**Azkaban**

# Azkaban概论

## 为什么需要工作流调度系统

1）一个完整的数据分析系统通常都是由大量任务单元组成：

shell脚本程序，java程序，mapreduce程序、hive脚本等

2）各任务单元之间存在时间先后及前后依赖关系

3）为了很好地组织起这样的复杂执行计划，需要一个工作流调度系统来调度执行；

例如，我们可能有这样一个需求，某个业务系统每天产生20G原始数据，我们每天都要对其进行处理，处理步骤如下所示：

（1）通过Hadoop先将原始数据同步到HDFS上；

（2）借助MapReduce计算框架对原始数据进行计算，生成的数据以分区表的形式存储到多张Hive表中；

（3）需要对Hive中多个表的数据进行JOIN处理，得到一个明细数据Hive大表；

（4）将明细数据进行复杂的统计分析，得到结果报表信息；

（5）需要将统计分析得到的结果数据同步到业务系统中，供业务调用使用。



## 常见工作流调度系统

1）简单的任务调度：直接使用linux的crontab来定义；

2）复杂的任务调度：开发调度平台或使用现成的开源调度系统，比如ooize、azkaban、 Cascading、Hamake等

## 各种调度工具特性对比

下面的表格对上述四种hadoop工作流调度器的关键特性进行了比较，尽管这些工作流调度器能够解决的需求场景基本一致，但在设计理念，目标用户，应用场景等方面还是存在显著的区别，在做技术选型的时候，可以提供参考

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | Hamake | Oozie | Azkaban | Cascading |
| 工作流描述语言 | XML | XML (xPDL based) | text file with key/value pairs | Java API |
| 依赖机制 | data-driven | explicit | explicit | explicit |
| 是否要web容器 | No | Yes | Yes | No |
| 进度跟踪 | console/log messages | web page | web page | Java API |
| Hadoop job调度支持 | no | yes | yes | yes |
| 运行模式 | command line utility | daemon | daemon | API |
| Pig支持 | yes | yes | yes | yes |
| 事件通知 | no | no | no | yes |
| 需要安装 | no | yes | yes | no |
| 支持的hadoop版本 | 0.18+ | 0.20+ | currently unknown | 0.18+ |
| 重试支持 | no | workflownode evel | yes | yes |
| 运行任意命令 | yes | yes | yes | yes |
| Amazon EMR支持 | yes | no | currently unknown | yes |

## Azkaban与Oozie对比

对市面上最流行的两种调度器，给出以下详细对比，以供技术选型参考。总体来说，ooize相比azkaban是一个重量级的任务调度系统，功能全面，但配置使用也更复杂。如果可以不在意某些功能的缺失，轻量级调度器azkaban是很不错的候选对象。

详情如下：

1）功能

两者均可以调度mapreduce，pig，java，脚本工作流任务

两者均可以定时执行工作流任务

2）工作流定义

Azkaban使用Properties文件定义工作流

Oozie使用XML文件定义工作流

3）工作流传参

Azkaban支持直接传参，例如${input}

Oozie支持参数和EL表达式，例如${fs:dirSize(myInputDir)}

4）定时执行

Azkaban的定时执行任务是基于时间的

Oozie的定时执行任务基于时间和输入数据

5）资源管理

Azkaban有较严格的权限控制，如用户对工作流进行读/写/执行等操作

Oozie暂无严格的权限控制

6）工作流执行

Azkaban有两种运行模式，分别是solo server mode(executor server和web server部署在同一台节点)和multi server mode(executor server和web server可以部署在不同节点)

Oozie作为工作流服务器运行，支持多用户和多工作流

7）工作流管理

Azkaban支持浏览器以及ajax方式操作工作流

Oozie支持命令行、HTTP REST、Java API、浏览器操作工作流

## Azkaban特点

Azkaban是由Linkedin开源的一个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定义了一种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用户界面维护和跟踪你的工作流。

它有如下功能特点：

1）Web用户界面

2）方便上传工作流

3）方便设置任务之间的关系

4）调度工作流

5）认证/授权(权限的工作)

6）能够杀死并重新启动工作流

7）模块化和可插拔的插件机制

8）项目工作区

9）工作流和任务的日志记录和审计

# Azkaban入门

## 单机模式

单机模式一般用于测试，不会用于生产环境。

将azkaban-solo-server-3.84.4.tar.gz拷贝到集群，并解压

[atguigu@hadoop102 software]$ tar /opt/software/azkaban-solo-server-3.84.4.tar.gz -C /opt/module

