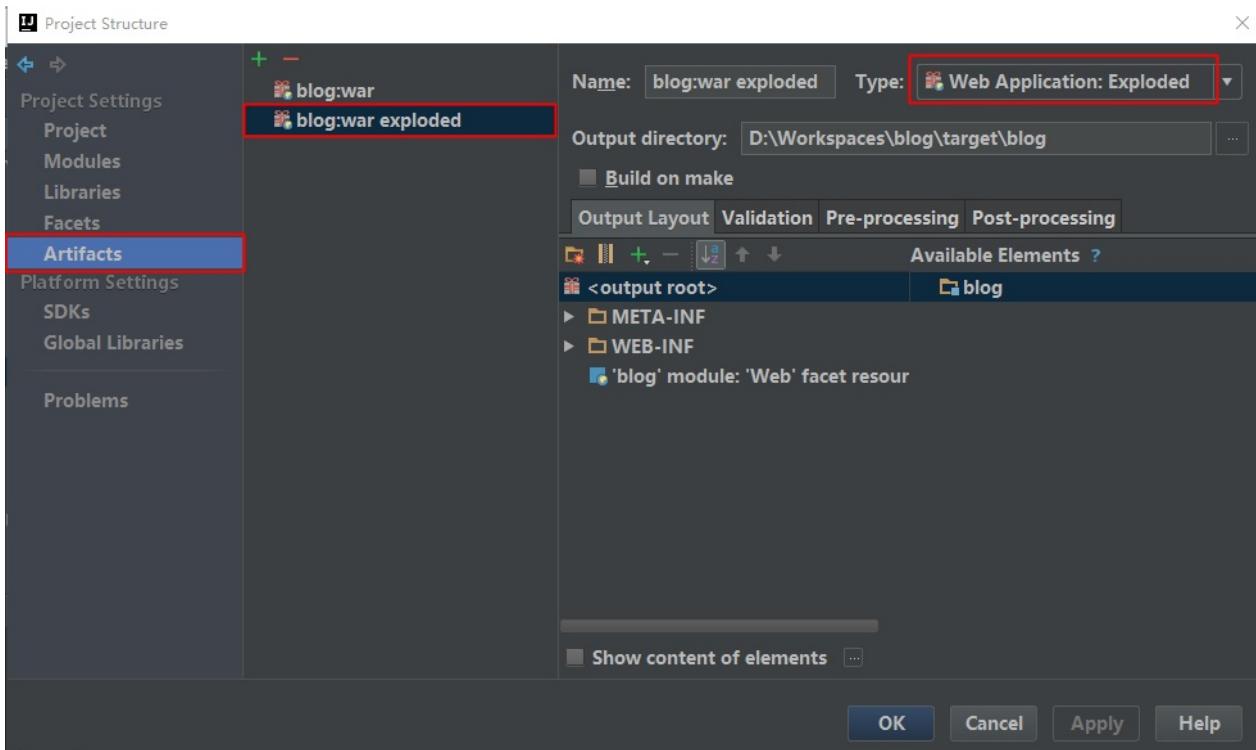
选中存放 jar包的文件夹。



配置“Facets”。

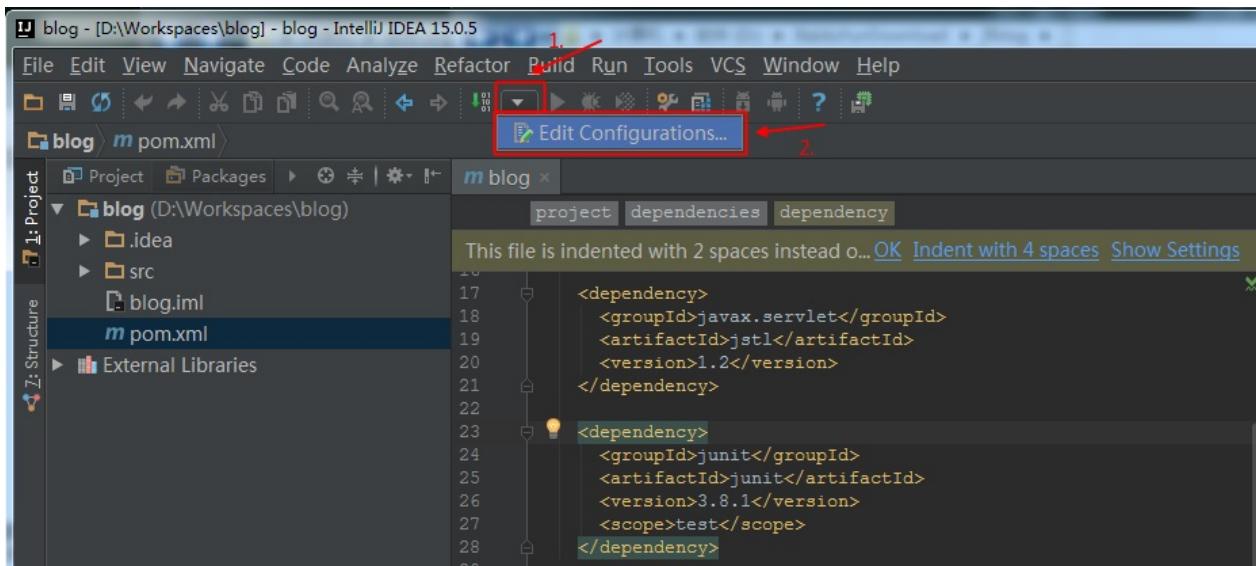


配置“Artifacts”。

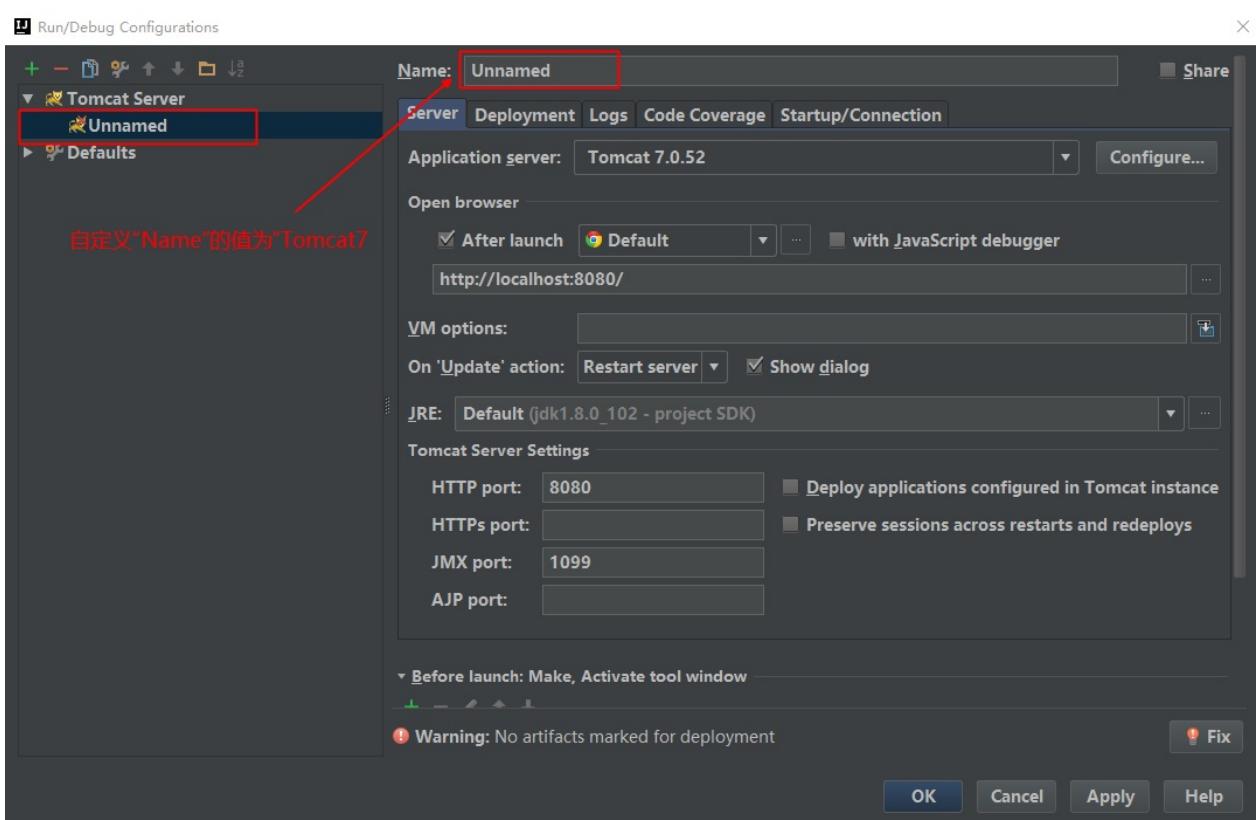
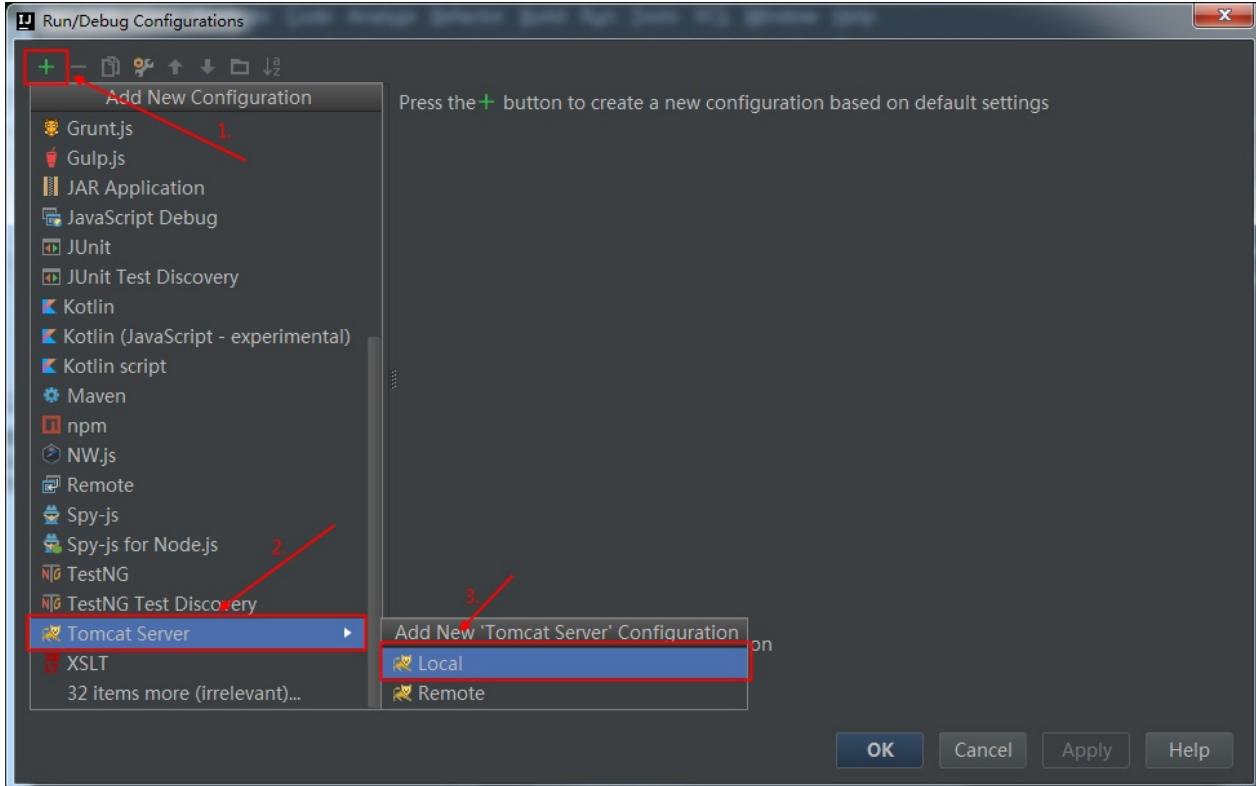


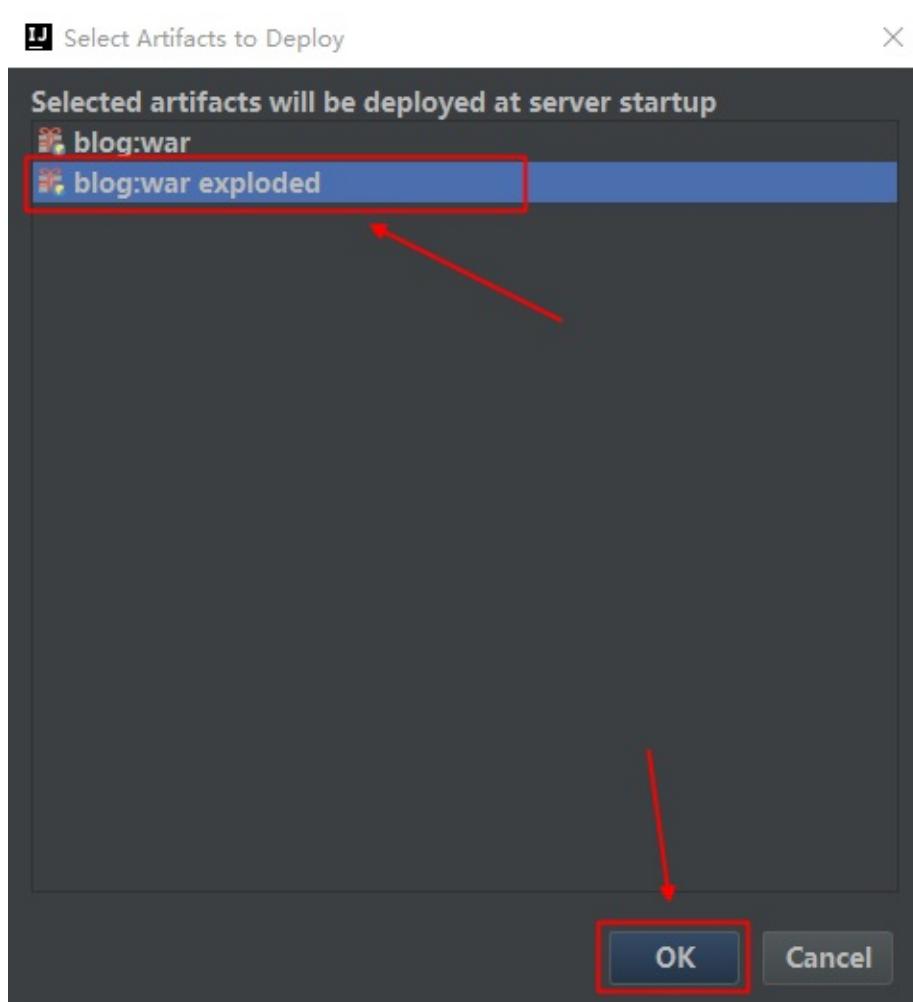
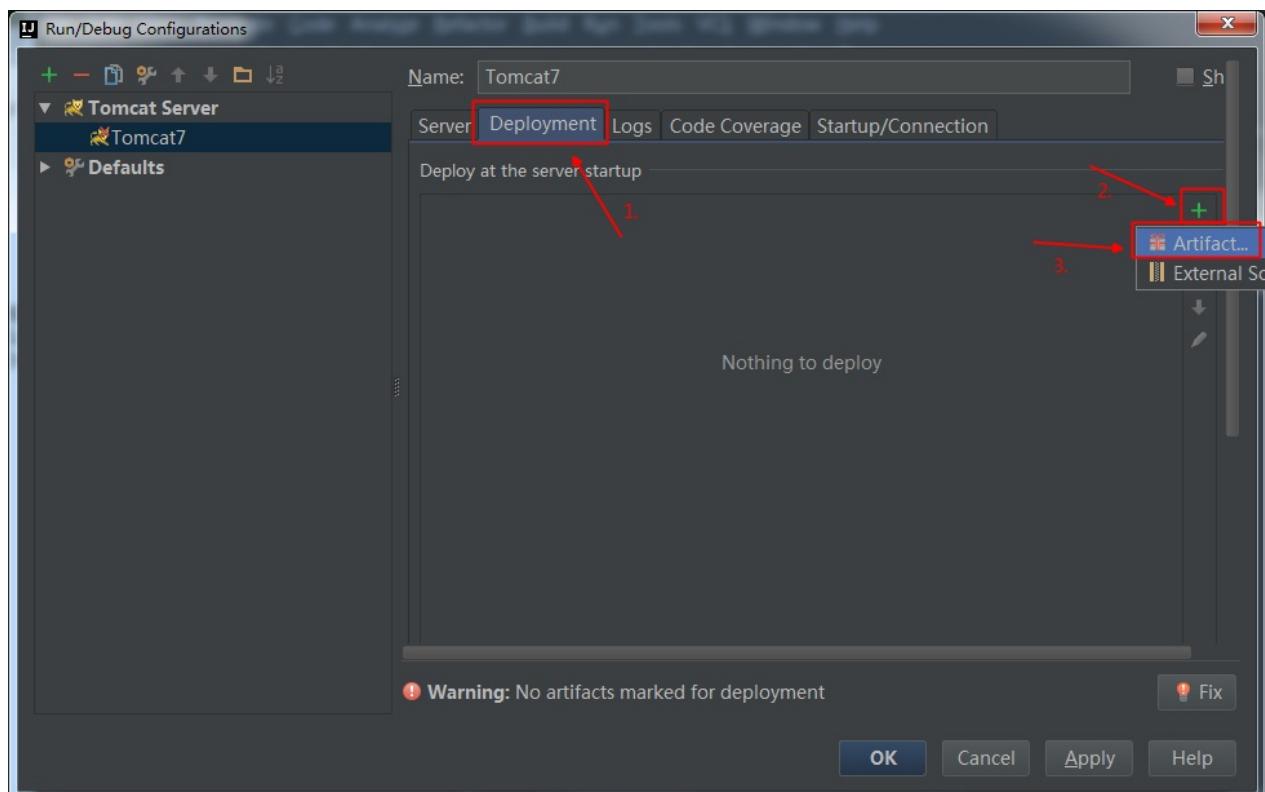
### 3. 配置Tomcat

