# OpenWebFlow 工作流引擎

# 用户手册与设计说明

V0.9

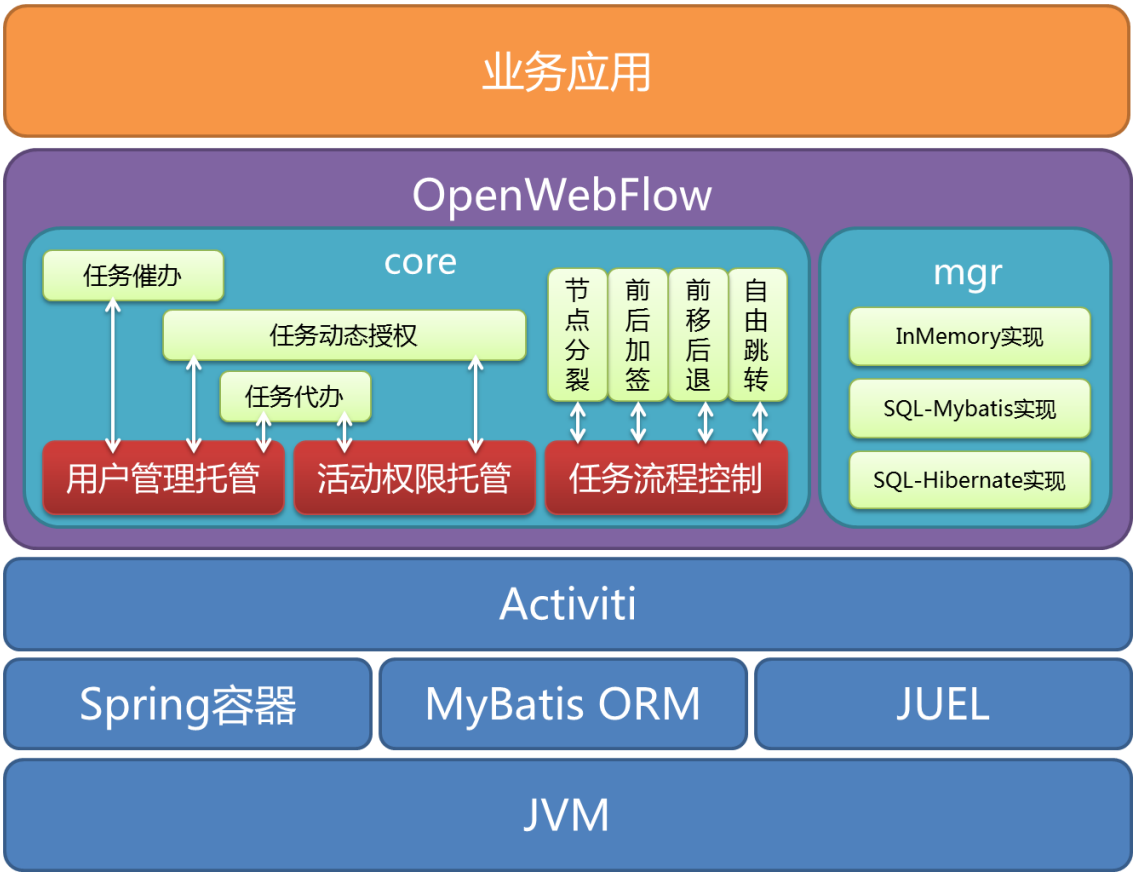
编写：bluejoe2008@gmail.com

2015 年 1 月 22 日

# 目 录

# 1. OpenWebFlow 概述

OpenWebFlow 是基于 Activiti 扩展的工作流引擎。Activiti （官方网站 http://activiti.org/，代码托管在 https://github.com/Activiti/Activiti）是一个新兴的基于 Apache 许可的支持 BPMN 2.0 标准的开源 BPM 产品，它是一个轻量级，可嵌入的 BPM 引擎，并且提供了功能丰富的开发和流程设计工具。OpenWebFlow 与业务应用系统之间的关系如下图所示。




相对于 Activiti，OpenWebFlow 扩展的功能包括：

1) **完全接管了 Activiti 对活动（activity）权限的管理。**

Activiti 允许在设计 model 的时候指定每个活动的执行权限，但是，业务系统可能需要根据实际情况动态设置这些任务的执行权限（如：动态的 Group）。OpenWebFlow 完全实现了与流程定义时期的解耦，即用户对活动的访问控制信息单独管理（而不是在流程定义中预先写死），这样有利于动态调整权限，详见自定义活动权限管理；

2) **完全接管了 Activiti 对用户表（IDENTITY_XXX 表）的管理。**

在标准的工作流定义中，每个节点可以指定其候选人和候选用户组，但是比较惨的是，Activiti 绑架了用户信息表的设计！这个是真正致命的，因为几乎每个业务系统都会属于自己的用户信息结构（包括 User/Group/Membership），但不一定它存储在 Activiti 喜欢的那个库中，表的结构也不一定一样，有的时候，某些信息（如：动态的 Group）压根儿就不采用表来存储。OpenWebFlow 剥离了用户信息表的统一管理，客户程序可以忘掉 Activiti 的用户表、群组表、成员关系表，详见自定义用户成员关系管理；

3) **允许运行时定义 activity！**

彻底满足“中国特色”，并提供了安全的（同时也是优雅的）催办、代办、加签（包括前加签/后加签）、自由跳转（包括前进/后）、分裂节点等功能；

# 2. 快速上手

## 2.1 引入 OpenWebFlow 框架

### 2.1.1 以 jar 的方式引入 OpenWebFlow

OpenWebFlow 的发布形式是一组正常的 jar，其中 openwebflow-core.XXX.jar 包含了核心的工作流控制模块，以及基于内存的管理器实现模块。

此外，OpenWebFlow 还提供了几个 jar：openwebflow-mgr-hibernate.XXX.jar，openwebflow-mgr-mybatis.XXX.jar，它们提供了管理器的 SQL 实现模块，分别选取 hibernate 和 mybatis 作为 ORM 模型。还有一个是 openwebflow-mgr-test.XXX.jar，它包含了几个测试类。

最新版本的下载地址：

https://github.com/bluejoe2008/openwebflow/blob/master/openwebflow-core/target/openwebflow-core-0.9-SNAPSHOT.jar

注意这些 jar 具有较多的依赖，https://github.com/bluejoe2008/openwebflow/tree/master/openwebflow-core/target/lib 列举了所有的依赖包如下：

activation-1.1.jar

activiti-bpmn-converter-5.16.1.jar

activiti-bpmn-layout-5.16.1.jar

activiti-bpmn-model-5.16.1.jar

activiti-crystalball-5.16.1.jar

activiti-engine-5.16.1.jar

activiti-explorer-5.16.1.jar

activiti-image-generator-5.16.1.jar

activiti-json-converter-5.16.1.jar

activiti-process-validation-5.16.1.jar

activiti-simple-workflow-5.16.1.jar

activiti-spring-5.16.1.jar

aopalliance-1.0.jar

commons-collections-2.0.jar

commons-dbcp-1.4.jar

commons-email-1.2.jar

commons-io-2.4.jar

commons-lang-2.6.jar

commons-lang3-3.3.2.jar

commons-logging-1.1.1.jar

commons-pool-1.5.4.jar

dcharts-widget-0.10.0.jar

groovy-all-2.1.3.jar

h2-1.3.168.jar

hamcrest-core-1.3.jar

imgscalr-lib-4.2.jar

jackson-annotations-2.2.3.jar

jackson-core-2.2.3.jar

jackson-databind-2.2.3.jar

javaGeom-0.11.1.jar

jcl-over-slf4j-1.7.6.jar

jgraphx-1.10.4.1.jar

joda-time-2.1.jar

junit-4.12.jar

log4j-1.2.17.jar

mail-1.4.1.jar

mybatis-3.2.8.jar

mybatis-spring-1.2.2.jar

mysql-connector-java-5.1.32.jar

servlet-api-2.5.jar

slf4j-api-1.7.2.jar

slf4j-jdk14-1.7.2.jar

slf4j-log4j12-1.7.6.jar

spring-aop-3.2.4.RELEASE.jar

spring-beans-3.2.4.RELEASE.jar

spring-context-3.2.4.RELEASE.jar

spring-core-3.2.4.RELEASE.jar

spring-expression-3.2.4.RELEASE.jar

spring-jdbc-3.2.4.RELEASE.jar

spring-orm-3.2.4.RELEASE.jar

spring-tx-3.2.4.RELEASE.jar

spring-web-3.2.4.RELEASE.jar

spring-webmvc-3.2.4.RELEASE.jar

vaadin-6.8.8.jar

### 2.1.2 以 maven 的方式引入 OpenWebFlow

以 maven 的方式引入 OpenWebFlow 比较简单，pom.xml 中的依赖项写成：

```
<dependency>
    <groupId>org.openwebflow</groupId>
    <artifactId>openwebflow-core </artifactId>
    <version>0.9-SNAPSHOT</version>
</dependency>
```

在引入依赖项之前可能需要先在本地仓库中安装 OpenWebFlow 项目。具体操作是在 eclipse 中选择 OpenWebFlow 项目，【右键菜单】【Maven】【install】。

## 2.2 配置文件

准备 Spring IoC 配置文件，分别是 settings.properties 、activiti.cfg.core.xml 和 activiti.cfg.mem.xml （或者是 activiti.cfg.sql.XXX.xml）：

