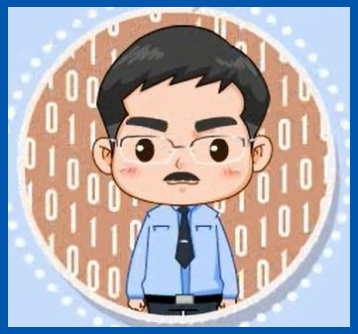


《Flink编程基础（Scala版）》

教材官网： http://dblab.xmu.edu.cn/post/flink/

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第6章 DataSet API

（PPT版本号：2021年3月版本）

扫一扫访问教材官网

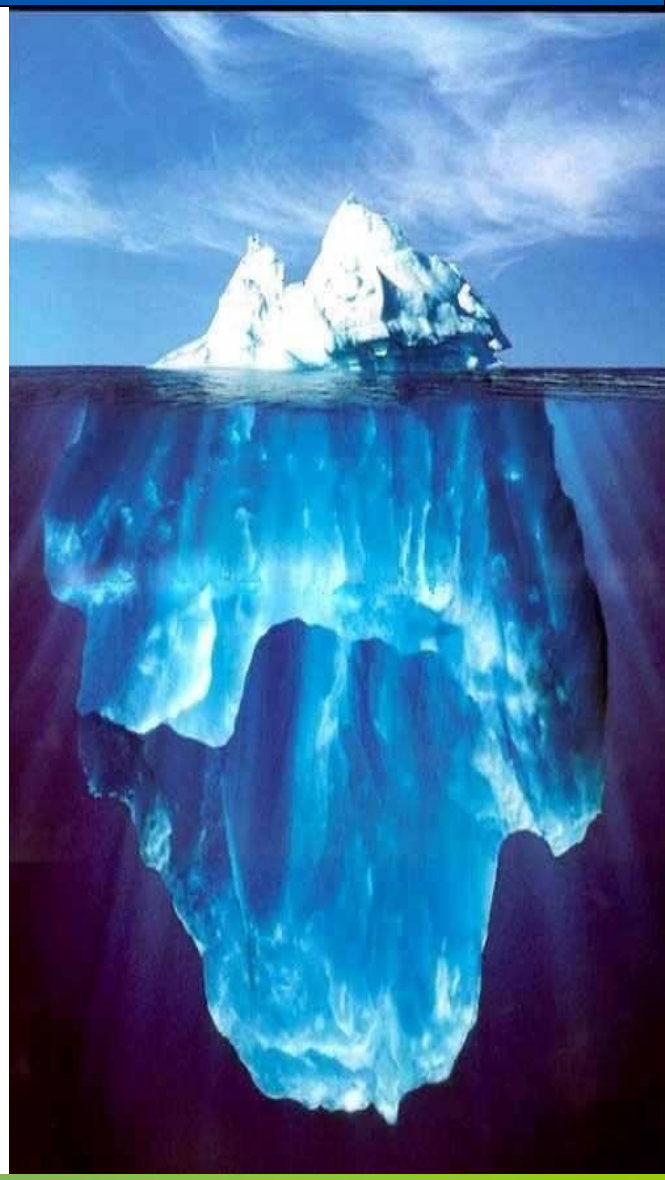
林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：http://dblab.xmu.edu.cn/linziyu

提纲



高校大数据课程
公 共 服 务 平 台

百度搜索厦门大学数据库实验室网站访问平台

6.1 DataSet编程模型

总体而言，Flink批处理程序的基本运行流程包括以下4个步骤：

- 创建执行环境；
- 创建数据源；
- 指定对数据进行的转换操作；
- 指定数据计算的输出结果方式。

上面第1步中创建批处理执行环境的方式如下：

```
val env = ExecutionEnvironment.getExecutionEnvironment
```

此外，还需要在pom.xml文件中引入flink-scala_2.12依赖库，具体如下：

```
<dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-scala_2.12</artifactId>
        <version>1.11.2</version>
</dependency>
```

6.2 数据源

6.2.1 文件类数据源
6.2.2 集合类数据源
6.2.3 通用类数据源
6.2.4 第三方文件系统

6.2.1 文件类数据源

Flink提供了从文件中读取数据生成DataSet的多种方法，具体如下：
•readTextFile(path)：逐行读取文件并将文件内容转换成DataSet类型数据集；
•readTextFileWithValue(path)：读取文本文件内容，并将文件内容转换成DataSet[StringValue]类型数据集。该方法与readTextFile(String)不同的是，其泛型是StringValue，是一种可变的String类型，通过StringValue存储文本数据可以有效降低String对象创建数量，减小垃圾回收的压力；
•readCsvFile(path)：解析以逗号(或其他字符)分隔字段的文件，返回元组或POJO对象；
•readSequenceFile(Key, Value, path)：读取SequenceFile，以Tuple2<Key, Value>类型返回。

以readTextFile(path)为例，可以使用如下语句读取文本文件内容：

```
val dataSet : DataSet[String] = env.readTextFile("file:///home/hadoop/word.txt")
```

6.2.1 文件类数据源

假设有一个CSV格式文件sales.csv，内容如下：

```
transactionId,customerId,itemId,amountPaid
111,1,1,100.0
112,2,2,505.0
113,1,3,510.0
114,2,4,600.0
115,3,2,500.0
```

6.2.1 文件类数据源

则可以使用如下程序读取该CSV文件：

```
package cn.edu.xmu.dblab

import org.apache.flink.api.scala.ExecutionEnvironment
import org.apache.flink.api.scala._

object ReadCSVFile{
  def main(args: Array[String]): Unit = {
    val bEnv = ExecutionEnvironment.getExecutionEnvironment
    val filePath="file:///home/hadoop/sales.csv"
    val csv = bEnv.readCsvFile[SalesLog](filePath,ignoreFirstLine = true)
    csv.print()
  }
  case class
SalesLog(transactionId:String,customerId:String,itemId:String,amountP
aid:Double)
}
```

6.2.1 文件类数据源

程序的执行结果如下：

SalesLog(111,1,1,100.0)
SalesLog(112,2,2,505.0)
SalesLog(113,1,3,510.0)
SalesLog(114,2,4,600.0)
SalesLog(115,3,2,500.0)

