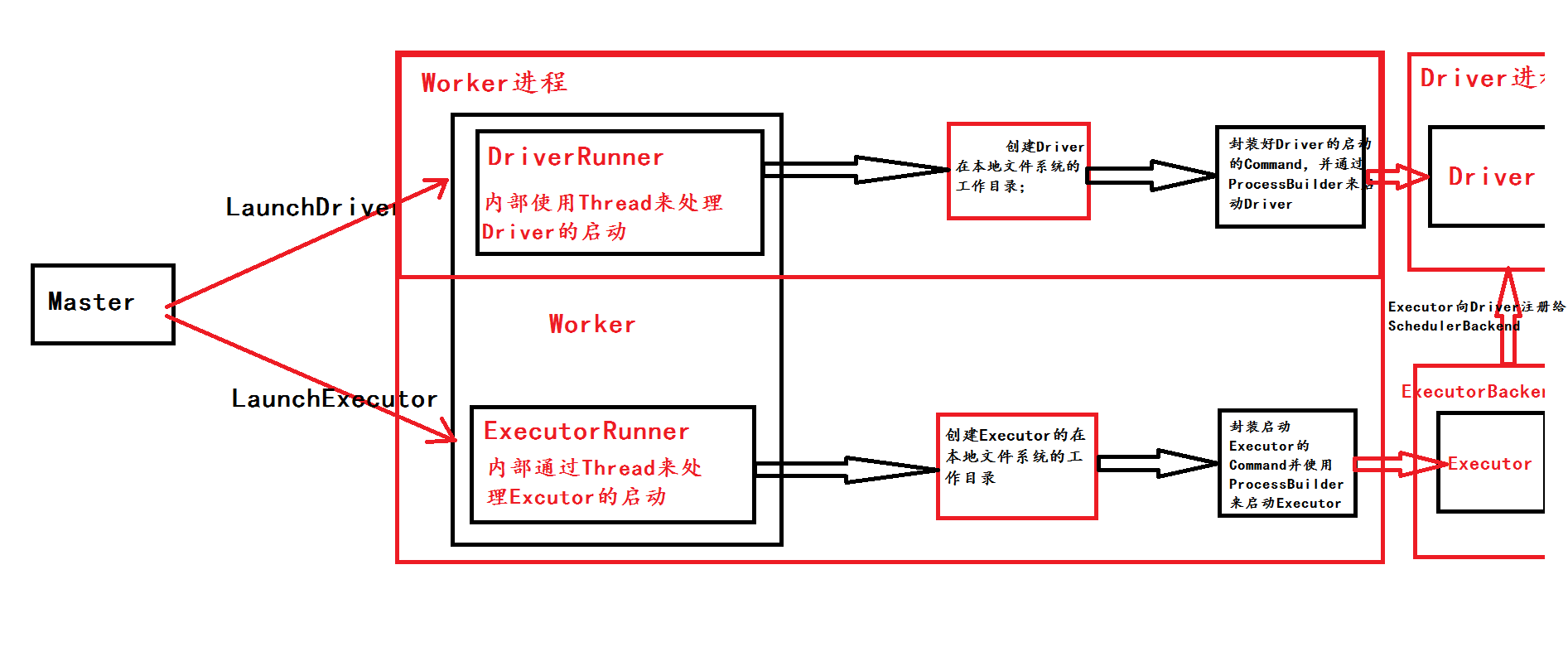


一：Worker原理内幕和流程机制



二：Worker启动Driver内幕

1，Cluster中的Driver失败的时候，如果supervise为true，则启动该Driver的Worker会负责重新启动该Driver；

2，DriverRunner启动进程是通过ProcessBuilder中的process.get.waitFor来完成；

Command：构建进程执行类的入口

LaunchDriver过程

1．launchDriver，Master发送指令给Driver

launchDriver(worker, driver)

2. hashMap里面的K是DriverId, v 是 DriverRunner,worker可能启动很多executor，根据executor的id来管理具体的DriverRunner，DriverRunner内部通过开辟线程的方式，启动了另一个进程。

**val** *drivers* = **new** HashMap[String, DriverRunner]

3. Driver就会new DriverRunner来管理executor

**case** *LaunchDriver*(driverId, driverDesc) => {  
 logInfo(s"Asked to launch driver **$**driverId")  
 **val** driver = **new** DriverRunner(

===========================================

*/\*\*  
 \* Manages the execution of one driver, including automatically restarting the driver on failure.  
 \* This is currently only used in standalone cluster deploy mode.  
 \*/***private**[deploy] **class** DriverRunner(

4. 设置surpervice参数为true，可以如果是standalone cluster的情况时，Driver失败的时候Worker会负责Driver重新启动。

private[deploy] case class DriverDescription(  
 jarUrl: String,  
 mem: Int,  
 cores: Int,  
 supervise: Boolean,  
 command: Command) {  
  
 override def toString: String = s"DriverDescription (${command.mainClass})"  
}

4. DriverRunner启动线程启动和管理Driver

*/\*\* Starts a thread to run and manage the driver. \*/***private**[worker] **def** start() = {  
 **new** Thread("DriverRunner for " + driverId) {  
 **override def** run() {  
 **try** {  
 **val** driverDir = createWorkingDirectory()  
 **val** localJarFilename = downloadUserJar(driverDir)  
  
 **def** substituteVariables(argument: String): String = argument **match** {  
 **case** "{{WORKER\_URL}}" => workerUrl  
 **case** "{{USER\_JAR}}" => localJarFilename  
 **case** other => other  
 }

5. 先创建driver的工作目录

*/\*\*  
 \* Creates the working directory for this driver.  
 \* Will throw an exception if there are errors preparing the directory.  
 \*/***private def** createWorkingDirectory(): File = {  
 **val** driverDir = **new** File(workDir, driverId)  
 **if** (!driverDir.exists() && !driverDir.mkdirs()) {  
 **throw new** IOException("Failed to create directory " + driverDir)  
}  
driverDir  
}

6. 从hdfs上面下载用户提交的Jar文件下载到本地

*/\*\*  
 \* Download the user jar into the supplied directory and return its local path.  
 \* Will throw an exception if there are errors downloading the jar.  
 \*/***private def** downloadUserJar(driverDir: File): String = {  
 **val** jarPath = **new** Path(driverDesc.jarUrl)  
  
 **val** hadoopConf = SparkHadoopUtil.*get*.newConfiguration(conf)  
 **val** destPath = **new** File(driverDir.getAbsolutePath, jarPath.getName)  
 **val** jarFileName = jarPath.getName  
 **val** localJarFile = **new** File(driverDir, jarFileName)  
 **val** localJarFilename = localJarFile.getAbsolutePath

7. command：构建进程执行类的入口

// *TODO: If we add ability to submit multiple jars they should also be added here* **val** builder = CommandUtils.*buildProcessBuilder*(driverDesc.command, securityManager,  
 driverDesc.mem, sparkHome.getAbsolutePath, substituteVariables)  
 launchDriver(builder, driverDir, driverDesc.supervise)  
}

8. 开始luanchDriver

**private def** launchDriver(builder: ProcessBuilder, baseDir: File, supervise: Boolean) {

9. 重定向了stdout和stderr，并且将他加入到

// Redirect stdout and stderr to files  
**val** stdout = **new** File(baseDir, "stdout")  
CommandUtils.*redirectStream*(process.getInputStream, stdout)  
  
**val** stderr = **new** File(baseDir, "stderr")

这样就可以通过log来查看曾经执行的情况。

**object** CommandUtils **extends** Logging {