- settings.properties：公共属性设置
- activiti.cfg.core.xml：用以配置工作流引擎的基本配置信息；
- activiti.cfg.mem.xml：用以定义一些用以支持 OpenWebFlow 工作的 manager，注意名字中的 mem，它暗示着仅提供了那些 manager 的基于内存实现的版本，类似的配置文件还可以是 activiti.cfg.sql.XXX.xml；

### 2.2.1 settings.properties

settings.properties 文件是一个正常的属性文件，用以 spring IOC 文件加载。如下是一个属性文件的内容：

```
1.  mail.host=smtp.bluejoe.cn
2.  mail.port=25
3.  mail.username=sdb-support@cnic.cn
4.  mail.password=sdbsupport
5.  mail.from=sdb-support@cnic.cn
6.  model.dir=../models
7.  alarm.mail.template=classpath:/alarm-template.txt
8.  hibernate.dialect=org.hibernate.dialect.MySQLDialect
9.  hibernate.hbm2ddl.auto=none
10. activitidb.url=jdbc:h2:mem:activiti;DB_CLOSE_DELAY=1000
11. activitidb.driver=org.h2.Driver
12. activitidb.username=sa
```

```
13. activitidb.password=
14. owfdb.url=jdbc:mysql://localhost:3306/openwebflow?useUnicode=true&amp;characterEncoding=UTF-8
15. owfdb.driver=com.mysql.jdbc.Driver
16. owfdb.username=root
17. owfdb.password=1
```

各属性的含义如下：

| 属性名 | 示例值 | 含义 |
|---|---|---|
| **mail.host** | smtp.bluejoe.cn | 催办邮件发件服务器主机地址 |
| **mail.port** | 25 | 催办邮件发件服务器端口号 |
| **mail.username** | sdb-support@cnic.cn | 催办邮件发件账号名 |
| **mail.password** | sdbsupport | 催办邮件发件账号密码 |
| **mail.from** | sdb-support@cnic.cn | 催办邮件发件人 |
| **model.dir** | ../models | 自动加载的 BPMN 模型路径 |
| **alarm.mail.template** | classpath:/alarm-template.txt | 催办邮件正文模板 |
| **hibernate.dialect** | org.hibernate.dialect.MySQLDialect | Hibernate 方言 |
| **hibernate.hbm2ddl.auto** | none | Hibernate DDL 设置 |
| **activitidb.url** | jdbc:h2:mem:activiti;DB_CLOSE_DELAY | Activiti 数据库 JDBC URL |
| **activitidb.driver** | org.h2.Driver | Activiti 数据库 JDBC 驱动 |
| **activitidb.username** | sa | Activiti 数据库账号名 |
| **activitidb.password** | | Activiti 数据库账号密码 |
| **owfdb.url** | jdbc:mysql://localhost:3306/openwebflow?useUnicode | OpenWebFlow 数据库 JDBC URL |
| **owfdb.driver** | com.mysql.jdbc.Driver | OpenWebFlow 数据库 JDBC 驱动 |
| **owfdb.username** | root | OpenWebFlow 数据库账号名 |
| **owfdb.password** | 1 | OpenWebFlow 数据库账号密码 |

### 2.2.2 activiti.cfg.core.xml 配置

activiti.cfg.core.xml 的配置与 Activiti 要求的那个配置文件有点相似，但可以多一些内容，如下是个例子：

```
1.    <!-- 工作流核心数据库配置 -->
2.    <bean id="activitiDataSource" class="org.apache.commons.dbcp.BasicDataSource"
3.        destroy-method="close">
4.        <property name="driverClassName" value="${activitidb.driver}" />
5.        <property name="url" value="${activitidb.url}" />
6.        <property name="username" value="${activitidb.username}" />
7.        <property name="password" value="${activitidb.password}" />
8.        <property name="initialSize" value="20" />
9.        <property name="maxActive" value="50" />
10.       <property name="maxIdle" value="20" />
11.       <property name="minIdle" value="10" />
12.   </bean>
13.
14.   <!-- 任务催办配置 -->
15.   <bean id="myTaskAlarmService"
      class="org.openwebflow.alarm.impl.TaskAlarmServiceImpl">
16.       <!-- 截止日期提前量 -->
17.       <property name="periodInAdvance" value="P2D" />
18.       <!-- 设置消息通知机制 -->
19.       <property name="messageNotifier">
20.           <!-- 采用邮件发送 -->
21.           <bean class="org.openwebflow.alarm.impl.MailMessageNotifier">
22.               <property name="subjectTemplate" value="请尽快处理#{'$'}{task.name}任务
      " />
23.               <property name="messageTemplateResource" value="${alarm.mail.template}"
      />
24.               <property name="mailSender">
25.                   <bean class="org.openwebflow.alarm.impl.MailSender">
26.                       <property name="serverHost" value="${mail.host}" />
27.                       <property name="serverPort" value="${mail.port}" />
28.                       <property name="authUserName" value="${mail.username}" />
29.                       <property name="authPassword" value="${mail.password}" />
30.                       <property name="mailFrom" value="${mail.from}" />
31.                   </bean>
32.               </property>
33.           </bean>
34.       </property>
35.       <property name="membershipManager" ref="myMembershipManager" />
```

```
36.         <property name="userDetailsManager" ref="myUserDetailsManager" />
37.         <property name="taskNotificationManager" ref="myTaskNotificationManager" />
38.     </bean>
39.
40.     <bean id="transactionManager"
41.         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
42.         <property name="dataSource" ref="activitiDataSource" />
43.     </bean>
44.
45.     <tx:annotation-driven transaction-manager="transactionManager" />
46.
47.     <!-- 配置对象 -->
48.     <bean id="processEngineConfiguration"
    class="org.openwebflow.cfg.ProcessEngineConfigurationEx">
49.         <property name="dataSource" ref="activitiDataSource" />
50.         <property name="transactionManager" ref="transactionManager" />
51.         <property name="databaseSchemaUpdate" value="true" />
52.         <property name="jobExecutorActivate" value="false" />
53.         <property name="startEngineEventListeners">
54.             <list>
55.                 <!-- 加载自定义表单元素类型 -->
56.                 <bean class="org.openwebflow.cfg.LoadDummyFormTypes">
57.                     <property name="typeNames" value="user" />
58.                 </bean>
59.                 <!-- 自定义成员关系管理 -->
60.                 <bean class="org.openwebflow.cfg.ReplaceMembershipManager">
61.                     <property name="customMembershipManager"
    ref="myMembershipManager" />
62.                 </bean>
63.                 <!-- 自定义活动权限管理 -->
64.                 <bean class="org.openwebflow.cfg.ReplaceTaskAssignmentHandler">
65.                     <!-- 授权处理器列表，会组成一个链，越靠后优先级越高（越靠外） -->
66.                     <property name="handlers">
67.                         <list>
68.                             <!-- 自定义授权项列表 -->
69.                             <bean
70.
     class="org.openwebflow.assign.permission.ActivityPermissionAssignmentHandler">
71.                                 <property name="activityPermissionManager"
    ref="myActivityPermissionManager" />
72.                             </bean>
73.                             <!-- 允许授权代理 -->
74.                             <bean
75.
```

```
 class="org.openwebflow.assign.delegation.TaskDelagationAssignmentHandler">
                                    <property name="delegationManager"
ref="myDelegationManager" />
                                    <property name="membershipManager"
ref="myMembershipManager" />
                                    <property name="hideDelegated" value="false" />
                                </bean>
                            </list>
                        </property>
                    </bean>
                    <!-- 自动导入流程模型 -->
                    <bean class="org.openwebflow.cfg.ImportDefinedProcessModels">
                        <property name="modelDir" value="${model.dir}" />
                    </bean>
                    <!-- 启动催办管理器 -->
                    <bean class="org.openwebflow.cfg.StartTaskAlarmService">
                        <property name="taskAlarmService" ref="myTaskAlarmService" />
                        <property name="runOnStartup" value="false" />
                    </bean>
                    <!-- 加载自定义activity -->
                    <bean class="org.openwebflow.cfg.LoadRuntimeActivityDefinitions">
                        <property name="activityDefinitionManager"
ref="myActivityDefinitionManager" />
                    </bean>
                </list>
            </property>
    </bean>

    <!-- processEngine -->
    <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
        <property name="processEngineConfiguration" ref="processEngineConfiguration" />
    </bean>

    <!-- 工作流流转服务对象工厂 -->
    <bean class="org.openwebflow.ctrl.impl.DefaultTaskFlowControlServiceFactory" />

    <!-- processEngineTool -->
    <bean id="processEngineTool" class="org.openwebflow.util.ProcessEngineTool" />

    <bean id="repositoryService" factory-bean="processEngine"
        factory-method="getRepositoryService" />
    <bean id="runtimeService" factory-bean="processEngine"
        factory-method="getRuntimeService" />
    <bean id="taskService" factory-bean="processEngine"
```

```
116.          factory-method="getTaskService" />
117.      <bean id="historyService" factory-bean="processEngine"
118.          factory-method="getHistoryService" />
119.      <bean id="managementService" factory-bean="processEngine"
120.          factory-method="getManagementService" />
```

其中 processEngineConfiguration 是增强型的工作流引擎配置对象，可以设置自定义用户群组成员关系管理策略、自定义活动权限管理策略等。

完整的例子参见：https://github.com/bluejoe2008/openwebflow/blob/master/openwebflow-test/src/test/resources/activiti.cfg.core.xml