6.2.2 集合类数据源

Flink提供了fromCollection()、fromElements()和generateSequence()等方法，来构建集合类数据源，具体如下：

•fromCollection()：从集合中创建DataSet数据集，集合中的元素数据类型相同；

•fromElements()：从给定数据元素序列中创建DataSet数据集，且所有的数据对象类型必须一致；

•generateSequence()：指定一个范围区间，然后在区间内部生成数字序列数据集,由于是并行处理的，所以最终的顺序不能保证一致。

```
val myArray = Array("hello world","hadoop spark flink")
val collectionSet = env.fromCollection(myArray)
val dataSet = env.fromElements("hadoop","spark","flink")
val numSet = env.generateSequence(1,10)
```

6.2.3 通用类数据源

这里以Flink内置的JDBCInputFormat类为实例，介绍通用类数据源的用法。

假设已经在Linux系统中安装了MySQL数据库，在Linux终端中执行如下命令启动MySQL：

```
$ mysql -u root -p
```

输入数据库登录密码以后，就可以启动MySQL了，然后，执行如下命令创建数据库，并添加数据：

```
$ create database flink
$ use flink
$ create table student(sno char(8),cno char(2),grade int);
$ insert into student values('95001','1',96);
$ insert into student values('95002','1',94);
```

6.2.3 通用类数据源

新建代码文件InputFromMySQL.scala，内容如下：

```
package cn.edu.xmu.dblab

import org.apache.flink.api.common.typeinfo.BasicTypeInfo
import org.apache.flink.api.java.io.jdbc.JDBCInputFormat
import org.apache.flink.api.java.typeutils.RowTypeInfo
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._

object InputFromMySQL{
  def main(args: Array[String]): Unit = {

    //创建执行环境
    val env = ExecutionEnvironment.getExecutionEnvironment
```

6.2.3 通用类数据源

```
//使用JDBC输入格式从关系数据库读取数据
  val inputMySQL = env.createInput(JDBCInputFormat.buildJDBCInputFormat()
    //数据库连接驱动名称
    .setDrivername("com.mysql.jdbc.Driver")
    //数据库连接驱动名称
    .setDBUrl("jdbc:mysql://localhost:3306/flink")
    //数据库连接用户名
    .setUsername("root")
    //数据库连接密码
    .setPassword("123456")
    //数据库连接查询SQL
    .setQuery("select sno,cno,grade from student")
    //字段类型、顺序和个数必须与SQL保持一致
    .setRowTypeInfo(new RowTypeInfo(BasicTypeInfo.STRING_TYPE_INFO,
      BasicTypeInfo.STRING_TYPE_INFO, BasicTypeInfo.INT_TYPE_INFO))
    .finish()
  )
  inputMySQL.print()
 }
}
```

6.2.3 通用类数据源

新建pom.xml文件，在里面添加与访问MySQL相关的依赖包，内容如下：

```
<project>
    <groupId>cn.edu.xmu.dblab</groupId>
    <artifactId>simple-project</artifactId>
    <modelVersion>4.0.0</modelVersion>
    <name>Simple Project</name>
    <packaging>jar</packaging>
    <version>1.0</version>
    <repositories>
        <repository>
            <id>alimaven</id>
            <name>aliyun maven</name>
            <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        </repository>
    </repositories>
```

6.2.3 通用类数据源

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-scala_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-scala_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
```

6.2.3 通用类数据源

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.40</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-connector-jdbc_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
  </dependencies>
```

6.2.3 通用类数据源

```
<build>
    <plugins>
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.4.6</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

6.2.3 通用类数据源

```
<plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>3.0.0</version>
            <configuration>
               <descriptorRefs>
                  <descriptorRef>jar-with-dependencies</descriptorRef>
               </descriptorRefs>
            </configuration>
            <executions>
               <execution>
                  <id>make-assembly</id>
                  <phase>package</phase>
                  <goals>
                     <goal>single</goal>
                  </goals>
               </execution>
            </executions>
         </plugin>
      </plugins>
   </build>
</project>
```

6.2.3 通用类数据源

使用Maven工具对程序进行编译打包，然后，提交到Flink中运行（请确认Flink已经启动）。运行结束以后，可以在屏幕上看到如下的输出结果：

```
95001,1,96
95002,1,94
```

6.2.4 第三方文件系统

Flink通过FileSystem类来抽象自己的文件系统，这个抽象提供了各类文件系统实现的通用操作和最低保证。每种数据源（比如HDFS、S3、Alluxio、XtreemFS、FTP等）可以继承和实现FileSystem类，将数据从各个系统读取到Flink中。

6.2.4 第三方文件系统

DataSet API中内置了HDFS数据源，这里给出一个读取HDFS文件系统的一个实例，代码如下：

```
package cn.edu.xmu.dblab

import org.apache.flink.api.scala.ExecutionEnvironment

object ReadHDFS{
  def main(args: Array[String]): Unit = {

    //获取执行环境
    val env = ExecutionEnvironment.getExecutionEnvironment

    //创建数据源
val inputHDFS = env.readTextFile("hdfs://localhost:9000/word.txt")

    //打印输出
inputHDFS.print()
  }
}
```

6.2.4 第三方文件系统

在pom.xml文件中，需要添加与访问HDFS相关的依赖包，内容如下：

```
<project>
    <groupId>cn.edu.xmu.dblab</groupId>
    <artifactId>simple-project</artifactId>
    <modelVersion>4.0.0</modelVersion>
    <name>Simple Project</name>
    <packaging>jar</packaging>
    <version>1.0</version>
    <repositories>
        <repository>
            <id>alimaven</id>
            <name>aliyun maven</name>
            <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        </repository>
    </repositories>
```

6.2.4 第三方文件系统

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-scala_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-scala_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients_2.12</artifactId>
        <version>1.11.2</version>
    </dependency>
```

6.2.4 第三方文件系统

```
<dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.1.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>3.1.3</version>
    </dependency>
  </dependencies>
```

6.2.4 第三方文件系统

```
<build>
    <plugins>
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.4.6</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

6.2.4 第三方文件系统

```
<plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>3.0.0</version>
            <configuration>
               <descriptorRefs>
                  <descriptorRef>jar-with-dependencies</descriptorRef>
               </descriptorRefs>
            </configuration>
            <executions>
               <execution>
                  <id>make-assembly</id>
                  <phase>package</phase>
                  <goals>
                     <goal>single</goal>
                  </goals>
               </execution>
            </executions>
         </plugin>
      </plugins>
   </build>
</project>
```

6.2.4 第三方文件系统

使用Maven工具对程序进行编译打包。
为了让Flink能够顺利访问HDFS，需要修改环境变量。如果在学习第5章内容时已经完成了修改，这里就不需要重复操作；如果还没有修改，执行如下命令修改环境变量：

