Apache Flink程序开发

Apache Flink在分布式数据集上实现多种transformation算子，包括filtering、mapping、updating state、joining、grouping及aggregating等。分布式数据集初始化时一般读取文件、kafka或者内存数据等，获取的结果写到文件或终端打印。Flink的分布式程序运行在Standalome或者其他Framework上，不同的运行模式有不同的Context钟。根据数据源的不同，Flink的程序可以分为批处理程序或者Streaming程序。在Flink中使用DataSet和DataStream代表上述两种程序的数据集合，DataSet是有限数据，DataStream为无边界的流数据。

在Flink中的数据集和Java Collection有一定区别，首先数据集一旦创建就无法add或者remove，同时也无法访问里面的元素，对数据集上的常见操作是根据现有的数据集使用算子生成新的数据集。

# 程序组成

在Flink中程序由以下部分构成：

* 获取一个执行环境
* 加载或者创建数据集
* 在数据集上使用算子
* 指定计算结果的输出方式和位置
* 触发执行的执行，在Spark上由Action触发，Flink通过调用execute触发执行

下面大概介绍每个步骤。

1. **获取执行环境**

在Flink中执行环境由ExecutionEnvironment来描述，可以使用类中的静态方法来获取其成员，如下所示：

*getExecutionEnvironment*

*createLocalEnvironment*

*createRemoteEnvironment(host,port,jarFile)*

getExecutionEnvironment方法或根据执行环境自动创建一个ExeuctionEnvironment，如果在IDE中运行，则返回Local Environment，如果将出现打包成jar后使用命令行提交则返回Cluster Environment。核心的Environment类如下所示：



1. 加载数据集

对于特定的数据源，ExecutionEnvironment有多种方法从文件中读取数据，可以从文件中逐行读取，或者指定format加载格式数据，例如：

*final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();*

*DataStream<String> text = env.readTextFile("file:///path/to/file");*

示例中从文件中读取数据流。

1. 在数据集上使用算子，Map算子的示例如下：

*DataStream<String> input = ...;*

*DataStream<Integer> parsed = input.map(new MapFunction<String, Integer>() {*

*@Override*

*public Integer map(String value) {*

*return Integer.parseInt(value);*

*}*

*});*

将String转换成整数。

1. 结果输出

当DataStream中已经包含了最终结果，通过创建sink将其写入到outside系统中，示例如下：

*writeAsText(String path)*

*print()*

1. 程序提交后，需要触发程序的执行，触发的方法为在StreamExecutionEnvironment上调用execute方法。根据执行的类型，可以提交到local主机或者集群中。

Execute方法的返回结果类型为JobExecutionResult，包括执行时间和计算结果。

# 程序特点

## 2.1 Lazy Evaluation

所有的Flink程序都是延迟执行的，当程序被执行的时候，所有加载的数据和转换算子都没有开始执行，而是每个operation将会创建并且添加到程序Plan中，类似于Spark Stage，当明确调用execute方法时，程序才会真正执行。

使用延迟计算的好处是Flink可以将复杂的程序转换成Plan，作为一个整体单元来执行，针对Plan进行执行的优化。

## 2.2 Specifying Keys

Flink程序中的算子（join,coGgroup,keyBy及groupBy）需要数据集中定义key，其他的算子如reduce,groupReduce,Aggregate,Windows等也是根据key进行分组。

DataSet根据key进行分组的示例：

*DataSet<...> input = // [...]*

*DataSet<...> reduced = input*

*.groupBy(/\*define key here\*/)*

*.reduceGroup(/\*do something\*/);*

DataStream中key的定义如下：

*DataStream<...> input =* // [...]

