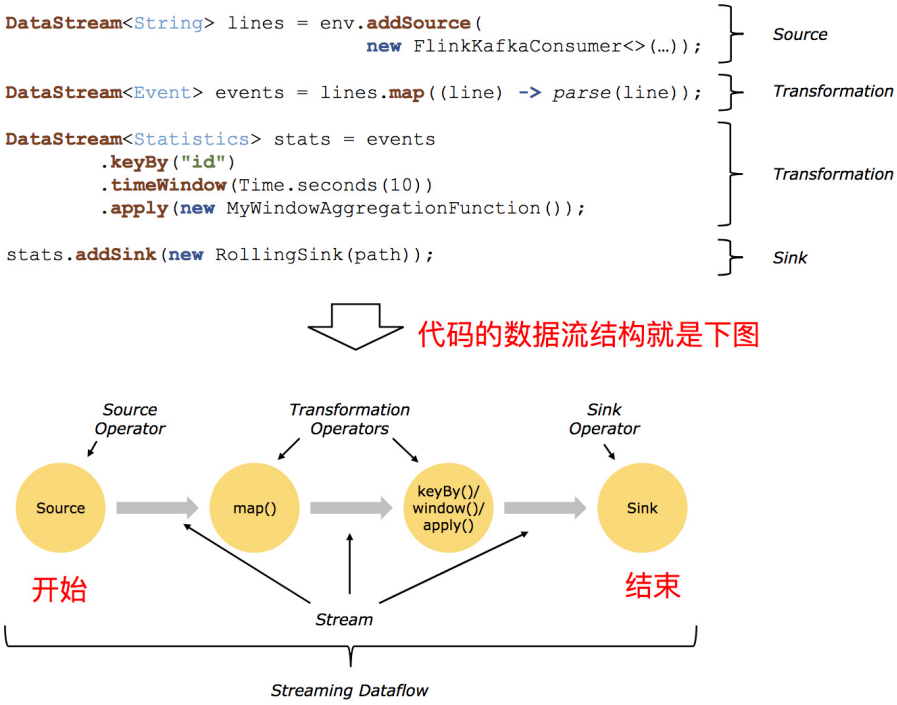
toc: true title: 《从0到1学习Flink》—— Flink Data transformation(转换) date: 2018-11-04 tags:

- Flink
- 大数据
- 流式计算

前言

在第一篇介绍 Flink 的文章 《《从0到1学习Flink》—— Apache Flink 介绍》 中就说过 Flink 程序的结构



Flink 应用程序结构就是如上图所示：

1、Source: 数据源，Flink 在流处理和批处理上的 source 大概有 4 类：基于本地集合的 source、基于文件的 source、基于网络套接字的 source、自定义的 source。自定义的 source 常见的有 Apache kafka、Amazon Kinesis Streams、RabbitMQ、Twitter Streaming API、Apache NiFi 等，当然你也可以定义自己的 source。

2、Transformation：数据转换的各种操作，有 Map / FlatMap / Filter / KeyBy / Reduce / Fold / Aggregations / Window / WindowAll / Union / Window join / Split / Select / Project 等，操作很多，可以将数据转换计算成你想要的数据。

3、Sink：接收器，Flink 将转换计算后的数据发送的地点 ，你可能需要存储下来，Flink 常见的 Sink 大概有如下几类：写入文件、打印出来、写入 socket 、自定义的 sink 。自定义的 sink 常见的有 Apache kafka、RabbitMQ、MySQL、ElasticSearch、Apache Cassandra、Hadoop FileSystem 等，同理你也可以定义自己的 sink。

