# JobContext

/\*\*

\* A read-only view of the job that is provided to the tasks while they

\* are running.

\*/

## JobID

由三部分构成：

job\_200707121733\_0003

job + JT的id + jobid

# Job

继承自JobContext，是用户的视图。

/\*\*

\* The job submitter's view of the Job. It allows the user to configure the

\* job, submit it, control its execution, and query the state. The set methods

\* only work until the job is submitted, afterwards they will throw an

\* IllegalStateException.

\*/

简单的把用户指定的一些属性提交到JobConf中，有一点要注意：

setJarByClass方法只是根据传入的class的值，查找其所在的jar包，要求要用到的所有类（包括map，reduce，combine等）都必须包含在这个jar包中。这个传入的值没有特殊的要求，只要能找到那个必须的jar包就可以

方法：

1. **public** **boolean** waitForCompletion(**boolean** verbose)
   1. 如果state为*DEFINE*，即还在定义阶段，就调用submit();提交
   2. 如果verbose为true，则要打印相关的job运行信息到控制台：jobClient.monitorAndPrintJob(conf, info);
   3. 否则，不打印消息，但是也要等待job运行结束才能退出：info.waitForCompletion();
2. **public** **void** submit()
   1. 连接到JT：connect();(其实就是初始化jobClient)
   2. 提交：info = jobClient.submitJobInternal(conf);
   3. 更新状态：state = JobState.*RUNNING*;

# JobClient

用于和JT交互

执行过程：

1. Checking the input and output specifications of the job.
2. Computing the [InputSplit](eclipse-javadoc:%E2%98%82=hadoop/src%5C/mapred%3Corg.apache.hadoop.mapred%7BJobClient.java%E2%98%83JobClient%E2%98%82InputSplit)s for the job.
3. Setup the requisite accounting information for the [DistributedCache](eclipse-javadoc:%E2%98%82=hadoop/src%5C/mapred%3Corg.apache.hadoop.mapred%7BJobClient.java%E2%98%83JobClient%E2%98%82DistributedCache) of the job, if necessary.
4. Copying the job's jar and configuration to the map-reduce system directory on the distributed file-system.
5. Submitting the job to the JobTracker and optionally monitoring it's status.

属性：

1. **private** JobSubmissionProtocol jobSubmitClient;
2. **private** Path sysDir = **null**;
3. **private** Path stagingAreaDir = **null**;
4. **private** FileSystem fs = **null**;
5. **private** UserGroupInformation ugi;
6. **private** **static** **final** String *TASKLOG\_PULL\_TIMEOUT\_KEY* = "mapreduce.client.tasklog.timeout";
7. **private** **static** **final** **int** *DEFAULT\_TASKLOG\_TIMEOUT* = 60000;
8. **static** **int** *tasklogtimeout*;

方法：

1. **public** **void** init(JobConf conf)
   1. 获取String tracker = conf.get("mapred.job.tracker", "local");
   2. 如果tracker是"local"，使用本地的JT：
      1. 设置map个数为1：conf.setNumMapTasks(1);
      2. 设置**this**.jobSubmitClient = **new** LocalJobRunner(conf);
   3. 否则，连接到远程的JT：**this**.jobSubmitClient = *createRPCProxy*(JobTracker.*getAddress*(conf), conf);
2. **private** **void** copyAndConfigureFiles(JobConf job, Path submitJobDir, **short** replication)
   1. 如果submitJobDir文件夹存在（每个job对应唯一的存储目录，见submitJobInternal中的例子），则抛出IOException，
   2. 创建这个目录FileSystem.*mkdirs*(fs, submitJobDir, mapredSysPerms);
   3. 构造三个目录，filesDir，archivesDir，libjarsDir，分别是上面的job目录下的三个子目录：files，archives和libjar
   4. 从配置中查询三个属性：
      1. String files = job.get("tmpfiles");如果files不为null：
         1. 创建目录FileSystem.*mkdirs*(fs, filesDir, mapredSysPerms);
         2. 把指定的文件（slpit files）复制到hdfs上的filesDir目录下
         3. 添加到DistributedCache
      2. String libjars = job.get("tmpjars");

类似files

* + 1. String archives = job.get("tmparchives");

