

Spark+hadoop+mllib 及相关概念与操作笔记

作者:	刘炜
版本:	0.1
时间:	2016-07-18

1、调研相关注意事项

a) 理解调研

调研的意义在于了解当前情况，挖掘潜在的问题，解决存在的疑问，并得到相应的方案。

b) 调研流程

首先明确和梳理现有的疑问是什么，要通过调研解决什么问题，然后再去做调研，发现问题，再解决问题。

c) 调研成果

最终需要得到结论与方案，以及详尽的论证理由，让别人信服。

d) 书写格式

版本与作者以及时间可以以表格的形式，整齐明了。
结论简洁明了，论证理由详尽写在后面。

2、Linux 常见命令

a) Locate 参数

Locate 可能查找不到最新添加的文件，因为它从数据库中找，有的时候没有更新就找不到。

locate 指令和 find 找寻档案的功能类似，但 locate 是透过 update 程序将硬盘中的所有档案和目录资料先建立一个索引数据库，在执行 locate 时直接找该索引，查询速度会较快，索引数据库一般是由操作系统管理，但也可以直接下达 update 强迫系统立即修改索引数据库。

b) 查看系统版本

Uname -a: 显示电脑及操作系统的相关信息

cat /proc/version: 说明正在运行的内核版本

cat /etc/issue: 显示的是发行版本的信息

lsb_release -a: (适用于所有的 linux, 包括 Redhat、SuSE、Debian 等发行版, 但是在 debian 下要安装 lsb)

3、Linux 环境变量的设置

Linux 中环境变量包括系统级和用户级, 系统级的环境变量是每个登录到系统的用户都要读取的系统变量, 而用户级的环境变量则是该用户使用系统时加载的环境变量。

a) 系统级:

/etc/profile: 用于交互的 Login shell, 交互式 shell 的环境变量。执行 bashrc 此文件为系统的每个用户设置环境信息, 当第一个用户登录时, 该文件被执行。并从/etc/profile.d 目录的配置文件中搜集 shell 的设置。

/etc/bashrc: 非交互式 shell 的环境变量。为每一个运行 bash shell 的用户执行此文件。当 bash shell 被打开时, 该文件被读取。有些 linux 版本中的/etc 目录下已经没有了 bashrc 文件。

/etc/environment: 在登录时操作系统使用的第二个文件, 系统在读取你自己的 profile 前, 设置环境文件的环境变量。

b) 用户级 (这些文件处于家目录下)

~/profile: 每个用户都可使用该文件输入专用于自己使用的 shell 信息, 当用户登录时, 该文件仅仅执行一次! 默认情况下, 他设置一些环境变量, 执行用户的.bashrc 文件。这里是推荐放置个人设置的地方

~/bashrc: 该文件包含专用于你的 bash shell 的 bash 信息, 当登录时以及每次打开新的 shell 时, 该文件被读取。

~/bash_profile 、~/bash_login: 如果有则不读取.profile(~/bash_profile or ~/bash_login - If one of these file exist, bash executes it rather than "~/profile" when it is started as a login shell. (Bash will prefer "~/bash_profile" to "~/bash_login"). However, these files won't influence a graphical session by default.)

~/pam_environment: 用户级的环境变量设置文件, 没有做测试, 不知道管不管用。

想对所有的用户生效, 那就需要设置系统级的环境变量。反之, 需要修改用户级的文件 (最好是修改.profile 文件, 理由上面已经讲了)。

c) 生效

Source ~/.profile

4、Hadoop 基本命令

Hadoop 文件系统命令：hadoop fs -linux 命令 参数

Eg: [root@master ~]# hadoop fs -ls /

Eg: [root@master ~]# hadoop fs -cat /lwtest/lw.txt

其中 fs 是 fileSystem，然后 - 后面接一般的 linux 指令就行，即 hadoop 文件系统的操作和 linux 文件系统的操作基本上一致，这样不用浪费时间去记很多命令了。

5、相关概念

Hbase：分布式、面向列的开源数据库

Yarn：Hadoop 资源管理器，可以为上层应用提供统一的资源管理和调度，提高资源利用率及方便数据共享

6、Hadoop, Hdfs 连接操作

a) 问题访问的是本地的文件系统而不是 HDFS

原因：由于路径写错了

```
//readFile("D:/pythonspace/1.py");  
| readFile("lwtest/test.txt");//错误,读取本地文件系统/user/lw_co/lwtest/test.txt  
readFile("/lwtest/test.txt");//正确,读取hdfs上
```

```
//读取文件的内容  
public static void readFile(String filePath) throws IOException{  
    // Configuration conf = new Configuration();  
  
    //conf.addResource("hdfs-site.xml");  
    //conf.addResource("core-site.xml");  
    System.out.println(conf.get("dfs.namenode.secondary.http-address"));  
    //FileSystem fs = FileSystem.get(conf);  
    Path srcPath = new Path(filePath);  
    InputStream in = null;  
    try {  
        in = fs.open(srcPath);  
        IOUtils.copyBytes(in, System.out, 4096, false); //复制到标准输出流  
    } finally {  
        IOUtils.closeStream(in);  
    }  
}
```

b) 建立连接

```
Configuration conf = new Configuration();  
//conf.set("fs.defaultFS", "hdfs://10.3.9.135:8020")
```

默认读取 Hdfs-site.xml,core-site.xml,mapred-site.xml 等配置来建立连接

7、Spark 集群

a) Spark web UI 控制端

<http://10.3.9.135:8180/>

可以在./sbin/start-master.sh 下查看

b) 命令: **clush -a -b jps** 显示当前运行的服务进程

c) Spark 集群部署方式

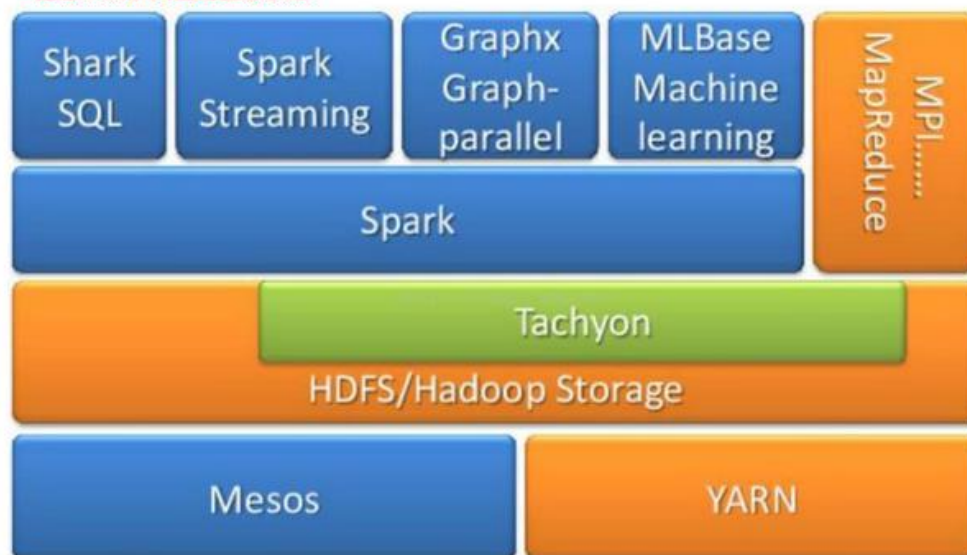
参考:

<http://www.cnblogs.com/liuyifeng/p/5690627.html>

<http://www.aboutyun.com/forum.php?mod=viewthread&tid=7115>

内存计算框架: Spark

已经形成了自己的生态系统



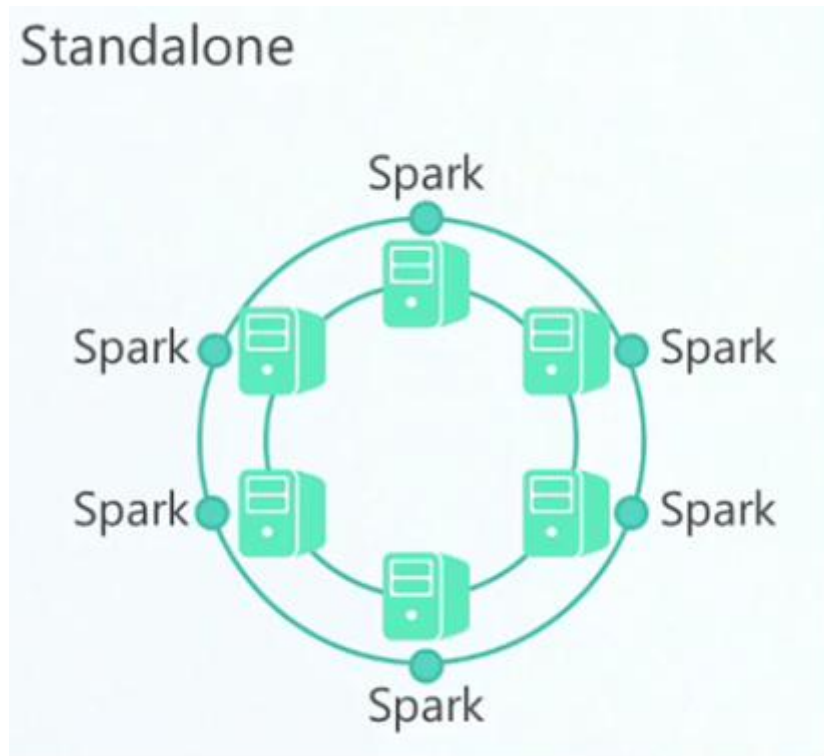
MR: 离线计算框架

Storm: 实时计算框架

Spark: 内存计算框架

i. Standalone

分别在集群的每台机器安装 Spark,再启动相应的 master 和 slave



ii. Spark On Mesos

iii. Spark On Yarn

使用 Spark 客户端向 yarn 提交任务运行。

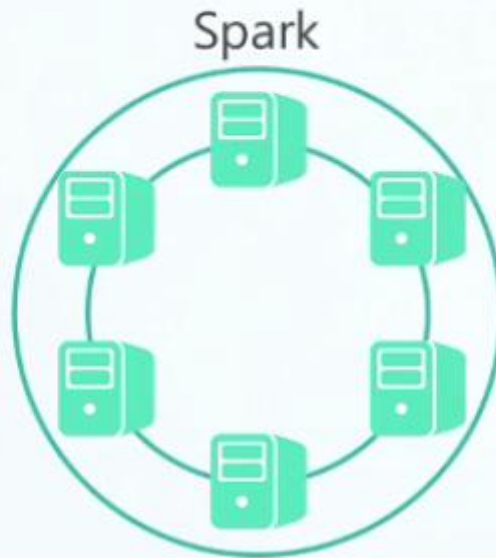
部署方式:

1.将 spark 部署包放到 yarn 集群某个节点上面

2.yarn 客户端读取 yarn 集群配置文件,在此过程中,spark 的 master 和 slave 节点不需要启动

Ps:Yarn(淘宝团队) Mesos(豆瓣)

- ▶ Spark On Mesos
- ▶ Spark On Yarn



d) Hadoop web 控制台端口

nameNodeIP:端口

Eg:

10.3.9.135:8088 Hadoop 资源管理

10.3.9.135:50070 HDFS 文件管理，节点信息

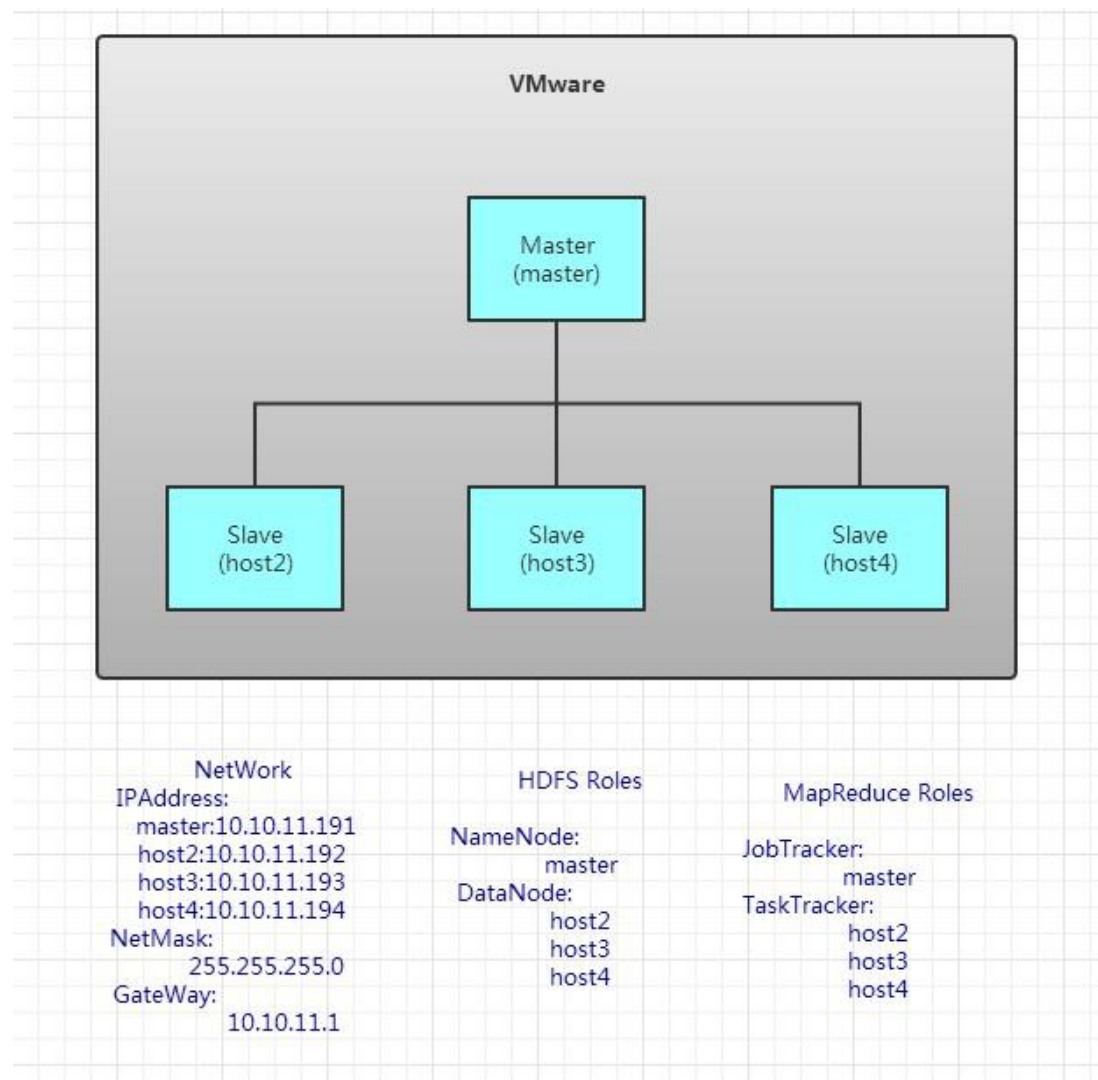
<http://10.3.9.135:50070/explorer.html#/> 可以查看文件

参见:

<http://www.cnblogs.com/laov/p/3433994.html>

<http://www.cnblogs.com/ggjucheng/archive/2012/04/17/2454590.html>

帮助理解:



图：Hadoop 分布图

Browse Directory

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	root	hdfs_superuser	0 B	2016年7月8日 下午2:30:04	0	0 B	hbase
-rw-rw-r--	root	hdfs_superuser	505.78 KB	2016年5月9日 下午4:46:54	3	128 MB	lhb_testdata
-rw-rw-r--	root	hdfs_superuser	125 B	2016年7月19日 下午3:50:02	3	128 MB	lw._COPYING_
drwxr-xr-x	root	hdfs_superuser	0 B	2016年7月19日 下午3:59:03	0	0 B	lwtest
-rw-rw-r--	root	hdfs_superuser	72 B	2016年7月20日 下午4:36:41	3	128 MB	mllibTestData
drwxr-xr-x	root	hdfs_superuser	0 B	2016年6月1日 下午10:15:22	0	0 B	system
-rw-rw-r--	root	hdfs_superuser	125 B	2016年7月19日 下午3:50:19	3	128 MB	test.txt
drwxr-xr-x	root	hdfs_superuser	0 B	2016年7月8日 下午1:23:27	0	0 B	tmp
drwxr-xr-x	root	hdfs_superuser	0 B	2016年5月24日 下午3:50:55	0	0 B	user

Hadoop, 2015.

