Flink.tw
Apache Flink Taiwan User Group

Apache Flink Tutorial

DataSet API

Who Am I?

- Newbie in Apache Flink
- BlueWell Technology
  - Big Data Architect
  - Focuses
    - Open DC/OS
    - CoreOS
    - Kubernetes
    - Apache Flink
    - Data Science

Agenda

- Apache Flink Type System
  - Atomic
  - Composite
  - Tuple
  - POJO
  - Scala case class
- Transformations
  - Transformations on DataSet
  - Rich Functions
  - Accumulators & Counters
  - Annotations

Agenda

- Iterations
    - Bulk iterations
    - Delta iterations

Flink.tw
Apache Flink Taiwan User Group

Basic Structure Of Apache Flink Programs

- For each Apache Flink Program, the basic structure is listed as follows.
  - Obtain an execution environment.
    - ExecutionEnvironment.getExecutionEnvironment()
  - Load/create DataSets from data sources.
    - readFile(), readTextFile(), readCsvFile(), etc.
    - fromElements(), fromCollection(), etc.
  - Execute some transformations on the DataSets.
    - filter(), map(), reduce(), etc.
  - Specify where to save results of the computations.
    - write(), writeAsCsv(), etc.
    - collect(), print(), etc.
  - Trigger the program execution.

Hands-on
BasicStructure

Apache Flink Type System

- Flink attempts to support all data types
  - Facilitate programming
  - Seamlessly integrate with legacy code
- Flink analyzes applications before execution for
  - Identifying used data types
  - Determining serializers and comparators
- Data types could be
  - Atomic data types
  - Composite data types

Composite Data Types

- Composite Data Types include
  - Tuples
    - In Java
    - In Scala
  - POJOs
  - Scala case class

Tuple Data Types

- Flink supports Tuple in
  - Java: org.apache.flink.api.java.tuple.Tuple<n>
    - n = 1, …, 25
  - Scala: premitive Tuple<n>
    - n = 1, …, 22
- Key declarations
  - Field index
  - E.g., dataset.groupBy(0).sum(1)
  - E.g., dataset.groupBy("_1").sum("_2")

POJOs

- POJOs – A java class with
    - A default constructor without parameters.
    - All fields are
        - public or
        - private but have getter & setter
        - Ex.

```
public class Car {
    public int id;
    public String brand;
    public Car() {};
    public Car(int id, String brand) {…};
}
```

POJOs

- Key Declarations
  - Field name as a string
  - E.g., cars.groupBy(“brand”)

Flink.tw
Apache Flink Taiwan User Group

Scala Case Class

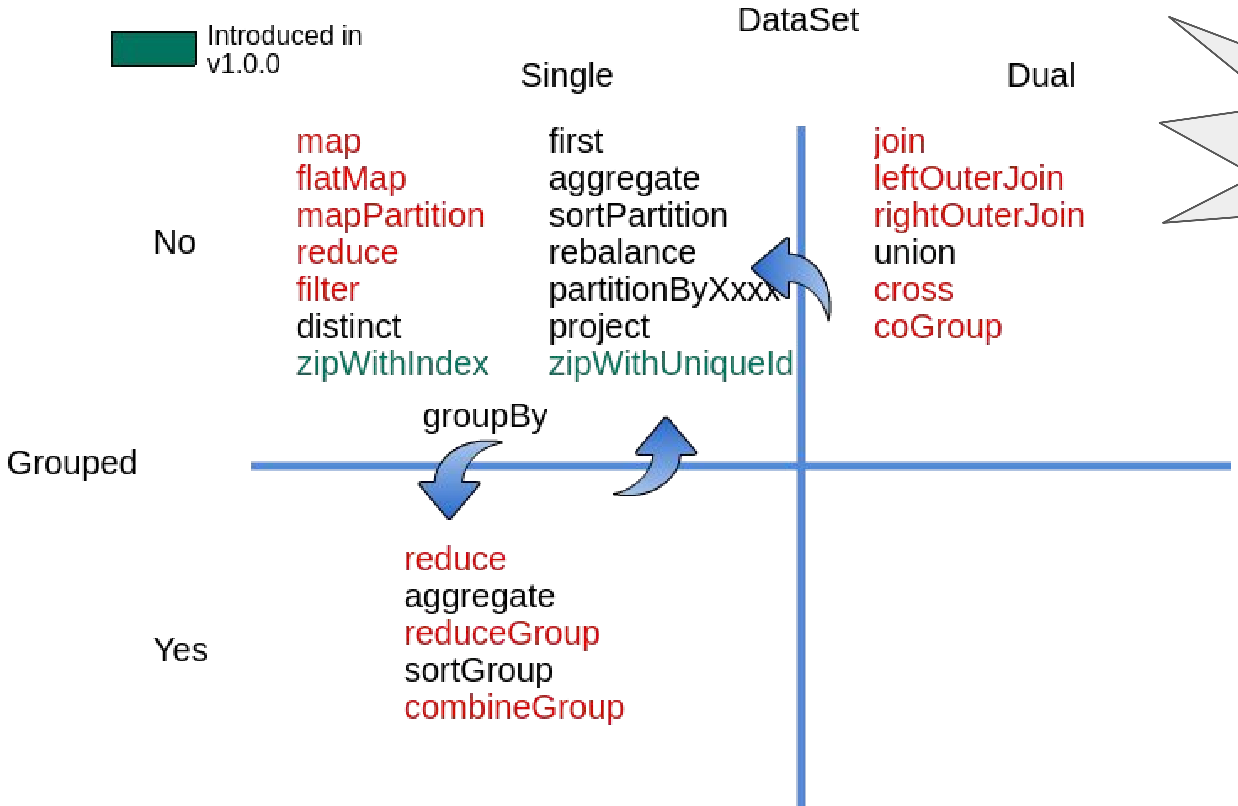
- Primitive Scala case classes are supported
  - E.g., case class Car(id: Int, brand: String)
- Key declarations
  - Field name as a string
    - E.g., cars.groupBy("brand")
  - Field name
    - E.g., cars.groupBy(_.brand)

