Spark 5.3 数据读写讲课脚本

슬라이드1: 5.3 数据读写（概述）

大家好！今天我们来学习Spark的5.3章节——数据读写。

我们来看看今天要学习的内容，总共分为三个主要部分。

首先是5.3.1本地文件系统的数据读写，也就是学习如何在本地文件系统中读写数据。

然后是5.3.2分布式文件系统HDFS的数据读写，我们会学习如何在分布式文件系统HDFS（Hadoop Distributed File System）中进行数据的读写操作。

最后是5.3.3读写MySQL数据库，我们将学习如何与MySQL数据库进行读写交互。

在实际的大数据处理中，数据如何输入和结果如何存储是非常关键的，所以请大家认真听讲。

슬라이드2: 5.3.1 本地文件系统的数据读写

好，现在我们开始学习第一个内容——本地文件系统的数据读写。

1.从文件中读取数据创建RDD

首先我们来看看如何从文件中读取数据来创建RDD（Resilient Distributed Dataset，弹性分布式数据集）。

大家看슬라이드上的代码：

```
>>> textFile = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> textFile.first()
```

'Hadoop is good'

我们使用sc.textFile()方法来读取文本文件。需要注意的是，文件路径前面要加上file://，这个是用来明确表示这是本地文件系统的。

textFile.first()方法是获取RDD的第一个元素。大家可以看到结果输出了'Hadoop is good'这个字符串。

这里我要给大家讲一个重要的概念。Spark采用了惰性机制（lazy evaluation）。

我们看下面这段代码：

```
>>> textFile = sc.textFile("file:///usr/local/spark/mycode/wordcount/word123.txt")
```

即使word123.txt这个文件不存在，spark-shell也不会立即报错。为什么呢？因为在执行转换操作（transformation operation）的时候，实际上并不进行真正的计算，要等到执行行动操作（action operation）的时候才会开始计算。

슬라이드3: 2.把RDD写入到文本文件中

现在我们来学习如何把RDD写入到文本文件中。

```
# -*- coding: utf-8 -*-
from pyspark import SparkConf, SparkContext
import shutil
import os

if __name__ == '__main__':
    try:
        # 1. Spark配置初始化
        conf = SparkConf().setMaster("local").setAppName("My App")  # 本地模式运行，应用名"My App"
        sc = SparkContext(conf=conf)  # 创建Spark上下文

        # 2. 输入文件路径配置
        input_path = 'file:///usr/local/spark/mycode/rdd/word.txt'  # 输入文件路径
        output_path = "file:///usr/local/spark/mycode/rdd/writeback"  # 输出目录路径

        # 3. 检查输出目录是否存在，存在则删除（Spark要求输出目录不能预先存在）
        if os.path.exists(output_path.replace("file://", "")):
            shutil.rmtree(output_path.replace("file://", ""))
            print(f"已删除已存在的输出目录: {output_path}")

        # 4. 读取文本文件
        textFile = sc.textFile(input_path)  # 创建RDD

        # 5. 输出第一行内容（调试用）
        print("文件第一行内容:", textFile.first())

        # 6. 保存处理结果
        textFile.saveAsTextFile(output_path)
```

```
        print(f"结果已成功保存到: {output_path}")

    except Exception as e:
        print("程序执行出错:", str(e))
    finally:
        # 7. 确保SparkContext被正确关闭
        if 'sc' in locals():
            sc.stop()
            print("SparkContext已关闭")
```

要把RDD中的数据保存到文本文件中，我们使用saveAsTextFile()方法。

这里有一个非常重要的注意事项！

saveAsTextFile()方法中提供的参数，不是文件名称，而是一个目录名称。

为什么呢？因为Spark通常是在分布式环境下执行的。RDD会存在多个分区（partition），由多个任务对这些分区进行并行计算，每个分区的计算结果都会保存到一个单独的文件中。

比如说，如果RDD有3个分区，saveAsTextFile()方法就会产生part-00000、part-00001和part-00002，以及一个_SUCCESS文件。

其中，part-00000、part-00001和part-00002包含了RDD中的数据，_SUCCESS文件只是用来表示写入操作已经成功执行，该文件里面是空的，可以忽略这个文件。

我们看实际的例子：

```
>>> textFile = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> textFile.saveAsTextFile("file:///usr/local/spark/mycode/rdd/writeback")
```

