**目录**

[一、 Flink开发环境搭建 1](#_Toc1480)

[二、 基本API概念 5](#_Toc29776)

[1、DataSet和DataStream 5](#_Toc26015)

[2、 解剖Flink程序 5](#_Toc29483)

[3、 惰性评估 6](#_Toc7569)

[4、 指定Keys 6](#_Toc32494)

[5、 为Tuples指定Keys 7](#_Toc27673)

[6、 指定转化函数 7](#_Toc6152)

[7、 支持的数据类型 8](#_Toc1115)

[8、 累加器和计数器 9](#_Toc20398)

[三、 DataStreaming API使用 10](#_Toc23225)

[1、 DataStream转化 10](#_Toc22717)

[2、 物理分区方法 12](#_Toc3764)

[3、 任务链 12](#_Toc29818)

[4、 数据源 13](#_Toc9499)

[5、 数据存放 13](#_Toc17036)

# Flink开发环境搭建

本文主要是基于java语言开发的环境搭建。

1. 必备工具，红色部分为必须安装的
2. Java 1.8版本
3. Eclipse
4. Maven
5. Netcat：用于模拟网络包发送
6. Git或者cgwin
7. Curl工具
8. 搭建步骤
9. 在Eclipse中创建maven工程，pom.xml配置信息如下：

*<dependencies>*

*<dependency>*

*<groupId>junit</groupId>*

*<artifactId>junit</artifactId>*

*<version>3.8.1</version>*

*<scope>test</scope>*

*</dependency>*

*<dependency>*

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

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

*<version>1.2.0</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>flink-streaming-java\_2.10</artifactId>*

*<version>1.2.0</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>flink-clients\_2.10</artifactId>*

*<version>1.2.0</version>*

*</dependency>*

*<dependency>*

*<groupId>org.codehaus.jackson</groupId>*

*<artifactId>jackson-core-asl</artifactId>*

*<version>1.9.0</version>*

*</dependency>*

*<dependency>*

*<groupId>com.google.code.findbugs</groupId>*

*<artifactId>jsr305</artifactId>*

*<version>3.0.2</version>*

*</dependency>*

*<dependency>*

*<groupId>com.google.protobuf</groupId>*

*<artifactId>protobuf-java</artifactId>*

*<version>3.2.0</version>*

*</dependency>*

*<dependency>*

*<groupId>com.sun.jersey</groupId>*

*<artifactId>jersey-core</artifactId>*

*<version>1.19.3</version>*

*</dependency>*

*<dependency>*

*<groupId>org.uncommons.maths</groupId>*

*<artifactId>uncommons-maths</artifactId>*

*<version>1.2.1</version>*

*</dependency>*

*<dependency>*

*<groupId>com.typesafe</groupId>*

*<artifactId>config</artifactId>*

*<version>1.2.0</version>*

*</dependency>*

*<dependency>*

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

*<artifactId>commons-lang3</artifactId>*

*<version>3.4</version>*

*</dependency>*

*</dependencies>*

官方提供的文档中只包含flink几个核心的依赖包，这个显然是会让IDE报错的，上面的配置经过实践操作是没有问题的。注意：其中某些包maven自动下载失败，我们可以从[http://mvnrepository.com](http://mvnrepository.com/)下载需要的包放到maven指定位置上。

1. 编写java代码

package com.hugh.demo.flink;

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

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

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

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

import org.apache.flink.streaming.api.windowing.time.Time;

import org.apache.flink.util.Collector;

public class WindowWordCount {

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

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

DataStream<Tuple2<String, Integer>> dataStreaming = env

.socketTextStream("localhost", 9999)

.flatMap(new Splitter())

.keyBy(0)

.timeWindow(Time.seconds(5))

.sum(1);

dataStreaming.print();

env.execute("window worcount");

}

public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>>{

@Override

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));

}

}

}

}

以上代码是官方文档提供的代码，主要功能是：根据用户的输入实时进行WordCount计算。（3）netcat工具调试程序

* 打开netcat工具，输入nc -lp 9999。
* 然后Eclipse上运行程序。
* 接着netcat进入可编辑状态，输入文字，Eclipse的控制台会打印处结果信息。

