Spark Beeline|ThriftServer

# 1.Beeline使用源码分析

可以使用Beeline连接Spark ThriftServer，进行数据库的操作，使用命令如下：

*bin/beeline -u "jdbc:hive2://hebsjzx-rzjq-master-4-20:10016/default;principal=hive/hivecluster@HEBRZKDC"*

Spark Beeline启动脚本：beeline -> spark-class ，在beeline脚本中，源码如下：

*CLASS="org.apache.hive.beeline.BeeLine"*

*exec "${SPARK\_HOME}/bin/spark-class" $CLASS "$@"*

其最终调用hive的Beeline类，通过JDBC通HiveCli进行连接，其方法调用流程：

*Beeline.main() —> mainWithInputRediretion() —> begin() —> execute()—>dispatch()*

dispatch方法源码如下：

*if (line.startsWith(COMMAND\_PREFIX)) {*

*Map<String, CommandHandler> cmdMap = new TreeMap<String, CommandHandler>();*

*line = line.substring(1);*

*for (int i = 0; i < commandHandlers.length; i++) {*

*String match = commandHandlers[i].matches(line);*

*if (match != null) {*

*CommandHandler prev = cmdMap.put(match, commandHandlers[i]);*

*....*

*}*

*}*

*if (cmdMap.size() > 1) {*

*// any exact match?*

*CommandHandler handler = cmdMap.get(line);*

*if (handler == null) {*

*return error(loc("multiple-matches", cmdMap.keySet().toString()));*

*}*

*return handler.execute(line);*

*}*

*return cmdMap.values().iterator().next()*

*.execute(line);*

*} //以上代码，用于处理 ‘！’前缀的命令*

*else {*

*return commands.sql(line); //执行sql查询*

*}*

SQL的执行是通过调用Commands.sql -> execute -> Statement#execute来执行：

*while (beeLine.getConsoleReader() != null && !(line.trim().endsWith(";"))*

*&& beeLine.getOpts().isAllowMultiLineCommand()) {*

*......*

*String[] cmds = line.split(";");*

*for (int i = 0; i < cmds.length; i++) {*

*String sql = cmds[i].trim();*

*if (sql.length() != 0) {*

*if (sql.startsWith(BeeLine.COMMAND\_PREFIX)) {*

*sql = sql.substring(1);*

*}*

*String prefix = call ? "call" : "sql"; //在beeline中的命令类型：call及sql*

*if (sql.startsWith(prefix)) {*

*sql = sql.substring(prefix.length());*

*}*

*if (beeLine.getBatch() != null) {*

*beeLine.getBatch().add(sql);*

*continue;*

*}*

*try {*

*Statement stmnt = null;*

*boolean hasResults;*

*Thread logThread = null;*

*try {*

*long start = System.currentTimeMillis();*

*if (call) {*

*stmnt = beeLine.getDatabaseConnection().getConnection().prepareCall(sql);*

*hasResults = ((CallableStatement) stmnt).execute();*

*} else {*

*stmnt = beeLine.createStatement();*

*if (beeLine.getOpts().isSilent()) {*

*hasResults = stmnt.execute(sql);*

*} else {*

*logThread = new Thread(createLogRunnable(stmnt));*

*logThread.setDaemon(true);*

*logThread.start();*

*hasResults = stmnt.execute(sql);*

*logThread.interrupt();*

*logThread.join(DEFAULT\_QUERY\_PROGRESS\_THREAD\_TIMEOUT);*

*}*

*}......}*

这里只介绍SQL的Statement的执行，具体实现类为HiveStatement 继承java.sql.statement：

*public boolean execute(String sql) throws SQLException {*

*checkConnection("execute");*

*closeClientOperation();*

*initFlags();*

*//创建Thrift请求*

*TExecuteStatementReq execReq = new TExecuteStatementReq(sessHandle, sql);*

*execReq.setRunAsync(true);*

*execReq.setConfOverlay(sessConf);*

*transportLock.lock();*

*try {*

*//client客户端端连接并将查询请求发送给ThriftServer*

*TExecuteStatementResp execResp = client.ExecuteStatement(execReq);*

*Utils.verifySuccessWithInfo(execResp.getStatus());*

*stmtHandle = execResp.getOperationHandle();*

*isExecuteStatementFailed = false;*

*} .....*

*//结束查询，并获取结果*

*if (!stmtHandle.isHasResultSet()) {*

*return false;*

*}*

*resultSet = new HiveQueryResultSet.Builder(this).setClient(client).setSessionHandle(sessHandle)*

*.setStmtHandle(stmtHandle).setMaxRows(maxRows).setFetchSize(fetchSize)*

*.setScrollable(isScrollableResultset).setTransportLock(transportLock)*

*.build();*

*return true;*

*}*

客户端client:TService.Iface调用的*ExecuteStatement(execReq)*，基于Thrift协议获取请求信息，同时将TCLIService的执行结果以Thrift协议返回，ThriftCliService#ExecuteStatement的源码如下：