```
$ vim ~/.bashrc
```

该文件中原有配置信息仍然保留，然后在.bashrc文件中继续增加如下配置信息：

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
export HADOOP_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

执行如下命令使得环境变量设置生效：

```
$ source ~/.bashrc
```

使用flink run命令把ReadHDFS程序提交到Flink中运行（请确认Flink和Hadoop已经启动），如果运行成功，就可以在屏幕上看到"hdfs://localhost:9000/word.txt"文件里面的内容了。

6.3 数据转换

表6-1 DataSet API中常用的算子

算子	功能
map	输入一个元素，然后返回一个元素，中间可以做一些清洗转换等操作
flatMap	输入一个元素，可以返回零个、一个或者多个元素
mapPartition	类似map，一次处理一个分区的数据
filter	过滤函数，对传入的数据进行判断，符合条件的数据会被留下
reduce	对数据进行聚合操作，结合当前元素和上一次reduce返回的值进行聚合操作，然后返回一个新的值
aggregate	聚合操作，包括sum、max、min等
distinct	返回一个数据集中去重之后的元素
join	内连接
cross	获取两个数据集的笛卡尔积
union	返回两个数据集的总和，数据类型需要一致
rebalance	在数据分区时，根据轮询调度算法，将数据均匀地分发给下一级节点
partitionByHash	在数据分区时，根据元组的某个属性域进行散列分区
partitionByRange	在数据分区时，根据某个属性的范围进行分区

6.3 数据转换

6.3.1 数据处理类算子
6.3.2 聚合操作类算子
6.3.3 多表关联类算子
6.3.4 集合操作类算子
6.3.5 分区操作类算子

6.3.1 数据处理类算子

1.map

map(func)操作将每个元素传递到函数func中，并将结果返回为一个新的数据集。例如：

```
val dataSet: DataSet[Int] = env.fromElements(1,2,3,4,5)
val mapDS: DataSet[Int] = dataSet.map(x=>x+10)
```

2.flatMap

flatMap(func)与map()相似，但每个输入元素都可以映射到0或多个输出结果。例如：

```
val dataSet: DataSet[String] = env.fromElements("Hadoop is
good","Flink is fast","Flink is better")
val flatMapDS: DataSet[String] = dataSet.flatMap(line => line.split(" "))
```

6.3.1 数据处理类算子

3.mapPartition

mapPartition的功能和map相似，只是mapPartition操作是在DataSet中基于分区对数据进行处理。

```
val dataSet: DataSet[String] =
env.fromElements("hadoop","spark","flink")
val mapPartitionDS: DataSet[(String,Int)] = dataSet.mapPartition(in =>
in.map( word=> (word,1)) )
```

4.filter

filter(func)操作会筛选出满足函数func的元素，并返回一个新的数据集。例如：

```
val dataSet: DataSet[String] = env.fromElements("Hadoop is good","Flink
is fast","Flink is better")
val filterDS: DataSet[String] = dataSet.filter(line => line.contains("Flink"))
```

6.3.2 聚合操作类算子

1.reduce

reduce算子将DataSet数据集通过传入的用户自定义的函数滚动地进行数据聚合处理，处理以后得到一个新的DataSet。reduce算子可以作用在整个数据集上，也可以作用在分组上。

（1）reduce算子作用在整个数据集上

```
val dataSet: DataSet[Int] = env.fromElements(1,2,3,4,5)
val reduceDS: DataSet[Int] = dataSet.reduce(_+_)
```

（2）reduce算子作用在分组上

可以首先使用groupBy操作将数据集进行分组，然后在每个分组上进行reduce操作。下面是一个具体实例：

6.3.2 聚合操作类算子

```
package cn.edu.xmu.dblab
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._
case class WordCount(word:String, count:Int)
object ReduceOperator{
  def main(args: Array[String]): Unit = {
    //获取执行环境
    val env = ExecutionEnvironment.getExecutionEnvironment
//创建数据源
val wordCountDS: DataSet[WordCount] =
      env.fromElements(
        WordCount("spark",1),
        WordCount("spark",2),
        WordCount("flink",1),
        WordCount("flink",1))
    //设定转换操作
val resultDS: DataSet[WordCount]  = wordCountDS
        .groupBy("word")
        .reduce((w1,w2)=>new WordCount(w1.word,w1.count+w2.count))
    resultDS.print()
  }
}
```

6.3.2 聚合操作类算子

2.aggregate

aggregate算子将一组元素值合并成单个值，可以作用在整个数据集上，也可以作用在分组上。Aggregate函数包括求和（SUM）、求最小值（MIN）和求最大值（MAX）。多个aggregate函数之间用and连接。

（1）作用于整个数据集上

```
package cn.edu.xmu.dblab

import org.apache.flink.api.java.aggregation.Aggregations
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._

object AggregationOperator{
  def main(args: Array[String]): Unit = {

//获取执行环境
val env = ExecutionEnvironment.getExecutionEnvironment
```

6.3.2 聚合操作类算子

```
//创建数据源
val input: DataSet[(Int,String,Double)] = env.fromElements(
      (1,"spark",3.0),
      (1,"spark",4.0),
      (1,"spark",4.0),
      (1,"flink",5.0),
      (1,"flink",6.0),
)

//指定针对数据集的转换操作
    val output: DataSet[(Int,String,Double)] = input
      .aggregate(Aggregations.SUM,0)
      .and(Aggregations.MIN,2)

    //打印输出
    output.print()
  }
}
```

该程序的输出结果如下：

(5,flink,3.0)

6.3.2 聚合操作类算子

（2）作用于分组上

```
package cn.edu.xmu.dblab
import org.apache.flink.api.java.aggregation.Aggregations
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._
object AggregationOperator{
  def main(args: Array[String]): Unit = {

    //获取执行环境
val env = ExecutionEnvironment.getExecutionEnvironment

//创建数据源
    val input: DataSet[(Int,String,Double)] = env.fromElements(
      (1,"spark",3.0),
      (1,"spark",4.0),
      (1,"spark",4.0),
      (1,"flink",5.0),
      (1,"flink",6.0),
)
```

6.3.2 聚合操作类算子

```
//指定针对数据集的转换操作
    val output: DataSet[(Int,String,Double)] = input
      .groupBy(1)
.aggregate(Aggregations.SUM,0)
      .and(Aggregations.MIN,2)

    //打印输出
output.print()
  }
}
```

输出结果如下：

```
(2,flink,5.0)
(3,spark,3.0)
```

6.3.2 聚合操作类算子

3.distinct

distinct算子用于去除DataSet中所有重复的记录，实例如下：

