# SQL查询分析器设计说明书

班级：18级软件工程1班 姓名：许航

**一、项目介绍**

基于通过编程的方式实现软件调用大数据集群进行计算，将大数据能力附加到软件中的实操课程，结合Spark环境下个人设计开发简单版Spark SQL查询分析器。分析器的主要要求如下：

1、可以配置Spark SQL连接信息；

2、列出当前连接下的库、表、字段，以便编写SQL脚本；

3、提供SQL编辑框，方便编写SQL和提交查询任务；

4、以列表的形式，展示查询结果；

**二、总体设计**

1、项目总体设计

采用maven项目下编写java文件与scala文件开发，通过scala调用JDBC接口连接Sparksql,通过JDBC的标准接口方式,通过JDBC提交计算任务让应用直接对接到大数据集群。

出于项目为简易版应用，采用java Swing的GUI界面开发，简单直接的设计界面，实现功能。

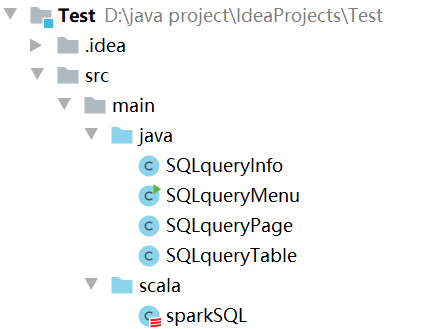
2、系统环境配置

软件：JDK 1.8、IntelliJ IDEA 2020.1.1(Commuity Editon)、spark jdbc dependencies.zip

操作系统：windows 10

**三、详细设计**

本项目主要由java文件夹下的SQLqueryInfo.java、SQLqueryMenu.java、SQLqueryPage.java、SQLqueryTable.java和scala文件夹下的sparkSQL.scala构成。



java文件集成了sparkSQL的函数调用和ui的界面编写。

1、界面设计

界面设计包括主要界面：配置Spark SQL连接信息页面和当前连接下的查询界面。

a. 配置Spark SQL连接信息页面



主要包括在类SQLqueryMenu中，采用javaSwing的Jframe创建配置连接信息窗口，通过javaSwing的Jlabel、JtextField、Jbotton等元素编写界面，利用Jbotton的addActionListener()重写actionPerformed(ActionEvent e)监听点击按钮的动作，判断SparkSQL连接成功后调用下一界面的类创建新的窗口。

1. **public** **class** SQLqueryMenu **extends** JFrame
2. {
3. **private** JTextField hostField = **new** JTextField();
4. **private** JTextField postField = **new** JTextField();
5. **private** JTextField dbField = **new** JTextField();
6. **private** JTextField userField = **new** JTextField();
7. **private** JPasswordField pwdField = **new** JPasswordField();
8. **private** JButton connectButton = **new** JButton("连接");
9. **public** SQLqueryMenu(){
10. **this**.setBounds(200, 100, 550, 400);
11. **this**.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
12. **this**.setTitle("新建连接");
13. **this**.setLayout(**null**);
14. **this**.setResizable(**false**);
15. **final** **int** xTitle = 120;
16. **final** **int** xBox = 220;
17. **final** **int** boxWidth = 150;
18. JLabel title = **new** JLabel("配置Spark SQL连接信息");
19. title.setBounds(180, 30, 200, 20);
20. JLabel host = **new** JLabel("主机：");
21. host.setBounds(xTitle, 90, 100, 20);
22. JLabel post = **new** JLabel("端口：");
23. post.setBounds(xTitle, 120, 100, 20);
24. JLabel database = **new** JLabel("初始数据库：");
25. database.setBounds(xTitle, 150, 100, 20);
26. JLabel userName = **new** JLabel("用户名：");
27. userName.setBounds(xTitle, 180, 100, 20);
28. JLabel pwdTest = **new** JLabel("密码：");
29. pwdTest.setBounds(xTitle, 210, 100, 20);