启动Azkaban

[atguigu@hadoop102 software]$ /opt/module azkaban-solo-server-3.84.4/bin/start-solo.sh

查看浏览器http://hadoop102:8081

关闭Azkaban

[atguigu@hadoop102 software]$ /opt/module azkaban-solo-server-3.84.4/bin/shutdown-solo.sh

## 集群模式

### 上传tar包

将azkaban-db-3.84.4.tar.gz，azkaban-exec-server-3.84.4.tar.gz，azkaban-web-server-3.84.4.tar.gz上传到集群

[atguigu@hadoop102 software]$ ll

总用量 35572

-rw-r--r--. 1 atguigu atguigu 6433 4月 18 17:24 azkaban-db-3.84.4.tar.gz

-rw-r--r--. 1 atguigu atguigu 16175002 4月 18 17:26 azkaban-exec-server-3.84.4.tar.gz

-rw-r--r--. 1 atguigu atguigu 20239974 4月 18 17:26 azkaban-web-server-3.84.4.tar.gz

新建/opt/module/azkaban目录，并将所有tar包解压到这个目录下

[atguigu@hadoop102 software]$ mkdir /opt/module/azkaban

[atguigu@hadoop102 software]$ ls /opt/software/\*.tar.gz | xargs -n1 tar zxC /opt/module/azkaban -f

### 设置MySQL

正常安装MySQL

登陆MySQL，创建Azkaban数据库

mysql> CREATE DATABASE azkaban;

创建Azkaban用户并赋予权限

mysql> set global validate\_password\_length=4;

mysql> set global validate\_password\_policy=0;

mysql> CREATE USER 'azkaban'@'%' IDENTIFIED BY '000000';

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

创建Azkaban表，完成后退出MySQL

mysql> use azkaban;

mysql> source /opt/module/azkaban/azkaban-db-3.84.4/create-all-sql-3.84.4.sql

mysql> quit;

更改MySQL包大小

[atguigu@hadoop102 software]$ sudo vim /etc/my.cnf

在[mysqld]下面加一行

[mysqld]

...

max\_allowed\_packet=1024M

重启MySQL

[atguigu@hadoop102 software]$ sudo systemctl restart mysqld

### 设置Executor Server

Azkaban Executor Server处理工作流和作业的实际执行。

编辑azkaban.properties

[atguigu@hadoop102 azkaban]$ vim /opt/module/azkaban/azkaban-exec-server-3.84.4/conf/azkaban.properties

修改如下属性

...

default.timezone.id=Asia/Shanghai

...

azkaban.webserver.url=http://hadoop102:8081

...

database.type=mysql

mysql.port=3306

mysql.host=hadoop102

mysql.database=azkaban

mysql.user=azkaban

mysql.password=000000

mysql.numconnections=100

在最后添加

executor.metric.reports=true

executor.metric.milisecinterval.default=60000

同步executor server到所有节点

[atguigu@hadoop102 azkaban]$ xsync /opt/module/azkaban/azkaban-exec-server-3.84.4

在三台机器上启动并激活executor server

[atguigu@hadoop102 lib]$ cd /opt/module/azkaban/azkaban-exec-server-3.84.4

[atguigu@hadoop102 azkaban-exec-server-3.84.4]$ bin/start-exec.sh

如果在目录下出现executor.port文件，说明启动成功，下面激活executor

[atguigu@hadoop102 azkaban-exec-server-3.84.4]$ curl -G "localhost:$(<./executor.port)/executor?action=activate" && echo

如果三台机器都出现如下提示，则表示激活成功

{"status":"success"}

### 设置Web Server

Azkaban Web Server处理项目管理，身份验证，计划和执行触发。

编辑azkaban.properties

[atguigu@hadoop102 azkaban]$ vim /opt/module/azkaban/azkaban-web-server-3.84.4/conf/azkaban.properties

修改如下属性

...

default.timezone.id=Asia/Shanghai

...

database.type=mysql

mysql.port=3306

mysql.host=hadoop102

mysql.database=azkaban

mysql.user=azkaban

mysql.password=000000

mysql.numconnections=100

...

azkaban.executorselector.filters=StaticRemainingFlowSize,CpuStatus

修改azkaban-users.xml文件，添加atguigu用户

[atguigu@hadoop102 azkaban-web-server-3.84.4]$ vim /opt/module/azkaban/azkaban-web-server-3.84.4/conf/azkaban-users.xml