### 2.2.3 activiti.cfg.mem.xml 配置

在 activiti.core.xml 中会用到一些 manager，activiti.mem.xml 定义基于内存的 manager 实现，默认的 activiti.mem.xml 内容如下：

```
1.    <!-- 自定义成员关系管理 -->
2.    <bean id="myMembershipManager"
      class="org.openwebflow.mgr.mem.InMemoryMembershipManager" />
3.    <bean id="myUserDetailsManager"
      class="org.openwebflow.mgr.mem.InMemoryUserDetailsManager" />
4.
5.    <!-- 自定义的活动权限表管理 -->
6.    <bean id="myActivityPermissionManager"
7.        class="org.openwebflow.mgr.mem.InMemoryActivityPermissionManager" />
8.
9.    <!-- 代理关系管理 -->
10.   <bean id="myDelegationManager"
      class="org.openwebflow.mgr.mem.InMemoryDelegationManager" />
11.
12.   <!-- 自定义的动态自定义活动管理 -->
13.   <bean id="myActivityDefinitionManager"
14.       class="org.openwebflow.mgr.mem.InMemoryRuntimeActivityDefinitionManager" />
15.
16.   <bean id="myTaskNotificationManager"
      class="org.openwebflow.mgr.mem.InMemoryTaskNotificationManager" />
```

这里面定义了 6 个 manager：

| Manager 类别 | 含义 |
| --- | --- |
| myMembershipManager | 自定义成员关系管理 |
| myUserDetailsManager | 自定义用户详细信息管理 |
| myActivityPermissionManager | 自定义的活动权限表管理 |
| myDelegationManager | 代理关系管理 |
| myActivityDefinitionManager | 自定义的动态自定义活动管理 |
| myTaskNotificationManager | 任务通知信息管理 |

完整的例子参见：https://github.com/bluejoe2008/openwebflow/blob/master/openwebflow-test/src/test/resources/activiti.cfg.mem.xml

与 activiti.cfg.core.xml 类似的可替代文件为 activiti.cfg.sql.hibernate.xml 和 activiti.cfg.sql.mybatis.xml。

### 2.2.4 activiti.cfg.sql.hibernate.xml 配置

activiti.sql.hibernate.xml 提供基于 SQL 的 manager 实现，采用的 ORM 框架为 Hibernate 4。

各 manager 的定义如下：

```
1.     <!-- 代理记录管理 -->
2.     <bean id="myDelegationManager"
3.         class="org.openwebflow.mgr.hibernate.service.SqlDelegationManager" />
4.     <!-- 自定义成员关系管理 -->
5.     <bean id="myMembershipManager"
6.         class="org.openwebflow.mgr.hibernate.service.SqlMembershipManager" />
7.     <!-- 自定义用户表 -->
8.     <bean id="myUserDetailsManager"
9.         class="org.openwebflow.mgr.hibernate.service.SqlUserDetailsManager" />
10.    <!-- 自定义的活动权限表管理 -->
11.    <bean id="myActivityPermissionManager"
12.        class="org.openwebflow.mgr.hibernate.service.SqlActivityPermissionManager" />
13.    <!-- 自定义的动态自定义活动管理 -->
14.    <bean id="myActivityDefinitionManager"
15.        class="org.openwebflow.mgr.hibernate.service.SqlRuntimeActivityDefinitionManager"
       />
16.    <bean id="myTaskNotificationManager"
```

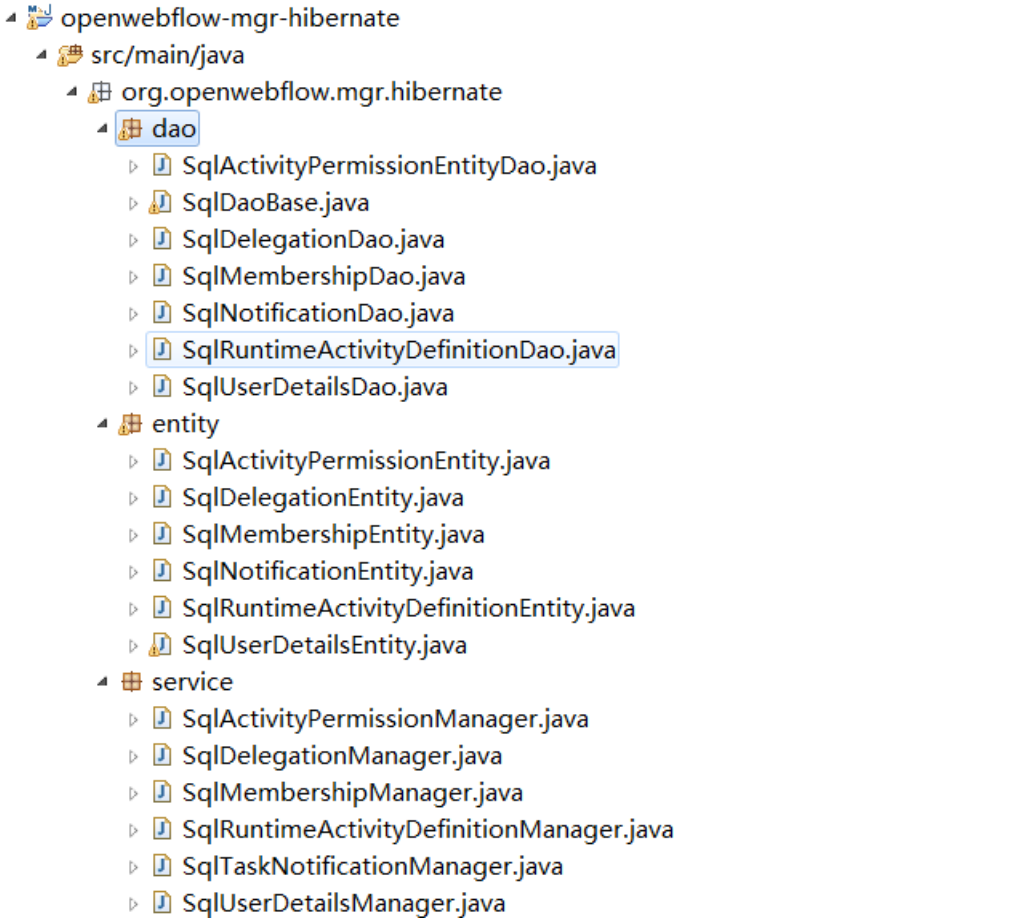
```
17.         class="org.openwebflow.mgr.hibernate.service.SqlTaskNotificationManager" />
```

此外，还需要定义数据源、Hibernate Session 工厂，以及事务。

```
1.     <context:component-scan base-package="org.openwebflow.mgr.hibernate.dao" />
2.
3.     <!-- 数据库脚本见openwebflow.sql -->
4.     <bean id="owfDataSource" class="org.apache.commons.dbcp.BasicDataSource"
5.         destroy-method="close">
6.         <property name="driverClassName" value="${owfdb.driver}" />
7.         <property name="url" value="${owfdb.url}" />
8.         <property name="username" value="${owfdb.username}" />
9.         <property name="password" value="${owfdb.password}" />
10.        <property name="initialSize" value="20" />
11.        <property name="maxActive" value="50" />
12.        <property name="maxIdle" value="20" />
13.        <property name="minIdle" value="10" />
14.    </bean>
15.
16.    <!-- 配置SessionFactory -->
17.    <bean id="sessionFactory"
18.        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
19.        <property name="dataSource" ref="owfDataSource" />
20.        <property name="hibernateProperties">
21.            <props>
22.                <prop key="hibernate.dialect">${hibernate.dialect}</prop>
23.                <!-- <prop
    key="hibernate.dialect">org.hibernate.dialect.SQLServer2008Dialect</prop> -->
24.                <!-- 服务启动通过实体创建数据库表信息 -->
25.                <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
26.                <prop key="hibernate.show_sql">true</prop>
27.                <prop key="hibernate.format_sql">true</prop>
28.                <prop key="hibernate.jdbc.batch_size">20</prop>
29.                <prop key="hibernate.connection.release_mode">auto</prop>
30.                <prop key="hibernate.autoReconnect">false</prop>
31.                <prop key="hibernate.connection.autocommit">true</prop>
32.                <prop key="hibernate.temp.use_jdbc_metadata_defaults">false</prop>
33.                <prop key="hibernate.jdbc.use_streams_for_binary">true</prop>
34.                <prop
    key="hibernate.current_session_context_class">org.springframework.orm.hibernate4.SpringSes
    sionContext
35.                </prop>
36.                <!--解决weblogic无法使用hql的问题 -->
37.                <prop key="hibernate.cglib.use_reflection_optimizer">true</prop>
```

```
38.             </props>
39.         </property>
40.
41.         <!-- 自动扫描备注解的实体 -->
42.         <property name="packagesToScan">
43.             <list>
44.                 <value>org.openwebflow.mgr.hibernate.entity</value>
45.             </list>
46.         </property>
47.     </bean>
48.
49.     <!-- 配置一个事务管理器 -->
50.     <bean id="transactionManager"
51.         class="org.springframework.orm.hibernate4.HibernateTransactionManager">
52.         <property name="sessionFactory" ref="sessionFactory" />
53.     </bean>
54.
55.     <tx:annotation-driven transaction-manager="transactionManager" />
```