32. hostField.setBounds(xBox, 90, 180, 20);
33. postField.setBounds(xBox, 120, 50, 20);
34. dbField.setBounds(xBox, 150, boxWidth, 20);
35. userField.setBounds(xBox, 180, boxWidth, 20);
36. pwdField.setBounds(xBox, 210, boxWidth, 20);
38. connectButton.setBounds(xBox, 250, 100, 30);
39. connectButton.addActionListener(**new** ActionListener() {
40. **public** **void** actionPerformed(ActionEvent e) {
41. **if** (e.getSource() == connectButton) {// 判断触发源是否为按钮
42. String url = hostField.getText() + ":" + postField.getText() + "/" + dbField.getText();
43. String userName = userField.getText();
44. String userPassword = pwdField.getPassword().toString();
45. sparkSQL s = **new** sparkSQL(url, userName, userPassword);
47. JOptionPane.showMessageDialog(**null**, "连接成功！", "提示", JOptionPane.PLAIN\_MESSAGE);
48. dispose();
49. **final** SQLqueryPage ui2 = **new** SQLqueryPage();
50. //                    ui2.SQLqueryPage("jdbc:hive2://bigdata115.depts.bingosoft.net:22115/user18\_db","user18","pass@bingo18");
51. ui2.SQLqueryPage(url, userName, userPassword);
52. }
53. }
54. });// 加入动作监听
55. **this**.add(title);
56. **this**.add(host);
57. **this**.add(post);
58. **this**.add(database);
59. **this**.add(userName);
60. **this**.add(pwdTest);
61. **this**.add(hostField);
62. **this**.add(postField);
63. **this**.add(dbField);
64. **this**.add(userField);
65. **this**.add(pwdField);
66. **this**.add(connectButton);
67. }
68. **public** **static** **void** main(String[] agrs)
69. {
70. SQLqueryMenu ui=**new** SQLqueryMenu();
71. ui.setVisible(**true**);
72. //            Demo.demo();
73. //      val url = "jdbc:hive2://bigdata115.depts.bingosoft.net:22115/user13\_db"
75. }
76. }

b.当前连接下的查询界面



主要包括在类SQLqueryPage中，该界面主要包括三个部分：一是元素展示区，展示当前连接的数据库；二是查询语句输入区；三是查询按钮事件触发后所展示的表格新窗口。

1. **public** **class** SQLqueryPage {
2. **private** JFrame frame = **new** JFrame();
3. **private** JButton queryBtn = **new** JButton("查询");
4. **private** JTextArea queryText = **new** JTextArea();
5. **private** JScrollPane jsp = **new** JScrollPane();
6. **public** **void** SQLqueryPage(String url, String userName, String userPassword){
7. **final** sparkSQL s = **new** sparkSQL(url,userName,userPassword);
8. frame.setBounds(200, 100, 850, 500);
9. frame.setTitle("新建连接");
10. frame.setLayout(**null**);
11. JLabel info = **new** JLabel("当前连接下的数据库信息：");
12. info.setBounds(20,20,200,20);
13. **int** index = url.lastIndexOf('/');
14. String db\_name = url.substring(index+1,url.length());
15. JLabel info1 = **new** JLabel("数据库："+db\_name);
16. info1.setBounds(20,40,200,20);
17. JLabel info2 = **new** JLabel("请输入查询语句：");
18. info2.setBounds(200,20,200,20);
19. queryText.setBounds(200,50,600,300);
20. queryBtn.setBounds(700,400,100,20);
22. queryBtn.addActionListener(**new** ActionListener() {
23. **public** **void** actionPerformed(ActionEvent e) {
24. String query = queryText.getText();
25. //              String[][] dataTable =new String[100][100];
26. String[][] dataTable =s.queryRes(query);
27. //                for(int i=0;i<100;i++){
28. //                    if(dataTable[i][0]==null){
29. //                        break;
30. //                    }
31. //                    for(int j=0;j<100;j++){
32. //                        if(dataTable[i][j]==null){
33. //                            System.out.println("|");
34. //                            break;
35. //                        }
36. //                        System.out.printf(dataTable[i][j]);
37. //                    }
38. //                }
39. SQLqueryTable table = **new** SQLqueryTable();
40. table.SQLqueryTable(dataTable);
41. }
42. });
43. SQLqueryInfo infos = **new** SQLqueryInfo();
44. String[] items=s.showTable();
45. ArrayList<String> tableNames=**new** ArrayList<String>();
46. **for**(**int** i=0;i<100;i++){
47. **if**(items[i]==**null**){
48. **break**;
49. }
50. tableNames.add(items[i]);
51. }
52. String[] tableName=tableNames.toArray(**new** String[tableNames.size()]);
53. JPanel infop = infos.SQLqueryInfo(url,userName,userPassword,tableName);
54. jsp.setViewportView(infop);
55. jsp.setBounds(20,70,165,280);
57. frame.add(info);
58. frame.add(info1);
59. frame.add(jsp);
60. frame.add(info2);
61. frame.add(queryBtn);
62. frame.add(queryText);
63. frame.setVisible(**true**);
64. }
66. }

