> 不懂的概念写在傍边，然后向别人讲述

RDD数据读取和保存

> Spark的数据读取及数据保存可以从两个维度来作区分：文件格式以及文件系统。
>
> 文件格式分为：Text文件、Json文件、Csv文件、Sequence文件以及Object文件；
>
> 文件系统分为：本地文件系统、HDFS、HBASE以及数据库。

文件类

读写Text文件

Spark的数据读取及数据保存可以从两个维度来作区分：文件格式以及文件系统。文件格式分为：Text文件、Json文件、Csv文件、Sequence文件以及Object文件；文件系统分为：本地文件系统、HDFS、HBASE以及数据库。

（1）数据读取

```
scala>  val hdfsFile = sc.textFile("hdfs://hadoop102:9000/fruit.txt")

hdfsFile: org.apache.spark.rdd.RDD[String] = hdfs://hadoop102:9000/fruit.txt
MapPartitionsRDD[21] at textFile at <console>:24
```

（2）数据保存

```
scala>  hdfsFile.saveAsTextFile("/fruitOut")
```

读写Json文件

> 如果JSON文件中**每一行就是一个JSON记录**，那么可以通过将JSON文件当做文本文件来读取，然后利用相关的JSON库对每一条数据进行JSON解析。

注意：使用RDD读取JSON文件处理很复杂，同时SparkSQL集成了很好的处理JSON文件的方式，所以应用中多是**采用SparkSQL处理JSON文件。**

（1）导入解析json所需的包

```
scala>  import scala.util.parsing.json.JSON
```

（2）上传json文件到HDFS

```
[zhutiansama@hadoop102 spark]$ hadoop fs -put student.json /
```

（3）读取文件

```
scala>  val json = sc.textFile("/student.json")
json: org.apache.spark.rdd.RDD[String] = /student.json MapPartitionsRDD[8] at
textFile at <console>:24
```

（4）解析json数据

```
scala>  val result  =  json.map(JSON.parseFull)
result: org.apache.spark.rdd.RDD[Option[Any]] = MapPartitionsRDD[10] at map at
<console>:27
```

（5）打印

```
scala>  result.collect
res11: Array[Option[Any]] = Array(Some(Map(name -> Michael)),  Some(Map(name ->
Andy,  age -> 30.0)),  Some(Map(name -> Justin,  age -> 19.0)))
```

读写SequenceFile文件

> SequenceFile文件是Hadoop用来存储二进制形式的key-value对而设计的一种平面文件(FlatFile)。Spark有专门用来读取SequenceFile的接口。在SparkContext中，可以调用sequenceFilekeyClass，valueClass。

注意：**SequenceFile文件只针对PairRDD。**

（1）创建一个RDD

```
scala>  val rdd = sc.parallelize(Array((1, 2), (3, 4), (5, 6)))
rdd: org.apache.spark.rdd.RDD[(Int,  Int)] = ParallelCollectionRDD[13] at
parallelize at <console>:24
```

（2）将RDD保存为Sequence文件

```
scala>  rdd.saveAsSequenceFile("file:///opt/module/spark/seqFile")
```

（3）查看该文件

```
[zhutiansama@hadoop102 seqFile]$ pwd
/opt/module/spark/seqFile

[zhutiansama@hadoop102 seqFile]$ cat part-00000
SEQ org.apache.hadoop.io.IntWritable org.apache.hadoop.io.IntWritable
```

（4）读取Sequence文件

```
scala>  val seq = sc.sequenceFile[Int, Int]("file:///opt/module/spark/seqFile")
seq: org.apache.spark.rdd.RDD[(Int,  Int)] = MapPartitionsRDD[18] at
sequenceFile at <console>:24
```

（5）打印读取后的Sequence文件

```
scala>  seq.collect
res14: Array[(Int,  Int)] = Array((1，2)，  (3，4)，  (5，6))
```

读写对象文件

> 对象文件是将对象序列化后保存的文件，采用Java的序列化机制。可以通过`objectFile[k,v](path)`函数接收一个路径，读取对象文件，返回对应的 RDD，也可以通过调用 saveAsObjectFile() 实现对对象文件的输出。因为是序列化所以要指定类型。

