链滴

剖析 spark-shell

作者：bian

原文链接：https://ld246.com/article/1497706979127

来源网站：链滴

我们首先来看看spark-shell 到底做了什么，spark-shell 中有一段脚本内容如下:

```
function main() {
  if $cygwin; then
    # Workaround for issue involving JLine and Cygwin
    # (see http://sourceforge.net/p/jline/bugs/40/).
    # If you're using the Mintty terminal emulator in Cygwin, may need to set the
    # "Backspace sends ^H" setting in "Keys" section of the Mintty options
    # (see https://github.com/sbt/sbt/issues/562).
    stty -icanon min 1 -echo > /dev/null 2>&1
    export SPARK_SUBMIT_OPTS="$SPARK_SUBMIT_OPTS -Djline.terminal=unix"
    "${SPARK_HOME}"/bin/spark-submit --class org.apache.spark.repl.Main --name "Spark shel
" "$@"
    stty icanon echo > /dev/null 2>&1
  else
    export SPARK_SUBMIT_OPTS
    "${SPARK_HOME}"/bin/spark-submit --class org.apache.spark.repl.Main --name "Spark shel
" "$@"
  fi
}
```

在上面的脚本中，实际上市执行了spark-submit,查看spark-submit代码：

```
if [ -z "${SPARK_HOME}" ]; then
  export SPARK_HOME="$(cd "`dirname "$0"`"/..; pwd)"
fi

# disable randomized hash for string in Python 3.3+
export PYTHONHASHSEED=0

exec "${SPARK_HOME}"/bin/spark-class org.apache.spark.deploy.SparkSubmit "$@"
```

非常简单，执行spark-class 并传入参数.继续查看spark-class 脚本内容:

```
if [ -z "${SPARK_HOME}" ]; then
  export SPARK_HOME="$(cd "`dirname "$0"`"/..; pwd)"
fi

. "${SPARK_HOME}"/bin/load-spark-env.sh
```

执行load-spark-env.sh 加载环境变量，稍后在讨论这个脚本

```
......
build_command() {
  "$RUNNER" -Xmx128m -cp "$LAUNCH_CLASSPATH" org.apache.spark.launcher.Main "$@"
  printf "%d\0" $?
}
......
CMD=()
while IFS= read -d '' -r ARG; do
  CMD+=("$ARG")
done < <(build_command "$@")
......
CMD=("${CMD[@]:0:$LAST}")
```

```
exec "${CMD[@]}"
```

c从上面可以看出执行了org.apache.spark.launcher.Main ,继续打开org.apache.spark.launcher.Mai 查看代码

```
public static void main(String[] argsArray) throws Exception {
  checkArgument(argsArray.length > 0, "Not enough arguments: missing class name.");

  List<String> args = new ArrayList<>(Arrays.asList(argsArray));
  String className = args.remove(0);

  boolean printLaunchCommand = !isEmpty(System.getenv("SPARK_PRINT_LAUNCH_COMM
ND"));
  AbstractCommandBuilder builder;
  if (className.equals("org.apache.spark.deploy.SparkSubmit")) {
    try {
      builder = new SparkSubmitCommandBuilder(args);
    } catch (IllegalArgumentException e) {
      printLaunchCommand = false;
      System.err.println("Error: " + e.getMessage());
      System.err.println();

      MainClassOptionParser parser = new MainClassOptionParser();
      try {
        parser.parse(args);
      } catch (Exception ignored) {
        // Ignore parsing exceptions.
      }

      List<String> help = new ArrayList<>();
      if (parser.className != null) {
        help.add(parser.CLASS);
        help.add(parser.className);
      }
      help.add(parser.USAGE_ERROR);
      builder = new SparkSubmitCommandBuilder(help);
    }
  } else {
    builder = new SparkClassCommandBuilder(className, args);
  }

  Map<String, String> env = new HashMap<>();
  List<String> cmd = builder.buildCommand(env);
  if (printLaunchCommand) {
    System.err.println("Spark Command: " + join(" ", cmd));
    System.err.println("========================================");
  }

  if (isWindows()) {
    System.out.println(prepareWindowsCommand(cmd, env));
  } else {
    // In bash, use NULL as the arg separator since it cannot be used in an argument.
    List<String> bashCmd = prepareBashCommand(cmd, env);
```

```
      for (String c : bashCmd) {
        System.out.print(c);
        System.out.print('\0');
      }
    }
  }
```

