## 综述

每一个 flink项目都是一个单独的普通maven版本的java项目，注意这是一个单独的项目并没有使用spring,springmvc,springboot等技术。

## 组件

在flink项目中有以下术语需要了解

### Environmnet

Environmnet 环境用来获取flink的运行环境，所有flink程序必须有环境

如

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

### Source

source数据源

数据源用来为计算做准备。任何计算都需要数据源

如

DataStreamSource<String> text = env.socketTextStream("10.0.192.31", port, "\n");

可以自定义一个数据源(这个没测试应当是这样写)

public class TestSourceFromPostgresql extends RichSourceFunction<String> {

public void open(Configuration parameters) throws Exception {

System.out.println("==================open11============");

}

public void close() throws Exception {

System.out.println("==================close============");

}

public void cancel() {

}

public void run(SourceContext<String> ctx) throws Exception {

String ssa="产生的字符串";

ctx.collect(ssa);//发送结果

}

}

### Transform

transform提供运算操作方案如

#### Map

输入一个参数产生一个参数，map 的功能是对输入的参数进行转换操作。  
val streamMap = stream.map { x => x \* 2 }

Flatmap

输入一个参数，产生 0、1 或者多个输出，这个多用于拆分操作。  
val streamFlatMap = stream.flatMap{  
x => x.split(" ")  
}

#### **Filter**

结算每个元素的布尔值，并返回为 true 的元素。  
val streamFilter = stream.filter{  
x => x == 1  
}

#### **Keyby**

DataStream → KeyedStream：输入必须是Tuple类型，逻辑地将一个流拆分成不相交的分区，每个分区包含具有相同key的元素，在内部以hash的形式实现的。  
注意：以下类型无法作为key。

POJO类，且没有实现 hashCode 函数

任意形式的数组类型

#### **Distinct**

去重

#### **Join or outjoin**

关联

#### **Cross**

求笛卡尔积

Reduce

滚动合并操作，合并当前元素和上一次合并的元素结果

//求各个渠道的累计个数

val value: DataStream[(Int, Int)] = env.fromElements((1, 2), (1, 3))

val kst: KeyedStream[(Int, Int), Tuple] = value.keyBy(0)

kst.reduce { (t1, t2) => (t1.\_1, t1.\_2 + t2.\_2) }.print().setParallelism(1)

env.execute()

#### **Fold**

用一个初始的一个值，与其每个元素进行滚动合并操作。

private def myFold(env: StreamExecutionEnvironment): Unit = {

val value: DataStream[(Int, Int)] = env.fromElements((1, 2), (1, 3))

val kst: KeyedStream[(Int, Int), Tuple] = value.keyBy(0)

val ds: DataStream[String] = kst.fold("")((str, i) => {

str + “-” + i

})

ds.print()

env.execute()

}

复杂的方法

aggregation

      KeyedStream --> DataStream：分组流数据的滚动聚合操作： min 和 minBy 的区别是 min 返回的是一个最小值，而 minBy 返回的是其字段中包含的最小值元素（同样原理使用与 max 和 maxBy）。

window

      KeyedStream --> DataStream：windows 是在一个分区的 KeyedStream 中定义的，windows 根据某些特性将每个 key 的数据进行分组（例如：在 5s 内到达的数据）。

windowAll

      DataStream --> AllWindowedStream：Windows 可以在一个常规的 DataStream 中定义，Windows 根据某些特性对所有的流（例如：5s内到达的数据）。这个操作在很多情况下都不是并行操作的，所有的记录都会聚集到一个 windowAll 的操作任务中。

window apply

      WindowedStream --> DataStream,AllWindowedStream --> DataStream：将一个通用的函数作为一个整体传递给window

window reduce

       WindowedStream --> DataStream：给窗口赋予一个reduce的功能，并返回一个reduce的结果。

window fold

       WindowedStream --> DataStream：给窗口赋予一个fold的功能，并返回一个fold后的结果

aggregation on windows

       WindowedStream --> DataStream：对 window 的元素做聚合操作，min 和 minBy 的区别是 min 返回的是最小值，而 minBy 返回的是包含最小值字段的元素。（同样原理适用于 max 和 maxBy ）

union

      DataStream --> DataStream：对两个或两个以上的 DataStream 做 union 操作，产生一个包含所有的 DataStream 元素的新 DataStream 。注意：如果将一个 DataStream 和自己做union操作，在新的 DataStream 中，将看到每个元素重复两次

private def myUnion(env: StreamExecutionEnvironment): Unit = {

//myConnAndCoMap(env)

val dsm: DataStream[Int] = env.fromElements(1, 3, 5)

val dsm01: DataStream[Int] = env.fromElements(2, 4, 6)

val unit: DataStream[Int] = dsm.union(dsm01)

unit.print()

env.execute()