<azkaban-users>

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

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

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

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

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

</azkaban-users>

在102启动web server

[atguigu@hadoop102 lib]$ cd /opt/module/azkaban/azkaban-web-server-3.84.4

[atguigu@hadoop102 azkaban-exec-server-3.84.4]$ bin/start-web.sh

访问<http://hadoop102:8081>,并用atguigu用户登陆

## Work Flow

### 入门

新建first.project文件，内容如下

azkaban-flow-version: 2.0

新建basic.flow文件，内容如下

nodes:

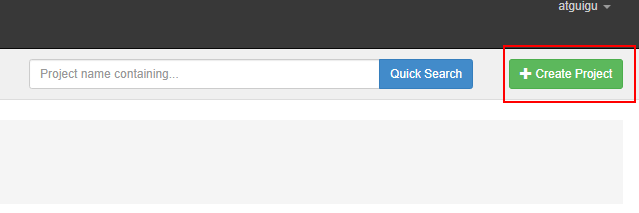
- name: jobA

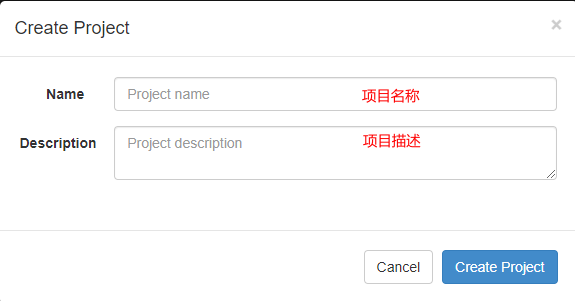
type: command

config:

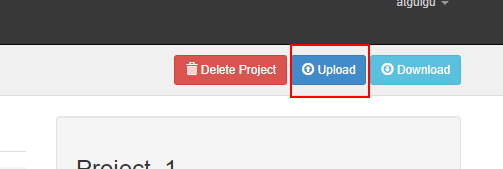
command: echo "This is an echoed text."

在WebServer新建项目：

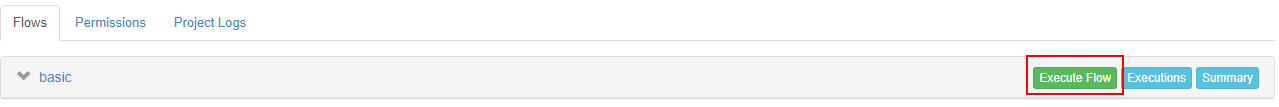




将两个文件打包为zip文件并上传并执行



上传完毕点击执行



### 作业设置

Azkaban作业以Key:Value形式定义，每个作业有如下配置：

* + - * 1. name：作业名称
        2. type：作业类型（详细类型配置见第3章）
        3. config：和作业类型相关的配置，也以KV值形式
        4. dependsOn：作业依赖

### 作业依赖案例

修改basic.flow为如下内容

nodes:

- name: jobC

type: command

# jobC 依赖 JobA和JobB

dependsOn:

- jobA

- jobB

config:

command: echo "I’m JobC"

- name: jobA

type: command

config:

command: echo "I’m JobA"

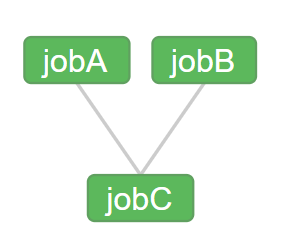
- name: jobB

type: command

config:

command: echo "I’m JobB"

打包上传执行，注意查看工作流



### 工作流配置

可以在工作流配置文件中添加config标签，对工作流进行全局配置，这些配置会应用到该工作流的所有作业中。例如，在basic.flow前面添加config：

---

config:

words.to.print: "This is for test!"

nodes:

- name: jobA

type: command

config:

command: echo ${words.to.print}

### 失败重试

在Job配置中添加两个参数，即可实现任务失败自动重试：

#任务重试次数

retries: 3

#任务重试延迟

retry.backoff: 10000

案例：

nodes:

- name: JobA

type: command

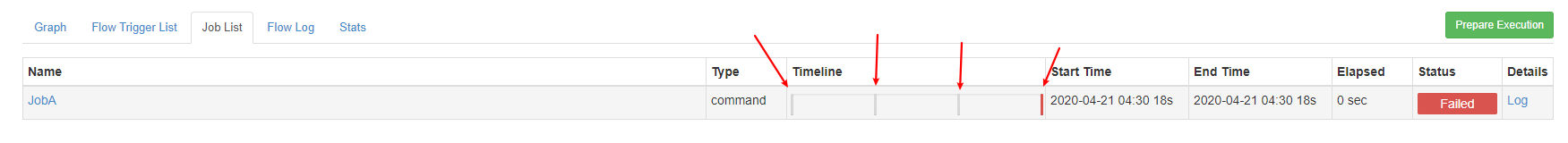
config:

command: sh /not\_exists.sh

retries: 3

retry.backoff: 10000

执行并观察：



任务在失败前重试了三次

也可以在任务日志中看到：



总共执行了4次。

也可以在Flow全局配置中添加任务失败重试配置，此时重试配置会应用到所有Job。

案例如下：

config:

retries: 3

retry.backoff: 10000

nodes:

- name: JobA

type: command

config:

command: sh /not\_exists.sh

### 内嵌工作流

工作流定义文件中可以添加子工作流，例如：

nodes:

- name: jobC

type: command

# jobC 依赖embedded\_flow

dependsOn:

- embedded\_flow

config:

command: echo "I’m JobC"

- name: embedded\_flow

type: flow

config:

prop: value

nodes:

- name: jobB

type: noop

dependsOn:

- jobA

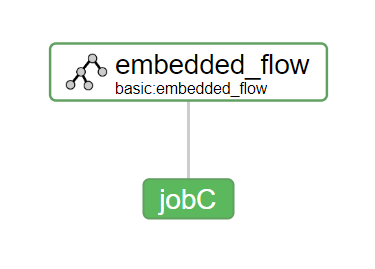
- name: jobA

type: command

config:

command: pwd

上传并执行，注意观察工作流图形



# Azkaban进阶

## 作业类型

### 命令行类型

命令行类型是最基本的内建类型，type类型为command，可用的配置为：

command：你要执行的命令

案例：

nodes:

- name: jobA

type: command

config:

command: echo "Command 1"

### JavaProcess

JavaProcess类型可以运行一个自定义主类方法，type类型为javaprocess，可用的配置为：

Xms：最小堆

Xmx：最大堆

java.class：要运行的Java对象，其中必须包含Main方法

案例：

* + - * 1. 首先新建一个Java项目，新建一个包含主方法的主类，内容如下：

package com.atguigu;

public class AzTest {

public static void main(String[] args) {

System.out.println("This is for testing!");

}

}

打包成jar包

新建testJava.flow，内容如下

nodes:

- name: test\_java

type: javaprocess

config:

Xms: 96M

Xmx: 200M

java.class: com.atguigu.AzTest

将Jar包、flow文件和project文件打包成zip，上传到集群并执行

## 条件工作流

条件工作流功能允许用户根据条件指定是否运行某些作业。条件由先前作业的运行时参数（例如输出）和预定义宏组成。在这些条件下，用户可以在确定作业执行逻辑时获得更大的灵活性。例如，只要父作业之一成功，他们就可以运行当前作业。他们可以在工作流内部实现分支逻辑。

### 运行时参数

运行时参数一般指作业的输出，使用时有以下几个条件：

使用 ${jobName:param} 来定义作业运行时参数的条件

“:” 用于分隔jobName和参数

job运行时，使用参数与条件中的字符串或数字进行比较

用户需要事先将参数的值写入 $JOB\_OUTPUT\_PROP\_FILE

支持的运算符：

* + - * 1. == 等于

!= 不等于

> 大于

>= 大于等于

< 小于

<= 小于等于

&& 与

|| 或

! 非

案例：

* + - * 1. 新建basic.flow

nodes:

- name: JobA

type: command

config:

command: sh /opt/module/write\_to\_props.sh

- name: JobB

type: command

dependsOn:

- JobA

config:

command: echo "This is JobB."

condition: ${JobA:param1} == "AAA"

- name: JobC

type: command

dependsOn:

- JobA

config:

command: echo "This is JobC."

condition: ${JobA:param1} == "BBB"