1. 部署Flink程序

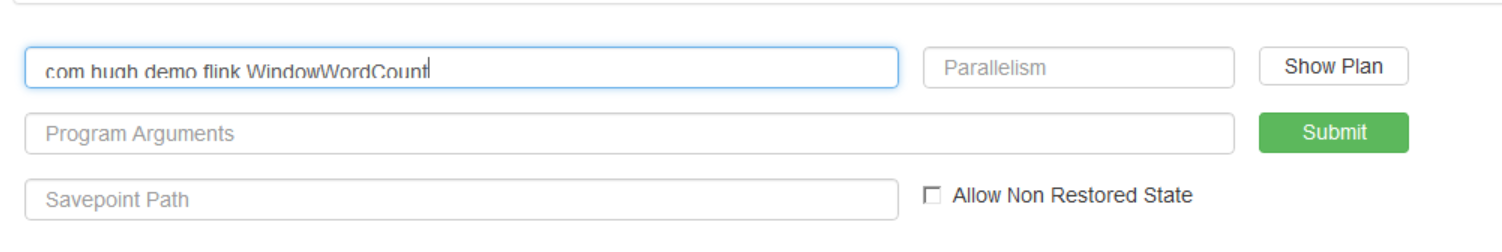
* 进入flink/bin目录，执行start-local.bat。控制台显示：

D:\programs\flink-1.2.0-bin-hadoop27-scala\_2.10\flink-1.2.0\bin>start-local.bat

Starting Flink job manager. Webinterface by default on http://localhost:8081/.

Don't close this batch window. Stop job manager by pressing Ctrl+C.

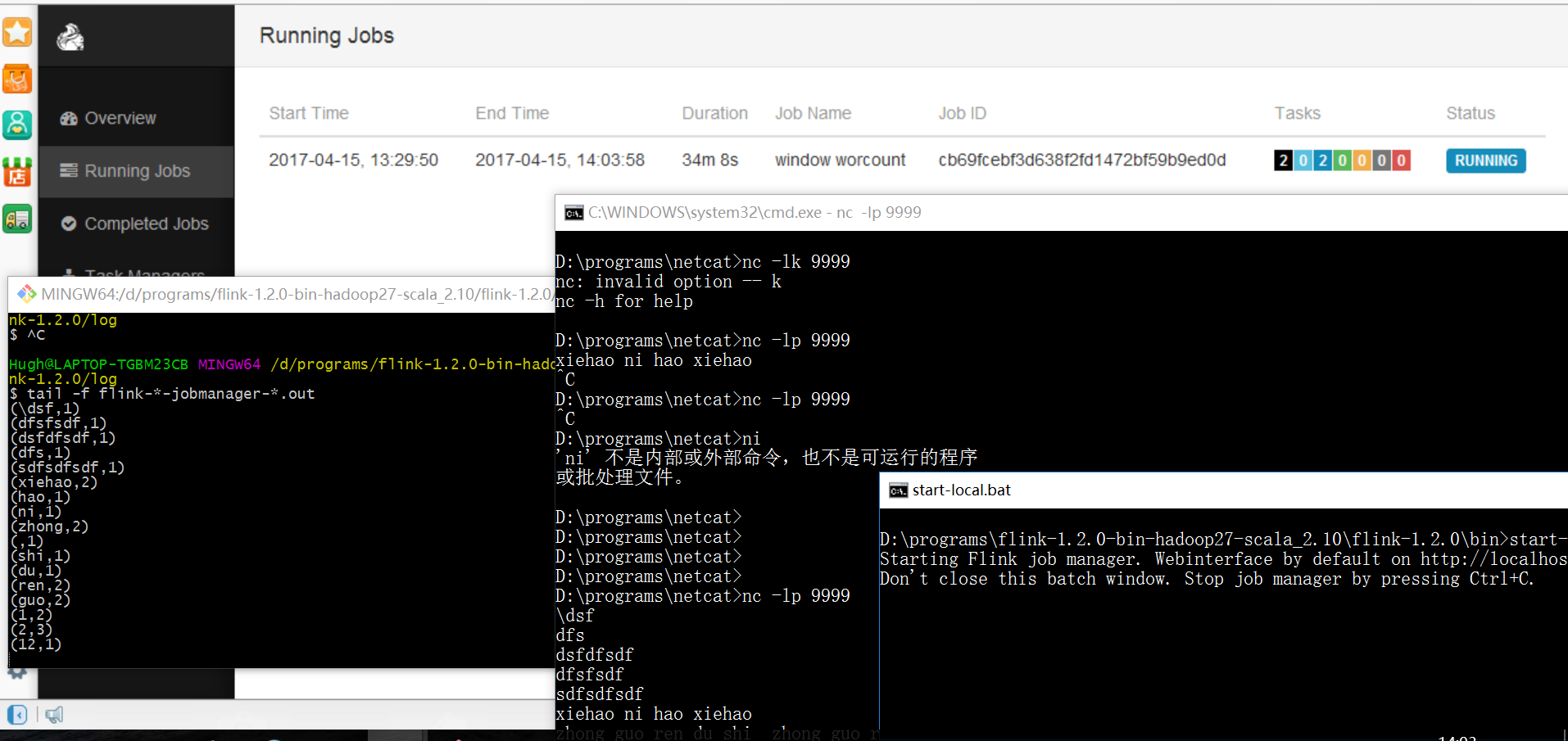
* 打开网页输入http://localhost:8081，进入Flink的任务管理界面
* Eclipse将WindowWordCount.java打包成jar包，然后在Flink管理页面的”Submit new Job”中提交任务。



