Day5 3月3日

软件51 庞建业 2151601012

配置Hadoop小记

首先环境准备，linux版本：CentOS

但JDK一定要用Oracle SUN官方的版本，请从官网下载，操作系统的自带的OpenJDK会有各种不兼容。
JDK：jdk-8u161-linux-x64.tar.gz

完全分步式的Hadoop集群，这个选择3台一样配置的虚拟机，通过内网的一个DNS服务器，指定3台虚拟机所对应的域名。
每台虚拟机，2G内存，硬盘20G。hadoop会存储在外接硬盘上面。

设置SSH自动登陆，通过ssh-keygen命令，生成id_rsa.pub，再合并到 authorized_keys的文件。再通过scp把authorized_keys复制到其他的虚拟机。循环生成authorized_keys并合并文件。使得3台虚拟机，都有了相互的SSH自动登陆的配置。

```
mkdir /hadoopDev
上传或下载 JDK
tar zxvf jdk-8u161-linux-x64.tar.gz
mv jdk1.8.0_161/ /jdk1.8
设置Java环境变量：
Vi /etc/profile

export JAVA_HOME=/jdk1.8
export PATH=$PATH:$JAVA_HOME/bin

source /etc/profile  //环境变量生效
验证：
[root@hmaster hadoopDev]# echo $JAVA_HOME
/jdk1.8
[root@hmaster hadoopDev]# java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)

第一次分配IP后，设置网络地址为静态：
vi /etc/sysconfig/network-scripts/ifcfg-ens33

TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
#BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no

IPV6INIT=yes
```

```
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=ac3279f6-ff27-4ac0-9fc4-1a1203b73bb9
DEVICE=ens33
ONBOOT=yes
IPADDR=192.168.31.131
NETMASK=255.255.255.0
GATEWAY=192.168.31.2
DNS1=114.114.114.114
```

克隆
启动slave1，修改IP地址为192.168.31.131
启动slave2，修改IP地址为192.168.31.132

配置ssh及密码
登录Hmaster：
ssh-keygen -t rsa　一直回车：
生成authorized_keys.
cat id_rsa.pub >> authorized_keys
在salve1和slave2上执行同样的命令。
拷贝slave1和slave2到 hmaster上：
Slave1上执行： scp /root/.ssh/id_rsa.pub root@hmaster:/root/.ssh/id_rsa_slave1.pub
Slave2上执行： scp /root/.ssh/id_rsa.pub root@hmaster:/root/.ssh/id_rsa_slave2.pub
切换到hmaster机器
ssh hamster ,追加slave1和slave2的公钥到授权文件：
cat id_rsa_slave1.pub >> authorized_keys
cat id_rsa_slave2.pub >> authorized_keys
将授权文件拷贝到slave1和slave2,并做无密码登录验证

安装Hadoop
解压
```
 tar zxvf Hadoop-3.0.0.tar.gz
```
移动到根目录下
```
mv hadoop-3.0.0  /hadoop
```
创建hdfs目录/hadoop/hdfs　及子目录name \data\tmp：
```
[root@hmaster /]# mkdir /hadoop/hdfs
[root@hmaster /]# cd /hadoop/hdfs/
[root@hmaster hdfs]# mkdir name
[root@hmaster hdfs]# mkdir data
[root@hmaster hdfs]# mkdir tmp
```