*DataStream<...> windowed = input*

*.keyBy(*/\*define key here\*/*)*

*.window(*/\*window specification\*/*);*

但是在Flink中数据模型不是基于Key/Value，Key是一个逻辑概念不需要将数据集在物理上格式成keys和values。下面介绍DataStrem API及keyBy的使用

1. 元组key，将一个或者多个字段作为key，示例如下：

*DataStream<Tuple3<Integer,String,Long>> input = // [...]*

*KeyedStream<Tuple3<Integer,String,Long>,Tuple> keyed = input.keyBy(0)*

将元组的第一个字段作为key。下面的示例是多个字段组合成key

*DataStream<Tuple3<Integer,String,Long>> input = // [...]*

*KeyedStream<Tuple3<Integer,String,Long>,Tuple> keyed = input.keyBy(0,1)*

遇到嵌套元组时，如下：

*DataStream<Tuple3<Tuple2<Integer, Float>,String,Long>> ds;*

如果将keyBy(0)作为Key值，Tuple2整个字段作为key

1. 使用字段表达式定义Key

可以使用基于String字段的表达式来引用嵌套字段，这些字段用作grouping,sorting,

joining及coGrouping，使用字段表达式进行组合类型如tuple何POJO类型的操作比较便捷，在下面的示例中定义wc POJO，包含两个字段”word”和count，基于word进行聚合，可以使用keyBy方法来传名称。

*public class WC {*

*public String word;*

*public int count;*

*}*

*DataStream<WC> words =* // [...]

*DataStream<WC> wordCounts = words.keyBy("word").window(*/\*window specification\*/*);*

Field表达式语法：

* POJO中字段名，例如user引用POJO中user字段
* Tuple类型中，使用fileld名称或者索引，例如f0和5来引用Tuple中第一和第六的字段
* 选择POJO和Tuples中的嵌套字段，例如user.zip引用POJO:user中的zip字段
* 可以使用\*来选择所有的字段作为key

Field表达式的示例如下：

*public static class WC {*

*public ComplexNestedClass complex; //nested POJO*

*private int count;*

*// getter / setter for private field (count)*

*public int getCount() {*

*return count;*

*}*

*public void setCount(int c) {*

*this.count = c;*

*}*

*}*

*public static class ComplexNestedClass {*

*public Integer someNumber;*

*public float someFloat;*

*public Tuple3<Long, Long, String> word;*

*public IntWritable hadoopCitizen;*

*}*

可以定义的Field表达式包括：

* count，WC类中count字段
* complex,递归的选择ComplexNestedClass中所有的字段
* complex.word.f2，选择嵌套Tuple3中的最后一个字段
* complex.hadoopCitizen，选择Hadoop中的IntWritable类型

1. 使用Key Selector函数来定义key

Key Selector使用单个元素作为输入，返回一个Key值，该Key可以是任意的类型，示例如下：

*public class WC {public String word; public int count;}*

*DataStream<WC> words = // [...]*

*KeyedStream<WC> kyed = words*

*.keyBy(new KeySelector<WC, String>() {*

*public String getKey(WC wc) { return wc.word; }*

*});*

示例中返回一个对象的字段

## 2.3 Specifying Transformation Functions

在Flink程序中，大部分的算子需要用户定义函数，下面介绍不同的定义方法。

1. 实现接口，最常用的方法

*class MyMapFunction implements MapFunction<String, Integer> {*

*public Integer map(String value) { return Integer.parseInt(value); }*

*});*

*data.map(new MyMapFunction());*

1. 匿名类，可以将一个函数作为匿名类来传递

*data.map(new MapFunction<String, Integer> () {*

*public Integer map(String value) { return Integer.parseInt(value); }*

*});*

3）Java 8 Lamda表达式

Flink支持Java API中的Java8 Lamda表达式，示例如下：

*data.filter(s -> s.startsWith("http://"));*

*data.reduce((i1,i2) -> i1 + i2);*

4）富函数

需要用户定义函数的算子都可以将rich函数作为参数，例如：

*class MyMapFunction implements MapFunction<String, Integer> {*

*public Integer map(String value) { return Integer.parseInt(value); }*

*});*

可以重写成，如下：

*class MyMapFunction extends RichMapFunction<String, Integer> {*

*public Integer map(String value) { return Integer.parseInt(value); }*

*});*

参数传递方式不变。在富函数中，增加了4个方法：open,close,getRuntimeContext及setRuntimeContext，这些方法用于创建local state、访问broadcast变量、访问runtime信息等。

