工作流调度器Azkaban学习

Azkaban是什么

我们在工作中应该都遇到过这样的场景：有一个任务，这个任务可以划分成多个较小的任务完成，之所以进行划分是因为小任务之间可以并发的进行，例如是一个shell脚本执行的命令吧，大任务A可以划分成B、C、D、E四个子任务（脚本）完成，而B和C是可以同时进行的，D依赖B和C的输出，E又依赖D的输出，于是我们一般的做法可能就是开两个终端同时执行B和C，等两个都执行完成之后再执行D，接着在执行E。整个执行的过程都需要我们参与，但是整个的执行过程类似一个有向无环图，每一个子任务的执行可以看作整个任务的一个流，我们可以同时从没有入度的节点开始执行，任何没有流向（两个节点之间没有通路）关系节点都可以并行得执行，人为的控制难免就有点力不从心了（因为很多任务都需要在深夜执行，一般我们都是写脚本并设置cron），这时候我们需要的就是一个工作流调度器。

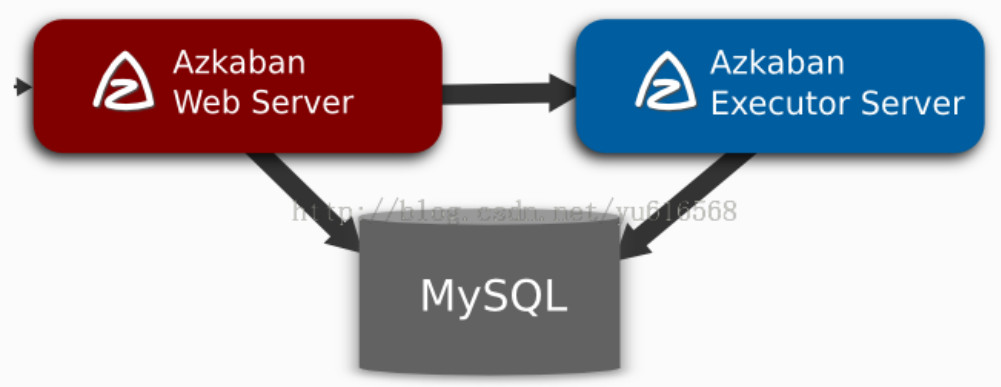
Azkaban就是完成这种任务的（其实主要还是用于对hadoop生态圈的任务的支持），它是由Linkedin实现并开源的，主要用于在一个工作流内以一个特定的顺序运行一组工作和流程，它的配置是通过简单的key:value对的方式，通过配置中的dependencies来设置依赖关系，这个依赖关系必须是无环的，否则会被视为无效的工作流。 Azkaban有如下功能特点：

> - Web用户界面
> - 方便上传工作流
> - 方便设置任务之间的关系
> - 调度工作流
> - 认证/授权(权限的工作)
> - 能够杀死并重新启动工作流
> - 模块化和可插拔的插件机制
> - 项目工作区
> - 工作流和任务的日志记录和审计

我觉得这些都是一些主流的工作流调度器应该支持的功能，我觉得azkaban的web页面做得比较好，这样可以大大降低管理成本，它支持的任务调度类型是基于插件的，这也就使得我们可以实现自己的插件来完成特定的需求。另外，它还能够在任务完成、失败、成功的时候发送email，支持SLA设置等功能，总体来说，功能还是很强大的。

安装部署

azkaban分为三个组建：mysql服务器、web服务器和executor服务器，其中mysql用于存储一些项目以及执行计划（所有任务的属性信息、执行计划、执行的结果以及输出），每次执行情况等信息；web服务器使用Jetty对外提供web服务，是用户可以通过web页面方便的管理；执行服务器是负责具体的工作流的提交，执行，可以启动多个执行服务器，它们通过mysql数据库来协调任务的执行。



首先需要从官网上下载各个模块，都是二进制的安装包格式，当然也可以使用源码编译，下载地址：http://azkaban.github.io/downloads.html

