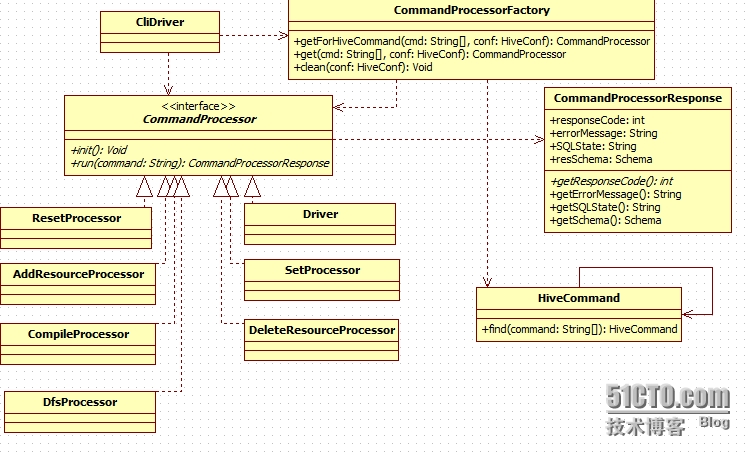
# Hive源码分析(四)-CommandProcessor相关类

 在 上一篇的CliDriver 类中介绍了CliDriver 类会引用到CommandProcessor相关类，主要是根据命令来判断具体实现类,比如通过本地的hive cli启动时，运行hive的命令（非list/source/shell命令等）时在processCmd方法中有如下实现:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | **try** {  CommandProcessor proc = CommandProcessorFactory.get(tokens, (HiveConf) conf);  // 根据命令判断具体的CommandProcessor 实现类  ret = processLocalCmd(cmd, proc, ss);  } **catch** (SQLException e) {      console.printError( "Failed processing command " + tokens[0] + " " + e.getLocalizedMessage(),      org.apache.hadoop.util.StringUtils.stringifyException(e));      ret = 1;  } |

具体的决定什么样的命令对应什么样的具体实现类由 CommandProcessorFactory 规定：如果是set,reset,dfs,add delete,compile等命令，返回对应的CommandProcessor实现类。其余有效命令比如select,insert 都是返回Driver类。

CommandProcessor相关类在org.apache.hadoop.hive.ql.processors包中，类的具体的uml图如下：

[](https://yq.aliyun.com/go/articleRenderRedirect?url=http://s3.51cto.com/wyfs02/M00/4C/F0/wKioL1RHw_LzbuQHAATW62gKz60796.jpg)

简单看下几个类的实现：

1.HiveCommand类，是一个迭代类，定义了非sql的一些语句

|  |
| --- |
| **public** **enum** HiveCommand {     SET(),     RESET(),     DFS(),     ADD(),     DELETE(),     COMPILE();  **private** **static** **final** Set<String> COMMANDS = **new** HashSet<String>();  **static** {  **for** (HiveCommand command : HiveCommand. values()) {         COMMANDS.add(command.name());       }     }  **public** **static** HiveCommand find(String[] command) {  **if** (**null** == command){  **return** **null**;       }      String cmd = command[0];  **if** (cmd != **null**) {         cmd = cmd.trim().toUpperCase();  **if** (command. length > 1 && "role".equalsIgnoreCase(command[1])) {           // special handling for set role r1 statement  **return** **null** ;         } **else** **if** (COMMANDS .contains(cmd)) {  **return** HiveCommand. valueOf(cmd);         }  }  **return** **null**;  }  } |

2.CommandProcessorFactory 类，主要用于获取具体的命令实现类

主要定义了get和getForHiveCommand方法

方法调用get----->getForHiveCommand，其中getForHiveCommand会调HiveCommand 类，HiveCommand 类是一个枚举类型，定义了一些命令。