## 2.4 支持的数据类型

DataSet和DataStream中使用的数据类型有一些限制，目前有以下六种数据类型：

* Java Tuples和Scala Case类
* Java POJOs
* Primitive Types，Integer,String,Double等
* Regular Classes
* Values
* Hadoop Writables，实现Writable接口
* Special Types，包括Scala中的Either,Option及Try等

## 2.5 Accumulators&Counters

累加器是支持add操作的结构体，当Job结束时返回final累加结果。计数器是累计器的直观实现。可以使用Accumulator.add(value)来增加值，在Job结束时将所有分区的结果进行聚合，并返回给客户端。

Flink目前内置了几种累加器，其实现了Accumulator接口

* IntCounter,LongCounter和DoubleCounter
* Histogram，离散数据的聚合值，内部是通过map<Integer,Integer>来实现，可以用来进行分布式计算，例如每行的单词统计。

累加器的使用如下所示：

1. 创建累加器

*private IntCounter numLines = new IntCounter();*

1. 注册累加器

*getRuntimeContext().addAccumulator("num-lines", this.numLines);*

3）执行add操作

*this.numLines.add(1);*

4）获取结果，最后使用由execute方法返回的JobExeuctionResult对象获取结果

*val myJobExecutionResult= env.execute()*

*myJobExecutionResult.getAccumulatorResult("num-lines")*

自定义一个累加器只需要实现Accumulator接口。

https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/api\_concepts.html

# DataStream API编程指南

DataStream是实现数据流上的transformation，包括filtering，updating state， defining windows，aggregating。数据流的数据源包括消息队列、Socket流、文件等，程序通过DataSink返回结果，如将数据写入文件或者发送到标准输出。下面是一个流式数据Example，计算出在Web Socket大小为5秒的窗口中出现的单词数量：

*public class WindowWordCount {*

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

*StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();*

*DataStream<Tuple2<String,Integer>> dataSteam =*

*env.socketTextStream("cmhhost1.novalocal", 9999)*

*.flatMap(new Splitter())*

*.keyBy(0)*

*.timeWindow(Time.seconds(5))*

*.sum(1);*

*dataSteam.print();*

*env.execute("Window WordCount");*

*}*

*public static class Splitter implements*

*FlatMapFunction<String,Tuple2<String,Integer>> {*

*public void flatMap(String sentence, Collector<Tuple2<String,Integer>> out)*

*throws Exception {*

*for (String word: sentence.split(" ")) {*

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

*}*

*}*

*}*

*}*

运行该程序，在cmhost1.novalocal终端上运行netcat开始输入流

*nc –lk 9999*

输入一些单词，作为程序的输入数据，在TaskManaster的out日志上可以看到输出。

1. DataStream Transfomrations

将一个或者多个DataStream转换成一个新的DataStream，具体算子见：

*https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/datastream\_api.html*

*https://www.cnblogs.com/lanyun0520/p/5730403.html*

1. Physical Partitioning，物理级分割

Flink同样提供了在进行一次transformation后针对精确stream分割的低层次的控制(low-level control)

*https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/datastream\_api.html*

*https://www.cnblogs.com/lanyun0520/p/5730403.html*

1. 链接任务以及资源组（Task chaining $ resource groups）

将两个transformation链接起来意味着将它们部署在一起（co-locating），共享同一个线程来获得更好的性能。Flink默认地尽可能地链接Operator（如两个连续的map transformation）。如有需要，API还给出了细粒度的链接控制：

使用StreamExecutionEnvironment.disableOperatorChaining()来关闭整个Job的链接操作。下面表格中的方法则是更加细粒度的控制函数，注意，由于这些函数引用的是前一个transformation，所以它们仅仅在一个DataStream的transformation后使用才是正确的，例如someStream.map( … ).startNewChain()是正确的，而someStream.startNewChain()是**错误**的。