图：Hadoop HDFS 文件浏览

常用的端口配置

HDFS端口

参数	描述	默认	配置文件	例子值	
fs.default.name	namenode RPC交互端口	8020	core-site.xml	hdfs://master:8020/	
dfs.http.address	NameNode web管理端口	50070	hdfs-site.xml	0.0.0.0:50070	
dfs.datanode.address	datanode 控制端口	50010	hdfs-site.xml	0.0.0.0:50010	
dfs.datanode.ipc.address	datanode的RPC服务器地址和端口	50020	hdfs-site.xml	0.0.0.0:50020	
dfs.datanode.http.address	datanode的HTTP服务器和端口	50075	hdfs-site.xml	0.0.0.0:50075	MR端口

参数	描述	默认	配置文件	例子值
mapred.job.tracker	job tracker交互端口	8021	mapred-site.xml	hdfs://master:8021/
mapred.job.tracker.http.address	job tracker的web管理端口	50030	mapred-site.xml	0.0.0.0:50030
mapred.task.tracker.http.address	task tracker的HTTP端口	50060	mapred-site.xml	0.0.0.0:50060

其他端口

参数	描述	默认	配置文件	例子值
dfs.secondary.http.address	secondary NameNode web管理端口	50090	hdfs-site.xml	0.0.0.0:28680

图：Hadoop 常用端口

e) Spark 例子运行

```
./bin/run-example SparkPi 10
./bin/spark-shell --master local[2]
```


The `--master` option specifies the master URL for a distributed cluster, or `local` to run locally with one thread, or `local[N]` to run locally with N threads. You should start by using `local` for testing. For a full list of options, run Spark shell with the `--help` option.

这`--master` 选项指定 `master url` 为一个分布式集群还是本地单线程的，或者 `local[N]`本地 N 线程的。你应该开始使用本地测试，运行 `Spark shell --help` 选项。

f) Mllib 例子运行

```
./bin/run-example mllib.JavaKMeans mllibTestData/kmeans_data.txt 3 100
```

`mllib.JavaKMeans` 为相应的机器学习程序，是 `mllib` 文件夹下的 `JavaKMeans`，后面三个是数据集 及相应参数。这里是 K 均值算法，后面 3 表示 K 的大小，100 表示迭代次数。

g) Job 提交

参考: <http://spark.apache.org/docs/latest/submitting-applications.html>

本地 8 线程运行应用

Run application locally on 8 cores

```
./bin/spark-submit \
```

```
--class org.apache.spark.examples.SparkPi \
```

```
--master local[8] \
```

```
/path/to/examples.jar \
```

```
100
```

Eg: `./bin/spark-submit --class org.apache.spark.examples.SparkPi --master local[2] ./lib/spark-examples-1.6.1-hadoop2.7.1.jar 10`

单集群

Run on a Spark standalone cluster in client deploy mode

```
./bin/spark-submit \
```

```
--class org.apache.spark.examples.SparkPi \
```

```
--master spark://207.184.161.138:7077 \
```

```
--executor-memory 20G \
```

```
--total-executor-cores 100 \
```

```
/path/to/examples.jar \
```

```
1000
```

Run on a Spark standalone cluster in cluster deploy mode with supervise

```
./bin/spark-submit \
```

```
--class org.apache.spark.examples.SparkPi \
```

```
--master spark://207.184.161.138:7077 \
```

```
--deploy-mode cluster \
```

```
--supervise \  
--executor-memory 20G \  
--total-executor-cores 100 \  
/path/to/examples.jar \  
1000
```

#在 Spark on Yarn 的集群上提交应用 （目前的集群）

#Run on a YARN cluster

export HADOOP_CONF_DIR=XXX

```
./bin/spark-submit \  
--class org.apache.spark.examples.SparkPi \  
--master yarn \  
--deploy-mode cluster \ # can be client for client mode  
--executor-memory 20G \  
--num-executors 50 \  
/path/to/examples.jar \  
1000
```

Eg:例子

```
[root@master spark-1.6.1]#  
./bin/spark-submit \  
--class org.apache.spark.examples.SparkPi \  
--master yarn \  
--deploy-mode cluster \  
./lib/spark-examples-1.6.1-hadoop2.7.1.jar \  
10
```

其中 SparkPi 是通过蒙特卡罗思想就 Pi 的算法，10 为其参数，大概指的是在单位为 10 的一个正方形里，正方形里包含一个半径为 5 的圆，然后生成随机数，均匀分布，数点落在圆里的数及全部的点数，通过面积的比例等于随机数分布的比例来求得 Pi。

--deploy-mode cluster 在 Yarn 集群中需要指定。

Eg:例子

```
root@master spark-1.6.1]#  
./bin/spark-submit \  
--class org.apache.spark.examples.mllib.JavaLR \  
--master yarn \  
--deploy-mode cluster \  
./lib/spark-examples-1.6.1-hadoop2.7.1.jar \  
mllibTestData/lr-data/random.data 1 10
```

注意最后资源也应该是在集群 hdfs 上的文件

h) 结果查看

参考:

http://www.360doc.com/content/14/0810/12/7853380_400765442.shtml

<http://www.iteblog.com/archives/1028>

```
System.out.println("Pi is roughly " + 4.0 * count / n);
```

在 yarn 上运行，程序默认输出直接输出到 logs 里面。

Yarn logs -applicationId xxx 可以查看运行结束的 Application 日志

Eg:

```
yarn logs -applicationId application_1467954918322_0037
```

输出:

...省略...

LogType:stdout

Log Upload Time:星期一 七月 25 11:18:59 +0800 2016

LogLength:23

Log Contents:

Pi is roughly 3.141328

End of LogType:stdout

...省略...

当然可以将结果输出到文本里面，代码:

```
counts.saveAsTextFile("/home/wyp/result");
```

或者:

```
counts.saveAsHadoopFile("/home/wyp/result",  
                        Text.class,  
                        IntWritable.class,  
                        TextOutputFormat.class);
```

将结果存储到 HDFS 上的/home/wyp/result 文件夹里，第一种输出格式是 (key,value),第二种输出格式为 key value.

我们可以根据自己的需要定义一个自己的输出格式，而且我们在输出的时候如果文件比较大，还可以指定输出文件的压缩方式。

i) 日志

```
hdfs dfs -ls
```

```
/tmp/logs/root/logs/tmp/logs/root/logs/application_1467954918322_0059
```

Hadoop 下 spark 应用日志存在/tmp/logs/root/logs/tmp/logs/root/logs/下面。每个应用 id 对应一个目录，里面有在不同节点上跑的日志的相应记录。同 Yarn logs -applicationId xxx 所看到的信息。

8、Maven 与 eclipse

a) eclipse 常用快捷键

Alt+/补全

Ctrl+/增加注释

Ctrl+D 删除当前行

b) 什么是 maven

Maven 项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。

Maven 这个单词来自于意第绪语（犹太语），意为知识的积累，最初在 Jakarta Turbine 项目中用来简化构建过程。当时有一些项目（有各自 Ant build 文件），仅有细微的差别，而 JAR 文件都由 CVS 来维护。于是希望有一种标准化的方式构建项目，一个清晰的方式定义项目的组成，一个容易的方式发布项目的信息，以及一种简单的方式在多个项目中共享 JARs。

c) 安装 Maven

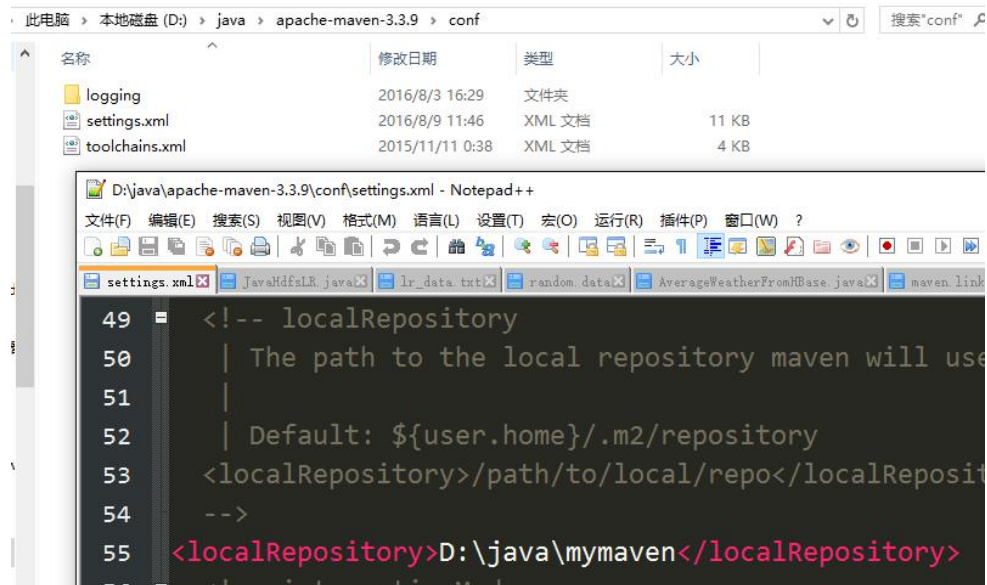
<http://dead-knight.iteye.com/blog/1841658>