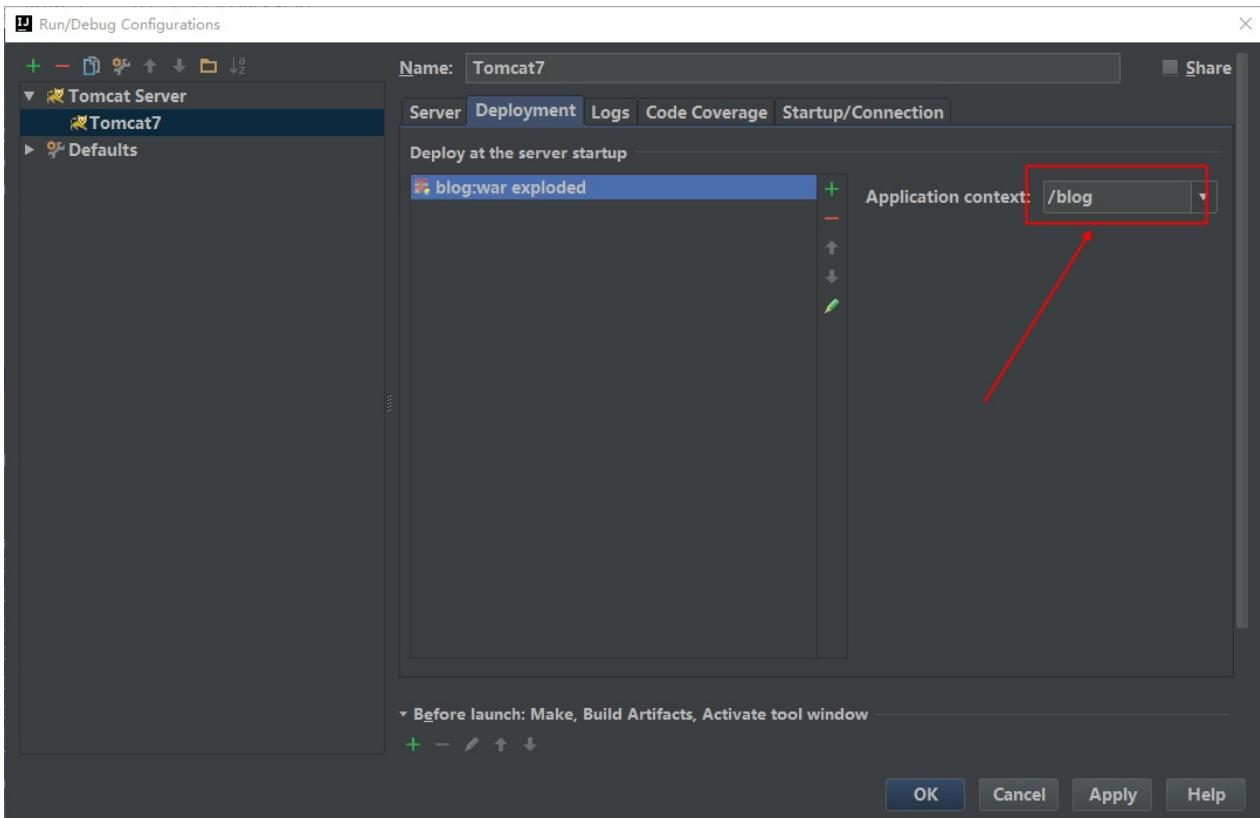
打开配置。



添加新Tomcat Server配置。







启动 Tomcat。

正确启动后会默认显示index.jsp页面的内容



## 四、简单搭建 **JFinal** 框架（搭建 **JFinal** 框架之前请先学习 **JFinal** 官方手册）

JFinal官网：<http://www.jfinal.com/>

详细语法说明请参考 **JFinal** 官方手册：

<http://download.jfinal.com/upload/2.2/jfinal-2.2-manual.pdf>

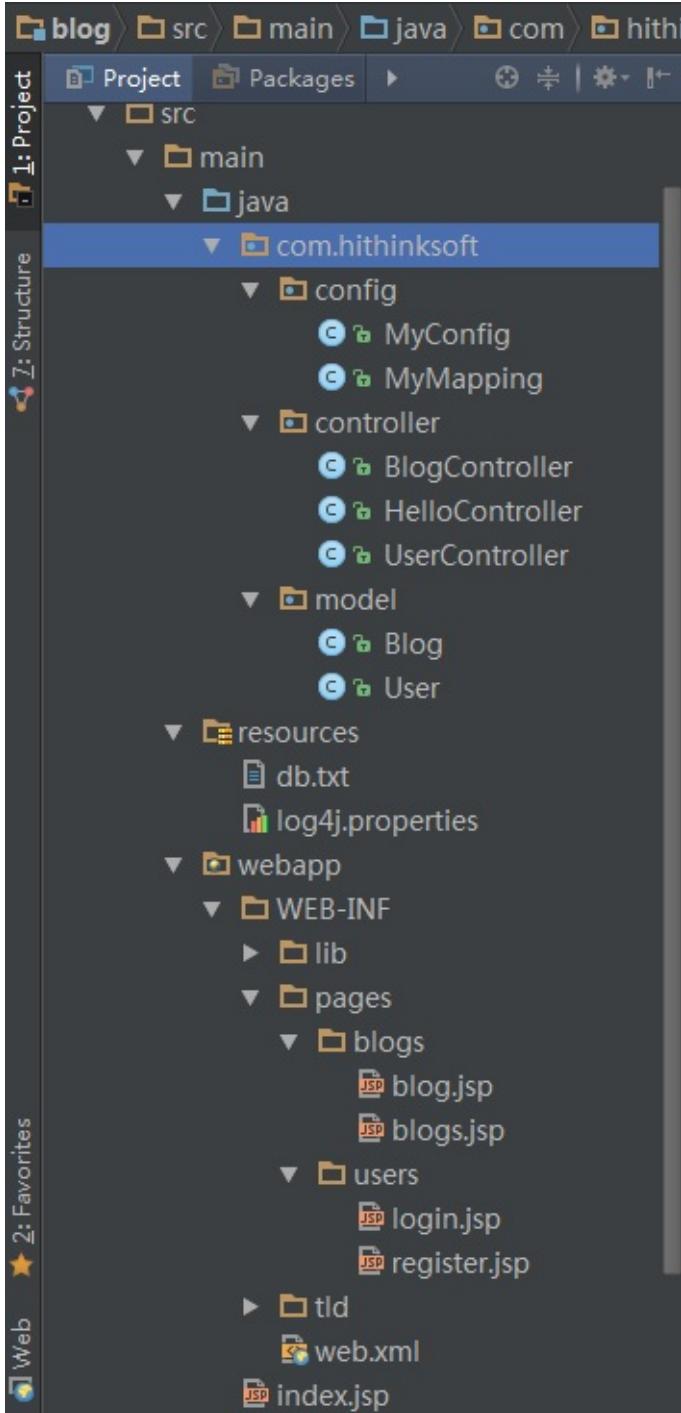
### 1. 搭建 **JFinal** 项目步骤+说明

1. 一个 JFinal 项目一般至少包括三个部分：配置（com.hithinksoft.config）、实

体类(`com.hithinksoft.model`)、控制器 (`com.hithinksoft.controller`) 。

2. 在`com.hithinksoft.config`包中创建一个名为`MyConfig` (类名自定义) 的类，它要继承`com.jfinal.config.JFinalConfig`，并且要实现`JFinalConfig`中的五个抽象方法。这个类是必需的，它会对整个 web 项目进行配置。
3. `com.hithinksoft.model`包下的实体类都要继承`com.jfinal.plugin.activerecord.Model`。
4. `com.hithinksoft.controller`包下的控制器都要继承`com.jfinal.core.Controller`。
5. 下面编写一个示例：
  - 定义两个实体类：`com.hithinksoft.model.User.java`和`com.hithinksoft.model.Blog.java`;
  - 定义两个控制器：`com.hithinksoft.controller UserController.java`和`com.hithinksoft.controller BlogController.java`。
  - 定义页面：在`src\main\webapp\WEB-INF\pages\blogs`下面定义四个页面，`blogs.jsp`、`blog.jsp`、`publish.jsp`和`update.jsp`。`blogs.jsp`页面时展示所有博客信息的，而`blog.jsp`页面是展示某一条博客的详细信息的，`publish.jsp`页面是用来发布博客的，`update.jsp`页面是用来更改博客的；在`src\main\webapp\WEB-INF\pages\users`下定义一个`login.jsp`页面和一个`register.jsp`页面来进行登录和注册。

具体目录结构图：



## 2. 添加依赖

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.0.8</version>
</dependency>

<dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
</dependency>

<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>mchange-commons-java</artifactId>
    <version>0.2.12</version>
</dependency>

<dependency>
    <groupId>com.jfinal</groupId>
    <artifactId>jfinal</artifactId>
    <version>2.2</version>
</dependency>
```

### 3. 配置 web.xml 文件

在 web.xml 中添加一下代码

```
<filter>
    <filter-name>jfinalFilter</filter-name>
    <filter-class>com.jfinal.core.JFinalFilter</filter-class>
    <init-param>
        <param-name>configClass</param-name>
        <param-value>com.hithinksoft.config.MyConfig</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>jfinalFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

上面标签中的和的是一样的，是自定义的。标签中的类就是上面的 MyConfig 类。

#### 4. 创建 **com.hithinksoft.config.MyConfig** 类，继承 **com.jfinal.config.JFinalConfig** 并实现其中的方法

```
package com.hithinksoft.config;

import com.jfinal.config.*;

public class MyConfig extends JFinalConfig {

    @Override
    public void configConstant(Constants constants) {

    }

    @Override
    public void configRoute(Routes routes) {

    }

    @Override
    public void configPlugin(Plugins plugins) {

    }

    @Override
    public void configInterceptor(Interceptors interceptors) {

    }

    @Override
    public void configHandler(Handlers handlers) {

    }
}
```

## 5. 验证项目的搭建是否成功

添加以下内容进行验证：

1. 在 MyConfig类的 configRoute方法中添加一个路由，例如：routes.add("/", HelloController.class)（也可以是别的Controller类）。
2. 在 HelloController.java中添加一个 index方法（方法名称自定义），在 index方

- 法里面添加一条 renderJsp("index.jsp")跳转语句。
3. 在 src\main\webapp\WEB-INF\目录下的 index.jsp 页面中随便添加一些内容。
  4. 启动Tomcat服务器，如果浏览器输入了 index.jsp 页面的内容，则说明 JFinal 项目的搭建成功。

## 6. MyConfig类的具体内容

```

public class MyConfig extends JFinalConfig {
    private static String VIEWURL = "/WEB-INF/pages/";

    @Override
    public void configConstant(Constants constants) {
        /*开发模式常量devMode设置为true，这样程序运行时会打印一些调试信息等*/
        constants.setDevMode(true);

        /*设置编码格式为UTF-8*/
        constants.setEncoding("UTF-8");

        /**
         * 设置视图类型为JSP
         * 如果页面中使用了FreeMarker，则应该写成：
         * constants.setViewType(ViewType.FREE_MARKER);
         */
        constants.setViewType(ViewType.JSP);
    }

    @Override
    public void configRoute(Routes routes) {
        routes.add("/", HelloController.class);
        /**
         * 当在浏览器中访问 "http://localhost:8080/blog/users/login" 时
         *
         * 会映射到 UserController.java 中的 login() 方法。
         * 第三个参数 MyConfig.VIEWURL 是个常量，它的值是： "/WEB-INF/page
         * s/" ,
         * 它表示 Controller 返回的视图的相对路径，
         * 它的作用是：当调用了 UserController 中的 login() 方法的 renderJsp
         * ("users/login.jsp") 语句时，
         * 代码会将 "/WEB-INF/pages/" 与 renderJsp 中的参数 "users/login.js
    }
}

```

```

p"拼接起来得到的
    * 路径作为即将打开的页面的路径。也就是说当在浏览器地址栏中输入了
    * "http://localhost:8080/blog/users/login"之后，浏览器会访问到
    * "/WEB-INF/pages/users/login.jsp"这个页面
    */
routes.add("/users", UserController.class, MyConfig.VIEWURL)
;
routes.add("/blogs", BlogController.class, MyConfig.VIEWURL)
;
}

@Override
public void configPlugin(Plugins plugins) {
    /**
     * 通过 Prokit工具类加载db.txt文件，db.txt是自己创建的文件，里面填写
     * 了一些连接数据库时
     * 必要的参数相似的也可以将其他一些配置信息（如configConstant()方法
     * 中的devMode、视图
     * 类型等等）放到一个文件中，然后通过这种方法来获取其中的参数。
     */
    PropKit.use("db.txt");

    /*从db.txt文件中获取以下四个属性值作为C3p0Plugin的参数来创建一个C3p0
    Plugin对象*/
    C3p0Plugin cp = new C3p0Plugin(PropKit.get("jdbcUrl"),
        PropKit.get("user"),
        PropKit.get("password").trim(),
        PropKit.get("driverClass"));

    /*将C3p0Plugin这个插件初始化完成后添加到工程项目中*/
    plugins.add(cp);

    /**
     * C3p0Plugin是数据源，通过这个数据源创建一个ActiveRecordPlugin对
     * 象
     * ActiveRecordPlugin是ActiveRecord的支持插件，ActiveRecord中定
     * 义了：
     * addMapping(String tableName, Class<? extends Model> model
     * class)方法
     * 该方法建立了数据库表名到Model的映射关系，这样就可以通过调用提供的 A