4) 数据源

数据源可以通过StreamExecutionEnvironment.addSource(sourceFunction)来创建数据源。你可以使用Flink提供的source方法，也可以通过实现SourceFunction来编写自定义的非并行数据源，也可以通过实现ParallelSourceFunction接口或继承RichParallelSourceFunction来编写自定义并行数据源。

以下是几个预定义的数据流源，可以通过StreamExecutionEnvironment来访问：

a)   基于文件的：

·  **readTextFile(path)** / **TextInputFormat** - 以行读取方式读文件并返回字符串

·  **readFile(path)** / 任意输入格式 - 按用输入格式的描述读取文件

·  **readFileStream** - 创建一个stream，在文件有改动时追加element

b) 基于Socket的：

· **socketTextStream** - 从socket读取，element可以通过分割符来分开

c) 基于Collection的：

· **fromCollection(Collection)** - 从Java.util.Collection创建一个数据流。collection中所有的element都必须是同一类型的。

·  **fromCollection(Iterator, Class)** - 从一个迭代器中创建一个数据流。class参数明确了迭代器返回的element的类型。

· **fromElement(T …)** - 从一个给定的对象序列创建一个数据流。所有对象都必须是同一类型的。

· **romParallelCollection(SplittableIterator, Class)** - 从一个迭代器中创建一个并行数据流。class参数明确了迭代器返回的element的类型。

·  **generateSequence(from, to)** - 从一个给定区间中生成一个并行数字序列。

d) 自定义：

· **addSource** - 附上一个新的source方法。例如，通过调用addSource(new FlinkKafkaConsumer08<>(…))来从Apache Kafka读取数据，更多信息见于[connector](https://ci.apache.org/projects/flink/flink-docs-release-1.0/apis/streaming/connectors/)

5）Data sink

Data Sink消耗DataStream并将它们转发到文件、socket、外部系统或打印它们。Flink自带了许多内置的输出格式，封装为DataStream的operation中：

· writeAsText() / TextOutputFormat - 以行字符串的方式写文件，字符串通过调用每个element的toString()方法获得。

· writeAsCsv(…) / CsvOutputFormat - 以逗号分隔的值来讲Tuple写入文件，行和域的分隔符是可以配置的。每个域的值是通过调用object的toString()方法获得的。

· print() / printToErr() - 将每个element的toString()值打印在标准输出 / 标准错误流中。可以提供一个前缀（msg）作为输出的前缀，使得在不同print的调用可以互相区分。如果并行度大于1，输出也会以task的标识符（identifier）为产生的输出的前缀。

· writeUsingOutputFormat() / FileOutputFormat - 自定义文件输出所用的方法和基类，支持自定义object到byte的转换。

· writeToSocket - 依据SerializationSchema将element写到socket中。

· addSink - 调用自定义sink方法，Flink自带连接到其他系统的connector（如Apache Kafka），这些connector都以sink方法的形式实现。

注意DataStream的write\*()函数主要用于debug，它们不参与Flink的检查点，这意味着这些方法通常处于“至少一次（at-least-once）“的执行语义下。flush到目标系统的数据依赖于OutputFormat的实现，这意味着不是所有发送到OutputFormat的element都会立即出现在目标系统中，此外，在失效的情况下，这些数据很可能会丢失。

6) 迭代（Iterations）

迭代流程序实现了一个阶段方法并将之嵌入到一个**IterativeStream**中。作为一个可能永远不会结束的程序，它没有最大迭代数，反之，你需要使用**split**或**filter**的transformation来明确流的哪一部分会被反馈到迭代中，哪一部分则继续转发到下游。这里，我们使用filter作为例子，我们定义**IterativeStream**：

IterativeStream**<**Integer**>** iteration **=** input**.**iterate**();**

然后，我们定义在循环中将要进行的逻辑处理，我们通过一系列transformation来实现（这里用了一个简单的map transformation）：

DataStream**<**Integer**>** iterationBody **=** iteration**.**map**(***/\* this is executed many times \*/***);**