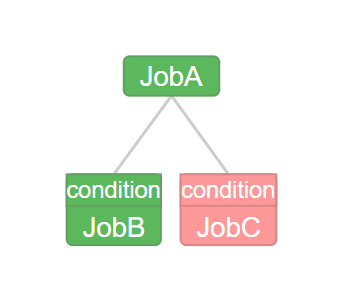
新建/opt/module/write\_to\_props.sh

vim /opt/module/write\_to\_props.sh

内容为：

echo '{"param1":"AAA"}' > $JOB\_OUTPUT\_PROP\_FILE

按照我们设定的条件，由于JobA输出为AAA，所以会执行JobB分支。上传执行，注意观察分支条件：



### 预定义宏

预定义宏将会在所有父作业上评估，即YAML文件中的dependsOn部分。可用的预定义宏如下：

all\_success: 全部成功(默认)

all\_done：全部完成

all\_failed：全部失败

one\_success：至少一个成功

one\_failed：至少一个失败

案例：

* + - * 1. 修改上个案例的basic.flow:

nodes:

- name: JobA

type: command

config:

command: sh /opt/module/write\_to\_props.sh

- name: JobB

type: command

dependsOn:

- JobA

config:

command: echo "This is JobB."

condition: ${JobA:param1} == "AAA"

- name: JobC

type: command

dependsOn:

- JobA

config:

command: echo "This is JobC."

condition: ${JobA:param1} == "BBB"

- name: JobD

type: command

dependsOn:

- JobB

- JobC

config:

command: echo "This is JobD."

condition: one\_success

- name: JobE

type: command

dependsOn:

- JobB

- JobC

config:

command: echo "This is JobE."

condition: all\_success

- name: JobF

type: command

dependsOn:

- JobB

- JobC

- JobD

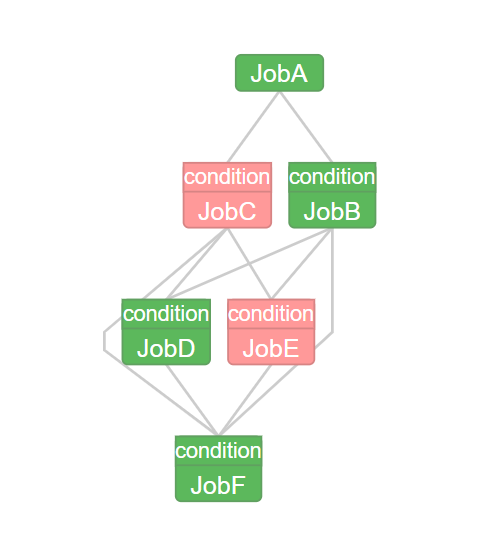
- JobE

config:

command: echo "This is JobF."