```
val dataSet: DataSet[String] =
env.fromElements("hadoop","hadoop","spark","flink")
val distinctDS: DataSet[String] = dataSet.distinct()
```

6.3.3 多表关联类算子

1.join

连接(join)操作根据指定的条件关联两个数据集，然后根据选择的字段形成一个数据集。

（1）不带连接函数的形式

```
val input1: DataSet[(Int,String)] = env.fromElements((1,"spark"),(2,"flink"))
val input2: DataSet[(String,Int)] = env.fromElements(("spark",1),("flink",2))
val result = input1.join(input2).where(0).equalTo(1)
result.print()
```

程序打印输出的结果如下：

((1,spark),(spark,1))
((2,flink),(flink,2))

6.3.3 多表关联类算子

（2）带JoinFunction连接函数的形式

Flink支持在连接的过程中指定自定义的JoinFunction，函数的输入为左边数据集中的数据元素和右边数据集中的数据元素所组成的元组，并返回一个经过计算处理后的数据。

```
package cn.edu.xmu.dblab
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._
case class Student(name: String, lesson: String, score: Int)
object JoinFunctionTest{
  def main(args: Array[String]): Unit = {

    //获取执行环境
val env = ExecutionEnvironment.getExecutionEnvironment

//设置程序并行度
env.setParallelism(1)
```

6.3.3 多表关联类算子

```
//创建数据源
    val students: DataSet[Student] = env
      .fromElements(Student("xiaoming","computer",90),Student("zhangmei
","english",94))

//指定针对数据源的转换操作
val weights: DataSet[(String,Double)] =
env.fromElements(("computer",0.7),("english",0.4))
    val weightedScores =
students.join(weights).where("lesson").equalTo(0){
      (left,right) => (left.name,left.lesson,left.score*right._2)
}

//打印输出
    weightedScores.print()
  }
}
```

该程序的输出结果如下：

```
(xiaoming,computer,62.99999999999999)
(zhangmei,english,37.6)
```

6.3.3 多表关联类算子

（3）带FlatJoinFunction连接函数的形式

JoinFunction和FlatJoinFunction的关系，和Map与FlatMap的关系是类似的。FlatJoinFunction可以返回一个或者多个元素，也可以不返回任何结果。实例如下：

```
package cn.edu.xmu.dblab
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._

case class Student(name: String, lesson: String, score: Int)
object FlatJoinFunctionTest{
  def main(args: Array[String]): Unit = {

    //获取执行环境
val env = ExecutionEnvironment.getExecutionEnvironment

//设置程序并行度
env.setParallelism(1)
```

6.3.3 多表关联类算子

```
 //指定针对数据集的转换操作
    val weights: DataSet[(String,Double)] =
env.fromElements(("computer",0.7),("english",0.4))
    val weightedScores =
students.join(weights).where("lesson").equalTo(0){
      (left,right,out:Collector[(String,String,Double)]) =>
        if (right._2>0.5) out.collect(left.name,left.lesson,left.score*right._2)
}
//打印输出
    weightedScores.print()
  }
}
```

该程序的输出结果如下：

(xiaoming,computer,62.99999999999999)

6.3.3 多表关联类算子

2.cross

cross算子将两个数据集合并成一个数据集，返回被连接的两个数据集所有数据行的笛卡尔积，返回的数据行数等于第一个数据集中符合查询条件的数据行数乘以第二个数据集中符合查询条件的数据行数。

```
package cn.edu.xmu.dblab
import org.apache.flink.api.scala.{DataSet, ExecutionEnvironment}
import org.apache.flink.api.scala._
import org.apache.flink.util.Collector

case class Coord(id:Int,x:Int,y:Int)
object CrossOperator {
  def main(args: Array[String]): Unit = {
    //获取执行环境
val env = ExecutionEnvironment.getExecutionEnvironment

//设置程序并行度
env.setParallelism(1)
```

6.3.3 多表关联类算子

```
//创建数据源
    val coords1: DataSet[Coord] = env.fromElements(Coord(1,4,5),Coord(2,6,7))
val coords2:DataSet[Coord] = env.fromElements(Coord(3,8,9),Coord(4,10,11))

//指定针对数据集的转换操作
    val distances = coords1.cross(coords2){
      (c1,c2) => val dist = math.sqrt(math.pow(c1.x-c2.x,2)+math.pow(c1.y-c2.y,2))
      (c1.id,c2.id,dist)
}

//打印输出
    distances.print()
  }
}
```

6.3.3 多表关联类算子

该程序的输出结果如下：

(1,3,5.65685424949238 1)
(2,3,2.828427124746190 3)
(1,4,8.48528137423857)
(2,4,5.65685424949238 1)

6.3.4 集合操作类算子

比较常用的集合类算子是union，主要用于合并两个DataSet数据集，两个数据集的数据元素格式必须相同，多个数据集可以连续合并。实例如下：

```
val dataSet1: DataSet[(Int,String)] = env.fromElements((1,"spark"),(2,"flink"))
val dataSet2: DataSet[(Int,String)] = env.fromElements((3,"hadoop"),(4,"storm"))
val result: DataSet[(Int,String)] = dataSet1.union(dataSet2)
```

6.3.5 分区操作类算子

1.Rebalance模式

该模式根据轮询调度算法，将数据均匀地分发给下一级节点。其用法如下：

```
val dataSet: DataSet[String] = ...
val result = dataSet.rebalance().map{...}
```

2.Hash-Partition模式

该模式根据元组的某个属性域进行散列分区。其用法如下：

```
val dataSet: DataSet[(String,Int)] = ...
val result = dataSet.partitionByHash(0).mapPartition{...}
```

3.Range-Partition模式

该模式根据某个属性的范围进行分区。其用法如下：

```
val dataSet: DataSet[(String,Int)] = ...
val result = dataSet.partitionByRange(0).mapPartition{...}
```

6.4 数据输出

1.基于文件的输出接口

Flink支持多种存储设备上的文件，包括本地文件和HDFS文件等，同时，Flink支持多种文件的存储格式，包括文本文件、CSV文件等。这里介绍文本文件的输出方法。
把数据集输出到本地文件的方法如下：

```
val dataSet: DataSet[(Int,String)] = env.fromElements((1,"spark"),(2,"flink"))
    dataSet.writeAsText("file:///home/hadoop/output")
    env.execute()
```

需要注意的是，必须调用execute()方法，否则无法让数据集输出到文件。

6.4 数据输出

把数据集输出到HDFS的方法如下：