元素展示区主要包括在类SQLqueryInfo中,采用Java Swing的JscrollPane 解决可能需要展示的表或者字段过多的问题。

2. **public** **class** SQLqueryInfo {
3. **public** JPanel SQLqueryInfo(String url, String userName, String userPassword,String[] tableNames){
4. JPanel table= **new** JPanel();
5. Box vBox = Box.createVerticalBox();
6. **final** sparkSQL s = **new** sparkSQL(url,userName,userPassword);
7. **for**(**int** i = 0; i < tableNames.length; i++) {
8. JLabel tableTitle = **new** JLabel(tableNames[i]);
9. String[] TableFieldNames = s.showTableField(tableNames[i]);
10. JList list=**new** JList(TableFieldNames);
11. vBox.add(tableTitle);
12. list.setVisibleRowCount(4);
13. JScrollPane jsp = **new** JScrollPane(list);
14. vBox.add(jsp);
15. }
16. table.add(vBox);
18. **return** table;
19. }
20. }

查询语句输入区包括在类SQLqueryPage中，采用Java Swing的JtextArea，能够读入用户的多行输入。

查询按钮事件触发后所展示的表格新窗口包括在类SQLqueryTable中，采用Java Swing的Jtable重新创建一个窗口用于展示查询结果。

1. **public** **class** SQLqueryTable {
2. **private** JFrame frame =**new** JFrame();
3. **public** **void** SQLqueryTable(String[][] dataTable){
4. frame.setSize(600,400);
5. frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);
6. Container contentPane=frame.getContentPane();
7. ArrayList<String> name = **new** ArrayList<String>();
8. **for**(**int** i = 0;i<100;i++){
9. **if**(dataTable[0][i] == **null**){
10. **break**;
11. }
12. name.add(dataTable[0][i]);
13. }
14. **int** col=name.size();
15. **int** row=0;
16. **for**(**int** j=0;j<100;j++){
17. **if**(dataTable[j][0]==**null**){
18. **break**;
19. }
20. row++;
21. }
23. Object[][] tableContent = **new** Object[row-1][col];
24. **for**(**int** i=1;i<row;i++){
25. **for**(**int** j=0;j<col;j++){
26. tableContent[i-1][j]=dataTable[i][j];
27. }
29. }
30. String[] arr = name.toArray(**new** String[name.size()]);
32. JTable table=**new** JTable(tableContent,arr);
33. contentPane.add(**new** JScrollPane(table));
34. frame.setVisible(**true**);
35. }
36. }

2、获取数据库数据函数介绍

该内容主要包括在类sparkSQL中，通过scala编写。主要包括配置sparkSQL的连接信息、获取当前数据库连接下的所有表名showTable()、获取当前数据库连接下的表的字段showTableField()、获取当前数据库连接下的查询语句所得的结果queryRes(query:String)，并返回一个二维数组。