*public TExecuteStatementResp ExecuteStatement(TExecuteStatementReq req) throws TException {*

*TExecuteStatementResp resp = new TExecuteStatementResp();*

*try {*

*if(HttpJMXServer.SINGLE.getHiveMxBean() != null)*

*HttpJMXServer.SINGLE.getHiveMxBean().getHiveInfo().addGetHQLCount();*

*SessionHandle sessionHandle = new SessionHandle(req.getSessionHandle());*

*String statement = req.getStatement();*

*Map<String, String> confOverlay = req.getConfOverlay();*

*Boolean runAsync = req.isRunAsync();*

*//此处的cliService为CliService变量*

*OperationHandle operationHandle = runAsync ?*

*cliService.executeStatementAsync(sessionHandle, statement, confOverlay)*

*: cliService.executeStatement(sessionHandle, statement, confOverlay);*

*resp.setOperationHandle(operationHandle.toTOperationHandle());*

*resp.setStatus(OK\_STATUS);*

*if(HttpJMXServer.SINGLE.getHiveMxBean() != null)*

*HttpJMXServer.SINGLE.getHiveMxBean().getHiveInfo().addGetSuccessCount();*

*} catch (Exception e) {*

*LOG.warn("Error executing statement: ", e);*

*resp.setStatus(HiveSQLException.toTStatus(e));*

*}*

*return resp;*

*}*

# 2.Spark ThriftServer

Spark ThriftServer接收到ThriftCli端的请求，处理的结果返回给客户端。使用Beeline与Spark进行交互，本质上是将Hive的ThriftServer上的CLIService重载下，下面是Hive的CLIService代码：

*public OperationHandle executeStatement(SessionHandle sessionHandle, String statement,*

*Map<String, String> confOverlay)*

*throws HiveSQLException {*

*OperationHandle opHandle = sessionManager.getSession(sessionHandle)*

*.executeStatement(statement, confOverlay);*

*LOG.debug(sessionHandle + ": executeStatement()");*

*return opHandle;*

*}*

其SessionManager为*org.apche.hive.service.cli.session.SessionManager*类。

SparkSQLCliService继承CLIService，同时重载init方法，将sessionManager为

SparkSQLSessionManager类：

*private[hive] class SparkSQLCLIService(hiveServer: HiveServer2, sqlContext: SQLContext)*

*extends CLIService(hiveServer)*

*with ReflectedCompositeService {*

*override def init(hiveConf: HiveConf) {*

*setSuperField(this, "hiveConf", hiveConf)*

*val sparkSqlSessionManager = new SparkSQLSessionManager(hiveServer, sqlContext)*

*setSuperField(this, "sessionManager", sparkSqlSessionManager)*

*addService(sparkSqlSessionManager)*

*var sparkServiceUGI: UserGroupInformation = null*

*var httpUGI: UserGroupInformation = null*

*if (UserGroupInformation.isSecurityEnabled) {*

*try {*

*HiveAuthFactory.loginFromKeytab(hiveConf)*

*sparkServiceUGI = Utils.getUGI()*

*setSuperField(this, "serviceUGI", sparkServiceUGI)*

*} catch {*

*case e @ (\_: IOException | \_: LoginException) =>*

*throw new ServiceException("Unable to login to kerberos with given principal/keytab", e)*

*}*

*// Try creating spnego UGI if it is configured.*

*val principal = hiveConf.getVar(ConfVars.HIVE\_SERVER2\_SPNEGO\_PRINCIPAL).trim*

*val keyTabFile = hiveConf.getVar(ConfVars.HIVE\_SERVER2\_SPNEGO\_KEYTAB).trim*

*if (principal.nonEmpty && keyTabFile.nonEmpty) {*

*try {*

*httpUGI = HiveAuthFactory.loginFromSpnegoKeytabAndReturnUGI(hiveConf)*

*setSuperField(this, "httpUGI", httpUGI)*

*} catch {*

*case e: IOException =>*

*throw new ServiceException("Unable to login to spnego with given principal " +*

*s"$principal and keytab $keyTabFile: $e", e)*

*}*

*}*

*}*

*initCompositeService(hiveConf)*

*}*

*......*

*private[thriftserver] trait ReflectedCompositeService { this: AbstractService =>*

*def initCompositeService(hiveConf: HiveConf) {*

*// Emulating `CompositeService.init(hiveConf)`*

*val serviceList = getAncestorField[JList[Service]](this, 2, "serviceList")*

*serviceList.asScala.foreach(\_.init(hiveConf))*

*// Emulating `AbstractService.init(hiveConf)`*

*invoke(classOf[AbstractService], this, "ensureCurrentState", classOf[STATE] -> STATE.NOTINITED)*