```
val dataSet: DataSet[(Int,String)] = env.fromElements((1,"spark"),(2,"flink"))
dataSet.writeAsText("hdfs://localhost:9000/output")
env.execute()
```

在使用Maven工具编译打包程序时，pom.xml文件的内容和6.2.4节中的相同，同时，也要参照6.2.4节中的方法完成环境变量的配置，才能够顺利写入HDFS。

6.4 数据输出

2.通用输出接口

在DataSet API中，可以使用自定义的OutputFormat方法来定义与具体存储系统对应的OutputFormat，例如JDBCOutputFormat和HadoopOutputFormat等。

下面是写入MySQL数据库的实例：

```
package cn.edu.xmu.dblab

import org.apache.flink.api.java.io.jdbc.{JDBCInputFormat, JDBCOutputFormat}
import org.apache.flink.api.java.typeutils.RowTypeInfo
import org.apache.flink.api.scala.{ExecutionEnvironment, _}
import org.apache.flink.types.Row
import org.apache.flink.api.scala.DataSet
import scala.collection.mutable.ArrayBuffer
```

6.4 数据输出

```
object WriteMySQL {
  def main(args: Array[String]): Unit = {
val env = ExecutionEnvironment.getExecutionEnvironment

val arr = new ArrayBuffer[Row]()

    val row1 = new Row(3)
    row1.setField(0, "95001")
    row1.setField(1, "2")
    row1.setField(2, 94)

    val row2 = new Row(3)
    row2.setField(0, "95002")
    row2.setField(1, "2")
    row2.setField(2, 88)

    arr.+=(row1)
    arr.+=(row2)
```

6.4 数据输出

```
 val data: DataSet[Row] = env.fromCollection(arr)
data.output(JDBCOutputFormat.buildJDBCOutputFormat()
  // 数据库连接驱动名称
  .setDrivername("com.mysql.jdbc.Driver")
  // 数据库连接地址
  .setDBUrl("jdbc:mysql://localhost:3306/flink")
  // 数据库连接用户名
  .setUsername("root")
  // 数据库连接密码
  .setPassword("123456")
  // 数据库插入SQL
  .setQuery("insert into student (sno,cno,grade) values(?,?,?)")
  .finish())

env.execute("insert data to mysql")
System.out.println("MySQL写入成功！ ")

  }
}
```

6.4 数据输出

使用Maven工具对该程序进行编译打包时，pom.xml文件的内容和6.2.3节中的相同。打包成功以后，提交到Flink中运行，就会在MySQL数据库中写入两条记录。

3.客户端输出

DataSet API提供的print()方法就属于客户端输出。需要注意的是，当调用print()方法把数据集输出到屏幕上以后，不能再调用execute()方法，否则会报错。

6.5迭代计算

所谓迭代计算是指给定一个初值，用所给的算法公式计算初值得到一个中间结果，然后将中间结果作为输入参数进行反复计算，在满足一定条件的时候得到计算结果。
迭代计算在批量数据处理过程中有着非常广泛的应用，比如机器学习和图计算等。DataSet API对迭代计算功能的支持相对比较完善，在性能上较其他分布式计算框架也具有明显的优势。Flink中的迭代计算主要包括两种模式，即全量迭代计算（Bulk Iteration）和增量迭代计算（Delt Iteration）。

6.5.1全量迭代
6.5.2增量迭代

6.5.1全量迭代

全量迭代会将整个数据输入，经过一定的迭代次数，最终得到结果。图6-1给出了全量迭代的基本过程，具体如下：

•Iteration Input（迭代输入）：是初始输入值或者上一次迭代计算的结果；

•Step Function（步骤函数）：步骤函数在每一次迭代中都会被执行，它由一系列算子组成，比如map、flatMap、join等。

•Next Partial Solution（中间结果）：每一次迭代计算的结果，会被发送到下一次迭代计算中。

•Iteration Result（迭代结果）：最后一次迭代输出的结果，会被输出到DataSink或者发送到下游处理（作为下一个算子的输入）。

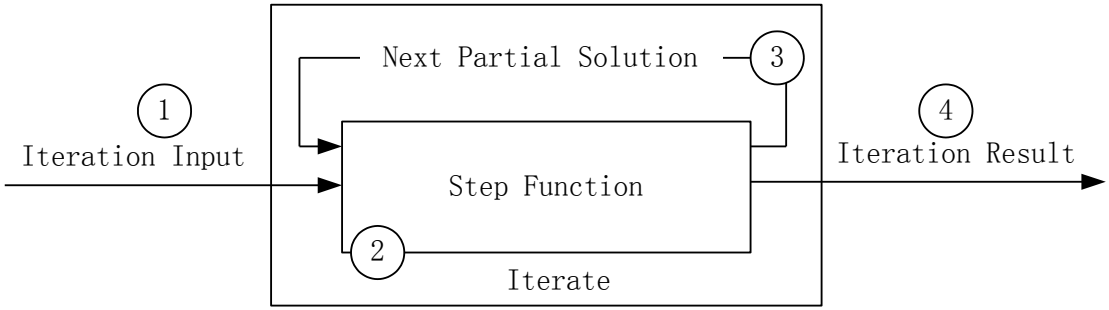


图6-1 全量迭代执行过程

6.5.1全量迭代

迭代的结束条件是：

•达到最大迭代次数：不需要任何其他条件，迭代执行到最大迭代次数就会停止；

•自定义收敛条件：允许用户自定义聚合函数和收敛条件。例如将终止条件设置为当Sum函数统计结果小于零则终止迭代。

6.5.1全量迭代

下面是一个实例（如图6-2所示），演示了通过迭代计算把数据集中的每个数都进行增加操作，具体如下：

•Iteration Input（迭代输入）：从一个数据源中读取初始输入数据，初始数据中包含了5个整数；

•Step Function（步骤函数）：步骤函数是一个map算子，它会把每个整数增加1；

•Next Partial Solution（中间结果）：步骤函数的输出结果又会成为map算子的输入；

•Iteration Result（迭代结果）：经过10次迭代，每个初始整数都增加了10。

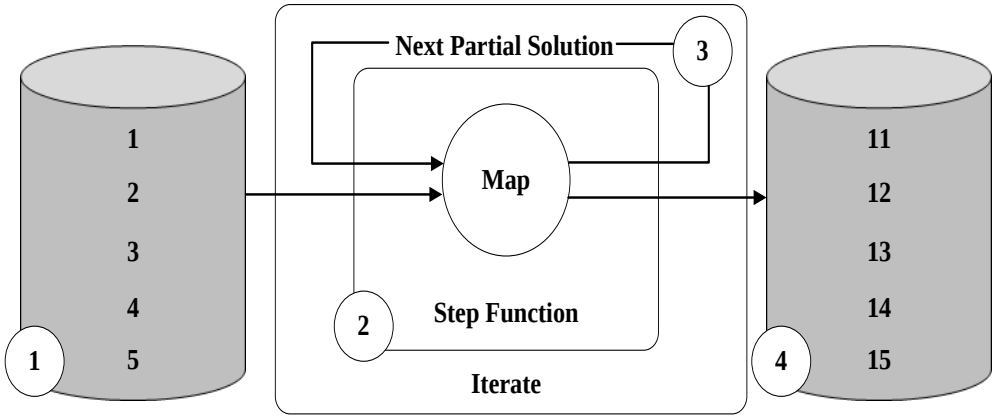


图6-2 一个全量迭代的实例

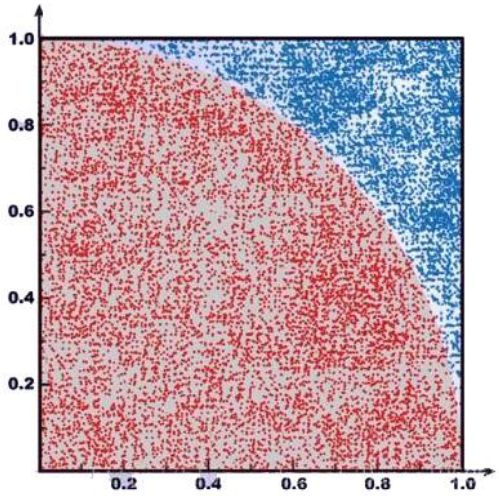
6.5.1全量迭代

第1次迭代	第2次迭代		第10次迭代
map(1) -> 2	map(2) -> 3	...	map(10) -> 11
map(2) -> 3	map(3) -> 4	...	map(11) -> 12
map(3) -> 4	map(4) -> 5	...	map(12) -> 13
map(4) -> 5	map(5) -> 6	...	map(13) -> 14
map(5) -> 6	map(6) -> 7	...	map(14) -> 15

图　迭代过程中数据集的变化情况

6.5.1全量迭代

这里介绍一个全量迭代的实例——使用蒙洛卡特方法来计算圆周率。
蒙洛卡特方法的核心思想是：假设有一个半径为1的圆，它的面积$S=Pi*R^2=Pi$，所以，我们只要计算出这个圆的面积就可以计算出圆周率了。这里我们可以在一个边长为1的正方形中计算圆的四分之一扇形的面积，这样扇形的面积的4倍就是整个圆的面积了。如何计算扇形的面积呢？可以使用概率的方法，假设在这个正方形中有n个点，其中，有m个点落在了扇形中，因此，$S_{扇形}:S_{正方形}=m:n$，这样就可以计算出扇形的面积（如图6-4所示），最终就可以计算出圆周率了。



6.5.1全量迭代

下面是使用蒙洛卡特方法计算圆周率的程序代码：

