Azkaban

第一章 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；

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、Airflow、DolphinScheduler等。

第二章 WorkFlow案例

1 HelloWorld案例

1）在windows环境，新建azkaban.project文件，编辑内容如下

```
azkaban-flow-version: 2.0
```

注意：该文件作用，是采用新的Flow-API方式解析flow文件。

2）新建basic.flow文件，内容如下

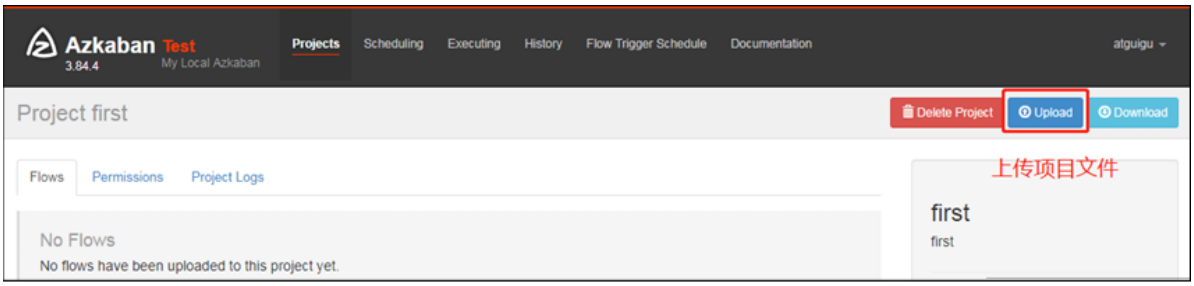
```
nodes:
  - name: jobA
    type: command
    config:
      command: echo "Hello World"
```