采用 Hibernate ORM，openwebflow-mgr-hibernate 的代码结构如下：



其中，DAO 类、实体类、服务类分别存放在 dao、entity、service 包下面。service 类的事务声明以及 entity 的映射皆采取注解方式。

完整的例子参见：https://github.com/bluejoe2008/openwebflow/blob/master/openwebflow-test/src/test/resources/activiti.cfg.sql.hibernate.xml

### 2.2.5 activiti.cfg.sql.mybatis.xml 配置

activiti.sql.mybatis.xml 提供基于 SQL 的 manager 实现，采用的 ORM 框架为 mybatis 3。

各 manager 的定义如下：

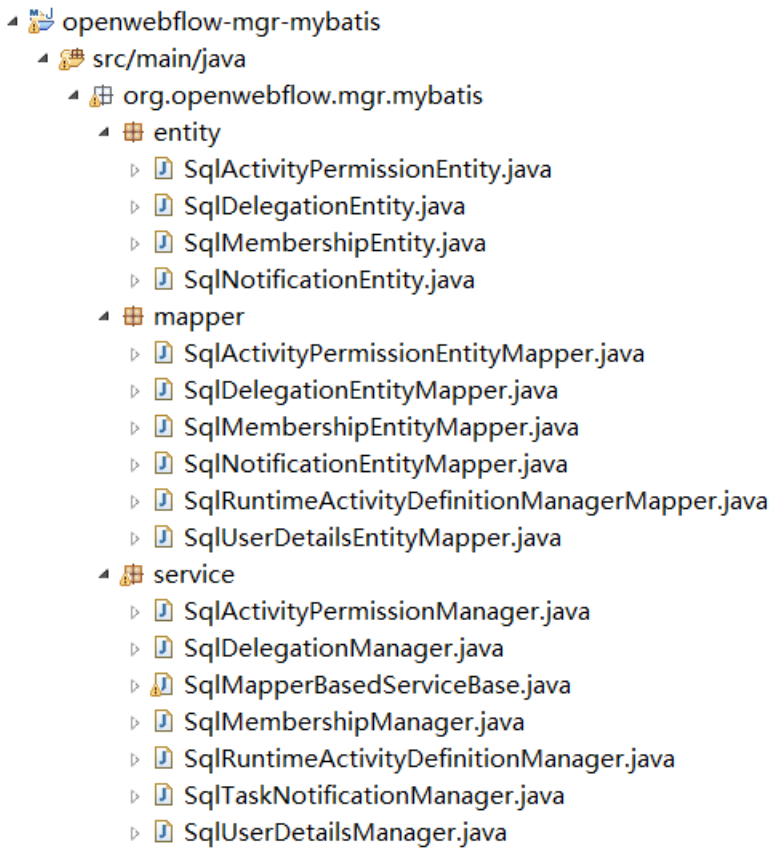
```
1.  <!-- 代理记录管理 -->
2.  <bean id="myDelegationManager"
3.      class="org.openwebflow.mgr.mybatis.service.SqlDelegationManager" />
4.  <!-- 自定义成员关系管理 -->
5.  <bean id="myMembershipManager"
6.      class="org.openwebflow.mgr.mybatis.service.SqlMembershipManager" />
7.  <!-- 自定义用户表 -->
8.  <bean id="myUserDetailsManager"
9.      class="org.openwebflow.mgr.mybatis.service.SqlUserDetailsManager" />
10. <!-- 自定义的活动权限表管理 -->
11. <bean id="myActivityPermissionManager"
12.     class="org.openwebflow.mgr.mybatis.service.SqlActivityPermissionManager" />
13. <!-- 自定义的动态自定义活动管理 -->
14. <bean id="myActivityDefinitionManager"
15.     class="org.openwebflow.mgr.mybatis.service.SqlRuntimeActivityDefinitionManager" />
16. <bean id="myTaskNotificationManager"
17.     class="org.openwebflow.mgr.mybatis.service.SqlTaskNotificationManager" />
```

此外，还需要定义数据源、SqlSessionFactory，以及事务。

```
1.  <!-- 数据库脚本见openwebflow.sql -->
2.  <bean id="owfDataSource" class="org.apache.commons.dbcp.BasicDataSource"
3.      destroy-method="close">
4.      <property name="driverClassName" value="${owfdb.driver}" />
5.      <property name="url" value="${owfdb.url}" />
6.      <property name="username" value="${owfdb.username}" />
7.      <property name="password" value="${owfdb.password}" />
8.      <property name="initialSize" value="20" />
```

```
9.          <property name="maxActive" value="50" />
10.         <property name="maxIdle" value="20" />
11.         <property name="minIdle" value="10" />
12.     </bean>
13.
14.     <!-- 创建SqlSessionFactory，同时指定数据源-->
15.     <bean id="owlSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
16.         <property name="dataSource" ref="owfDataSource" />
17.     </bean>
18.
19.     <!-- 配置一个事务管理器 -->
20.     <bean id="transactionManager"
21.         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
22.         <property name="dataSource" ref="owfDataSource" />
23.     </bean>
24.
25.     <tx:annotation-driven transaction-manager="transactionManager" />
```

采用 Mybatis ORM，openwebflow-mgr-mybatis 的代码结构如下：




其中，Mapper 接口、实体类、服务类分别存放在 mapper、entity、service 包下面。service 类的事务声明以及 Mapper 的映射皆采取注解方式。

完整的例子参见：

https://github.com/bluejoe2008/openwebflow/blob/master/openwebflow-test/src/test/resources/activiti.cfg.sql.mybatis.xml

## 2.3 数据库设计

首先，Activiti 引擎本身需要用到一系列的数据表，设置好数据源后，Activiti 会自动生成这些表。

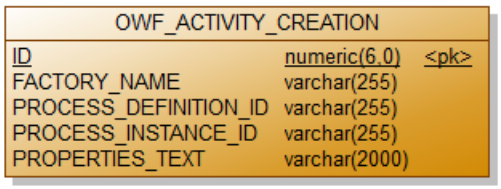
Activiti 的表都以 ACT_开头，第二部分是表示表的用途的两个字母标识。用途也和服务的 API 对应。

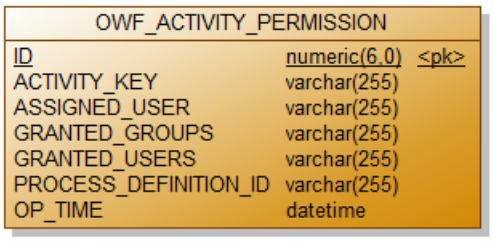
- ACT_RE_*: 'RE'表示 repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。
- ACT_RU_*: 'RU'表示 runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti 只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。
- ACT_ID_*: 'ID'表示 identity。 这些表包含身份信息，比如用户，组等等。
- ACT_HI_*: 'HI'表示 history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。
- ACT_GE_*: 通用数据， 用于不同场景下。

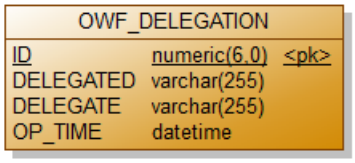
OpenWebFlow 为一系列 manager 提供了基于数据库的实现，需要用到一些数据表，对应的建库脚本参见 https://github.com/bluejoe2008/openwebflow/tree/master/doc 目录下的：

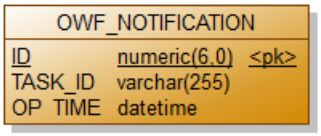
- openwebflow-mysql4.sql：MySQL4 脚本
- openwebflow-mysql5.sql：MySQL5 脚本
- openwebflow-sqlserver2008.sql：SQLServer2008 脚本
- openwebflow-oracle10g.sql：Oracle 脚本

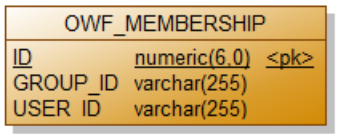
其中共定义了 6 张表格：

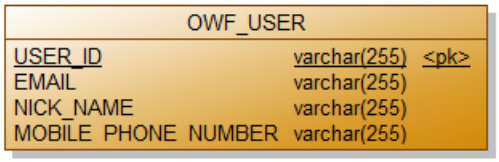



- OWF_ACTIVITY_CREATION：用以存储自定义的活动定义信息
- OWF_ACTIVITY_PERMISSION：用以存储自定义的活动权限信息
- OWF_DELEGATION：用以存储用户代理信息
- OWF_NOTIFICATION：用以存储催办通知记录
- OWF_MEMBERSHIP：用以存储用户组成员关系
- OWF_USER：用以存储用户信息

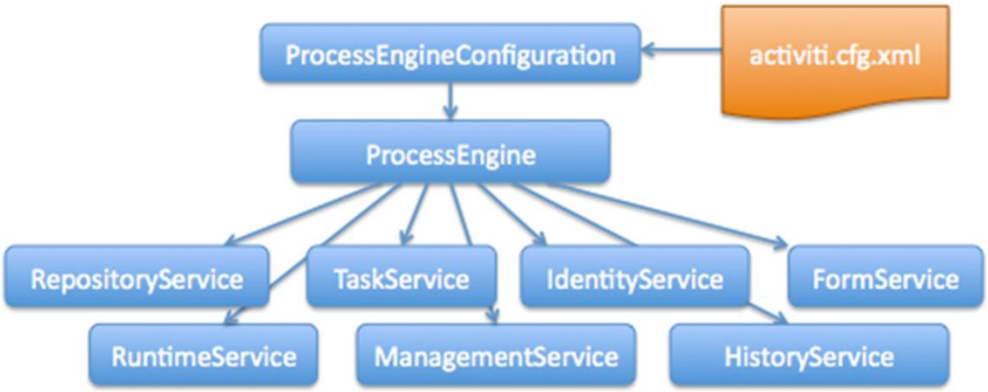
注意：OWF_MEMBERSHIP 和 OWF_USER 仅为测试使用，建议用户使用自己的数据表（OpenWebFlow 本身努力的一个方向就是将用户及成员关系管理与工作流引擎剥离开），并包装自己的 Manager。

