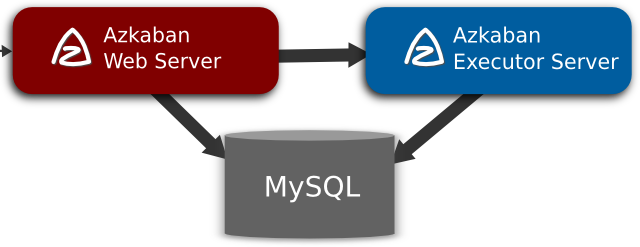
Apache Azkaban

数据分析系统由大量的任务单元组成：shell脚本程序、java程序、MR程序等，各个任务之间存在时间先后及前后依赖关系，为了组织这样的复杂的执行计划，需要一个工作流任务调度器。Azkaban是由Linkedin开源的批量工作流任务调度器，用于在一个工作流内以一个特定的顺序运行一组工作和流程。Azkaban由以下3个组件构成：



1. **MySQL数据库**

Azkaban使用MySQL来存储状态信息，如项目名称、描述、权限、任务状态及SLA规则等，We Server和ExecutorServer都需要访问数据库。

1. **Azkaban WebServer**

Azkaban WebSever是整个工作流的主要管理者，其负责Project管理、用户登录认证、定制执行工作流、跟踪工作流执行进度等。同时，其还提供Web服务接口，使用该接口，用户可以使用curl或者ajax来执行相关操作。WebServer在以下几个方面使用数据库：

* Project管理
* 跟踪工作流执行进度
* 访问历史工作流的运行信息
* 定时执行工作流任务
* 记录所有SLA规则

1. **Azkaban ExecutorServer**

用于执行任务，并把任务执行的输出日志保存到MySQL。ExecutorServer使用以下几个方面使用数据库：

* 获取Project信息
* 执行工作流
* 存储工作流运行日志
* 如果一个工作流在不同执行器上运行，从DB中获取状态

Azkaban定义了KV文件格式来建立任务之间的依赖关系，并提供一个易于使用的Web用户维护和跟踪工作流。

# Azkaban的编译安装

Azkaban有三种运行模式：

* Solo Server Mode，最简单的模式（内置H2数据库），WebServer和ExecutorServer都在一个进程中运行，任务量不大的项目可以采用此模式
* Two Server Mode，数据库为MySQL，WebServer和ExecutorServer在不同进程，这种模式下两个进程之间互不影响
* Multiple Executor Mode，该模式与Two Server Mode不同的是有多个ExecutorServer

下面介绍第二种方式的安装模式

1. 下载源代码并编译

*git clone <https://github.com/azkaban/azkaban.git>*

*cd azkaban/ //编译成tar包*

*./gradlew distTar //将编译好的tar包拷贝出来*

*cp azkaban/azkaban-\*/build/distributions/\*.tar.gz /opt/soft/azkabanTar*

编译成功后，所用tarball如下：

*azkaban-db-3.48.0-5-gbff791b.tar.gz*

*azkaban-solo-server-3.48.0-5-gbff791b.tar.gz*

*azkaban-exec-server-3.48.0-5-gbff791b.tar.gz*

*azkaban-web-server-3.48.0-5-gbff791b.tar.gz*

*azkaban-hadoop-security-plugin-3.48.0-5-gbff791b.tar.gz*

1. 安装MySQL

安装后，创建azkaban数据库并创建azkaban用户，密码为azkaban，并设置权限

*mysql> CREATE DATABASE azkaban;*

*mysql> CREATE USER 'azkaban'@'%' IDENTIFIED BY 'azkaban';*

*mysql> GRANT SELECT,INSERT,UPDATE,DELETE ON azkaban.\* to 'azkaban'@'%' WITH GRANT OPTION;*

