|  |
| --- |
| **public boolean** waitForCompletion(**boolean** verbose) **throws** IOException, InterruptedException, ClassNotFoundException {  **if** (**this**.state == Job.JobState.DEFINE) {  //提交作业的方法  **this**.submit();  }  //waitForCompletion方法传递过来的值  **if** (verbose) {  //显示并且打印日志  **this**.monitorAndPrintJob();  } **else** {  **int** completionPollIntervalMillis = getCompletionPollInterval(**this**.cluster.getConf());   **while**(!**this**.isComplete()) {  **try** {  Thread.sleep((**long**)completionPollIntervalMillis);  } **catch** (InterruptedException var4) {  ;  }  }  }   **return this**.isSuccessful(); } |
| **public void** submit() **throws** IOException, InterruptedException, ClassNotFoundException {  //确定状态  **this**.ensureState(Job.JobState.DEFINE);  //切换新的api，防止出现不兼容效果  **this**.setUseNewAPI();  // 初始化创建Cluster对象，包含两个关键属性： ①文件系统，负责读入数据到程序，写出数据，保存结果  ②运行Job的客户端，如果Job运行方式是Local，使用LocalJobRunner，如果Job运行方式是YARN，使用YarnRunner  **this**.connect();  **final** JobSubmitter submitter = **this**.getJobSubmitter(**this**.cluster.getFileSystem(), **this**.cluster.getClient());  **this**.status = (JobStatus)**this**.ugi.doAs(**new** PrivilegedExceptionAction<JobStatus>() {  **public** JobStatus run() **throws** IOException, InterruptedException, ClassNotFoundException {  **return** submitter.submitJobInternal(Job.**this**, Job.**this**.cluster);  }  });  //提交job任务  **this**.state = Job.JobState.RUNNING;  LOG.info(**"The url to track the job: "** + **this**.getTrackingURL()); } |
| **private synchronized void** connect() **throws** IOException, InterruptedException, ClassNotFoundException {  //根据用户的configuration，创建相应的Cluster，负责运行Job  **if** (**this**.cluster == **null**) {  //这是提供链接集群的一种链接方式，有本地的和yarn的  **this**.cluster = (Cluster)**this**.ugi.doAs(**new** PrivilegedExceptionAction<Cluster>() {  **public** Cluster run() **throws** IOException, InterruptedException, ClassNotFoundException {  **return new** Cluster(Job.**this**.getConfiguration());  }  });  }  } |
| **public** Cluster(InetSocketAddress jobTrackAddr, Configuration conf) **throws** IOException {  //文件系统  **this**.fs = **null**;  //path路径  **this**.sysDir = **null**;  //作业资源存放  **this**.stagingAreaDir = **null**;  //历史路径  **this**.jobHistoryDir = **null**;  //客户端通信协议提供者  **this**.providerList = **null**;  //配置  **this**.conf = conf;  //用户信息  **this**.ugi = UserGroupInformation.getCurrentUser();  //完成初始化  **this**.initialize(jobTrackAddr, conf); } |
| **private void** initialize(InetSocketAddress jobTrackAddr, Configuration conf) **throws** IOException {  **this**.initProviderList();  //已迭代的方式循环客户端的provider  Iterator i$ = **this**.providerList.iterator();  //通过create方法构建实例  **while**(i$.hasNext()) {  //ctrl+h会看到这个类的两个子类一个是local的一个yarn的  ClientProtocolProvider provider = (ClientProtocolProvider)i$.next();  LOG.debug(**"Trying ClientProtocolProvider : "** + provider.getClass().getName());  ClientProtocol clientProtocol = **null**;   **try** {  **if** (jobTrackAddr == **null**) {  clientProtocol = provider.create(conf);  } **else** {  clientProtocol = provider.create(jobTrackAddr, conf);  }   **if** (clientProtocol != **null**) {  **this**.clientProtocolProvider = provider;  **this**.client = clientProtocol;  LOG.debug(**"Picked "** + provider.getClass().getName() + **" as the ClientProtocolProvider"**);  **break**;  }   LOG.debug(**"Cannot pick "** + provider.getClass().getName() + **" as the ClientProtocolProvider - returned null protocol"**);  } **catch** (Exception var7) {  LOG.info(**"Failed to use "** + provider.getClass().getName() + **" due to error: "**, var7);  }  }   **if** (**null** == **this**.clientProtocolProvider || **null** == **this**.client) {  **throw new** IOException(**"Cannot initialize Cluster. Please check your configuration for mapreduce.framework.name and the correspond server addresses."**);  } } |
