

Omni Workflow 工作流引擎 101

By Howie

工作流應用目的

工作流(Workflow/Process)是对工作流程及其各操作步骤之间业务规则的抽象和概括描述. 每一個工作流都會對應一個主要的業務模型(BizModel), 各步驟(WorkflowStep)對應/定義此模型的一個狀態(Status),以及進入此狀態下系統要執行的動作(Action)和可能返回的動作結果(ActionResponse). 操作可以是系統自動完成的(比如調用另一個外部服務)也可以是需要人工操作完成的(比如等待倉庫人員卸貨下架).

定時任務(Scheduler)是用來定時定點或者相隔一段時間執行一些系統操作的進程, 一般情況都是循環執行, 不需要人工干預. 在 OmniWorkflow 的實現裡面, 定時任務也是工作流.

OmniWorkflow 引擎配置

工作流和定時任務必須跑在工作流引擎上, 跟著下面的步驟可以讓引擎啟動起來

1. 在 maven 的 pom.xml 裡面增加對應的版本號

```
<dependency>
  <groupId>com.omniselling</groupId>
  <artifactId>common-workflow</artifactId>
  <version>3.3.0</version>
</dependency>
```

2. 在 Spring 裡面引用 common-workflow 的啟動

Spring 的 applicationContext.xml 裡面的配置加上

```
<bean class="com.omniselling.wf.configuration.OmniWorkflowConfiguration">
  </bean>
```

如果使用注釋方式的配置

```
@Import({OmniWorkflowConfiguration.class})
```

3. 配置可選參數 configure.properties 或者引用一個 workflow.properties

```
#workflow data source 配置引擎的 DB 連接
workflow.jdbc.driverClassName=com.mysql.jdbc.Driver
workflow.jdbc.url=jdbc:mysql://127.0.0.1:3306/omniwms_fw
```

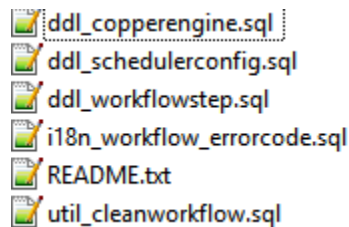
```
workflow.jdbc.username=sb  
workflow.jdbc.password=won'tunderstand
```

#flow config 工作流其它相關配置

```
workflow.persistent.batcher.size=10    #每批次處理工作流數量  
workflow.multiEngineMode=true          #多 JVM 共享模式,正式環境建議不要使用  
workflow.engine.id=engine_wms          #引擎 ID,每個 JVM 需要唯一
```

4. 準備數據庫, 跑工作流引擎對應的 SQL (DDL)

用 7zip/winrar 打開 common-workflow-3.2.8.jar 裡面的 mysql 目錄



- 1) ddl_copperengine.sql - copperengine 的原始 ddl
- 2) ddl_workflowstep.sql - 工作流引擎步驟表的 ddl
- 3) ddl_schedulerconfig.sql - 定時任務配置 ddl
- 4) i18n_workflow_errorcode.sql - 通用 errorcode, insert 到 omniv5_core 的國際化表

util_cleanworkflow.sql - 這個是用來清理所有的工作流實例的, 只允許在測試環境使用

到此引擎就能正常啟動了.

如果想要了解更多關於 OmniWorkflow 基於的 copper-engine 請參考

官網: <http://copper-engine.org/docs>

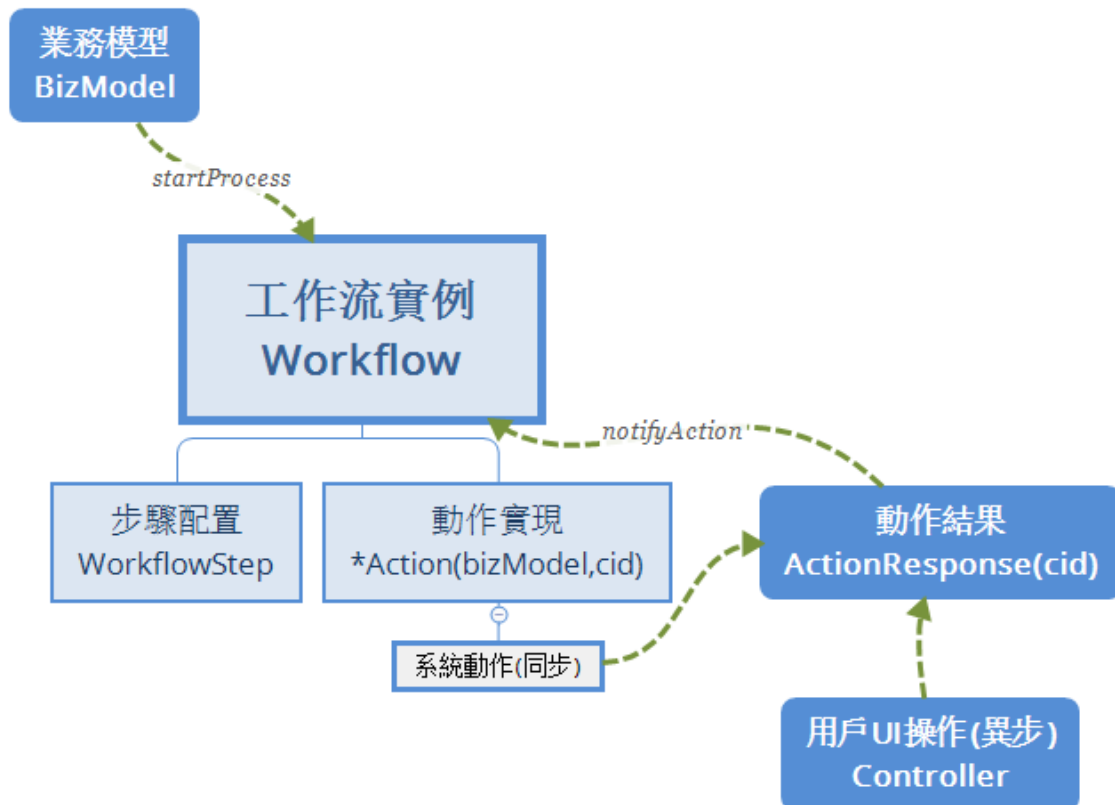
社區: <https://groups.google.com/forum/#!forum/copper-engine>