```

```
PI去操作数据库了
*/
ActiveRecordPlugin arp = new ActiveRecordPlugin(cp);
plugins.add(arp);
arp.addMapping("user", User.class);
arp.addMapping("blog", Blog.class);
}

@Override
public void configInterceptor(Interceptors interceptors) {

}

@Override
public void configHandler(Handlers handlers) {

}
```

## 7. 用户登录

```
webapp/WEB-INF/pages/blogs/login.jsp

<form method="post" action="/blog/users/loginUser"
      style="width:200px; margin:100px auto;">
    帐号 : <input type="text" name="user.username"/><br/><br/>
    密码 : <input type="password" name="user.password"/><br/><br/>
    <input type="submit" value="登录"/>
</form>
```

填写正确信息并点击“登录”按钮后，代码会根据“/blog/users/loginUser”映射到 UserController里面的 loginUser()方法里面继续执行。

```

com.hithinksoft.controller.UserController.java

public void loginUser() {
    User user = getModel(User.class);
    String username = user.getStr("username");
    String password = user.getStr("password");
    /*验证登录的账户和密码是否匹配，如果匹配就获取该 User对象*/
    user = user.validate(username, password);
    if (user != null) {
        setSessionAttr("user", user);
        redirect("/blogs/blogs");
    } else {
        redirect("/users/login.jsp");
    }
}

```

getModel用来接收页面表单域传递过来的model对象，表单域的名称以“modelName.attrName”方式命名，如帐号中 name属性的命令方式为“user.username”，密码中 name属性的命名方式为“user.password”。上述代码中的 getModel(User.class)表示接收页面表单域传递过来的 User实体对象，它里面保存有在页面填写的账户和密码。

redirect("/blogs/blogs")方法的作用是跳转页面的，相当于在浏览器地址栏中输入了“localhost:8080/blog/blogs/blogs”，代码会进入 com.hithinksoft.controller.BlogController 中的 blogs() 方法（注：redirect方法参数中的第一个“/”就代表根路径“localhost:8080/blog/”）。

```

com.hithinksoft.controller.BlogController

public void blogs() {
    //查询所有的博客
    List<Blog> blogs = Blog.blogRepository.find("select * from b
log");
    //将查询到的博客保存进 request作用域中，这样就可以在前端页面中获取到这
些博客
    setAttr("blogs", blogs);
    renderJsp("blogs/blogs.jsp");
}

```

上述代码最后是跳转到“src/main/webapp/WEB-INF/pages/blogs/blogs.jsp”页面。`renderJsp`是`render`系列方法中的一种，它的作用就是渲染不同类型的视图并返回给客户端，简单地说就是按某种视图类型在浏览器中打开一个已经存在的页面。它与前面的`redirect`的区别是`redirect`是用来跳转请求的，`redirect`的请求结果是请求服务器来打开一个页面，而`render`系列方法就是打开一个页面。这跟`Servlet`中的请求分发和重定向类似。

## 8. 浏览博客

```
com.hithinksoft.controller.BlogController

public void blog() {
    Integer blogId = getParaToInt(0);
    Blog blog = Blog.blogRepository.findById(blogId);
    setAttr("blog", blog);
    renderJsp("blogs/blog.jsp");
}
```

跳转页面

```
webapp/WEB-INF/pages/blogs/blog.jsp

<div style="width:600px;margin:50px auto;border:1px solid #A9A9A9;padding:10px 20px;">
    <div style="padding:10px 0px 40px; border-bottom:1px solid #A9A9A9;text-align: center;position:relative;">
        <div>${blog.title}</div>
        <span style="position:absolute;right:40px;bottom:10px;">
作者 : &nbsp;${blog.getUser().username}</span>
    </div>
    <div style="padding:10px 0px;">
        ${blog.content}
    </div>
</div>
```

## 9. 发布博客

```
webapp/WEB-INF/pages/blogs/publish.jsp

<form method="post" action="/blog/blogs/publishBlog/${user.id}"
style="width:600px;margin:50px auto; border:1px solid #A9A9A9; padding:10px 20px;">
    标题：
    <input type="text" name="blog.title" value="${blog.title}"
           style="width: 600px;padding: 4px;"/><br/><br/>

    内容：
    <textarea name="blog.content" style="width:600px;padding:4px
; height:400px;">
        ${blog.content}
    </textarea>
    <input type="submit" value="发布"/>
</form>
```

上述代码也是个表单，所以它的表单元素的 `name` 属性值得写法与 `login.jsp` 页面的写法一样都是“`modelName.attrName`”。当点击“发布”按钮时代码会根据 `action` 中的“`/blog/blogs/publishBlog/${user.id}`”映射到 `BlogController`类中的 `publishBlog` 方法，`action`最后面的“`/${user.id}`”是要传递的参数（具体语法参考 JFinal 官方文档），这个 `user`就是在登录后调用 `setSessionAttr("user", user)`方法保存到 `session`作用域中的 `user`。

```
com.hithinksoft.controller.BlogController

public void publishBlog() {
    //获取 action传递过来的参数
    Integer userId = getParaToInt(0);
    Blog blog = getModel(Blog.class);
    blog.set("user_id", userId);
    //将发布的博客持久化到数据库中
    blog.save();
    redirect("/blogs/blogs");
}
```

## 10. 修改已发布的博客

webapp/WEB-INF/pages/blogs/update.jsp

```
<form method="post" action="/blog/blogs/updateBlog/${blog.id}"  
      style="width:600px;margin:50px auto; border:1px solid #A9A9  
A9;padding:10px 20px;">  
    标题：  
    <input type="text" name="blog.title" value="${blog.title}"  
          style="width: 600px;padding: 4px;"/><br/><br/>  
  
    内容：  
    <textarea name="blog.content" style="width: 600px;padding: 4  
px; height: 400px;">  
        ${blog.content}  
    </textarea>  
    <input type="submit" value="修改"/>  
</form>
```

点击“修改”按钮进入 BlogController中的 updateBlog方法，并且用被修改的博客的 id作为参数。

```
com.hithinksoft.controller.BlogController  
  
public void updateBlog() {  
    Integer blogId = getParaToInt(0);  
    Blog blog = getModel(Blog.class);  
    blog.set("id", blogId);  
    blog.update();  
    redirect("/blogs/blogs");  
}
```

## 11. 删除已发布的博客

```
com.hithinksoft.controller.BlogController

public void delete() {
    Integer blogId = getParaToInt(0);
    Blog.blogRepository.deleteById(blogId);
    redirect("/blogs/blogs");
}
```

## 五、IDEA的常用快捷键

Alt+回车 导入包，自动修正  
Ctrl+N 查找类  
Ctrl+Shift+N 查找文件  
Ctrl+Alt+L 格式化代码  
Ctrl+Alt+O 优化导入的类和包  
Alt+Insert 生成代码(如get, set方法, 构造函数等)  
Ctrl+E或者Alt+Shift+C 最近更改的代码  
Ctrl+R 替换文本  
Ctrl+F 查找文本  
Ctrl+Shift+Space 自动补全代码  
Ctrl+空格 代码提示  
Ctrl+Alt+Space 类名或接口名提示  
Ctrl+P 方法参数提示  
Ctrl+Shift+Alt+N 查找类中的方法或变量  
Alt+Shift+C 对比最近修改的代码  
Shift+F6 重构-重命名  
Ctrl+Shift+先上键  
Ctrl+X 删除行  
Ctrl+D 复制行  
Ctrl+/ 或 Ctrl+Shift+ / 注释 (// 或者/\*...\*/ )  
Ctrl+J 自动代码  
Ctrl+E 最近打开的文件  
Ctrl+H 显示类结构图  
Ctrl+Q 显示注释文档  
Alt+F1 查找代码所在位置  
Alt+1 快速打开或隐藏工程面板  
Ctrl+Alt+ left/right 返回至上次浏览的位置  
Alt+ left/right 切换代码视图  
Alt+ Up/Down 在方法间快速移动定位  
Ctrl+Shift+Up/Down 代码向上/下移动。  
F2 或 Shift+F2 高亮错误或警告快速定位  
代码标签输入完成后，按Tab，生成代码。  
选中文本，按Ctrl+Shift+F7，高亮显示所有该文本，按Esc高亮消失。  
Ctrl+W 选中代码，连续按会有其他效果  
选中文本，按Alt+F3，逐个往下查找相同文本，并高亮显示。  
Ctrl+Up/Down 光标跳转到第一行或最后一行下  
Ctrl+B 快速打开光标处的类或方法



# jFinal快速入门

## 前置技能

1. java基础
2. B/S应用开发经验

## 入门推荐学习路线

强烈要求，看本文档前，至少结合官方demo看过一遍pdf文档

- 官网：<http://www.jfinal.com/>
- 手册：<http://download.jfinal.com/upload/2.2/jfinal-2.2-manual.pdf>

# IntelliJ IDEA开发环境搭建

## 前置条件

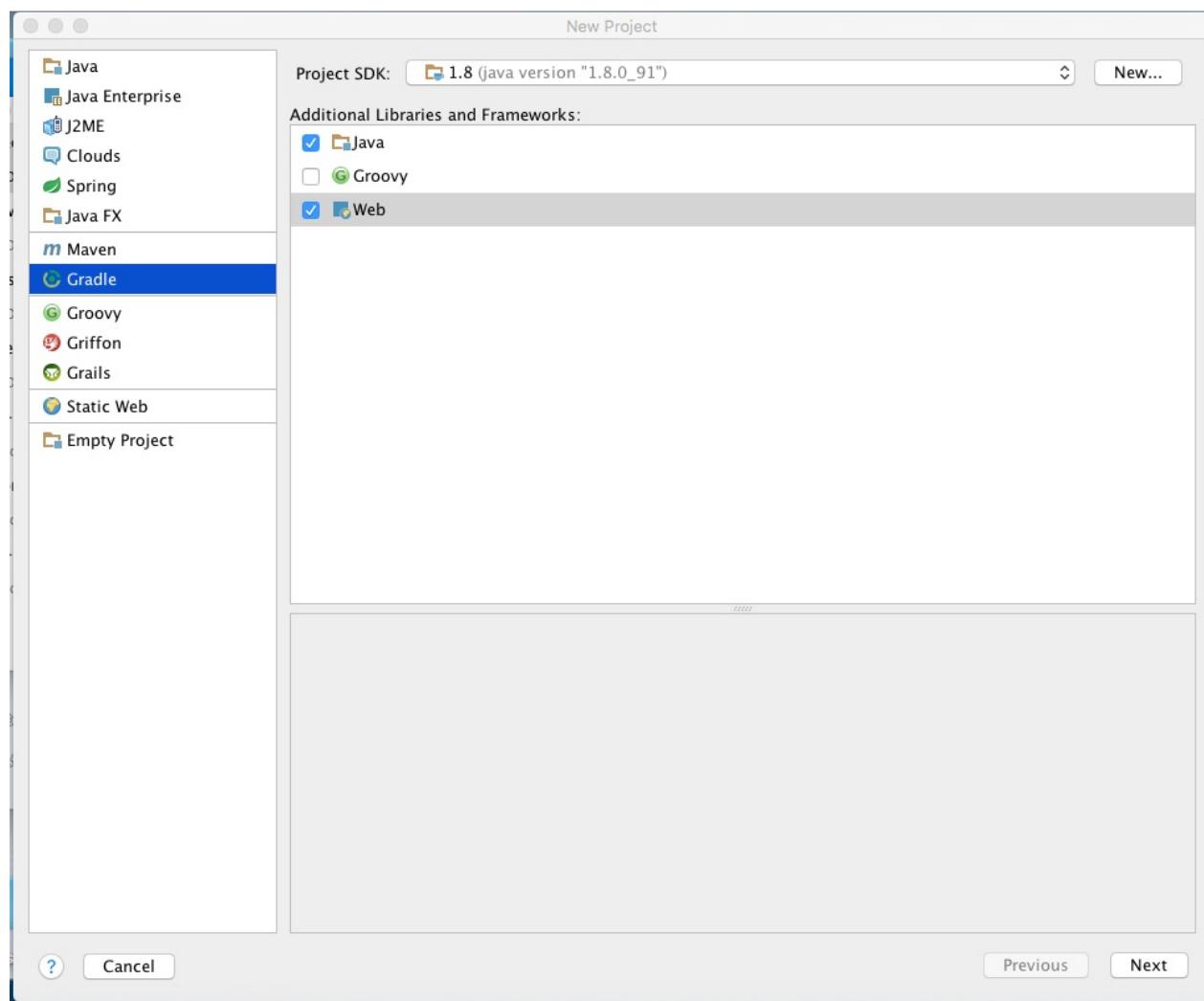
1. 操作系统已经安装IntelliJ IDEA 14以上，并已经配好java环境
2. 对gradle有一点了解
3. 对maven有一点了解
4. 对java Web开发有一定了解
5. mysql的基础知识

## 搭建步骤

### 一、新建项目

- 1) 点击“Create New Project”；
- 2) 在“Additional Libraries and Frameworks”中，选择Java和Web，点击next；