3）将`azkaban.project`、`basic.flow`文件压缩到一个`zip`文件，文件名称`必须是英文`。

4）在WebServer 新建项目： http://master:8081/index

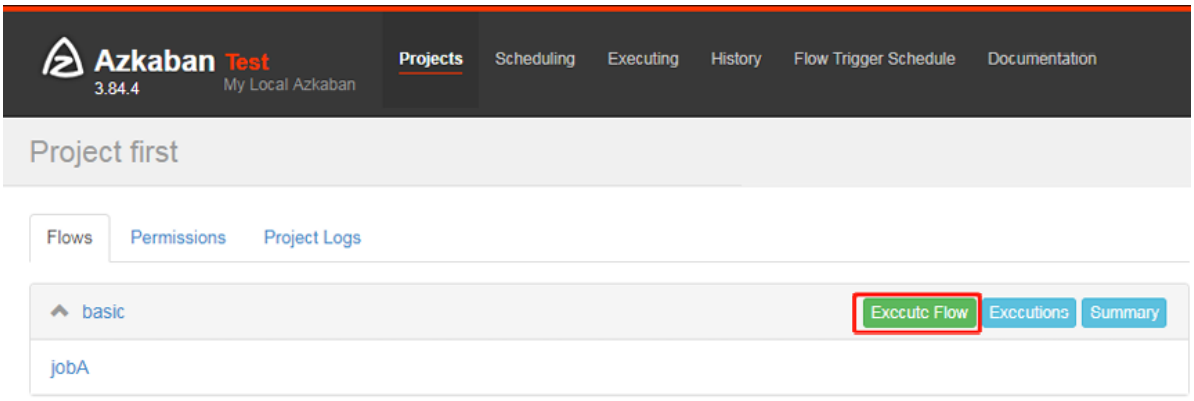
5）给项目名称命名和添加项目描述

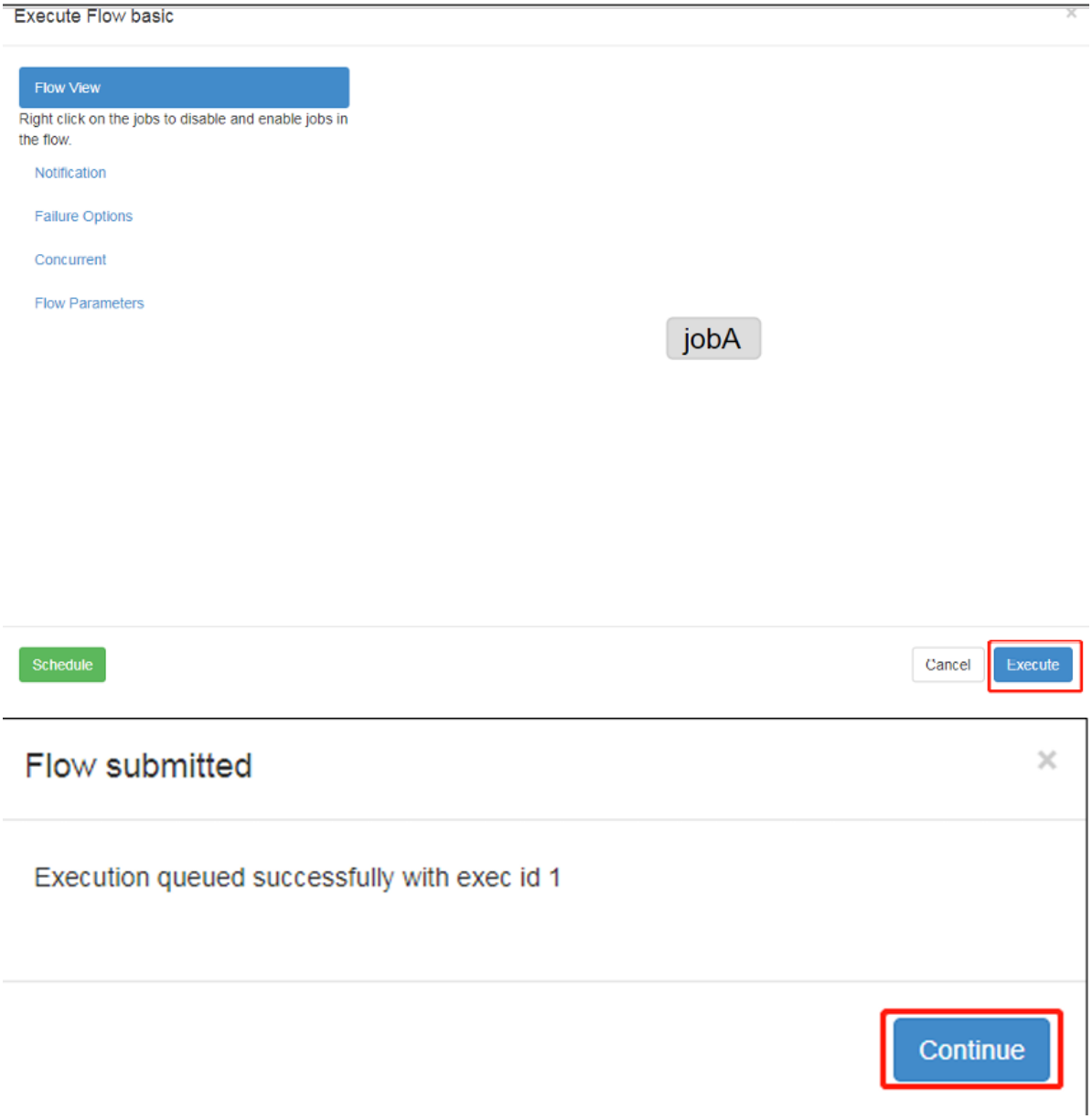
6）first.zip文件上传



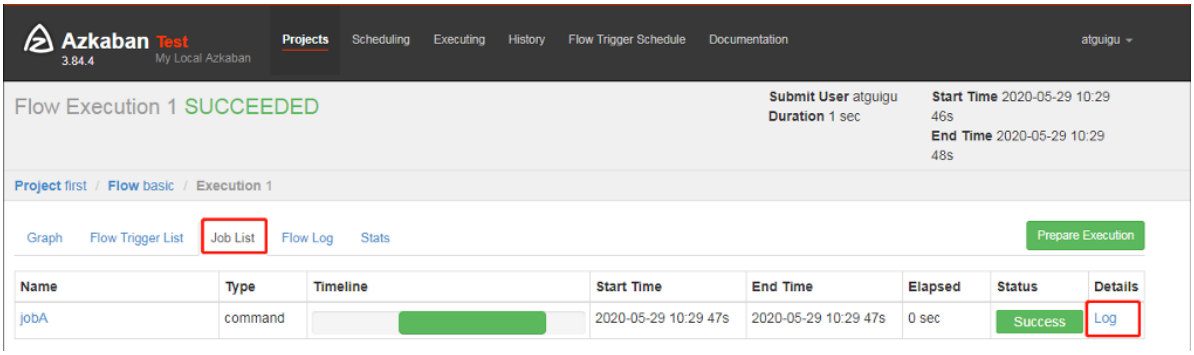
7）选择上传的文件

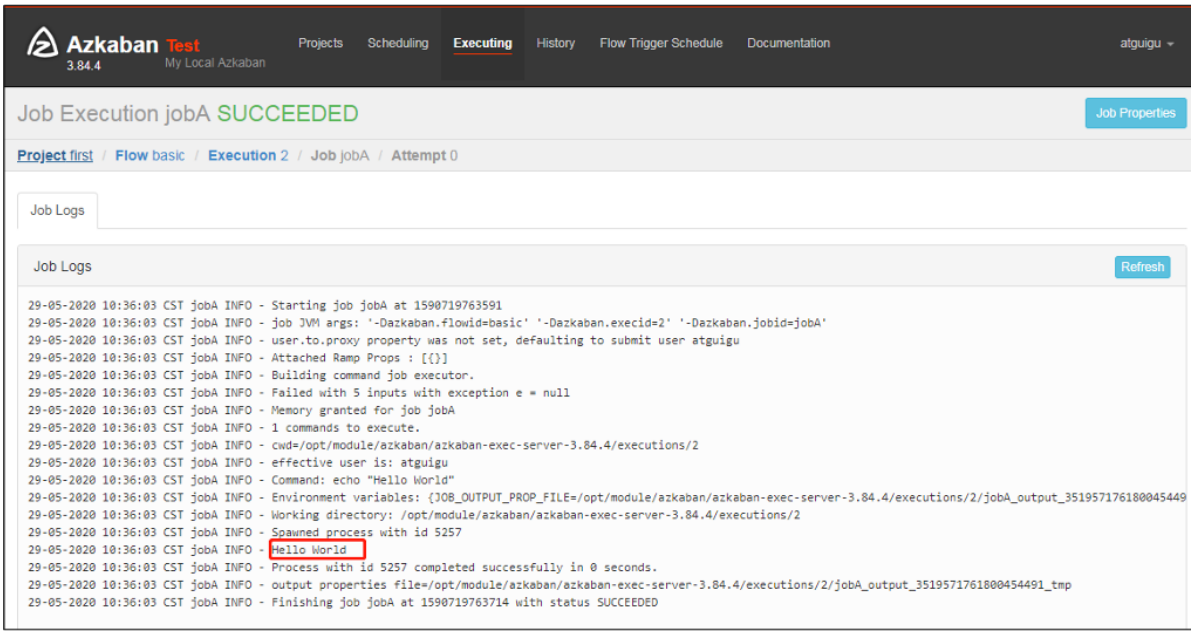
8）执行任务流





9）在日志中，查看运行结果





2 作业依赖案例

需求：JobA和JobB执行完了，才能执行JobC

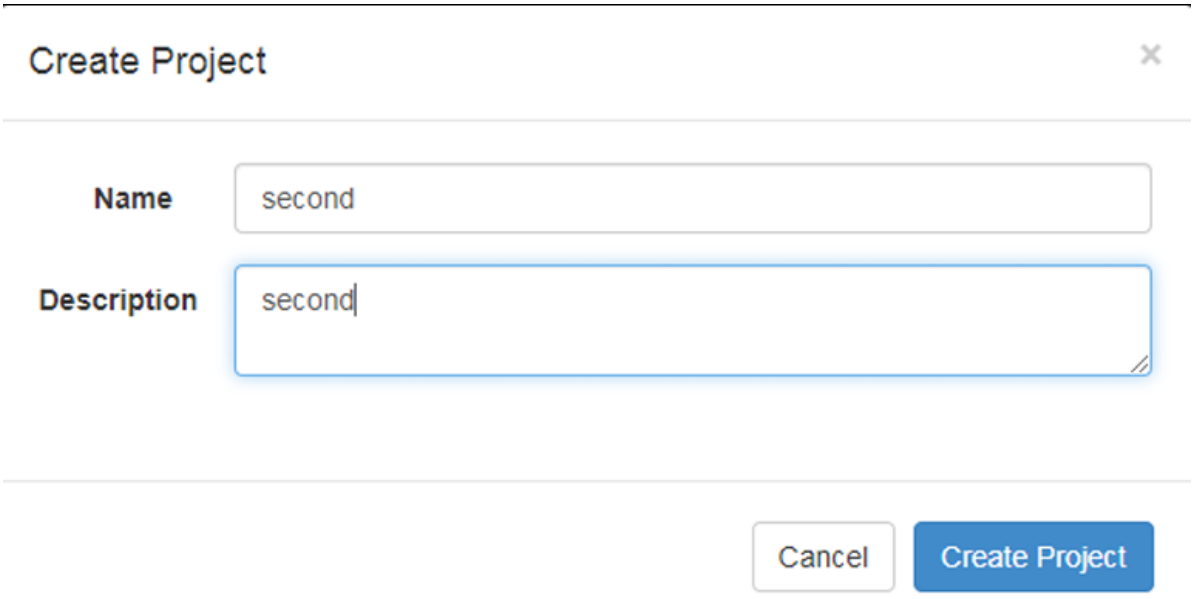
具体步骤：

1）修改basic.flow为如下内容

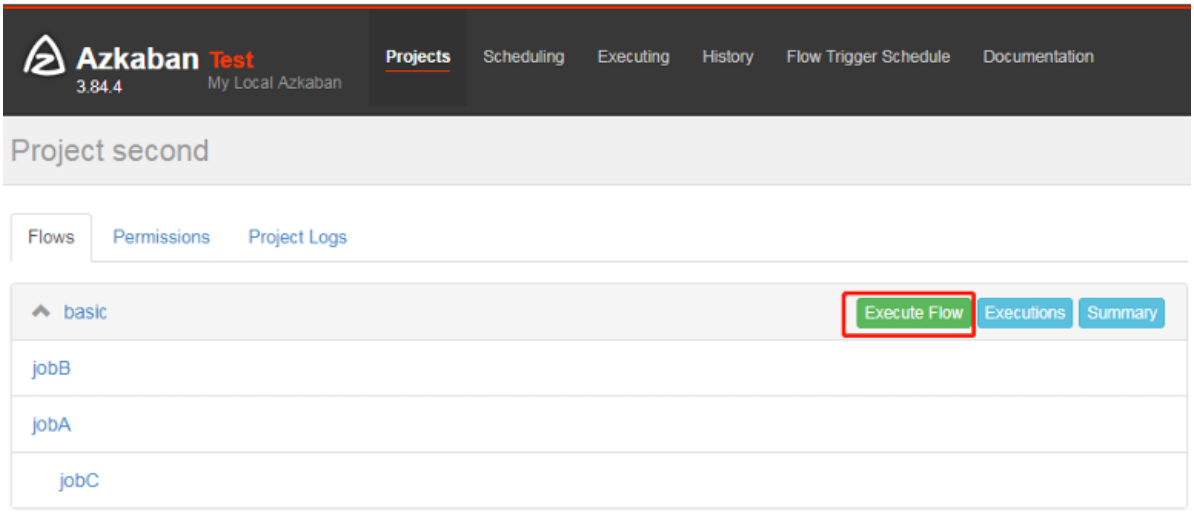
```
nodes:
  - name: jobC
    type: command
     # jobC 依赖 JobA和JobB
    dependsOn:
       - jobA
       - jobB
    config:
       command: echo "I' m JobC"

  - name: jobA
    type: command
    config:
       command: echo "I' m JobA"

  - name: jobB
    type: command
    config:
       command: echo "I' m JobB"
```