需要翻牆或者在易達云公司內網查看

Workflow workflow配置

工作流的配置是完全基于流程和对业务模型的理解的,可以独立于业务代码的实现而进行单独测试,以下是一个完整工作流需要配置的东西.

注意: 请分清楚工作流步骤配置(definition)和工作流实例(instance), 其实就是 Class 与 Object 的关系. Definition 只有一个,但是实例会有多个,根据传入的 bizmodel 和过程中用户操作,其最终流程可能也会不一样.



1. 業務模型(BizModel)

業務模型應該事先就有定義好,可能是一個業務單,一個工單或者一個 String,只要有它的意義就可以.它在每一個步驟執行動作 (Action) 的時候將傳給開發者自定的 Action 實現,由開發者自己決定如何去使用

注意: 業務模型需要 implement Serializable

2. 動作實現(*Action)

Action 是进入每一個工作流步驟时执行的动作,或者說是業務模型改變到一個狀態的時候需要做的一些動作. 这些 Action 要用 Java 代码实现并放在 Spring 的容器里面以 bean 的形式存在. 执行到这个步骤的时候马上会执行此 Action. 目前简单的做法是将他们放在任何以 com.omniselling.workflowadapter.* 开头的 package 下面注释成 @Component, OmniWorkflow 会自动扫描这些而不用额外配置.

所有的 Action 都应该 extends BaseWorkflowAdapterImpl
implements IWorkflowExternalAction<M>

比如:

```
package com.omniselling.workflowadapter;

import org.apache.logging.log4j.LogManager;

@Component
public class FirststripReceiveActionImpl extends BaseWorkflowAdapterImpl
    implements IWorkflowExternalAction<InboundOrder>
```

需要实现的方法只有一个: ActionAck execute(M bizModel, String cid)

```
/**
 * 注意基類BaseWorkflowAdapterImpl
 * @author Atomic
 *
 * @param <M>
 */
public interface IWorkflowExternalAction<M extends Serializable> extends IWorkflowEngineNotifier
{
    /**
     * 執行動作
     * @param bizModel
     * @param cid - 系統生成的唯一id用來標識此次交互，不強制要求使用，如果願意也可以生成自己的cid放到ActionAck裡面
     * @return ActionAck - 如果ActionAck.needWaiting = true 則只是發起了一個任務，需要等待調用方的回復 (notify)，
     * 否則ActionAck.response是返回值
     */
    ActionAck execute(final M bizModel, final String cid);
}
```

execute 接口里面做需要做的处理, 可以是调用一个外部接口, 也可以是等待用户的一个操作甚至开启另一个工作流. 跟 Action 相关的一些关键点:

cid - correlation id, 需要是全局唯一, 系统会生成或是自己设置都行, 它是异步操作的关联 key, 用于把 Action 和 ActionResponse 串起来让引擎知道哪个步骤的用户操作完成了.

ActionAck --用来通知工作流引擎此步骤的 action 已执行, 必须包含 cid, 如果是同步操作也必须包含 ActionResponse, 因为应该已经知道结果了

ActionResponse -- 用来通知工作流引擎此步骤 action 执行结果, 必须包含同样的 cid 以及一个用户操作的完成结果 result (比如 Complete 或者 Cancel)

异步模式 -- 此步骤的完成需要用户在页面上进行操作(大部分应该是这种情况), 开发者需要记录 cid, 并让用户操作完成的时候返回 ActionResponse(cid,result)到工作流引擎. 页面的操作完成以后可以使用通用的 workflowNotifier.notifyAction 来完成, 参照下章介绍接口 IWorkflowEngineNotifier

同步模式 -- 也就是此步骤的完成不需要等待用户操作的情况, 则需要马上返回操作结果: actionAck.setActionResponse(actionResponse)
同时设置等待为 false: actionAck.setNeedWaiting(false)

所有 Action 都是标准的 bean 可以引用其它 bean 做事, 但是要注意如果 execute 里面任何 exception 抛了出来就会造成此定时任务进入错误状态. 需要修复后 restart

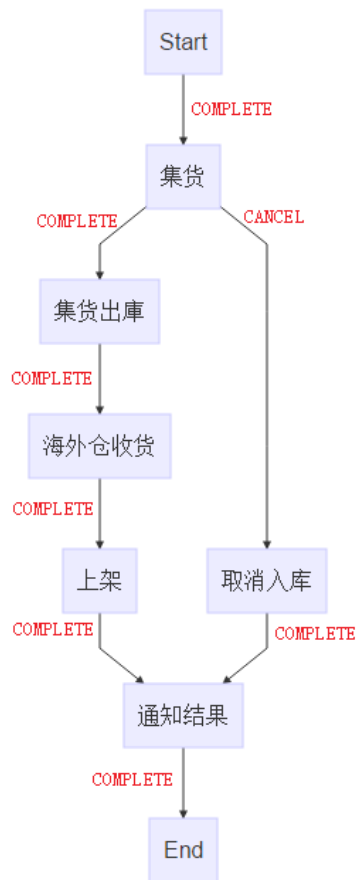
3. 步驟配置

如果有 UI 的話配置起來會比較直觀, 不過目前的版本還沒有做. 所以需要直接在 DB 裡面進行, 建議先做成 csv 的形式然後導入.