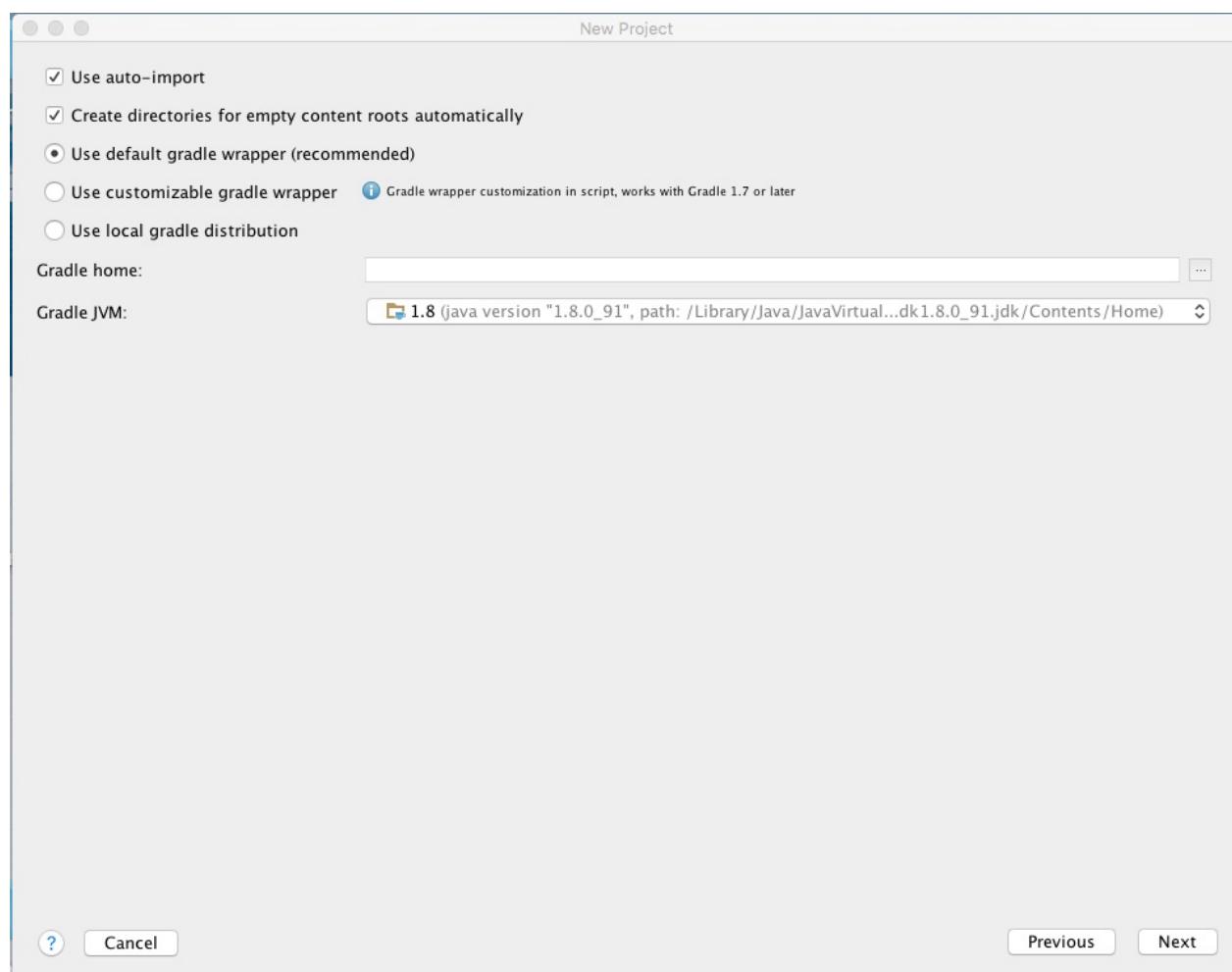
## 1 基于Gradle的项目构建



3) 填写GroupId和ArtifactId，这两项可以根据需要随便填，之后点击next；

4) 这里直接使用idea自带的gradle，并勾选前面两项，如下图：

## 1 基于Gradle的项目构建



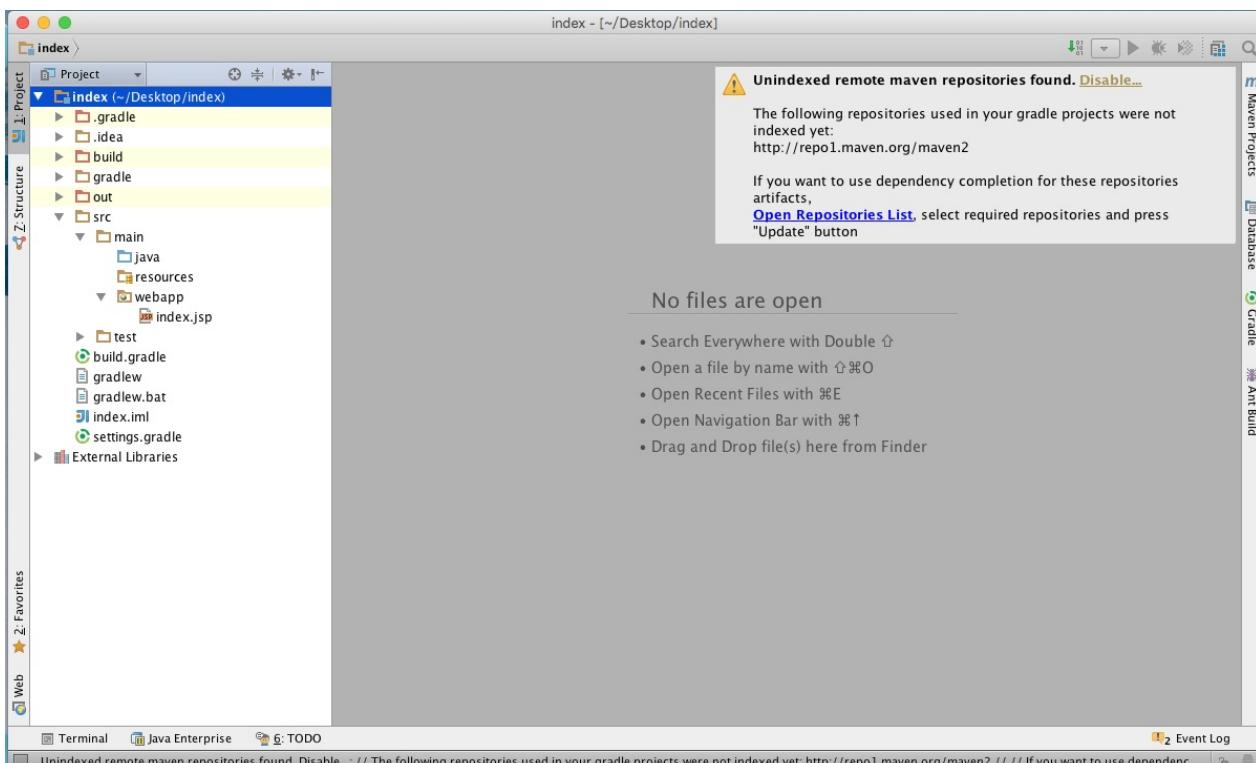
点击next；

5) 修改必要参数，点击Finish；

## 二、配置项目

1) 等待idea下方的进度条完成，之后项目的目录结构如下图：

# 1 基于Gradle的项目构建



其中有个警告，可以忽略，这是由于中国防火墙问题，导致maven repositorise的索引跟新失效。

2) 打开build.gradle文件，这是自动化构建的主文件，首先添加gretty插件，用于开发时运行web服务器容器，代码如下：

```
apply from: 'https://raw.github.com/akhikhil/gretty/master/pluginScripts/gretty.plugin'
```

3) 将mave的依赖库替换成阿里的：

```
mavenCentral() 替换为 maven { url  
'http://maven.aliyun.com/nexus/content/groups/public/' }
```

4) 添加jfinal相关依赖：

```
compile group: 'com.jfinal', name: 'jfinal', version: '2.2'  
testCompile group: 'junit', name: 'junit', version: '4.11'  
compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.38'  
compile group: 'c3p0', name: 'c3p0', version: '0.9.1.2'
```

最终文件类似如下：

```
group 'com.hithinksoft'
version '1.0-SNAPSHOT'

apply plugin: 'java'
apply plugin: 'war'

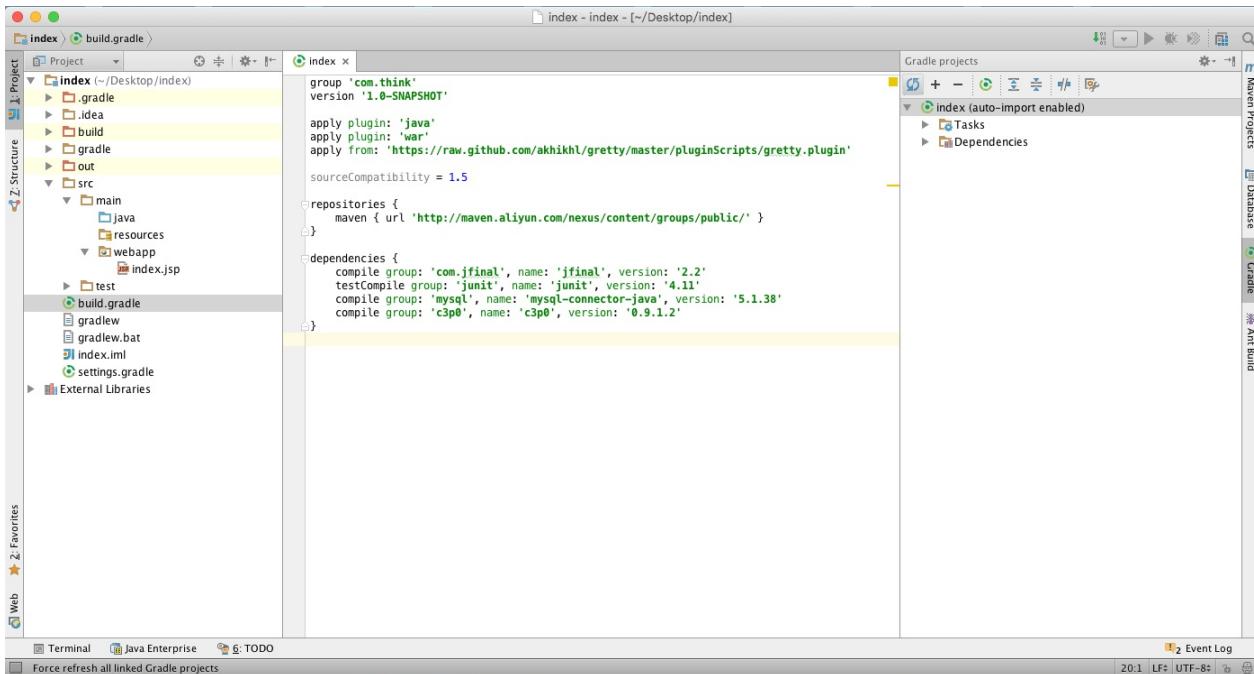
apply from: 'https://raw.github.com/akhikhl/gretty/master/plugin
Scripts/gretty.plugin'

sourceCompatibility = 1.5

repositories {
    maven { url 'http://maven.aliyun.com/nexus/content/groups/pu
blic/' }
}

dependencies {
    compile group: 'com.jfinal', name: 'jfinal', version: '2.2'
    testCompile group: 'junit', name: 'junit', version: '4.11'
    compile group: 'mysql', name: 'mysql-connector-java', versio
n: '5.1.38'
    compile group: 'c3p0', name: 'c3p0', version: '0.9.1.2'
}
```

5) 点击右侧的Gradle，弹出弹窗后点击刷新按钮（当然，你修改gradle文件时，可能他已经自动刷新，但是保险起见，手动刷新下），如下图：



### 三、运行一个jFinal例子

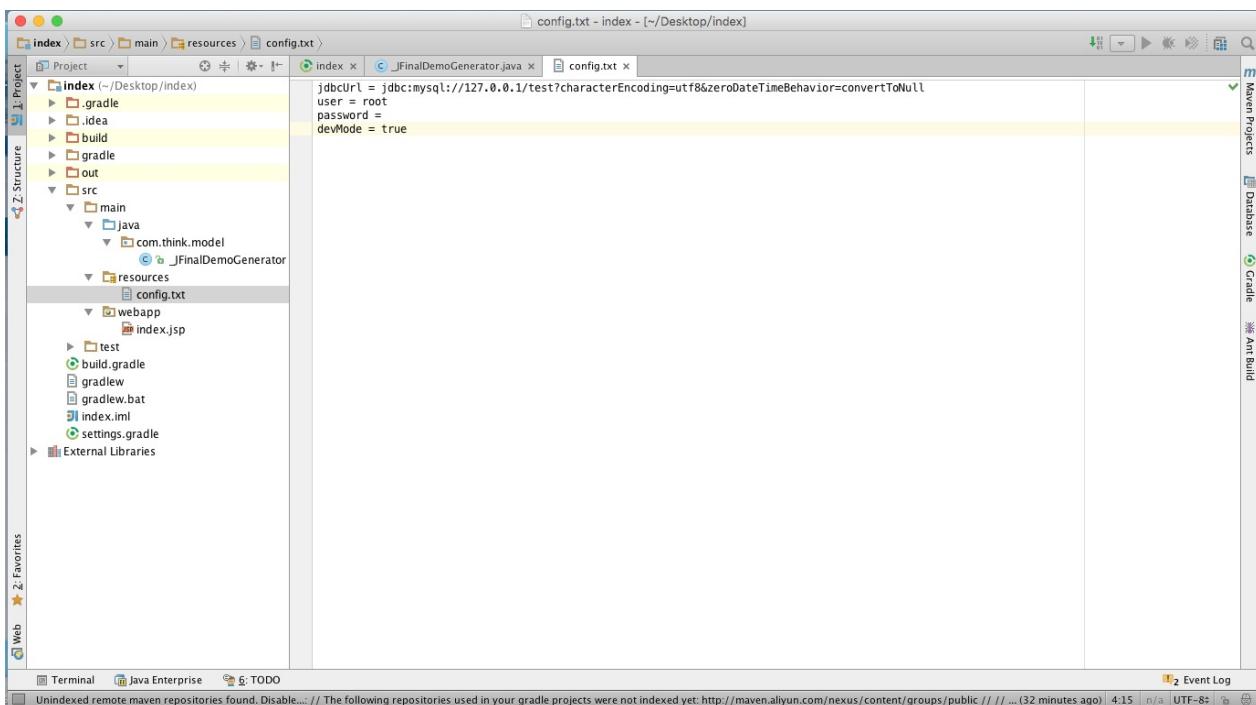
步骤二完成后，jfinal的开发环境基本完成，下面步骤，将带领大家真正跑起一个项目。

1) 准备模拟数据。方便起见，这里直接导入jfinal官方demo的数据：首先创建数据库，本例子中取名为“test”，下载官方demo，解压->源代码->blog.sql，在数据库客户端中执行代码即可。

2) 编写模型生成器。

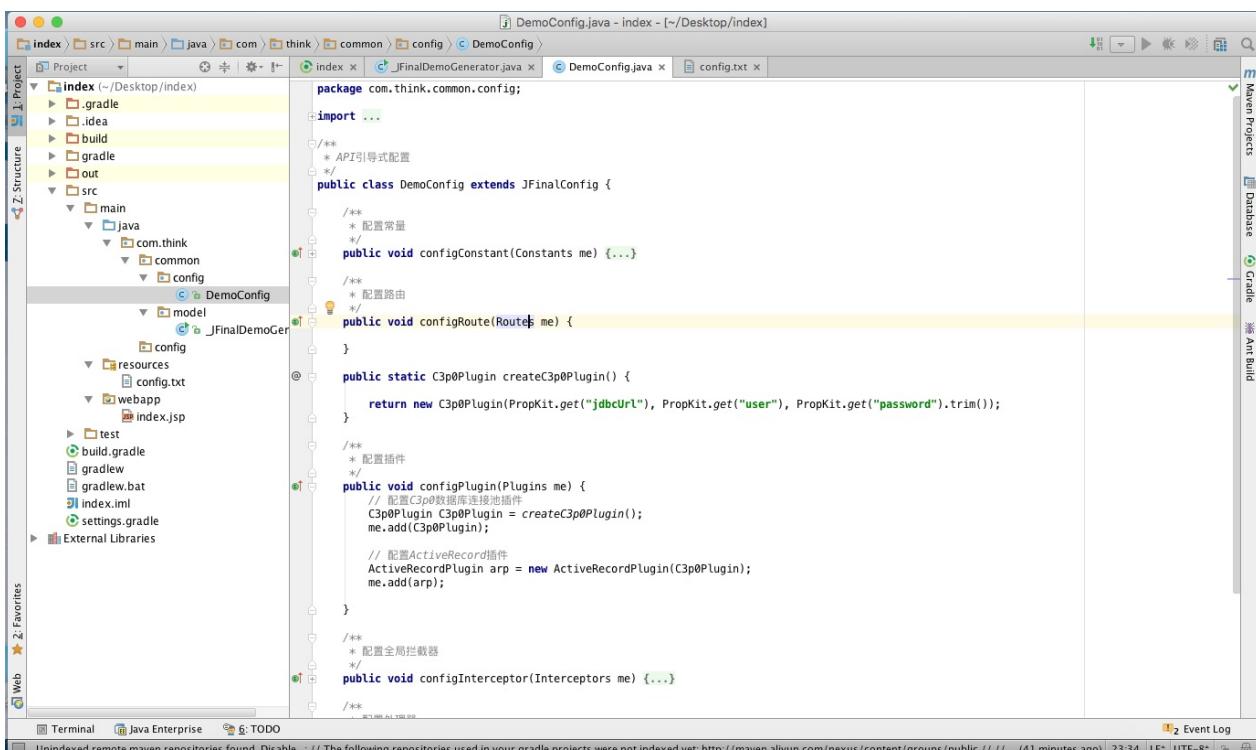
a. 找到src->main->resources这个静态资源文件，右击创建config.tex，项目的配置文件，并填好相关配置项，如下图：

