Hive作为Mondrian的数据源

需求

项目中的报表系统使用开源的mondrian和saiku作为工具实现的，现在自己又不得不去熟悉OLAP这一块的东西，首先要面对的就是mondrian这一座大山，听他们之前的开发人员介绍说mondrian里面会有很多坑，尤其是性能问题，在之前自己的测试过程中自己也遇到了一些问题，但是当时没怎么记录过了一两个月就差不多忘记怎么解决的了。

不过当时对于mondrian的慢还是深有体会的，至于他是怎么实现的我也没有看过源代码，只是通过查看执行结果发现的，我使用的数据源是官方提供的foodmart数据库，在之前的文章中介绍了如何生成这些数据和如何将这些数据导入到hive中，然后我在自己的测试集群上搭了伪分布式的集群，使用mondrian分别将mysql和hive作为数据源进行测试，一个MDX查询mysql可以几秒钟查到的结果，在hive中整整跑了半个小时，当时观察着hive的history记录发现每一个sql都会跑一个甚至多个mapreduce任务，至于原因嘛，我觉得首先mondrian生成的sql就是比较多的，我记录mysql的查询记录发现一个MDX查询会生成大概18条SQL查询，其中不乏group by和join之类的查询，因为mondrian没有对底层数据源进行优化（我认为），所以对于mysql和hive生成的SQL理论上是一样的（没有进行比较），所以这些冗长的SQL拖慢了整个查询的速度；其次hive不是mysql那样的面向TPS的数据库，它是面向吞吐量的，所以每一个SQL查询的响应时间是很久的，通过观察发现生成的这些SQL是顺序执行的，为什么不慢呢！

虽然慢，但是先实功能再说其他的吧，之前测试hive的过程中记得自己对hive的jdbc源码进行了修改，主要是修改了一些hive在实现jdbc中没有实现但是抛出异常的接口，而mondrian会调用这些接口导致下面的流程走不下去了，整体的修改应该说还是比较简单的。另外一个问题是当时的hive是没有使用任何认证机制的，包括hadoop也是没有认证机制的，现在在公司的hadoop集群上跑需要使用kerberos认证，这一块自己还不熟悉，还只是知道怎么用，所以还需要恶补了一下关于kerberos认证的知识。

如何使用Hive

之前的准备工作已经做好了，首先我已经测试了使用mysq作为数据源使用mondrian作为MDX查询引擎的流程、然后记录了如果生成mondrian的foodmart数据以及如何将数据到导入到hive中，另外，还有如何部署hive的hiveserver2，然后使用jdbc连接这个server。万事俱备，只欠东风了。

首先既然hiveserver是用了kerberos认证的，其实它是一个代理服务器，使用代理用户hive代理其他用户提交mapreduce任务和执行HDFS操作，所以在代码中首先需要使用kerberos认证，认证的流程如下：

```
        Configuration conf = new Configuration();
        conf.setBoolean( "hadoop.security.authorization" , true);
        conf.set( "hadoop.security.authentication" , "kerberos" );
        UserGroupInformation. setConfiguration(conf);
         try {
               UserGroupInformation. loginUserFromKeytab("intern/bigdata", "C:\\Users\\Administrator\\Desktop\\intern.keytab" );
        } catch (IOException e) {
                // TODO Auto-generated catch block
               e.printStackTrace();
        }
```

初始化了kerberos之后，接下来需要指定hive的jdbc驱动器，在使用mysql的时候貌似没有指定，但是mondrian并没有官方支持hive的版本（其实mondrian只是支持提供了jdbc接口的数据源），所以需要在代码的开始指定hive数据源

```
    try {
       Class. forName("org.apache.hive.jdbc.HiveDriver");
     } catch (ClassNotFoundException e) {
       e.printStackTrace();
     }
```