## 2.4 使用 ApplicationContext 定义的 bean

采用 Spring IoC 框架加载完 XML 配置文件之后，ApplicationContext 中会包含如下变量，可供客户程序使用：

- processEngine：工作流引擎对象，标准的 Activiti 对象
- processEngineTool：针对 processEngine 提供了一些工具方法
- defaultTaskFlowControlServiceFactory：任务流控制器的工厂对象
- repositoryService：提供了管理和控制发布包和流程定义的操作
- RuntimeService：负责启动一个流程定义的新实例
- TaskService：与任务相关相关的操作

- IdentityService：管理（创建，更新，删除，查询...）群组和用户
- FormService：提供了启动表单和任务表单两个概念
- HistoryService：提供了 Activiti 引擎手机的所有历史数据
- ManagementService：可以查询数据库的表和表的元数据




如下是使用 OpenWebFlow 的示例代码，可以看出来与 Activiti 的用法完全一致：

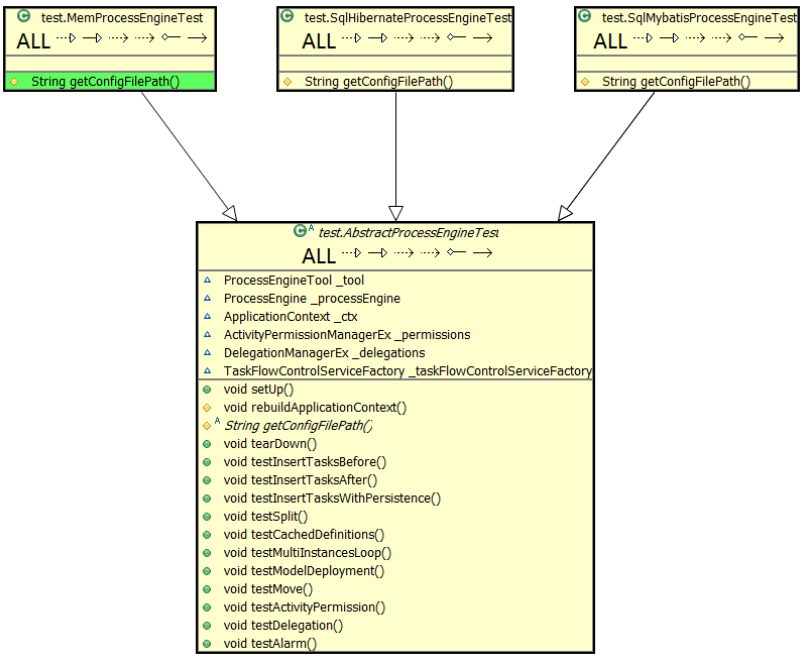
```
1. ApplicationContext ctx = new ClassPathXmlApplicationContext("classpath:activiti.cfg.mem.xml");
2. ProcessEngineTool tool = ctx.getBean(ProcessEngineTool.class);
3. ProcessEngine processEngine = tool.getProcessEngine();
4. // 启动流程实例
5. ProcessInstance instance = processEngine.getRuntimeService().startProcessInstanceByKey("test1");
6. TaskService taskService = processEngine.getTaskService();
7. //会自动跳转到第一个task
8. //management可以访问该task
9. Assert.assertEquals(1, taskService.createTaskQuery().taskCandidateGroup("management").count());
```

## 2.5 运行测试用例

在 openwebflow-test 项目源代码中，用户可以找到一组测试用例来对工作流引擎的功能进行测试，它们分别是：

- MemProcessEngineTest：基于内存的 manager 测试
- SqlHibernateProcessEngineTest：基于 hibernate ORM 的 manager 测试
- SqlMybatisProcessEngineTest：基于 mybatis ORM 的 manager 测试

以上 3 个测试类都继承于 AbstractProcessEngineTest：

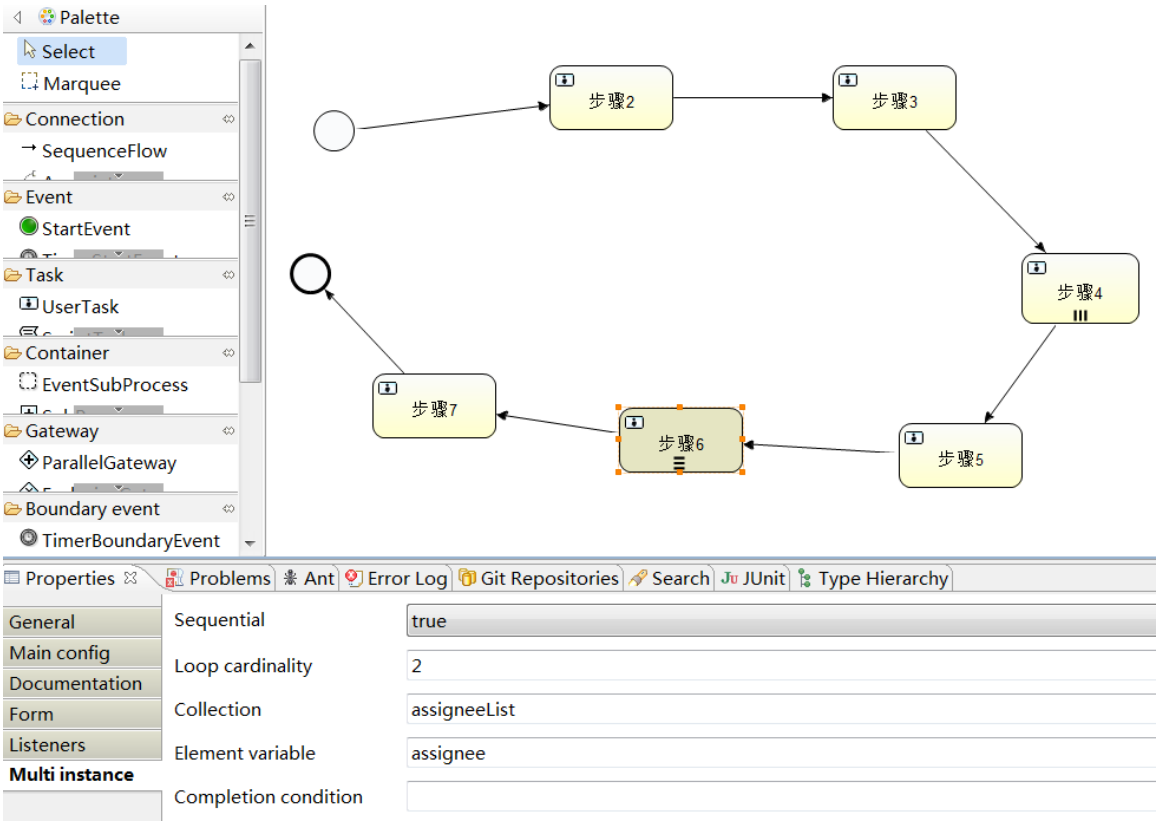



AbstractProcessEngineTest 提供了测试方法：

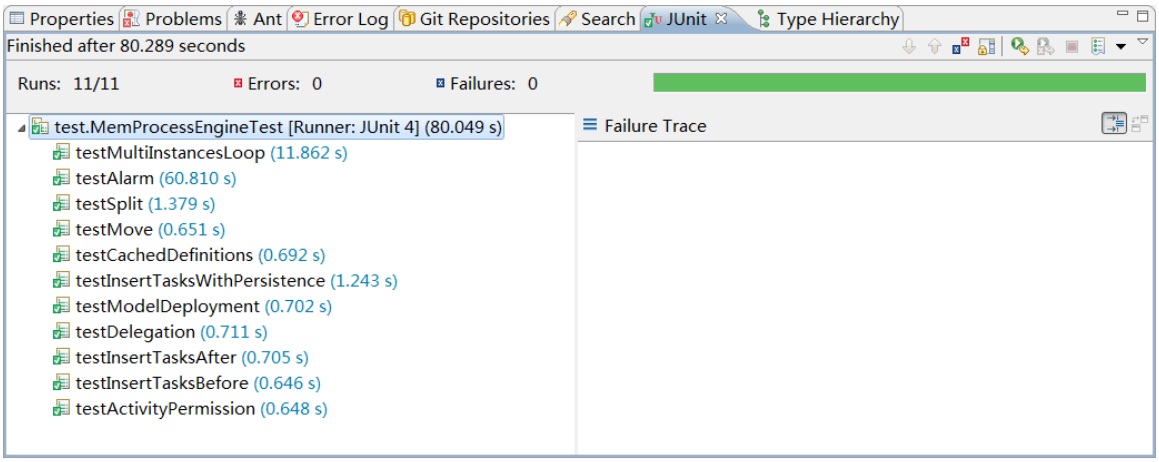
| 方法摘要 | |
|---|---|
| void | testActivityPermission()<br>测试流程动态授权 |
| void | testAlarm()<br>测试催办功能 |
| void | testCachedDefinitions()<br>测试 TaskDefinition |
| void | testDelegation()<br>测试代理功能 |
| void | testInsertTasksAfter()<br>测试后加签 |
| void | testInsertTasksBefore()<br>测试前加签 |
| void | testInsertTasksWithPersistence()<br>测试加签功能的持久化 |
| void | testModelDeployment()<br>测试流程模型部署 |

| | |
|---|---|
| void | testMove()<br>测试自由跳转 |
| void | testMultiInstancesLoop()<br>测试多实例节点 |
| void | testSplit()<br>测试测试节点分裂 |