i. 安装 maven，并设置环境变量（网上有的是）

ii. 查看配置是否完成。mvn -v 出现版本信息，表示配置成功

```
D:\技术书籍\算法>mvn -v
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: D:\java\apache-maven-3.3.9
Java version: 1.8.0_74, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_74\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

iii. 设置仓库位置

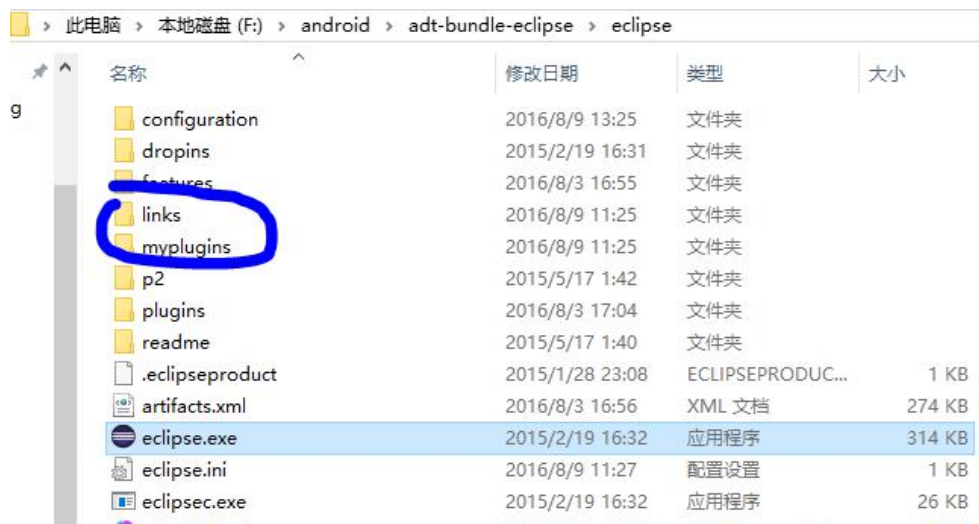


默认的 maven 仓库位置为：C:\Users\Administrator\.m2（其中 Administrator 为当前账号）
仓库是用来存 Maven 运行时的一些插件的。比如说 archetyper 的插件。

d) Eclipse 安装 maven 插件

Maven 插件下载：<http://pan.baidu.com/s/1i5weBZZ>

解压后，把 links、myplugins 文件夹放到 eclipse 安装目录下，如下图所示：

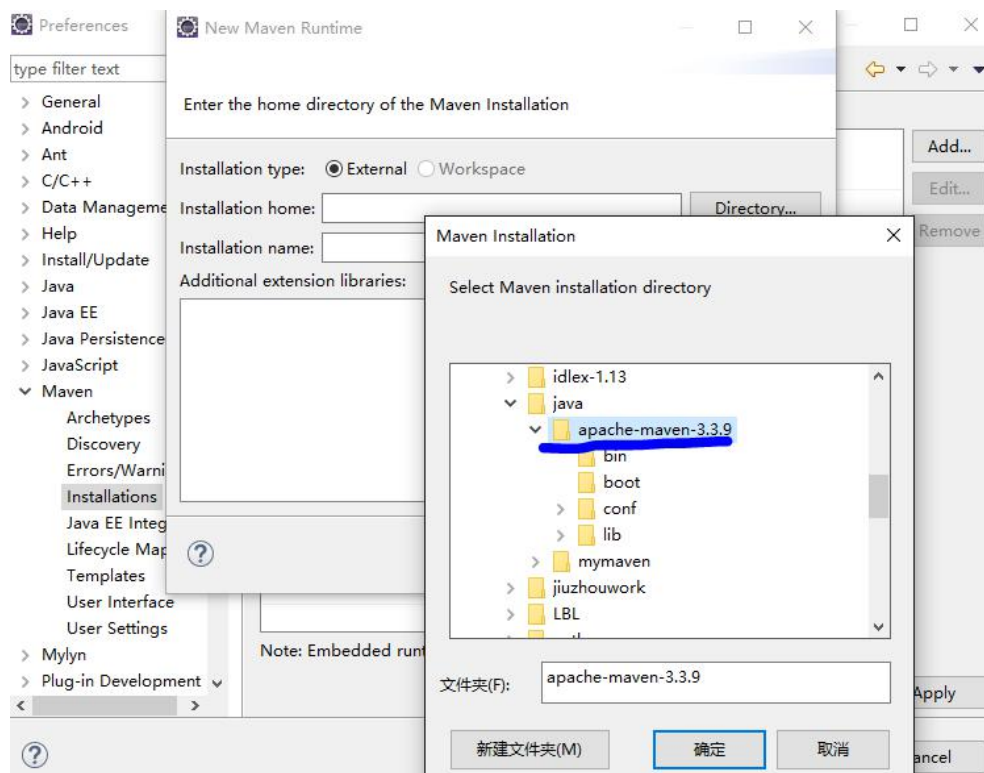
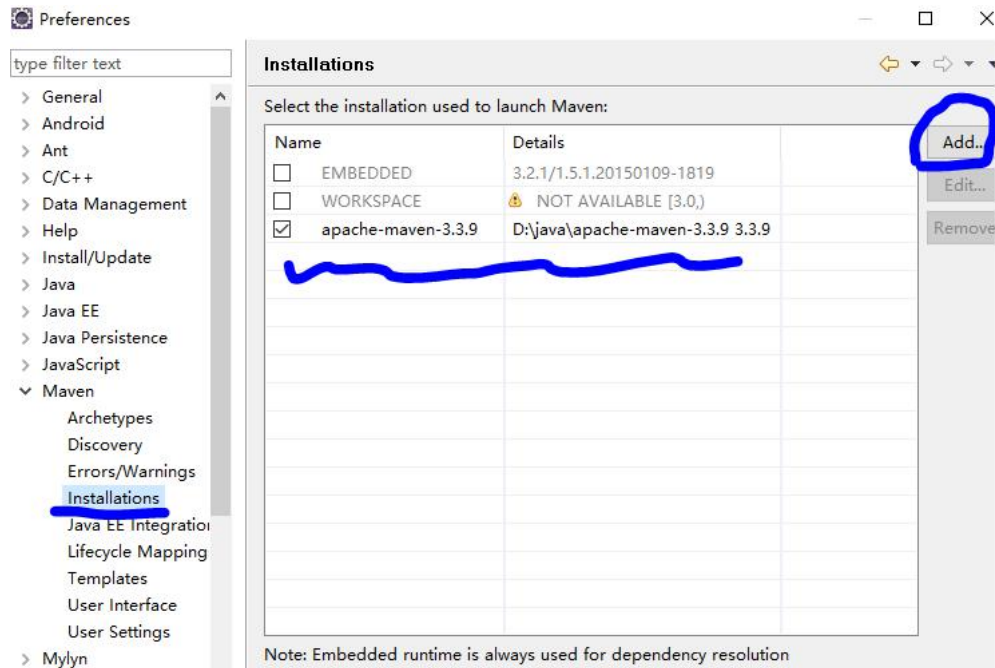


并且修改 links 下的 maven.link 文件。指向 myplugins 目录即可。我的配置为：

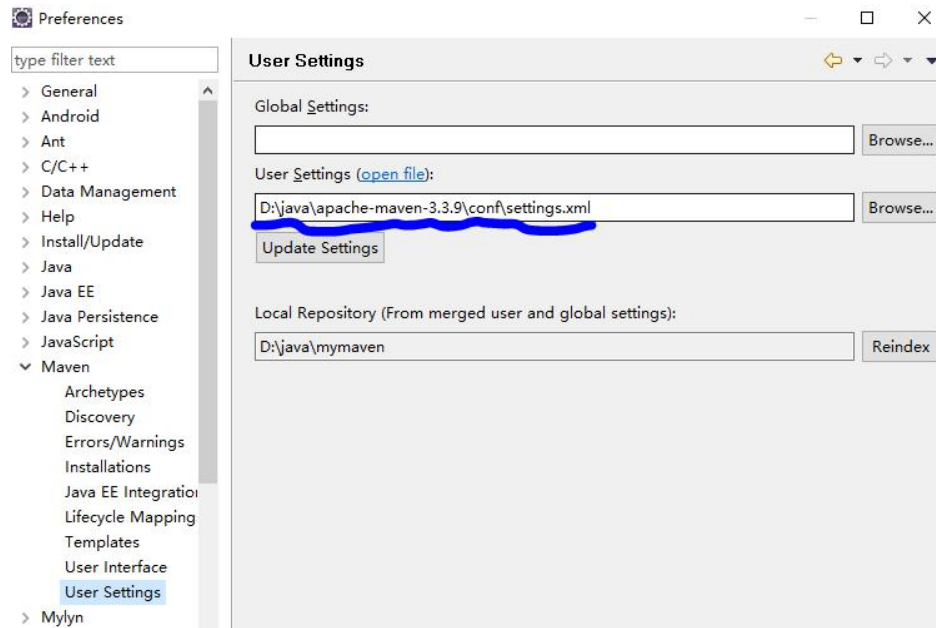
path=F:/android/adt-bundle-eclipse/eclipse/myplugins/maven

重启 eclipse

选择 window->preferences->maven，如下图所示：

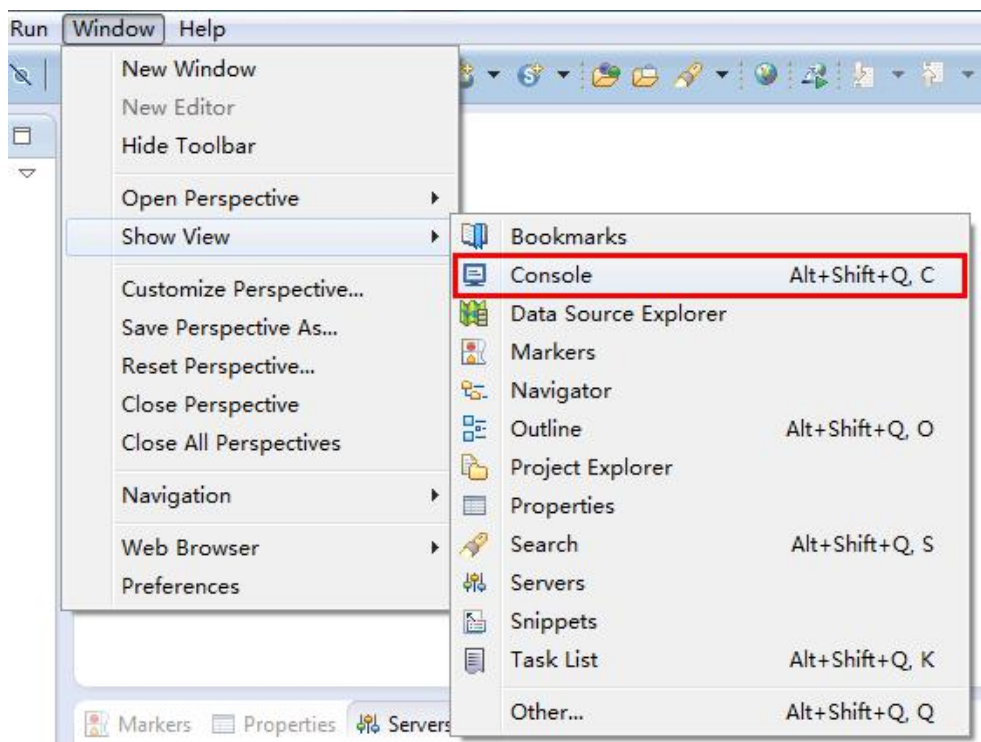


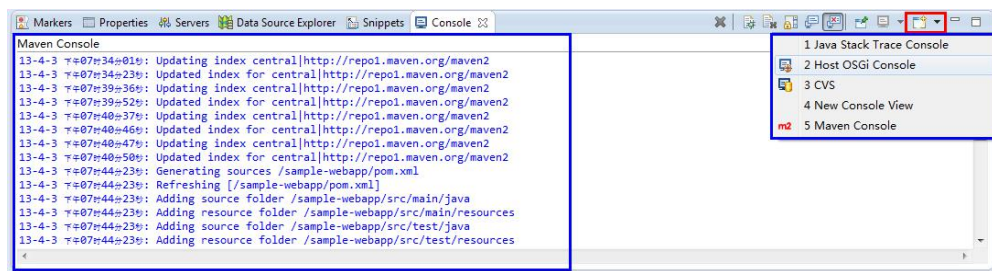
继续选择“User Settings”，配置 maven 的 setting 文件，如下图所示：



好，此时 maven 插件安装完毕。

e) 查看 maven 的 console





f) 注意事项

Maven-archetype-quickstart 与 maven-archetype-profiles 结构是一样的，即 profiles 即为 quickstart.

g) 错误处理

i. Could not resolve archetype

org.apache.maven.archetypes:maven-archetype-quickstart

解决方式：

1. 从

<http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.jar> 下载最新版 maven-archetype-quickstart-1.1.jar

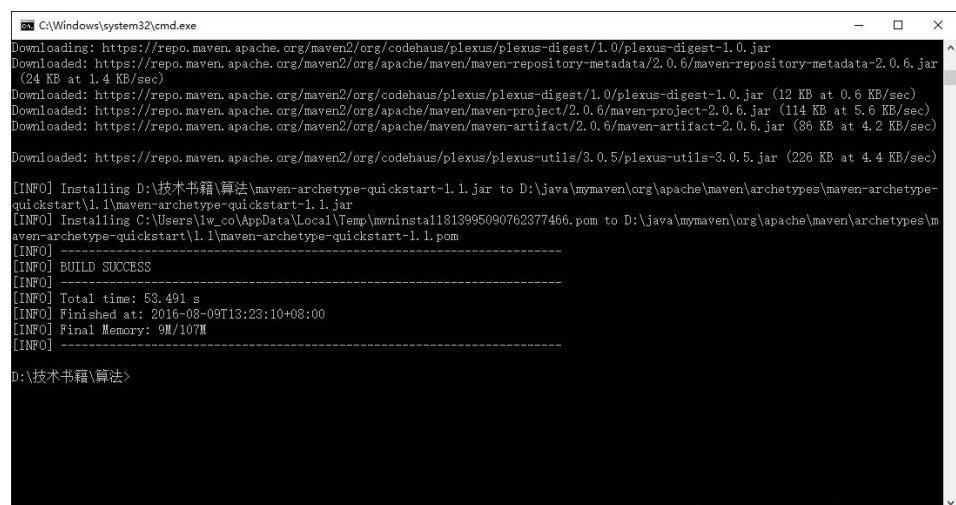
2. 命令行到下载目录下执行 mvn install:install-file