（1）创建一个RDD

```
scala> val rdd = sc.parallelize(Array(1,2,3,4))

rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[19] at parallelize at
<console>:24
```

（2）将RDD保存为Object文件

```
scala> rdd.saveAsObjectFile("file:///opt/module/spark/objectFile")
```

（3）查看该文件

```
[atguigu@hadoop102 objectFile]$ pwd
/opt/module/spark/objectFile

[atguigu@hadoop102 objectFile]$ cat part-00000
SEQ!org.apache.hadoop.io.NullWritable"org.apache.hadoop.io.BytesWritableW@`l
```

（4）读取Object文件

```
scala> val objFile = sc.objectFile[(Int)]("file:///opt/module/spark/objectFile")
objFile: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[31] at objectFile at
<console>:24
```

（5）打印读取后的Sequence文件

```
scala> objFile.collect
res19: Array[Int] = Array(1, 2, 3, 4)
```

文件系统类

HDFS上读写文件

> Spark的整个生态系统与Hadoop是完全兼容的，所以对于Hadoop所支持的文件类型或者数据库类型，Spark也同样支持.另外，由于Hadoop的API有新旧两个版本，所以Spark为了能够兼容Hadoop所有的版本，也提供了两套创建操作接口.对于外部存储创建操作而言，hadoopRDD和newHadoopRDD是最为抽象的两个函数接口
>
> 主要包含以下四个参数.

> 1）输入格式(InputFormat):制定数据输入的类型，如TextInputFormat等，新旧两个版本所引用的版本分别是org.apache.hadoop.mapred.InputFormat和org.apache.hadoop.mapreduce.InputFormat(NewInputFormat)
>
> 2）键类型:指定[K，V]键值对中K的类型
>
> 3）值类型:指定[K，V]键值对中V的类型
>
> 4）分区值:指定由外部存储生成的RDD的partition数量的最小值，如果没有指定，系统会使用默认值defaultMinSplits。

注意:其他创建操作的API接口都是为了方便最终的Spark程序开发者而设置的，是这两个接口的高效实现版本.例如，对于textFile而言，只有path这个指定文件路径的参数，其他参数在系统内部指定了默认值。

1.在Hadoop中以压缩形式存储的数据，不需要指定解压方式就能够进行读取，因为Hadoop本身有一个解压器会根据压缩文件的后缀推断解压算法进行解压；

2.如果用Spark从Hadoop中读取某种类型的数据不知道怎么读取的时候，上网查找一个使用map-reduce的时候是怎么读取这种这种数据的，然后再将对应的读取方式改写成上面的hadoopRDD和newAPIHadoopRDD两个类就行了。

Mysql上读写数据

支持通过JavaJDBC访问关系型数据库。需要通过JdbcRDD进行，示例如下:

（1）添加依赖

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.27</version>
</dependency>
```

（2）从Mysql读取数据

```
import java.sql.DriverManager
import org.apache.spark.rdd.JdbcRDD
import org.apache.spark.{SparkConf， SparkContext}

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

  //1.创建spark配置信息
  valsparkConf:SparkConf = new SparkConf()
        .setMaster("local[*]")
        .setAppName("JdbcRDD")

  //2.创建SparkContext
  val sc = new SparkContext(sparkConf)

  //3.定义连接mysql的参数
  val driver = "com.mysql.jdbc.Driver"
  val url = "jdbc:mysql://hadoop102:3306/rdd"
  val userName = "root"
  val passWd = "123456"

  //创建JdbcRDD
```

```
  val rdd = new JdbcRDD(sc, () => {
   Class.forName(driver)
   DriverManager.getConnection(url, userName, passWd)
  },
   "select * from `rddtable` where `id`>=? and `id`<=?;", 1, 10, 1,
   r => (r.getInt(1), r.getString(2))
  )

  //打印最后结果
  println(rdd.count())
  rdd.foreach(println)
  sc.stop()