Hadoop配置/hadoop/etc/hadoop目录下：
```
vi core-site.xml
<configuration>
<configuration>
 <property>
     <name>fs.defaultFS</name>
          <value>hdfs://hmaster:9000</value>
```

```
      </property>
  <property>
        <name>hadoop.tmp.dir</name>
              <value>/hadoop/hdfs/tmp</value>
   </property>
</configuration>
</configuration>
```
修改hdfs-site.xml，集群机器数少1
```
vi hdfs-site.xml
<configuration>
 <property>
    <name>dfs.replication</name>
       <value>2</value>
 </property>
 <property>
    <name>dfs.namenode.name.dir</name>
    <value>/hadoop/hdfs/name</value>
 </property>
 <property>
     <name>dfs.datanode.data.dir</name>
     <value>/hadoop/hdfs/data</value>
 </property>
</configuration>
```
修改workers，hmaster节点也加上，否则master节点没有dataNode
```
vi workers
hmaster
slave1
slave2
```
修改：mapred-site.xml
```
vi mapred-site.xml
<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
  </property>
<property>
 <name>mapreduce.application.classpath</name>
  <value>
    /hadoop/etc/hadoop,
    /hadoop/share/hadoop/common/*,
    /hadoop/share/hadoop/common/lib/*,
    /hadoop/share/hadoop/hdfs/*,
    /hadoop/share/hadoop/hdfs/lib/*,
    /hadoop/share/hadoop/mapreduce/*,
    /hadoop/share/hadoop/mapreduce/lib/*,
    /hadoop/share/hadoop/yarn/*,
    /hadoop/share/hadoop/yarn/lib/*
   </value>
</property>
</configuration>
```
编辑yarn-site.xml
```
<configuration>

<property>
```

```
      <name>yarn.nodemanager.aux-services</name>
       <value>mapreduce_shuffle</value>
  </property>
<property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandle</value>
</property>
<property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>hmaster:8025</value>
</property>
<property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>hmaster:8030</value>
</property>
<property>
    <name>yarn.resourcemanager.address</name>
    <value>hmaster:8040</value>
    </property>
</configuration>
编辑hadoop-env.sh中配置java_home
vi hadoop-env.sh
```

```
设置Hadoop系统变量HADOOP_HOME：

export JAVA_HOME=/jdk1.8
export HADOOP_HOME=/hadoop
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin
```

```
格式化namenode
hdfs namenode -format
```

```
启动hdfs 和yarn
/hadoop/sbin/start-all.sh
报错：
ERROR: but there is no HDFS_NAMENODE_USER defined. Aborting operation.
Starting datanodes
ERROR: Attempting to operate on hdfs datanode as root
ERROR: but there is no HDFS_DATANODE_USER defined. Aborting operation.
Starting secondary namenodes [hmaster]
ERROR: Attempting to operate on hdfs secondarynamenode as root
ERROR: but there is no HDFS_SECONDARYNAMENODE_USER defined. Aborting operation.
设置变量，重新编辑vi hadoop.env.sh
export JAVA_HOME=/jdk1.8
export HDFS_NAMENODE_USER=root
export HDFS_DATANODE_USER=root
export HDFS_SECONDARYNAMENODE_USER=root
export YARN_RESOURCEMANAGER_USER=root
export YARN_NODEMANAGER_USER=root
```

```
拷贝Hadoop文件夹到slave1和slave2
```

上面还忘了把/hadoop整个目录拷贝到slave1和slave2
重新进行文件夹远程复制：

```
scp -r /hadoop root@salve1:/
scp -r /hadoop root@salve2:/
```

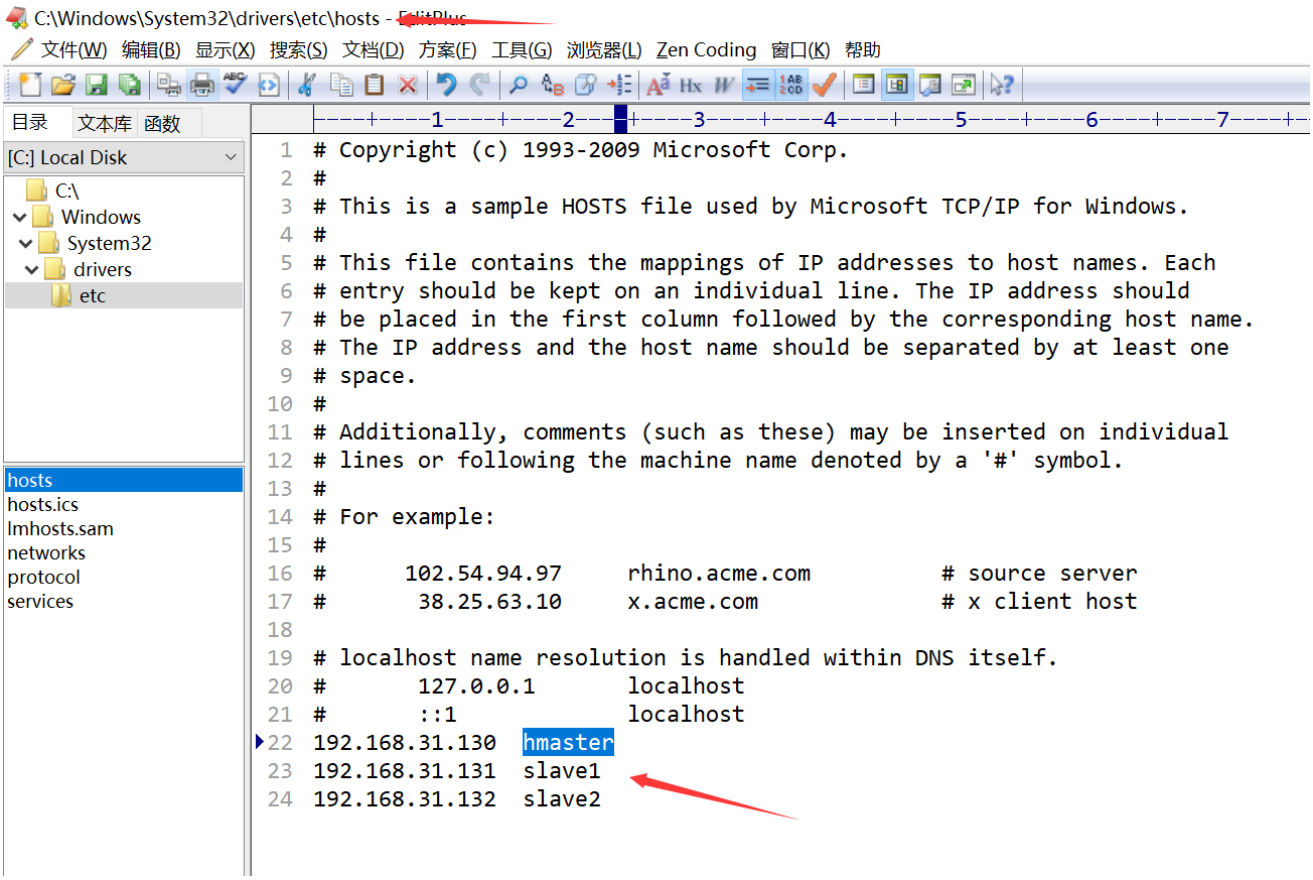
执行完，如果前面有已经启动的进程，先执行

```
/hadoop/sbin/stop-all.sh
```