我们可以调用**IterativeStream**的**closeWith(feedbackStream)**函数来关闭一个迭代并定义迭代尾。传递给**closeWith**方法的DataStream将会反馈回迭代头。分割出用来反馈的stream的部分和向前传播的stream部分通常的方法便是使用filter来进行分割。这些filter可以定义诸如"termination"逻辑，即element将会传播到下游，而不是被反馈回去。

iteration**.**closeWith**(**iterationBody**.**filter**(***/\* one part of the stream \*/***));**  
DataStream**<**Integer**>** output **=** iterationBody**.**filter**(***/\* some other part of the stream \*/***);**

# DataSet API编程指南

Flink中第二种常规是DataSet程序，在DataSet上的操作包括filtering、mapping、joining及grouping等操作。DataSet的数据源可以是从文件或者本地集合，通过sinks返回结果，例如将数据写到文件或者输出到控制台中。WordCount Example的示例如下：

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

*import org.apache.flink.api.java.DataSet;*

*import org.apache.flink.api.java.ExecutionEnvironment;*

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

*import org.apache.flink.util.Collector;*

*public class WordCountExample {*

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

*final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();*

*DataSet<String> text = env.fromElements("Who's there?",*

*"I think I hear them. Stand, ho! Who's there?");*

*DataSet<Tuple2<String,Integer>> wordCounts = text*

*.flatMap(new LineSplitter())*

*.groupBy(0)*

*.sum(1);*

*wordCounts.print();*

*}*

*public static class LineSplitter implements*

*FlatMapFunction<String,Tuple2<String,Integer>> {*

*public void flatMap(String line,*

*Collector<Tuple2<String,Integer>> out){*

*for (String word : line.split(" ")) {*

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

*}*

*}*

*}}*

Maven依赖如下：

*<dependencies>*

*<dependency>*

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

*<artifactId>flink-java</artifactId>*

*<version>${flink.version}</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>flink-streaming-java\_2.11</artifactId>*

*<version>${flink.version}</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>flink-clients\_2.11</artifactId>*

*<version>${flink.version}</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>flink-connector-wikiedits\_2.11</artifactId>*

*<version>${flink.version}</version>*

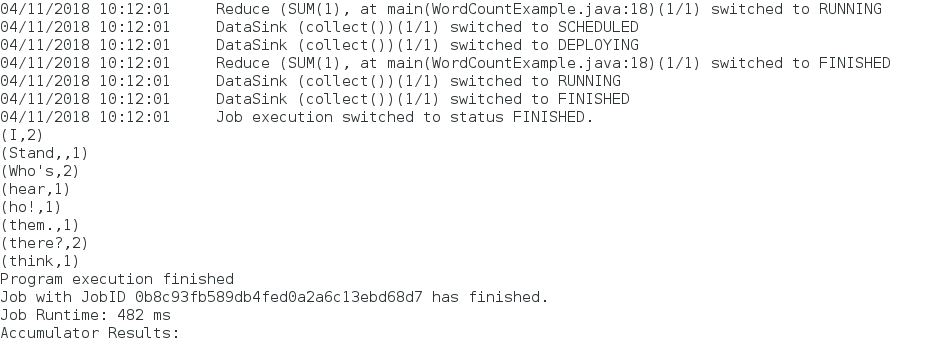
*</dependency>*

*</dependencies>*

编译后，执行如下：

*bin/flink run flink-1.0-SNAPSHOT.jar*

输出：



1）DataSet Transformations

Transformations将数据集转换重新的数据集，程序可以组合使用多个transformation算子，大部分算子和Spark一样，不再详述。

https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/batch/dataset\_transformations.html

2）Data Source

Data Source创建初始数据集，例如读取文件或者java Collections，Flink支持多种格式的数据源。

1. File-based

* readTextFile(path) /TextInputFormat，从文件中读取行数据，转换成Strings
* readTextFileWithValue / TextValueInputFormat，读取行数据，返回StringValues
* readCsvFile/CsvInputFormat，文件中以逗号为间隔来区分字段，返回tuples或者POJO
* readFileOfPrivitive(path,class)/PrimitiveInputFormat，读取文件解析成String或者Integer
* readHadooopFile / FileInputFormat，从HDFS中读取文件，输出tuple2<Key,Value>
* readSequenceFile(key,value,path)/SequenceFileInputFormat