| JobStatus submitJobInternal(Job job, Cluster cluster) **throws** ClassNotFoundException, InterruptedException, IOException {  // 验证输出目录是否合法和存在  **this**.checkSpecs(job);  Configuration conf = job.getConfiguration();  //将添加配置到分布式缓存  addMRFrameworkToDistributedCache(conf);  //获取当前job作业的作用区域  Path jobStagingArea = JobSubmissionFiles.getStagingDir(cluster, conf);  //获取ip地址  InetAddress ip = InetAddress.getLocalHost();  **if** (ip != **null**) {  //设置提交作业的主机地址和主机名  **this**.submitHostAddress = ip.getHostAddress();  **this**.submitHostName = ip.getHostName();  conf.set(**"mapreduce.job.submithostname"**, **this**.submitHostName);  conf.set(**"mapreduce.job.submithostaddress"**, **this**.submitHostAddress);  } //生成job id  JobID jobId = **this**.submitClient.getNewJobID();  //将jobid添加到整个job  job.setJobID(jobId);  //提交job作业的路径  Path submitJobDir = **new** Path(jobStagingArea, jobId.toString());  JobStatus status = **null**;   JobStatus var24;  **try** {  //设置一些参数  conf.set(**"mapreduce.job.user.name"**, UserGroupInformation.getCurrentUser().getShortUserName());  conf.set(**"hadoop.http.filter.initializers"**, **"org.apache.hadoop.yarn.server.webproxy.amfilter.AmFilterInitializer"**);  conf.set(**"mapreduce.job.dir"**, submitJobDir.toString());  LOG.debug(**"Configuring job "** + jobId + **" with "** + submitJobDir + **" as the submit dir"**);  //获取路径的权限  TokenCache.obtainTokensForNamenodes(job.getCredentials(), **new** Path[]{submitJobDir}, conf);  **this**.populateTokenCache(conf, job.getCredentials());  //获取密钥和令牌，并且将他们存储到令牌的缓存TokenCache中  **if** (TokenCache.getShuffleSecretKey(job.getCredentials()) == **null**) {  KeyGenerator keyGen;  **try** {  keyGen = KeyGenerator.getInstance(**"HmacSHA1"**);  keyGen.init(64);  } **catch** (NoSuchAlgorithmException var19) {  **throw new** IOException(**"Error generating shuffle secret key"**, var19);  }   SecretKey shuffleKey = keyGen.generateKey();  TokenCache.setShuffleSecretKey(shuffleKey.getEncoded(), job.getCredentials());  }   **if** (CryptoUtils.isEncryptedSpillEnabled(conf)) {  conf.setInt(**"mapreduce.am.max-attempts"**, 1);  LOG.warn(**"Max job attempts set to 1 since encrypted intermediatedata spill is enabled"**);  }  //配置相关的文件  **this**.copyAndConfigureFiles(job, submitJobDir);  //获取配置文件路径  Path submitJobFile = JobSubmissionFiles.getJobConfPath(submitJobDir);  LOG.debug(**"Creating splits at "** + **this**.jtFs.makeQualified(submitJobDir));  **int** maps = **this**.writeSplits(job, submitJobDir);  //获取map的数量  conf.setInt(**"mapreduce.job.maps"**, maps);  LOG.info(**"number of splits:"** + maps);  //获取队列  String queue = conf.get(**"mapreduce.job.queuename"**, **"default"**);  //获取队列的访问权限列表  AccessControlList acl = **this**.submitClient.getQueueAdmins(queue);  conf.set(QueueManager.toFullPropertyName(queue, QueueACL.ADMINISTER\_JOBS.getAclName()), acl.getAclString());  //清空令牌  TokenCache.cleanUpTokenReferral(conf);  //根据参数判断是否追踪令牌ID  **if** (conf.getBoolean(**"mapreduce.job.token.tracking.ids.enabled"**, **false**)) {  //通过job获取令牌ID，并且缓存到trackingIds 中  ArrayList<String> trackingIds = **new** ArrayList();  Iterator i$ = job.getCredentials().getAllTokens().iterator();   **while**(i$.hasNext()) {  Token<? **extends** TokenIdentifier> t = (Token)i$.next();  trackingIds.add(t.decodeIdentifier().getTrackingId());  }   conf.setStrings(**"mapreduce.job.token.tracking.ids"**, (String[])trackingIds.toArray(**new** String[trackingIds.size()]));  }   ReservationId reservationId = job.getReservationId();  **if** (reservationId != **null**) {  conf.set(**"mapreduce.job.reservation.id"**, reservationId.toString());  }   **this**.writeConf(conf, submitJobFile);  **this**.printTokens(jobId, job.getCredentials());  //提交jobid，作业路径，令牌  //这里的submitClient就是之前的LocalJobRunning和YARNRunning  status = **this**.submitClient.submitJob(jobId, submitJobDir.toString(), job.getCredentials());  **if** (status == **null**) {  **throw new** IOException(**"Could not launch job"**);  }   var24 = status;  } **finally** {  **if** (status == **null**) {  LOG.info(**"Cleaning up the staging area "** + submitJobDir);  **if** (**this**.jtFs != **null** && submitJobDir != **null**) {  //注销  **this**.jtFs.delete(submitJobDir, **true**);  }  }   }   **return** var24; } |
|  |