*setAncestorField(this, 3, "hiveConf", hiveConf)*

*invoke(classOf[AbstractService], this, "changeState", classOf[STATE] -> STATE.INITED)*

*getAncestorField[Log](this, 3, "LOG").info(s"Service: $getName is inited.")*

*}*

*}*

在Spark HiveThriftServer2中，调用SparkSQLCLIService来接收并处理请求，具体的请求处理由*SparkSQLSessionManager*来完成，该类重载了SessionManager中的openSession方法：

*override def openSession(*

*protocol: TProtocolVersion,*

*username: String,*

*passwd: String,*

*ipAddress: String,*

*sessionConf: java.util.Map[String, String],*

*withImpersonation: Boolean,*

*delegationToken: String): SessionHandle = {*

*val sessionHandle =*

*super.openSession(protocol, username, passwd, ipAddress, sessionConf, withImpersonation,*

*delegationToken)*

*val session = super.getSession(sessionHandle)*

*HiveThriftServer2.listener.onSessionCreated(*

*session.getIpAddress, sessionHandle.getSessionId.toString, session.getUsername)*

*val ctx = if (sqlContext.conf.hiveThriftServerSingleSession) {*

*sqlContext*

*} else {*

*sqlContext.newSession()*

*}*

*ctx.setConf("spark.sql.hive.version", HiveUtils.hiveExecutionVersion)*

*if (sessionConf != null && sessionConf.containsKey("use:database")) {*

*ctx.sql(s"use ${sessionConf.get("use:database")}")*

*}*

*sparkSqlOperationManager.sessionToContexts.put(sessionHandle, ctx)*

*sessionHandle*

*}*

具体的执行由SparkSQLOperationManager来完成，该类重载newExecuteStatementOperation函数，类的实现如下：

*private[thriftserver] class SparkSQLOperationManager()*

*extends OperationManager with Logging {*

*val handleToOperation = ReflectionUtils*

*.getSuperField[JMap[OperationHandle, Operation]](this, "handleToOperation")*

*val sessionToActivePool = new ConcurrentHashMap[SessionHandle, String]()*

*val sessionToContexts = new ConcurrentHashMap[SessionHandle, SQLContext]()*

*override def newExecuteStatementOperation(*

*parentSession: HiveSession,*

*statement: String,*

*confOverlay: JMap[String, String],*

*async: Boolean): ExecuteStatementOperation = synchronized {*

*val sqlContext = sessionToContexts.get(parentSession.getSessionHandle)*

*require(sqlContext != null, s"Session handle: ${parentSession.getSessionHandle} has not been" +*

*s" initialized or had already closed.")*

*val conf = sqlContext.sessionState.conf*

*val runInBackground = async && conf.getConf(HiveUtils.HIVE\_THRIFT\_SERVER\_ASYNC)*

*val operation = new SparkExecuteStatementOperation(parentSession, statement, confOverlay,*

*runInBackground)(sqlContext, sessionToActivePool)*

*handleToOperation.put(operation.getHandle, operation)*

*logDebug(s"Created Operation for $statement with session=$parentSession, " +*

*s"runInBackground=$runInBackground")*

*operation*

*}*

*}*

返回一个SparkExecuteStatementOperation对象。

一条SQL的执行工作在SparkExecuteStatementOperation对象中通过runInternal来调用execute，源代码如下：

*private def execute(): Unit = {*

*statementId = UUID.randomUUID().toString*

*sqlContext.sparkContext.setJobGroup(statementId, statement)*

*try {*

*result = sqlContext.sql(statement) //真正调用的是SQLContext.sql*

*logDebug(result.queryExecution.toString())*

*result.queryExecution.logical match {*

*case SetCommand(Some((SQLConf.THRIFTSERVER\_POOL.key, Some(value)))) =>*

*sessionToActivePool.put(parentSession.getSessionHandle, value)*

*logInfo(s"Setting spark.scheduler.pool=$value for future statements in this session.")*

*case \_ =>*

*}*

*HiveThriftServer2.listener.onStatementParsed(statementId, result.queryExecution.toString())*

*iter = {*

*if (sqlContext.getConf(SQLConf.THRIFTSERVER\_INCREMENTAL\_COLLECT.key).toBoolean) {*

*resultList = None*

*result.toLocalIterator.asScala*

*} else {*

*resultList = Some(result.collect())*

*resultList.get.iterator*

*}*

*}*

*dataTypes = result.queryExecution.analyzed.output.map(\_.dataType).toArray*

*}*

*HiveThriftServer2.listener.onStatementFinish(statementId)*

*}*

http://blog.csdn.net/crispy\_rice/article/details/71273637