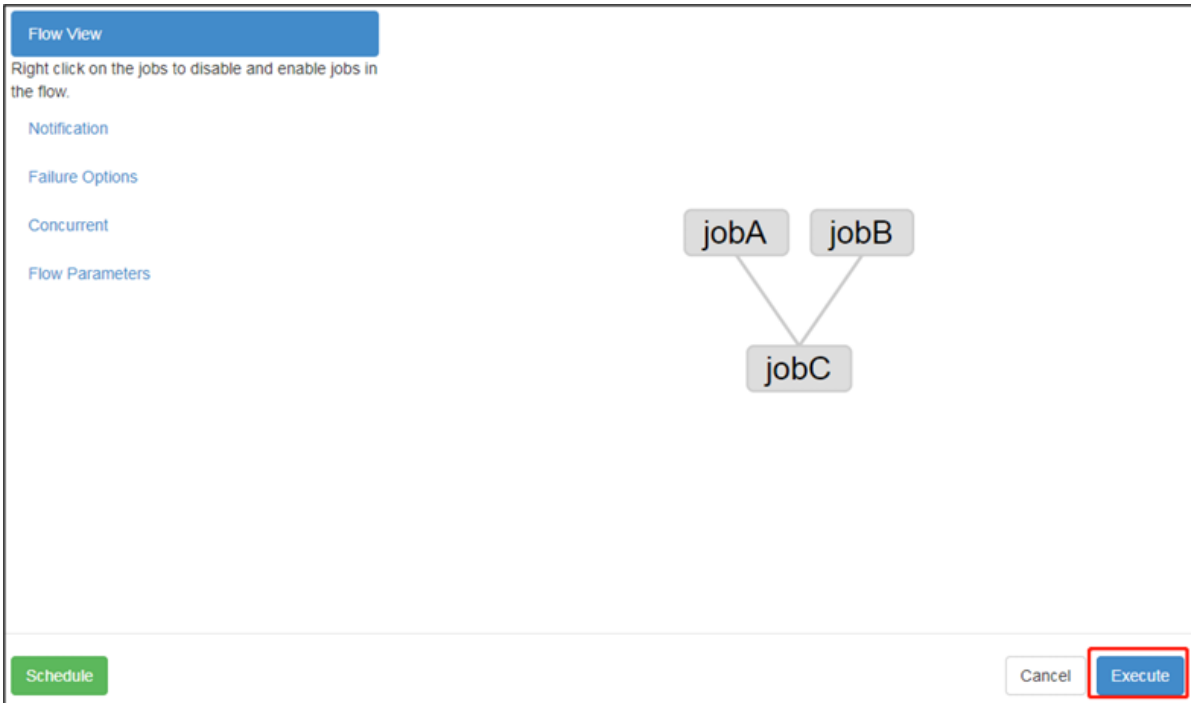
2）将修改后的basic.flow和azkaban.project压缩成second.zip文件

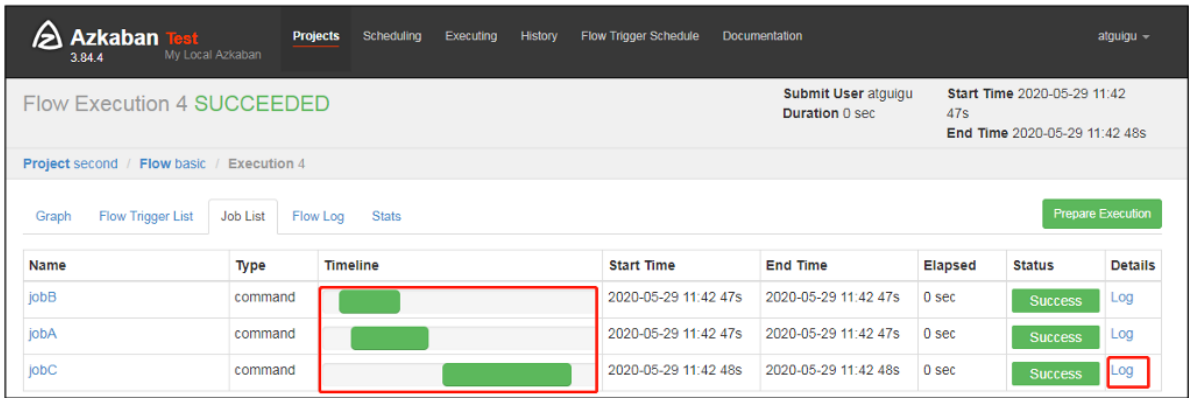
3）重复HelloWorld案例后续步骤。

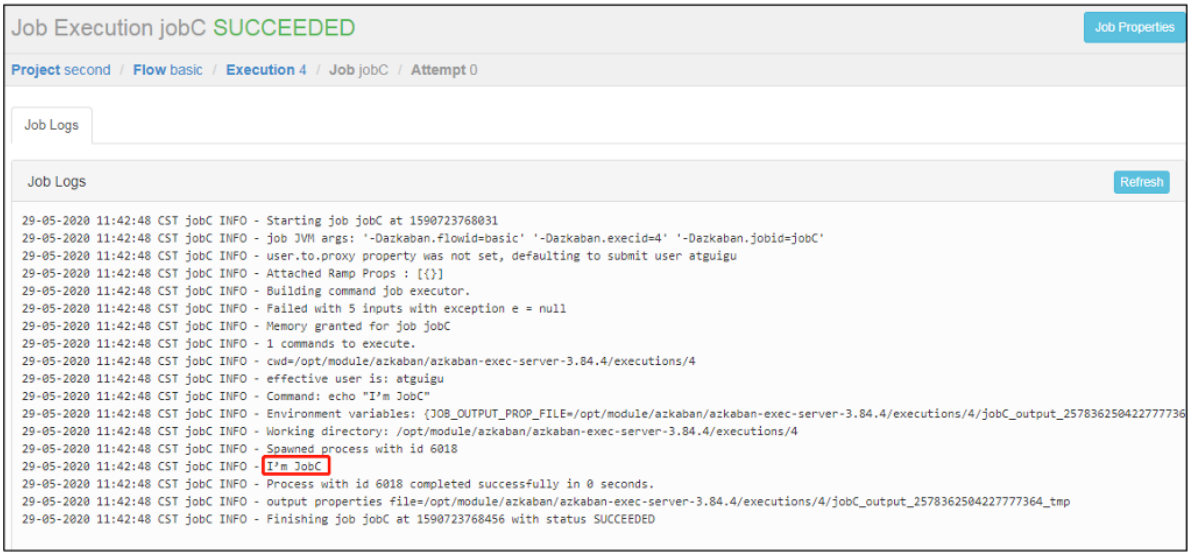

Create Project
Name
second
Description
second
Cancel
Create Project


Upload Project Files
Job Archive
选择文件
second.zip
Cancel
Upload


Azkaban Test
3.84.4
My Local Azkaban
Projects
Scheduling
Executing
History
Flow Trigger Schedule
Documentation
Project second
Flows
Permissions
Project Logs
basic
Execute Flow
Executions
Summary
jobB
jobA
jobC