解压azkaban-db-3.48.0-5-gbff791b.tar.gz，并进入目录后进行数据库初始化：

*mysql> use azkaban*

*mysql> source create-all-sql-3.48.0-5-gbff791b.sql*

创建的表如下：

*mysql> show tables;*

*| Tables\_in\_azkaban |*

*| QRTZ\_BLOB\_TRIGGERS |*

*| QRTZ\_CALENDARS |*

*29 rows in set (0.00 sec)*

1. 安装azkaban-web-server

解压azkaban-web-server-3.48.0-5-gbff791b.tar.gz

* 创建SSL配置，执行命令：

*keytool -keystore keystore -alias jetty -genkey -keyalg RSA*

输入所需要的信息后，将生成的keystore文件复制到conf目录下

* 配置azkaban.properties，重要的配置为：

*......*

*azkaban.default.servlet.path=/index*

*web.resource.dir=web/*

*default.timezone.id=America/Los\_Angeles*

*# Azkaban UserManager class*

*user.manager.class=azkaban.user.XmlUserManager*

*user.manager.xml.file=conf/azkaban-users.xml*

*# Loader for projects*

*executor.global.properties=conf/global.properties*

*azkaban.project.dir=projects*

*......*

*database.type=mysql*

*mysql.port=3306*

*mysql.host=10.139.4.83*

*mysql.database=azkaban*

*mysql.user=azkaban*

*mysql.password=123456*

*mysql.numconnections=100*

* 配置azkaban-users.xml，增加管理员用户

*<azkaban-users>*

*<user username="azkaban" password="azkaban" roles="admin" groups="azkaban" />*

*<user username="metrics" password="metrics" roles="metrics"/>*

*<user username="admin" password="admin" roles="admin,metrics"/>*

*<role name="admin" permissions="ADMIN" />*

*<role name="metrics" permissions="METRICS"/>*

*</azkaban-users>*

* 配置log4j.properties

*log4j.rootLogger=INFO,C*

*log4j.appender.C=org.apache.log4j.ConsoleAppender*

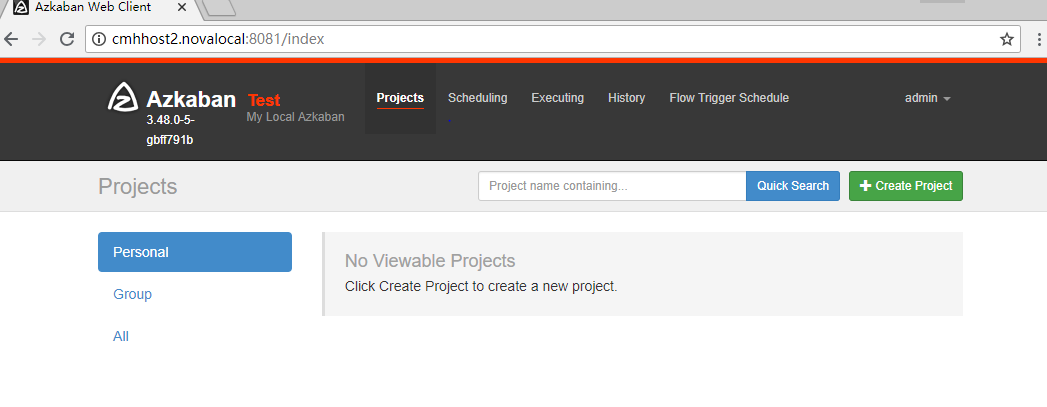
*log4j.appender.C.Target=System.err*

*log4j.appender.C.layout=org.apache.log4j.PatternLayout log4j.appender.C.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n*