步驟配置对应的 db 表: workflowstep

#	Name	Datatype	Len...	Unsigned	Allow NULL	Default	Comment
1	id	BIGINT	20	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AUTO_INC...	
2	wfName	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	No default	工作流名
3	stepName	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	No default	工作流步驟名
4	parentName	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	父工作流步驟名
5	description	VARCHAR	1024	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	工作流簡短說明/補充
6	triggerEvents	VARCHAR	2048	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	觸發此工作流的事件, 應該由上一個stepName+action完成產生的結果組成, 可以是多個
7	actionClass	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	action的class全名, 應該是一個service, 需要獲取這個bean
8	waitTimeOutSec	BIGINT	20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	此步驟等待時間, 如果超時將執行超時動作
9	timeoutActionClass	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	超時的時候執行的動作, 執行完以後會繼續等待
10	resultEvents	VARCHAR	2048	<input type="checkbox"/>	<input type="checkbox"/>	No default	此步驟可能返回的結果事件
11	defaultPoolId	VARCHAR	32	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	默認工作流池ID, 如果為空則進入默認池
12	defaultPriority	INT	10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	默認執行優先級, 如果為空則設為默認優先級(5), 數字越小優先級越高, 1為最高
13	createdBy	BIGINT	20	<input type="checkbox"/>	<input type="checkbox"/>	No default	
14	createdDate	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	No default	
15	modifiedBy	BIGINT	20	<input type="checkbox"/>	<input type="checkbox"/>	No default	
16	modifiedDate	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	No default	

这里举个例子说明一下最好理解, 假设一个简单的流程如下所示



那么它所对应的配置应该是这样的 6 条记录, start 和 end 不需要配:

id	wfName	description	stepName	triggerEvent	actionClass	resultEvents
1	INBOUND_FLOW_TEST_V1	集货	FIRSTRIPRECEIVE		com.omniselling.workflowadapter.FirstripReceiveActionImpl	COMPLETE,CANCEL
2	INBOUND_FLOW_TEST_V1	取消发货	CANCELORDER	FIRSTRIPRECEIVE-CANCEL	com.omniselling.workflowadapter.CancelActionImpl	COMPLETE
6	INBOUND_FLOW_TEST_V1	集货出库	FIRSTRIDISPATCH	FIRSTRIPRECEIVE-COMplete	com.omniselling.workflowadapter.FirstripReceiveActionImpl	COMPLETE
15	INBOUND_FLOW_TEST_V1	海外仓收货	DESTRECEIVE	FIRSTRIDISPATCH-COMplete	com.omniselling.workflowadapter.DestReceiveActionImpl	COMPLETE
16	INBOUND_FLOW_TEST_V1	上架	DESTONSHelf	DESTRECEIVE-COMplete	com.omniselling.workflowadapter.DestOnshelfActionImpl	COMPLETE
17	INBOUND_FLOW_TEST_V1	通知结果	NOTIFYCUST	DESTONSHelf-COMplete, CANCELORDER-COMplete	com.omniselling.workflowadapter.NotifyCustActionImpl	END

需要注意的地方:

- wfName+stepName 需要唯一, stepName 就是步骤名/状态名
- triggerEvent 由 前步骤+前步骤操作结果组成, 如果为空就是第一个步骤(每个工作流只能由一个),多个触发情况需要用逗号(“,”)分割 (比如 id=17 的那一行)
- actionClass 这个就是配置步骤 2 里面的 ActionClass 的全名, 走到这一步将立即执行此 action
- resultEvents 此步骤完成以后所可能产生的操作结果, 也就是 ActionResponse 里面的 result, 可以有多个也是用逗号(“,”)分割 (比如 id=1 的那一行)
- 一个 step 完成以后(resultEvent 通过 ActionResponse 返回以后)会生成 stepName-resultEvent 的 trigger, 并找到下一个需要执行的步骤
- 如果有任何一个步骤完成以后 stepName-resultEvent 对不上任何一个步骤整个工作流就会结束, 所以 END 是不用配置的
- 特别注意, 所有 Action.execute 里面的业务代码如果有 throw exception 出来会让整个工作流进入错误状态, 需要修复代码以后调用 IWorkflowManagement.restartErrorProcesses(List<String>, BaseOperatorInfo) 重做这个步骤