Flow View
Right click on the jobs to disable and enable jobs in the flow.
Notification
Failure Options
Concurrent
Flow Parameters
jobA
jobB
jobC
Schedule
Cancel
Execute


Azkaban Test
3.84.4
My Local Azkaban
Projects
Scheduling
Executing
History
Flow Trigger Schedule
Documentation
atguigu
Flow Execution 4 SUCCEEDED
Submit User atguigu
Duration 0 sec
Start Time 2020-05-29 11:42 47s
End Time 2020-05-29 11:42 48s
Project second / Flow basic / Execution 4
Graph
Flow Trigger List
Job List
Flow Log
Stats
Prepare Execution
Name
Type
Timeline
Start Time
End Time
Elapsed
Status
Details
jobB command 2020-05-29 11:42 47s 2020-05-29 11:42 47s 0 sec Success Log
jobA command 2020-05-29 11:42 47s 2020-05-29 11:42 47s 0 sec Success Log
jobC command 2020-05-29 11:42 48s 2020-05-29 11:42 48s 0 sec Success Log


Job Execution jobC SUCCEEDED
Job Properties
Project second / Flow basic / Execution 4 / Job jobC / Attempt 0
Job Logs
Job Logs
Refresh
29-05-2020 11:42:48 CST jobC INFO - Starting job jobC at 1590723768031
29-05-2020 11:42:48 CST jobC INFO - job JVM args: '-Dazkaban.flowid=basic' '-Dazkaban.execid=4' '-Dazkaban.jobid=jobC'
29-05-2020 11:42:48 CST jobC INFO - user.to.proxy property was not set, defaulting to submit user atguigu
29-05-2020 11:42:48 CST jobC INFO - Attached Ramp Props : [{}]
29-05-2020 11:42:48 CST jobC INFO - Building command job executor.
29-05-2020 11:42:48 CST jobC INFO - Failed with 5 inputs with exception e = null
29-05-2020 11:42:48 CST jobC INFO - Memory granted for job jobC
29-05-2020 11:42:48 CST jobC INFO - 1 commands to execute.
29-05-2020 11:42:48 CST jobC INFO - cwd=/opt/module/azkaban/azkaban-exec-server-3.84.4/executions/4
29-05-2020 11:42:48 CST jobC INFO - effective user is: atguigu
29-05-2020 11:42:48 CST jobC INFO - Command: echo "I'm JobC"
29-05-2020 11:42:48 CST jobC INFO - Environment variables: {JOB_OUTPUT_PROP_FILE=/opt/module/azkaban/azkaban-exec-server-3.84.4/executions/4/jobC_output_257836250422777736
29-05-2020 11:42:48 CST jobC INFO - Working directory: /opt/module/azkaban/azkaban-exec-server-3.84.4/executions/4
29-05-2020 11:42:48 CST jobC INFO - Spawned process with id 6018
29-05-2020 11:42:48 CST jobC INFO - I'm JobC
29-05-2020 11:42:48 CST jobC INFO - Process with id 6018 completed successfully in 0 seconds.
29-05-2020 11:42:48 CST jobC INFO - output properties file=/opt/module/azkaban/azkaban-exec-server-3.84.4/executions/4/jobC_output_2578362504227777364_tmp
29-05-2020 11:42:48 CST jobC INFO - Finishing job jobC at 1590723768456 with status SUCCEEDED

3 自动失败重试案例

需求：如果执行任务失败，需要重试3次，重试的时间间隔10000ms

具体步骤：

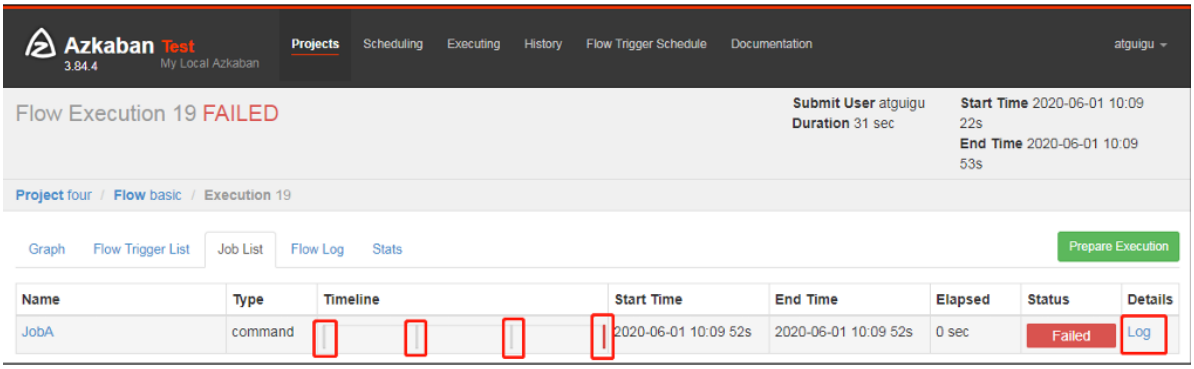
1）编译配置流

```
nodes:
  - name: JobA
    type: command
    config:
      command: sh /not_exists.sh
      retries: 3
      retry.backoff: 10000
```

2）将修改后的basic.flow和azkaban.project压缩成four.zip文件

3）重复HelloWorld案例后续步骤。

4）执行并观察到一次失败+三次重试



5）也可以点击上图中的Log，在任务日志中看到，总共执行了4次。

4 手动失败重试案例

需求：JobA -> JobB（依赖于A） -> JobC -> JobD -> JobE -> JobF。生产环境下任何Job都有可能挂掉，可以根据需求执行想要执行的Job。