# 1 基于Gradle的项目构建



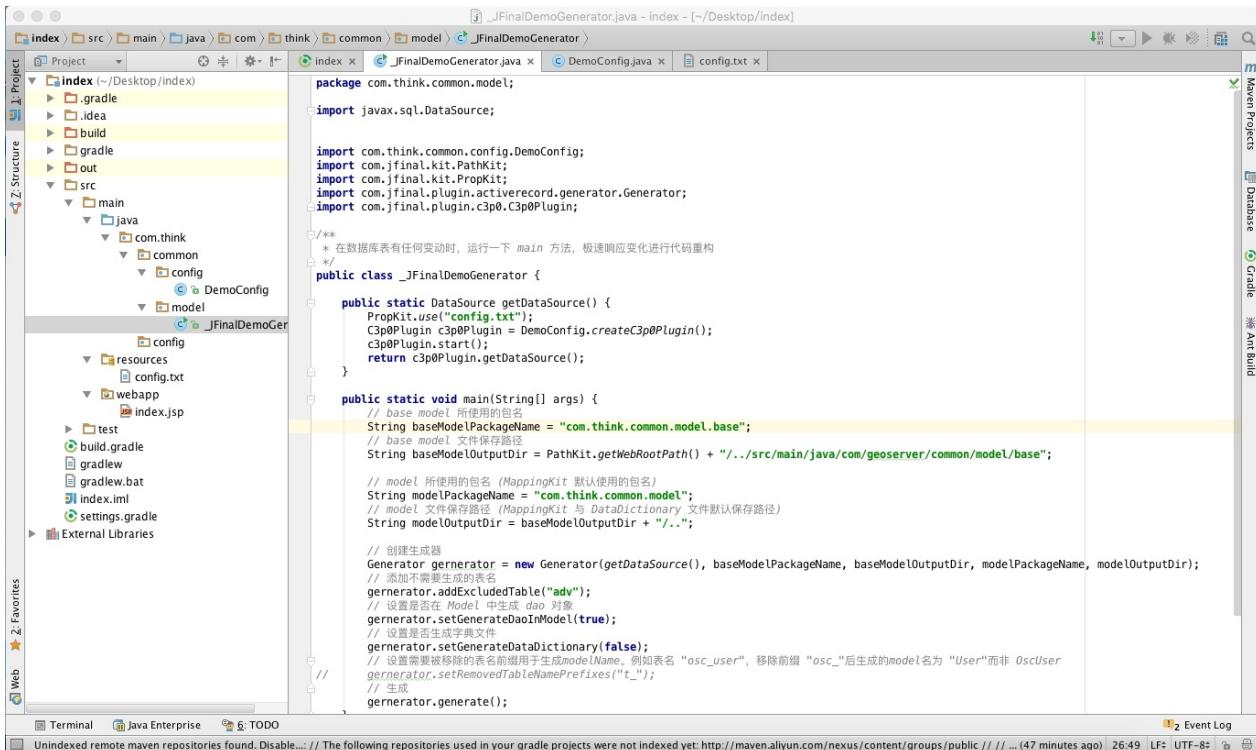
主要是配置jdbc链接，以及数据库的用户名和密码；

b. 找到src->main->java这个源代码目录，右击创建包，点击创建的包，右击创建DemoConfig文件，jFinal主配置文件，并编写数据库连接池代码，如下图：



c. 找到src->main->java这个源代码目录，右击创建包，点击创建的包，右击创建\_JFinalDemoGenerator文件，并编写相关模型生成器代码，如下图：

# 1 基于Gradle的项目构建



```
package com.think.common.model;

import javax.sql.DataSource;

import com.think.common.config.DemoConfig;
import com.jfinal.kit.PathKit;
import com.jfinal.kit.PropKit;
import com.jfinal.plugin.activerecord.generator.Generator;
import com.jfinal.plugin.c3p0.C3p0Plugin;

/*
 * 在数据库表有任何变动时，运行一下 main 方法，极速响应变化进行代码重构
 */
public class _JFinalDemoGenerator {

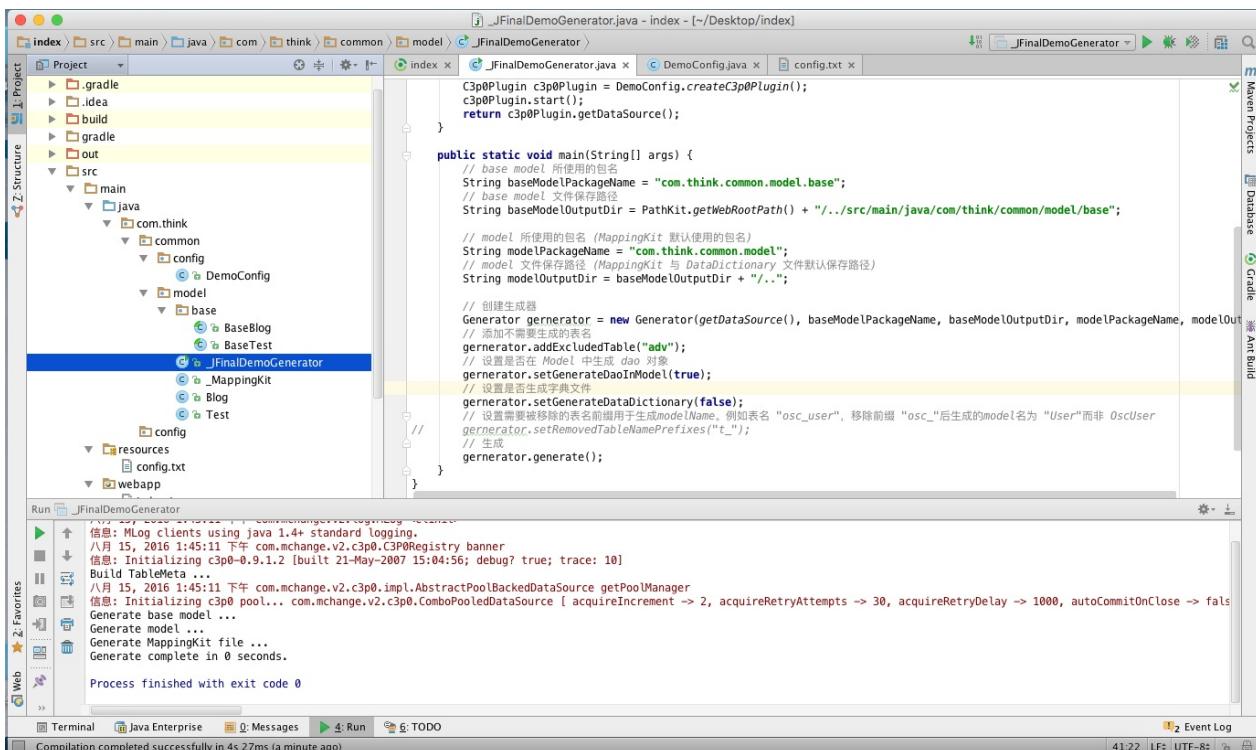
    public static DataSource getDataSource() {
        PropKit.use("config.txt");
        C3p0Plugin c3p0Plugin = DemoConfig.createC3p0Plugin();
        c3p0Plugin.start();
        return c3p0Plugin.getDataSource();
    }

    public static void main(String[] args) {
        // base model 所使用的包名
        String baseModelPackageName = "com.think.common.model.base";
        // base model 文件保存路径
        String baseModelOutputDir = PathKit.getWebRootPath() + "/../src/main/java/com/geoserver/common/model/base";

        // model 所使用的包名 (MappingKit 默认使用的包名)
        String modelPackageName = "com.think.common.model";
        // model 文件保存路径 (MappingKit 与 DataDictionary 文件默认保存路径)
        String modelOutputDir = baseModelOutputDir + "/..";

        // 创建生成器
        Generator generator = new Generator(getDataSource(), baseModelPackageName, baseModelOutputDir, modelPackageName, modelOutputDir);
        // 添加不需要生成的表名
        generator.addExcludedTable("adv");
        // 设置是否在 Model 中生成 dao 对象
        generator.setGenerateDaoInModel(true);
        // 设置是否生成字典文件
        generator.setGenerateDataDictionary(false);
        // 设置需要被移除的表名前缀用于生成modelName。例如表名 "osc_user"，移除前缀 "osc_" 后生成的model名为 "User" 而非 OscUser
        generator.setRemovedTableNamePrefixes("t_");
        // 生成
        generator.generate();
    }
}
```

d. 右击 `_JFinalDemoGenerator->run`，稍等片刻，模型生成器就会为我们生成各种模型类，如下图：



```
C3p0Plugin c3p0Plugin = DemoConfig.createC3p0Plugin();
c3p0Plugin.start();
return c3p0Plugin.getDataSource();

public static void main(String[] args) {
    // base model 所使用的包名
    String baseModelPackageName = "com.think.common.model.base";
    // base model 文件保存路径
    String baseModelOutputDir = PathKit.getWebRootPath() + "/../src/main/java/com/think/common/model/base";

    // model 所使用的包名 (MappingKit 默认使用的包名)
    String modelPackageName = "com.think.common.model";
    // model 文件保存路径 (MappingKit 与 DataDictionary 文件默认保存路径)
    String modelOutputDir = baseModelOutputDir + "/..";

    // 创建生成器
    Generator generator = new Generator(getDataSource(), baseModelPackageName, baseModelOutputDir, modelPackageName, modelOutputDir);
    // 添加不需要生成的表名
    generator.addExcludedTable("adv");
    // 设置是否在 Model 中生成 dao 对象
    generator.setGenerateDaoInModel(true);
    // 设置是否生成字典文件
    generator.setGenerateDataDictionary(false);
    // 设置需要被移除的表名前缀用于生成modelName。例如表名 "osc_user"，移除前缀 "osc_" 后生成的model名为 "User" 而非 OscUser
    generator.setRemovedTableNamePrefixes("t_");
    // 生成
    generator.generate();
}

MLog clients using java 1.4+ standard logging.
八月 15, 2016 1:45:11 下午 com.mchange.v2.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
Build TableMeta ...
八月 15, 2016 1:45:11 下午 com.mchange.v2.C3P0.impl.AbstractPoolBackedDataSource getPoolManager
信息: Initializing c3p0 pool... com.mchange.v2.c3p0.ComboPooledDataSource [ acquireIncrement -> 2, acquireRetryAttempts -> 30, acquireRetryDelay -> 1000, autoCommitOnClose -> false, generateBaseTables -> true, generateStat -> false, identityToken -> null, initialPoolSize -> 5, maxAdministrativeTaskTime -> 0, maxConnectionAge -> 1800, maxIdleTime -> 900, maxPoolSize -> 20, maxStatements -> 0, maxWaitTime -> 10000, minPoolSize -> 5, numHelperThreads -> 2, statementCacheNumDeferredLoadingThreshold -> 50, statementCacheNumFullFetchedStatements -> 50, statementCacheNumSoftlyCachableStatements -> 50, statementCacheNumTotalStatements -> 50, statementCacheSoftlyCachable -> true, validateAtCheckin -> true ]
Generate base model ...
Generate MappingKit file ...
Generate complete in 0 seconds.

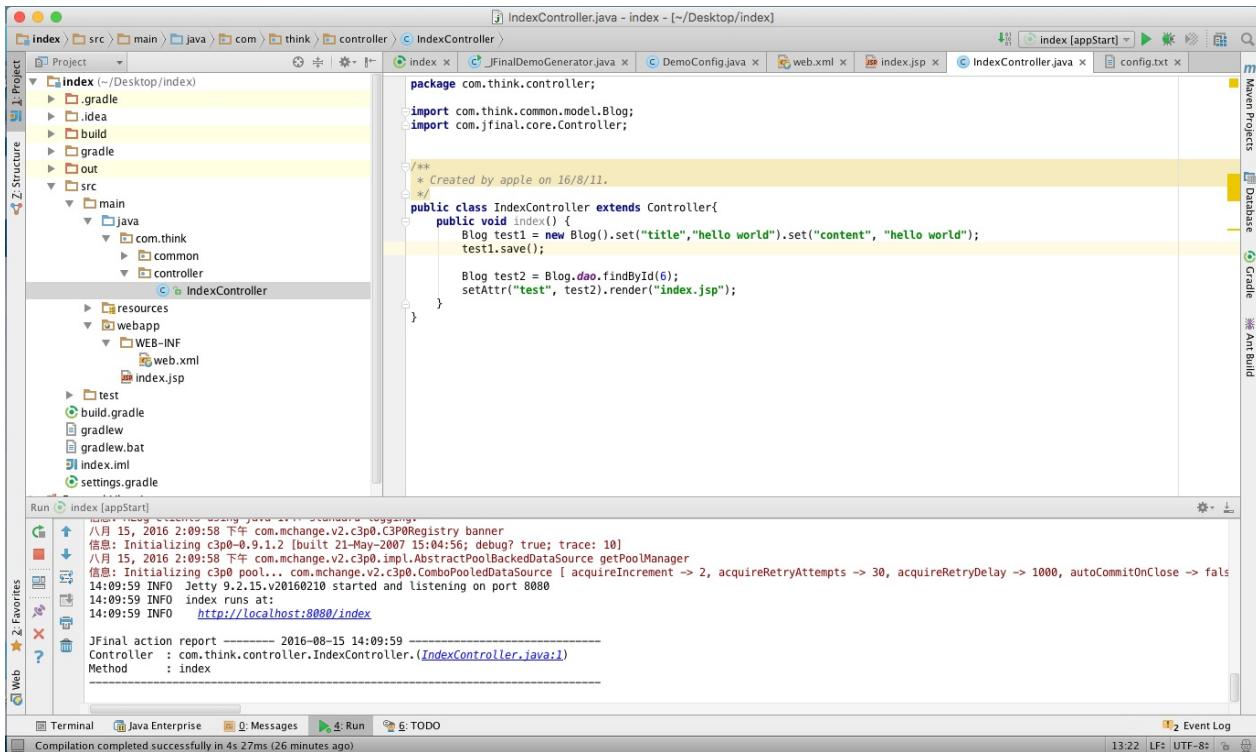
Process finished with exit code 0
```