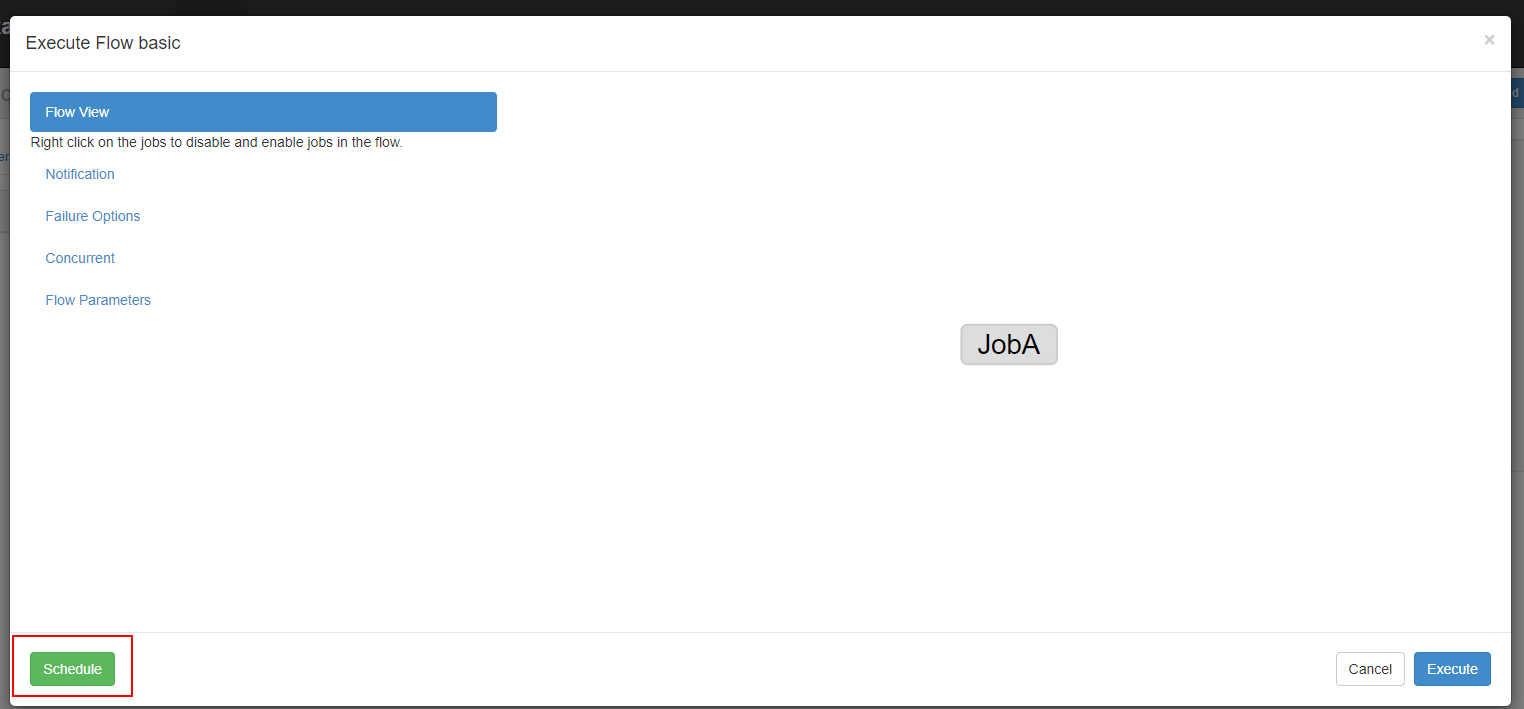
condition: all\_done

打包执行，注意观察任务的执行情况

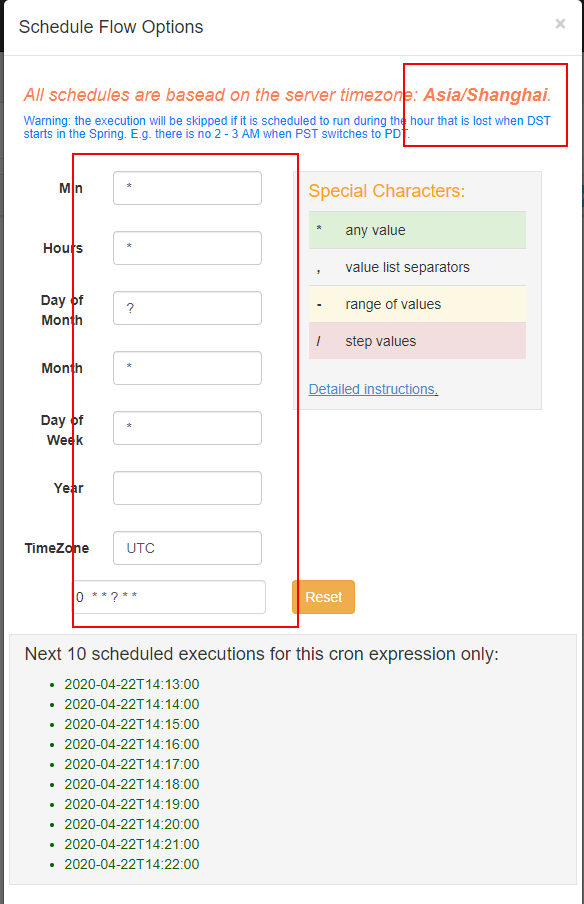


## 定时执行

Azkaban可以定时执行工作流。在执行工作流时候，选择左下角Schedule



右上角注意时区是上海，然后在左面填写具体执行事件，即可定时执行任务



## 失败报警

### 默认邮件报警

Azkaban默认支持通过邮件对失败的任务进行报警，配置方法如下：

在web-server节点（102）编辑/opt/module/azkaban/azkaban-web-server-3.84.4/conf/azkaban.properties，修改如下内容：

#这里设置邮件发送服务器，需要 申请邮箱，切开通stmp服务，以下只是例子

mail.sender=atguigu@126.com

mail.host=smtp.126.com

mail.user=atguigu@126.com

mail.password=password

#这里设置工作流成功或者失败默认向哪里发送服务

job.failure.email=atguigu@126.com

[job.success.email=atguigu@126.com](mailto:job.success.email=atguigu@126.com)