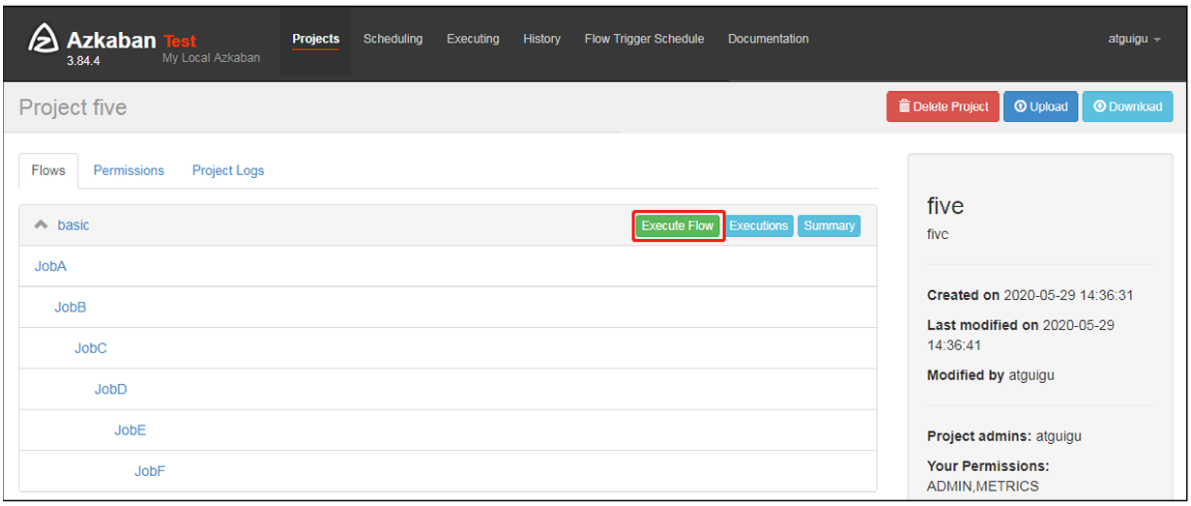
具体步骤：

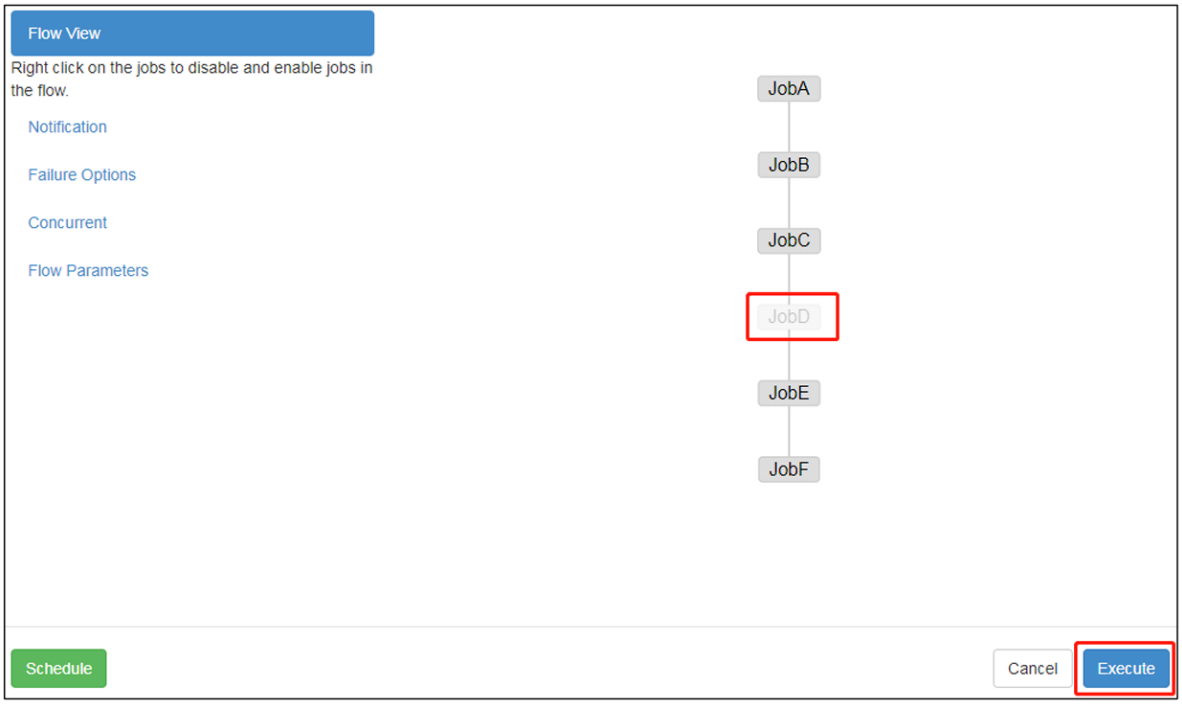
1）编译配置流

```
nodes:
  - name: JobA
    type: command
    config:
      command: echo "This is JobA."

  - name: JobB
    type: command
    dependsOn:
      - JobA
    config:
      command: echo "This is JobB."

  - name: JobC
    type: command
    dependsOn:
      - JobB
    config:
      command: echo "This is JobC."

  - name: JobD
    type: command
    dependsOn:
      - JobC
    config:
      command: echo "This is JobD."

  - name: JobE
    type: command
    dependsOn:
      - JobD
    config:
      command: echo "This is JobE."

  - name: JobF
    type: command
    dependsOn:
      - JobE
    config:
      command: echo "This is JobF."
```

2）将修改后的basic.flow和azkaban.project压缩成five.zip文件

3）重复HelloWorld案例后续步骤。





Enable和Disable下面都分别有如下参数：

Parents：该作业的上一个任务

Ancestors：该作业前的所有任务

Children：该作业后的一个任务

Descendents：该作业后的所有任务

Enable All：所有的任务

4）可以根据需求选择性执行对应的任务。

第三章 Azkaban进阶

JavaProcess作业类型案例

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

classpath：类路径

java.class：要运行的 Java 对象，其中必须包含Main方法

main.args：main 方法的参数

案例：

1）新建一个azkaban的maven工程

2）创建包名：com.atguigu

3）创建AzTest类

```
package com.atguigu;

public class AzTest {
    public static void main(String[] args) {
        System.out.println("This is for testing!");
    }
}
```

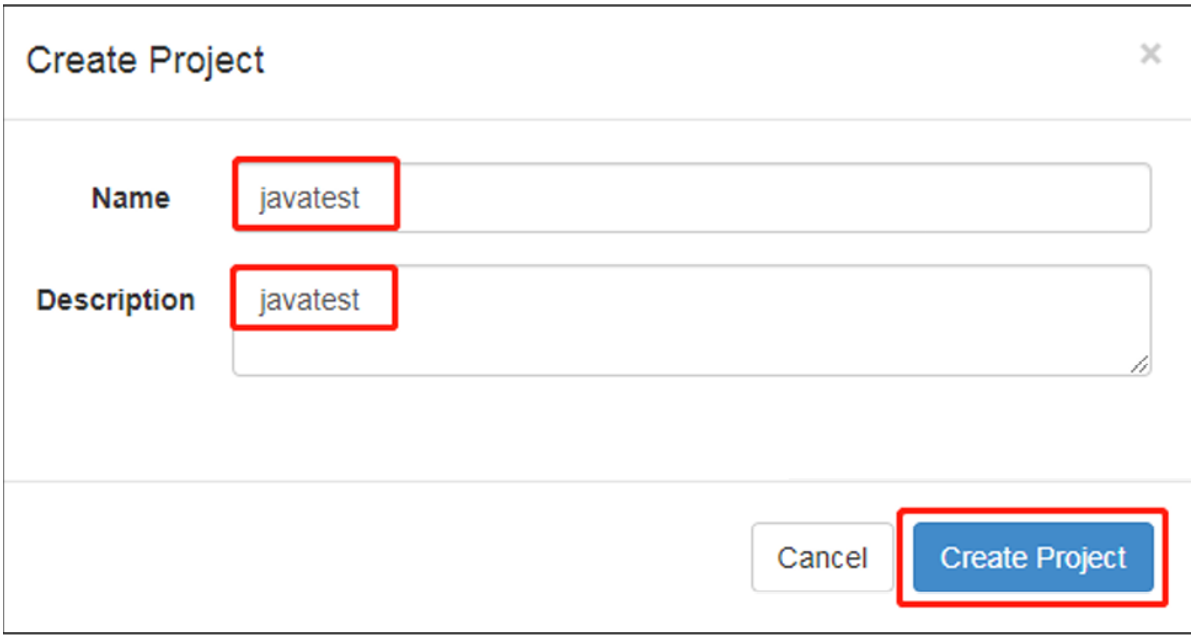
4）打包成jar包azkaban-1.0-SNAPSHOT.jar

5）新建testJava.flow，内容如下

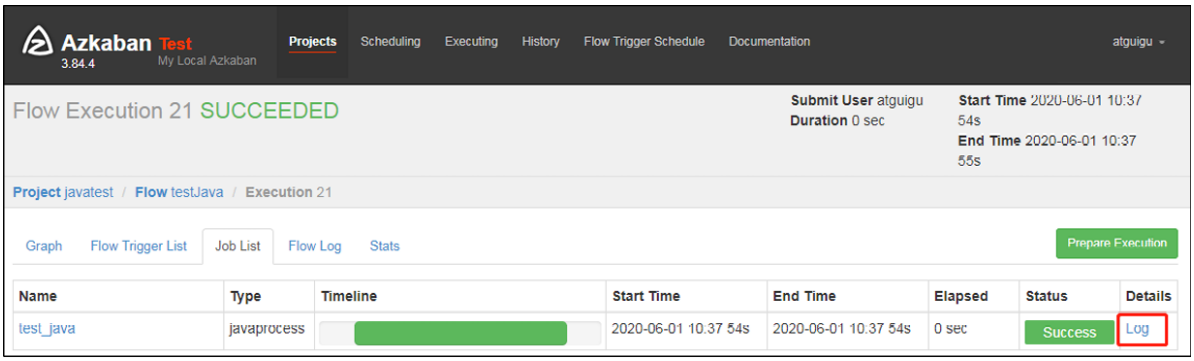
```
nodes:
  - name: test_java
    type: javaprocess
    config:
      Xms: 96M
      Xmx: 200M
      java.class: com.atguigu.AzTest
```