启动命令如下：

*bin/start-web.sh*

默认端口为8081，启动后页面如下：



1. 安装及启动ExecutorServer

解压azkaban-exec-server-3.48.0-5-gbff791b.tar.gz，进行配置核心参数：

*executor.port=12321*

*executor.maxThreads=50*

*executor.flow.threads=30*

*executor.connector.stats=true*

*......*

启动命令如下：

*bin/start-exec.sh*

# Azkaban的使用

在Azkaban中主要的元素包括以下三个：

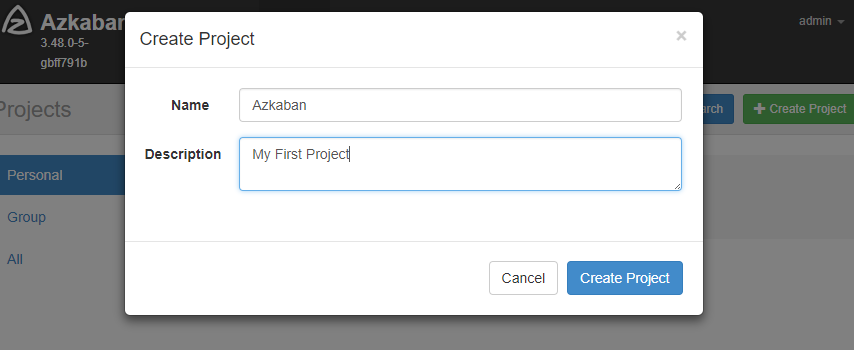
1. Project，可以理解为某个项目，项目中包含许多需要执行的任务
2. Job，Azkaban可调度Job类型包括：linux命令、脚本、Java程序、Hadoop MR、Hive及Spark等。
3. Flow，工作流，由多个Job组成

通过界面使用Azkaban，首页包含四个菜单：

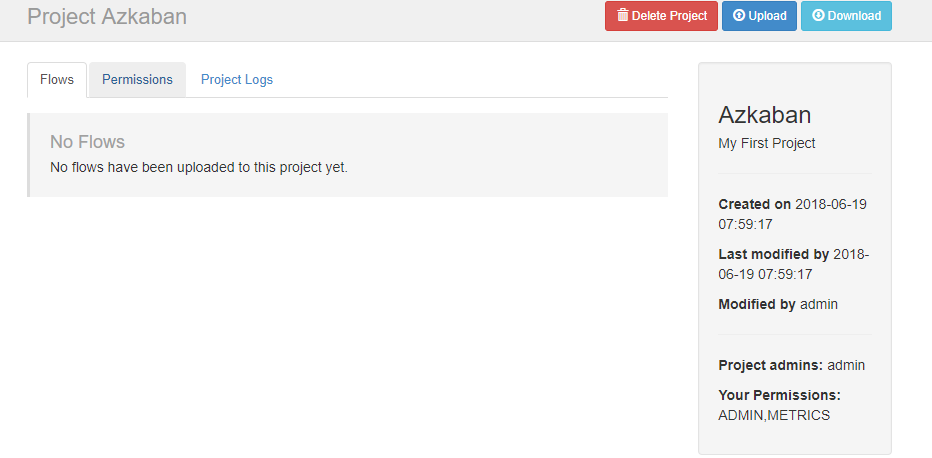
* Projects，创建工程，所有的Flows在该工程中珍惜
* Scheduling，显式定时任务
* Executiong，显式当前运行的任务
* History，显式历史运行任务