然后再将连接mysql时指定的olap4j的url，其中只需要替换了Jdbc部分换成hive的的jdbc的url，替换之后的url如下：

```
  Connection connection = DriverManager. getConnection("Provider=mondrian;" +
          "Jdbc=jdbc:hive2://bitest0.server.163.org:10000/foodmart;principal=hive/app-20.photo.163.org@HADOOP.HZ.NETEASE.COM;" +
          "Catalog=http://db-53.photo.163.org:16161/cube/queryxml/60548;"
          + "DynamicSchemaProcessor=mondrian.i18n.NrptLocalizingDynamicSchemaProcessor;"
          + "Locale=zh_CN",
         null);
```

连接URL

建立好了连接，接下来就可以执行一条mdx查询，需要注意的是这里的Catalog是一个url而不是文件了，因为这个url是我们项目中提供下载cube定义的xml文件，因此也可以看到在Catalog参数中只需要指定能够获取xml文件的地址就可以了，无论是一个url还是本地文件路径。接着执行如下的MDX查询:

```
  Query query = connection.parseQuery( "SELECT NON EMPTY {Hierarchize({[时间].[年份].Members})} ON COLUMNS, "
                + " NON EMPTY {[Measures].[销售总额]} ON ROWS FROM [TestFoodMart]");
   Result result = connection. execute(query);
   PrintWriter pw = new PrintWriter(System.out);
   result.print(pw);
   pw.flush();
```

这样应该就可以了，但是还是出现了这样的错误：

```
Caused by: org.apache.thrift.transport.TTransportException: Peer indicated failure: Unsupported mechanism type PLAIN
   at org.apache.thrift.transport.TSaslTransport.receiveSaslMessage(TSaslTransport.java:190)
   at org.apache.thrift.transport.TSaslTransport.open( TSaslTransport.java:288)
   at org.apache.thrift.transport.TSaslClientTransport.open(TSaslClientTransport.java:37)
   at org.apache.hive.jdbc.HiveConnection.openTransport(HiveConnection.java:203)
   ... 20 more
```

这是提示我使用的验证方式不对，因为服务端使用的是kerberos认证而客户端没有，但是我分明已经在url里面里面加上了principal这个参数了啊，尝试了一下直接连接hive但是jdbc的url不加principal这个参数，果然出现了相同的错误，这也就是说这个principal没有生效，为什么呢？

通过观察olap4j的url发现它正好是使用";"分割的，而principal和前面的hiveserver的服务器端口号之间也使用的是";"分割，查看了hiveserver2的文档发现原来在使用kerberos认证的时候必须使用这种方式的url（host:port;principal=xxx），这就有点纠结了，mondrian里面把";"当成了分隔符，就会只把前面的host:ip作为jdbc的url了，当然连接出现错误了啊。

也尝试过使用properties参数创建连接，但是尝试都失败了，因为hive的url必须将principal和host：port用";"分割，之后只有查看一下源代码如何对mondrian的url进行分割的，创建连接是在mondrian.olap.DriverManager类的静态函数getConnection完成的，该函数如下：

```
45     public static Connection More ...getConnection(
46         String connectString,
47         CatalogLocator locator)
48     {
49         Util.PropertyList properties = Util.parseConnectString(connectString);
50         return getConnection(properties, locator);
51     }

2729    public static PropertyList More ...parseConnectString(String s) {
2730        return new ConnectStringParser(s).parse();
2731    }
```

在Util.parseConnectString函数中对url进行划分，划分之后的结果使用key:value的形式保存在一个List中，pair的first为key，second为value，可想而知在解析的时候是根据"="作为key和value的分割符，使用";"作为一个key:value对直接的分隔符。

在ConnectStringParser类中有如下几个函数：parsePair（解析一个key:value对，分为解析key和解析value两部分）、parseName（解析一个pair的key）、parseValue（解析一个pair的value）和parseQuoted（解析特殊符号），在因为这个url是作为jdbc这个key的value，查看parseValue这个函数发现了如下的代码：

```
2830            if (c == '"' || c == '\'') {
2831                String value = parseQuoted(c);
2832                // skip over trailing white space
2833                while (i < n && (c = s.charAt(i)) == ' ') {
2834                    i++;
2835                }
2836                if (i >= n) {
2837                    return value;
2838                } else if (s.charAt(i) == ';') {
2839                    i++;
2840                    return value;
2841                } else {
2842                    throw new RuntimeException(
```

```
2843                        "quoted value ended too soon, at position " + i
2844                        + " in '" + s + "'");
2845                }
2846            }
```