提交”Submit”之后，就可以在Running Jobs中看到该任务了。

* 打开git工具，在flink根目录下输入命令：tail -f log/flink-\*-jobmanager-\*.out

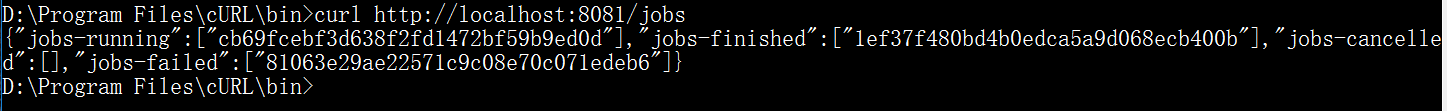
然后在netcat工具中输入字符串，就可以看到git中实时显示计算结果信息。



在Job Manager的Stdout中也可以看到结果信息，只是显示内容比较滞后。

* 通过curl获取任务执行信息，这块可以提供给监控系统使用

获取jobs信息的示例如下：



# 基本API概念

## 1、DataSet和DataStream

它们是数据的不可变集合，DataSet的数据是有限的，而DataStream的数据是无界的。不可变意味着对它们增减元素，你也不能简单地检查里面的元素。一个集合通过在Flink程序中增加数据源初始化，而一个新的集合通过针对之前的集合map、filter转化得到。

## 解剖Flink程序

一个程序的基本构成：

* 获取execution environment
* 加载/创建原始数据
* 指定这些数据的转化方法
* 指定计算结果的存放位置
* 触发程序执行

所有DataSet API在org.apache.flink.api.java包中，所有的DataStream API在org.apache.flink.streaming.api中。

StreamExecutionEnvironment是所有Flink程序的基础，获取方法有：

getExecutionEnvironment()

createLocalEnvironment()

createRemoteEnvironment(String host, int port, String ... jarFiles)

一般情况下使用getExecutionEnvironment。如果你在IDE或者常规java程序中执行可以通过createLocalEnvironment创建基于本地机器的StreamExecutionEnvironment。如果你已经创建jar程序希望通过invoke方式获取里面的getExecutionEnvironment方法可以使用createRemoteEnvironment方式。

以sequence of lines方式读取文件可以参考：

final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

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

然后通过调用DataStream的转化函数执行转化操作，一个map转化如下：

DataStream<String> input = ...;

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

@override

public Integer map(String value){

Return Integer.parseInt(value);

}

});

如果DataStream中包含了最终结果就可以创建sink（数据输出），常见方法：

writeAsText(String path)

print()

通过execute()触发程序在StreamExecutionEnvironment中执行，它返回JobExecutionResult，其中包含了执行时间和收集器结果。

## 惰性评估

所有Flink程序都是惰性执行，当程序的主方法执行时数据的加载、转化操作不会立即执行而执行加入到执行计划中，当在执行环境中调用execute()时才真正执行。好处是：复杂程序让Flink作为整体规划单元执行，内部有优化。

## 指定Keys

一些转化操作（join,coGroup,keyBy,groupBy）需要在元素集中指定一个key，另外的转化操作（Reduce,GroupReduce,Aggregate,Windows)在执行时指定分组的字段。

DataSet分组参考：

DataSet<...> intput = ...

DataSet<...> reduced = input

.groupBy(..)

.reduceGroup(...);

DataStream分组参考：

DataStream<...> input = ...

DataStream<...> windowed = input

.keyBy(...)

.window(...);

Flink的数据模型不是基于key-value pairs。因而，你不需要物理的将数据转化成key和value。Key是虚构的，它们在数据的group操作中做指引作用而已。

## 为Tuples指定Keys

在Tuple（元组）的一个或者多个字段进行分组的举例：

一个字段分组：

DataStream<Tuple3<Integer, String, Long>> input = ...