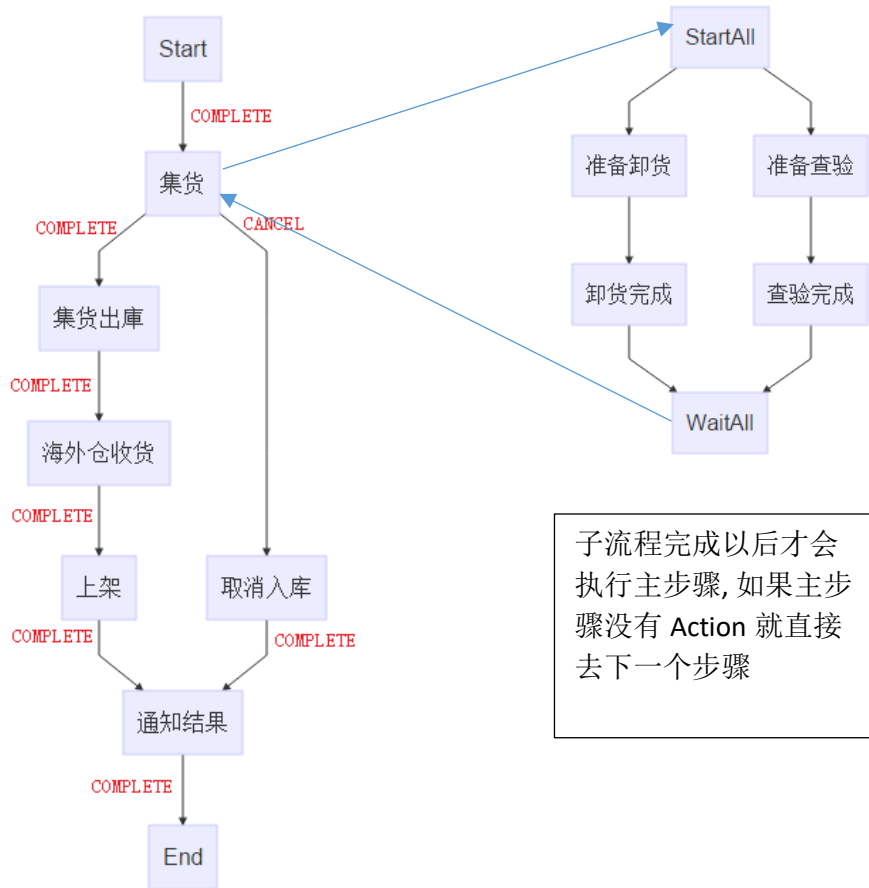
4. 复杂步骤/子流程的设定

复杂步骤(主步骤)指的是一个步骤由其它子流程组成, 运行的时候会并行启动子工作流去处理. 配置的时候需要只要给予流程的步骤都加上 parentName 设定成父步骤的 stepName 就好.

每个子流程第一个步骤就像主工作流的第一个步骤一样 triggerEvent 也要设为空.

父步骤会先分发所有子流程并等待所有子流程完成之后才会去执行自己的 action(如果有的话), 复杂步骤可以不设定 ActionClass, 那么下一个 step 的 eventTrigger 要设置成主步骤的 stepName

假设现在我们上一个例子里面的简单流程第一步前面需要有一个并行的任务“卸货”和“查验”:



这样的话把对应的 4 个 actionClass 添加进入, 用户页面的服务里面的 4 个操作结果配置好, 然后 workflowStep 配置里面多加这 4 行:

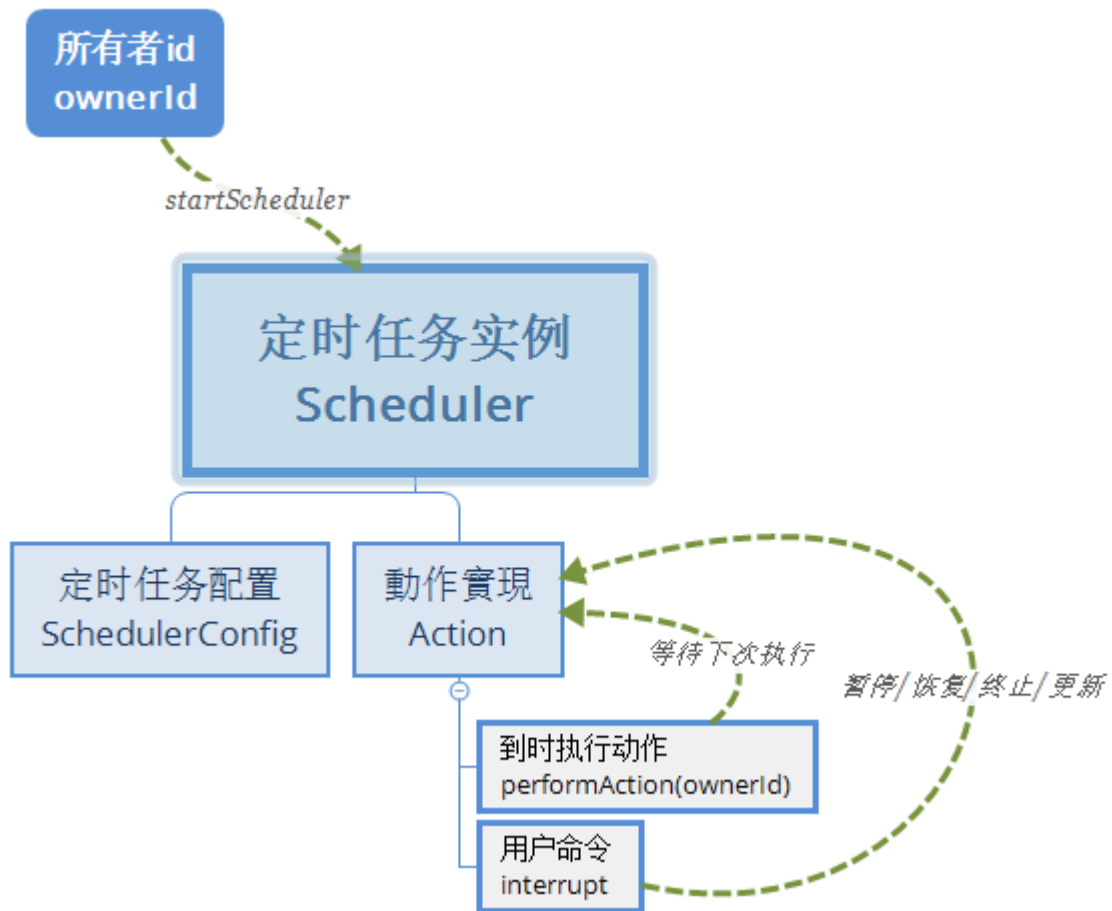
id	wfName	description	stepName	parentName	triggerEvent	actionClass	resultEvents
1	INBOUND_FLOW_TEST_V1	集货	FIRSTSTRIPRECEIVE			com.omniselling.workflowadapter.FirststripReceiveActionImpl	COMPLETE,CANCEL
101	INBOUND_FLOW_TEST_V1	准备卸货	OFFLOAD	FIRSTSTRIPRECEIVE		com.omniselling.workflowadapter.warehouse.OffloadActionImpl	COMPLETE
102	INBOUND_FLOW_TEST_V1	卸货完成	OFFLOAD_FINISH	FIRSTSTRIPRECEIVE	OFFLOAD-COMPLETE	com.omniselling.workflowadapter.warehouse.OffloadFinishActionImpl	COMPLETE
111	INBOUND_FLOW_TEST_V1	准备查验	VERIFYPRODUCT	FIRSTSTRIPRECEIVE		com.omniselling.workflowadapter.warehouse.VerifyProductActionImpl	COMPLETE
112	INBOUND_FLOW_TEST_V1	查验完成	VERIFYPRODUCT_FINISH	FIRSTSTRIPRECEIVE	VERIFYPRODUCT-COMPLETE	com.omniselling.workflowadapter.warehouse.VerifyProductFinishActionImpl	COMPLETE
2	INBOUND_FLOW_TEST_V1	取消发货	CANCELORDER		FIRSTSTRIPRECEIVE-CANCEL	com.omniselling.workflowadapter.CancelActionImpl	COMPLETE
6	INBOUND_FLOW_TEST_V1	集货出库	FIRSTSTRIPDISPATCH		FIRSTSTRIPRECEIVE-COMPLETE	com.omniselling.workflowadapter.FirststripReceiveActionImpl	COMPLETE
15	INBOUND_FLOW_TEST_V1	海外仓收货	DESTRECEIVE		FIRSTSTRIPDISPATCH-COMPLETE	com.omniselling.workflowadapter.DestReceiveActionImpl	COMPLETE
16	INBOUND_FLOW_TEST_V1	上架	DESTONSHelf		DESTRECEIVE-COMPLETE	com.omniselling.workflowadapter.DestOnshelfActionImpl	COMPLETE
17	INBOUND_FLOW_TEST_V1	通知结果	NOTIFYCUST		DESTONSHelf-COMPLETE, CANCELORDER-COMPLETE	com.omniselling.workflowadapter.NotifyCustActionImpl	END