-DgroupId=org.apache.maven.archetypes

-DartifactId=maven-archetype-quickstart -Dversion=1.1

-Dpackaging=jar -Dfile=maven-archetype-quickstart-1.1.jar

出现 BUILD SUCCESS 则表示安装成功了，就可以通过 eclipse 建 quickstart 结构的工程了。如下图所示：



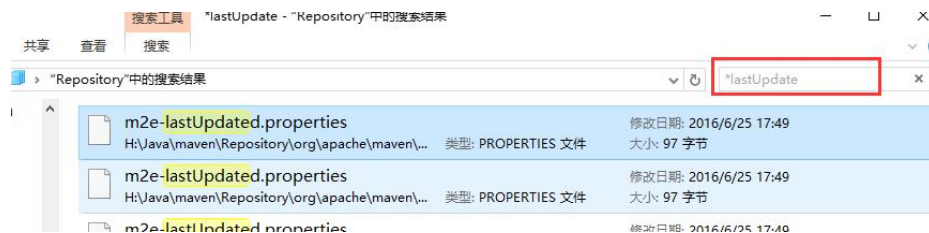
h) Pom.xml 红叉

<http://www.cnblogs.com/mymelody/p/5616685.html>

- i. dependency 格式不对, 或者 jar 包没有, 使得对应的包没有下载下来
- ii. eclipse 没有自己下载, 出现 missing 情况

解决方法:

1. 找到我们的本地 maven 仓库目录比如我的 D:\java\mymaven
2. 搜索出该目录下的*lastUpdated.properties 文件并删除, 如下图所示, 可以通过模糊搜索匹配出这样的文件



3. Maven 更新当前项目, maven 就会继续下载缺失的依赖 jar 包, 直至缺失 jar 包下载完成, 上述问题就解决了。

i) 远程仓库

参考: <http://www.cnblogs.com/dingyingsi/p/3856456.html>

仓库:

<http://repo2.maven.org/maven2/>

<http://uk.maven.org/maven2/>

<http://repository.apache.org>

<https://maven.java.net/content/groups/public/>

Other:

<https://repository.jboss.org/nexus/content/repositories/releases/>

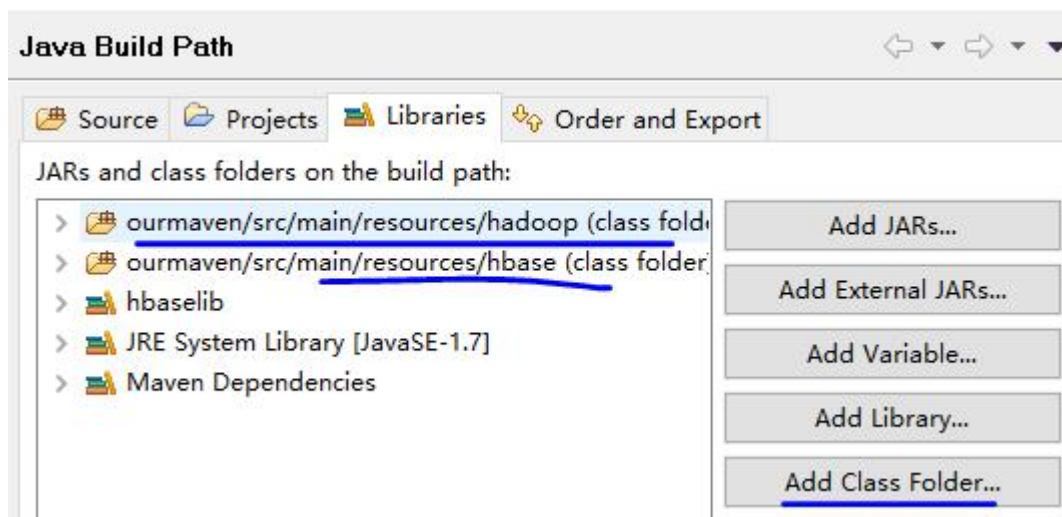
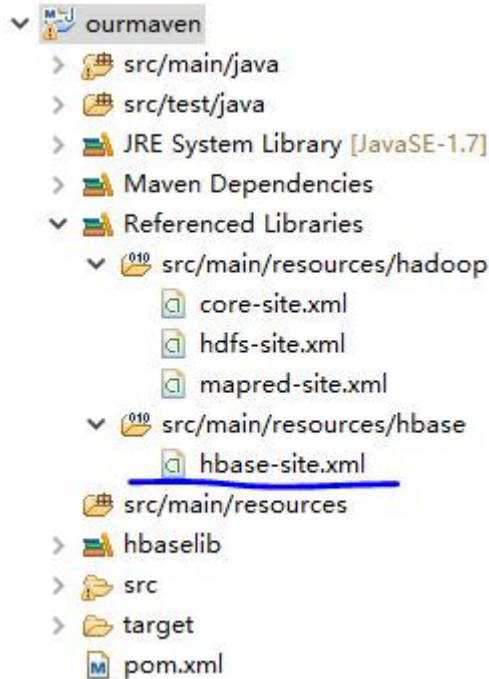
<http://repository.jboss.org/nexus/content/groups/public/>

9、Eclipse 与 HBASE 开发环境搭建

a) 首先

将 Hbase 的配置文件 hbase-site.xml 下载到本地, 建立工程后将其引入到工程里。这里将其放到 main 下的 resources 文件夹下。

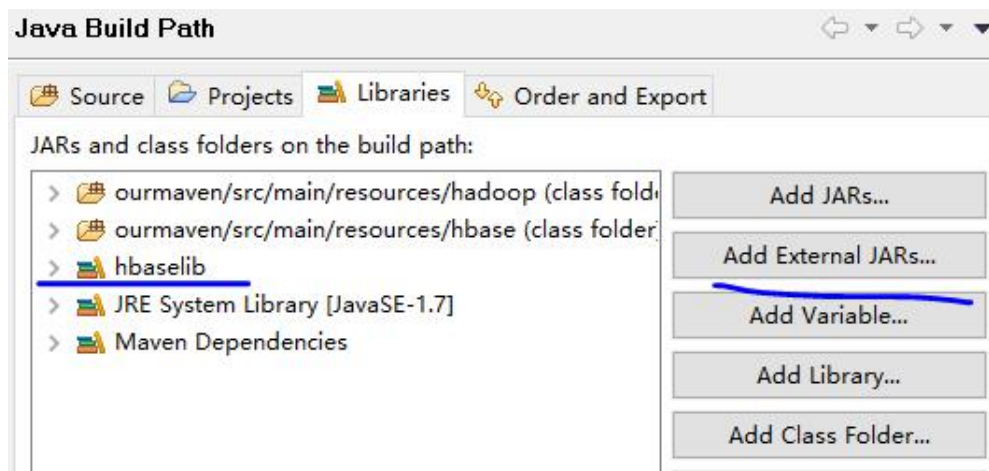
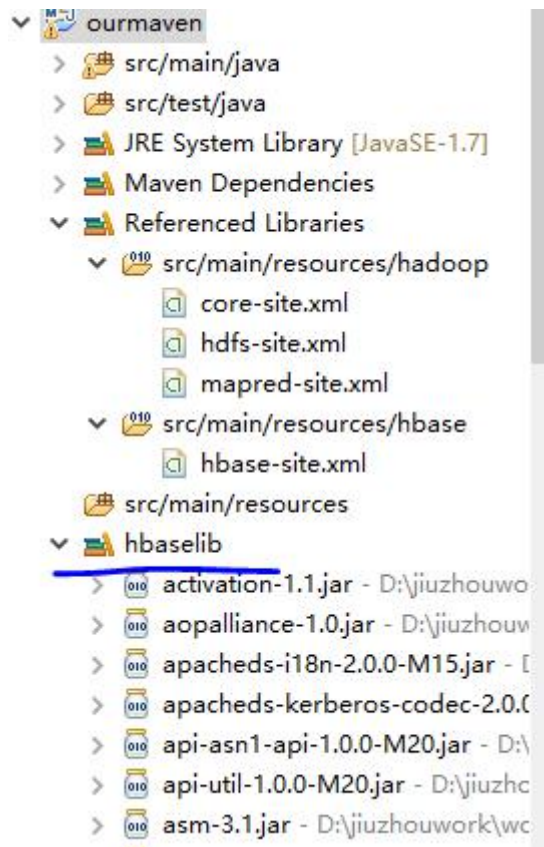
Ps: 为什么要加入到 build path, 因为在编译时会通过 path 去查找和检查文件, 然后在运行时候利用。



如果要用到 MapReduce 的话还应用把 hadoop 相应的配置文件放进来
如果要用到 Spark 的话当然要用到 spark 的配置文件

b) 不用 maven

将 Hbase 安装包下 lib 下的 jar 包拷贝下来，然后放到本地，将 jar 包加入到 build path 里面。



这样就可以了，import 这些就可以正确的找到了。