 }

}
```

(3) 往Mysql写入数据

```
def main(args: Array[String]) {
 val sparkConf = new SparkConf().setMaster("local[2]").setAppName("MysqlApp")
 val sc = new SparkContext(sparkConf)
 val data = sc.parallelize(List("Female", "Male", "Female"))
 data.foreachPartition(insertData)
}


def insertData(iterator: Iterator[String]): Unit = {
Class.forName ("com.mysql.jdbc.Driver").newInstance()
 val conn =
java.sql.DriverManager.getConnection("jdbc:mysql://hadoop102:3306/rdd", "root",
"123456")
iterator.foreach(data => {
val ps = conn.prepareStatement("insert into rddtable(name) values (?)")

ps.setString(1, data)
ps.executeUpdate()

})

}
```

HBase上读写数据

由于 org.apache.hadoop.hbase.mapreduce.TableInputFormat类的实现，
Spark可以通过Hadoop输入格式访问HBase。这个输入格式会返回键值对数据，
其中键的类型为org.apache.hadoop.hbase.io.ImmutableBytesWritable，
而值的类型为org.apache.hadoop.hbase.client.Result。

(1) 添加依赖

```
<dependency>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-server</artifactId>
<version>1.3.1</version>
</dependency>

<dependency>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-client</artifactId>
<version>1.3.1</version>
</dependency>
```

(2) 从HBase读取数据

```
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.hadoop.hbase.util.Bytes

object HBaseSpark {
 def main(args: Array[String]): Unit = {
  //创建spark配置信息
  val sparkConf: SparkConf = new
SparkConf().setMaster("local[*]").setAppName("JdbcRDD")

  //创建SparkContext
  val sc = new SparkContext(sparkConf)

  //构建HBase配置信息
  val conf: Configuration = HBaseConfiguration.create()
  conf.set("hbase.zookeeper.quorum", "hadoop102,hadoop103,hadoop104")
  conf.set(TableInputFormat.INPUT_TABLE, "rddtable")

  //从HBase读取数据形成RDD
  val hbaseRDD: RDD[(ImmutableBytesWritable, Result)] = sc.newAPIHadoopRDD(
   conf,
   classOf[TableInputFormat],
   classOf[ImmutableBytesWritable],
   classOf[Result])

  val count: Long = hbaseRDD.count()
  println(count)

  //对hbaseRDD进行处理
  hbaseRDD.foreach {
   case (_, result) =>
val key: String = Bytes.toString(result.getRow)
val name: String = Bytes.toString(result.getValue(Bytes.toBytes("info"),
Bytes.toBytes("name")))

val color: String = Bytes.toString(result.getValue(Bytes.toBytes("info"),
Bytes.toBytes("color")))
```

```
println("RowKey:" + key + ",Name:" + name + ",Color:" + color)
  }

  //关闭连接
  sc.stop()
 }

}
```

3）往HBase写入

```
def main(args: Array[String]) {
//获取Spark配置信息并创建与spark的连接
val sparkConf = new SparkConf().setMaster("local[*]").setAppName("HBaseApp")
val sc = new SparkContext(sparkConf)

//创建HbaseConf
val conf = HBaseConfiguration.create()
 val jobConf = new JobConf(conf)
 jobConf.setOutputFormat(classOf[TableOutputFormat])
 jobConf.set(TableOutputFormat.OUTPUT_TABLE, "fruit_spark")

//定义往Hbase插入数据的方法
 def convert(triple: (Int, String, Int)) = {
  val put = new Put(Bytes.toBytes(triple._1))
  put.addImmutable(Bytes.toBytes("info"), Bytes.toBytes("name"),
Bytes.toBytes(triple._2))
  put.addImmutable(Bytes.toBytes("info"), Bytes.toBytes("price"),
Bytes.toBytes(triple._3))
  (new ImmutableBytesWritable, put)
 }

//创建一个RDD
val initialRDD = sc.parallelize(List((1,"apple",11), (2,"banana",12),
(3,"pear",13)))

//将RDD内容写到Hbase
val localData = initialRDD.map(convert)
localData.saveAsHadoopDataset(jobConf)
}
```