保存并重启web-server。

编辑basic.flow，加入如下属性：

nodes:

- name: jobA

type: command

config:

command: echo "This is an echoed text."

failure.emails: atguigu@126.com

success.emails: atguigu@126.com

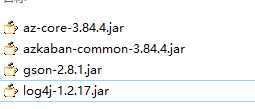
notify.emails: [atguigu@126.com](mailto:atguigu@126.com%20)

### 自定义报警

有时任务执行失败后邮件报警接收不及时，需要自定义报警装置，比如电话报警。此时需要首先找一个电话通知服务商，比如onealter.com，购买相应服务后，获取通知API。然后进行MailAlter二次开发。

新建一个普通的Java项目

在项目的lib里添加4个Jar包：



新建com.atguigu.PhoneAlter类实现azkaban.alert.Alerter

package com.atguigu;

import azkaban.alert.Alerter;

import azkaban.executor.ExecutableFlow;

import azkaban.executor.Executor;

import azkaban.executor.ExecutorManagerException;

import azkaban.sla.SlaOption;

import azkaban.utils.Props;

import com.google.gson.JsonObject;

import org.apache.log4j.Logger;

import java.util.List;

public class PhoneAlterter implements Alerter {

private static final Logger logger = Logger.getLogger(PhoneAlterter.class);

private String appKey;

private String url;

public PhoneAlterter(Props props) {

appKey = props.getString("my.alert.appKey", "");

url = props.getString("my.alert.url", "");

logger.info("Appkey: " + appKey);

logger.info("URL: " + url);

}

/\*\*

\* 成功的通知

\*

\* @param exflow

\* @throws Exception

\*/

@Override

public void alertOnSuccess(ExecutableFlow exflow) throws Exception {

}

/\*\*

\* 出现问题的通知

\*

\* @param exflow

\* @param extraReasons

\* @throws Exception

\*/

@Override

public void alertOnError(ExecutableFlow exflow, String... extraReasons) throws Exception {

//一般来说网络电话服务都是通过HTTP请求发送的，这里可以调用shell发送HTTP请求

JsonObject alert = new JsonObject();

alert.addProperty("app", appKey);

alert.addProperty("eventId", exflow.getId());

alert.addProperty("eventType", "trigger");

alert.addProperty("alarmContent", exflow.getId() + " fails!");

alert.addProperty("priority", "2");

String[] cmd = new String[8];

cmd[0] = "curl";

cmd[1] = "-H";

cmd[2] = "Content-type: application/json";

cmd[3] = "-X";

cmd[4] = "POST";

cmd[5] = "-d";

cmd[6] = alert.toString();

cmd[7] = url;

logger.info("Sending phone alert!");

Runtime.getRuntime().exec(cmd);

}

/\*\*

\* 首次出现问题的通知

\*

\* @param exflow

\* @throws Exception

\*/

@Override

public void alertOnFirstError(ExecutableFlow exflow) throws Exception {

}

@Override

public void alertOnSla(SlaOption slaOption, String slaMessage) throws Exception {

}

@Override

public void alertOnFailedUpdate(Executor executor, List<ExecutableFlow> executions, ExecutorManagerException e) {

}

}

新建/opt/module/azkaban/azkaban-web-server-3.84.4/plugin/alerter/phone-alerter文件夹，并在内部新建conf和lib两个目录

[atguigu@hadoop102 azkaban-web-server-3.84.4]$ mkdir -p /opt/module/azkaban/azkaban-web-server-3.84.4/plugins/alerter/phone-alerter/conf /opt/module/azkaban/azkaban-web-server-3.84.4/plugins/alerter/phone-alerter/lib

在新建的conf目录里，新建plugin.properties

#name一定要设置email，用以覆盖默认的邮件报警

alerter.name=email

alerter.external.classpaths=lib

alerter.class=com.atguigu.PhoneAlterter

#这两个参数和你使用的AlertAPI有关系

my.alert.appKey=cf3e2ce2-40ba-c3cd-1c74-xxxxxxxxxxxx

my.alert.url=http://some.example.url

将代码打包，并将Jar上传到/opt/module/azkaban/azkaban-web-server-3.84.4/lib文件夹，并重启web服务。

# 参考资料

## Azkaban完整配置

见官网文档：<https://azkaban.readthedocs.io/en/latest/configuration.html>

## YAML语法

Azkaban2.0工作流文件是用YAML语法写的，相关教程如下：