```
package cn.edu.xmu.dblab
import org.apache.flink.api.scala._
object ComputePi{
  def main(args: Array[String]): Unit = {

    //创建执行环境
    val env = ExecutionEnvironment.getExecutionEnvironment

    // 创建初始数据集
    val initial = env.fromElements(0)
```

6.5.1全量迭代

```
 //执行迭代计算
val count = initial.iterate(10000) { iterationInput: DataSet[Int] =>
      val result = iterationInput.map { i =>
        val x = Math.random()
        val y = Math.random()
        i + (if (x * x + y * y < 1) 1 else 0)
      }
      result
    }

    //计算圆周率
val result = count map { c => c / 10000.0 * 4 }

//打印输出
    result.print()
  }
}
```

6.5.2增量迭代

如图6-5所示，增量迭代并不是每次去迭代全量数据，而是有两个数据集——WorkSet和SolutionSet，每次输入这两个数据集进行迭代计算，然后对WorkSet进行迭代运算并且不断更新SolutionSet，直到达到迭代次数或者WorkSet为空，然后输出迭代计算结果。

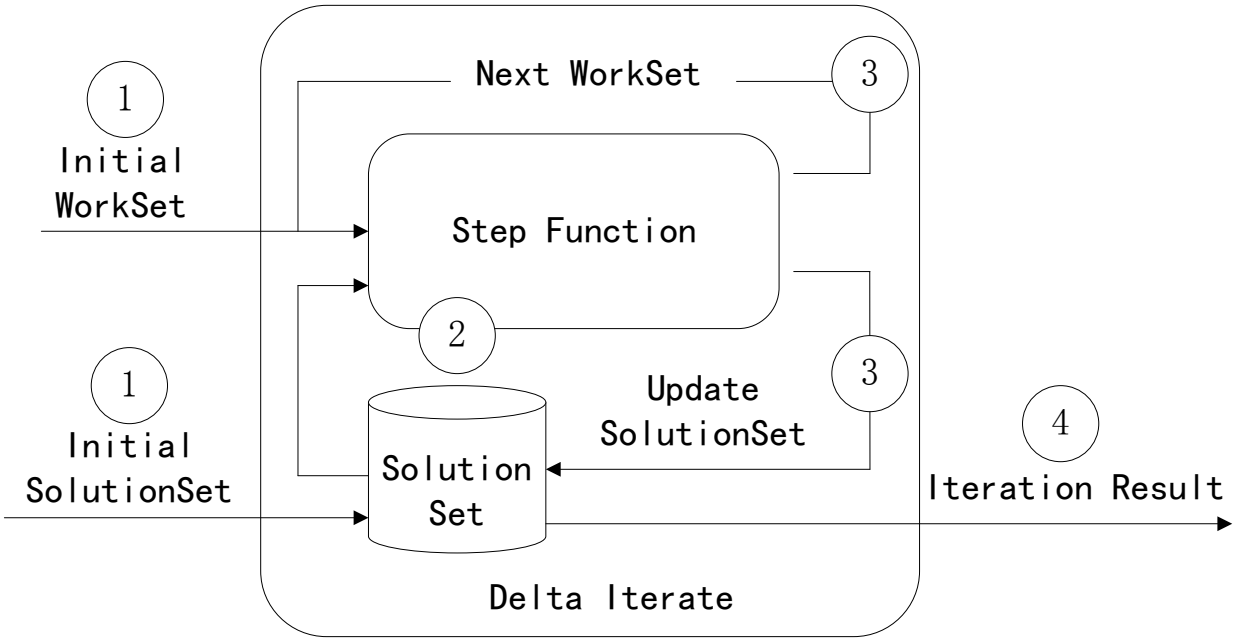


图6-5 增量迭代执行过程

6.5.2增量迭代

增量迭代主要包括以下步骤：
•Iteration Input（迭代输入）：读取初始WorkSet和初始SolutionSet作为第一次迭代计算的输入；
•Step Function（步骤函数）：在每次迭代过程中使用的计算方法，可以是类似于map、flatMap、join等方法；
•Next Workset/Update Solution Set（中间结果）：Next WorkSet用于驱动迭代计算，会被反馈到下一次迭代计算中，而且SolutionSet将被不断更新。两个数据集都可以被步骤函数中的算子更新；
•Iteration Result（迭代结果）：最后一次迭代计算的输出，会被输出到DataSink或者发送到下游处理（作为下一个算子的输入）。

增量迭代的终止条件可以指定为：
•WorkSet为空：如果下一次迭代的输入WorkSet为空，则终止迭代；
•最大迭代次数：当计算次数超过指定迭代的最大次数，则终止迭代。

6.5.2增量迭代

下面是一个增量迭代的用法实例：