从上面的分析我们可知,spark-submit 传递给spark-class 的参数为org.apache.spark.deploy.SparkSubmit,所以在org.apache.spark.launcher.Main 执行的应该是

```
builder = new SparkSubmitCommandBuilder(args);
```

设置一些参数信息

继续往下执行**builder.buildCommand(env);** 查看 buildCommand 内容

```
@Override
  public List<String> buildCommand(Map<String, String> env)
      throws IOException, IllegalArgumentException {
    if (PYSPARK_SHELL.equals(appResource) && isAppResourceReq) {
      return buildPySparkShellCommand(env);
    } else if (SPARKR_SHELL.equals(appResource) && isAppResourceReq) {
      return buildSparkRCommand(env);
    } else {
      return buildSparkSubmitCommand(env);
    }
  }
```

判断启动时的哪种环境py,shell,or submit 然后构建命令，继续查看**buildSparkSubmitCommand**数

其中有如下代码:

```
...
addPermGenSizeOpt(cmd);
    cmd.add("org.apache.spark.deploy.SparkSubmit");
    cmd.addAll(buildSparkSubmitArgs());
    return cmd;
```

好了继续查看org.apache.spark.deploy.SparkSubmit

spark main 线程dump 信息

```
"main" #1 prio=5 os_prio=31 tid=0x00007f807180c800 nid=0x1c03 runnable [0x000070000308000]
   java.lang.Thread.State: RUNNABLE
    at java.io.FileInputStream.read0(Native Method)
    at java.io.FileInputStream.read(FileInputStream.java:207)
    at jline.internal.NonBlockingInputStream.read(NonBlockingInputStream.java:169)
    - locked <0x00000007830bf508> (a jline.internal.NonBlockingInputStream)
    at jline.internal.NonBlockingInputStream.read(NonBlockingInputStream.java:137)
    at jline.internal.NonBlockingInputStream.read(NonBlockingInputStream.java:246)
    at jline.internal.InputStreamReader.read(InputStreamReader.java:261)
    - locked <0x00000007830bf508> (a jline.internal.NonBlockingInputStream)
    at jline.internal.InputStreamReader.read(InputStreamReader.java:198)
```

```
    - locked <0x00000007830bf508> (a jline.internal.NonBlockingInputStream)
    at jline.console.ConsoleReader.readCharacter(ConsoleReader.java:2145)
    at jline.console.ConsoleReader.readLine(ConsoleReader.java:2349)
    at jline.console.ConsoleReader.readLine(ConsoleReader.java:2269)
    at scala.tools.nsc.interpreter.jline.InteractiveReader.readOneLine(JLineReader.scala:57)
    at scala.tools.nsc.interpreter.InteractiveReader$$anonfun$readLine$2.apply(InteractiveRead
r.scala:37)
    at scala.tools.nsc.interpreter.InteractiveReader$$anonfun$readLine$2.apply(InteractiveRead
r.scala:37)
    at scala.tools.nsc.interpreter.InteractiveReader$.restartSysCalls(InteractiveReader.scala:44)
    at scala.tools.nsc.interpreter.InteractiveReader$class.readLine(InteractiveReader.scala:37)
    at scala.tools.nsc.interpreter.jline.InteractiveReader.readLine(JLineReader.scala:28)
    at scala.tools.nsc.interpreter.ILoop.readOneLine(ILoop.scala:404)
        at scala.tools.nsc.interpreter.ILoop.loop(ILoop.scala:413)
    at scala.tools.nsc.interpreter.ILoop$$anonfun$process$1.apply$mcZ$sp(ILoop.scala:923)
    at scala.tools.nsc.interpreter.ILoop$$anonfun$process$1.apply(ILoop.scala:909)
    at scala.tools.nsc.interpreter.ILoop$$anonfun$process$1.apply(ILoop.scala:909)
    at scala.reflect.internal.util.ScalaClassLoader$.savingContextLoader(ScalaClassLoader.scala:
7)
    at scala.tools.nsc.interpreter.ILoop.process(ILoop.scala:909)
    at org.apache.spark.repl.Main$.doMain(Main.scala:68)
    at org.apache.spark.repl.Main$.main(Main.scala:51)
    at org.apache.spark.repl.Main.main(Main.scala)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
3)
    at java.lang.reflect.Method.invoke(Method.java:497)
    at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMa
n(SparkSubmit.scala:736)
    at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:185)
    at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:210)
    at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:124)
    at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)

   Locked ownable synchronizers:
    - None
```