IWorkflowManagement 接口裡面有 validate 和 graph 的接口用来验证和把工作流全流程画出来, 参照使用的章节

Scheduler 定时任务配置

这里的定时任务跟 Quartz 提供的 cronjob 定时任务是不一样的. OmniWorkflow 是持久的, 可带一个自己指定的 model/id 带到自己的 action 里面去处理逻辑, 并且可以在 instance 运行时即时更改 cron 参数或者 interval, 灵活性相对较多.

如果你只需要一个很简单的定时器, 不需要做到在运行时进行更改, 是不需要使用 OmniWorkflow 的定时任务的.



1. 所有者 id(ownerId)

类似工作流的 bizModel, 这个 ownerId 指明此定时任务是为了哪个角色执行的. 一般可能是 accountId, 或者仓库的 warehouseId. 这个 id 会在 action 的 performTask 的时候传给自定义的动作逻辑

2. SchedulerAction 定时任务动作配置

SchedulerAction 在时间到的时候会被系统执行, 配置方式类似工作流的 Action, 每个定时任务只要配一个 SchedulerAction. 这些 Action 要用 Java 代码实现并放在 Spring 的容器里面以 bean 的形式存在. 目前简单的做法是将他们放在任何以 com.omniselling.scheduleradapter.* 开头的包下面注释成@Component, OmniWorkflow 会自动扫描这些而不用额外配置.

所有的 Action 都应该 implements ISchedulerAction, 例如:

```
package com.omniselling.scheduleradapter;





import org.springframework.stereotype.Component;

@Component
public class TestSchedulerAction implements ISchedulerAction
{
    @Override
    public void performTask(String ownerId)
    {
    }
}
```

Action 是标准的 bean, 可以引用其它 bean 做事, 但是要注意如果 performTask 里面任何 exception 抛了出来就会造成此定时任务进入错误状态. 需要修复后 restart 详见“工作流/定时任务 debug”