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

多个字段分组：

DataStream<Tuple3<Integer, String, Long>> input = ...

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

嵌套结构的分组：

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

指定keyBy(0)会使用整个Tuple2（Integer和Float）作为一个Key，使用POJO方式可以只指定一个作为Key。

## 指定转化函数

大部分转化操作需要用户自定义的函数，下面罗列了不同的方式：

1. 实现接口方式

class MyMapFunction implements MapFunction<String, Integer>{

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

});

data.map(new MyMapFunction());

1. 匿名类方式

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

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

});

1. Java 8 Lambdas方式

data.filter(s -> s.startsWith(“http://”));

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

1. Rich Functions方式

所有需要用户自定义函数的转化操作都可以用rich function替换，比如：

class MyMapFunction implements MapFunction<String, Integer>{

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

});

替换成：

class MyMapFunction extends RichMapFunction<String, Integer>{

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

});

执行转化操作：

Data.map(new MyMapFunction());

## 支持的数据类型

7类数据类型如下：

* Java Tuples以及Scala Case Classes
* Java POJOS
* Primitive Types
* Regular Classes
* Values
* Hadoop Writables
* Special Types

1. Tuples

Tuples由不同类型固定数量的字段组成，JAVA API提供了Tuple1到Tuple25。每一个tuple字段又可以由Tuples组成，形成嵌套结构。可以通过字段名直接获取tuple的字段，比如tuple.f4，或者tuple.getField(int position)，其中field索引从0开始。

DataStream<Tuple2<String, Integer>> wordCounts = env.fromElements(

new Tuple2<String, Integer>(“hello”, 1);

new Tuple2<String, Integer>(“world”, 2);

wordCounts.map(new MapFunction<Tuple2<String, Integer>, Integer>(){

@Override

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

Return value.f1;

}

});

wordCounts.keyBy(0);

1. POJOs

public class WordWithCount{

public String word;

public int count;

public WordWithCount(){}

public WordWithCount(String word, int count){

this.word = word;

this.count = count;

}

}

DataStream<WordWithCount> wordCounts = env.fromElements(

new WordWithCount(“hello”, 1),

new WordWithCount(“world”, 2));

wordCounts.keyBy(“word”);

1. Primitive类型

Java的原始类型，比如Integer，String，Double等

1. General Class类型

Flink支持大部分的Java和Scala（API和custom）。严格限制不能序列化的字段的使用，比如文件指针、IO流以及其他本地资源。所有没有被定义为POJO类型的类都被Flink当作general class类型。Flink把这些数据类型当作黑盒处理而无法访问他们的内容（比如有效的sorting），这些类型通过Kryo框架进行序列化和反序列化。

1. Value类型

Value类型主要用于手工序列化和反序列化。用read和write来实现org.apache.flinktypes.Value接口进行序列化和反序列化。

基本的数据类型有：ByteValue, ShortValue, IntValue, LongValue, FloatValue, DoubleValue, StringValue, CharValue, BooleanValue。

1. Hadoop Writables

实现org.apache.hadoop.Writable接口

1. Special类型

Java API中Either接口相关，它可以包含两个可能的类型Left和Right，主要用于错误的处理或者需要输出两种返回值类型的记录。

## 累加器和计数器

累加器由加法操作和最终累加结果组成，他们用在在任务结束后。最简单的累加器是counter，使用Accumulator.add(V value)方法。在Flink任务结束会汇总（合并）各部分结果然后传递给客户端。内嵌的累加器接口：IntCounter, LongCounter和DoubleCounter。

怎么使用累加器：

首先在用户自定义函数中创建一个累加器实例

private IntCounter numLines = new IntCounter()

其次是登记累加器实例

getRuntimeContext().addAccumulator(“num-line”, this.numLines);

然后就可以在函数中使用累加器，包含在open()和close()方法中

this.numLInes.add(1)

最终结果保存在execute()方法返回的JobExecutionResult对象中

myJobExecutionResult.getAccumulatorResult(“num-lines”)

# DataStreaming API使用

## DataStream转化

1. **Map方式**：DataStream -> DataStream

功能：拿到一个element并输出一个element，类似Hive中的UDF函数

举例：

DataStream<Integer> dataStream = //...

dataStream.map(new MapFunction<Integer, Integer>() {

@Override

public Integer map(Integer value) throws Exception {

return 2 \* value;