Hands-on TypeSystem

DataSet Transformations

🟥 Rich functions are available

🟩 Introduced in v1.0.0



Hands-on Transformation

Introduction To Rich Functions

- Purpose
  - Implement complicated user-defined functions
  - Broadcast variables
  - Access to Accumulators & Counters
- Structure
  - open(): initialization, one-shot setup work.
  - close(): tear-down, clean-up work.
  - get/setRuntimeContext(): access to RuntimeContext.
  - corresponding transformation method, e.g., map, join, etc.

Broadcast Variables

- Register
  - dataset.map(new RichMapFunction())
    .withBroadcastSet(toBroadcast, “varName”)
- Access in Rich Functions
  - Initialize the broadcasted variables in open() via
  - getRuntimeContext().getBroadcastVariable(“varName”)
  - Access them in the whole class scope.

Accumulators & Counters

- Purpose
  - Debugging
  - First glance of DataSets
- Counters are kinds of accumulator
- Structure
  - An add operation
  - A final accumulated result (available after the job ended)
- Flink will automatically sum up all partial results.

Accumulators & Counters

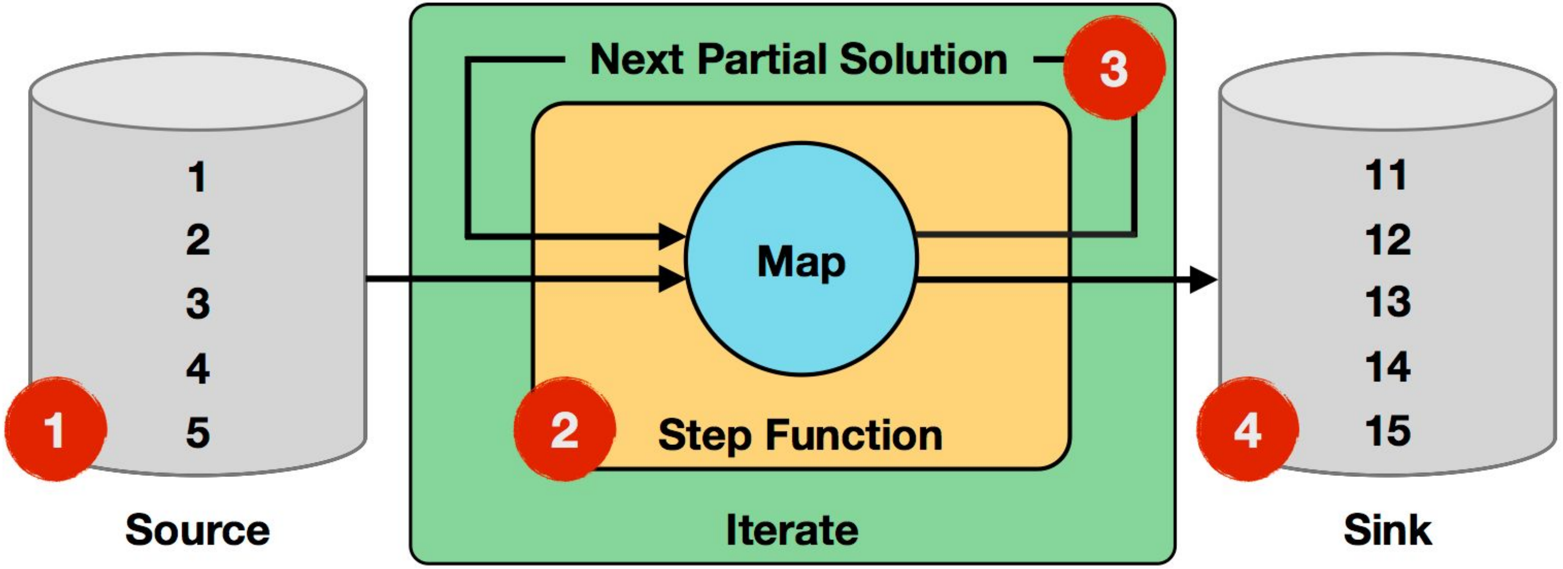
- Built-in Accumulators
  - IntCounter, LongCounter, DoubleCounter
  - Histogram: map from integer to integer, distributions
- Register
  - new IntCounter()
  - getRuntimeContext().addAccumulator(“accuName”, counter)
- Access
  - In Rich Functions
    - getRuntimeContext().getAccumulator(“accuName”)
  - In the end of job
    - JobExecutionResult.getAccumulatorResult(“accuName")

Semantic Annotations

- Give Flink hints about the behavior of a function
    - A powerful means to speed up execution
    - Reusing sort orders or partitions across multiple operations
    - Prevent programs from unnecessary data shuffling or unnecessary sorts
- Types of Annotation
    - Forwarded fields annotations (@ForwardedFields)
    - Non-forwarded fields annotations (@NonForwardedFields)
        - Black or White in place
    - Read fields annotations (@ReadFields)
    - Fields to be read and evaluated

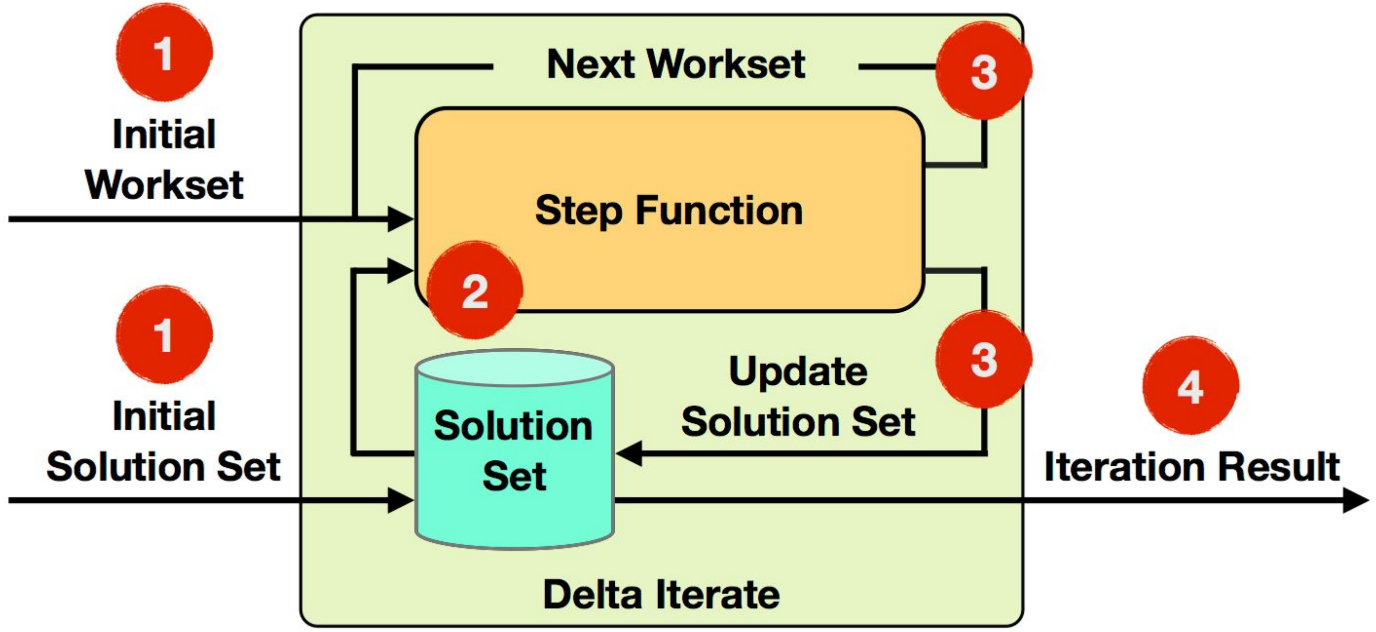
Iterations

- Bulk iterations
  - Partial Solution
  - Iteration Result



Iterations

- Delta Iterations
  - Workset / Update Solution Set
  - Iteration Result



The End

Thanks!!