}

window join

      DataStream，DataStream --> DataStream：根据给定的 key 和 window 对两个 DataStream 做 join 操作

window coGroup

      DataStream，DataStream --> DataStream：根据一个给定的 key 和 window 对两个 DataStream 做 CoGroups 操作

connect

      DataStream，DataStream --> ConnectedStreams：连接两个保持她们类型的数据流。

12. coMap 、coFlatMap

      ConnectedStreams --> DataStream：作用于 connected 数据流上，功能与 map 和 flatMap 一样

//合并以后打印

private def myConnAndCoMap(env: StreamExecutionEnvironment): Unit = {

env.setParallelism(1)

val src: DataStream[Int] = env.fromElements(1, 3, 5)

val stringMap: DataStream[String] = src.map(line => "x " + line)

val result = stringMap.connect(src).map(new CoMapFunction[String, Int, String] {

override def map2(value: Int): String = {

"x " + (value + 1)

}

override def map1(value: String): String = {

value

}

})

result.print()

env.execute()

split

      DataStream --> SplitStream：根据某些特征把一个 DataStream 拆分成两个或多个 DataStream

select

      SplitStream --> DataStream：从一个 SplitStream 中获取一个或多个 DataStream

private def selectAndSplit(env: StreamExecutionEnvironment): Unit = {

val dsm: DataStream[Long] = env.fromElements(1l, 2l, 3l, 4l)

val split:SplitStream[Long] = dsm.split(new OutputSelector[Long] {

override def select(out: Long): lang.Iterable[String] = {

val list = new util.ArrayList[String]()

if (out % 2 == 0) {

list.add("even")

} else {

list.add("odd")

}

list

}

})

split.select("odd").print().setParallelism(1)

env.execute()

}

iterate

      DataStream --> IterativeStream --> DataStream：在流程中创建一个反馈循环，将一个操作的输出重定向到之前的操作，这对于定义持续更新模型的算法来说很有意义的。

extract timestamps

      DataStream --> DataStream：提取记录中的时间戳来跟需要事件时间的 window 一起发挥作用.

### **Sink**

sink数据计算结果，也就是将计算结果 写入到哪儿去，如写入到文本，写入到控制台，写入到redis等

程序默认自带的写入目标有

Kafka sink

Redis sink

Elasticsearch sink

自定义sink将计算结果写入自己需要的地方如http请求

package com.vsked.flink;

import org.apache.flink.api.java.tuple.Tuple2;

import org.apache.flink.configuration.Configuration;

import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;

public class HttpSink extends RichSinkFunction<Tuple2<String, Integer>>{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public void open(Configuration parameters) throws Exception {

System.out.println("==================open11============");

}

public void close() throws Exception {

System.out.println("==================close============");

}

public void invoke(Tuple2<String, Integer> value) throws Exception {

System.out.println("result:"+value);

//在这里通过http接口发送给服务端

//服务端收到数据以后把消息送到websocket

}

}

## 完整示例

本示例实现了从kafka读数据

统计每个单词出现次数

把计算结果写入http 请求

//实际计算类

package com.vsked.flink;

import java.util.Properties;

import org.apache.flink.api.common.serialization.SimpleStringSchema;

import org.apache.flink.api.java.tuple.Tuple2;

import org.apache.flink.streaming.api.TimeCharacteristic;

import org.apache.flink.streaming.api.datastream.DataStream;

import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class TestFlink3 {

private static final Logger log = LoggerFactory.getLogger(TestFlink3.class);

public static void main(String[] args) {

log.debug("start flink3");

try{

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

env.enableCheckpointing(5000);//非常关键，一定要设置启动检查点!!

env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

Properties props = new Properties();

props.setProperty("bootstrap.servers", "192.168.111.95:9092,192.168.111.96:9092,192.168.111.97:9092");

props.setProperty("group.id", "flink-group");

String topicName="testFlink";

FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<String>(topicName, new SimpleStringSchema(), props);

consumer.setStartFromEarliest(); // start from the earliest record possible

//consumer.setStartFromLatest(); // start from the latest record

//consumer.setStartFromTimestamp(...); // start from specified epoch timestamp (milliseconds)