OK！到此，模型已经准备完毕，接下来就是Controller和View了。

## 3) Controller和View

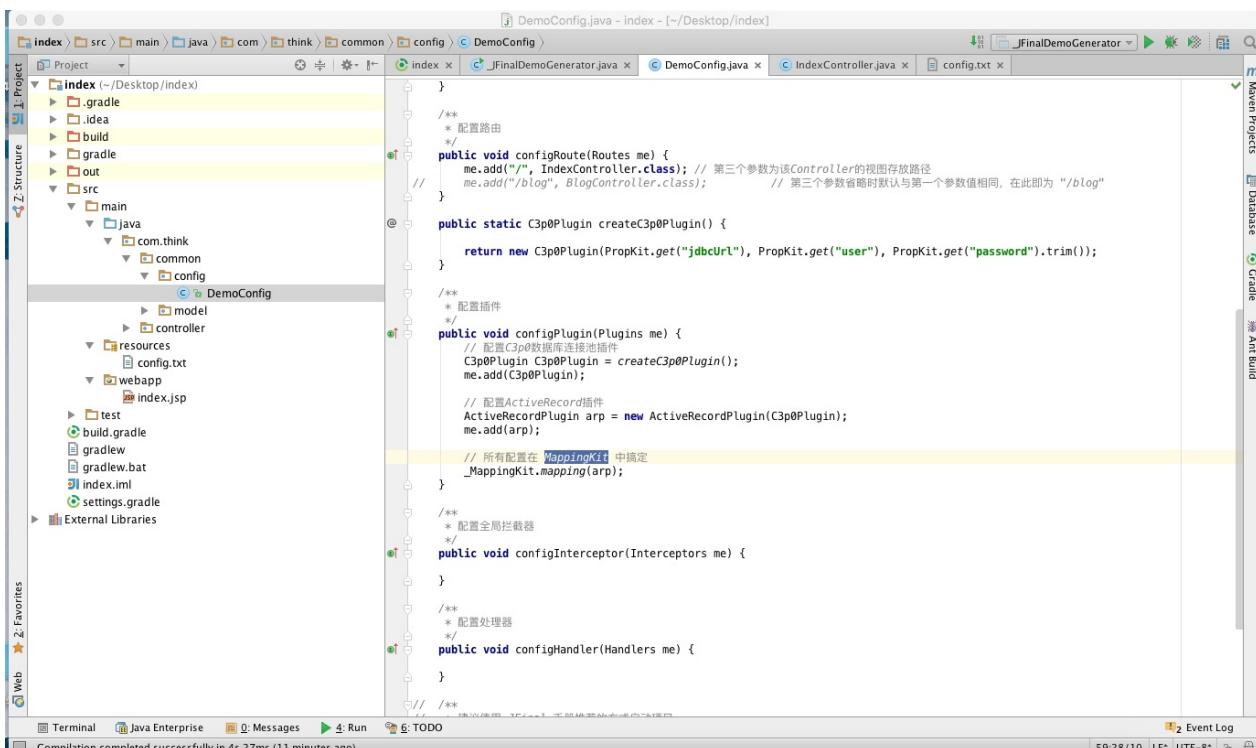
# 1 基于Gradle的项目构建

a. 新建一个IndexController并，编写相关数据操作方法，这里就存取了一下，如下图：



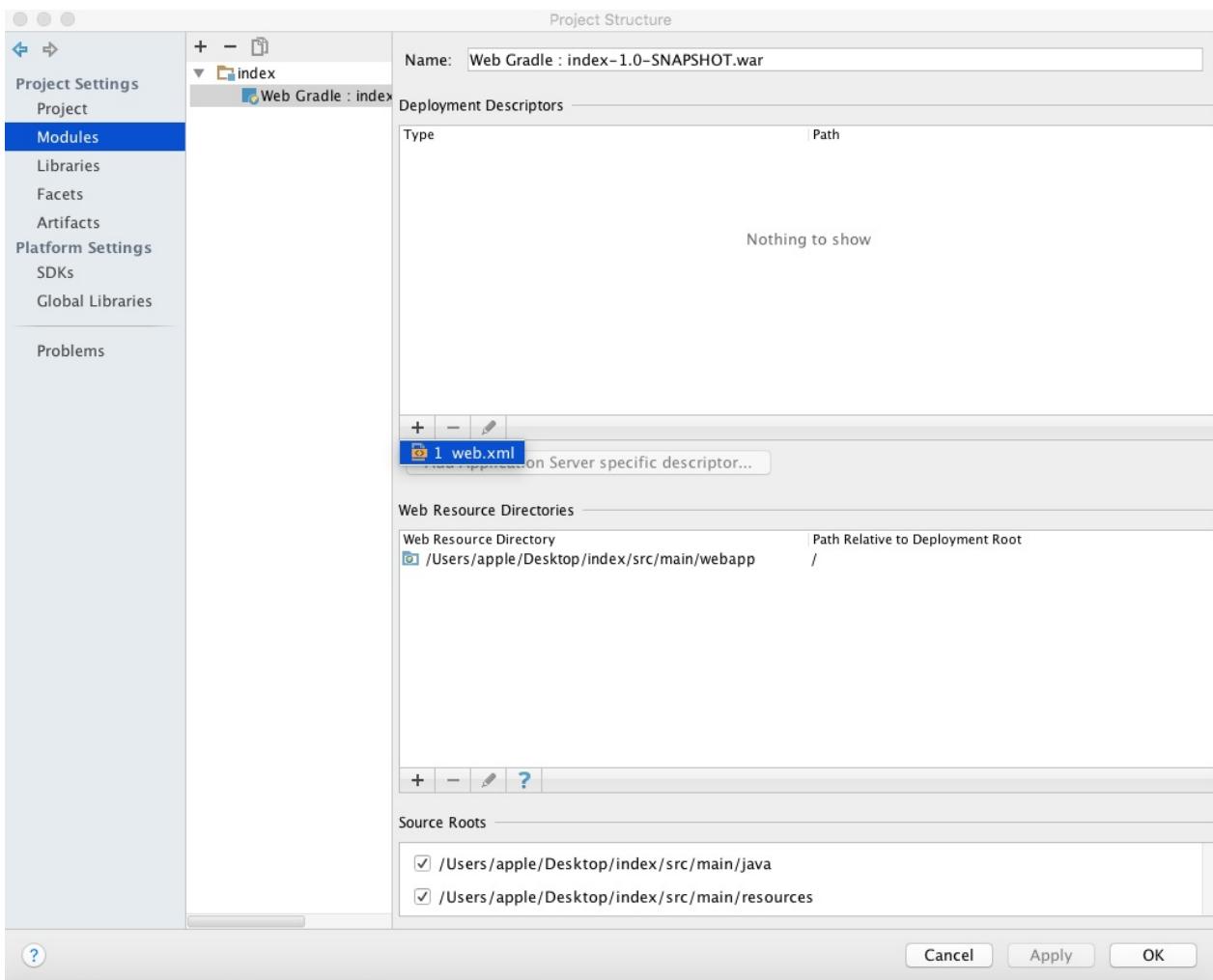
b. 在DemoConfig中配置路由，以及一些常量配置、插件配置等，如下图：

图看不清没关系，demo在压缩包中，自己看情况查看

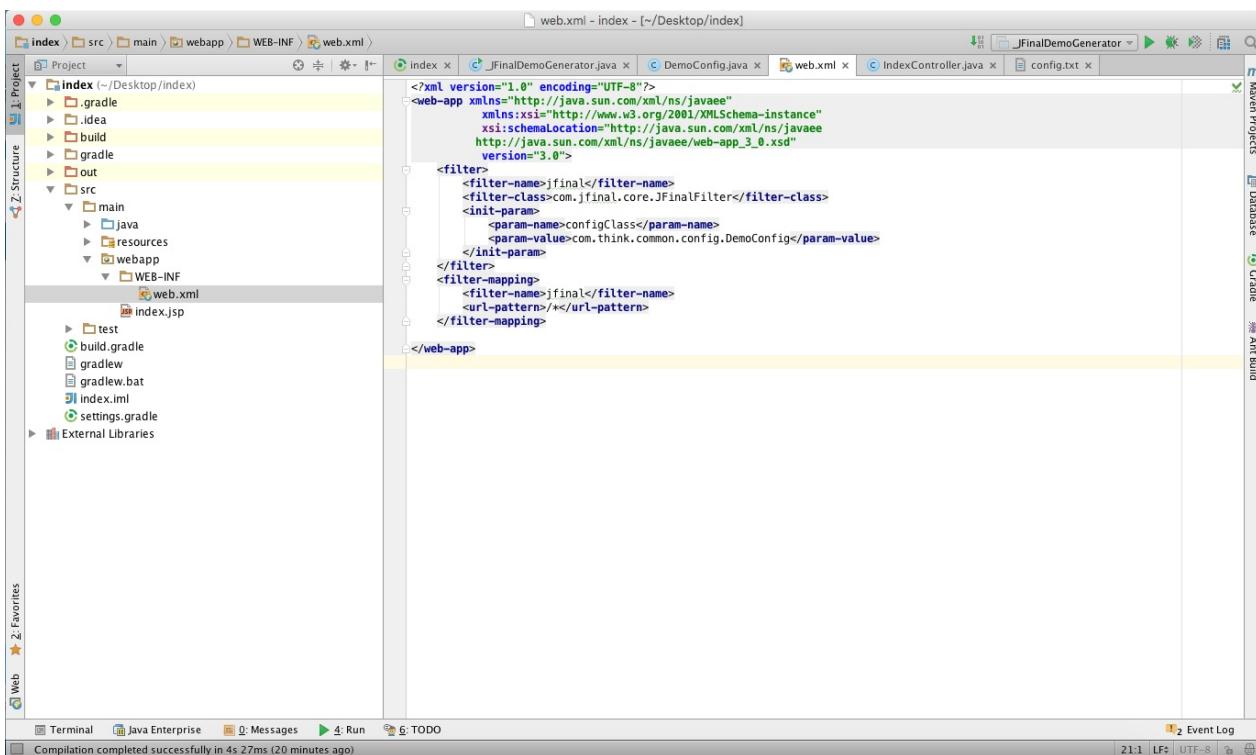


c. 添加web.xml。File->Project Structure,选择第二个Modules，选择Web Gradle:...,选择右边第一个框的加号，如下图：

# 1 基于Gradle的项目构建

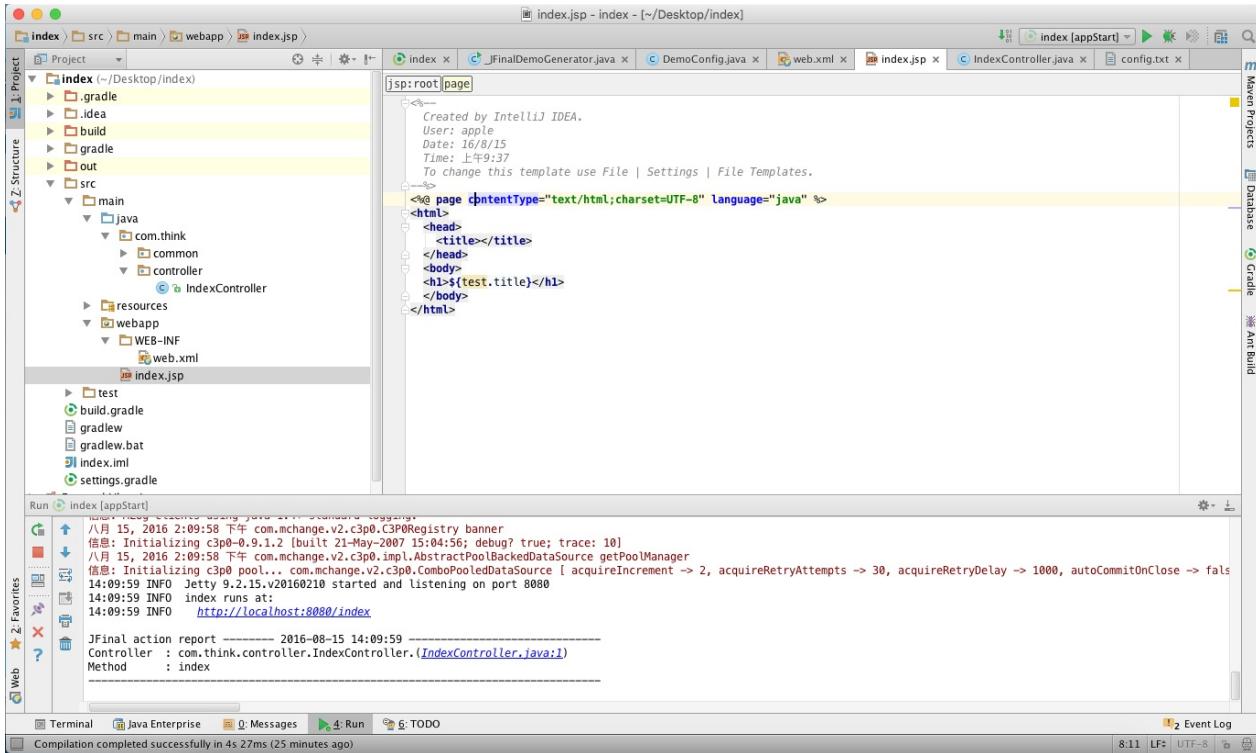


d. 选择web.XML,修改路径到src/main/webapp下面，点击完成，修改jFinal配置，如下图：



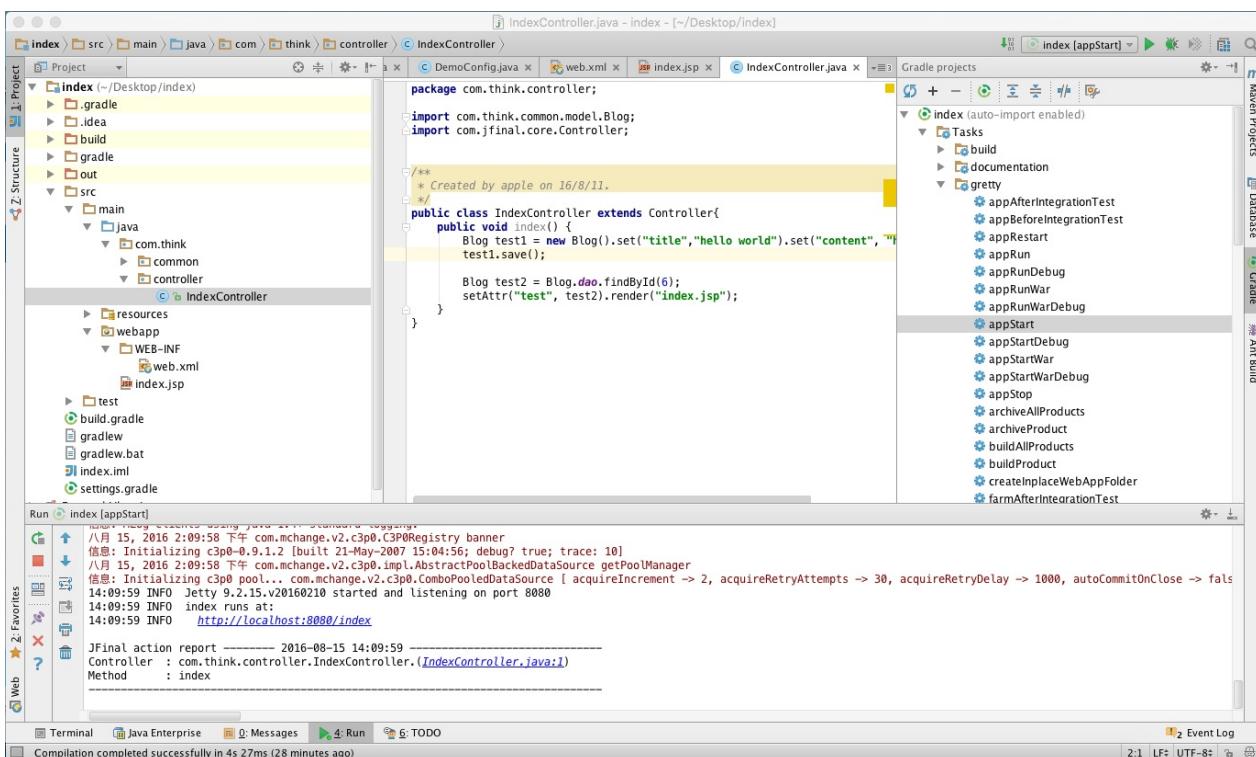
# 1 基于Gradle的项目构建

e. 修改index.jsp，如下图：



## 4) 运行程序

选择右边的Gradle，选择Tasks，选择gretty，双击appStart，运行服务器，如下图：



浏览器中打开<http://localhost:8080/index>，即可看到Hello World；

到此整个**demo**都运行起来了，真正开发过程中，各功能自己稍微摸索下就可以了，这里不再累述。

## 概述

开源实时日志分析ELK平台(ElasticSearch, Logstash, Kibana组成)，能很方便的帮我们收集日志，进行集中化的管理，并且能很方便的进行日志的统计和检索，下面基于ELK的最新版本5.2.2进行一次整合测试。