在上四篇文章介绍了 Source 和 Sink：

1、《从0到1学习Flink》—— Data Source 介绍

2、《从0到1学习Flink》—— 如何自定义 Data Source ？

3、《从0到1学习Flink》—— Data Sink 介绍

4、《从0到1学习Flink》—— 如何自定义 Data Sink？

那么这篇文章我们就来看下 Flink Data Transformation 吧，数据转换操作还是蛮多的，需要好好讲讲！

Transformation

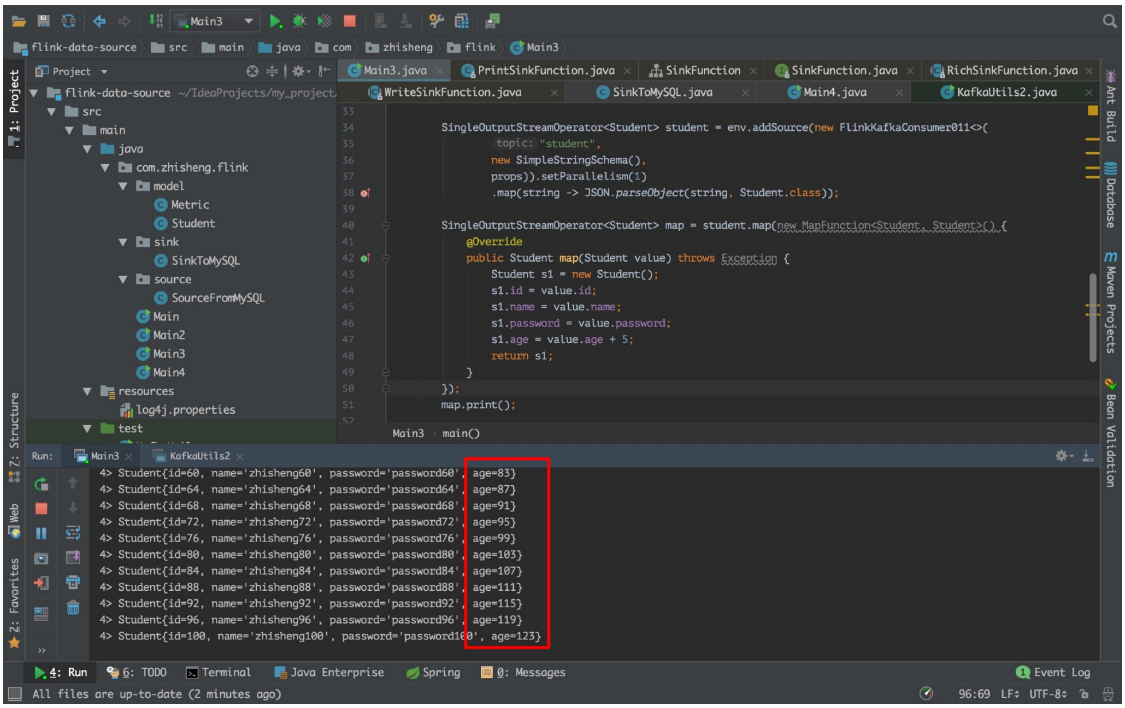
Map

这是最简单的转换之一，其中输入是一个数据流，输出的也是一个数据流：

还是拿上一篇文章的案例来将数据进行 map 转换操作：

```
SingleOutputStreamOperator<Student> map = student.map(new
MapFunction<Student, Student>() {
    @Override
    public Student map(Student value) throws Exception {
        Student s1 = new Student();
        s1.id = value.id;
        s1.name = value.name;
        s1.password = value.password;
        s1.age = value.age + 5;
        return s1;
    }
});
map.print();
```