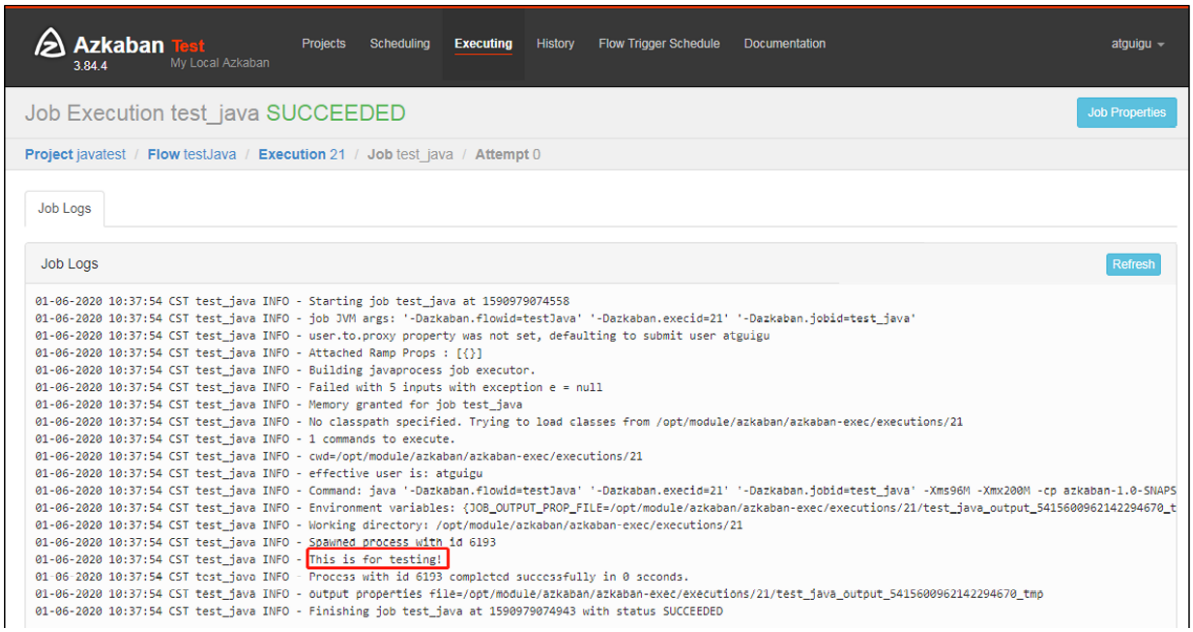
6）将Jar包、flow文件和project文件打包成javatest.zip

7）创建项目 -> 上传javatest.zip -> 执行作业 -> 观察结果


Create Project
Name
javatest
Description
javatest
Cancel
Create Project


Upload Project Files
Job Archive
选择文件
javatest.zip
Cancel
Upload


Azkaban Test
3.84.4
My Local Azkaban
Projects
Scheduling
Executing
History
Flow Trigger Schedule
Documentation
atguigu
Flow Execution 21 SUCCEEDED
Submit User atguigu
Duration 0 sec
Start Time 2020-06-01 10:37 54s
End Time 2020-06-01 10:37 55s
Project javatest / Flow testJava / Execution 21
Graph
Flow Trigger List
Job List
Flow Log
Stats
Prepare Execution
Name
Type
Timeline
Start Time
End Time
Elapsed
Status
Details
test_java
javaprocess
2020-06-01 10:37 54s
2020-06-01 10:37 54s
0 sec
Success
Log



条件工作流案例

条件工作流功能允许用户自定义执行条件来决定是否运行某些Job。条件可以由当前Job 的父Job输出的运行时参数构成，也可以使用预定义宏。在这些条件下，用户可以在确定Job 执行逻辑时获得更大的灵活性，例如，只要父Job之一成功，就可以运行当前Job。

运行时参数案例

1）基本原理

（1）父Job将参数写入JOB_OUTPUT_PROP_FILE环境变量所指向的文件

（2）子Job使用${jobName:param}来获取父Job输出的参数并定义执行条件

2）支持的条件运算符：

（1）== 等于

（2）!= 不等于

（3）> 大于

（4）>= 大于等于

（5）< 小于

（6）<= 小于等于

（7）&& 与

（8）|| 或

（9）! 非

3）案例：

需求：JobA 执行一个shell脚本。JobB 执行一个shell脚本，但JobB不需要每天都执行，而只需要每个周一执行。

（1）新建JobA.sh

```
#!/bin/bash
echo "do JobA"
wk=`date +%w`
echo "{\"wk\":$wk}" > $JOB_OUTPUT_PROP_FILE
```

（2）新建JobB.sh

```
#!/bin/bash
echo "do JobB"
```

（3）新建condition.flow

```
nodes:
  - name: JobA
    type: command
    config:
      command: sh JobA.sh
  - name: JobB
    type: command
    dependsOn: - JobA
    config:
      command: sh JobB.sh
      condition: ${JobA:wk} == 1
```

（4）将JobA.sh、JobB.sh、condition.flow 和 azkaban.project 打包成 condition.zip

（5）创建condition 项目 -> 上传condition.zip 文件 -> 执行作业 -> 观察结果

（6）按照我们设定的条件，JobB会根据当日日期决定是否执行。

预定义宏案例

Azkaban中预置了几个特殊的判断条件，称为 `预定义宏` 。预定义宏会根据所有父Job的完成情况进行判断，再决定是否执行。可用的预定义宏如下：

（1）all_success: 表示父 Job 全部成功才执行(默认)

（2）all_done：表示父Job全部完成才执行

（3）all_failed：表示父 Job 全部失败才执行

（4）one_success：表示父Job至少一个成功才执行

（5）one_failed：表示父Job 至少一个失败才执行

1）案例

需求：JobA 执行一个shell脚本，JobB 执行一个shell脚本，JobC执行一个shell脚本，要求JobA、JobB中有一个成功即可执行

（1）新建JobA.sh

```
#!/bin/bash
echo "do JobA"
```

（2）新建JobC.sh

```
#!/bin/bash
echo "do JobC"
```

（3）新建macro.flow

```
nodes:
  - name: JobA
    type: command
    config:
      command: sh JobA.sh
  - name: JobB
    type: command
    config:
      command: sh JobB.sh
  - name: JobC
    type: command
    dependsOn: - JobA
                 - JobB
    config:
      command: sh JobC.sh
      condition: one_success
```