这里是当遇到"\""和"\'"这两个字符的时候需要特殊处理，在parseQuoted它会将这两个引号内部的所有内容都作为value，那样在Jdbc的参数中使用一个引号就能够解决分隔符的问题了，这里也给我一个提示，在字符串分隔符的时候一定要加上将引号括起来作为一个完整的字符的功能，否则可能会影响到使用（如果没有对引号的特殊处理，hive这个问题将很难解决）。

修改之后的url如下：

```
   Connection connection = DriverManager. getConnection( "Provider=mondrian;" +
           "Jdbc=\"jdbc:hive2://bitest0.server.163.org:10000/foodmart;principal=hive/app-20.photo.163.org@HADOOP.HZ.NETEASE.COM\";
           "Catalog=http://db-53.photo.163.org:16161/cube/queryxml/60548;"
           + "DynamicSchemaProcessor=mondrian.i18n.NrptLocalizingDynamicSchemaProcessor;"
           + "Locale=zh_CN" ,
          null );
```

修改Hive源码

再次尝试一下，之前的错误不再出现了，但是出现了另外的一个错误：

```
Caused by: java.sql.SQLException: Method not supported
   at org.apache.hive.jdbc.HiveDatabaseMetaData.isReadOnly(HiveDatabaseMetaData.java:770)
   at org.apache.commons.dbcp.DelegatingDatabaseMetaData.isReadOnly(DelegatingDatabaseMetaData.java:679)
   at mondrian.spi.impl.JdbcDialectImpl.deduceReadOnly(JdbcDialectImpl.java:196)
   ... 19 more
```

这里说明在hive的jdbc实现中不支持isReadOnly函数，但是抛出了异常，而在mondrian创建连接的时候会调用这个函数，但是没有捕获这个异常，因此就终止了，所以需要修改一下hive的jdbc的源代码，将这个抛异常的语句给注释了。

下载了hive源码，然后只需要编译一下jdbc部分的代码，完成之后修改src/java/org/apache/hive/jdbc/HiveDatabaseMetaData.java文件中的isReadOnly函数，原来是抛出异常，修改为return false。接着再次运行就能够执行MDX查询了。

经过这次配置和测试终于能够尝试使用hive作为mondrian的数据源了，所以功能上是可以实现的，接下来要对我们项目中的数据源进行添加，后期主要的工作应该还是放在数据源对于MDX查询的优化，例如预先计算、加缓存等。

总结

在这次尝试使用mondrian连接hive数据库的时候遇到了三个问题： * hiveServer2的配置和kerberos认证问题，这个让我不得不了解一下kerberos的原理。 * 通过jdbc直接连接hiveserver2完成之后，使用相同的jdbc连接mondrian就出现问题，使用相同的分隔符的问题也阻碍也一段时间，最后通过查看源代码才解决这个问题，另外在遇到使用分隔符的时候一定要加上可以使用引号括起来组成完整一段的方式。 * jdbc源代码需要修改，这个还算是比较轻松，只需要找到问题所在，替换jar包就可以了。

至于性能问题，我使用了相同的MDX查询语句“SELECT NON EMPTY {Hierarchize({[时间].[年份].Members})} ON COLUMNS, NON EMPTY {[Measures].[销售总额]} ON ROWS FROM [TestFoodMart]”对mysql和hive使用相同的数据集进行测试，得到的结果是相同的： Axis #0: {} Axis #1: {[时间].[1998]} Axis #2: {[Measures].[销售总额]} Row #0: 1,079,147

但是mysql使用的时间是300ms左右，hive使用的时间为150000ms左右，差不多500倍的差距。

准备工作：

> - 生成mondrian测试数据库footmart以及导入到hive
> - 使用mysql作为mondrian数据源
> - 部署0.14.0版本的hiveserver服务器