## 创建工程

点击Web界面右上角create project，在弹出的窗口中写入工程名和描述，创建工程



创建成功后，在界面上显式如下：



## **创建Job并提交**

创建Job，仅需要创建一个以.job结尾的文本文件即可，比如：

*# foo.job*

*type=command*

*command=echo foo*

如果是多个Job并且有依赖关系，可以使用dependencies参数指定依赖关系，比如：

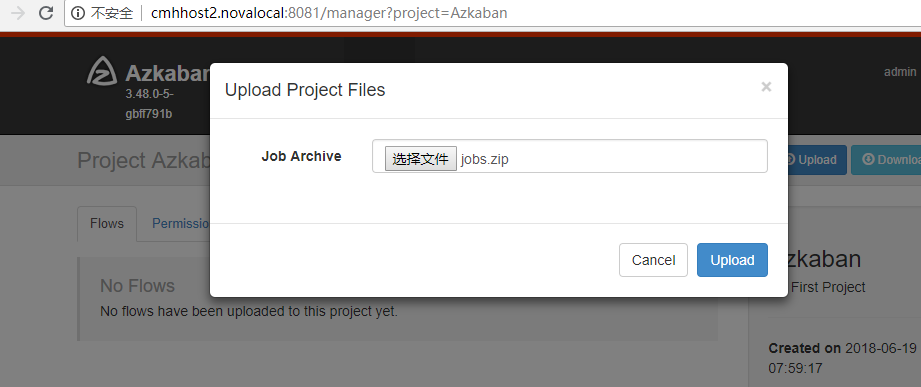
*# bar.job*

*type=command*

*dependencies=foo*

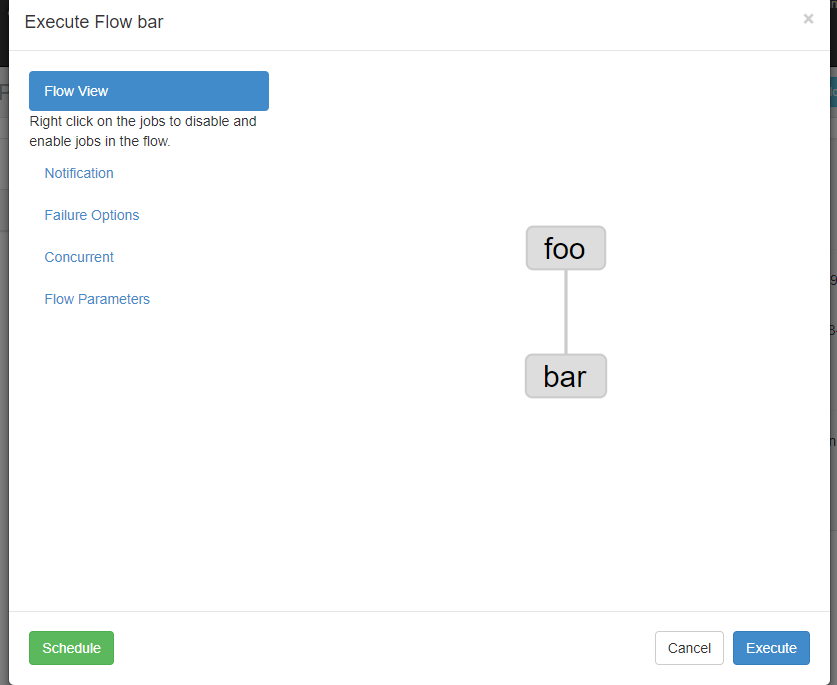
*command=echo bar*

这样就创建好了Job。将这两个Job打包成zip，在页面上点击update上传，如下图所示：



## **执行**

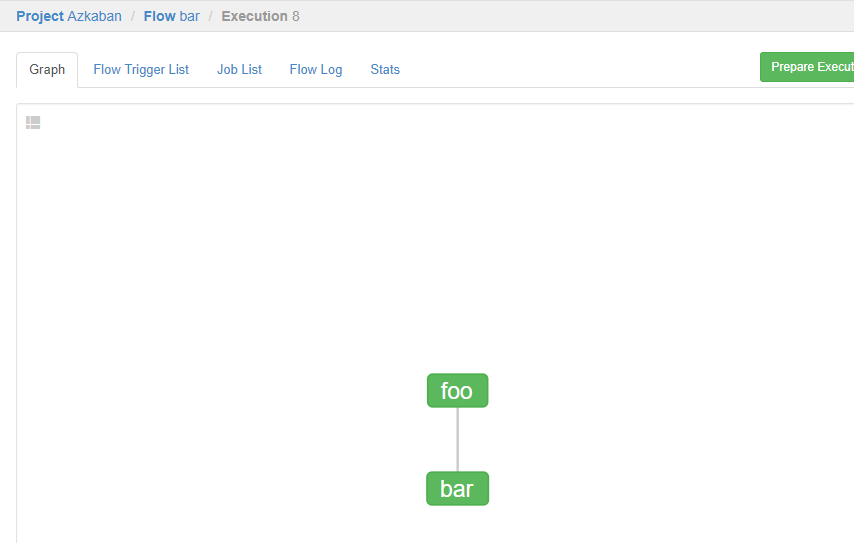
上传后点击Execute Flow，执行如下：