（4）JobA.sh、JobC.sh、macro.flow、azkaban.project 文件，打包成 macro.zip。注意：没有JobB.sh。

（5）创建macro项目 -> 上传macro.zip文件 -> 执行作业 -> 观察结果

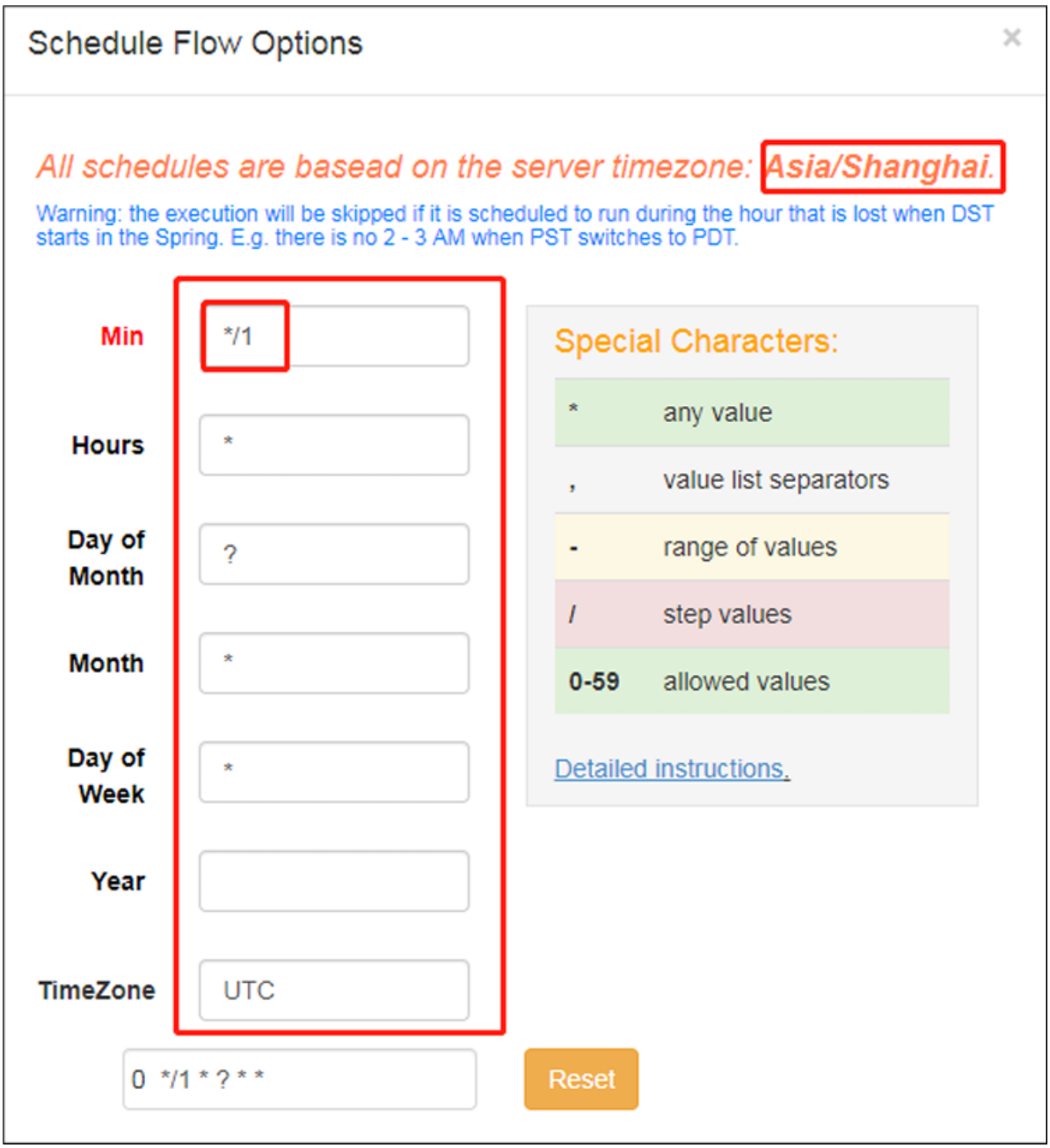
定时执行案例

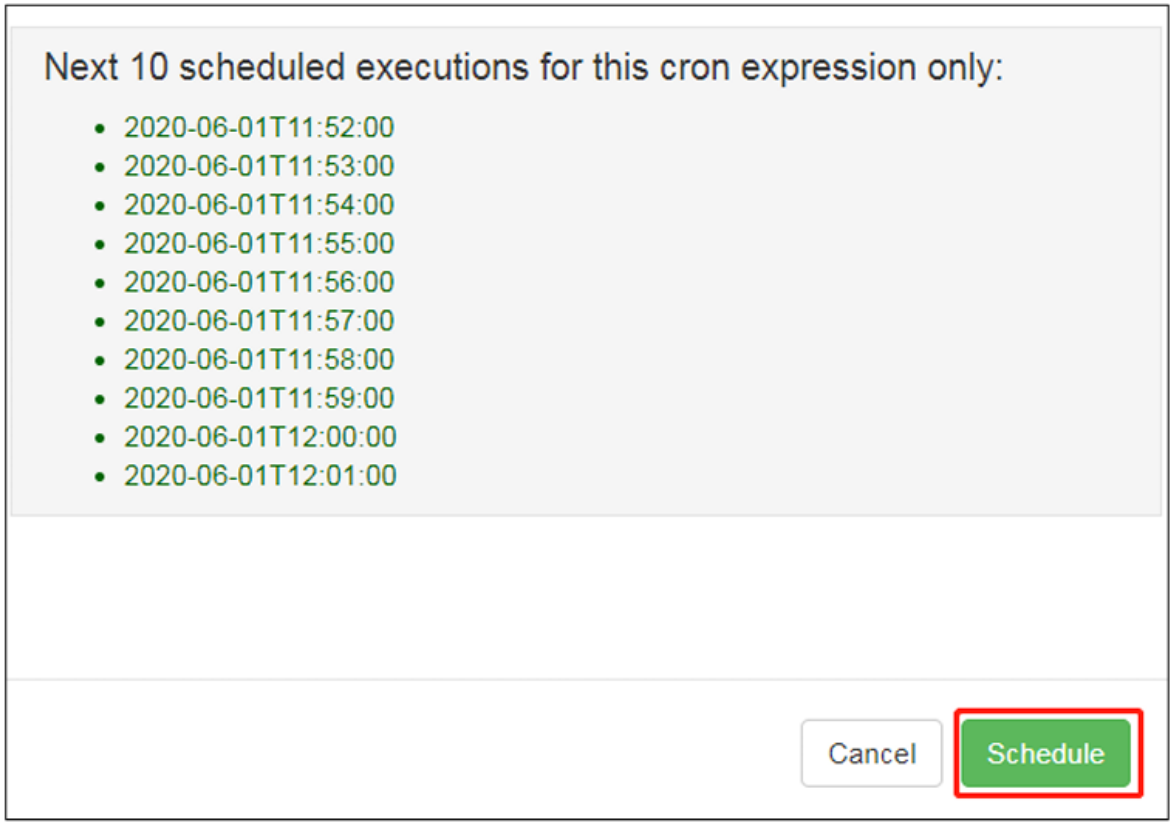
需求：JobA每间隔1分钟执行一次；

具体步骤：

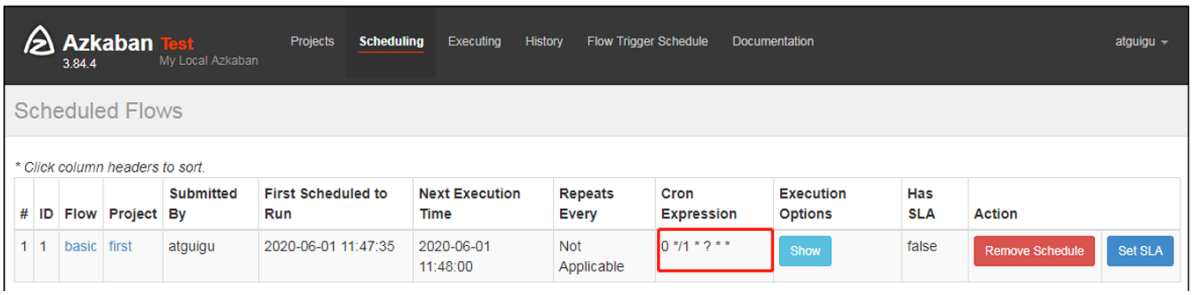
1）Azkaban 可以定时执行工作流。在执行工作流时候，选择左下角Schedule。