|  |
| --- |
| getForHiveCommand方法中：  **public** **static** CommandProcessor getForHiveCommand(String[] cmd, HiveConf conf)  **throws** SQLException {      HiveCommand hiveCommand = HiveCommand.find(cmd);  // sql语句返回值为null  **if** (hiveCommand == **null** || isBlank(cmd[0])) {  **return** **null**;      }  **if** (conf == **null**) {        conf = **new** HiveConf();      }      Set<String> availableCommands = **new** HashSet<String>();  **for** (String availableCommand :  conf.getVar(HiveConf.ConfVars.HIVE\_SECURITY\_COMMAND\_WHITELIST).split( ",")) {        availableCommands.add(availableCommand.toLowerCase().trim());      }  **if** (!availableCommands.contains(cmd[0].trim().toLowerCase())) {  **throw** **new** SQLException("Insufficient privileges to execute " + cmd[0], "42000");      }  **switch** (hiveCommand) {  // 每种语句对应的具体的processor类  **case** SET:  **return** **new** SetProcessor();  **case** RESET:  **return** **new** ResetProcessor();  **case** DFS:          SessionState ss = SessionState.get();  **return** **new** DfsProcessor(ss.getConf());  **case** ADD:  **return** **new** AddResourceProcessor();  **case** DELETE:  **return** **new** DeleteResourceProcessor();  **case** COMPILE:  **return** **new** CompileProcessor();  **default**:  **throw** **new** AssertionError("Unknown HiveCommand " + hiveCommand);      }    }  get方法:  **public** **static** CommandProcessor get(String[] cmd, HiveConf conf)  **throws** SQLException {      CommandProcessor result = getForHiveCommand(cmd, conf);  **if** (result != **null**) {  **return** result;  // 如果result不为空，即命令在HiveCommand 的迭代器中定义的话，直接返回对应的结果      }  **if** (isBlank(cmd[0])) {  **return** **null**;      } **else** {  // 为空的话返回Driver类的实例  **if** (conf == **null**) {  **return** **new** Driver();        }        Driver drv = mapDrivers.get(conf);  **if** (drv == **null**) {          drv = **new** Driver();          mapDrivers.put(conf, drv);        }        drv.init();  **return** drv;      }    } |

3.CommandProcessorResponse类封装了processor的返回信息，比如返回码，错误信息等。

4.CommandProcessor 类是一个接口，具体的实现类由下面几个:

|  |
| --- |
| AddResourceProcessor/CompileProcessor/DeleteResourceProcessor/DfsProcessor/ResetProcessor/SetProcessor/Driver |

主要实现方法在各个实现类的run方法中,run方法返回一个CommandProcessorResponse的对象。

下面简单的说下常用的几个实现类：

a.AddResourceProcessor类是处理add xxx命令的。

主要有两个步骤：

1）判断命令的合法性（长度，类型是否在FILE,JAR,ARCHIVE 3种之内）

2）调用SessionState的add\_resource方法(

|  |
| --- |
| SessionState.add\_resource方法---->调用SessionState.downloadResource--->  调用FileSystem的copyToLocalFile方法，把文件下载到本地 |

)

|  |
| --- |
| **public** CommandProcessorResponse run(String command) {      SessionState ss = SessionState.get();      command = **new** VariableSubstitution().substitute(ss.getConf(),command);      String[] tokens = command.split( "\\s+");      SessionState.ResourceType t;  **if** (tokens. length < 2          || (t = SessionState.find\_resource\_type(tokens[0])) == **null**) {        console.printError( "Usage: add ["            + StringUtils.join(SessionState .ResourceType.values(), "|" )            + "] <value> [<value>]\*" );  **return** **new** CommandProcessorResponse(1);      }  **for** ( **int** i = 1; i < tokens.length ; i++) {        String resourceFile = ss.add\_resource(t, tokens[i]);  **if**(resourceFile == **null**){          String errMsg = tokens[i]+ " does not exist.";  **return** **new** CommandProcessorResponse(1,errMsg,**null**);        }      }  **return** **new** CommandProcessorResponse(0);    } |

b.相反的DeleteResourceProcessor是用来处理delete  xxx命令的。

最终调用了SessionState的delete\_resource方法,把resource从HashMap中去掉。

|  |
| --- |
| SessionState的delete\_resource方法  **public** **boolean** delete\_resource(ResourceType t, String value) {  **if** (resource\_map.get(t) == **null**) {  **return** **false**;      }  **if** (t. hook != **null**) {  **if** (!t. hook.postHook( resource\_map.get(t), value)) {  **return** **false**;        }      }  **return** ( resource\_map.get(t).remove(value));    } |

c.DfsProcessor类用来处理dfs 命令，即已“!dfs”开头的命令，最终调用了FsShell的run方法

d.SetProcessor类用来处理set xxx等命令，可以用来设置参数，变量等。

设置参数时

1）以system: 开头的调用了 System.getProperties().setProperty方法。

比如

|  |
| --- |
| hive> set system:user.name=xxxx;  hive> set system:user.name;  system:user.name=xxxx |

2）以hiveconf:开头：

调用了HiveConf的verifyAndSet方法

3）以hivevar:开头：

ss.getHiveVariables().put方法

Driver的实现比较复杂，放在下篇讲解。