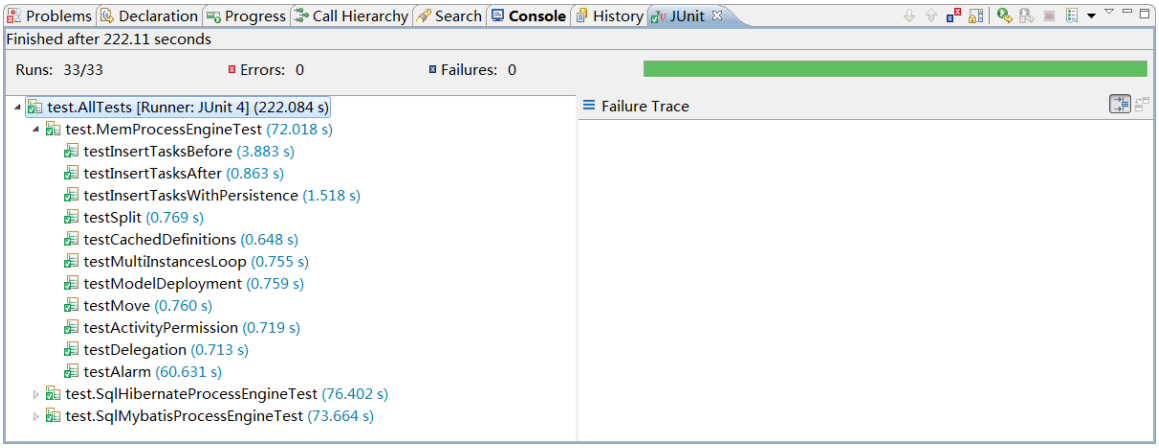
其中为了配合测试设计了一个复杂的流程（models/test2.bpmn）如图所示：




选择指定的测试单元（如：MemProcessEngineTest），以“JUnit 测试”的方式运行（Run As…），即可观察到测试结果：



也可以选择运行测试套件 AllTests：




# 3. 熟悉 OpenWebFlow 代码

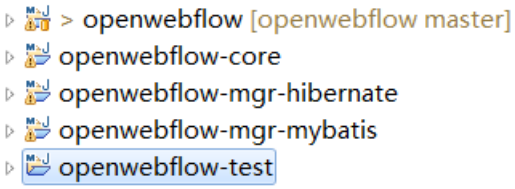
## 3.1 下载源码

用户可以下载 OpenWebFlow 的 zip 包，下载地址为：https://github.com/bluejoe2008/openwebflow/archive/master.zip

也可以通过 git 方式获取到最新源码，git 资源库地址为：https://github.com/bluejoe2008/openwebflow.git

## 3.2 代码结构

OpenWebFlow 源码包含 5 个 maven 工程，其中 openwebflow 工程是父工程，它声明了包含 openwebflow-core 、 openwebflow-mgr-hibernate 、 openwebflow-mgr-mybatis、openwebflow-test 等 4 个 model。




openwebflow-core：核心工程，包含 OpenWebFlow 扩展引擎的所有核心内容、以及基于内存的 manager 实现。

openwebflow-mgr-hibernate：依赖于 openwebflow-core，提供了基于数据库的 manager 实现，ORM 框架采用 Hibernate。

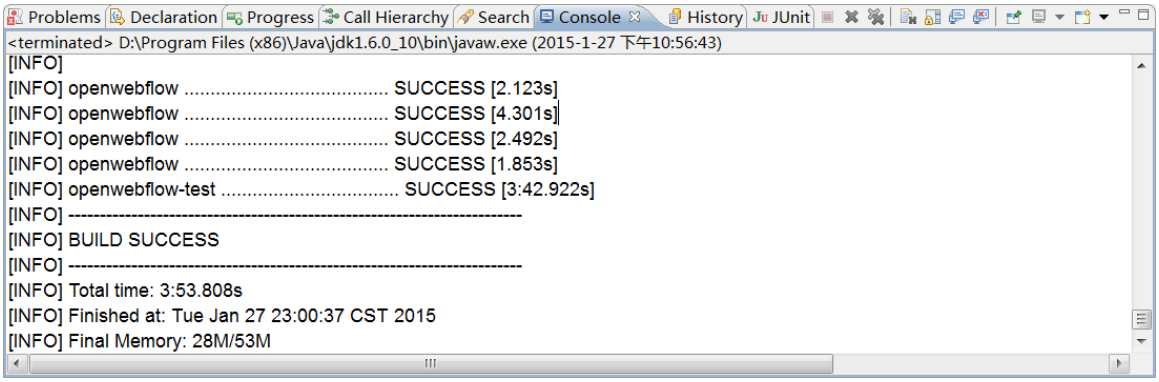
openwebflow-mgr-mybatis：依赖于 openwebflow-core，提供了基于数据库的

manager 实现，ORM 框架采用 MyBatis。

openwebflow-test：依赖于以上项目，提供了测试用例，包括配置文件、测试类等。

# 3.3 build 项目

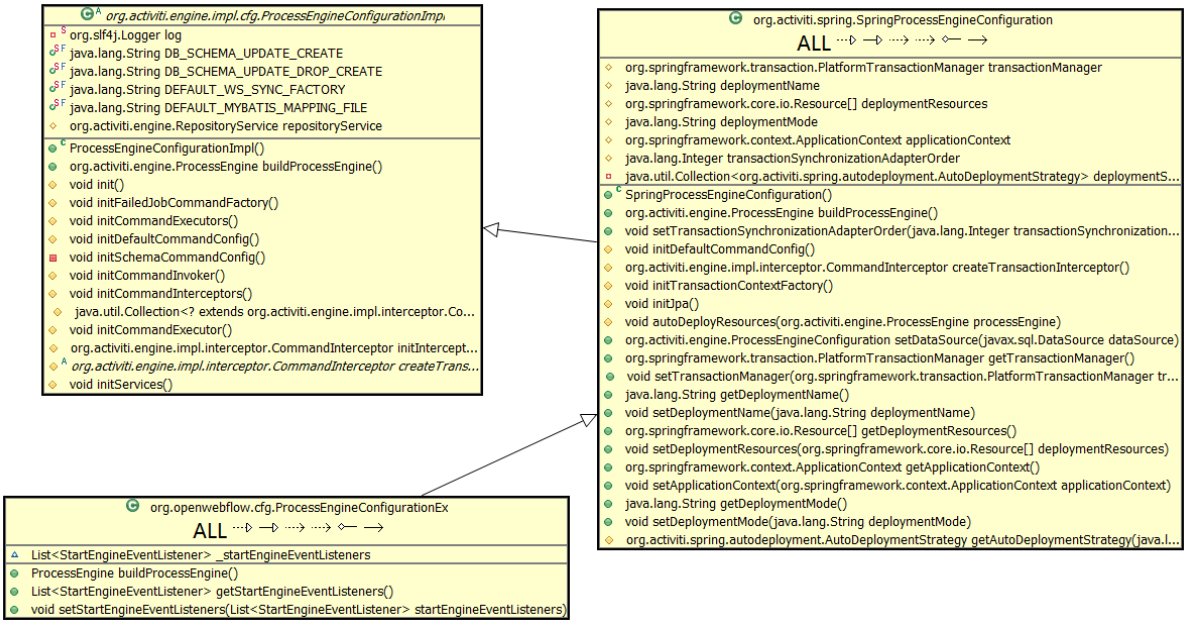
获取到的项目源码可以采用 Maven 完成 build 和 install，如下为 build 截图：




# 3.4 核心对象

## 3.4.1 ProcessEngineConfigurationEx

ProcessEngineConfigurationEx 是针对 Activiti 提供的 ProcessEngineConfiguration 类的派生类：




二者大部分参数完全一致，唯一不同的是，ProcessEngineConfigurationEx 提供了一个属性：startEngineEventListeners。

startEngineEventListeners 用以定义工作流引擎启动的时候需要同时启动的其它任务，startEngineEventListeners 是个 List，因此可以随意增加新的任务，默认的 core.xml 中会加载如下任务:LoadDummyFormTypes、ReplaceMembershipManager、ReplaceTaskAssignmentHandler 、 ImportDefinedProcessModels 、StartTaskAlarmService、LoadRuntimeActivityDefinitions。各任务及类属性列表如下：

| 任务类 | 用途 | 属性名 | 属性含义 |
|---|---|---|---|
| LoadDummyFormTypes | 加载一些无用的 Form 类型，用以屏蔽一些自定义 Form 带来的错误 | typeNames | 需要屏蔽的 Form 类型名，以;分隔，如：user |
| ReplaceMembershipManager | 直接接管用户组成员关系 | customMembershipManager | 指定客户程序自定义的管理器 |
| ReplaceTaskAssignmentHandler | 接管 Activiti 的用户权限管理，如果你想实现动态的节点权限分配，那必须要打开它。 | handlers | 定义一个授权处理器列表，值类型为 List，运行时刻各授权处理器列表会组成一个链，越靠后优先级越高（越靠外） |
| ImportDefinedProcessModels | 自动从指定目录导入 BPMN 模型 | modelDir | 用以指定模型的路径，可以是 classpath:等路径 |
| StartTaskAlarmService | 启动任务催办服务 | taskAlarmService | 设置催办服务对象 |
| | | runOnStartup | 是否一开始就启动（默认为 true） |
| LoadRuntimeActivityDefinitions | 加载运行时的节点定义，主要用来支持在运行时刻定义新的节点 | activityDefinitionManager | 指定节点定义管理器 |

