6.6 使用Spark SQL 读写数据库 슬라이드 1

好，现在我们进入第二个大部分——使用Spark SQL读写数据库。

在实际的大数据项目中，我们经常需要和各种数据库打交道。Spark SQL的一个强大之处就是它可以支持多种数据源（data sources）。

具体来说，Spark SQL可以支持：

- Parquet文件（Parquet files）
- JSON文件（JSON files）
- Hive数据仓库（Hive data warehouse）
- 通过JDBC连接的关系型数据库（relational databases via JDBC）

今天我们重点学习如何通过JDBC（Java Database Connectivity）连接MySQL数据库，实现数据的读取和写入。

我们的学习内容分三个部分：准备工作、读取MySQL数据、向MySQL写入数据。

6.6.1 슬라이드 2 准备工作

首先，我们需要做一些准备工作。

第一步是安装MySQL数据库。MySQL安装我们之前已经完成了。
因此这里我们只需启动MySQL即可。

首先启动MySQL服务：(우리는 이미 MySQL을 시작하였으므로 현재 상태 확인 필요, 我们已经启动了MySQL，因此需要检查当前状态)

service mysql start 혹은 상태확인 : systemctl status mysql

然后登录MySQL：

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

系统会提示你输入密码。

接下来，我们创建数据库和表：

```
create database spark;
```

```
use spark;
create table student (id int(4), name char(20), gender char(4), age int(4));
```

然后插入一些测试数据：

```
insert into student values(1,'Xueqian','F',23);
insert into student values(2,'Weiliang','M',24);
```

最后验证一下：

```
select * from student;
```

这样，我们就有了一个包含两条学生记录的测试表。这个表结构很简单：

id（学号）、name（姓名）、gender（性别）、age（年龄）。

6.6.1 슬라이드 3 准备工作

接下来，

我们就可以启动pyspark了：

```
cd /usr/local/spark
./bin/pyspark
```

这样，我们的开发环境就准备好了。

6.6.2 슬라이드 4 读取MySQL数据库中的数据 슬라이드

现在我们来学习如何从MySQL数据库读取数据。

大家看这段代码，我来逐行解释：

```
jdbcDF = spark.read ₩
.format("jdbc") ₩
.option("driver","com.mysql.jdbc.Driver") ₩
.option("url", "jdbc:mysql://localhost:3306/spark") ₩
.option("dbtable", "student") ₩
.option("user", "root") ₩
.option("password", "123456") ₩
.load()
```

这里使用了建造者模式（Builder pattern）：

- format("jdbc")：指定数据源格式为JDBC
- option("driver",...)：指定JDBC驱动类名
- option("url",...)：指定数据库连接URL
- option("dbtable",...)：指定要读取的表名
- option("user",...)和option("password",...)：数据库认证信息
- load()：执行加载操作

执行jdbcDF.show()后，我们可以看到数据已经成功读取到DataFrame中了。

这种方式的好处是，一旦数据加载到DataFrame，我们就可以使用所有的Spark SQL功能了。

6.6.2 슬라이드 5 读取MySQL数据库中的数据 슬라이드

표 설명이므로 슬라이드 보면서 ....오히려 pycharm 실습이 필요합니다.

除了在pyspark交互式环境中运行，我们也可以编写完整的应用程序。

这段代码展示了一个完整的独立应用程序：

```
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession ₩
        .builder ₩
        .master("local[*]") ₩
        .appName("SparkReadMySQL") ₩
        .getOrCreate()

    jdbcDF=spark.read.₩
        format("jdbc").₩
        option("driver","com.mysql.cj.jdbc.Driver").₩
        option("url","jdbc:mysql://localhost:3306/spark?useSSL=false").₩
        option("dbtable","students").₩
        option("user","root").₩
        option("password","123456").₩
        load()

    jdbcDF.show()
    spark.stop()
```

注意这几个要点：

- 我们创建了一个SparkSession对象
- appName给应用程序起了个名字
- master('local[*]')表示在本地模式运行，使用所有可用的CPU核心
- 最后记得调用spark.stop()释放资源

这种方式适合在IDE（集成开发环境）如PyCharm中开发和调试。在生产环境中，我们通常也是这样编写Spark应用程序的。

6.6.3 슬라이드 6 向MySQL数据库写入数据

现在我们来看写入数据的代码。

首先，我们需要定义数据的模式（schema）：

```
schema = StructType([StructField("id", IntegerType(), True), ₩
                     StructField("name", StringType(), True), ₩
                     StructField("gender", StringType(), True), ₩
                     StructField("age", IntegerType(), True)])
```

注意这里我们使用了不同的数据类型：

- IntegerType()：整数类型，用于id和age
- StringType()：字符串类型，用于name和gender

这样定义模式的好处是，可以确保数据类型的一致性，避免类型转换错误。

大家看，这种编程方式给了我们更精确的控制，特别是在处理复杂数据类型时非常有用。

6.6.3 슬라이드 7 向MySQL数据库写入数据

接下来，我们创建要插入的数据：

```
studentRDD = spark ₩
    .sparkContext ₩
    .parallelize(["3 Rongcheng M 26","4 Guanhua M 27"]) ₩
    .map(lambda x:x.split(" "))
```

这里我们使用parallelize方法创建RDD，然后用map操作分割字符串。

接着创建Row对象：

```
rowRDD = studentRDD.map(lambda p:Row(int(p[0].strip()), p[1].strip(), p[2].strip(), int(p[3].strip())))
```