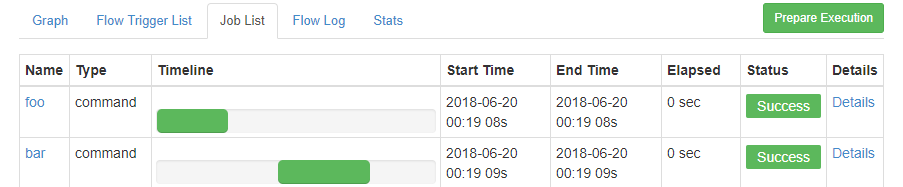
左边的选项卡依次：

* Flow View：流程视图，可以禁用或者启用某些Job
* Notification: 定义任务或者失败是否发送邮件
* Failure Options: 定义一个Job失败后，剩下的Job怎么执行
* Concurrent，并行任务执行设置
* Flow Parameters，参数设置

左下角的Schedule为设置调度执行，右下角的Execute为直接运行，运行之后在Graph中可以看到执行结果如下：



在Job List中可以看到Flow的执行起始终止时间：



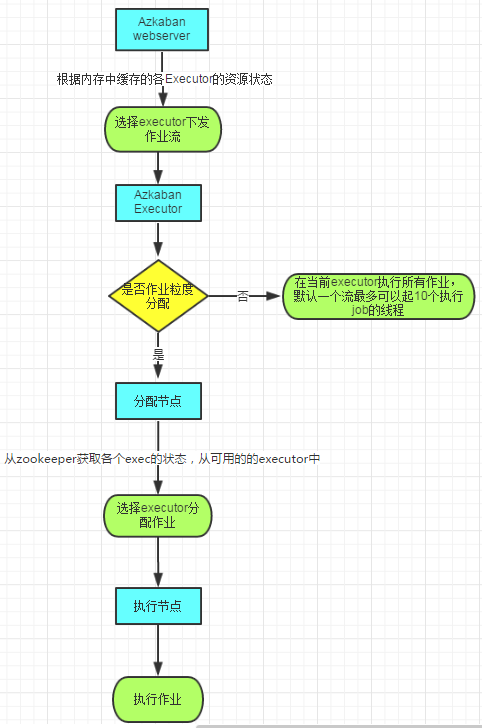
这样工作流的调度就执行成功。

# Azkaban源码分析

Azkaban为开源的调度系统，其执行代码主要分为三个模块：

* webServer，主要提供Web界面展示和仅接收Http请求的模块
* execserver，具体执行任务的模块
* Common，公共模块，提供访问数据库及告警等公共类