3. 定时任务配置

Name:	schedulerconfig
Comment:	所有定时任务都需要在这里配置默认属性

Columns:	 Add	 Remove	 Up	 Down
#	Name	Allow NULL	Comment	
1	id	<input type="checkbox"/>	AI	
2	schedulerName	<input type="checkbox"/>	scheduler名字, 自定义	
3	triggerType	<input type="checkbox"/>	触发类型, 参照TriggerType枚举类型目前支持INTERVAL_REPEAT, TIME_REPEAT, RULE(暂时不支持)	
4	autoStart	<input checked="" type="checkbox"/>	设为1: jvm启动的时候会检测是否已经有启动, 如果没有会开启1个instance, 否则设为0或null	
5	defaultOwnerId	<input checked="" type="checkbox"/>	默认的所有者ID, 可以是用户ID, 仓库ID等等, 在performAction的时候会传递过来	
6	enabled	<input type="checkbox"/>	是否启用: 1为启用, 0为不启用, 不影响当前正在运行的scheduler	
7	repeatInterval	<input checked="" type="checkbox"/>	下次执行间隔时间INTERVAL_REPEAT类型必填, 其它类型不起总用	
8	timeUnit	<input checked="" type="checkbox"/>	INTERVAL_REPEAT类型必填其它类型无视, 时间单位, 目前支持的: MILLISECONDS, SECONDS, MINUTES, HOURS, DAYS	
9	cronExp	<input checked="" type="checkbox"/>	TIME_REPEAT类型必填, 使用的触发cron expression其它类型无视	
10	firstRunTime	<input checked="" type="checkbox"/>	首次触发时间, 如果为空则等待配置的间隔时间	
11	totalRunCount	<input checked="" type="checkbox"/>	总执行次数, 如果为空则永远执行下去	
12	actionClass	<input type="checkbox"/>	执行行为的class全名, 应该是一个service, 需要获取这个bean	
13	noCatchUp	<input checked="" type="checkbox"/>	如果设为true则不会在当前时间比下次执行时间晚的情况下立即触发任务. 例子: 上次触发时间是8:30分, 设定是...	
14	defaultPoolId	<input checked="" type="checkbox"/>	默认工作流池ID, 如果为空则进入默认池	
15	defaultPriority	<input checked="" type="checkbox"/>	默认执行优先级, 如果为空则设为默认优先级(5), 数字越小优先级越高, 1为最高	
16	createdBy	<input type="checkbox"/>		
17	createdDate	<input type="checkbox"/>		
18	modifiedBy	<input type="checkbox"/>		
19	modifiedDate	<input type="checkbox"/>		

每个定时任务+默认所有者 id 指定唯一, 默认所有者 id 可以为空那说明你的 action 里面不需要用到.

定时任务可以使用 `ISchedulerService.startScheduler` 启动, 同一个定时任务一样的 `ownerId` 只能启动一个.



如果在配置表里面设 `autoStart=1`, 在 JVM 启动的时候这个定时任务实例会自动运行起来, `action` 里面的 `ownerId` 也会变成自行设定的 `defaultOwnerId`. 但是如果这时候这个定时任务实例已经存在就不会再启动了. 如果这个定时任务实例存在, 但是它进入了错误状态, 重启将尝试恢复它(自动调用 `ISchedulerService.restartErrorProcesses`)

`triggerType` -- 如果定时任务比较复杂, 或者你懂如何设置 `cron Expression` 那就用 `TIME_REPEAT` 类型, 它的使用方式跟 Quartz 的定时任务一样, 请参照这个网站设置 `cronExp`: <http://www.freeformatter.com/cron-expression-generator-quartz.html> 如果只是固定一个时间段的可以设置成 `INTERVAL_REPEAT`, 并配上 `repeatInterval` 和 `timeUnit`

`totalRunCount` -- 设置总共执行几次, 设空就会一直执行下去, 直到被终止

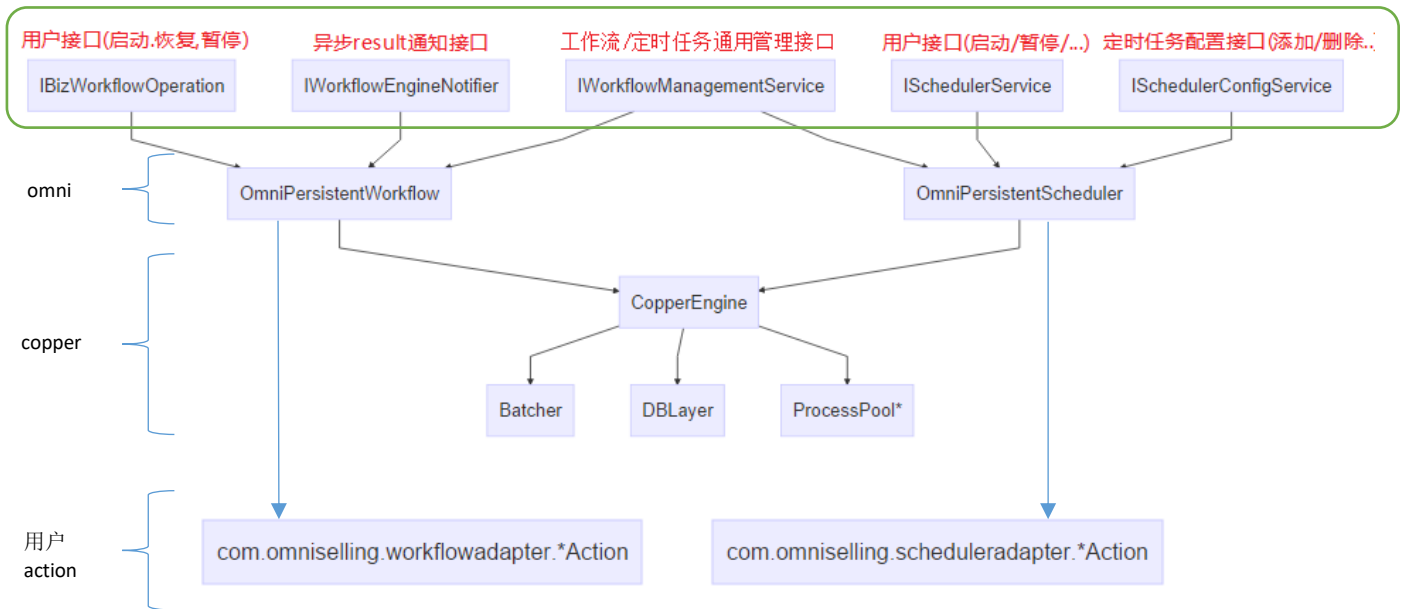
`actionClass` -- 配置方式跟工作流一样, 也是一个完整的 `class path` 指向自定的 `bean`