//consumer.setStartFromGroupOffsets(); // the default behaviour

//第一步从kafka读数据

DataStream<String> stream = env.addSource(consumer);

//第二步计算数据

DataStream<Tuple2<String, Integer>> windowCount = stream.flatMap(new LineSplitter()).keyBy(0).sum(1);

windowCount.addSink(new HttpSink());//第三步设置自定义sink计算结果写入

//注意：因为flink是懒加载的，所以必须调用execute方法，上面的代码才会执行

env.execute("streaming word count");

}catch(Exception e){

log.error(e.getMessage(),e);

log.error("error=========================");

}

log.debug("end flink3");

}

}

//自定义计算结果输出类

package com.vsked.flink;

import org.apache.flink.api.java.tuple.Tuple2;

import org.apache.flink.configuration.Configuration;

import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;

public class HttpSink extends RichSinkFunction<Tuple2<String, Integer>>{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public void open(Configuration parameters) throws Exception {

System.out.println("==================open11============");

}

public void close() throws Exception {

System.out.println("==================close============");

}

public void invoke(Tuple2<String, Integer> value) throws Exception {

System.out.println("result:"+value);

//在这里通过http接口发送给服务端

//服务端收到数据以后把消息送到websocket

//第四步将计算结果发送给websocket

}

}

//数据源计算类

package com.vsked.flink;

import org.apache.flink.api.common.functions.FlatMapFunction;

import org.apache.flink.api.java.tuple.Tuple2;

import org.apache.flink.util.Collector;

public class LineSplitter implements FlatMapFunction<String, Tuple2<String, Integer>> {

private static final long serialVersionUID = 1L;

public void flatMap(String value, Collector<Tuple2<String, Integer>> out) {

String[] tokens = value.toLowerCase().split("\\W+");

for (String token : tokens) {

if (token.length() > 0) {

out.collect(new Tuple2<String, Integer>(token, 1));

}

}

}

}

pom文件示例

<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.vsked</groupId>

<artifactId>flink3</artifactId>

<version>1.0</version>

<name>flink3</name>

<description>flink3</description>

<properties>

<!-- 声明项目配置依赖编码格式为 utf-8 -->

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

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<!-- java jdk 版本声明 可变更 根据自己配置去匹配 -->

<java.version>1.8</java.version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<jackson.version>2.9.8</jackson.version>

<slf4j.version>1.7.29</slf4j.version>

<log4j2.version>2.12.1</log4j2.version>

<!-- 解决pom头部unkonow错误 -->

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>1.1.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.glassfish.web/jstl-impl -->

<dependency>

<groupId>org.glassfish.web</groupId>

<artifactId>jstl-impl</artifactId>

<version>1.2</version>

<exclusions>

<exclusion>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- slf4j start -->

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<!-- slf4j end -->

<!-- log4j2 start -->

<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl -->

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>${log4j2.version}</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>${log4j2.version}</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>${log4j2.version}</version>

</dependency>

<!-- log4j2 end -->

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-java -->

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-java</artifactId>

<version>1.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-java -->

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-streaming-java\_2.12</artifactId>

<version>1.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-clients -->

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-clients\_2.12</artifactId>

<version>1.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-connector-kafka -->

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-connector-kafka\_2.12</artifactId>

<version>1.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.lettuce/lettuce-core -->

<dependency>

<groupId>io.lettuce</groupId>

<artifactId>lettuce-core</artifactId>

<version>5.2.1.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.squareup.okio/okio -->

<dependency>

<groupId>com.squareup.okio</groupId>

<artifactId>okio</artifactId>

<version>2.4.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.squareup.okhttp3/okhttp -->

<dependency>

<groupId>com.squareup.okhttp3</groupId>

<artifactId>okhttp</artifactId>

<version>4.2.2</version>

</dependency>

<dependency>

<groupId>com.squareup.retrofit2</groupId>

<artifactId>retrofit</artifactId>

<version>2.6.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skipTests>true</skipTests><!--默认关掉单元测试 -->

</configuration>

</plugin>

<plugin>

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

<artifactId>maven-jar-plugin</artifactId>

<version>2.6</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>com.vsked.flink.TestFlink3</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

<plugin>

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

<artifactId>maven-dependency-plugin</artifactId>

<version>2.10</version>

<executions>

<execution>

<id>copy-dependencies</id>

<phase>package</phase>

<goals>

<goal>copy-dependencies</goal>

</goals>

<configuration>

<outputDirectory>${project.build.directory}/lib</outputDirectory>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>