接下来的安装过程可以参考：http://blog.javachen.com/2014/08/25/install-azkaban/　　因为web客户端是通过https的方式进行访问的，所以这里需要创建一个keystore证书文件，使用命令：keytool -keystore keystore -alias jetty -genkey -keyalg RSA，按照提示输入需要的信息，最后输入的”输入 的密钥口令“可以和密钥库口令一样，并且需要在web服务器的配置文件azkaban.properties中修改Jetty服务器的属性，其中

```
jetty.keystore=keystore
jetty.password=redhat
jetty.keypassword=redhat
jetty.truststore=keystore
jetty.trustpassword=redhat
```

设置为你生成的证书文件的信息。接着就可以在浏览器中输入https://ip:8443访问azkaban了（登录的用户名和密码是在web服务器的user配置文件中设置的，这里我们使用的是admin）。

测试

这里我们进行简单的测试，由于azkaban原生是支持shell命令（所以也就可以支持shell脚本以及python等其他脚本程序）的，所以可以使用简单的shell命令进行测试，我们创建4个子任务，每一个子任务的配置都是任务名.job文件。它们的配置如下：

```
test.job
type=command
command=sleep 3
command.1=echo "Hello World"

start.job

type=command
command=sleep 5
command.1=echo "start execute"

sleep.job
type=command
dependencies=test, start
command=sleep 10

finish.job
type=command
dependencies=sleep
command=echo "finish"
```

这里通过dependencies属性来标识该任务依赖的任务，可以有一个或者多个，通过","分割，这些任务的type都是command，azkaban也支持其它类型的命令类型，有些需要安装插件才能支持。

然后我们将这四个job文件放在一个目录下压缩成一个zip文件，在Azkaban的web界面的首页可以通过”Create Project“按钮来创建新的一个工作流，输入必要的信息之后会进入到project界面，我们可以通过upload上传我们要执行的任务流，可以重复upload进行覆盖。但是之前的任务流的执行结果不会被覆盖。如果工作流的配置有问题（例如出现相互依赖），上传会不成功，但是没有看到提示。等待压缩文件上传成功之后，我们可以通过界面查看各个任务的依赖关系图：

可以通过”Execute Flow“按钮来启动一个工作流的一次执行，点击之后会进入配置界面，包括其中包括”Flow View“、”Notification“、”Failure Options“、”Concurrent“、”Flow Parameters“，另外还需要注意的是左下角的Schedule按钮，这里可以设置工作流的定时执行。注意，这里是每一次工作流执行的时候都需要设置的，目前没有看到保存历史设置的情况，当然如果希望重复之前执行的一次设置的话可以找到之前的那次执行，然后再次运行（这时候还是需要进入配置页面，但是会保存那次运行的配置）。其中需要注意的是在”Failure Options“和”Concurrent“中的配置，他们分别配置了在工作流中一个任务执行失败之后的处理和这个project的多次执行流（多次Execute）如果存在并行时的处理。我们在这里不进行配置，直接执行命令：　　提交之后会提示本次执行的id（我觉得这里通过一个可识别的字符串进行标示会更好一些），这个id是全局唯一的，也就是说多个project的每一次执行都会递增得到新的exec id。

这些配置项的作用如下：
* Flow View：可以激活或者取消其中的某一个job，这里应该是在执行flow的时候执行或者不需要执行某个job再执行下面的jobs
* Notification：设置执行完成（失败或者成功执行）之后设置通知的email，这里可以通过修改代码添加其他的通知方式，例如短信等
* Failure Options：这里可以设置某一个job失败之后的动作，目前有三个选项：1、等待着所有正在执行的job的完成（可能在一个flow中有多个job可以并行执行）；2、全部取消，立即终止全部的job，flow执行失败；3、尽可能的继续执行，只要它的依赖jobs能够执行完成。
* Concurrent：因为一个flow的执行时间可能比较长，这里设置多个flow并发执行时的策略，有三个选项：1、顺序执行，如果该flow正在执行则不再执行；2、并发执行，不管并发执行的flow；3、pipeline，有两种策略，第一种是等到jobA需要等到前一个flow的jobA执行完成之后才能执行，第二种是jobA需要等到前一个flow中所有依赖A的job都执行完成之后才能执行。
* Flow Parameters：设置flow执行的私有配置项