这样就把textFile变量中的内容重新写回到另外一个目录writeback中了。

슬라이드4: 执行结果查看

上面的语句执行后，请大家打开一个新的Linux终端，进入到"/usr/local/spark/mycode/rdd/"目录，可以看到这个目录下新增了一个名称为"writeback"的子目录。

进入writeback子目录以后，可以看到该目录中生成了3个文件——part-00000、part-00001和_SUCCESS（可以忽略）。part-00000和part-00001文件包含了刚才写入的数据。

那为什么writeback目录下包含两个part文件（即part-00000和part-00001），而不是一个part文件呢？

这是因为我们在启动进入pyspark环境时，使用了如下命令：

$ cd /usr/local/spark

$ ./bin/pyspark

/output-path/

```
├──── _SUCCESS              # 成功标志文件（空文件）
├──── part-00000            # 实际数据文件（文本/二进制格式）
├──── part-00000.crc        # part-00000的CRC校验文件
├──── part-00001            # 第二个数据分片
└──── part-00001.crc        # 对应分片的校验文件
```

1. **_SUCCESS 文件**
    - **性质**：0 字节的空文件
    - **作用**：标记作业成功完成
    - **特点**：
        - 仅当所有任务成功执行时才会生成
        - 分布式系统（如 HDFS）通过检查此文件确认作业完整性
2. **part-xxxxx 文件**

- **存储内容**：实际输出数据（每个分片对应一个文件）
- **命名规则**：
  - part-00000：第一个分片
  - part-00001：第二个分片
  - ...（依此类推）
- **文件格式**：取决于输出方式（如 saveAsTextFile()生成文本格式）

3. **.crc 校验文件**
   - **全称**：Cyclic Redundancy Check（循环冗余校验）
   - **功能**：
     - 用于验证对应数据文件的完整性
     - 防止数据在存储/传输过程中损坏
   - **特点**：
     - 每个 part 文件都有对应的.crc 文件
     - 文件体积很小（通常 12 字节）
     - Spark/Hadoop 在读取数据时会自动校验

슬라이드5: 分区数与文件数的关系

在上述启动命令中，pyspark命令后面没有带上任何参数，所以系统默认采用local[*]模式启动pyspark。

local[*]中的*表示使用所有可用的CPU核心。当本地计算机的CPU有两个核时，系统就会使用两个Worker线程本地化运行Spark。

而且，在读取文件时，sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")语句的圆括号中的参数只有文件地址，并没有包含分区数量，这时，系统会自动为textFile这个RDD生成两个分区，因此saveAsTextFile()最终生成的文件就有两个，即part-00000和part-00001。

作为对比，下面我们在启动pyspark时设置masterURL为"local[1]"，具体如下：

$ cd /usr/local/spark

$ ./bin/pyspark --master local[1]

首先删除之前生成的writeback目录，然后再次执行如下语句：

>>> textFile = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")

>>> textFile.saveAsTextFile("file:///usr/local/spark/mycode/rdd/writeback")