将每个人的年龄都增加 5 岁，其他不变。

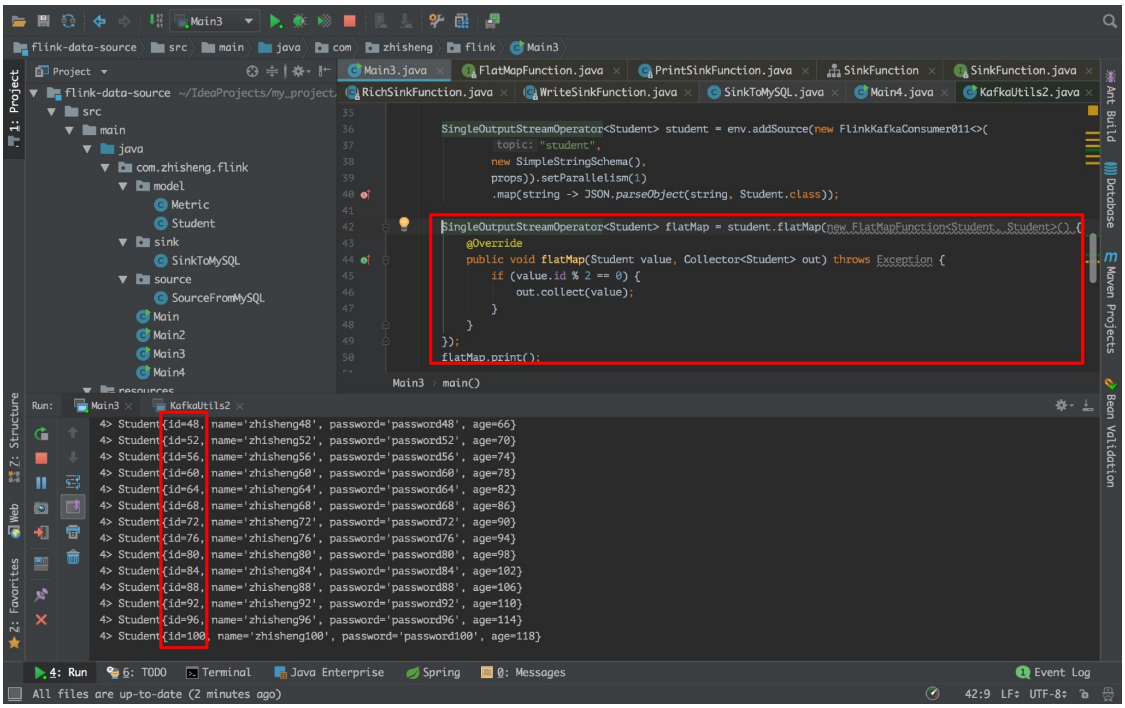


FlatMap

FlatMap 采用一条记录并输出零个，一个或多个记录。

```
SingleOutputStreamOperator<Student> flatMap = student.flatMap(new
FlatMapFunction<Student, Student>() {
    @Override
    public void flatMap(Student value, Collector<Student> out)
throws Exception {
        if (value.id % 2 == 0) {
            out.collect(value);
        }
    }
});
flatMap.print();
```