## 3.4.2 ProcessEngineTool

ProcessEngineTool 提供了一些工具方法，这些方法的功能一般很难通过 ProcessEngine 直接拿到：

## 方法摘要

| | |
|---|---|
| org.activiti.engine.repository.Model | createNewModel(java.lang.String name, java.lang.String description)<br>创建一个空白的 Model 对象 |
| org.activiti.engine.repository.Deployment | deployModel(java.lang.String modelId)<br>部署一个已注册的 model |
| org.activiti.engine.impl.pvm.process.ActivityImpl | getActivity(java.lang.String processDefId, java.lang.String activityId)<br>获取指定名字的活动 |
| java.util.Map<java.lang.String,java.lang.Object> | getHistoricProcessVariables(java.lang.String processId)<br>获取指定历史流程的变量列表 |
| org.activiti.engine.impl.persistence.entity.ProcessDefinitionEntity | getProcessDefinition(java.lang.String processDefId)<br>获取指定 ID 的流程定义 |
| org.activiti.engine.ProcessEngine | getProcessEngine() |
| void | grantPermission(org.activiti.engine.impl.pvm.process.ActivityImpl activity, java.lang.String assigneeExpression, java.lang.String candidateGroupIdExpressions, java.lang.String candidateUserIdExpressions)<br>设置指定活动的用户权限，包括钦定用户、候选用户、候选组 |
| void | grantPermission(java.lang.String processDefId, java.lang.String activityId, |

| | |
|---|---|
| | java.lang.String assigneeExpression, java.lang.String candidateGroupIdExpressions, java.lang.String candidateUserIdExpressions)<br>设置指定活动的用户权限，包括钦定用户、候选用户、候选组 |
| void | setProcessEngine(org.activiti.engine.ProcessEngine processEngine) |

### 3.4.3 各种 Utils

OpenWebFlow 提供了一些常见的工具类，如下所示：

| 类摘要 | |
|---|---|
| CloneUtils | 实现对象的克隆功能 |
| ExpressionUtils | 实现常见类型的 expression 的包装和转换 |
| IdentityUtils | 实现用户、成员关系等相关操作 |
| ModelUtils | 包装了对 BPMN 模型的部署、注册等功能 |
| ProcessDefinitionUtils | 流程定义相关操作的封装 |

### 3.4.4 TaskFlowControlService

TaskFlowControlService 用以实现流程的自由控制，它提供的方法如下：

| 方法摘要 | |
|---|---|
| org.activiti.engine.impl.pvm.process.ActivityImpl[] | insertTasksAfter(java.lang.String targetTaskDefinitionKey, java.lang.String... assignees)<br>后加签 |
| org.activiti.engine.impl.pvm.process.ActivityImpl[] | insertTasksBefore(java.lang.String targetTaskDefinitionKey, java.lang.String... assignees) |

| | |
|---|---|
| | 前加签 |
| void | moveBack()<br>后退一步 |
| void | moveBack(org.activiti.engine.impl.persistence.entity.TaskEntity currentTaskEntity)<br>后退至指定活动 |
| void | moveForward()<br>前进一步 |
| void | moveForward(org.activiti.engine.impl.persistence.entity.TaskEntity currentTaskEntity)<br>前进至指定活动 |
| void | moveTo(java.lang.String targetTaskDefinitionKey)<br>跳转（包括回退和向前）至指定活动节点 |
| void | moveTo(java.lang.String currentTaskId, java.lang.String targetTaskDefinitionKey)<br>跳转（包括回退和向前）至指定活动节点 |
| void | moveTo(org.activiti.engine.impl.persistence.entity.TaskEntity currentTaskEntity, java.lang.String targetTaskDefinitionKey)<br>跳转（包括回退和向前）至指定活动节点 |
| org.activiti.engine.impl.pvm.process.ActivityImpl | split(java.lang.String targetTaskDefinitionKey, boolean isSequential, java.lang.String... assignees)<br>分裂某节点为多实例节点 |
| org.activiti.engine.impl.pvm.process.ActivityImpl | split(java.lang.String targetTaskDefinitionKey, java.lang.String... assignee)<br>分裂某节点为多实例节点 |

TaskFlowControlService 需要一个 TaskFlowControlServiceFactory 来创建，

可以从 applicationcontext 中获取到该工厂对象。

# 4. 核心功能的设计与使用

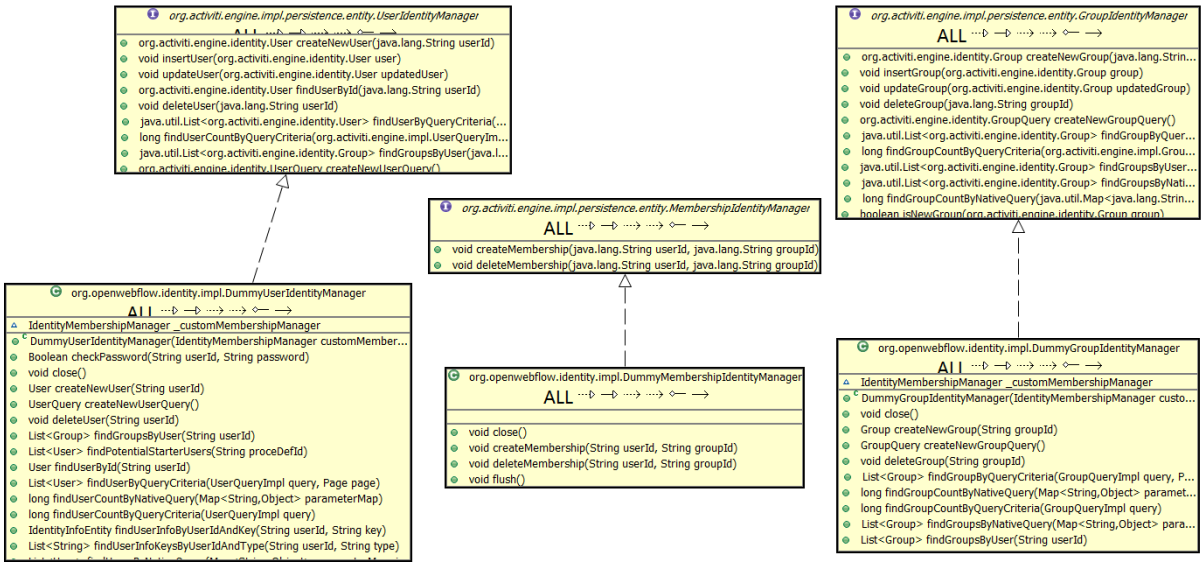
## 4.1 用户管理托管

### 4.1.1 设计方案

OpenWebFlow 通过 ReplaceMembershipManager 的 beforeStartEngine()事件，设置 ProcessEngineConfigurationImpl 的 sessionFactories，将自定义的用户管理器、群组管理器、用户成员关系管理器注射进去：

```
1.          sessionFactories.add(new SessionedEntityManagerFactory(UserIdentityManager.class,
    new DummyUserIdentityManager(
2.                  _customMembershipManager)));
3.          sessionFactories.add(new SessionedEntityManagerFactory(GroupIdentityManager.class,
4.                  new DummyGroupIdentityManager(_customMembershipManager)));
5.          sessionFactories.add(new
    SessionedEntityManagerFactory(MembershipIdentityManager.class,
6.                  new DummyMembershipIdentityManager()));
```

DummyUserIdentityManager 、 DummyGroupIdentityManager 和 DummyMembershipIdentityManager 是三个适配器，实现用户自定义 Manager 到 Activiti 所需的 UserIdentityManager 、 GroupIdentityManager 和 MembershipIdentityManager 的适配。



### 4.1.2 使用方法

如果需要挂接应用的用户成员关系管理框架，用户需要提供一个 IdentityMembershipManager 实现类，替换掉配置文件中的 membershipManager 即可。

如果需要挂接应用的用户详细信息管理框架，用户提供一个 UserDetailsManager 实现类，替换掉配置文件中的 userDetailsManager 即可。

参见“5.5 用户详细信息管理”“5.6 用户组成员关系管理”。

## 4.2 任务权限框架托管

### 4.2.1 设计方案

OpenWebFlow 的任务权限框架托管的核心思想在于 org.openwebflow.assign.TaskAssignmentHandler：

```
1.  public interface TaskAssignmentHandler
2.  {
3.    void handleAssignment(TaskAssignmentHandlerChain chain, TaskEntity task,
      ActivityExecution execution);
4.  }
```