通过java.sql.connection类获取数据库执行sql查询语句所返回的resultSet，按照所需数据库内容编写函数。

1. **class** sparkSQL(url:String,userName:String,userPassword:String) {
2. **import** java.sql.DriverManager
3. val properties = **new** Properties()
4. properties.setProperty("driverClassName", "org.apache.hive.jdbc.HiveDriver")
5. properties.setProperty("user", userName)
6. properties.setProperty("password", userPassword)
7. val connection = DriverManager.getConnection(url, properties)
8. val statement = connection.createStatement
9. def showTable(): Array[String] = {
10. var resultSet = statement.executeQuery("show tables")
11. var tableNames = **new** Array[String](100)
12. **try** {
13. var a=0;
14. **while** (resultSet.next) {
15. val tableName = resultSet.getString(1)
16. tableNames(a) = tableName
17. a+=1
18. }
19. resultSet.close()
20. }
21. **return** tableNames
22. }
23. def showTableField(tableName:String): Array[String] = {
24. var resultSet = statement.executeQuery("select\*from "+tableName+" limit 10")
25. var data = resultSet.getMetaData()
26. //    if(data.getColumnCount > 20){
27. //      var tableFieldNames = new Array[String](21)
28. //      try {
29. //        var a=0;
30. //        for(a <- 1 to 20) {
31. //          tableFieldNames(a-1) = data.getColumnName(a)
32. //        }
33. //        tableFieldNames(20)="..."
34. //        resultSet.close()
35. //      }
36. //      return tableFieldNames
37. //    }
38. //    else{
39. var tableFieldNames = **new** Array[String](data.getColumnCount)
40. **try** {
41. var a=0;
42. **for**(a <- 1 to data.getColumnCount) {
43. tableFieldNames(a-1) = data.getColumnName(a)
44. }
45. resultSet.close()
46. }
47. **return** tableFieldNames
48. //      }
49. }
50. def queryRes(query:String): Array[Array[String]] = {
51. //Array[Array[String]]
52. var resultSet = statement.executeQuery(query)
53. var dataTable = Array.ofDim[String](100, 100)
54. **try** {
55. var data=resultSet.getMetaData()
56. var a =1
57. **for** (a <- 1 to data.getColumnCount){
58. **if**(a==1){
59. dataTable(0)(a - 1)="序列\\列名"
60. }
61. dataTable(0)(a)=data.getColumnName(a)
62. }
63. var b=1
64. **while**(resultSet.next){
65. a=1
66. **for** (a <- 1 to data.getColumnCount){
67. **if**(a==1) {
68. dataTable(b)(a - 1) = b.toString
69. }
70. dataTable(b)(a)=resultSet.getString(a)
71. }
72. b+=1
73. }
74. resultSet.close()
75. **return** dataTable
76. }
77. }
79. }

**四、总结**

本次项目开发主要为了体会并理解大数据离线计算的方式，不同的框架下离线计算的速度和对不同操作的运行效果，在做出项目界面后进行尝试insert语句发现sparksql数据库并不能很好的支持这一语句，但通过greenplum框架却能比较轻松的执行insert语句，令我一度困惑是否scala代码使用错误，在尝试了网上搜索出的用于insert语句的函数statement.execute ()、statement.updateQuery()等，依然不能执行insert语句，实训的老师给出了专业的解释：虽然spark 、hive 都提供了JDBC的方式操作数据，但他们的场景是面向OLAP的数据分析的，对数据insert支持不好，更多使用方式是把数据写入到分布式式文件系统 HDFS、S3等，通过外部表的方式，把表和数据映射上，做数据的统计分析。另外一个MPP数据库 greenplum，就对insert操作支持的很好。

通过这次项目实践，对使用JDBC接口连接数据库获取数据库信息有了比较深刻的理解，也对不同的大数据框架有了一定的认识。