Spark与Mysql(JdbcRDD)整合开发

如果你需要将RDD写入到Mysql等关系型数据库，请参见《Spark RDD写入RMDB(Mysql)方法二》和《Spark将计算结果写入到Mysql中》文章。

Spark的功能是非常强大，在本博客的文章中，我们讨论了《Spark和Hbase整合》、《Spark和Flume-ng整合》以及《和Hive的整合》。今天我们的主题是聊聊Spark和Mysql的组合开发。

过往记忆博客：http://www.iteblog.com

在Spark中提供了一个JdbcRDD类，该RDD就是读取JDBC中的数据并转换成RDD，之后我们就可以对该RDD进行各种的操作。我们先看看该类的构造函数：

```
JdbcRDD[T: ClassTag](
    sc: SparkContext,
    getConnection: () => Connection,
    sql: String,
    lowerBound: Long,
    upperBound: Long,
    numPartitions: Int,
    mapRow: (ResultSet) => T = JdbcRDD.resultSetToObjectArray _)
```

这个类带了很多参数，关于这个函数的各个参数的含义，我觉得直接看英文就可以很好的理解，如下：

@param getConnection a function that returns an open Connection.
The RDD takes care of closing the connection.
@param sql the text of the query.
The query must contain two ? placeholders for parameters used to partition the results.
E.g. "select title, author from books where ? < = id and id <= ?"
@param lowerBound the minimum value of the first placeholder
@param upperBound the maximum value of the second placeholder
The lower and upper bounds are inclusive.
@param numPartitions the number of partitions.
Given a lowerBound of 1, an upperBound of 20, and a numPartitions of 2,
the query would be executed twice, once with (1, 10) and once with (11, 20)
@param mapRow a function from a ResultSet to a single row of the desired result type(s).
This should only call getInt, getString, etc; the RDD takes care of calling next.
The default maps a ResultSet to an array of Object.

上面英文看不懂？？那好吧，我给你翻译一下。

1、getConnection 返回一个已经打开的结构化数据库连接，JdbcRDD会自动维护关闭。

2、sql
是查询语句，此查询语句必须包含两处占位符?来作为分割数据库ResulSet的参数，例如："select title, author from books where ? < = id and id <= ?"

3、lowerBound, upperBound, numPartitions
分别为第一、第二占位符，partition的个数。例如，给出lowebound 1，upperbound 20，numpartitions 2，则查询分别为(1, 10)与(11, 20)

4、mapRow 是转换函数，将返回的ResultSet转成RDD需用的单行数据，此处可以选择Array或其他，也可以是自定义的case class。默认的是将ResultSet 转换成一个Object数组。

下面我们说明如何使用该类。

```
package scala

import java.sql.DriverManager

import org.apache.spark.SparkContext
import org.apache.spark.rdd.JdbcRDD

/**
 * User: 过往记忆
 * Date: 14-9-10
 * Time: 下午13:16
 * bolg: https://www.iteblog.com
 * 本文地址：https://www.iteblog.com/archives/1113
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
```

```
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */
object SparkToJDBC {

  def main(args: Array[String]) {
    val sc = new SparkContext("local", "mysql")
    val rdd = new JdbcRDD(
      sc,
      () => {
        Class.forName("com.mysql.jdbc.Driver").newInstance()
        DriverManager.getConnection("jdbc:mysql://localhost:3306/db", "root", "123456")
      },
      "SELECT content FROM mysqltest WHERE ID >= ? AND ID <= ?",
      1, 100, 3,
      r => r.getString(1)).cache()

    print(rdd.filter(_.contains("success")).count())
    sc.stop()
  }
}
```

代码比较简短，主要是读mysqltest 表中的数据，并统计ID >=1 && ID < = 100 && content.contains("success")的记录条数。我们从代码中可以看出JdbcRDD的sql参数要带有两个？的占位符，而这两个占位符是给参数lowerBound和参数upperBound定义where语句的上下边界的。从JdbcRDD类的构造函数可以知道，参数lowerBound和参数upperBound都只能是Long类型的，并不支持其他类型的比较，这个使得JdbcRDD使用场景比较有限。而且在使用过程中sql参数必须有类似 ID >= ? AND ID <= ?这样的where语句，如果你写成下面的形式：

```
    val rdd = new JdbcRDD(
      sc,
      () => {
        Class.forName("com.mysql.jdbc.Driver").newInstance()
        DriverManager.getConnection("jdbc:mysql://localhost:3306/db", "root", "123456")
      },
      "SELECT content FROM mysqltest",
      1, 100, 3,
      r => r.getString(1)).cache()
```