执行完成之后，可以通过web界面查看每一个任务流的执行结果以及每一个子任务的执行结果。

在Graph标签下可以查看每一个任务执行的情况、当前执行到哪一个任务了，Flow Log中会实时得输出工作流的运行日志，点击每一个子任务可以查看子任务的运行状态以及实时输出的日志信息，总体来说还是非常方便的。

这里涉及的几个概念：project、flow和job，首先一个project是一个要执行任务的整体，它可以包含多个flow，每一个project可以上传一个.zip的文件，flow之间是相互独立的，但是有一个总的flow，这个flow可能引用其他的flow作为它执行的一部分（相当于总flow的一个子job，但是这个job是一个flow）。每一个flow包含多个job，这些job是相互独立的，通过job文件中dependencies设置依赖关系，每一个flow的结束job可以作为这个flow的标识

（flow名），我们可以通过这样的方式将一个flow作为一个job加入到另外的flow中： jobGroup.job type=flow flow.name=finish dependencies=realStart

finish是之前定义的flow的标识（因为它是终止job），这个flow作为一个job可以设置其他的依赖关系，下面是一个包含子flow的任务依赖图：

我觉得之所以要设计成这样是为了将每个flow独立出来，方便flow的重用。

用户管理

azkaban中有用户和用户组的概念，用户和用户组以及权限的配置信息保存在配置文件azkaban-users.xml中，认证的方式是由azkaban.user.XmlUserManager来实现的，具体的配置可以在azkaban.properties（web服务器的conf下）进行配置：

```
Parameter   Default
user.manager.class  azkaban.user.XmlUserManager
user.manager.xml.file   azkaban-users.xml
```

我们在azkaban-users.xml可以配置三类内容：user、group和role，user项可以配置username、password、roles、group信息，分别配置用户名、密码、用户的权限以及所属的组；group项可以配置name和roles，分别用于配置组名和这个组使用的权限；role定义了权限信息，可以配置name和permissions，分别表示规则名和赋予的权限信息。azkaban支持的权限包括：

```
Permissions     Values
ADMIN           可以做任务事情，包括给其他用户添加、修改权限
READ            只能访问每一个project的内容和日志信息
WRITE           可以在已创建的project上传、修改任务的属性，可以删除任何的project
EXECUTE         允许用户执行任何的任务流
SCHEDULE        允许用户添加、删除任何任务流的调度信息
CREATEPROJECTS  在禁止创建project的情况下仍允许创建新的project
```

这里的权限设置没有细化到每一个user在每一个project中，每一个用户所拥有的权限可以在每一个project下面执行相同的操作，另外用户和用户组之间的权限信息还不是很明确，如果使用用户组作为权限的分配单位（即一个用户组下的所有用户拥有相同的权限），每个用户再次指定权限就有点多余了。

API

azkaban也提供了API接口来使用，这样可以基于azkaban实现自己的管理方式，这些接口是通过HTTPS的方式与web服务器进行通信的，因为在azkaban中有用户和权限的概念，所以在调用API之前需要登录，登录成功之后会返回用户一个session id，之后所有的操作都需要携带这个id以判断用户是否有权限。如果session id无效，那么调用API会返回"error" : "session"的信息，如果不携带session.id参数，会返回登陆界面的html文件内容（有些session id的访问也会返回这样的内容）。azkaban提供的API包括：具体请参照官方文档：http://azkaban.github.io/azkaban/docs/2.5/#ajax-api