```
//读取初始数据集
val initialSolutionSet: DataSet[(Long, Double)] = // [...]
val initialWorkset: DataSet[(Long, Double)] = // [...]

//设置迭代次数
val maxIterations = 100
val keyPosition = 0
```

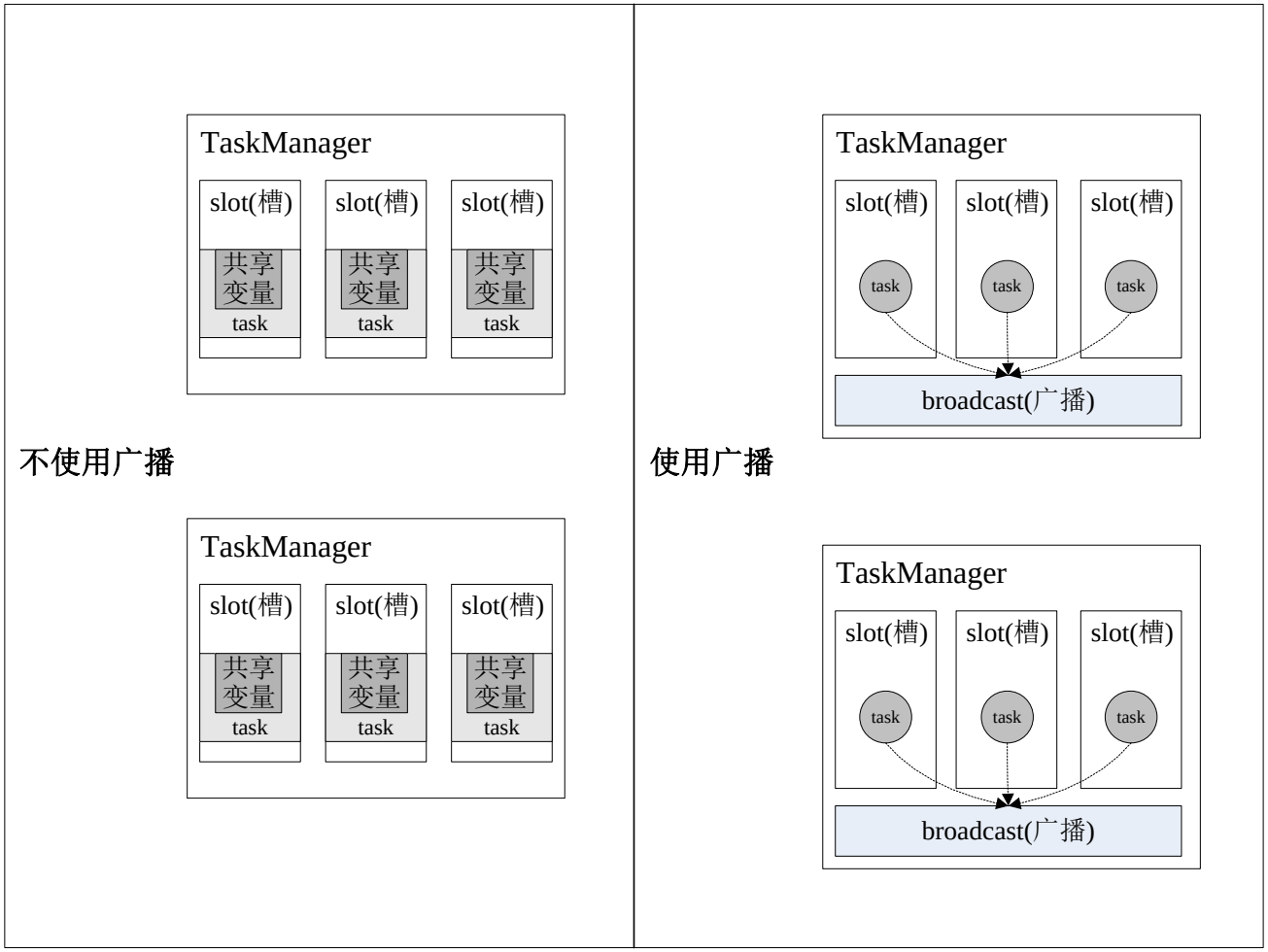
6.5.2增量迭代

```
//应用增量迭代方法
val result = initialSolutionSet.iterateDelta(initialWorkset, maxIterations,
Array(keyPosition)) {
  (solution, workset) =>
    val candidateUpdates = workset.groupBy(1).reduceGroup(new
ComputeCandidateChanges())
    val deltas = candidateUpdates.join(solution).where(0).equalTo(0)(new
CompareChangesToCurrent())

    val nextWorkset = deltas.filter(new FilterByThreshold())