下面是 2 个例子, 一个是自启动的复杂任务, 用来在星期 1~5 打结算单的配置, 一个是简单的每 5 秒重复一次的定时任务配置

 schedulerName	triggerType	autoStart	 defaultOwnerId	enabled	repeat...	timeUnit	cronExp	firstRun Time	totalRu...	actionClass
CreateManifestRecordsScheduler	TIME_REPEAT	1	CATR-1	1	(NULL)	(NULL)	0 0 * * * MON,TUE,WED,THU,FRI *	(NULL)	(NULL)	com.omnisell
TestScheduler	INTERVAL_REPEAT	0	(NULL)	1	5	SECONDS	(NULL)	(NULL)	(NULL)	com.omnisell

OmniWorkflow 在代码中的使用

将 workflow 步骤配置完毕, 把需要做的 `ActionClass` 实现完毕以后, 就可以启动 workflow 了, workflow 的使用主要包含对以下 5 个接口以及其涉及到的接口 `model` 的理解, 其中 workflow 2 个接口, 定时任务 2 个接口同时还有 1 个通用的管理接口, 请仔细阅读相关接口代码(`Ctrl+Shift+T`) 输入对应的 `interface` 名称:



`IBizWorkflowOperation` -- 工作流的用户接口, 用来启动/暂停/恢复工作流实例的服务, 在 `spring` 容器中 `bean` 的 `id=workflowService` 可以直接 `@Resource` 引用, 需要详细查看接口代码了解, 引用例子:

```
@Resource
IBizWorkflowOperation<M> workflowService; //M 为你自己的 model
```

`IWorkflowEngineNotifier` -- 用于给业务逻辑通知引擎某个步骤操作的结果, 比如 `COMPLETE`, `CANCE` 在 `spring` 容器中 `bean` 的 `id=workflowNotifier`

`IWorkflowManagementService` -- 工作流/定时任务通用管理服务 (因为定时任务是一种特别的工作流) 具体的方法说明请直接查看代码. `beanId=workflowManagementService`

`ISchedulerService` -- 定时任务用户接口, 用于启动/暂停/恢复/终止/运行时更新定时任务实例的一些参数, 比如 `cronExp`, `beanId=schedulerService`, 需要详细查看接口代码了解

`ISchedulerConfigService` -- 定时任务配置接口, 因为定时任务相对简单, 所以定时任务的配置的增删改查可以直接用这个接口来做 `beanId=schedulerConfigService`

注意: 接口的具体的方法说明请一定要打开直接查看代码内部注释说明.

工作流/定时任务 debug 指南

对工作流和定时任务的实例进行 debug 需要熟悉实例层的 DB 表, 懂得使用管理接口, 查服务器 log 和了解 OmniWorkflow 引擎层的 2 个固定基类工作流 OmniPersistentWorkflow 和 OmniPersistentScheduler 的基本运作过程.

实例 DB 表结构解析:

cop_queue

工作流启动前会先放到这个表里面排队, 启动成功以后会从这里删除掉, 注意 WORKFLOW_INSTANCE_ID 就是工作流的实例 ID(下同)
启动相对是很快的, 所以这里一般都是空的

cop_workflow_instance (重要)

工作流/定时任务实例表, 工作流/定时任务在启动以后会将这个实例, 和它所对应的状态保存在这张表中, 注意目前的设定是完成的工作流会从系统中删除掉.

- ID - 此 instance 的唯一识别 ID, 也是 startProcess/startScheduler 成功以后返回的 instanceId. 如果是工作流的话需要将此 ID 保存起来以便出问题的时候 debug
- STATE - 实例状态, 目前的取值范围是 2-等待中, 5-错误, 0-排队等待启动, 1-正在处理. 如果 STATE=5 应该是 action 执行的时候抛异常造成工作流进入错误状态, 这时候应该去查 cop_workflow_instance_error, 修复错误, 然后用管理接口 restartErrorProcesses(Arrays.asList(instanceId)) 再执行一次 Action 并往下继续

cop_workflow_instance_error

工作流/定时任务报错信息表, 这个表只记录出错信息, workflow_instance_id 对应上张表的 ID, 这个表不会自动清空

cop_wait

正在等待操作结果的工作流/定时任务都会有 2 个记录在这里, 1 个是等待异步 Action 的 ActionResponse 或是时间到, 一个是 INTERRUPT 用于处理暂停/恢复/更新... 等用户操作. 完成的工作流这里面应该没有东西

- CORRELATION_ID - 异步交互的 cid
- WORKFLOW_INSTANCE_ID - 等待所属的工作流/定时任务 id
- TIMEOUT_TS - 等待超时时间, 对定时任务来说就是下次执行时间

cop_response

异步操作的返回结果保存在这里, 正常的 response 很快会被引擎取走所以一般是空的. 如果这里面有记录, 需要检查一下是否有人预先传, 多次传, 或者传了错误的 (没

有在等待的) `ActionResponse` 到引擎里面. 这样的话需要查 `IWorkflowEngineNotifier` 的调用者是否有传递正确的 `cid` 和 `result`

cop_audit_trail_event

这个表记录整个工作流/定时任务的过程, 每个步骤 `Action` 的交互, 每个定时任务下次执行时间. 以及报错信息. 这是 **debug 很重要的一个表**. 这张表的内容不会被清掉.