注意类型转换：id和age需要转换为整数，name和gender保持字符串。

然后创建DataFrame：

```
studentDF = spark.createDataFrame(rowRDD, schema)
```

最后，我们设置数据库连接属性并写入：

```
prop = {}
prop['user'] = 'root'
prop['password'] = '123456'
prop['driver'] = "com.mysql.jdbc.Driver"
studentDF.write.jdbc("jdbc:mysql://localhost:3306/spark",'student','append', prop)
```

'append'模式表示追加数据，不会覆盖原有记录。

```
from pyspark.sql import Row
from pyspark.sql.types import *
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

if __name__ == '__main__':
    spark = SparkSession ₩
        .builder ₩
        .config(conf=SparkConf()) ₩
```

```
        .getOrCreate()

# 下面设置模式信息
schema=StructType([StructField("id",IntegerType(),True), ₩
                   StructField("name",StringType(),True), ₩
                   StructField("gender",StringType(),True), ₩
                   StructField("age",IntegerType(),True)])
# 下面设置两条数据，表示两个学生的信息
studentRDD=spark ₩
          .sparkContext ₩
          .parallelize(["20251222 RongCheng M 26","4 GuanHua M 27"]) ₩
          .map(lambda x:x.split(" "))

# 下面创建Row对象，每个Row对象都是rowRDD中的一行
# 对第1列去除空白后转整数,
# 对第2列去除空白,
# 对第3列去除空白,
# 对第4列去除空白后转整数
rowRDD=studentRDD.map(lambda p:Row(int(p[0].strip()),p[1].strip(),p[2].strip(),int(p[3].strip())))

# 建立起Row对象和模式之间的对应关系，也就是把数据和模式对应起来
studentDF=spark.createDataFrame(rowRDD,schema)

# 写入数据库
prop = {
    'user': 'root',
    'password': '123456',
    'driver': "com.mysql.cj.jdbc.Driver"
}

studentDF.write.jdbc("jdbc:mysql://localhost:3306/spark?useSSL=false",'students','append',prop)

spark.stop()
```

6.6.3 슬라이드 8 向MySQL数据库写入数据

这张图清楚地展示了数据转换的过程。

我们从字符串开始：

- "3 Rongcheng M 26"
- "4 Guanhua M 27"

经过split操作变成列表：

- ["3","Rongcheng","M","26"]
- ["4","Guanhua","M","27"]

最后创建Row对象：

- Row(3,"Rongcheng","M",26)
- Row(4,"Guanhua","M",27)

这个过程体现了数据处理的流水线思想（pipeline processing）。每一步都是数据的变换，最终得到我们需要的格式。

大家要注意数据类型的处理，特别是字符串和数字之间的转换。在实际项目中，数据清洗（data cleaning）往往占用很多时间，这些技巧非常有用。

```
from pyspark.sql import Row
from pyspark.sql.types import *
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

if __name__ == '__main__':
    spark = SparkSession ₩
        .builder ₩
        .config(conf=SparkConf()) ₩
        .getOrCreate()

    # 下面设置模式信息
    schema=StructType([StructField("id",IntegerType(),True), ₩
                       StructField("name",StringType(),True), ₩
```

```
                        StructField("gender",StringType(),True), ₩
                        StructField("age",IntegerType(),True)])
# 下面设置两条数据，表示两个学生的信息
studentRDD=spark ₩
                .sparkContext ₩
                .parallelize(["20251222 RongCheng M 26","4 GuanHua M 27"]) ₩
                .map(lambda x:x.split(" "))

# 下面创建Row对象，每个Row对象都是rowRDD中的一行
# 对第1列去除空白后转整数,
# 对第2列去除空白,
# 对第3列去除空白,
# 对第4列去除空白后转整数
rowRDD=studentRDD.map(lambda p:Row(int(p[0].strip()),p[1].strip(),p[2].strip(),int(p[3].strip())))

# 建立起Row对象和模式之间的对应关系，也就是把数据和模式对应起来
studentDF=spark.createDataFrame(rowRDD,schema)

# 写入数据库
prop = {
    'user': 'root',
    'password': '123456',
    'driver': "com.mysql.cj.jdbc.Driver"
}

studentDF.write.jdbc("jdbc:mysql://localhost:3306/spark?useSSL=false",'students','append',prop)

spark.stop()
```

슬라이드 19

最后，我们来验证写入操作的结果。

回到MySQL，执行查询：

mysql> select * from student;

太好了！我们可以看到表中现在有4条记录了：

1. Xueqian（F，23岁）
2. Weiliang（M，24岁）
3. Rongcheng（M，26岁）- 新插入的
4. Guanhua（M，27岁）- 新插入的

这说明我们的Spark程序成功地向MySQL数据库写入了数据。

总结

今天我们学习了两个重要主题：

第一部分：RDD到DataFrame的转换

- 反射机制：适合简单数据结构，代码简洁
- 编程方式：适合复杂数据结构，控制精确

第二部分：Spark SQL与数据库交互

- 读取MySQL数据：使用JDBC连接器
- 写入MySQL数据：支持多种写入模式

这些技能在实际的大数据项目中非常重要。Spark SQL的强大之处在于它能够统一处理各种数据源，让我们可以用一套API处理文件、数据库、数据仓库等不同的数据源。

大家课后要多练习，特别是要尝试连接不同类型的数据库，比如PostgreSQL、Oracle等。数据库连接是大数据工程师的基本技能，一定要熟练掌握。

有什么问题吗？