1. Collection-based

* fromCollection(Collection)，基于Collection构建DataSet
* fromCollection(Iterator, Class)
* fromElements(T ...)
* fromParallelCollection(SplittableIterator, Class)
* generateSequence(from, to)

1. Generic

* readFile(inputFormat,path) /FileInputFormat
* createInput(inputFomrat)/InputFormat

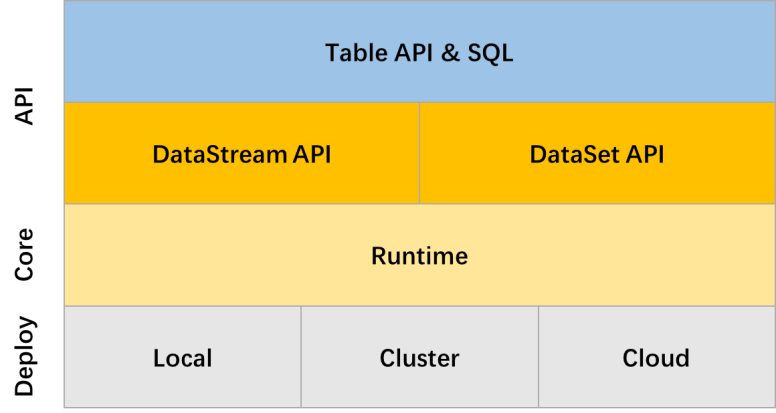
3) Data Sinks

Data Sink是将数据集中的数据存储或者返回，由OutputFormat指定输出文件格式，Flink默认构建了一些输出文件格式：

* writeAsText() / TextOutputFormat，以文本的方式输出
* writeAsCsv/CsvOutputFormat，以csv的方式输出
* print/FileOutputFormat 输出到控制台
* output/OutputFormat，通用的泛型方法，将数据写出去，但不是以文本的方式。例如写入数据库等

# Table API & SQL Beta

Flink Table API是一种关系型类SQL API，用户可以像操作表一样操作数据，而不需要像DataStream一样编写大量的Function。Table & SQL API将DataStream和DataSet API进行统一，对于批查询会随着输入数据的结束并生成有限结果集，数据流上的查询可以一直运行并生成结果流，其关系如下所示：



Flink SQL是基于Apache Calcite来实现标准SQL语法。

SQL API的使用示例如下：

*public class WordCountSQL {*

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

*ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();*

*BatchTableEnvironment tEnv = TableEnvironment.getTableEnvironment(env);*

*DataSet<WC> input = env.fromElements(*

*new WC("Hello",1),*

*new WC("Ciao",1),*

*new WC("Hello",1)*

*);*

*tEnv.registerDataSet("WordCount", input, "word, frequency");*

*Table table = tEnv.sqlQuery(*

*"SELECT word, SUM(frequency) as frequency FROM WordCount GROUP BY word");*

*DataSet<WC> result = tEnv.toDataSet(table, WC.class);*

*result.print();*

*}*

*public static class WC {*

*public String word;*

*public long frequency;*

*// public constructor to make it a Flink POJO*

*public WC() {}*

*public WC(String word, long frequency) {*

*this.word = word;*

*this.frequency = frequency;*

*}*

*@Override*

*public String toString() {*

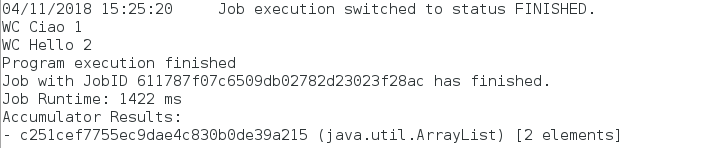
*return "WC " + word + " " + frequency;*

*}*

*}*

*}*

输出日志如下：



https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/table/common.html

https://blog.csdn.net/dax1n/article/details/70207540