从堆栈中信息中我们可以看出程序的调用顺序:SparkSubmit.main => repl.Main.main => ILoop.process

ILoop.process 中如下代码:

```
def process(settings: Settings): Boolean = savingContextLoader {
    this.settings = settings
    createInterpreter()

    // sets in to some kind of reader depending on environmental cues
    in = in0.fold(chooseReader(settings))(r => SimpleReader(r, out, interactive = true))
    globalFuture = future {
      intp.initializeSynchronous()
      loopPostInit()
      !intp.reporter.hasErrors
    }
```

```
    loadFiles(settings)
    printWelcome()

    try loop() match {
      case LineResults.EOF => out print Properties.shellInterruptedString
      case _               =>
    }
    catch AbstractOrMissingHandler()
    finally closeInterpreter()

    true
  }
```

在process中我们发现调用了loadFiles并且打印Welcome信息

SparkLoop 继承了loadFiles并且复写了loadFiles 方法 如下:

```
override def loadFiles(settings: Settings): Unit = {
    initializeSpark()
    super.loadFiles(settings)
  }
```

在loadFiles中调度initalizeSpark ,查看源码如下:

```
def initializeSpark() {
    intp.beQuietDuring {
      processLine("""
        @transient val spark = if (org.apache.spark.repl.Main.sparkSession != null) {
            org.apache.spark.repl.Main.sparkSession
          } else {
            org.apache.spark.repl.Main.createSparkSession()
          }
        @transient val sc = {
          val _sc = spark.sparkContext
          _sc.uiWebUrl.foreach(webUrl => println(s"Spark context Web UI available at ${webUrl}")

          println("Spark context available as 'sc' " +
            s"(master = ${_sc.master}, app id = ${_sc.applicationId}).")
          println("Spark session available as 'spark'.")
          _sc
        }
        """)
      processLine("import org.apache.spark.SparkContext._")
      processLine("import spark.implicits._")
      processLine("import spark.sql")
      processLine("import org.apache.spark.sql.functions._")
      replayCommandStack = Nil // remove above commands from session history.
    }
  }
```

从上面可以看出，如果SparkSession 已存在，那么直接返回，否则调用**createSparkSession**

最后从SparkSession中返回SparkContext 查看**createSparkSession**源码

```
def createSparkSession(): SparkSession = {
```

```
    val execUri = System.getenv("SPARK_EXECUTOR_URI")
    conf.setIfMissing("spark.app.name", "Spark shell")
    // SparkContext will detect this configuration and register it with the RpcEnv's
    // file server, setting spark.repl.class.uri to the actual URI for executors to
    // use. This is sort of ugly but since executors are started as part of SparkContext
    // initialization in certain cases, there's an initialization order issue that prevents
    // this from being set after SparkContext is instantiated.
    conf.set("spark.repl.class.outputDir", outputDir.getAbsolutePath())
    if (execUri != null) {
      conf.set("spark.executor.uri", execUri)
    }
    if (System.getenv("SPARK_HOME") != null) {
      conf.setSparkHome(System.getenv("SPARK_HOME"))
    }

    val builder = SparkSession.builder.config(conf)
    if (conf.get(CATALOG_IMPLEMENTATION.key, "hive").toLowerCase == "hive") {
      if (SparkSession.hiveClassesArePresent) {
        // In the case that the property is not set at all, builder's config
        // does not have this value set to 'hive' yet. The original default
        // behavior is that when there are hive classes, we use hive catalog.
        sparkSession = builder.enableHiveSupport().getOrCreate()
        logInfo("Created Spark session with Hive support")
      } else {
        // Need to change it back to 'in-memory' if no hive classes are found
        // in the case that the property is set to hive in spark-defaults.conf
        builder.config(CATALOG_IMPLEMENTATION.key, "in-memory")
        sparkSession = builder.getOrCreate()
        logInfo("Created Spark session")
      }
    } else {
      // In the case that the property is set but not to 'hive', the internal
      // default is 'in-memory'. So the sparkSession will use in-memory catalog.
      sparkSession = builder.getOrCreate()
      logInfo("Created Spark session")
    }
    sparkContext = sparkSession.sparkContext
    Signaling.cancelOnInterrupt(sparkContext)
    sparkSession
  }
```

这里最后使用SparkConf 设置一些必要的参数并且通过Builder 创建sparkSession 并且判断是否需启用Hive的支持。