```

1 package mytest.ourmaven;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import org.apache.hadoop.conf.Configuration;
8 import org.apache.hadoop.hbase.HBaseConfiguration;
9 import org.apache.hadoop.hbase.HColumnDescriptor;
10 import org.apache.hadoop.hbase.HTableDescriptor;
11 import org.apache.hadoop.hbase.KeyValue;
12 import org.apache.hadoop.hbase.MasterNotRunningException;
13 import org.apache.hadoop.hbase.ZooKeeperConnectionException;
14 import org.apache.hadoop.hbase.client.Delete;
15 import org.apache.hadoop.hbase.client.Get;
16 import org.apache.hadoop.hbase.client.HBaseAdmin;
17 import org.apache.hadoop.hbase.client.HTable;
18 import org.apache.hadoop.hbase.client.Result;
19 import org.apache.hadoop.hbase.client.ResultScanner;
20 import org.apache.hadoop.hbase.client.Scan;
21 import org.apache.hadoop.hbase.client.Put;
22 import org.apache.hadoop.hbase.util.Bytes;
23
24 public class MyCase {

```

Ps: 在依赖包里没有的但却能正确的运用的, 说明在 java 的本身包里面, 或者在 classpath 里面, 或者在 jre/lib/, jre/lib/ext 里面, 这里的包将被自动搜索。《Java 核心技术第八版》第 4.8 类路径

c) 用 maven

i. Maven 工程

1. GroupId: 为组织名倒写如 com.arvidlw
2. ArtifactId: 这个一般为工程名, 如 bigdata
3. Project: 项目名

ii. Maven 注意

1. 打包时默认不加入依赖包的
<http://blog.csdn.net/jbgtwang/article/details/38226459>
2. 所以想打出一个独立的可运行 jar 包的话直接 mvn clean install package 是不行的。需要略改动下 pom 文件, 加入如下 plugin

iii. 依赖包

```

<dependencies>
  <dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>

```

```

        <version>0.98.8-hadoop2</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>

```

远程仓库比如说是: <http://uk.maven.org/maven2/>

则第一个意思是: 在这个仓库下的

org/apache/hbase/hbase-client/0.98.8-hadoop2/下的 jar 文件中去找
hbase-client-0.98.8-hadoop2.jar

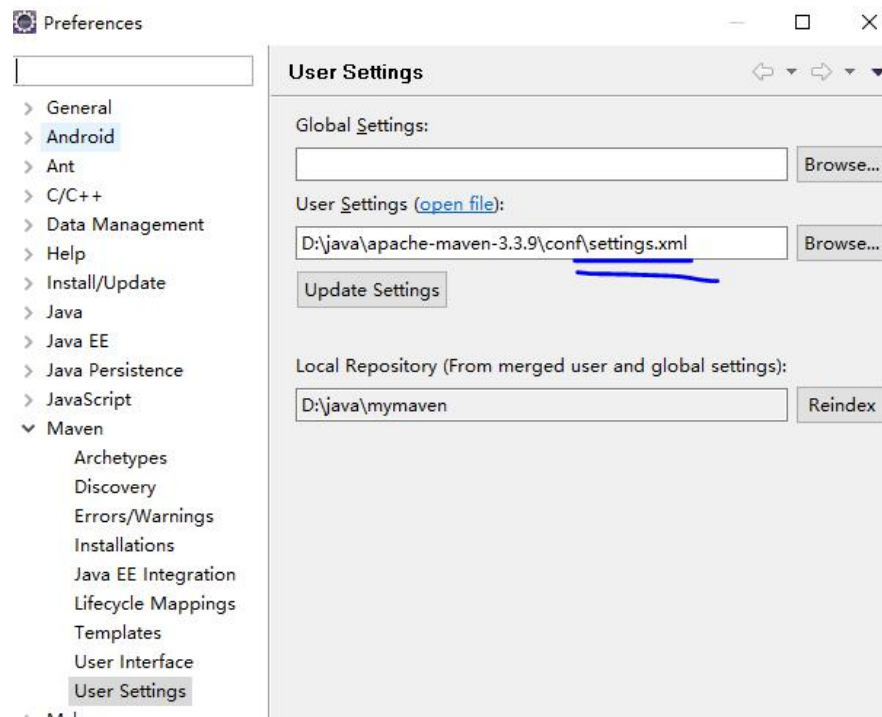
iv. Pom.xml 出错出现红叉

要么格式不对, 相应的文件在远程仓库里没有, 要么就是没有把对应的文件下载下来, 要么就是联网缓慢下载不下来。

解决办法 (三种):

<http://blog.csdn.net/dmlcq/article/details/51865887>

1. 右键项目, 点击 maven, 然后 update project, 可以选择 force update。
2. 找到项目 pom.xml 的目录, 然后用 mvn clean 与 mvn install 重新下载编译库。
3. 换一个远程仓库, 在 user setting 中找到对仓库的配置, 然后, 编辑设置文件, 更换远程仓库地址。



<https://maven2-repository.java.net/>

Eg1:在 settings 中

```

<profiles>
  <profile>
    <id>maven-repository</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>java.net-Public</id>
        <name>Maven Java Net Snapshots and
Releases</name>
        <url>https://maven.java.net/content/groups/public/</url>
      </repository>
    </repositories>
    <pluginRepositories>
      ...
    </pluginRepositories>
    ...
  </profile>
</profiles>

```

Eg2:在 pom.xml 中

```

<repositories>
  <repository>
    <id>java.net-Public</id>
    <name>Maven Java Net Snapshots and Releases</name>

    <url>https://maven.java.net/content/groups/public/</url>
    <updatePolicy>never</updatePolicy>
  </repository>
</repositories>

```



```

    <profile>
    <id>maven-repository</id>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
        <repository>
            <id>maven2</id>
            <name>ukmaven2</name>
            <url>http://uk.maven.org/maven2/</url>
        </repository>
    </repositories>

    </profile>
</profiles>

```

当用新的仓库时，再次更新的时候就会从新的远程仓库上下载，上面的库类。

d) Log4j 放在哪

<http://blog.csdn.net/lifuxiangcaohui/article/details/11042375>

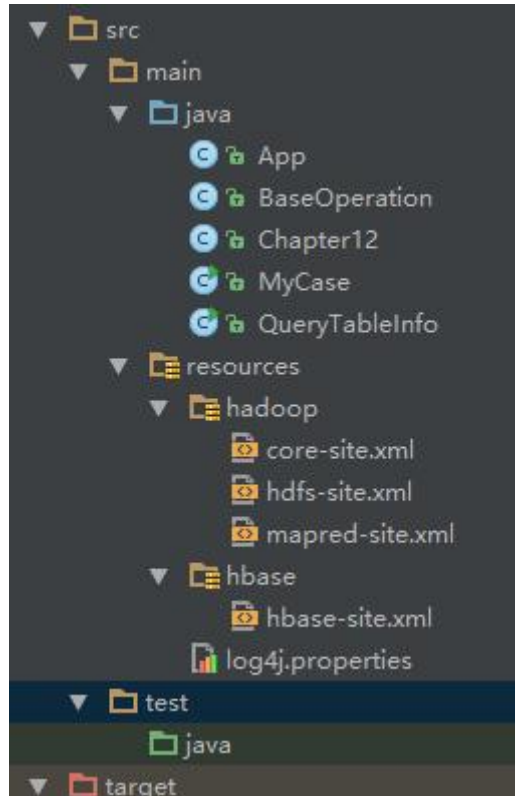
放在 build_path 包含的路径里面，即它开始会去找的地方。

可以放在资源文件中，因为开始会去找到然后读

PropertyConfigurator.configure("../log4j.properties");

括号中为路径，只要能找到就 ok.

在这时我放在 resoures 中



出现 WARN Please initialize the log4j system properly

由于项目下没有对 log4j 进行配置

<http://javapub.iteye.com/blog/866664>

<http://www.cnblogs.com/jbelial/archive/2012/06/05/2536814.html>

解决:

建立 log4j.properties 文件, 写入如下:

log4j.rootLogger=DEBUG, stdout

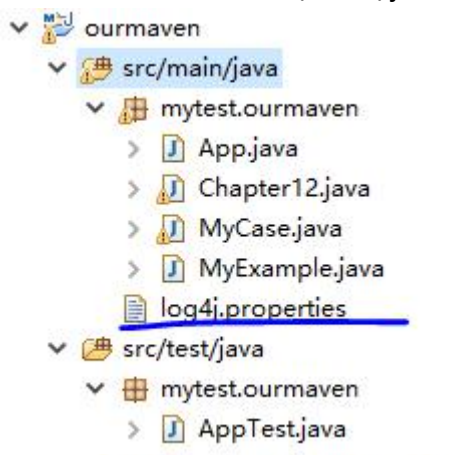
log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%c{1} - %m%n

log4j.logger.java.sql.PreparedStatement=DEBUG

这个文件这里放在 src/main/java 的根目录中

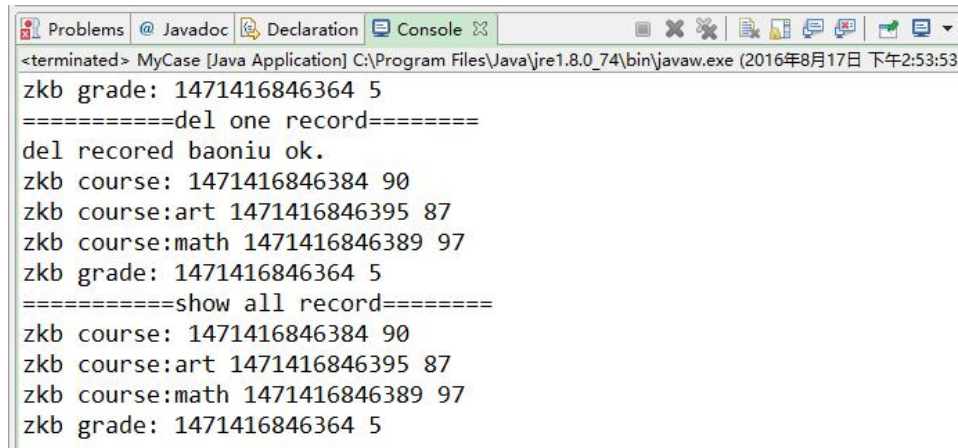


e) 出现 **not locate executable null\bin\winutils.exe**

<http://www.tuicool.com/articles/iABZJj>

程序需要找到 **winutils.exe**, 具体做什么的还不清楚, 就是设置在 windows 运行的启动环境 (不管他程序还是可以正常运行吧, 2.X)。

Winutils.exe, 是通过它来执行我们编写的程序来连接 **hbase** 吧



