课程内容

使用java开发spark 实战

一：环境搭建 安装jdk 和maven。

1. 安装jdk并配置环境变量

系统变量→新建 JAVA\_HOME 变量 。

变量值填写jdk的安装目录（本人是 E:\Java\jdk1.7.0)

系统变量→寻找 Path 变量→编辑

在变量值最后输入 %JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）

系统变量→新建 CLASSPATH 变量值填写 .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar（注意最前面有一点）

1. Maven的安装和配置

解压apache-maven-3.1.1-bin.zip，并把解压后的文件夹下的apache-maven-3.1.1文件夹移动到D:\Java下，如果没有Java这个文件夹的话，请自行创建

新建系统变量 MAVEN\_HOME 变量值：D:\Java\apache-maven-3.1.1。编辑系统变量 Path 添加变量值： ;%MAVEN\_HOME%\bin。

在mave 的目录中修改conf/settings.xml， 在 localRepository属性后添加<localRepository>D:/repository</localRepository> 修改maven下载jar 的位置。

1. eclipse 中java 和maven 的配置

点击 window ->java ->Installed JREs ->add ->standard vm ,点击next ，然后选择jdk 的安装路径点击finish即可。

点击window ->Maven ->Installations ->add 在弹出页面选择mave 的安装路径，然后点击finish。然后在列表中选择我们自己刚添加的那个maven信息。

然后点击window ->Maven ->User Setings 在 右侧的User Settings 点击browse 现在maven conf目录下的setttings.xml .（主要是修改maven下载依赖包存放的位置）

二：创建maven项目

1. 创建maven项目

点击file ­->new ->others ->maven project 点击next，选择maven-archetype-quickstart ，点击next，group id 为 com.dt.spark，artifact id 为 sparkApps，然后点击finish。

1. 修改jdk 和pom文件

创建maven项目后，默认的jdk 是1.5要改成我们前面安装好的jdk1.8 。在项目上右击 build path ->configure build path 。在弹出页面点击Libraries，选中jre system library 。点击 edit，在弹出框选择 workspace default jre ，然后点击finish。 然后在点击ok。将pom文件修改为如下内容，然后等待eclipse 下载好maven依赖的jar包，并编译工程。编译好工程后有个错误提示，在此错误列上，右击选择 quick fix ，在弹出页面点击finish即可。

<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.dt.spark</groupId>

<artifactId>SparkApps</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SparkApps</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-hive\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>2.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-graphx\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<testSourceDirectory>src/main/test</testSourceDirectory>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<maniClass></maniClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.3.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>false</includeProjectDependencies>

<classpathScope>compile</classpathScope>

<mainClass>com.dt.spark.SparkApps.WordCount</mainClass>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

1. 创建 包路径以及java代码

在包路径 com.dt.spark.SparkApps 上 右击 new ->package 在弹出页面name中填写com.dt.spark.SparkApps.cores,点击finish的。

在包路径下com.dt.spark.SparkApps.cores上 右击 new ->class ，在弹出窗口中name 中填写 WordCount ，点击finish。然后在 WordCount 中编写如下代码。

**package** com.dt.spark.SparkApps.cores;

**import** java.util.Arrays;

**import** org.apache.spark.SparkConf;

**import** org.apache.spark.api.java.JavaPairRDD;

**import** org.apache.spark.api.java.JavaRDD;

**import** org.apache.spark.api.java.JavaSparkContext;

**import** org.apache.spark.api.java.function.FlatMapFunction;

**import** org.apache.spark.api.java.function.Function2;

**import** org.apache.spark.api.java.function.PairFunction;

**import** org.apache.spark.api.java.function.VoidFunction;

**import** scala.Function;

**import** scala.Tuple2;

/\*\*

\* 使用java的方式开发进行本地测试spark的 wordcount 程序

\* **@author** DT大数据梦工厂

\*

\*/

**public** **class** WordCount {

**public** **static** **void** main(String[] args) {

/\*\*

\* 第1步：创建Spark的配置对象SparkConf，设置Spark程序的运行时的配置信息，

\* 例如说通过setMaster来设置程序要链接的Spark集群的Master的URL,如果设置

\* 为local，则代表Spark程序在本地运行，特别适合于机器配置条件非常差（例如

\* 只有1G的内存）的初学者 \*

\*/

SparkConf conf =**new** SparkConf().setAppName("Spark WordCount written by java").setMaster("local");