    (deltas, nextWorkset)
}

//输出迭代计算的结果
result.writeAsCsv(outputPath)

env.execute()
```

6.6 广播变量

广播变量可以理解为是一个公共的共享变量（如图6-6所示），通过使用广播变量，可以把一个数据集广播出去，然后不同的任务在节点上都能够获取到，并在每个节点上只会存在一份，而不是在每个并发线程中存在。如果不使用广播变量，则在每个节点中的每个任务中都需要拷贝一份数据集，这样会比较浪费内存。

6.6 广播变量


TaskManager
slot(槽)
slot(槽)
slot(槽)
共享变量
task
不使用广播
TaskManager
slot(槽)
slot(槽)
slot(槽)
共享变量
task
TaskManager
slot(槽)
slot(槽)
slot(槽)
task
broadcast(广播)
使用广播
TaskManager
slot(槽)
slot(槽)
slot(槽)
task
broadcast(广播)

6.6 广播变量

可以使用DataSet API提供的withBroadcastSet(DataSet,String)方法来定义广播变量，这个方法包含了两个参数，其中，第1个餐数据是需要广播的DataSet数据集，需要在广播之前创建该数据集，第2个参数是广播变量的名称。DataSet API支持在RichFunction接口中通过RuntimeContext读取到广播变量。首先在RichFunction中实现Open()方法，然后调用getRuntimeContext()方法获取应用的RuntimeContext，接着调用getBroadcastVariable()方法通过广播变量名称获取广播变量。

6.6 广播变量

```
package cn.edu.xmu.dblab

import org.apache.flink.api.common.functions.RichMapFunction
import org.apache.flink.api.scala.ExecutionEnvironment
import org.apache.flink.configuration.Configuration
import scala.collection.mutable.ListBuffer
import org.apache.flink.api.scala._

object BroadcastDemo {
  def main(args: Array[String]): Unit = {
    val env = ExecutionEnvironment.getExecutionEnvironment

    val rawtData = ListBuffer[Tuple2[String,Int]]()
    rawtData.append(("hadoop",48))
    rawtData.append(("spark",42))
    rawtData.append(("flink",46))
val tupleData = env.fromCollection(rawtData)
```

6.6 广播变量

```
//创建需要广播的数据集
    val broadcastData = tupleData.map(x=>{
      Map(x._1->x._2)
    })

    val books = env.fromElements("hadoop","spark","flink")

    val result = books.map(new RichMapFunction[String,String] {

      var listData: java.util.List[Map[String,Int]] = null
      var allMap  = Map[String,Int]()
```

6.6 广播变量

```
override def open(parameters: Configuration): Unit = {
      super.open(parameters)
      //获取广播变量数据集
      this.listData =
getRuntimeContext.getBroadcastVariable[Map[String,Int]]("broadcastMapName")
      val it = listData.iterator()
      while (it.hasNext){
        val next = it.next()
        allMap = allMap.++(next)
      }
    }
    override def map(value: String) = {
      val amount = allMap.get(value).get
      "The amount of "+value+" is:"+amount
    }
  }).withBroadcastSet(broadcastData,"broadcastMapName")

  result.print()
 }
}
```

6.6 广播变量

该程序的输出结果如下：

```
The amount of hadoop is:48
The amount of spark is:42
The amount of flink is:46
```

6.7本章小结

相对于DataStream API而言，DataSet API的应用不是特别广泛，但是，并不代表Flink不擅长批量处理数据。Flink把批数据看成是流数据的特例，因此，可以通过一套引擎来同时处理批数据和流数据。本章对DataSet编程模型做了简要介绍，并阐述了多种不同类型的数据源，包括文件类数据源、集合类数据源、通用类数据源和第三方文件系统等。同时，Flink提供了非常丰富的转换算子，主要包括数据处理类算子、聚合操作类算子、多表关联类算子、集合操作类算子和分区操作类算子等，本章对这些算子的用法进行了介绍。最后，本章还介绍了数据输出、迭代计算和广播变量的相关知识。

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过200万次，累计访问量超过1000万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/

开设大数据专业导论课的优质教材

扫一扫访问教材官网

附录F：《大数据技术原理与应用（第3版）》教材

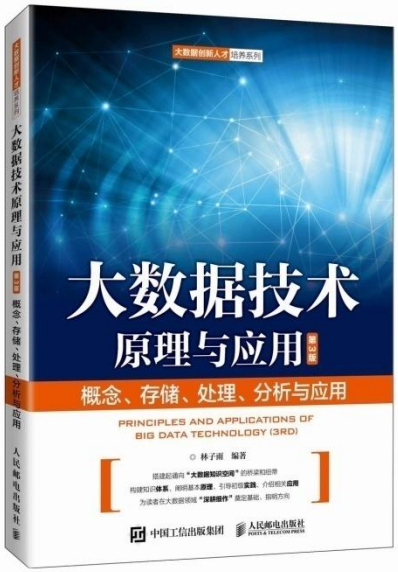
《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社　ISBN:978-7-115-54405-6　定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

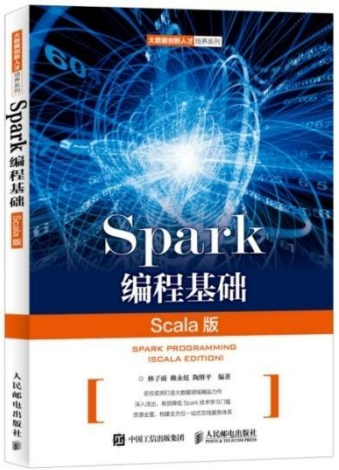
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

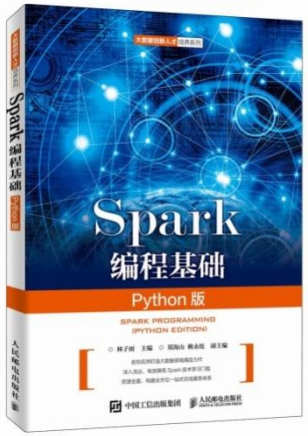
厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3
教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

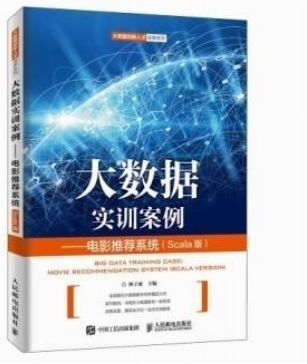
扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

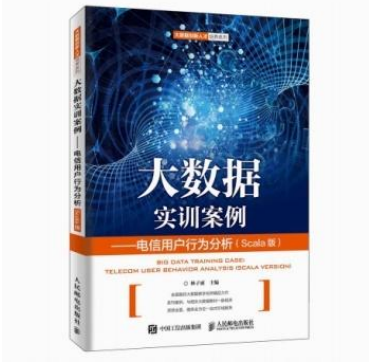
附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）
《电信用户行为分析》（已经于2019年5月出版）
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！
http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页

Department of Computer Science, Xiamen University, 2021