# TaskRunner

继承自Thread，负责使用一个新的进程（新的JVM）运行用户定义的map和reduce任务，这样可以防止因为用户定义方法导致的错误影响到MapRed框架

属性：

1. **private** TaskTracker.TaskInProgress tip;
2. **private** Task t;
3. **private** TaskTracker tracker;
4. **private** **final** TaskDistributedCacheManager taskDistributedCacheManager;
5. **private** String[] localdirs;
6. **protected** JobConf conf;
7. JvmManager jvmManager;
8. **protected** MapOutputFile mapOutputFile;

for cleaning up old map outputs，什么意思？

方法：

1. **public** **final** **void** run()
   1. 调用prepare()，如果返回false，则直接**return**
   2. 获取classpath：List<String> classPaths = *getClassPaths*(conf, workDir, taskDistributedCacheManager);
   3. 计算JVM参数：Vector<String> vargs = getVMArgs(taskid, workDir, classPaths, logSize);
   4. 获取ulimit设置参数：String setup = getVMSetupCmd();
   5. 获取重定向的文件File[] logFiles = prepareLogFiles(taskid, t.isTaskCleanupTask());（包括stdout和stderr）
   6. errorInfo = getVMEnvironment(errorInfo, user, workDir, conf, env, taskid, logSize);
   7. 开始执行：launchJvmAndWait(setupCmds, vargs, stdout, stderr, logSize, workDir);
   8. 周后调用tip.reportTaskFinished(**false**);通知完成
2. **private** Vector<String> getVMArgs(TaskAttemptID taskid, File workDir, List<String> classPaths, **long** logSize)
   1. 添加java执行路径到vargs
   2. 把配置文件中mapred.map.child.java.opts或mapred.reduce.child.java.opts的值添加到vargs
   3. 创建临时目录Path childTmpDir = *createChildTmpDir*(workDir, conf, **false**);并添加到vargs.add("-Djava.io.tmpdir=" + childTmpDir);
   4. 把classpath添加到vargs：String classPath = StringUtils.*join*(*SYSTEM\_PATH\_SEPARATOR*, classPaths);
   5. 添加log4j参数：setupLog4jProperties(vargs, taskid, logSize);
   6. 添加profile信息到vargs
   7. 添加子进程的主类：vargs.add(Child.**class**.getName());
   8. 主类添加完后，开始添加主类的运行参数，参考Child的main方法，包括umbilical地址和task log位置

## MapTaskRunner

继承自TaskRunner

覆盖了父类的getChildJavaOpts方法，返回配置中的mapred.map.child.java.opts参数

## ReduceTaskRunner

继承自TaskRunner

覆盖了父类的getChildJavaOpts方法，返回配置中的mapred.reduce.child.java.opts参数

# JvmManager

属性：

1. **private** JvmManagerForType mapJvmManager;
2. **private** JvmManagerForType reduceJvmManager;

主要是针对map和reduce封装了JvmManagerForType

## JvmManagerForType

属性：

1. Map <JVMId,TaskRunner> jvmToRunningTask =**new** HashMap<JVMId, TaskRunner>();
2. Map <TaskRunner,JVMId> runningTaskToJvm =**new** HashMap<TaskRunner, JVMId>();

每个jvm可能运行多个task，1和2维护的是正在运行的taks

1. Map <JVMId, JvmRunner> jvmIdToRunner =**new** HashMap<JVMId, JvmRunner>();
2. Map <JVMId, String> jvmIdToPid =**new** HashMap<JVMId, String>();
3. **private** TaskTracker tracker;

方法：

1. **private** **synchronized** **void** reapJvm(TaskRunner t, JvmEnv env)

获取jvm的顺序：

* + 1. 如果没有超过限制，则新建一个JVM
    2. 如果超限了，则查找一个同一job下的空闲JVM
    3. 杀掉一个其他job的空闲KVM
  1. 查询JobID jobId = t.getTask().getJobID();
  2. 当前jvm个数：**int** numJvmsSpawned = jvmIdToRunner.size();
  3. 如果numJvmsSpawned小于maxJvms：
     1. 调用spawnNewJvm(jobId, env, t);新建jvm
  4. 如果numJvmsSpawned大于等于maxJvms：
     1. 遍历jvmIdToRunner，对每个JvmRunner jvmRunner：
        1. JobID jId = jvmRunner.jvmId.getJobId();
        2. 如果jId.equals(jobId)，且jvmRunner.isBusy()为false，且jvmRunner.ranAll()为false：
           1. 可以重用这个jvm：setRunningTaskForJvm(jvmRunner.jvmId, t);
           2. **return**;
           3. 同一个job，且jvmRunner为空闲，且jvmRunner没有运行完全部任务，则可以重用这个jvm
        3. 如果(jId.equals(jobId)且jvmRunner.ranAll()) 或者 （!jId.equals(jobId)且!jvmRunner.isBusy()）：
           1. 可以kill这个jvm：runnerToKill = jvmRunner;
           2. 设置spawnNewJvm = **true**;
           3. 有两种情况要kill一个jvm：

同一个job，要等待这个job运行完所有的任务，即jvmRunner.ranAll()为true

不同的job，要求不是busy状态

* + - * 1. 注意，此处并没有break，因为重用jvm的优先级更高，继续循环下去寻找可以被重用的jvm
    1. 如果循环结束依然没能找到可重用的jvm：
       1. 杀掉找到的空闲jvm：killJvmRunner(runnerToKill);
       2. 新建一个spawnNewJvm(jobId, env, t);

1. **private** **void** spawnNewJvm(JobID jobId, JvmEnv env, TaskRunner t)
   1. 新建JvmRunner jvmRunner = **new** JvmRunner(env, jobId, t.getTask());
   2. 添加到jvmIdToRunner.put(jvmRunner.jvmId, jvmRunner);
   3. 设置该task：setRunningTaskForJvm(jvmRunner.jvmId, t);
   4. 启动jvmRunner.start();

### JvmRunner

继承自Thread

属性：

1. JvmEnv env;
2. **volatile** **boolean** killed = **false**;
3. **volatile** **int** numTasksRan;
4. **final** **int** numTasksToRun;
5. JVMId jvmId;
6. **volatile** **boolean** busy = **true**;
7. **private** ShellCommandExecutor shexec;
8. **private** Task firstTask;
9. **private** List<Task> tasksGiven = **new** ArrayList<Task>();

方法：

1. **public** **void** runChild(JvmEnv env)
   1. 查询TaskRunner runner = jvmToRunningTask.get(jvmId);
   2. 然后调用tracker.getTaskController().launchTask加载task
   3. 最后调用kill();
   4. 更新状态updateOnJvmExit(jvmId, exitCode);
   5. 删除临时文件*deleteWorkDir*(tracker, firstTask);
2. **synchronized** **void** kill()
   1. 如果sleeptimeBeforeSigkill为true：
      1. 新建一个延时发送**new** DelayedProcessKiller(user, pid, sleeptimeBeforeSigkill, Signal.*KILL*).start();
      2. 先发送一个term：controller.signalTask(user, pid, Signal.*TERM*);
   2. 否则，直接发送kill：controller.signalTask(user, pid, Signal.*KILL*);

# Child

# TaskMemoryManagerThread

# DistributedCache