类似files

* 1. 使用TrackerDistributedCacheManager检测上面的三种目录文件
  2. 获取jar包：String originalJarPath = job.getJar();(参考Job中的说明，此处获取需要分布式运行的class所在的jar包)
  3. 获取jar包存放的位置：Path submitJarFile = JobSubmissionFiles.*getJobJar*(submitJobDir);（即job存储目录下的job.jar文件）
  4. 重新设置jar地址：job.setJar(submitJarFile.toString());
  5. 复制本地jar包到job目录：fs.copyFromLocalFile(**new** Path(originalJarPath), submitJarFile);
  6. 设置jar文件的副本fs.setReplication(submitJarFile, replication);（默认为10，设置这么大，可能是为了本地化？）

1. RunningJob submitJobInternal(**final** JobConf job)
   1. 获取mapred文件存储地址：Path jobStagingArea = JobSubmissionFiles.*getStagingDir*(JobClient.**this**, jobCopy);（eg：hdfs://192.168.1.121:9000/tmp/hadoop-gaoxiao/mapred/staging/gaoxiao/.staging）
   2. 获取jobId：JobID jobId = jobSubmitClient.getNewJobId();并构建对应的存储目录：Path submitJobDir = **new** Path(jobStagingArea, jobId.toString());
   3. 设置属性：jobCopy.set("mapreduce.job.dir", submitJobDir.toString());
   4. 调用copyAndConfigureFiles(jobCopy, submitJobDir);上传文件
   5. 获取job配置文件：Path submitJobFile = JobSubmissionFiles.*getJobConfPath*(submitJobDir); （即job存储目录下的job.xml文件）
   6. 检查输出目录：output.checkOutputSpecs(context);
   7. 创建splits：**int** maps = writeSplits(context, submitJobDir);然后设置jobCopy.setNumMapTasks(maps);
   8. 写job.xml到job目录（即job的配置文件）：jobCopy.writeXml(out);
   9. 提交到JT：status = jobSubmitClient.submitJob(jobId, submitJobDir.toString(), jobCopy.getCredentials());
   10. 获取JobProfile prof = jobSubmitClient.getJobProfile(jobId);
   11. 返回**new** NetworkedJob(status, prof, jobSubmitClient);

## NetworkedJob

实现了RunningJob接口，负责和正在运行的job交互，获取job的信息

属性：

1. **private** JobSubmissionProtocol jobSubmitClient;
2. JobProfile profile;
3. JobStatus status;
4. **long** statustime;

# JobStatus

描述job当前的状态

属性：

1. **private** JobID jobid;
2. **private** **float** mapProgress;
3. **private** **float** reduceProgress;
4. **private** **float** cleanupProgress;
5. **private** **float** setupProgress;
6. **private** **int** runState;
7. **private** **long** startTime;
8. **private** String user;
9. **private** JobPriority priority;
10. **private** String schedulingInfo="NA";
11. **private** String failureInfo = "NA";

# JobProfile

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* A JobProfile is a MapReduce primitive. Tracks a job,

\* whether living or dead.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

上面的注释貌似不对，因为JobProfile并没有job的状态信息？这应该是JobStatus的功能

属性：

1. String user;
2. **final** JobID jobid;
3. String jobFile;
4. String url;
5. String name;
6. String queueName;

# JobSplitWriter

/\*\*

\* The class that is used by the Job clients to write splits (both the meta

\* and the raw bytes parts)

\*/

meta（按顺序）：

1. "META-SPL"
2. splitMetaInfoVersion
3. split个数
4. 对每个split：
   1. locations
   2. startOffset（注意，这个offset是指split在raw文件中的偏移）
   3. inputDataLength

raw（按顺序）：

1. "SPL"
2. *splitVersion*
3. 对每个split：
   1. split.getClass().getName()
   2. split的实体

# JobSplit

split信息分为两部分：

1. split元数据

JT用来计算数据本地化的信息

1. split信息

map任务用来获取数据

## SplitMetaInfo

属性：

1. **private** **long** startOffset;

start offset in actual split

1. **private** **long** inputDataLength;

data length that will be processed in this split

1. **private** String[] locations;

hosts on which this split is local

## TaskSplitMetaInfo

属性：

1. **private** TaskSplitIndex splitIndex;
2. **private** **long** inputDataLength;
3. **private** String[] locations;

## TaskSplitIndex

属性：

1. **private** String splitLocation;
2. **private** **long** startOffset;