- Authenticate：用户登录操作，需要携带用户名和密码，如果成功登录则返回一个session id用于之后的请求。
- Create a Project：创建一个新的project，这需要在任何关于这个project操作之前进行，需要输入project的name作为这个project的唯一标示，还需要包含这个project的描述信息，其实和在web页面上创建project的输入一样。
- Delete a Project：删除一个已经存在的project，该请求没有回复信息，需要输入project的标识。
- Upload a Project Zip：上传一个zip文件到一个project，一般在创建一个project完成之后，之后的上传将覆盖以前上传的内容。
- Fetch Flows of a Project：获取一个project下的所有flow信息，输入需要指定project的标识，一个project下面可能存在多个flow，输出的flow只包含flowId标识每一个flow。
- Fetch Jobs of a Flow：获取一个flow下所有job的信息，因为在API端每个命令都是独立的，所以这里需要输入project的标识和flow的标识，输出包含每一个job的信息，包括job的标识（id）、job类型以及这个job直接以来的job。
- Fetch Executions of a Flow：获取flow的执行情况，需要制定特定的project和flow，这个接口可以分页返回，所以需要制定start指定开始的index和length指定返回的个数，因为每一个flow都可以单独的或者作为其他flow的子flow执行，这里返回该flow指定区间内的每一次执行的信息。每一个执行信息包括起始时间、提交执行的用户、执行的状态、提交时间、这次执行在全局的id（递增的execid），projectid、结束时间和flowId。
- Fetch Running Executions of a Flow：获取当前正在执行的flow的执行信息，输入包括project和flow的标识，返回的是该flow正在执行的所有执行id（全局的exec id）。
- Execute a Flow：启动一个flow的执行，这个输入比较多，因为在web界面上每次启动flow的执行都需要设置几项配置，可以在该接口设置出了调度之外的乞讨配置信息，输入还需要包括project和flow的标识，输出为这个flow的id和本次执行的exec id
- Cancel a Flow Execution：取消一次flow的执行，需要输入的是全局的exec id，因为这个id是全局唯一的，那么可以通过它来进行标识，不需要再输入project和flow的标识了，如果这个执行已经结束，会返回错误信息。
- Pause a Flow Execution：暂停一次执行，输入为exec id。如果这个执行不是处于running状态，会返回错误信息。
- Resume a Flow Execution：重新启动一次执行，输入为exec id，如果这次执行已经在进行，不返回任何错误，如果它不再运行则返回错误信息。
- Fetch a Flow Execution：获取一次执行的所有信息，输入为exec id，输出包括这次执行的属性（参见7），还包括这次执行的所有的job的执行情况。

- Fetch Execution Job Logs：获取一次执行中的一个job的执行日志，可以将job的执行日志作为一个文件，这里需要制定exec id、job的标识以及读取这个文件内容的返回（offset+length），返回的为指定范围的日志内容。
- Fetch Flow Execution Updates：这个是返回上次查看之后每个任务的执行情况？这个有点疑惑。应该是在flow执行的时候执行进度的信息获取。

从这里的接口可以看出，azkaban提供的API只能用于简单创建project、flow，查看project、flow、execute等操作，而web界面的操作要比这丰富得多，如果我们希望基于azkaban进行开发的话，在这些接口的基础上，我觉得还可以对azkaban的数据库进行分析，从数据库中得到我们想要的信息（基本的写操作都能够通过这些API实现，所以我们只需要从数据库中读取）。但是这样相对于使用API还是有个弊端，毕竟随着版本的更新数据库的结构可能会发生变化，但是这也不失为一种方式。

总结

好了，本文主要介绍了azkaban的安装以及使用情况，但是它主要还是用来执行hadoop生态圈里面的各种操作以及java程序的，但是简单的使用还是让我认识到这个工具的强大，但是我还是有一个疑问，azkaban的三个模块的主要功能分别是：mysql用于数据的存储，web服务器用来更方面的使用和图形化展示，executor才是真正的执行任务的服务器，所以所有job的执行都需要在executor所在的机器上进行，job的执行时启动一个子进程的方式（可以通过在job执行是查看正在执行的job命令判断），那么这个executor需要安装所有的支持的任务的工具、jar包等。如果是对于那种占用资源比较多的job（例如一个java程序CPU使用率达到100%），那么就会对其他的job的执行有影响，所以这种架构的可扩展性是否有点欠缺？或者是由于这个工具主要是执行一些hadoop任务，客户端的压力并不大，所以没有考虑这方面。

不过总体来说这是一个比较好的工具，至少web界面可以很方便和直观的查看任务的执行以及运行结果（P.S.azkaban对任务执行结是否成功是怎么判断的？），虽然文档上说它可以支持多个executor，但是实际上并没有发现这么用的，我觉得可以继续改进它来实现多个机器之间任意的程序之间的并行，例如有多个job可以并行执行的，我有多台executor服务器，我可以将任何一个job部署到任何一个executor上执行，充分利用所有的硬件资源。我靠，这不就成了hadoop中jobTracker和TaskTracker的结果了？算了。这个就纯属个人的瞎扯了。