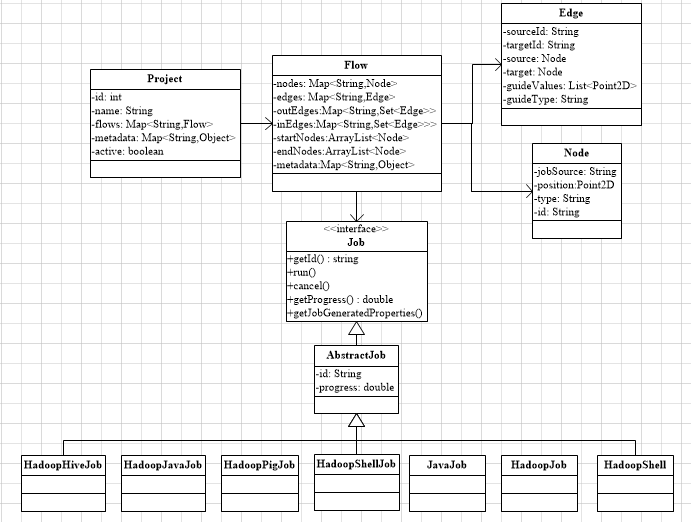
Azkaban作业流执行过程如下：



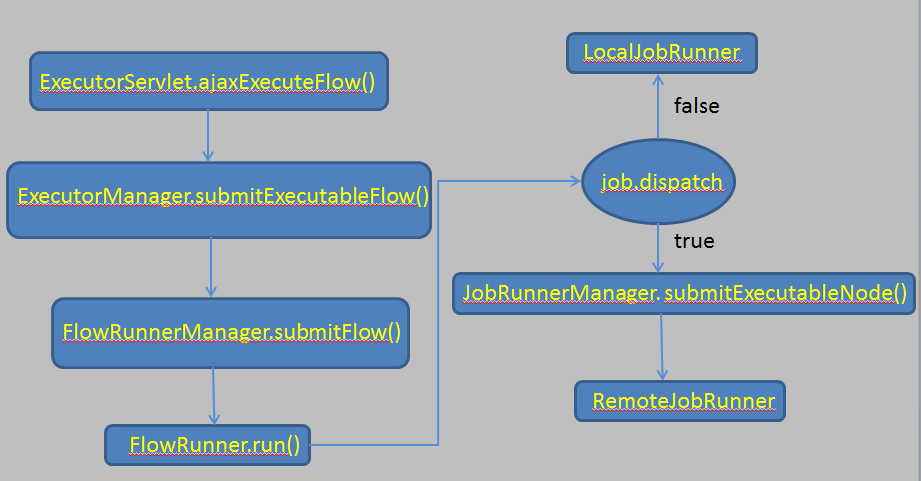
1. WebServer根据内存中缓存的各Executor的资源状态（通过线程遍历Active Executor，去发送Http请求获取其资源状态信息缓存到内存中），按照选择策略（根据Executor资源状态、最近执行流的个数）选择一个Executor下发作业流
2. 然后Executor判断是否设置作业粒度分配，为设置则当前Executor执行所有作业
3. 然后设置作业粒度分配，则当前节点会成为作业分配的决策者，即分配节点
4. 分配节点获取各个Executor的资源状态信息，然后根据策略选择Executor分配作业
5. 被分配到的作业Exeuctor即成为执行节点，执行作业，然后更新数据库

## **Azkaban概念**

Azkaban的任务执行主要元素为：project、Job和flow，其相关的类图如下所示：



整个过程的执行如下图所示：



## **WebServer提交作业**

1）Web UI通过ExecutorServlet提交作业

通过Web UI发起作业的执行，其通过Webapp#ExecutorServlet类发起作业流的执行：

*private void ajaxExecuteFlow(final HttpServletRequest req,*

*final HttpServletResponse resp, final HashMap<String, Object> ret, final User user)*

*throws ServletException {*

*......*

*final ExecutableFlow exflow = FlowUtils.createExecutableFlow(project, flow);*

*exflow.setSubmitUser(user.getUserId());*

*......*

*final String message =*

*this.executorManager.submitExecutableFlow(exflow, user.getUserId());*

*}*

构造ExecutableFlow对象，设定执行参数、并发及失败策略

1. ExecutorManager提交任务流，WebServer端下发作业流的实现逻辑

* 从exflow实例获取作业流的flowId
* 判断queuedFlows是否满，未满则继续执行
* 获取作业流中所有正在运行的实例id， running:List<Integer>
* 获取执行options，从中获取流的执行参数，判断是否running
* 调用executorLoader.uploadExecutableFlow，将其写入数据库表
* 判断是否为多执行节点模式，如果不是则将执行流标记为active，写数据库表

active\_executing\_flows。将流dispatch到唯一执行节点执行

* 如果是多执行节点模式，则将该执行流的状态标记为active，并放入执行队列queuedFlows中
* 多执行节点模式下，ExecutorManager类在构造函数中调用setupMultiExecutorMode方法。通过CandinateSelector#getBest(CandiateList<Executor>)。该方法会创建线程通过processQueuedFlows方法去消费队列中的首个作业流。其执行如下：