那不好意思，运行的时候会出现以下的错误：

```
2014-09-10 15:47:45,621 (Executor task launch worker-0) [ERROR -
org.apache.spark.Logging$class.logError(Logging.scala:95)] Exception in task ID 1
java.sql.SQLException: Parameter index out of range (1 > number of parameters, which is 0).
  at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:1074)
  at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:988)
  at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:974)
  at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:919)
  at com.mysql.jdbc.PreparedStatement.checkBounds(PreparedStatement.java:3813)
  at com.mysql.jdbc.PreparedStatement.setInternal(PreparedStatement.java:3795)
  at com.mysql.jdbc.PreparedStatement.setInternal(PreparedStatement.java:3840)
  at com.mysql.jdbc.PreparedStatement.setLong(PreparedStatement.java:3857)
  at org.apache.spark.rdd.JdbcRDD$$anon$1.<init>(JdbcRDD.scala:84)
  at org.apache.spark.rdd.JdbcRDD.compute(JdbcRDD.scala:70)
  at org.apache.spark.rdd.JdbcRDD.compute(JdbcRDD.scala:50)
  at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:262)
  at org.apache.spark.CacheManager.getOrCompute(CacheManager.scala:77)
  at org.apache.spark.rdd.RDD.iterator(RDD.scala:227)
  at org.apache.spark.rdd.FilteredRDD.compute(FilteredRDD.scala:34)
  at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:262)
  at org.apache.spark.rdd.RDD.iterator(RDD.scala:229)
  at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:111)
  at org.apache.spark.scheduler.Task.run(Task.scala:51)
  at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:183)
  at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
  at java.lang.Thread.run(Thread.java:619)
```

看下JdbcRDD类的compute函数实现就知道了：

```
override def compute(thePart: Partition, context: TaskContext) = new NextIterator[T] {
    context.addOnCompleteCallback{ () => closeIfNeeded() }
    val part = thePart.asInstanceOf[JdbcPartition]
    val conn = getConnection()
    val stmt = conn.prepareStatement(sql, ResultSet.TYPE_FORWARD_ONLY,
                         ResultSet.CONCUR_READ_ONLY)

    if (conn.getMetaData.getURL.matches("jdbc:mysql:.*")) {
      stmt.setFetchSize(Integer.MIN_VALUE)
      logInfo("statement fetch size set to: " + stmt.getFetchSize +
                         " to force mySQL streaming ")
    }

    stmt.setLong(1, part.lower)
```

```
stmt.setLong(2, part.upper)

.....
```

不过值得高兴的是，我们可以自定义一个JdbcRDD，修改上面的计算思路，这样就可以得到符合我们自己要求的JdbcRDD。

PS: 在写本文的时候，本来我想提供一个JAVA例子，但是JdbcRDD类的最后一个参数很不好传，网上有个哥们是这么说的：

I don't think there is a completely Java-friendly version of this class. However you should be able to get away with passing something generic like "ClassTag.MODULE.<k>apply(Object.class)" There's probably something even simpler.

下面是我发邮件到Spark开发组询问如何在Java中使用JdbcRDD，开发人员给我的回复信息如下：

> The two parameters which might cause you some difficulty in Java are
>
> getConnection, which is a Function0[Connection], i.e. a 0 argument function that returns a jdbc connection
> and
> mapRow, which is a Function1[ResultSet, T], i.e. a 1 argument function that takes a result set and returns whatever type you want to convert it to.
>
> You could try and include the scala library as a compile-time dependency in your project, and make your own instances of classes that implement the Function0 and Function1 interfaces. I've never tried that, so your mileage may vary. The mapRow parameter might be skippable - it has as a default a function that just returns an array of object, which you could then cast.
>
> It would probably be easy to make the JdbcRDD interface more usable from Java8, but I don't know how much demand there is for that.

所以也只能放弃了。如果你知道怎么用Java实现，欢迎留言。