}

});

1. **FlatMap方式**：DataStream -> DataStream

功能：拿到一个element，输出多个值，类似Hive中的UDTF函数

举例：

dataStream.flatMap(new FlatMapFunction<String, String>() {

@Override

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

throws Exception {

for(String word: value.split(" ")){

out.collect(word);

}

}

});

1. **Filter方式**：DataStream -> DataStream

功能：针对每个element判断函数是否返回true，最后只保留返回true的element

举例：

dataStream.filter(new FilterFunction<Integer>() {

@Override

public boolean filter(Integer value) throws Exception {

return value != 0;

}

});

1. **KeyBy方式**：DataStream -> KeyedStream

功能：逻辑上将流分割成不相交的分区，每个分区都是相同key的元素

举例：

dataStream.keyBy("someKey") // Key by field "someKey"

dataStream.keyBy(0) // Key by the first element of a Tuple

1. **Reduce方式**：KeyedStream -> DataStream

功能：在keyed data stream中进行轮训reduce。

举例：

keyedStream.reduce(new ReduceFunction<Integer>() {

@Override

public Integer reduce(Integer value1, Integer value2)

throws Exception {

return value1 + value2;

}

});

1. **Aggregations方式**：KeyedStream -> DataStream

功能：在keyed data stream中进行聚合操作

举例：

keyedStream.sum(0);

keyedStream.sum("key");

keyedStream.min(0);

keyedStream.min("key");

keyedStream.max(0);

keyedStream.max("key");

keyedStream.minBy(0);

keyedStream.minBy("key");

keyedStream.maxBy(0);

keyedStream.maxBy("key");

1. **Window方式**：KeyedStream -> WindowedStream

功能：在KeyedStream中进行使用，根据某个特征针对每个key用windows进行分组。

举例：

dataStream.keyBy(0).window(TumblingEventTimeWindows.of(Time.seconds(5))); // Last 5 seconds of data

1. **WindowAll方式**：DataStream -> AllWindowedStream

功能：在DataStream中根据某个特征进行分组。

举例：

dataStream.windowAll(TumblingEventTimeWindows.of(Time.seconds(5))); // Last 5 seconds of data

1. **Union方式**：DataStream\* -> DataStream

功能：合并多个数据流成一个新的数据流

举例：

dataStream.union(otherStream1, otherStream2, ...);

1. **Split方式**：DataStream -> SplitStream

功能：将流分割成多个流

举例：

SplitStream<Integer> split = someDataStream.split(new OutputSelector<Integer>() {

@Override

public Iterable<String> select(Integer value) {

List<String> output = new ArrayList<String>();

if (value % 2 == 0) {

output.add("even");

}

else {

output.add("odd");

}

return output;

}

});

1. **Select方式**：SplitStream -> DataStream

功能：从split stream中选择一个流

举例：

SplitStream<Integer> split;

DataStream<Integer> even = split.select("even");

DataStream<Integer> odd = split.select("odd");

DataStream<Integer> all = split.select("even","odd");

## 物理分区方法

Flink提供了low-level控制的分区函数

1. Custom partitioning

功能：使用用户自定义的分区器来选择目标任务的元素

举例：

dataStream.partitionCustom(partitioner, "someKey");

dataStream.partitionCustom(partitioner, 0);

1. Random partitioning

功能：均匀分布地切分元素

举例：

dataStream.shuffle();

1. Broadcasting

功能：传播元素到每个分区

举例：dataStream.broadcast();

## 任务链

把很多转化操作的任务链接在一起放到同一个thread中执行，可以获得更好的性能。使用 StreamExecutionEnvironment.disableOperatorChaining()可以在整个job中去除某个链节点。

1. Start new chain

举例：omeStream.filter(...).map(...).startNewChain().map(...);

1. Disable chaining

举例：someStream.map(...).disableChaining();

1. Set slot sharing group

举例：someStream.filter(...).slotSharingGroup("name");

## 数据源

StreamExecutionEnvironment提供的一些访问数据源的接口

1. 基于文件的数据源

* readTextFile(path)
* readFile(fileInputFormat, path)
* readFile(fileInputFormat, path, watchType, interval, pathFilter, typeInfo)

1. 基于Socket的数据源

* socketTextStream

1. 基于Collection的数据源

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

## 数据存放

* writeAsText()
* writeAsCsv(...)
* print() / printToErr()
* writeUsingOutputFormat() / FileOutputFormat
* writeToSocket
* addSink