/\*\*

\* 第2步：创建SparkContext对象

\* SparkContext是Spark程序所有功能的唯一入口，无论是采用Scala、Java、Python、R等都必须有一个SparkContext(不同的语言具体的类名称不同，如果是java 的为javaSparkContext)

\* SparkContext核心作用：初始化Spark应用程序运行所需要的核心组件，包括DAGScheduler、TaskScheduler、SchedulerBackend

\* 同时还会负责Spark程序往Master注册程序等

\* SparkContext是整个Spark应用程序中最为至关重要的一个对象

\*/

JavaSparkContext sc=**new** JavaSparkContext(conf); //其底层就是scala的sparkcontext

/\*\*

\* 第3步：根据具体的数据来源（HDFS、HBase、Local FS、DB、S3等）通过SparkContext来创建RDD

\* JavaRDD的创建基本有三种方式：根据外部的数据来源（例如HDFS）、根据Scala集合、由其它的RDD操作

\* 数据会被JavaRDD划分成为一系列的Partitions，分配到每个Partition的数据属于一个Task的处理范畴

\*/

JavaRDD<String> lines =sc.textFile("D://spark-1.6.0-bin-hadoop2.6//README.md");

/\*\*

\* 第4步：对初始的JavaRDD进行Transformation级别的处理，例如map、filter等高阶函数等的编程，来进行具体的数据计算

\* 第4.1步：讲每一行的字符串拆分成单个的单词

\*/

JavaRDD<String> words = lines.flatMap(**new** FlatMapFunction<String,String>(){ //如果是scala由于Sam转化所以可以写成一行代码

@Override

**public** Iterable<String> call(String line) **throws** Exception {

// **TODO** Auto-generated method stub

**return** Arrays.*asList*(line.split(" "));

}

});

/\*\*

\* 第4步：对初始的JavaRDD进行Transformation级别的处理，例如map、filter等高阶函数等的编程，来进行具体的数据计算

\* 第4.2步：在单词拆分的基础上对每个单词实例计数为1，也就是word => (word, 1)

\*/

JavaPairRDD<String,Integer> pairs=words.mapToPair(**new** PairFunction<String, String, Integer>() {

@Override

**public** Tuple2<String, Integer> call(String word) **throws** Exception {

// **TODO** Auto-generated method stub

**return** **new** Tuple2<String, Integer>(word,1);

}

});

/\*\*

\* 第4步：对初始的RDD进行Transformation级别的处理，例如map、filter等高阶函数等的编程，来进行具体的数据计算

\* 第4.3步：在每个单词实例计数为1基础之上统计每个单词在文件中出现的总次数

\*/

JavaPairRDD<String,Integer> wordsCount =pairs.reduceByKey(**new** Function2<Integer, Integer, Integer>() { //对相同的Key，进行Value的累计（包括Local和Reducer级别同时Reduce）

@Override

**public** Integer call(Integer v1, Integer v2) **throws** Exception {

// **TODO** Auto-generated method stub

**return** v1+v2;

}

});

wordsCount.foreach(**new** VoidFunction<Tuple2<String,Integer>>() {

@Override

**public** **void** call(Tuple2<String, Integer> pairs) **throws** Exception {

// **TODO** Auto-generated method stub

System.***out***.println(pairs.\_1+" : " +pairs.\_2);

}

});

sc.close();

}

}

在代码区右击 run as -> java application 。来运行此程序并查看运行结果。如果要开发cluster 的代码，请参考前面第8课或者第9课。