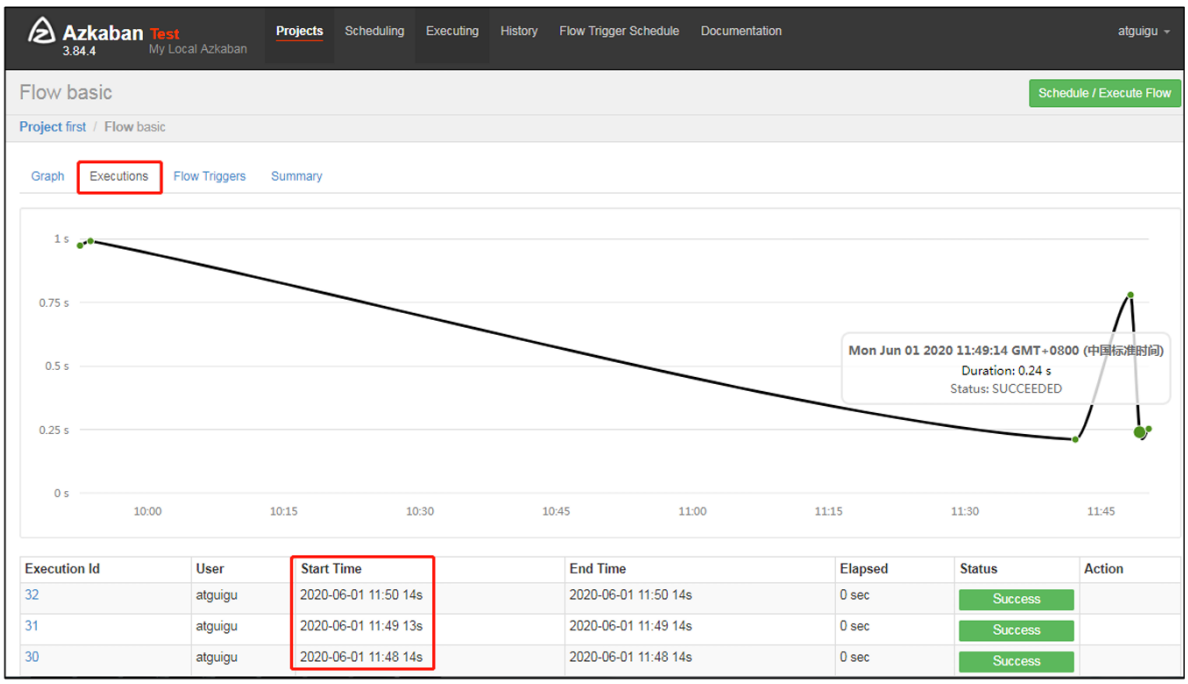
2）右上角注意时区是上海，然后在左面填写具体执行事件，填写的方法和crontab配置定时 任务规则一致。


Schedule Flow Options
All schedules are basead on the server timezone: Asia/Shanghai.
Warning: the execution will be skipped if it is scheduled to run during the hour that is lost when DST starts in the Spring. E.g. there is no 2 - 3 AM when PST switches to PDT.
Min
*/1
Hours
*
Day of Month
?
Month
*
Day of Week
*
Year
TimeZone
UTC
Special Characters:
* any value
, value list separators
- range of values
/ step values
0-59 allowed values
Detailed instructions.
0 */1 * ? * *
Reset



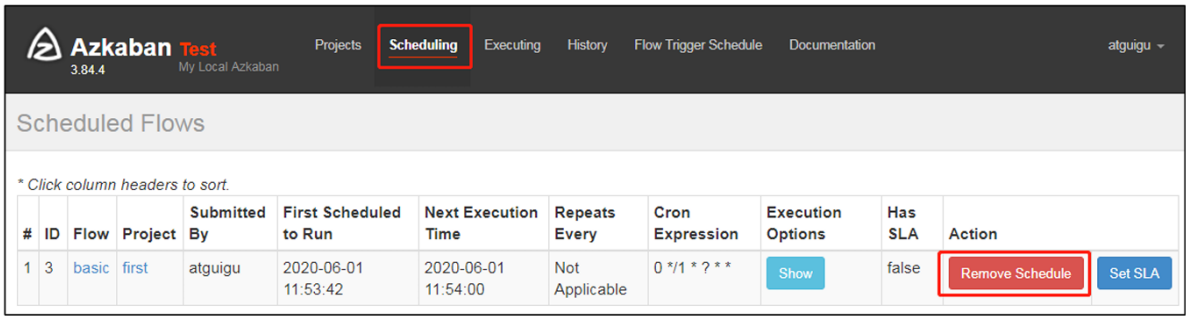
3）观察结果





4）删除定时调度

点击remove Schedule即可删除当前任务的调度规则。



多Executor模式注意事项

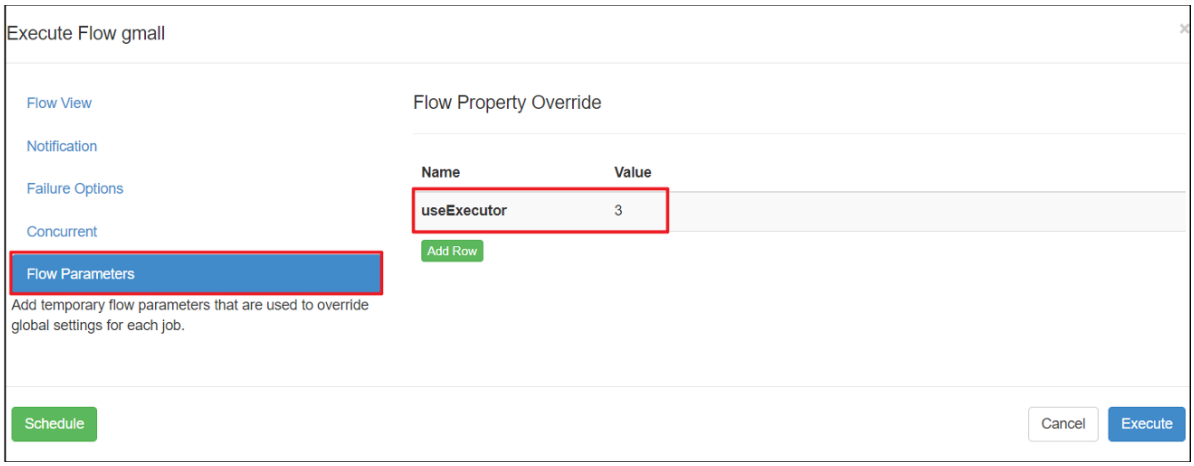
Azkaban多Executor模式是指，在集群中多个节点部署Executor。在这种模式下，Azkaban web Server会根据策略，选取其中一个Executor去执行任务。为确保所选的Executor能够准确的执行任务，我们须在以下两种方案任选其一，推荐使用方案二。

方案一：指定特定的Executor（hadoop102）去执行任务。

1）在MySQL中azkaban数据库executors表中，查询hadoop102上的Executor的id。

```
use azkaban;
select * from executors;
```

2）在执行工作流程时加入useExecutor属性，如下



方案二：在Executor所在所有节点部署任务所需脚本和应用。