启动后，用jps命令查看java进程，master上的进程有：

```
[root@hmaster hadoop]# jps
12052 DataNode
12280 SecondaryNameNode
12521 ResourceManager
12954 Jps
11915 NameNode
```

修改windows主机，映射3个虚拟系统机器名



停止防火墙
9870 访问hdfs地址和YARN地址
hamster:9870

运行一个mapreduce程序

```
cd /hadoop/share/hadoop/mapreduce
hadoop jar hadoop-mapreduce-examples-3.0.0.jar pi 2 10
```

```
检查hadoop启动是否成功
1.jps
2.netstat -nl
```

对Hadoop进行学习

推荐算法

推荐算法分类：

按数据使用划分：

协同过滤算法：UserCF, ItemCF, ModelCF 基于内容的推荐: 用户内容属性和物品内容属性 社会化过滤：基于用户的社会网络关系 **按模型划分：**

最近邻模型:基于距离的协同过滤算法 Latent Factor Mode(SVD)：基于矩阵分解的模型 Graph：图模型，社会网络图模型

Hadoop编程调用HDFS

```
系统环境
ls操作
rmr操作
mkdir操作
copyFromLocal操作
cat操作
copyToLocal操作
创建一个新文件，并写入内容
```

```
Hadooop命令行：java FsShell
```

```
新建文件：HdfsDAO.java，用来调用HDFS的API。

public class HdfsDAO {

    //HDFS访问地址
    private static final String HDFS = "hdfs://192.168.1.210:9000/";

    public HdfsDAO(Configuration conf) {
        this(HDFS, conf);
    }

    public HdfsDAO(String hdfs, Configuration conf) {
        this.hdfsPath = hdfs;
        this.conf = conf;

    }
```

```
    //hdfs路径
    private String hdfsPath;
    //Hadoop系统配置
    private Configuration conf;

    //启动函数
    public static void main(String[] args) throws IOException {
        JobConf conf = config();
        HdfsDAO hdfs = new HdfsDAO(conf);
        hdfs.mkdirs("/tmp/new/two");
        hdfs.ls("/tmp/new");
    }

    //加载Hadoop配置文件
    public static JobConf config(){
        JobConf conf = new JobConf(HdfsDAO.class);
        conf.setJobName("HdfsDAO");
        conf.addResource("classpath:/hadoop/core-site.xml");
        conf.addResource("classpath:/hadoop/hdfs-site.xml");
        conf.addResource("classpath:/hadoop/mapred-site.xml");
        return conf;
    }

    //API实现
    public void cat(String remoteFile) throws IOException {...}
    public void mkdirs(String folder) throws IOException {...}

    ...
}
```

ls操作

```
 public void ls(String folder) throws IOException {
        Path path = new Path(folder);
        FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);
        FileStatus[] list = fs.listStatus(path);
        System.out.println("ls: " + folder);
        System.out.println("==========================================================");
        for (FileStatus f : list) {
            System.out.printf("name: %s, folder: %s, size: %d\n", f.getPath(), f.isDir(),
f.getLen());
        }
        System.out.println("==========================================================");
        fs.close();
    }


    public static void main(String[] args) throws IOException {
```

```
        JobConf conf = config();
        HdfsDAO hdfs = new HdfsDAO(conf);
        hdfs.ls("/");
    }
```

mkdir操作

```
public void mkdirs(String folder) throws IOException {
        Path path = new Path(folder);
        FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);
        if (!fs.exists(path)) {
            fs.mkdirs(path);
            System.out.println("Create: " + folder);
        }
        fs.close();
    }

    public static void main(String[] args) throws IOException {
        JobConf conf = config();
        HdfsDAO hdfs = new HdfsDAO(conf);
        hdfs.mkdirs("/tmp/new/two");
        hdfs.ls("/tmp/new");
    }
```