这里将 id 为偶数的聚集出来。

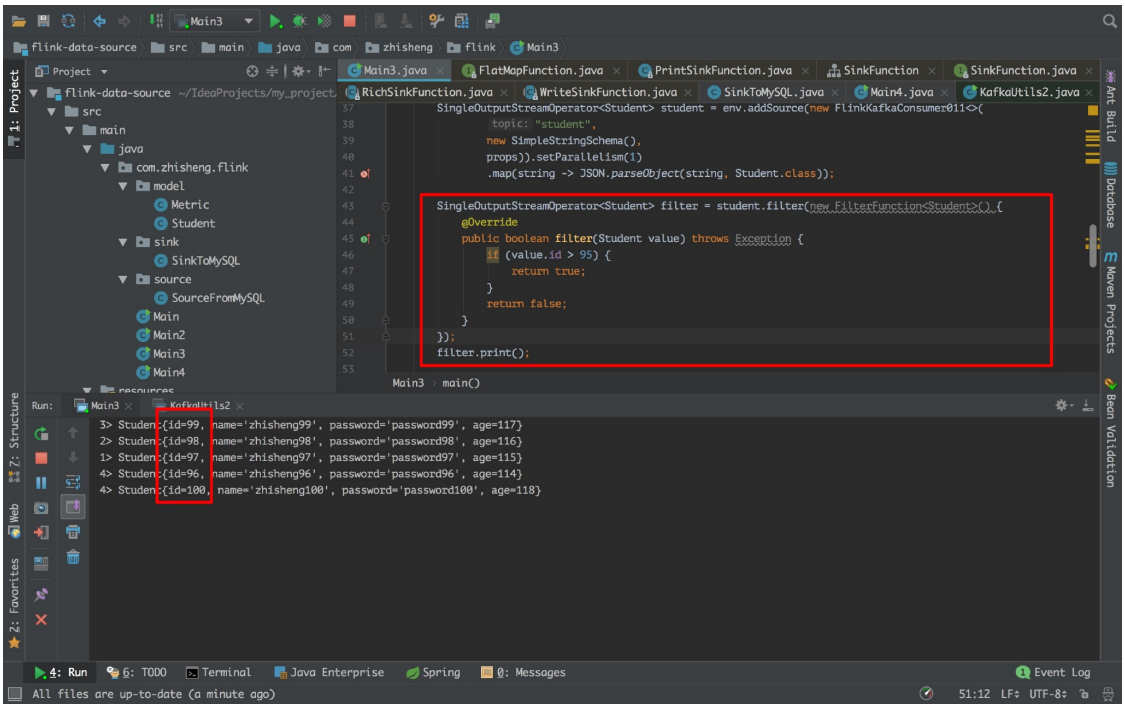


Filter

Filter 函数根据条件判断出结果。

```
SingleOutputStreamOperator<Student> filter = student.filter(new
FilterFunction<Student>() {
    @Override
    public boolean filter(Student value) throws Exception {
        if (value.id > 95) {
            return true;
        }
        return false;
    }
});
filter.print();
```

这里将 id 大于 95 的过滤出来，然后打印出来。



KeyBy

KeyBy 在逻辑上是基于 key 对流进行分区。在内部，它使用 hash 函数对流进行分区。它返回 KeyedDataStream 数据流。

```
KeyedStream<Student, Integer> keyBy = student.keyBy(new
KeySelector<Student, Integer>() {
    @Override
    public Integer getKey(Student value) throws Exception {
        return value.age;
    }
});
keyBy.print();
```

上面对 student 的 age 做 KeyBy 操作分区

Reduce

Reduce 返回单个的结果值，并且 reduce 操作每处理一个元素总是创建一个新值。常用的方法有 average, sum, min, max, count，使用 reduce 方法都可实现。

```
SingleOutputStreamOperator<Student> reduce = student.keyBy(new
KeySelector<Student, Integer>() {
    @Override
    public Integer getKey(Student value) throws Exception {
        return value.age;
    }
}).reduce(new ReduceFunction<Student>() {
    @Override
    public Student reduce(Student value1, Student value2) throws
Exception {
        Student student1 = new Student();
        student1.name = value1.name + value2.name;
        student1.id = (value1.id + value2.id) / 2;
        student1.password = value1.password + value2.password;
        student1.age = (value1.age + value2.age) / 2;
        return student1;
    }
});
reduce.print();
```

上面先将数据流进行 keyby 操作，因为执行 reduce 操作只能是 KeyedStream，然后将 student 对象的 age 做了一个求平均值的操作。

Fold

Fold 通过将最后一个文件夹流与当前记录组合来推出 KeyedStream。 它会发回数据流。

```
KeyedStream.fold("1", new FoldFunction<Integer, String>() {
    @Override
    public String fold(String accumulator, Integer value) throws
Exception {
        return accumulator + "=" + value;
    }
})
```

Aggregations

DataStream API 支持各种聚合，例如 min，max，sum 等。 这些函数可以应用于 KeyedStream 以获得 Aggregations 聚合。

```
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")
```

max 和 maxBy 之间的区别在于 max 返回流中的最大值，但 maxBy 返回具有最大值的键， min 和 minBy 同理。

Window

Window 函数允许按时间或其他条件对现有 KeyedStream 进行分组。 以下是以 10 秒的时间窗口聚合：

```
inputStream.keyBy(0).window(Time.seconds(10));
```