ElasticSearch是一个高可扩展的开源的全文搜索分析引擎。它允许你快速的存储、搜索和分析大量数据。ElasticSearch通常作为后端程序，为需要复杂查询的应用提供服务。

Elasticsearch是一个基于Lucene的开源分布式搜索引擎，具有分布式多用户能力。Elasticsearch是用java开发，提供Restful接口，能够达到实时搜索、高性能计算；同时Elasticsearch的横向扩展能力非常强，不需要重启服务，基本上达到了零配置。

写到一半，发现一个非常好的Logstash的文章集合，大家直接参考吧。我之后会实验为主，至于理论详解，我感觉以下三篇文章，写的比我好多好多了。

[Logstash 最佳实践](#)

[Elasticsearch 权威指南](#)

[ELKstack 中文指南](#)

## 环境要求：

- JDK1.8
- ElasticSearch 5.2.2

## 下载地址

<https://www.elastic.co/downloads/elasticsearch>

## 安装过程

- 1.解压ElasticSearch并进入目录：

```
chu888chu888@hadoopmaster:~$ tar xvfz elasticsearch-5.2.2.tar.gz
```

- 2. 启动ElasticSearch

```
chu888chu888@hadoopmaster:~/elasticsearch-5.2.2$ cd bin  
chu888chu888@hadoopmaster:~/elasticsearch-5.2.2/bin$ ls  
chu888chu888@hadoopmaster:~/elasticsearch-5.2.2/bin$ ./elasticse  
arch
```

日志中启动了两个端口分别是：9300和9200,9300用于跟其他的节点的传输，9200用于接受HTTP请求，**ctrl+c**可以结束进程

如果想要后台运行，可以用以下方法

```
./bin/elasticsearch -d
```

- 

- 1. 测试ElasticSearch

```
chu888chu888@hadoopmaster:~$ curl 127.0.0.1:9200  
{  
  "name" : "ivu1waD",  
  "cluster_name" : "elasticsearch",  
  "cluster_uuid" : "J0GwcsnES6Gr12RJapiHyA",  
  "version" : {  
    "number" : "5.2.1",  
    "build_hash" : "db0d481",  
    "build_date" : "2017-02-09T22:05:32.386Z",  
    "build_snapshot" : false,  
    "lucene_version" : "6.4.1"  
  },  
  "tagline" : "You Know, for Search"  
}  
chu888chu888@hadoopmaster:~$
```

也可以使用curl尝试远程关闭

```
chu888chu888@hadoopmaster:~$ curl -XPOST 'http://localhost:9200/_shutdown'
```

- 

### 1. Elasticsearch配置文件

ElasticSearch的配置文件一般放置在安装目录下的config目录中。config目录中有两个文件,分别是elasticsearch.yml 和logging.yml。前者用来配置ElasticSearch不同模块的属性,比如网络地址,路径等,后者则用来配置自身的日志记录选项。

配置文件是YAML格式的,下面简要地介绍一些配置参数。

网络地址,指定网络相关模块的绑定和发布地址

```
network:  
  host:127.0.0.1
```

路径指定数据和日志文件的路径

```
path:  
  logs:/var/log/elasticsearch  
  data:/var/data/elasticsearch
```

集群名,指定生产集群的名字,集群将根据这个名字来自动发现和加入节点

```
cluster:  
  name:<NAME OF YOUR CLUSTER>
```

节点名,指定每个节点的默认名称

```
node:  
  name:<NAME OF YOUR NODE>
```

因为elasticsearch安装在虚拟机里面,我希望我的主机也可以访问,需要 config/elasticsearch.yml进行配置:

```
network.host: 192.168.1.159
```

重新启动后会出现错误

```
/bin$ ./elasticsearch
```

解决办法：切换到root用户，修改配置limits.conf

```
chu888chu888@hadoopmaster:/etc/security$ sudo nano limits.conf

* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096
```

修改配置sysctl.conf

```
chu888chu888@hadoopmaster:/etc/security$ sudo nano /etc/sysctl.conf

vm.max_map_count=655360
```

重新再启动后，成功

```
chu888chu888@hadoopmaster:~/elasticsearch-5.2.2/bin$ ./elasticsearch
[2017-03-13T15:19:11,746][INFO ][o.e.n.Node] [ ] initializing ...
[2017-03-13T15:19:11,819][INFO ][o.e.e.NodeEnvironment] [Vz87mNE] using [1] data paths, mounts [[/ (/dev/mapper/hadoopmaster--vg-root)]], net usable_space [40.9gb], net total_space [50.8gb], spins? [possibly], types [ext4]
[2017-03-13T15:19:11,819][INFO ][o.e.e.NodeEnvironment] [Vz87mNE] heap size [1.9gb], compressed ordinary object pointers [true]
[2017-03-13T15:19:11,837][INFO ][o.e.n.Node] [ ] node
```

```

name [Vz87mNE] derived from node ID [Vz87mNE2QAmvwodx2uB94Q]; set [node.name] to override
[2017-03-13T15:19:11,839][INFO ][o.e.n.Node                ] vers
ion[5.2.2], pid[5208], build[f9d9b74/2017-02-24T17:26:45.835Z],
OS[Linux/4.2.0-27-generic/amd64], JVM[Oracle Corporation/Java Ho
tSpot(TM) 64-Bit Server VM/1.8.0_121/25.121-b13]
[2017-03-13T15:19:12,810][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [aggs-matrix-stats]
[2017-03-13T15:19:12,811][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [ingest-common]
[2017-03-13T15:19:12,811][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [lang-expression]
[2017-03-13T15:19:12,811][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [lang-groovy]
[2017-03-13T15:19:12,811][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [lang-mustache]
[2017-03-13T15:19:12,812][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [lang-painless]
[2017-03-13T15:19:12,812][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [percolator]
[2017-03-13T15:19:12,812][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [reindex]
[2017-03-13T15:19:12,812][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [transport-netty3]
[2017-03-13T15:19:12,818][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded module [transport-netty4]
[2017-03-13T15:19:12,819][INFO ][o.e.p.PluginsService    ] [Vz8
7mNE] loaded plugin [analysis-icu]
[2017-03-13T15:19:15,112][INFO ][o.e.n.Node              ] init
ialized
[2017-03-13T15:19:15,115][INFO ][o.e.n.Node              ] [Vz8
7mNE] starting ...
[2017-03-13T15:19:15,319][INFO ][o.e.t.TransportService  ] [Vz8
7mNE] publish_address {192.168.1.159:9300}, bound_addresses {192
.168.1.159:9300}
[2017-03-13T15:19:15,322][INFO ][o.e.b.BootstrapChecks   ] [Vz8
7mNE] bound or publishing to a non-loopback or non-link-local ad
dress, enforcing bootstrap checks
[2017-03-13T15:19:18,386][INFO ][o.e.c.s.ClusterService ] [Vz8
7mNE] new_master {Vz87mNE}{Vz87mNE2QAmvwodx2uB94Q}{q1ibubCJQD0jd

```

```

yySwkrH0w}{192.168.1.159}{192.168.1.159:9300}, reason: zen-disco
-elected-as-master ([0] nodes joined)
[2017-03-13T15:19:18,424][INFO ][o.e.h.HttpServer           ] [Vz8
7mNE] publish_address {192.168.1.159:9200}, bound_addresses {192
.168.1.159:9200}
[2017-03-13T15:19:18,424][INFO ][o.e.n.Node             ] [Vz8
7mNE] started
[2017-03-13T15:19:18,995][INFO ][o.e.g.GatewayService ] [Vz8
7mNE] recovered [2] indices into cluster_state
[2017-03-13T15:19:19,255][INFO ][o.e.c.r.a.AllocationService] [V
z87mNE] Cluster health status changed from [RED] to [YELLOW] (re
ason: [shards started [[.kibana][0]] ...]).
```



```
{
  "name": "ivu1waD",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "J0GWcsnES6GrI2RJapiHyA",
  "version": {
    "number": "5.2.1",
    "build_hash": "db0d481",
    "build_date": "2017-02-09T22:05:32.386Z",
    "build_snapshot": false,
    "lucene_version": "6.4.1"
  },
  "tagline": "You Know, for Search"
}
```

## ElasticSearch插件

ElasticSearch有各种插件，可以简化诸如管理索引、集群等任务。其中一些常用的插件有kopf Marvel Sense Shield等等。

## ElasticSearch安装错误FAQ

### Elasticsearch5.0 安装问题集锦

elasticsearch 5.0 安装过程中遇到了一些问题，通过查找资料几乎都解决掉了，这里简单记录一下，供以后查阅参考，也希望可以帮助遇到同样问题的你。

- 1. 问题一：警告提示

```
[2016-11-06T16:27:21,712][WARN ][o.e.b.JNANatives ] unable to install syscall filter:
```

```
java.lang.UnsupportedOperationException: seccomp unavailable: requires kernel 3.5+ with CONFIG_SECCOMP and CONFIG_SECCOMP_FILTER compiled in
at org.elasticsearch.bootstrap.Seccomp.linuxImpl(Seccomp.java:349) ~[elasticsearch-5.0.0.jar:5.0.0]
at org.elasticsearch.bootstrap.Seccomp.init(Seccomp.java:630) ~[elasticsearch-5.0.0.jar:5.0.0]
```

报了一大串错误，其实只是一个警告。

解决：使用比较新的linux版本，就不会出现此类问题了。

- 2. 问题二：ERROR: bootstrap checks failed

```
max file descriptors [4096] for elasticsearch process likely too low, increase to at least [65536]
max number of threads [1024] for user [lishang] likely too low, increase to at least [2048]
```

解决：切换到root用户，编辑limits.conf 添加类似如下内容

```
vi /etc/security/limits.conf
```

添加如下内容：

```
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096
```

- 3. 问题三：max number of threads [1024] for user [lish] likely too low, increase

to at least [2048]

解决：切换到root用户，进入limits.d目录下修改配置文件。

```
vi /etc/security/limits.d/90-nproc.conf
```

修改如下内容：

```
* soft nproc 2048
```

- 4. 问题四：max virtual memory areas vm.max\_map\_count [65530] likely too low, increase to at least [262144]

解决：切换到root用户修改配置sysctl.conf

```
vi /etc/sysctl.conf
```

添加下面配置：

```
vm.max_map_count=655360
```

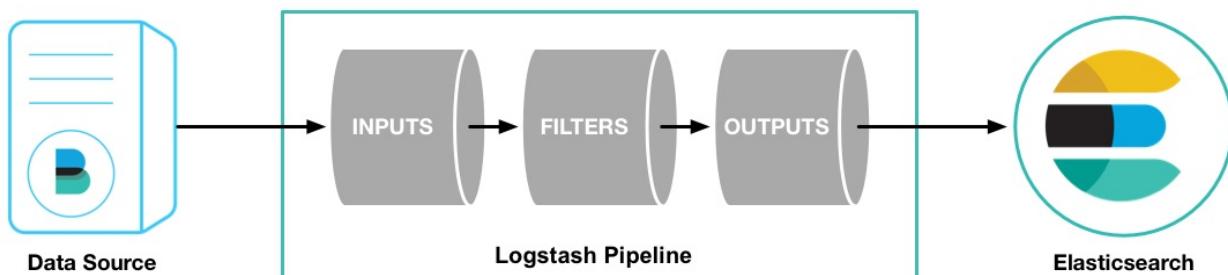
并执行命令：

```
sysctl -p
```

然后，重新启动elasticsearch，即可启动成功。

## Logstash

Logstash是一个完全开源的工具，可以对你的日志进行收集、过滤，并将其存储供以后使用，参考官网的介绍图：



写到一半，发现一个非常好的Logstash的文章集合，大家直接参考吧。

[Logstash 最佳实践](#)

[Elasticsearch 权威指南](#)

[ELKstack 中文指南](#)

## 安装

- 1.解压进入目录

```
chu888chu888@hadoopmaster:~$ unzip logstash-5.2.2.zip
```

- 2.添加配置文件

```
chu888chu888@hadoopmaster:~/logstash-5.2.2/config$ nano logstash.conf
```

- 3.添加如下内容

我们通过一个股票的例子（从yahoo下载下来的数据），来讲解通过logstash导入数据到elasticsearch中,<http://finance.yahoo.com/q/hp?s=GOOG>

这个数据集中最重要的字段包括 date(日期) open price(开盘价格) close price(收盘价格) high price(最高价) volume(成交量) adjusted price(调整价格)

Date,Open,High,Low,Close,Volume,Adj Close  
2017-03-10,843.280029,844.909973,839.50,843.25,1701100,843.25  
2017-03-09,836.00,842.00,834.210022,838.679993,1259900,838.67999  
3  
2017-03-08,833.51001,838.150024,831.789978,835.369995,988700,835  
.369995  
2017-03-07,827.400024,833.409973,826.52002,831.909973,1016600,83  
1.909973  
2017-03-06,826.950012,828.880005,822.400024,827.780029,1105800,8  
27.780029  
2017-03-03,830.559998,831.359985,825.750977,829.080017,888900,82  
9.080017  
2017-03-02,833.849976,834.51001,829.640015,830.630005,937700,830  
.630005  
2017-03-01,828.849976,836.255005,827.26001,835.23999,1491400,835  
.23999  
2017-02-28,825.609985,828.539978,820.200012,823.210022,2252300,8  
23.210022  
2017-02-27,824.549988,830.50,824.00,829.280029,1099500,829.28002  
9  
2017-02-24,827.72998,829.00,824.200012,828.640015,1386600,828.64  
0015  
2017-02-23,830.119995,832.460022,822.880005,831.330017,1470100,8  
31.330017  
2017-02-22,828.659973,833.25,828.640015,830.76001,982900,830.760  
01  
2017-02-21,828.659973,833.450012,828.349976,831.659973,1247700,8  
31.659973  
2017-02-17,823.02002,828.070007,821.655029,828.070007,1597800,82  
8.070007  
2017-02-16,819.929993,824.400024,818.97998,824.159973,1281700,82  
4.159973