上述语句执行后，大家可以发现，在writeback子目录下只有1个part文件，即part-00000。

这样就证明了分区数决定输出文件的个数。

슬라이드6: 5.3.2 分布式文件系统HDFS的数据读写

现在我们来学习第二个主题——分布式文件系统HDFS的数据读写。

从分布式文件系统HDFS中读取数据，也是采用textFile()方法，可以为textFile()方法提供一个HDFS文件或目录地址。

如果是一个文件地址，它会加载该文件；如果是一个目录地址，它会加载该目录下的所有文件的数据。

我们看例子：

```
# -*- coding: utf-8 -*-
"""
简单的PySpark HDFS数据处理脚本
适合学生学习使用
"""

from pyspark import SparkConf, SparkContext
import subprocess
import sys


def hdfs_path_exists(hdfs_path):
    """检查HDFS路径是否存在"""
    try:
        cmd = ["hdfs", "dfs", "-test", "-e", hdfs_path]
        result = subprocess.run(cmd, capture_output=True, timeout=30)
        return result.returncode == 0
    except Exception as e:
        print(f"检查路径失败: {str(e)}")
        return False


def delete_hdfs_path(hdfs_path):
    """删除HDFS路径"""
    try:
        if hdfs_path_exists(hdfs_path):
            cmd = ["hdfs", "dfs", "-rm", "-r", "-f", hdfs_path]
            subprocess.run(cmd, check=True, timeout=60)
```

```
                print(f"已删除HDFS目录: {hdfs_path}")
                return True
            else:
                print(f"路径不存在: {hdfs_path}")
                return True
        except Exception as e:
            print(f"删除路径失败: {str(e)}")
            return False


def main():
    """主函数"""
    # 配置参数
    namenode = "hadoop01:9000"
    input_path = f"hdfs://{namenode}/user/hadoop/word.txt"
    output_path = f"hdfs://{namenode}/user/hadoop/output"

    print("开始HDFS数据处理...")

    # 检查输入文件是否存在
    if not hdfs_path_exists(input_path):
        print(f"错误: 输入文件不存在 {input_path}")
        return 1

    # 删除输出目录（如果存在）
    if not delete_hdfs_path(output_path):
        print("错误: 无法清理输出目录")
        return 1

    # 初始化Spark
    try:
        conf = SparkConf() ₩
            .setMaster("local[2]") ₩
            .setAppName("HDFS文本处理") ₩
            .set("spark.hadoop.validateOutputSpecs", "false")

        sc = SparkContext(conf=conf)
        sc.setLogLevel("WARN")  # 减少日志输出
```

```
        print("Spark初始化成功")

except Exception as e:
        print(f"Spark初始化失败: {str(e)}")
        return 1

try:
        # 读取HDFS文件
        print(f"正在读取文件: {input_path}")
        text_rdd = sc.textFile(input_path)

        # 显示文件行数
        line_count = text_rdd.count()
        print(f"文件总行数: {line_count}")

        # 显示第一行内容
        if line_count > 0:
                first_line = text_rdd.first()
                print(f"第一行内容: {first_line}")

        # 数据处理：转换为大写并过滤空行
        processed_rdd = text_rdd ₩
                .filter(lambda line: line.strip()) ₩
                .map(lambda line: line.upper())

        # 显示处理后行数
        processed_count = processed_rdd.count()
        print(f"处理后行数: {processed_count}")

        # 保存到HDFS
        print(f"正在保存到: {output_path}")
        processed_rdd.saveAsTextFile(output_path)

        # 检查是否成功
        if hdfs_path_exists(f"{output_path}/_SUCCESS"):
                print("✓ 处理完成！")
                return 0
```

```
            else:
                print("X 处理失败，未找到成功标记")
                return 1

        except Exception as e:
            print(f"数据处理错误: {str(e)}")
            return 1

        finally:
            # 关闭Spark
            try:
                sc.stop()
                print("Spark已关闭")
            except:
                pass


if __name__ == '__main__':
    exit_code = main()
    sys.exit(exit_code)
```

슬라이드7: 5.3.3 读写MySQL数据库 - 准备工作

第三个主题是MySQL数据库的读写操作。

1.准备工作

首先我们需要启动MySQL数据库，在Linux中执行如下命令：

```
$ service mysql start
$ mysql -u root -p #屏幕会提示输入密码
```

在MySQL Shell环境中，输入下面SQL语句完成数据库和表的创建：

```
mysql> create database spark;
mysql> use spark;
mysql> create table student (id int(4), name char(20), gender char(4), age int(4));
mysql> insert into student values(1,'Xueqian','F',23);
mysql> insert into student values(2,'Weiliang','M',24);
mysql> select * from student;
```

这样我们就在spark数据库中创建了student表，并插入了两条样本数据。

슬라이드8: MySQL驱动程序设置和PyMySQL安装

要想顺利连接MySQL数据库，还需要使用MySQL数据库驱动程序。

请到MySQL官网下载MySQL的JDBC驱动程序，或者直接到本教材官网的"下载专区"的"软件"目录中下载驱动程序文件mysql-connector-java-5.1.40.tar.gz。

对该文件进行解压，找到文件mysql-connector-java-5.1.40-bin.jar，然后，把该驱动程序JAR包复制到Spark的安装目录的jars子目录中（即"/usr/local/spark/jars"）。

利用pyspark类库编写程序访问MySQL数据库时，还需要在Python环境中安装pymysql模块，需要在hadoop01节点的Linux终端中执行如下命令：

$ conda activate pyspark

$ pip install pymysql

这样我们就完成了与MySQL连接的准备工作。

슬라이드9: 2.读取MySQL数据库中的数据

现在我们来实际编写从MySQL数据库中读取数据的代码。

新建一个代码文件SparkReadMySQL.py，内容如下：

```
# -*- coding: utf-8 -*-
# Page 119
# MySQL与PySpark集成示例：从MySQL数据库读取数据并转换为RDD

import pymysql  # MySQL数据库连接库
from pyspark import SparkConf, SparkContext  # Spark核心组件

if __name__ == '__main__':
    # 1. 配置并初始化Spark
    conf = SparkConf().setAppName("MySQL RDD Example")  # 设置应用程序名称
    sc = SparkContext(conf=conf)  # 创建SparkContext实例

    # 2. MySQL数据库连接参数配置
    host = "localhost"  # MySQL服务器地址
    port = 3306  # MySQL端口号（默认3306）
    user = "root"  # 数据库用户名
    password = "123456"  # 数据库密码
    database = "spark"  # 目标数据库名
    table = "students"  # 目标数据表名

    # 3. 创建MySQL数据库连接
    conn = pymysql.Connect(
        host=host,
        port=port,
        user=user,
        password=password,
        database=database
    )
    cursor = conn.cursor()  # 创建游标对象，用于执行SQL语句

    # 4. 执行SQL查询语句
    sql = "select * from {}".format(table)  # 构建查询所有记录的SQL语句
    cursor.execute(sql)  # 执行SQL查询
```

```
# 5. 将查询结果转换为Spark RDD
rdd = sc.parallelize(cursor.fetchall())  # 获取所有查询结果并转换为RDD

# 6. 输出RDD中的所有数据
rdd.foreach(print)  # 遍历RDD并打印每一行数据

# 7. 关闭资源
cursor.close()  # 关闭游标
conn.close()  # 关闭数据库连接
 sc.stop()  # 停止SparkContext
```

我来给大家解释一下这段代码的核心部分：

1. 首先导入SparkConf和SparkContext，还有pymysql模块
2. 设置MySQL连接信息
3. 使用pymysql.connect()连接到MySQL
4. 执行SQL查询获取数据
5. 使用sc.parallelize()将获取到的数据转换为RDD
6. 用rdd.foreach(print)输出RDD的所有元素

슬라이드10: MySQL读取程序执行结果

程序的执行结果如下：

(1, 'Xueqian', 'F', 23)

(2, 'Weiliang', 'M', 24)

大家可以看到，每一行都以元组（tuple）的形式输出。这说明MySQL表的每一行都成功转换为了RDD的元素。

这样我们就成功地将MySQL数据库中的数据读取到Spark RDD中了。

슬라이드11: 3.向MySQL数据库中写入数据

最后我们来学习如何向MySQL数据库中写入数据。

我们编写如下代码：

```
#Page 119
import pymysql
from pyspark import SparkConf, SparkContext
if __name__=='__main__':
    conf=SparkConf().setAppName("MySQL RDD Example")
    sc=SparkContext(conf=conf)
    myList=[(13,"ywlee","M",66),(14,"YoonSangOk","M",56)]
    rddData=sc.parallelize(myList)
    def insertIntoMySQL(tuple):
        host="localhost"
        port=3306
        user="root"
        password="123456"
        database="spark"
        table="students"
        #Create MySQL Interface
        conn=pymysql.Connect(host=host, port=port, user=user, password=password, database=database)
        cursor=conn.cursor()
        # execute SQL Query
        sql="INSERT INTO {} (id, name, gender, age) VALUES (%d, '%s','%s',%d)".format(table)
        value=(tuple[0], tuple[1],tuple[2],tuple[3])
        cursor.execute(sql % value)
        conn.commit()
        cursor.close()
        conn.close()
        # transformation from result to RDD
    rddData.foreach(insertIntoMySQL)
     sc.stop()
```

我来解释一下这段代码的关键点：

1. 首先创建一个包含两个新学生数据的RDD
2. 定义一个insertIntoMySQL函数，这个函数接受一个元组作为参数，然后将其插入到M

ySQL中

3. 在函数内部创建MySQL连接，执行INSERT SQL语句
4. 用conn.commit()提交事务，然后关闭连接
5. 使用rddData.foreach(insertIntoMySQL)对RDD的每个元素执行这个函数

执行该程序以后，到MySQL中就可以看到新插入了两条数据。

课程总结

今天我们学习了Spark中数据读写的三种主要方法：

1. **本地文件系统**：使用sc.textFile()和saveAsTextFile()
2. **HDFS分布式文件系统**：使用相同的方法，但指定HDFS路径
3. **MySQL数据库**：利用pymysql进行连接和通过sc.parallelize()进行RDD转换

每种方法在实际的大数据处理中都起着非常重要的作用。特别是RDD的分区概念和惰性求值概念，希望大家能够很好地理解。

下次课我们将学习更高级的数据处理技术。大家辛苦了！