Flink 定义数据片段以便（可能）处理无限数据流。 这些切片称为窗口。 此切片有助于通过应用转换处理数据块。 要对流进行窗口化，我们需要分配一个可以进行分发的键和一个描述要对窗口化流执行哪些转换的函数

要将流切片到窗口，我们可以使用 Flink 自带的窗口分配器。 我们有选项，如 tumbling windows, sliding windows, global 和 session windows。 Flink 还允许您通过扩展 WindowAssginer 类来编写自定义窗口分配器。 这里先预留下篇文章来讲解这些不同的 windows 是如何工作的。

WindowAll

windowAll 函数允许对常规数据流进行分组。 通常，这是非并行数据转换，因为它在非分区数据流上运行。

与常规数据流功能类似，我们也有窗口数据流功能。 唯一的区别是它们处理窗口数据流。 所以窗口缩小就像 Reduce 函数一样，Window fold 就像 Fold 函数一样，并且还有聚合。

```
inputStream.keyBy(0).windowAll(Time.seconds(10));
```

Union

Union 函数将两个或多个数据流结合在一起。 这样就可以并行地组合数据流。 如果我们将一个流与自身组合，那么它会输出每个记录两次。

```
inputStream.union(inputStream1, inputStream2, ...);
```

Window join

我们可以通过一些 key 将同一个 window 的两个数据流 join 起来。

```
inputStream.join(inputStream1)
           .where(0).equalTo(1)
           .window(Time.seconds(5))
           .apply (new JoinFunction () {...});
```

以上示例是在 5 秒的窗口中连接两个流，其中第一个流的第一个属性的连接条件等于另一个流的第二个属性。

Split

此功能根据条件将流拆分为两个或多个流。 当您获得混合流并且您可能希望单独处理每个数据流时，可以使用此方法。

```
SplitStream<Integer> split = inputStream.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;
    }
});
```

Select

此功能允许您从拆分流中选择特定流。

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

Project

Project 函数允许您从事件流中选择属性子集，并仅将所选元素发送到下一个处理流。

```
DataStream<Tuple4<Integer, Double, String, String>> in = // [...]
DataStream<Tuple2<String, String>> out = in.project(3,2);
```

上述函数从给定记录中选择属性号 2 和 3。 以下是示例输入和输出记录：

```
(1,10.0,A,B)=> (B,A)
(2,20.0,C,D)=> (D,C)
```

最后

本文主要介绍了 Flink Data 的常用转换方式：Map、FlatMap、Filter、KeyBy、Reduce、Fold、Aggregations、Window、WindowAll、Union、Window Join、Split、Select、Project 等。并用了点简单的 demo 介绍了如何使用，具体在项目中该如何将数据流转换成我们想要的格式，还需要根据实际情况对待。

关注我

转载请务必注明原创地址为：http://www.54tianzhisheng.cn/2018/11/04/Flink-Data-transformation/

另外我自己整理了些 Flink 的学习资料，目前已经全部放到微信公众号了。你可以加我的微信：zhisheng_tian，然后回复关键字：Flink 即可无条件获取到。

Github 代码仓库

https://github.com/zhisheng17/flink-learning/

以后这个项目的所有代码都将放在这个仓库里，包含了自己学习 flink 的一些 demo 和博客

相关文章

1、《从0到1学习Flink》—— Apache Flink 介绍

2、《从0到1学习Flink》—— Mac 上搭建 Flink 1.6.0 环境并构建运行简单程序入门

3、《从0到1学习Flink》—— Flink 配置文件详解

4、《从0到1学习Flink》—— Data Source 介绍

5、《从0到1学习Flink》—— 如何自定义 Data Source ？

6、《从0到1学习Flink》—— Data Sink 介绍

7、《从0到1学习Flink》—— 如何自定义 Data Sink ？

8、《从0到1学习Flink》—— Flink Data transformation(转换)

9、《从0到1学习Flink》—— 介绍Flink中的Stream Windows

10、《从0到1学习Flink》—— Flink 中的几种 Time 详解

11、《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch