dataArtisans

Apache Flink® Training

DataSet API Basics

June 3rd, 2015

DataSet API

- Batch Processing
- Java, Scala, and Python
- All examples here in Java
- Many concepts can be translated to the DataStream API
- Documentation available at flink.apache.org

DataSet API by Example

WordCount: main method

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

Execution Environment

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

Data Sources

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

Data types

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

Transformations

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

User functions

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

DataSinks

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

Execute!

```
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            //sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
```

WordCount: Map

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

  @Override
  public void flatMap(String value,
                      Collector<Tuple2<String, Integer>> out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(
                   new Tuple2<String, Integer>(token, 1));
            }
        }
    }
}
```

WordCount: Map: Interface

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

  @Override
  public void flatMap(String value,
                      Collector<Tuple2<String, Integer>> out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(
                   new Tuple2<String, Integer>(token, 1));
            }
        }
    }
}
```

WordCount: Map: Types

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

  @Override
  public void flatMap(String value,
                      Collector<Tuple2<String, Integer>> out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(
                    new Tuple2<String, Integer>(token, 1));
            }
        }
    }
}
```

WordCount: Map: Collector

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

  @Override
  public void flatMap(String value,
                      Collector<Tuple2<String, Integer>> out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(
                   new Tuple2<String, Integer>(token, 1));
            }
        }
    }
}
```

WordCount: Reduce

```
public static class SumWords implements
 GroupReduceFunction<Tuple2<String, Integer>,
                     Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                       Collector<Tuple2<String, Integer>> out) {
          int count = 0;
          String word = null;
          for (Tuple2<String, Integer> tuple : values) {
              word = tuple.f0;
              count++;
          }
          out.collect(new Tuple2<String, Integer>(word, count));
    }

}
```

WordCount: Reduce: Interface

```
public static class SumWords implements
 GroupReduceFunction<Tuple2<String, Integer>,
                     Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                       Collector<Tuple2<String, Integer>> out) {
          int count = 0;
          String word = null;
          for (Tuple2<String, Integer> tuple : values) {
              word = tuple.f0;
              count++;
          }
          out.collect(new Tuple2<String, Integer>(word, count));
    }

}
```

WordCount: Reduce: Types

```
public static class SumWords implements
 GroupReduceFunction<Tuple2<String, Integer>,
                     Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                       Collector<Tuple2<String, Integer>> out) {
          int count = 0;
          String word = null;
          for (Tuple2<String, Integer> tuple : values) {
              word = tuple.f0;
              count++;
          }
          out.collect(new Tuple2<String, Integer>(word, count));
    }

}
```

WordCount: Reduce: Collector

```
public static class SumWords implements
 GroupReduceFunction<Tuple2<String, Integer>,
                     Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                       Collector<Tuple2<String, Integer>> out) {
          int count = 0;
          String word = null;
          for (Tuple2<String, Integer> tuple : values) {
              word = tuple.f0;
              count++;
          }
          out.collect(new Tuple2<String, Integer>(word, count));
    }

}
```

DataSet API Concepts

Data Types

- Basic Java Types
  - String, Long, Integer, Boolean,…
  - Arrays
- Composite Types
  - Tuples
  - Many more (covered in the advanced slides)

Tuples

- The easiest and most lightweight way of encapsulating data in Flink
- Tuple1 up to Tuple25

```
Tuple2<String, String> person =
  new Tuple2<String, String>("Max", "Mustermann”);

Tuple3<String, String, Integer> person =
  new Tuple3<String, String, Integer>("Max", "Mustermann", 42);

Tuple4<String, String, Integer, Boolean> person =
  new Tuple4<String, String, Integer, Boolean>("Max", "Mustermann", 42, true);

// zero based index!
String firstName = person.f0;
String secondName = person.f1;
Integer age = person.f2;
Boolean fired = person.f3;
```

Transformations: Map

```
DataSet<Integer> integers = env.fromElements(1, 2, 3, 4);

// Regular Map - Takes one element and produces one element
DataSet<Integer> doubleIntegers =
     integers.map(new MapFunction<Integer, Integer>() {
     @Override
          public Integer map(Integer value) {
            return value * 2;
          }
     });

doubleIntegers.print();
> 2, 4, 6, 8


// Flat Map - Takes one element and produces zero, one, or more elements.
DataSet<Integer> doubleIntegers2 =

     integers.flatMap(new FlatMapFunction<Integer, Integer>() {
          @Override
          public void flatMap(Integer value, Collector<Integer> out) {
               out.collect(value * 2);
          }
     });

doubleIntegers2.print();
> 2, 4, 6, 8
```

Transformations: Filter

```
// The DataSet
DataSet<Integer> integers = env.fromElements(1, 2, 3, 4);

DataSet<Integer> filtered =

    integers.filter(new FilterFunction<Integer>() {
        @Override
        public boolean filter(Integer value) {
            return value != 3;
        }
    });

integers.print();
> 1, 2, 4
```

Groupings and Reduce

- DataSets can be split into groups
- Groups are defined using a common key

Name	Age
Stephan	18
Fabian	23
Julia	27
Romeo	27
Anna	18

```
// (name, age) of employees
DataSet<Tuple2<String, Integer>> employees = …

// group by second field (age)
DataSet<Integer, Integer> grouped = employees.groupBy(1)
  // return a list of age groups with its counts
  .reduceGroup(new CountSameAge());
```

AgeGroup	Count
18	2
23	1
27	2

GroupReduce

```
public static class CountSameAge implements GroupReduceFunction
<Tuple2<String, Integer>, Tuple2<Integer, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                       Collector<Tuple2<Integer, Integer>> out) {

        Integer ageGroup = 0;
        Integer countsInGroup = 0;

        for (Tuple2<String, Integer> person : values) {
            ageGroup = person.f1;
            countsInGroup++;
        }

        out.collect(new Tuple2<Integer, Integer>
                                            (ageGroup, countsInGroup));
    }
}
```

Joining two DataSets

Authors		
Id	Name	email
1	Fabian	fabian@..
2	Julia	julia@...
3	Max	max@...
4	Romeo	romeo@.

Posts		
Title	Content	Author id
…	…	2
..	..	4
..	..	4
..	..	1
..	..	2

```
// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;
// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

DataSet<Tuple2<
            Tuple3<Integer, String, String>,
            Tuple3<String, String, Integer>
        >> archive = authors.join(posts).where(0).equalTo(2);
```

Joining two DataSets

```
// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;
// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

DataSet<Tuple2<
            Tuple3<Integer, String, String>,
            Tuple3<String, String, Integer>
        >> archive = authors.join(posts).where(0).equalTo(2);
```

Archive					
Id	Name	email	Title	Content	Author id
1	Fabian	fabian@..	..	..	1
2	Julia	julia@...	..	..	2
2	Julia	julia@...	..	..	2
3	Romeo	romeo@...	..	..	4
4	Romeo	romeo@.	..	..	4

Join with join function

```
// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;
// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

// (title, author name)
DataSet<Tuple2<String, String>> archive =
    authors.join(posts).where(0).equalTo(2)
    .with(new PostsByUser());

public static class PostsByUser implements
  JoinFunction<Tuple3<Integer, String, String>,
               Tuple3<String, String, Integer>,
               Tuple2<String, String>> {
    @Override
    public Tuple2<String, String> join(
        Tuple3<Integer, String, String> left,
        Tuple3<String, String, Integer> right) {
            return new Tuple2<String, String>(left.f1, right.f0);
    }
}
```

Archive	
Name	Title
Fabian	..
Julia	..
Julia	..
Romeo	..
Romeo	..

Data Sources

Text

- `readTextFile("/path/to/file")`

CSV

- `readCsvFile("/path/to/file")`

Collection

- `fromCollection(collection)`
- `fromElements(1,2,3,4,5)`

Data Sources: Collections

```
ExecutionEnvironment env =
    ExecutionEnvironment.getExecutionEnvironment();

// read from elements
DataSet<String> names = env.fromElements("Some", "Example",
"Strings");

// read from Java collection
List<String> list = new ArrayList<String>();
list.add("Some");
list.add("Example");
list.add("Strings");

DataSet<String> names = env.fromCollection(list);
```

Data Sources: File-Based

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();


// read text file from local or distributed file system
DataSet<String> localLines =
    env.readTextFile(”/path/to/my/textfile");


// read a CSV file with three fields
DataSet<Tuple3<Integer, String, Double>> csvInput =
                env.readCsvFile(“/the/CSV/file")
               .types(Integer.class, String.class, Double.class);


// read a CSV file with five fields, taking only two of them
DataSet<Tuple2<String, Double>> csvInput =
               env.readCsvFile(“/the/CSV/file")
               // take the first and the fourth field
               .includeFields("10010")
               .types(String.class, Double.class);
```

Data Sinks

Text

- `writeAsText("/path/to/file")`
- `writeAsFormattedText("/path/to/file", formatFunction)`

CSV

- `writeAsCsv("/path/to/file")`

Return data to the Client

- `Print()`
- `Collect()`
- `Count()`

Data Sinks

- Lazily executed when `env.execute()` is called

```
DataSet<…> result;

// write DataSet to a file on the local file system
result.writeAsText(“/path/to/file");

// write DataSet to a file and overwrite the file if it exists
result.writeAsText("/path/to/file”,FileSystem.WriteMode.OVERWRITE);

// tuples as lines with pipe as the separator "a|b|c”
result.writeAsCsv("/path/to/file", "\n", "|");

// this wites values as strings using a user-defined TextFormatter
object
result.writeAsFormattedText("/path/to/file",
        new TextFormatter<Tuple2<Integer, Integer>>() {
            public String format (Tuple2<Integer, Integer> value) {
                return value.f1 + " - " + value.f0;
            }
        });
```

Data Sinks

- Eagerly executed

```
DataSet<Tuple2<String, Integer> result;


// print
result.print();

// count
int numberOfElements = result.count();

// collect
List<Tuple2<String, Integer> materializedResults = result.collect();
```

Best Practices

Some advice

- Use `env.fromElements(..)` or `env.fromCollection(..)` to quickly get a DataSet to experiment with
- Use `print()` to quickly print a DataSet
- Use `collect()` to quickly retrieve a DataSet
- Use `name()` on an Operator to find it easily in the logs