每行代表了某一天的股票价格，字段之间以逗号分割。

```
input
{
  file
  {
    path=>"/home/chu888chu888/table.csv"
    start_position=>"beginning"
  }
}

filter
{
  csv
  {
    columns=>["date_of_record", "open", "high", "low", "close", "volume", "adj_close"]
    separator=>", "
  }
  date
  {
    match=>["date_of_record", "yyyy-MM-dd"]
  }
  mutate
  {
    convert=>["open", "float"]
    convert=>["high", "float"]
    convert=>["low", "float"]
    convert=>["close", "float"]
    convert=>["volume", "integer"]
    convert=>["adj_close", "float"]
  }
}

output
{
  elasticsearch
  {
    hosts => [ "192.168.1.159:9200" ]
  }
}
```

- 4. 启动服务

```
chu888chu888@hadoopmaster:~/logstash-5.2.2/bin$ ./logstash -f /home/chu888chu888/logstash.conf
```

- 5. 启动成功后的日志

```
chu888chu888@hadoopmaster:~/logstash-5.2.2/bin$ ./logstash -f /home/chu888chu888/logstash.conf
Sending Logstash's logs to /home/chu888chu888/logstash-5.2.2/logs which is now configured via log4j2.properties
[2017-03-13T15:25:37,977][INFO ][logstash.outputs.elasticsearch]
  Elasticsearch pool URLs updated {:changes=>{:removed=>[], :added=>[http://192.168.1.159:9200/]}}
[2017-03-13T15:25:37,979][INFO ][logstash.outputs.elasticsearch]
  Running health check to see if an Elasticsearch connection is working {:healthcheck_url=>http://192.168.1.159:9200/, :path=>"/"}
}
[2017-03-13T15:25:38,173][WARN ][logstash.outputs.elasticsearch]
  Restored connection to ES instance {:url=>#<URI::HTTP:0x63e64c30 URL:http://192.168.1.159:9200/>}
[2017-03-13T15:25:38,178][INFO ][logstash.outputs.elasticsearch]
  Using mapping template from {:path=>nil}
[2017-03-13T15:25:38,288][INFO ][logstash.outputs.elasticsearch]
  Attempting to install template {:manage_template=>{"template"=>"logstash-*", "version"=>50001, "settings"=>{"index.refresh_interval"=>"5s"}, "mappings"=>{"_default_"=>{"_all"=>{"enabled"=>true, "norms"=>false}, "dynamic_templates"=>[{"message_field"=>{"path_match"=>"message", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false}}, {"string_fields"=>{"match"=>"*", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false, "fields"=>{"keyword"=>{"type"=>"keyword"}}}]}, "properties"=>{@timestamp=>{"type"=>"date", "include_in_all"=>false}, "@version"=>{"type"=>"keyword", "include_in_all"=>false}, "geoip"=>{"dynamic"=>true, "properties"=>{"ip"=>{"type"=>"ip"}, "location"=>{"type"=>"geo_point"}, "latitude"=>{"type"=>"half_float"}, "longitude"=>{"type"=>"half_float"}}}]}
[2017-03-13T15:25:38,315][INFO ][logstash.outputs.elasticsearch]
  New Elasticsearch output {:class=>"LogStash::Outputs::ElasticSe
```

```
arch", :hosts=>[#<URI::Generic:0x2fc46ddb URL://192.168.1.159:9200>]}
[2017-03-13T15:25:38,316][INFO ][logstash.pipeline] Starting pipeline {"id"=>"main", "pipeline.workers"=>1, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>5, "pipeline.max_inflight"=>125}
[2017-03-13T15:25:38,646][INFO ][logstash.pipeline] Pipeline main started
[2017-03-13T15:25:38,742][INFO ][logstash.agent] Successfully started Logstash API endpoint {:port=>9600}
```

## Kibana

Kibana可以为Logstash和ElasticSearch提供的日志分析友好的Web界面，可以帮助您汇总、分析和搜索重要数据日志。

### 1.解压进入目录

```
chu888chu888@hadoopmaster:~$ tar xvfz kibana-5.2.2-linux-x86_64.tar.gz
```

### 2.修改配置文件

```
nano config/kibana.yml
```

### 3.添加如下配置项

```
server.port: 5601
server.host: "192.168.1.159"
elasticsearch.url: "http://192.168.1.159:9200"
kibana.index: ".kibana"
```

### 4.启动服务

```
./bin/kibana
```

### 5.启动成功日志如下

```
chu888chu888@hadoopmaster:~/kibana-5.2.2-linux-x86_64/bin$ ./kibana
```

### 6.浏览器访问

默认第一次需要Configure an index pattern，默认的Index name是logstash-\*，直接create就行了。

```
http://192.168.1.159:5601/app/kibana#/management/kibana/index/?_g=()
```

The screenshot shows the Kibana Management interface. On the left, there's a sidebar with icons for Discover, Visualize, Dashboard, Timeline, Dev Tools, and Management. The main area has a header 'Management / Kibana' and tabs for 'Index Patterns', 'Saved Objects', and 'Advanced Settings'. A warning message says 'No default index pattern. You must select or create one to continue.' Below it, the 'Configure an index pattern' section has two checked options: 'Index contains time-based events' and 'Use event times to create index names [DEPRECATED]'. There's a text input field with 'logstash-\*' and a checkbox for 'Do not expand index pattern when searching (Not recommended)'. At the bottom, a message says 'Unable to fetch mapping. Do you have indices matching the pattern?'. In the top right corner, there are status indicators for CPU (18%), RAM (1.4K/8), and Disk (0.7%).

# Hadoop Training Sheet

#

## 1. 操作系统

- Linux操作系统（检查操作系统版本号）
- Linux操作系统概述
- 安装Linux操作系统
  - | CentOS、Ubuntu
- Linux Shell相关工具的使用
  - | Xshell、Xftp
- Linux图形界面、字符界面
- Linux基本命令
  - 查看主机名：hostname
  - 硬件配置：df、free、ethtool
  - 文件/文件夹操作：cd、mkdir、rm、cp、mv、touch、du、lsof
  - 查看文本：cat、less、tail、vim、vi
  - 查找类：grep、find
  - 压缩解压缩：tar、gzip、zip、unzip
  - 软件安装类：rpm、yum、apt-get
  - 帮助类：man
  - 时间类：date
  - IO类：lstat

- 权限类：sudo、chown、chmod、chgrp
- 端口连接类：netstat、ping、telnet
- 启停服务类：etc/init.d/mysql [start|stop|restart]
- 网页类：elinks <http://192.168.1.210:60010>
- 挂载类：mount、umount
- 用户、组群和权限管理
- 文件系统管理
- 软件包管理与系统备份
- Linux网络配置
- Linux基本服务配置
  - | DNS服务、DHCP服务、HTTP服务、FTP服务、SMTP服务、POP3服务
- Linux Shell命令
  - 文件及文本常用命令：tar、find、cut、wc、split、grep、head、tail、sed、awk
  - 系统运行状况命令：top、watch、free、mpstat、vmstat、iostat、pidstat、df、du
  - 系统运行进程命令：ps、nice、renice、lsof、pgrep、pkill
  - 追踪命令：strace
  - 排序命令：sort
  - 删除重复行：uniq
  - 正则表达式
- Linux Shell脚本编写
  - 定时备份系统日志
  - 自动监控其它主机硬盘及内存使用状况

- 自动化安装JDK、Tomcat

## 2. 数据库

- 关系型数据库原理
- 在Linux上安装Mysql、SQL-Server、DB2、Oracle数据库
- DDL、DML、DCL语法
  - | Mysql、SQL-Server、Oracle、DB2
- SQL基础
  - 基本语句：insert、select、where、update、delete、join、group by、having、desc、asc、limit、isnull、等
  - 函数：日期函数、嵌套函数、字符串函数、数字函数、聚集函数
- SQL高级
  - PL/SQL、if、case、loop、while、for、游标、视图、索引、存储过程、事务、SQL编程
- 数据库管理
  - 容量规划
  - 安全
  - 性能
  - 对象
  - 存储管理
  - 变化管理
  - 任务调度
  - 网络管理
  - 故障排查
  - 管理工具

- Mysql : Workbench、Navicat
- SQL-Server : SSMSE
- Oracle : OEM、PL/SQL developer、Toad
- DB2 : DB2top、Toad for db2
- 备份与恢复
  - 文件：参数文件、控制文件、数据文件、转储文件、重做日志
  - 备份：冷备份、热备份
  - 还原和恢复：备份控制文件、归档日志文件
- 数据库优化
  - 表：建立分区表、重建索引
  - I/O：将数据、日志、索引放在不同I/O设备
  - 切分：横/纵向分割表
  - 硬件：升级CPU、内存、网络带宽等
  - 业务分离：OLTP与OLAP分离
  - 字段选取：where后的查询字段避免冗余

### 3. 大数据

#### 一、原生Hadoop

#

- Hadoop框架
  - 大数据概念及应用场景
  - Hadoop介绍
  - Hadoop组件
- HDFS组件

- HDFS读取过程
- HDFS基本命令：cat、chgrp、chmod、chown、cp、df、du、find、get、ls、lsr、mkdir、mv、put、rm、rmdir、rmdir、tail、stat等
- Hive组件
  - Hive表结构与数据存储
  - Hive与RDBMS区别
  - Hive数据库与表
  - 基本HiveQL语法
  - 向Hive装载数据
- Sqoop组件
  - Sqoop工作原理
  - Sqoop数据流转
- Flume组件
  - Flume工作原理
  - Flume参数配置
  - 实时将系统日志文件导入HDFS
- HBase组件
  - HBase概念及应用场景
  - HBase与RDBMS联系与区别
  - HBase表结构与数据存储

## 二、TDH发行版本

#

安装前准备

- 操作系统版本 CentOS 6.3-6.5/REHL 6.3-6.5/ Suse SP2-SP3/操作系统是否干净？
- 是否需要配置sudo用户安装TDH？
- 机器硬件配置 CPU/MEM是否满足要求？/ 系统根分区大于300G?/千兆以上网络？
- 是否配置了SSD？
- 是否操作系统双盘RAID1，数据盘RAID0？
- 配置是否对称同构
  - (1) 磁盘同构： 数据盘对应的每块磁盘是否一样大？（严禁大小磁盘混合做数据盘，例如300G /mnt/ disk1, 2.7T /mnt/disk2）
  - (2) 网络同构： 每台机器网卡配置是否相同？
  - (3) CPU/内存大小是否同构：
- 系统时间是否正确。 > date -s ‘2015-11-11 09:45:00’
- 确认网络解析是用/etc/hosts文件还是DNS server。
  - (1) 推荐使用hosts文件解析。
  - (2) 若用hosts文件解析，确保/etc/resolv.conf 为空或隐掉。并保证/etc/nsswitch.conf 中 files 解析在DNS解析之前
  - (3) 各节点尽可能的在一个网段
- hostname只能是以字母和数字的组合(中间允许'-')，不能有“,” / “.” / “\_”等特殊字符。

### TDH安装与运维

- 安装
  - root安装、非root安装
  - 配置RACK（机架命名一定要以’/’开头，如 /default）
  - 添加节点、添加硬盘、升级Licence

- 配置检查

- Zookeeper的重要配置 Zookeeper 配置个数是否检查？（奇数个，10个节点以下3个，10-50个节点 5个）
- HDFS的重要配置 HDFS 的1个目录配置是否只包含 /mnt/disk\*的数据盘，SSD是否排除在外？
- YARN的重要配置
  - (1) YARN 的2个目录配置是否只包含 /mnt/disk\*的数据盘，SSD是否排除在外？
  - (2) YARN 的 vcore/Mem配置是否配置成了1个core对应2G内存？
- Inceptor的重要配置 (1) Inceptor 是否配置了HiveServer2（推荐 Kerberos+LDAP HA模式）
  - (2) Inceptor 的 fastdisk 是否配置了SSD？
  - (3) Inceptor 的localdir 配置里是否只包含 /mnt/disk\*，SSD是否排除在外？
  - (4) Inceptor 的资源配置是否合理？每个core是否都分配了1.5-2G内存？
- Hyperbase的重要配置
- Hmaster个数是否为奇数？（3个或者5个）
- Fair Schedule配置

- 日志相关

- Zookeeper的日志位置（/var/log/zookeeper1）
- HDFS的日志位置（/var/log/hdfs1）
- YARN的日志位置（/var/log/yarn1）
- Hyperbase的日志位置（/var/log/hyperbase1）
- Inceptor的日志位置（/var/log/inceptor1）

- 服务启停

- 查看机器已启动的服务
  - 各服务启停的顺序
  - Zookeeper的启停
  - HDFS的启停
  - Hyperbase的启停
  - YARN的启停
  - Inceptor的启停
- 管理页面
    - HDFS/YARN/Hyperbase/Inceptor重要的管理界面
    - HDFS健康状态的检查
    - YARN状态的检查
    - Hyperbase状态的检查
    - Inceptor运行状态的检查
  - 安全相关
    - 开启Kerberos
    - 添加/删除用户
  - HDFS状态检查
    - 查看HDFS状态
    - 查看损坏文件
    - fsimage和editlog存放的位置
  - Inceptor操作
    - Hive、Inceptor默认的分隔符
    - 创建外表以及数据存放位置
    - 创建ORC格式表及数据存放位置

- 创建Transaction ORC表及数据存放位置
- 创建Hyperbase外表及数据存放位置
- 数据迁移
  - Sqoop应用场景
  - Sqoop工具的使用
- 常用BI工具对接
  - Tableau对接
  - JDBC程序对接
  - SQuirreL对接
- Hyperbase操作
  - 全局索引、local索引、全文索引的概念与区别
  - localmode、clustermode的区别
  - Hyperbase.reader=true的含义
  - 4040页面上的表征
  - 怎样查看Hyperbase相关状态
  - 一个RegionServer最多host多少个Region？
  - 哪些情况会导致数据写入热点？
  - ObjectStore 程序怎么写？Json支持程度？
  - Batchinsert是什么？语法怎么写
- Bulkload相关
  - Bulkload的意义和本质
  - 各个阶段的目的
  - Bulkload有几个要点？
  - 什么是SQLbulkload？

- SQLBulkload操作（TPCDS一张表的数据）
- TPC-DS相关
  - 什么是TPC-DS？
  - TPC-DS中有多少个SQL？
  - 怎样运行TPC-DS？
  - TPC-DS截图

## 实验任务：

- 1、掌握安装**TDH**及其相关组件，熟悉安装前的**Checklist**和各组件的作用
- 2、使用**Sqoop**工具实现将**RDBMS**（**Mysql**、**DB2**、**Oracle**）与**HDFS**之间的数据迁移，在迁移的过程中完成数据的清洗和稽查，额外的可以通过**Sqoop**将数据导入**Inceptor**和**Hyperbase**表中
- 3、熟悉**Inceptor**中各函数的使用，特别是日期函数、字段截取函数、批量插入函数、行列转换函数等
- 4、使用**Flume**工具实现将半结构化数据批量导入**HDFS**
- 5、掌握在**Inceptor**中建立托管表、外表、**ORC**格式表、分区分桶表、内存表、**Hyperbase**表，并通过**Squirrel**和**Tableau**工具通过**JDBC**、**ODBC**连接**Inceptor**
- 6、掌握将全量数据和增量数据插入普通表、分区、分桶表中

- 7、掌握**TPC-DS**和**Bulkload**的使用方法
- 8、掌握**TDH**的**Guardian**组件的安装和**Kerberos**安全认证的配置
- 9、掌握**Hyperbase**中添加全局索引和**Elastic Search**索引
- 10、掌握在**HUE**的管理界面中添加**Oozie**工作流，并完成一个定时任务计划
- 11、熟悉在**CentOS6.5**下使用**PXE+Kickstart**实现无人值守安装操作系统
- 12、掌握通过日志的错误信息进行错误的排查