```
<terminated> MyCase [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (2016年8月17日 下午2:53:53)
zkb grade: 1471416846364 5
=====del one record=====
del recored baoniu ok.
zkb course: 1471416846384 90
zkb course:art 1471416846395 87
zkb course:math 1471416846389 97
zkb grade: 1471416846364 5
=====show all record=====
zkb course: 1471416846384 90
zkb course:art 1471416846395 87
zkb course:math 1471416846389 97
zkb grade: 1471416846364 5
```

Java:

java.exe 用于启动 window console 控制台程序

javaw.exe 用于启动 GUI 程序

javaws.exe 用于 web 程序。

jvm.dll 就是 java 虚拟机规范在 windows 平台上的一种实现

<http://blog.csdn.net/topwqp/article/details/8595936>

作用:

1.读取 **hbase** 数据你首先需要一个 **client**, **jar** 包里不集成 windows 环境 **client** 的。

2.你需要一个可以运行的 **client**——windows 环境下就是 **exe**(**hbase** 默认是安装在 linux 下的)。

3.**hbase** 是完全依赖 **hadoop** 的, **hadoop** 为了满足 windows 用户提供了启动环境, 在 **hadoopX.X/bin/**下边的 **winutils.exe**。

4.悲伤的是, 从 **hadoop2.2** 开始, 此文件莫名其妙地不打包了! 任性!

5.此路径的引用是用 **HADOOP_HOME** 变量或者是 **hadoop.home.dir** 配置都可以读取的

6.不需要完整安装 **hadoop2.2**, 只需要 **winutils.exe**, 并指定位置就好

7.**System.setProperty("hadoop.home.dir", "X:/yyy");**java 设置系统变量, 并在 **X:/yyy/bin/**下放好 **winutils.exe** 就可以了, 完全不需要安装, 配置环境变量什么的 --

解决方法:

i. 下载 **winutils** 的 windows 版本

GitHub 上, 有人提供了 **winutils** 的 windows 的版本, 项目地址是:

<https://github.com/ArvidLW/hadoop-common-2.2.0-bin> , 直接下载此项目的 **zip** 包, 下载后是文件名是 **hadoop-common-2.2.0-bin-master.zip**, 随便解压到一个目录

ii. 配置环境变量

增加用户变量 HADOOP_HOME，值是下载的 zip 包解压的目录，然后在系统变量 path 里增加\$HADOOP_HOME\bin 即可。

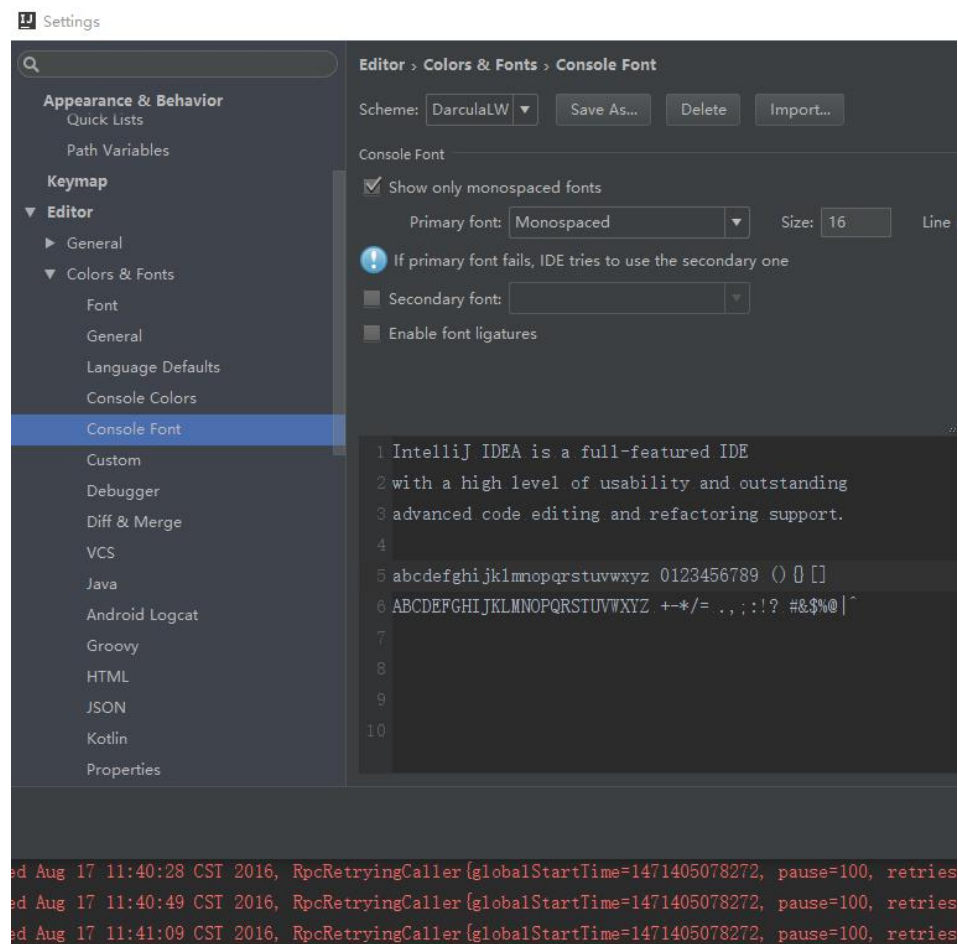
再次运行程序，正常执行。

10、IntelliJ idea 与 HBASE 开发环境配置

a) intellij 设置字体

编辑器字体：settings->editor

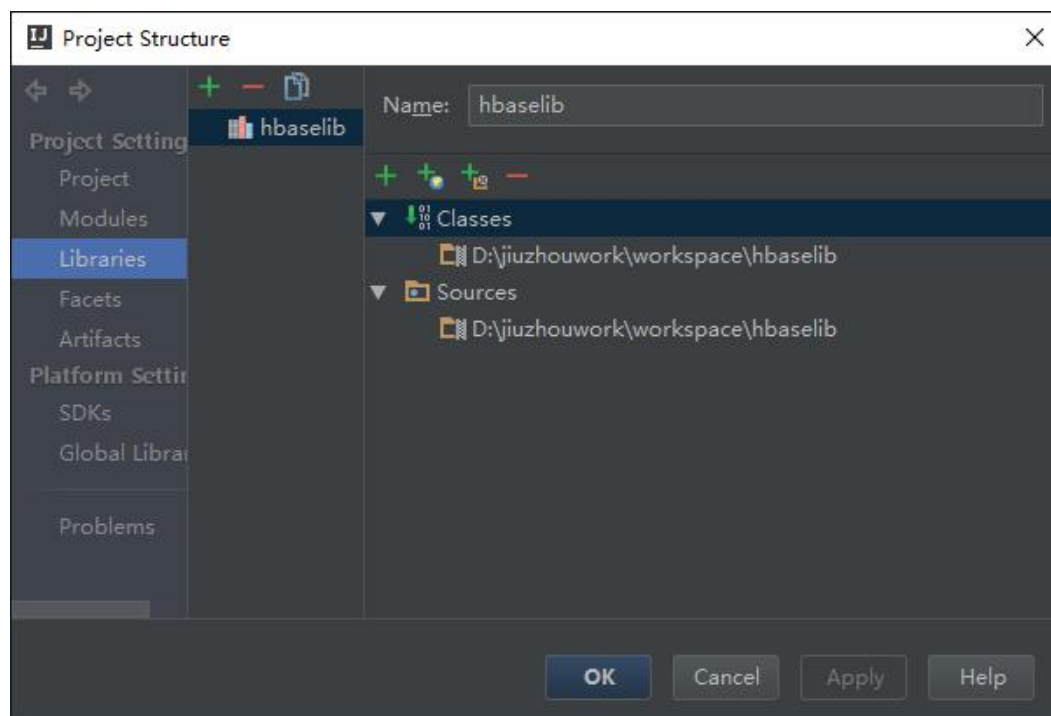
运行窗口字体：settings->editor->colors&Fonts->console Font



b) 方法

与 eclipse 相似,intellij idea 设计的更加人性化,要开发 HBASE 应用就在 project Structure 将 hbaselib 下的所有包加入到 libraries, 这里 hbaselib 是我把 Hbase lib 下的所有包都复制过来了然后建了个 hbaselib 文件夹放在下面。

加入 **libraries** 设置这个就相当于把依赖包路径加入到了编译命令中，这些依赖包在服务器上会在环境变量 **path** 中找，再在程序指定的依赖包中找，所以只要找到了就能愉快的运行了。

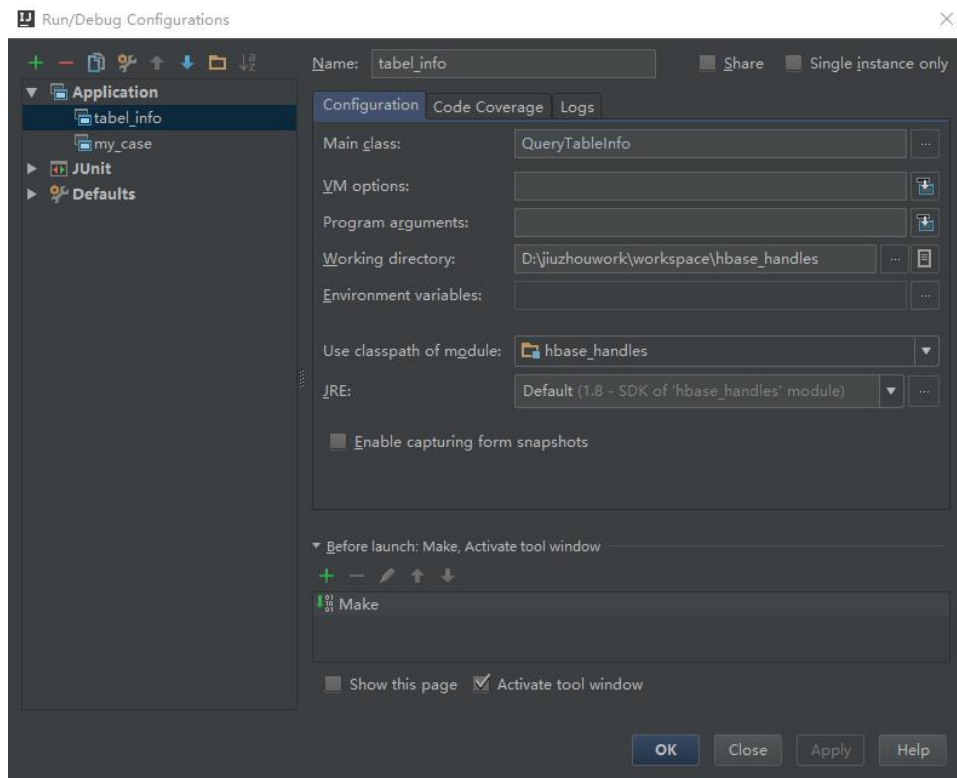


c) 遇到错误

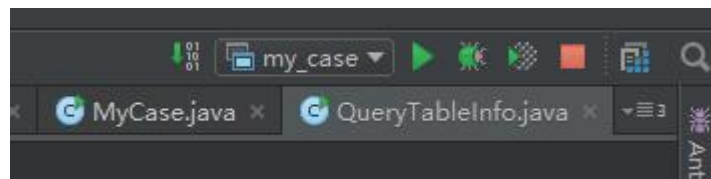
Error1:0 test class found in package '<default package>'

解决: Run->configuration,主要设置入口函数,不过 eclipse 不用设置啊,点到哪个运行哪个,好神奇,idea 可以这样吗?

不过可以配置多个应用入口,点击+号, application。



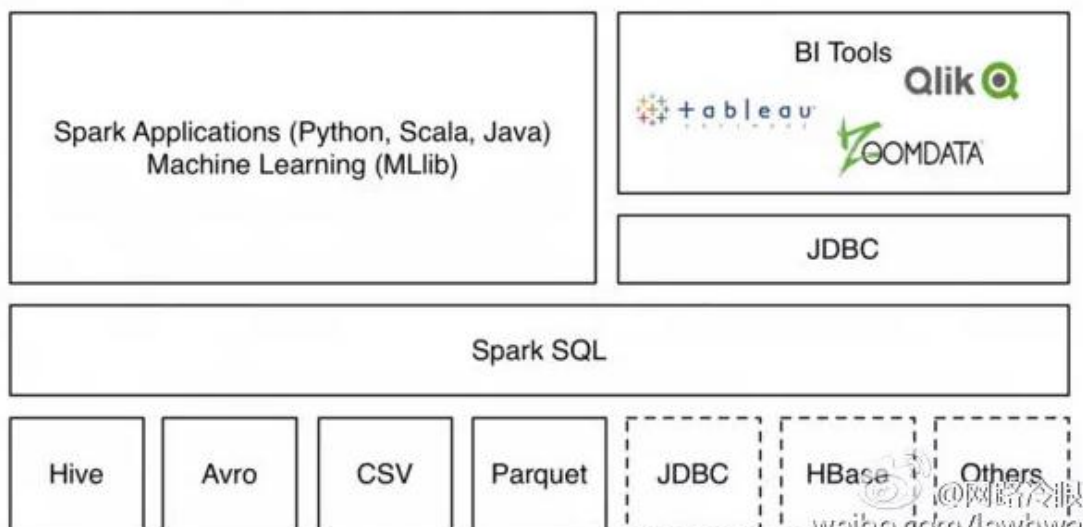
然后，我们在运行时可以选择相应的以不同 class 为入口的应用



11、数据接入

<http://blog.csdn.net/woshiwanxin102213/article/details/17584043>

<http://www.csdn.net/article/2015-02-13/2823955?ref=myread>



Hive: 是建立在 Hadoop 上的数据仓库基础构架。所有 Hive 数据都存储在 Hadoop 兼容的文件系统中（例：Amazon S3、HDFS）中。Hive 在加载数据过程中不会对数据进行任何修改，只是将数据移动到 HDFS 中 Hive 设定目录下，因此，Hive 不支持对数据的改写和添加。

HBase: 分布式面向列的开源数据库。不同于一般的关系数据库，它适用于非结构化存储数据库。

JDBC: 关系型数据库方访问 API，可以访问多种数据库

Spark SQL: 摆脱了对 Hive 的依赖，兼容 Hive，可以从 RDD、PARQUET 文件、JSON 文件中获取数据。

http://www.cnblogs.com/shishanyuan/p/4723604.html?utm_source=tuicool

<http://www.csdn.net/article/2015-04-03/2824407>

<http://www.cnblogs.com/gaopeng527/p/4315808.html>

外部数据源API增强



Formats and Sources supported by DataFrames

内部:

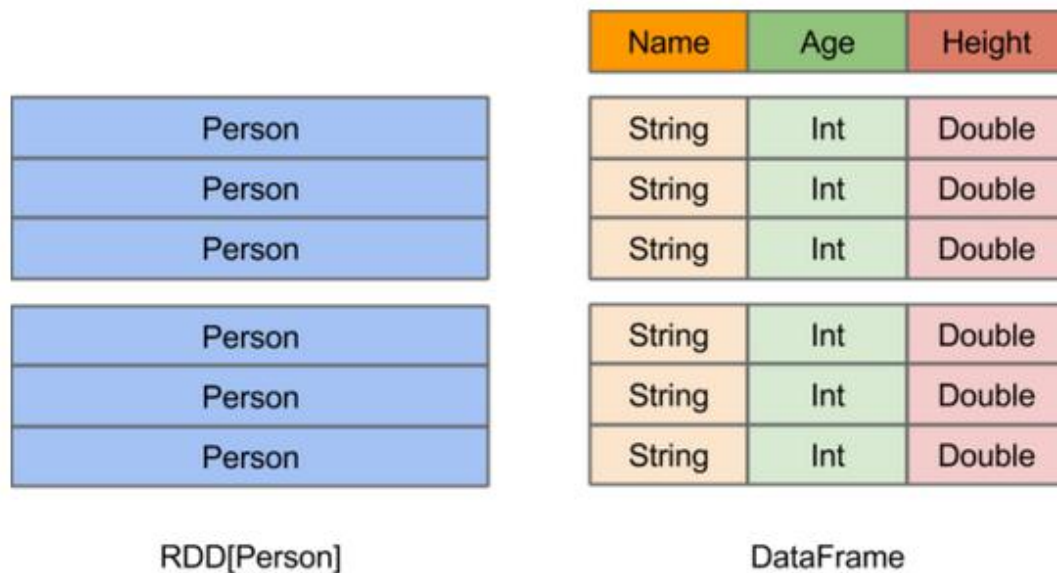
<http://developer.51cto.com/art/201603/507668.htm>

RDD: (Resilient Distributed Datasets)弹性分布式数据集

DataFrame: 是一种以 RDD 为基础的分布式数据集，类似于传统数据库中的二维表格，被用于 SQLContext 相关操作。

DataSet: Dataset 可以认为是 DataFrame 的一个特例，主要区别是 Dataset 每一个 record 存储的是一个强类型值而不是一个 Row。

在Spark中，DataFrame是一种以RDD为基础的分布式数据集，类似于传统数据库中的二维表格。DataFrame与RDD的主要区别在于，前者带有schema元信息，即DataFrame所表示的二维表数据集的每一列都带有名称和类型。这使得Spark SQL得以洞察更多的结构信息，从而对藏于DataFrame背后的数据源以及作用于DataFrame之上的变换进行了针对性的优化，最终达到大幅提升运行时效率的目标。反观RDD，由于无从得知所存数据元素的具体内部结构，Spark Core只能在stage层面进行简单、通用的流水线优化。



a) Spark sql 连接 mysql

- i. 下载 mysql-connector-java-5.1.39-bin.jar
<http://www.itcast.cn/news/20151229/16012088060.shtml>

12、clush

顾名思义：cluster shell, 用于集群上服务及状态的查询

clush -a -b -c /etc/yum.repos.d/*

-a: 所有节点 all

-b: 相同输出结果合并 combine

-w: 指定节点 which

执行结构:

Clush+参数+linux 命令+参数

意义:

将 linux 命令发送到集群的各个主机，得到相应信息返回到 clush 服务整理输出，各主机通过 ssh 相关联

Eg: clush -a -b -c jps

Jps: 查看 java 进程, 当前用户的 java 进程

查看本地集群信息, 可以看到有这么多的应用任务。

HBase 的服务应该是在 namenode 上为 HRegionserver, HRegionServer 主要负责响应用户 I/O 请求, 向 HDFS 文件系统中读写数据, 是 HBase 中最核心的模块。

在 Master 上应用为 HMaster

HMaster 的作用:

- 为 Region server 分配 region

- 负责 Region server 的负载均衡

- 发现失效的 Region server 并重新分配其上的 region

- HDFS 上的垃圾文件回收

- 处理 schema 更新请求

HRegionServer 作用:

- 维护 master 分配给他的 region, 处理对这些 region 的 io 请求

- 负责切分正在运行过程中变的过大的 region

可以看到, client 访问 hbase 上的数据并不需要 master 参与 (寻址访问 zookeeper 和 region server, 数据读写访问 region server), master 仅仅维护 table 和 region 的元数据信息 (table 的元数据信息保存在 zookeeper 上), 负载很低。

HRegionServer 存取一个子表时, 会创建一个 HRegion 对象, 然后对表的每个列族创建一个 Store 实例, 每个 Store 都会有一个 MemStore 和 0 个或多个 StoreFile 与之对应, 每个 StoreFile 都会对应一个 HFile, HFile 就是实际的存储文件。因此, 一个 HRegion 有多少个列族就有多少个 Store。

- 一个 HRegionServer 会有多个 HRegion 和一个 HLog。