rmr操作

```
  public void rmr(String folder) throws IOException {
        Path path = new Path(folder);
        FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);
        fs.deleteOnExit(path);
        System.out.println("Delete: " + folder);
        fs.close();
    }

    public static void main(String[] args) throws IOException {
        JobConf conf = config();
        HdfsDAO hdfs = new HdfsDAO(conf);
        hdfs.rmr("/tmp/new/two");
        hdfs.ls("/tmp/new");
    }
```

```
copyFromLocal操作
```

```
public void copyFile(String local, String remote) throws IOException {
        FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);
        fs.copyFromLocalFile(new Path(local), new Path(remote));
        System.out.println("copy from: " + local + " to " + remote);
        fs.close();
    }

    public static void main(String[] args) throws IOException {
        JobConf conf = config();
        HdfsDAO hdfs = new HdfsDAO(conf);
        hdfs.copyFile("datafile/randomData.csv", "/tmp/new");
        hdfs.ls("/tmp/new");
    }
```

对Spring Boot进行学习

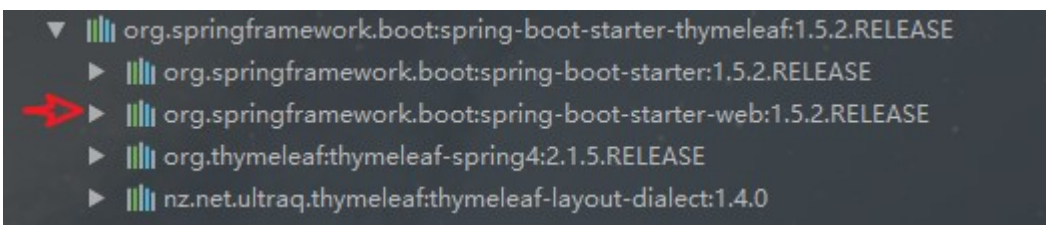
Thymeleaf

```
Spring Boot支持多种模版引擎包括:
FreeMarker
Groovy
Thymeleaf(官方推荐)
Mustache
```

Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎。类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。与其它模板引擎相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用。它的功能特性如下：

- Spring MVC中@Controller中的方法可以直接返回模板名称，接下来Thymeleaf模板引擎会自动进行渲染
- 模板中的表达式支持Spring表达式语言（Spring EL)
- 表单支持，并兼容Spring MVC的数据绑定与验证机制

```
引入依赖
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```




```
编写controller
```

```
@Controller
@RequestMapping("/learn")
public class LearnResourceController {
    @RequestMapping("/")
    public ModelAndView index(){
        List<LearnResouce> learnList =new ArrayList<LearnResouce>();
        LearnResouce bean =new LearnResouce("官方参考文档","Spring Boot Reference
Guide","http://docs.spring.io/spring-boot/docs/1.5.1.RELEASE/reference/htmlsingle/#getting-
started-first-application");
        learnList.add(bean);
        ModelAndView modelAndView = new ModelAndView("/index");
        modelAndView.addObject("learnList", learnList);
        return modelAndView;
    }
}
```

编写html
引入依赖后就在默认的模板路径src/main/resources/templates下编写模板文件

Thymeleaf的默认参数配置

application.properties中可以配置thymeleaf模板解析器属性

```
# THYMELEAF (ThymeleafAutoConfiguration)
#开启模板缓存（默认值：true）
spring.thymeleaf.cache=true
#Check that the template exists before rendering it.
spring.thymeleaf.check-template=true
#检查模板位置是否正确（默认值:true）
spring.thymeleaf.check-template-location=true
#Content-Type的值（默认值：text/html）
spring.thymeleaf.content-type=text/html
#开启MVC Thymeleaf视图解析（默认值：true）
spring.thymeleaf.enabled=true
#模板编码
spring.thymeleaf.encoding=UTF-8
#要被排除在解析之外的视图名称列表，用逗号分隔
spring.thymeleaf.excluded-view-names=
#要运用于模板之上的模板模式。另见StandardTemplate-ModeHandlers(默认值：HTML5)
spring.thymeleaf.mode=HTML5
#在构建URL时添加到视图名称前的前缀（默认值：classpath:/templates/）
spring.thymeleaf.prefix=classpath:/templates/
#在构建URL时添加到视图名称后的后缀（默认值：.html）
spring.thymeleaf.suffix=.html
#Thymeleaf模板解析器在解析器链中的顺序。默认情况下，它排第一位。顺序从1开始，只有在定义了额外的
TemplateResolver Bean时才需要设置这个属性。
spring.thymeleaf.template-resolver-order=
#可解析的视图名称列表，用逗号分隔
spring.thymeleaf.view-names=
```