- `SEQ_ID` - 序列号, 用这个来把事件排序
- `LOGLEVEL` - `INFO-0`, `WARN-1`, `ERROR-2`, 可以过滤来看哪里出错的
- `CONTEXT` - 工作流记录的是步骤名, 定时任务记录的可以无视
- `INSTANCE_ID` - 工作流实例 ID
- `LONG_MESSAGE` - 流程报出来的消息, 包括 `action` 的事件交互消息

Workflow 运行过程

1. 工作流需要被 `IBizWorkflowOperation.startProcess` 开启, 这个方法首先会初步验证整个流程的合法性, 比如是否有 `actionClass`, 是否有设定第一个步骤, 但是不会验证 `actionClass` 是否真是一个 Spring bean, 如果有问题会 `throw exception` 出来, 不要忽略它!
2. 如果初步验证 `ok` 就会异步启动工作流, 引擎会先把这个工作流放在 `cop_queue` 里面, 然后读到内存里面, 如果无法启动(比如找不到定义的工作流名称(`wfName`))就会报错
3. 读取所有此工作流的所有 `workflowStep`, 并根据 `stepName`, `result`, `triggerEvent` 确定整个流程
4. 确定第一个步骤 `firstStep`, 然后调用递归方法 `process`
5. `process` 逻辑:
 - a. 检查此步骤有没有子流程, 有的话启动他们并等待他们全部完成
 - b. 执行 `action`, 从容器获取对应的 `bean`, 调用 `.execute`
 - c. 检查 `execute` 的返回值 `ActionAck`
 - i. 如果 `needWaiting=true` (默认情况), 异步等待此 `action` 的操作结果(`ActionResponse`), 保存流程状态到 `cop_workflow_instance`, 释放线程
 - a) 等待用户操作(页面/API)完成, 并根据操作结果把 `ActionResponse` 传递回引擎
`IWorkflowNotifier.notifyAction`
 - ii. 如果 `needWaiting=false`, 从 `ActionAck.getActionResponse()` 获取操作结果 `ActionResponse`
 - d. 用 `ActionResponse` 里面的 `result`+当前 `stepname` 组成 `triggerEvent` 找到下一个步骤 `nextStep`
 - i. 如果能找到下一个步骤则调用 `process` 传入下一个步骤 `nextStep`
 - ii. 否则整个工作流结束, (`audit` 表和服务器 `log` 里面会有一行 `OmniWorkflow Ended, triggerEvent=...`)

定时任务运行过程相对就比较简单了, 自己打开 OmniPersistantScheduler 读一读代码吧, 最重要的是 3 点:

1. 要保证 Action 的 bean 能被扫描到
2. TIME_REPEAT 类型的 cronExp 要是合法的, 包括运行时传过来 update 也要合法
3. 任务行为 performTask 里面不要报错, 如果报错的话定时任务将进入错误状态, 需要修复代码以后使用 ISchedulerService.restartErrorProcesses 重启如果是自动启动的定时任务(autoStart=1), 重启 jvm 将自动重启这些定时任务

一些常见问题和 debug 方法:

1. 关联 bizModel 和工作流

不少同事反映很难关联 bizModel 和其对应的工作流, 其实工作流的 instanceId 是应该要保存起来的, 可以直接 save 在 model 里面, 不过也有一个比较偷懒的方法 -- 使用 IBizWorkflowOperation 指定 pid 的 startProcess 方法, 将 businessNum 放到 pid 里面, 例如 (这个例子里面做了上面 2 点):

```
String pid = bizModel.getBusinessNum()+"@"+UUID.randomUUID().toString();
bizModel.setWfPid(pid);
updateModel(bizModel, o);
workflowService.startProcess(bizModel,WmsConstant.WF_OUTBOUND_ORD_FLOW_V1, pid, o);
logger.info("Process started id = " + pid+", businessnum = "+bizModel.getBusinessNum());
```

2. 流程没有启动

- a. 查 server log 检查引擎是否有启动, db 连线是否有通
- b. 查 server log 检查是否有流程启动的报错

3. 流程不见了,这一般是因为回传的用户操作结果对不上 triggerEvent

- a. 查 auditLog 表(cop_audit_trail_event), 看此流程最后一个 response 的 result 是什么, 用这个去对流程配置表(workflowStep)看是不是找不到下一个步骤结束了, 比如一个步骤期待的 results 为“COMPLETE”但是用户操作回传了“END”或者“COMP”, 造成无法找到下一个步骤而终止
- b. 查 server log 跟踪看上一个用户操作

4. 流程到一个步骤卡住/或订单到一个步骤卡住

- a. 查 cop_workflow_instance 表看是此流程是否存在
- b. 查 error 表看是否有异常, 比如是否 action 里面报错
- c. 查 server log 跟踪看上一个用户操作
- d. 用户操作结果返回 response 的时候设置的 cid 要跟 action 里面产生的 cid 一致, 不一致的话流程会卡住(因为 response 错了)

5. 启动工作流的异步问题,事务问题

- a. workflowService.startProcess 是异步启动工作流的, 所以在启动之前 model 需要保存, 启动的方法不能在事务(transaction)里面.
- b. 同样, 因为是异步启动, 所以 startProcess 以后任何对相关 bizmodel 的改动都有可能覆盖掉 process 里面的改动, 所以要修改 bizmodel 要在 startProcess 之前.同理, startProcess 不要放在事务里面, 否则流程里面的 action 对 model 读取可能读不到还没有 commit 的数据
- c. Action 是工作流执行的, 而操作是用户执行的, 这个过程中会有时间差, 需要注意

- d. 需要保证 ActionAck 和 ActionResponse 中 cid 的一致性, 最好使用系统生成的 cid, 而不是自己乱拼

6. Action 的异步交互问题

- a. 要保证 cid 的唯一性, 一个 cid 只能等待一个 response
- b. 一定要先有等待再有 notify, 否则超过一个小时的 response 将会从 cop_response 里面删除(详见 workflow 交互最佳实践)
- c.
- d.

workflow 设计最佳实践

1. UI 步骤 vs 流程步骤
2. 保证流程完整性
3. `Action.execute` 完整性
4. 同步/异步交互最佳实践

workflow数据修复指南