```

[root@master bin]# clush -a -b jps
hadoop.node1: ssh: connect to host hadoop.node1 port 22: No route to host
-----
hadoop.node2
-----
28099 QuorumPeerMain
4265 DataNode
26126 HRegionServer
9449 CoarseGrainedExecutorBackend
4399 NodeManager
3485 Jps
-----
hadoop.node3
-----
29008 NodeManager
28878 DataNode
30142 ApplicationMaster
19919 HRegionServer
18812 Jps
30228 CoarseGrainedExecutorBackend
26481 QuorumPeerMain
-----
master
-----
13989 ResourceManager
4764 Bootstrap
28375 QuorumPeerMain
13609 NameNode
28747 Jps
3872 AmbariServer
13785 SecondaryNameNode
clush: hadoop.node1: exited with exit code 255
[root@master bin]#

```

13、HBase

a) HBase web 控制端

默认端口 16010: <http://10.3.9.135:16010/master-status>

也可以自己设, eg:

<http://www.cnblogs.com/captainlucky/p/4710642.html>

```

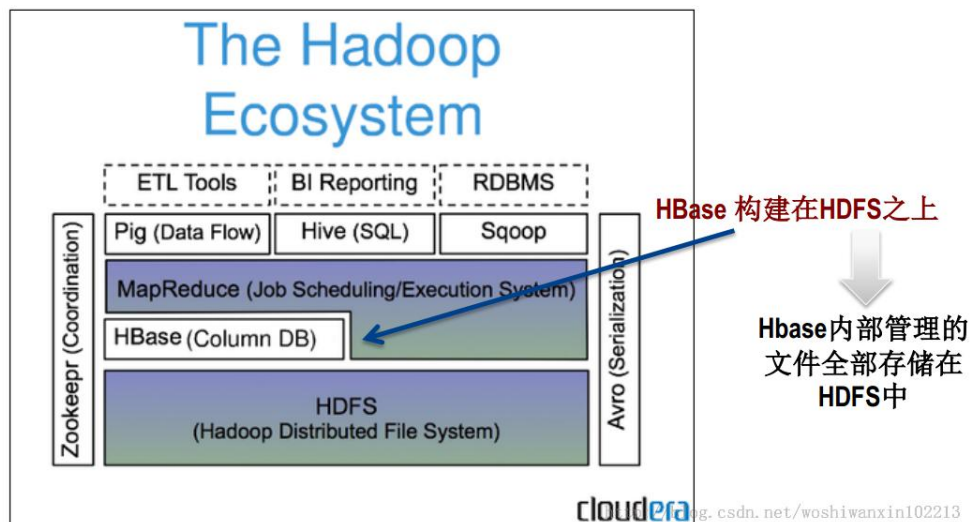
<property>
<name>hbase.rootdir</name>
<value>hdfs://appcluster/hbase</value>
<description>The directory shared by RegionServers.</description>
</property>

```

b) 简介

<http://blog.csdn.net/woshiwanxin102213/article/details/17584043>

HBase 是种大数据数据库，类似于 google, bigTable。其使用和用其实数据库一样，可以用过其提供的端口进行操作，可以在 eclipse 中开发应用连接 HBase 并进行相关操作。



c) Hbase 表的特点

大：一个表可以有数十亿行，上百万列；

无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一张表中不同的行可以有截然不同的列；

面向列：面向列（族）的存储和权限控制，列（族）独立检索；

稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；

数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元格插入时的时间戳；

数据类型单一：Hbase 中的数据都是字符串，没有类型。

d) Hbase 基本概念

RowKey: 是 Byte array, 是表中每条记录的“主键”，方便快速查找，Rowkey 的设计非常重要。

Column Family: 列族，拥有一个名称(string), 包含一个或者多个相关列

Column: 属于某一个 columnfamily, familyName:columnName, 每条记录可动态添加

Version Number: 类型为 Long, 默认值是系统时间戳，可由用户自定义

Value(Cell): Byte array

e) Hbase 物理模型

每个 column family 存储在 HDFS 上的一个单独文件中，空值不会被保存。

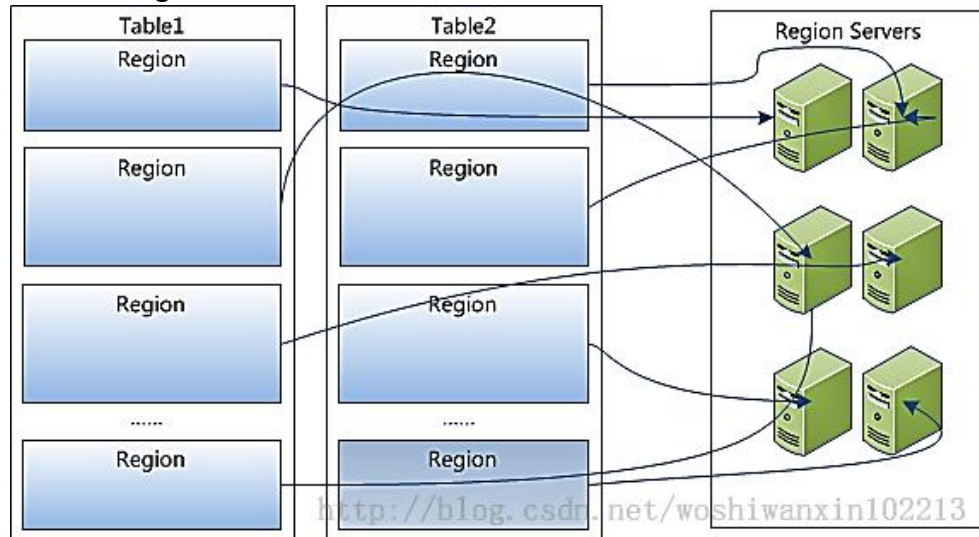
Key 和 Version number 在每个 column family 中均有一份；

HBase 为每个值维护了多级索引，即：<key, column family, column name,

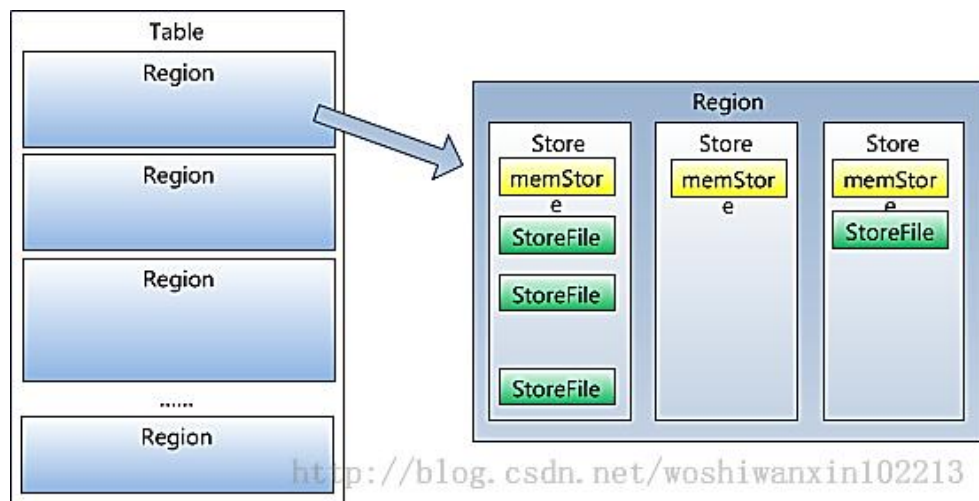
timestamp>

物理存储

- 1、Table 中所有行都按照 row key 的字典序排列；
- 2、Table 在行的方向上分割为多个 Region；
- 3、Region 按大小分割的，每个表开始只有一个 region，随着数据增多，region 不断增大，当增大到一个阈值的时候，region 就会等分会两个新的 region，之后会有越来越多的 region；
- 4、Region 是 Hbase 中分布式存储和负载均衡的最小单元，不同 Region 分布到不同 RegionServer 上。



- 5、Region 虽然是分布式存储的最小单元，但并不是存储的最小单元。Region 由一个或者多个 Store 组成，每个 store 保存一个 columns family；每个 Store 又由一个 memStore 和 0 至多个 StoreFile 组成，StoreFile 包含 HFile；memStore 存储在内存中，StoreFile 存储在 HDFS 上。



f) Hbase shell

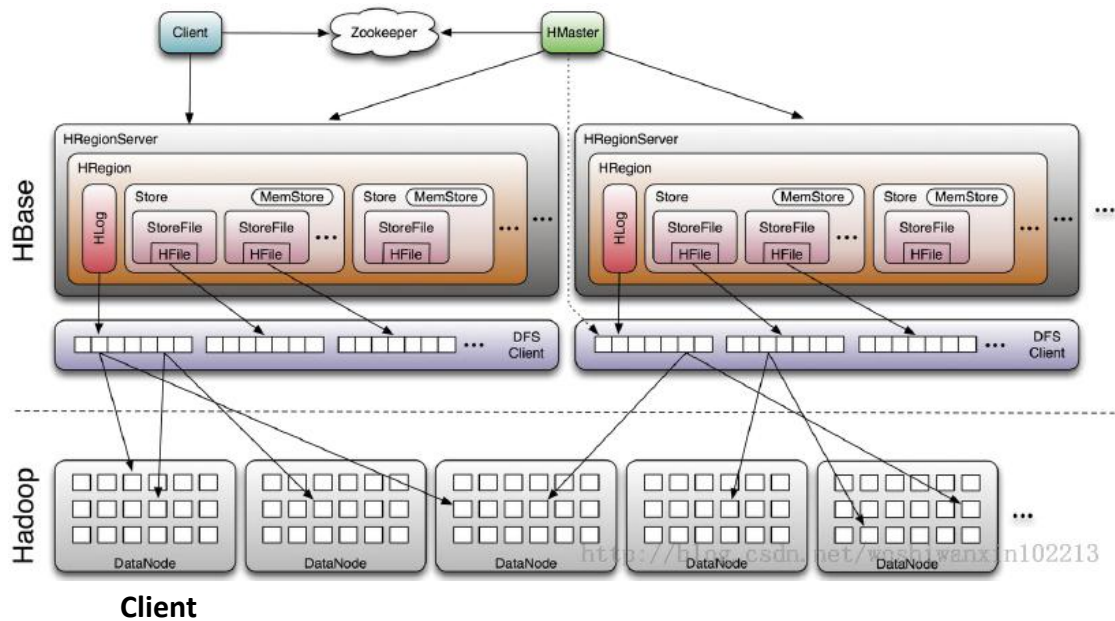
./hbase shell 进入 hbase shell.

<http://www.cnblogs.com/heyCoding/archive/2012/11/09/2762334.html>

scan 'CityWeather', {VERSION => 10}

注意 VERSION 与 VERSIONS 的区别，一个是指定版本，一个是列出几个版本。

g) HBase 架构及基本组件



包含访问 HBase 的接口，并维护 cache 来加快对 HBase 的访问，比如 region 的位置信息

Master

为 Region server 分配 region

负责 Region server 的负载均衡

发现失效的 Region server 并重新分配其上的 region

管理用户对 table 的增删改查操作

Region Server

Regionserver 维护 region，处理对这些 region 的 IO 请求

Regionserver 负责切分在运行过程中变得过大的 region

Zookeeper 作用

通过选举，保证任何时候，集群中只有一个 master，Master 与

RegionServers 启动时会向 ZooKeeper 注册

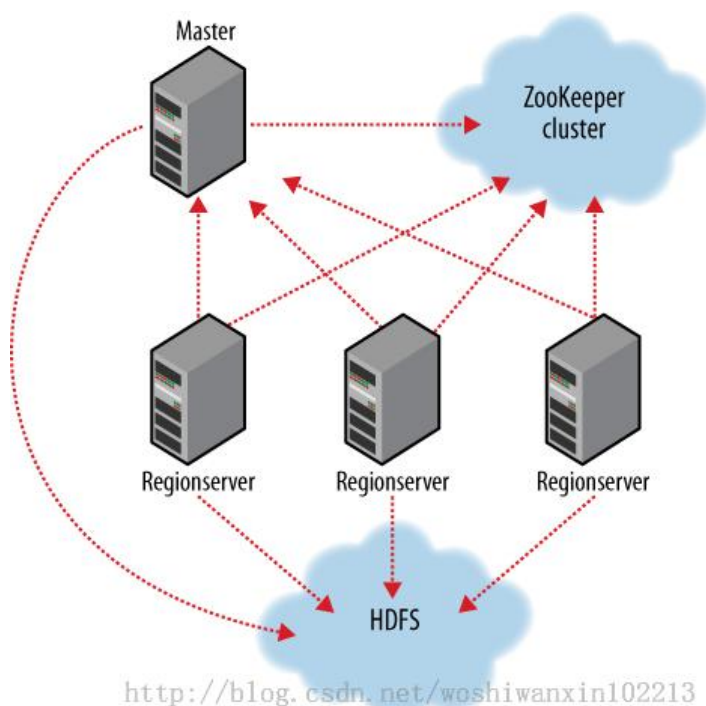
存储所有 Region 的寻址入口

实时监控 Region server 的上线和下线信息。并实时通知给 Master

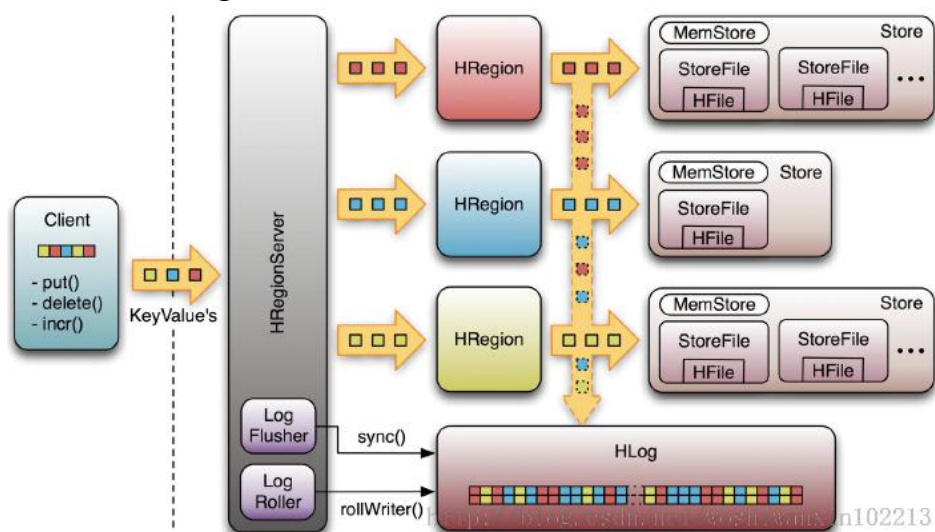
存储 HBase 的 schema 和 table 元数据

默认情况下，HBase 管理 ZooKeeper 实例，比如，启动或者停止 ZooKeeper

Zookeeper 的引入使得 Master 不再是单点故障



Write-Ahead-Log (WAL)



该机制用于数据的容错和恢复:

每个 HRegionServer 中都有一个 HLog 对象, HLog 是一个实现 Write Ahead Log 的类, 在每次用户操作写入 MemStore 的同时, 也会写一份数据到 HLog 文件中 (HLog 文件格式见后续), HLog 文件定期会滚动出新的, 并删除旧的文件 (已持久化到 StoreFile 中的数据)。当 HRegionServer 意外终止后, HMaster 会通过 Zookeeper 感知到, HMaster 首先会处理遗留的 HLog 文件, 将其中不同 Region 的 Log 数据进行拆分, 分别放到相应 region 的目录下, 然后再将失效的 region 重新分配, 领取 到这些 region 的 HRegionServer 在 Load Region 的过程中, 会发现历史 HLog 需要处理, 因此会 Replay HLog 中的数据到 MemStore 中, 然后 flush 到 StoreFiles, 完成数据恢复

h) HBase 容错性

Master 容错: Zookeeper 重新选择一个新的 Master

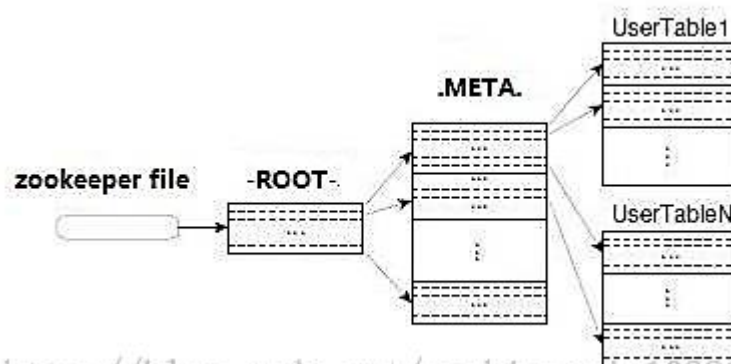
无 Master 过程中, 数据读取仍照常进行;

无 master 过程中, region 切分、负载均衡等无法进行;

RegionServer 容错: 定时向 Zookeeper 汇报心跳, 如果一旦时间内未出现心跳, Master 将该 RegionServer 上的 Region 重新分配到其他 RegionServer 上, 失效服务器上“预写”日志由主服务器进行分割并派送给新的 RegionServer

Zookeeper 容错: Zookeeper 是一个可靠地服务, 一般配置 3 或 5 个 Zookeeper 实例

Region 定位流程:



<http://blog.csdn.net/woshiwanxin102213>

找 RegionServer

Zookeeper--> -ROOT-(单 Region)--> .META.--> 用户表

-ROOT-

表包含.META.表所在的 region 列表, 该表只会有一个 Region;

Zookeeper 中记录了-ROOT-表的 location。

.META.

表包含所有的用户空间 region 列表, 以及 RegionServer 的服务器地址。

i) Hbase 使用场景

storing large amounts of data(100s of TBs)
need high write throughput
need efficient random access(key lookups) within large data sets
need to scale gracefully with data
for structured and semi-structured data
don't need fullRDMS capabilities(cross row/cross table transaction, joins,etc.)

大数据量存储, 大数据量高并发操作

需要对数据随机读写操作

读写访问均是非常简单的操作

Hbase 与 HDFS 对比

两者都具有良好的容错性和扩展性, 都可以扩展到成百上千个节点;

HDFS 适合批处理场景

不支持数据随机查找

不适合增量数据处理

不支持数据更新

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

<http://blog.csdn.net/woshiwanxin102213>

j) HBase 高可用性配置

一个分布式运行的 Hbase 依赖一个 zookeeper 集群, 所有的节点和客户端都必须能够访问 zookeeper。默认的情况下 Hbase 会管理一个 zookeeper 集群。这个集群会随着 Hbase 的启动而启动。当然, 你也可以自己管理一个 zookeeper 集群, 但需要配置 Hbase。你需要修改 HBASE_MANAGES_ZK 来切换。这个值默认是 true 的, 作用是让 Hbase 启动的时候同时也启动 zookeeper。

应该可以不用管 HBASE_MANAGES_ZK，按上面所说，这个为 true 就是为了当重启的时候也重启 zookeeper。

<http://www.tuicool.com/articles/Af6vYb>

<http://www.cnblogs.com/captainlucky/p/4710642.html>

HMaster 没有单点问题，HBase 中可以启动多个 HMaster，通过 Zookeeper 的 Master Election 机制保证总有一个 Master 运行。

所以这里要配置 HBase 高可用的话，只需要启动两个 HMaster，让 Zookeeper 自己去选择一个 Master Active。

命令：hbase-daemon.sh start master

检测：通过 kill 现在 active Hmaster，当执行程序时 zookeeper 就会调用另一个 standby Hmaster 使它 active。

可以通过端口 16010 查看，每台主机节点都有

Eg:<http://10.3.9.231:16010/master-status>

Start Time	Description	State	Status
Wed Aug 17 15:04:17 CST 2016	Master startup	RUNNING (since 12mins, 11sec ago)	Another master is the active master, master,16000,1471417175590; (since 12mins, 11sec ago)

表明这台机器上的 HMaster 是 standby 的。

当我 kill 10.3.9.231 的 HMaster 后

```
13609 NameNode
[root@master bin]# jps
3872 AmbariServer
25460 Jps
13989 ResourceManager
24262 HMaster
28375 QuorumPeerMain
13785 SecondaryNameNode
13609 NameNode
[root@master bin]# kill 24262
[root@master bin]#
```

[Show Client Operations](#)
[View as JSON](#)

Start Time	Description	State	Status
Wed Aug 17 15:04:17 CST 2016	Master startup	RUNNING (since 15mins, 9sec ago)	Another master is the active master, master,16000,1471417175590; v (since 15mins, 9sec ago)

其变成的 active HMaster

至于怎么在日志中看到，我还不清楚。

k) HBase 日志

http://hbase.apache.org/book.html#_configuration_files

i. 配置 logs,在安装目录 conf 下的 hbase-env.sh

```
# export HBASE_BACKUP_MASTERS=${HBASE_HOME}/conf/backup-masters

# Extra ssh options. Empty by default.
# export HBASE_SSH_OPTS="-o ConnectTimeout=1 -o SendEnv=HBASE_CONF_DIR"

# Where log files are stored. $HBASE_HOME/logs by default.
export HBASE_LOG_DIR=/home/data/$USER/hbase/logs

# Enable remote JDWP debugging of major HBase processes. Meant for Core Developers
# export HBASE_MASTER_OPTS="$HBASE_MASTER_OPTS -Xdebug -Xrunjdpw:transport=dt_socket,server=y,transport=dt_socket,interface=localhost,serverport=5005"
# export HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS -Xdebug -Xrunjdpw:transport=dt_socket,server=y,transport=dt_socket,interface=localhost,serverport=5005"
```

env 里面设置的是程序启动前的一些配置，日志是程序没跑起来都需要的。

- ii. 在 HBase 的安装目录下的 conf 的 hbase-site.xml 里面有配置，hbase.tmp.dir，hbase 运行临时目录。

hbase-site.xml

The main HBase configuration file. This file specifies configuration options which override HBase's default configuration. You can view (but do not edit) the default configuration file at *docs/hbase-default.xml*. You can also view the entire effective configuration for your cluster (defaults and overrides) in the HBase Configuration tab of the HBase Web UI.

Hbase-default.xml

<https://github.com/apache/hbase/blob/master/hbase-common/src/main/resources/hbase-default.xml>

```
<property>
  <name>hbase.tmp.dir</name>
  <value>${java.io.tmpdir}/hbase-${user.name}</value>
  <description>Temporary directory on the local filesystem.
  Change this setting to point to a location more permanent
  than '/tmp', the usual resolve for java.io.tmpdir, as the
  '/tmp' directory is cleared on machine restart.</description>
</property>
<property>
  <name>hbase.rootdir</name>
  <value>${hbase.tmp.dir}/hbase</value>
  <description>The directory shared by region servers and into
  which HBase persists. The URL should be 'fully-qualified'
  to include the filesystem scheme. For example, to specify the
  HDFS directory '/hbase' where the HDFS instance's namenode is
  running at namenode.example.org on port 9000, set this value to:
  hdfs://namenode.example.org:9000/hbase. By default, we write
  to whatever ${hbase.tmp.dir} is set too -- usually /tmp --
  so change this configuration or else all data will be lost on
  machine restart.</description>
</property>
```