*this.executorLoader.assignExecutor(choosenExecutor.getId(),*

*exflow.getExecutionId());*

* 在ExecutingManagerUpdaterThread的执行过程中，获取排队的ExecutableFlow，并提交到Exeuctor中执行：

*results =ExecutorManager.this.apiGateway.callWithExecutionId(*

*executor.getHost(),executor.getPort(), ConnectorParams.UPDATE\_ACTION,*

*null, null, executionIds, updateTimes);*

## **Executor作业执行**

WebServer通过REST将执行流发送到Executor端，此时数据中的状态为preparing，execapp#ExecutorServlet类根据action请求参数，去调用handleAjaxExecute，如下所示：

*private void handleAjaxExecute(final HttpServletRequest req,*

*final Map<String, Object> respMap, final int execId) {*

*try {*

*this.flowRunnerManager.submitFlow(execId);*

*} ......*

*}*

调用FlowRunnerManager来submitFlow(execId)，其执行流程如下：

* 先判断runningFlows是否包含该execId对应的实例
* 调用ExecutorLoader#fetchExecutableFlow，从数据库中获取ExecutableFlow
* 调用FlowPrepare#setup，进行初始化，包括创建项目、执行目录等
* 获取执行所需要的ExecutionOptions，包括pipelineExecId，同时构建FlowWatcher来监控flow中各个Job的执行状态
* 构造FlowRunner，配置runner、判断runningFlows是否包含该execId的执行实例，并将其添加到runningFlow的map
* 提交到TrackingThreadPool（工作线程池），并加入到submmittedFlows
* 调用FlowRunner#run，执行流入如下：
* 创建Flow内部的Job线程池flow
* setFlowExecution/updateFlowReference/updateFlow更新flow的状态信息
* loadAllProperties，载入job参数和共享的参数
* 判断输入参数是否包含job.dispatch，如果包含且为true，新的线程updaterThread，跟踪该作业流下各个作业的执行状态
* 执行runFlow，根据DAG图算法一次执行Job，从流的开始节点开始递归调用runReadyJob来执行作业，然后updateFlow。如果流没有结束，根据重试设置，决定是否重跑失败的作业

核心方法为runReadyJob，其初始执行：

*for (final String startNodeId : ((ExecutableFlowBase) node).getStartNodes()) {*

*final ExecutableNode startNode = flow.getExecutableNode(startNodeId);*

*runReadyJob(startNode);*

*}*

其内部调用runExecutableNode(Node: ExecutableNode)，通过JobRunner来执行Job.run，以ProgressJob为例：

*public void run() throws Exception {*

*......*

*List<String> commands = = getCommandList();*

*for (String command : commands) {*

*AzkabanProcessBuilder builder = null;*

*if (isExecuteAsUser) {*

*command =*

*String.format("%s %s %s", executeAsUserBinaryPath, effectiveUser, command);*

*builder =*

*new AzkabanProcessBuilder(partitionCommandLine(command))*

*.setEnv(envVars).setWorkingDir(getCwd()).setLogger(getLog())*

*.enableExecuteAsUser().setExecuteAsUserBinaryPath(executeAsUserBinaryPath)*

*.setEffectiveUser(effectiveUser);*

*...*

*this.process = builder.build();*

*}*

*....*

*this.process.run();*

*this.success = true;*

*....*

*}*

# 参考文献

1）Executor选择： https://zhuanlan.zhihu.com/p/25437536

2）Azkaban执行流程： https://www.cnblogs.com/znicy/p/5742711.html

3）ExecutorManager源码分析： https://www.jianshu.com/p/d4761729f8d1

4）Azkaban安装：

<https://yq.aliyun.com/articles/25689>

https://blog.csdn.net/guaishushu1ss/article/details/77507916

5）Github: https://github.com/azkaban/azkaban

6）官网：https://azkaban.github.io/azkaban/docs/latest/

7）安装问题：Failed to change current working directory ownership

https://blog.csdn.net/lemonZhaoTao/article/details/79427629