```

<property>

<!--
/**
 *表示标识符号{},标识要获取{}中的值,其中user.name应该是从某个容器中得到值
http://www.cnblogs.com/mlloc-clove/p/3550498.html
EL 存取变量数据的方法很简单,例如:${username}。
它的意思是取出某一范围中名称为username的变量。
因为我们并没有指定哪一个范围的username,所以它的默认值会先从Page 范围找,
假如找不到,再依序到Request、Session、Application范围。(pageScope、requestScope、sessionScope、
applicationScope)
假如途中找到username,就直接回传,不再继续找下去,
但是假如全部的范围都没有找到时,就回传null,
当然EL表达式还会做出优化,页面上显示空白,而不是打印输出NULL。

这里user.name是linux用户的name,例: root什么的。可以在linux的/home/data/用户名/hbase/下看到
本地文件系统的临时文件夹。可以修改到一个更为持久的目录上。( /tmp会在重启时清楚)
默认: /tmp/hbase-${user.name}
*/
-->
<name>hbase.tmp.dir</name>
<value>/home/data/${user.name}/hbase</value>
</property>
</property>

```

```

48      <name>hbase.tmp.dir</name>
49      <value>/home/data/${user.name}/hbase</value>
50    </property>
51  </property>
52    <name>hbase.zookeeper.quorum</name>
53    <value>master,hadoop.node1,hadoop.node2,hadoop.node3</value>
54  </property>
55  <property>
56    <name>hbase.zookeeper.property.clientPort</name>
57    <value>2181</value>
58  </property>
59  <property>
60    <name>hbase.zookeeper.property.dataDir</name>
61    <value>/home/data/zookeeper</value>
62  </property>
63  <property>
64    <!-- HBase的运行模式。false是单机模式, true是分布式模式。若为false,HBase和Z
65    默认: false -->
66    <name>hbase.cluster.distributed</name>
67    <value>true</value>
68  </property>
69  <property>
70    <name>zookeeper.session.timeout</name>
71    <value>120000</value>
72  </property>
73  <property>
74    <name>fs.hdfs.impl</name>
75    <value>org.apache.hadoop.hdfs.DistributedFileSystem</value>
76  </property>
77 </configuration>
78

```

- iii.
- iv. 在 HBase 的安装文件夹下通过 `find ./ -name "*log*"` 找到

```

./docs/images/big_h_logo.svg
./docs/images/hbase_logo.svg
./docs/images/hbase_logo.png
./docs/xref/org/apache/hadoop/hbase/master/CatalogJanitor.html
./docs/xref/org/apache/hadoop/hbase/http/log
./docs/_chapters/images/hadoop-logo.jpg
./docs/_chapters/images/hbase_logo_with_orca.xcf
./docs/_chapters/images/hbase_logo_with_orca.png
./docs/_chapters/images/big_h_logo.png
./docs/_chapters/images/big_h_logo.svg
./docs/_chapters/images/hbase_logo.svg
./docs/_chapters/images/hbase_logo.png
./docs/xref-test/org/apache/hadoop/hbase/master/TestCatalogJanitor.html
./docs/xref-test/org/apache/hadoop/hbase/http/log
./docs/apidocs/org/apache/hadoop/hbase/http/log
./lib/slf4j-log4j12-1.7.5.jar
./lib/log4j-1.2.17.jar
./lib/commons-logging-1.2.jar
./lib/ruby/shell/commands/catalogjanitor_run.rb
./lib/ruby/shell/commands/catalogjanitor_enabled.rb
./lib/ruby/shell/commands/catalogjanitor_switch.rb
./conf/log4j.properties
./hbase-webapps/static/hbase_logo_med.gif
./hbase-webapps/static/hbase_logo_small.png
./hbase-webapps/static/hbase_logo.png
[root@master hbase-1.1.3]#

```

可以看到这里有 log4j 的配置，也就是我们在运行 Hbase 相关程序时，它会通过这个配置来打印与操作我们的信息。至于它是否保存在哪或者怎么设置还不太清楚。

这个 log4j.properties 可以放在我们的项目的资源文件夹下，这样当我们运行程序时会根据配置打印相关内容，方便我们查看

```

# Define some default values that can be overridden by system p
hbase.root.logger=INFO,console
hbase.security.logger=INFO,console
hbase.log.dir=.
hbase.log.file=hbase.log

# Define the root logger to the system property "hbase.root.log
log4j.rootLogger=${hbase.root.logger}

# Logging Threshold
log4j.threshold=ALL

#
# Daily Rolling File Appender
#
log4j.appender.DRFA=org.apache.log4j.DailyRollingFileAppender
log4j.appender.DRFA.File=${hbase.log.dir}/${hbase.log.file}

```

上面是部分配置信息，可以看到日志是通过 INFO，控制台输出的

```

+  tabel.info  tabel.info
+  [C:\Program Files\Java\jdk1.8.0_74\bin\java" ...
+  执行完毕1
+  执行完毕2
+  2016-08-17 14:30:52,952 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x4cf4d528 connecting to ZooKeeper
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=HP110
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8_0_74
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=C:\Program Files\Java\jdk1.8.0_74\jre
+  2016-08-17 14:30:52,976 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=C:\Program Files\Java\jdk1.8.0_74\lib
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:java.library.path=C:\Program Files\Java\jdk1.8.0_74\bin
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:java.io.tmpdir=C:\Users\lw_co\AppData\Local\Temp\
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:os.name=Windows 10
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:os.arch=amd64
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:os.version=10.0
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:user.name=lw_co
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:user.home=C:\Users\lw_co
+  2016-08-17 14:30:52,979 INFO [main] zookeeper.ZooKeeper: Client environment:user.dir=D:\jiuzhouwork\workspace\hbase_handles
+  2016-08-17 14:30:52,981 INFO [main] zookeeper.ZooKeeper: Initiating client connection, connectString=master:2181,hadoop.node1:2181,h
+  2016-08-17 14:30:53,189 INFO [main-SendThread(hadoop.node2:2181)] zookeeper.ClientCnxn: Opening socket connection to server hadoop.n
+  2016-08-17 14:30:53,193 INFO [main-SendThread(hadoop.node2:2181)] zookeeper.ClientCnxn: Socket connection established to hadoop.node
+  2016-08-17 14:30:53,370 INFO [main-SendThread(hadoop.node2:2181)] zookeeper.ClientCnxn: Session establishment complete on server had
+  [Log: apache.hadoop.hbase.HTableDescriptor:@60129b9a
+  执行完毕4
+  Process finished with exit code 0
```

上面是程序运行时输出和 Log 输出

I) ERROR

Bin 文件下的命令

```

[root@master bin]# ls
draining_servers.rb      hbase-daemons.sh        rolling-restart.sh
get-active-master.rb     hbase-jruby              shutdown_regionserver.rb
graceful_stop.sh         hirb.rb                  start-hbase.cmd
hbase                    local-master-backup.sh   start-hbase.sh
hbase-cleanup.sh         local-regionserver.sh    stop-hbase.cmd
hbase.cmd                master-backup.sh         stop-hbase.sh
hbase-common.sh          region_mover.rb          test
hbase-config.cmd         regionserver.sh          thread-pool.rb
hbase-config.sh          region_status.rb         zookeepers.sh
hbase-daemon.sh          replication
[root@master bin]#
```

ERROR:org.apache.hadoop.hbase.PleaseHoldException
Resolution:Please stop hbase on your cluster first. And restart them with
certain sequences: first regionserver on all nodes, then hmaster.
运行 stop-hbase.sh 然后运行 start-hbase.sh

m) 编程

Hbase 客户端 API 中，对 HBASE 的任何操作都需要首先创建 HbaseConfiguration 类的实例，为 HBaseConfiguration 类继承自 Configuration 类，而 Configuration 类属于 Hadoop 核心包中实现的类，该类的主要作用是提供对配置参数的访问途径。

i. Java 连接 HBase

<http://my.oschina.net/u/160697/blog/516362>

Eg1:

```
private static HBaseConfiguration hbaseConfig=null;
static {
    Configuration config=new Configuration();
    config.set("hbase.zookeeper.quorum","192.168.1.98");
    config.set("hbase.zookeeper.property.clientPort", "2181");
    hbaseConfig = new HBaseConfiguration(config);
}
```

Eg2:

```
Configuration configuration = HBaseConfiguration.create();

configuration.set("hbase.zookeeper.property.clientPort",
"2181"); //设置 zookeeper client 端口
configuration.set("hbase.zookeeper.quorum", "192.168.1.19");
// 设置 zookeeper quorum

configuration.addResource("/usr/local/hbase-1.0.1.1/conf/hbase-site.xml");
//将 hbase 的配置加载
```

```
configuration.set(TableInputFormat.INPUT_TABLE,
"heartSocket");
```

在调用 HBaseConfiguration.create()方法时，HBASE 首先会在 classpath 下寻找 hbase-site.xml 文件，将里面的信息解析出来封装到 Configuration 对象中，如果 hbase-site.xml 文件不存在，则使用默认的 hbase-core.xml 文件。

除了将 hbase-site.xml 放到 classpath 下，开发人员还可通过 config.set(name, value)方法来手工构建 Configuration 对象：

```
Configuration.set(String name, String value);
```

ii. 相关操作

http://blog.csdn.net/javaman_chen/article/details/7220216

创建表：HBaseAdmin admin = new HBaseAdmin(conf);

HTable 封装表格对象，进行增删改查：

```
HTable table=new HTable(config,tableName);
```

Get,put,Delete,Scan 操作，每个方法对应相应的操作对象

Eg:Delete delete=new Delete();

```
Table.delete(delete);
```

Hbase-site.xml 参数详解

<http://greatwqs.iteye.com/blog/1837178>

iii. 表的结构

<http://blog.csdn.net/woshisap/article/details/43777135>

有多个列族：family

每个列族可以有一个或多个列成员，每个列成员
同一个列族存在同一个目录下，写操作锁行的，每一行是一个原子元素，都可以加锁。
映射：行键、行键+时间戳或行键+列
稀疏存储某些列可以空白

概念视图来看每个表格是由很多行组成的，但是在物理存储上面，它是按照列来保存的。
也就是说一个行关键字对应的信息由多的列族来各自保存。

"<family>:<qualifier>"

Family 为列族，比如有两列，列的字段和数量是提前指定好的不能改变，qualifier 限定修饰符是可以随意加的。
定位某一单元格由行键，列族：限定符，时间戳唯一决定。

其实就是相当于 mysql 中的表格结构，row key 就相当于主键，Family 就是相当于某列，多个 family 就是多列，family 是事先指定的，family 就是列簇，因为它可以包含多个 qualifier 就是限定符。限定符可以任意增加，然后存的时候，考虑的到是稀疏的所以按列 Family 存放，即如果某个 rowkey 有多个列 family 属性，那么每个行的分开存，不存为空的值。

<http://blog.csdn.net/dbanote/article/details/8904003>

HBase以表的形式存储数据。表由行和列族组成。列划分为若干个列族(row family)，其逻辑视图如下：

行 键	时间戳	列族 contents	列族 anchor	列族 mime
"com.cnn.www"	t9		anchor:cnnsi.com="CNN"	
	t8		anchor:my.look.ca="CNN.com"	
	t6	contents:html="<html>..."		mime:type="text/html"
	t5	contents:html="<html>..."		
	t3	contents:html="<html>..."		

单元格由行键，列族：限定符，时间戳唯一决定。
Cell 中数据是没有类型的，全部以字节码形式存储

Row Key	Time Stamp	Column Family:c1		Column Family:c2	
		列	值	列	值
r1	t7	c1:1	value1-1/1		
	t6	c1:2	value1-1/2		
	t5	c1:3	value1-1/3		
	t4			c2:1	value1-2/1
	t3			c2:2	value1-2/2
t2	t2	c1:1	value2-1/1		
	t1			c2:1	value2-1/1

Row Key	Time Stamp	Column Family:c1	
		列	值
r1	t7	c1:1	value1-1/1
	t6	c1:2	value1-1/2
	t5	c1:3	value1-1/3

表：HBase数据的物理视图（1）

Row Key	Time Stamp	Column Family:c2	
		列	值
r1	t4	c2:1	value1-2/1
	t3	c2:2	value1-2/2

表：HBase数据的物理视图（2）

14、算法运行及结果

a) 运行样例:

Run-example:

```
./bin/run-example mllib.JavaKMeans mllibTestData/kmeans_data.txt 2 100
```

mllibTestData 为 HDFS 上的文件

如果类名不对，它会 not Found，如果参数不够它会提示输入参数

Job:

```
./bin/spark-submit --class org.apache.spark.examples.mllib.JavaKMeans  
--master yarn --deploy-mode cluster ./lib/spark-examples-1.6.1-hadoop2.7.1.jar  
mllibTestData/kmeans_data.txt 3 20
```

如果参数不够: exitCode:-1000

如果类名不对: exitCode:10

b) linear regression (线性回归)

i. JavaHdfsLR <file> <iters>

javaHdfsLR<文件><迭代次数>

本地:

```
./bin/spark-submit --class org.apache.spark.examples.JavaHdfsLR  
--master local ./lib/spark-examples-1.6.1-hadoop2.7.1.jar  
mllibTestData/lr_data.txt 1
```

集群:

```
./bin/spark-submit --class org.apache.spark.examples.JavaHdfsLR  
--master yarn --deploy-mode  
cluster ./lib/spark-examples-1.6.1-hadoop2.7.1.jar  
mllibTestData/lr_data.txt 1
```

结果:

LogType:stdout

Log Upload Time:星期一 七月 25 16:46:12 +0800 2016

LogLength:442

Log Contents:

Initial w: [0.4551273600657362, 0.36644694351969087,
-0.38256108933468047, -0.4458430198517267, 0.33109790358914726,
0.8067445293443565, -0.2624341731773887, -0.44850386111659524,
-0.07269284838169332, 0.5658035575800715]

On iteration 1

Final w: [246.25860765580322, 270.70869288178557,
371.354007739464, 357.4478152969409, 261.9494718512335,
210.01734831542458, 366.7061915626602, 381.34754796597383,
335.20416843810943, 240.24079035615807]
End of LogType:stdout

ii. JavaLR <input_dir> <step_size> <niters>

JavaLR<输入文件><步长大小><迭代次数>
./bin/spark-submit \
--class org.apache.spark.examples.mllib.JavaLR \
--master yarn \
--deploy-mode cluster \
./lib/spark-examples-1.6.1-hadoop2.7.1.jar \
mllibTestData/lr-data/random.data 1 10
结果:
LogType:stdout
Log Upload Time:星期一 七月 25 17:03:14 +0800 2016
LogLength:47
Log Contents:
Final w: [0.6262094779767464,0.535113798652265]End of
LogType:stdout