TaskAssignmentHandler 配合自定义的 UserTaskActivityBehavior，用以篡改 Activiti 的任务权限分配方法：

```
1.  public class MyUserTaskActivityBehavior extends UserTaskActivityBehavior
2.  {
3.    List<TaskAssignmentHandler> _handlers;
4.
5.    public MyUserTaskActivityBehavior(List<TaskAssignmentHandler> handlers,
      TaskDefinition taskDefinition)
6.    {
7.        super(taskDefinition);
8.        _handlers = handlers;
9.    }
10.
11.   protected TaskAssignmentHandlerChainImpl createHandlerChain()
```

```
12.     {
13.         TaskAssignmentHandlerChainImpl handlerChain = new
        TaskAssignmentHandlerChainImpl();
14.         final MyUserTaskActivityBehavior myUserTaskActivityBehavior = this;
15.         handlerChain.addHandler(new TaskAssignmentHandler()
16.         {
17.             @Override
18.             public void handleAssignment(TaskAssignmentHandlerChain chain, TaskEntity
        task, ActivityExecution execution)
19.             {
20.                 myUserTaskActivityBehavior.superHandleAssignments(task, execution);
21.             }
22.         });
23.
24.         handlerChain.addHandlers(_handlers);
25.         return handlerChain;
26.     }
27.
28.     @Override
29.     protected void handleAssignments(TaskEntity task, ActivityExecution execution)
30.     {
31.         createHandlerChain().resume(task, execution);
32.     }
33.
34.     protected void superHandleAssignments(TaskEntity task, ActivityExecution execution)
35.     {
36.         super.handleAssignments(task, execution);
37.     }
38. }
```

`TaskAssignmentHandlerChainImpl` 将一系列的 `TaskAssignmentHandler` 从内到外组成一个处理链，然后再委托给 `MyUserTaskActivityBehavior` 来执行。查看配置文件可以观察到这个链的结构：

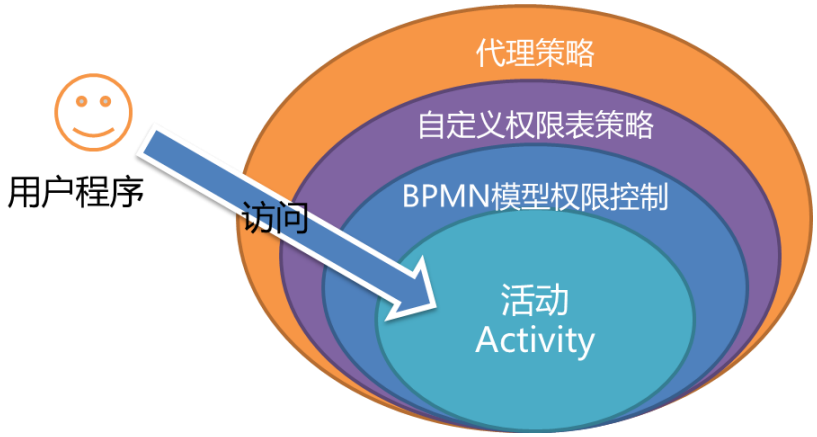
```
1.<!-- 授权处理器列表，会组成一个链，越靠后优先级越高（越靠外） -->
2.<property name="handlers">
3.  <list>
4.      <!-- 自定义授权项列表 -->
5.      <bean
6.
    class="org.openwebflow.assign.permission.ActivityPermissionAssignmentHandler">
7.          <property name="activityPermissionManager" ref="myActivityPermissionManager"
```

```
    />
8.      </bean>
9.      <!-- 允许授权代理 -->
10.     <bean
11.         class="org.openwebflow.assign.delegation.TaskDelagationAssignmentHandler">
12.         <property name="delegationManager" ref="myDelegationManager" />
13.         <property name="membershipManager" ref="myMembershipManager" />
14.         <property name="hideDelegated" value="false" />
15.     </bean>
16. </list>
17.</property>
```

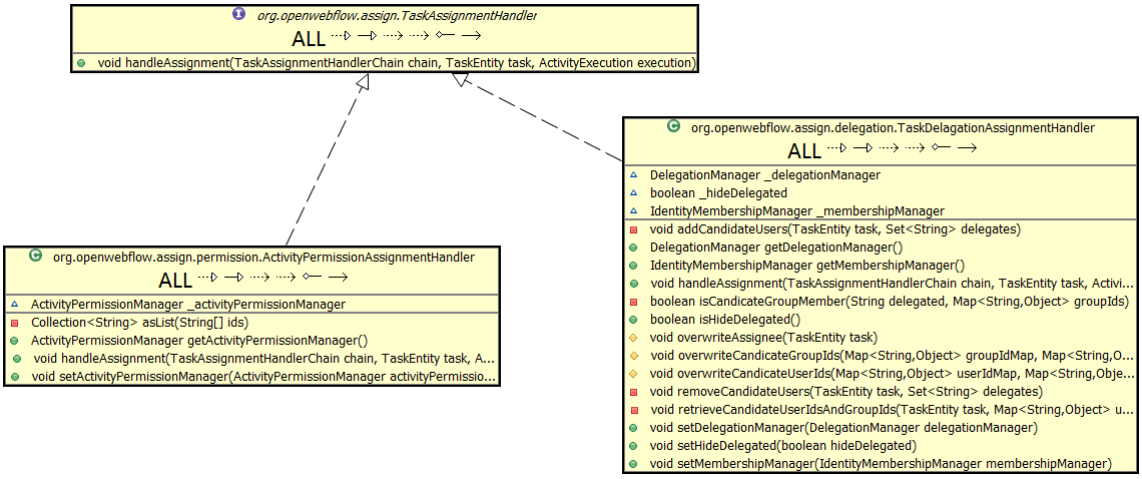
可以看出链列表里的内容依次为：

(BPMN 静态模型权限定义) → ActivityPermissionAssignmentHandler → TaskDelagationAssignmentHandler

用户程序通过 TaskAssignmentHandlerChainImpl 访问活动的顺序如下图所示：




其中，BPMN 模型权限控制由 createHandlerChain()方法体内的一个匿名类包装完成，自定义权限表策略由 ActivityPermissionAssignmentHandler 类完成，代理策略由 TaskDelagationAssignmentHandler 类完成。ActivityPermissionAssignmentHandler 和 TaskDelagationAssignmentHandler 是 TaskAssignmentHandler 的两个实现类，如下图所示。




ActivityPermissionAssignmentHandler 会实现活动权限运行时自定义的功能，相关内容参见“5.1 活动定义管理”。

TaskDelagationAssignmentHandler 会实现任务代办的功能，相关内容参见“0

欲使用任务权限框架托管功能，需要遵循如下步骤：

- 配置 myActivityPermissionManager 、 myDelegationManager 以及 myMembershipManager；
- 在 activiti.cfg.core.xml 中的 processEngineConfiguration 里打开 ReplaceTaskAssignmentHandler（默认情况下这是打开的）；

接下来，不需要任何设置，在查询任务的时候，会自动根据以上授权策略进行筛选。

任务代办”。

### 4.2.2 使用方法

欲使用任务权限框架托管功能，需要遵循如下步骤：

- 配置 myActivityPermissionManager 、 myDelegationManager 以及 myMembershipManager；
- 在 activiti.cfg.core.xml 中的 processEngineConfiguration 里打开 ReplaceTaskAssignmentHandler（默认情况下这是打开的）；

接下来，不需要任何设置，在查询任务的时候，会自动根据以上授权策略进行筛选。

## 4.3 任务代办

Activiti 本身提供了任务代办功能，TaskService 具有如下方法：

```
void delegateTask(String taskId,
```

String userId)

Delegates the task to another user. This means that the assignee is set and the delegation state is set to DelegationState.PENDING. If no owner is set on the task, the owner is set to the current assignee of the task.

注意 OpenWebFlow 关注的“任务代办”功能与上面的概念不同，它的含义是：用户 A 暂时委派用户 B 代理处理“所有可能的任务”，而不是专指“某个任务”。

### 4.3.1 设计方案

通过任务权限框架的托管，OpenWebFlow 任务代办的实现比较简便。TaskDelagationAssignmentHandler 通过 DelegationManager 和 IdentityMembershipManager 两个字段，实现了任务代理的功能。核心的实现方法如下：

```
1.  @Override
2.  public void handleAssignment(TaskAssignmentHandlerChain chain, TaskEntity task,
    ActivityExecution execution)
3.  {
4.      //先执行其它规则
5.      chain.resume(task, execution);
6.
7.      overwriteAssignee(task);
8.
9.      Map<String, Object> userIdMap = new HashMap<String, Object>();
10.     Map<String, Object> groupIdMap = new HashMap<String, Object>();
11.     retrieveCandidateUserIdsAndGroupIds(task, userIdMap, groupIdMap);
12.     Map<String, Object> newUserIdMap = new HashMap<String, Object>();
13.     Map<String, Object> removeUserIdMap = new HashMap<String, Object>();
14.
15.     //遍历所有的被代理人
16.     List<DelegationEntity> entries = _delegationManager.listDelegationEntities();
17.     overwriteCandicateUserIds(userIdMap, newUserIdMap, removeUserIdMap, entries);
18.     overwriteCandicateGroupIds(groupIdMap, newUserIdMap, entries);
19.
20.     addCandidateUsers(task, newUserIdMap.keySet());
21.     removeCandidateUsers(task, removeUserIdMap.keySet());
22. }
```

DelegationManager 的设计参见“5.4.1DelegationManager”。

### 4.3.2 使用方法

欲使用任务代办功能，只需要遵循如下步骤：

- 配置 myDelegationManager 和 myMembershipManager；
- 在业务代码中通过调用 DelegationManagerEx 的 saveDelegation()来设置一条代理规则；
- 在 activiti.cfg.core.xml 中的 processEngineConfiguration 里打开 TaskDelagationAssignmentHandler（默认情况下这是打开的）；

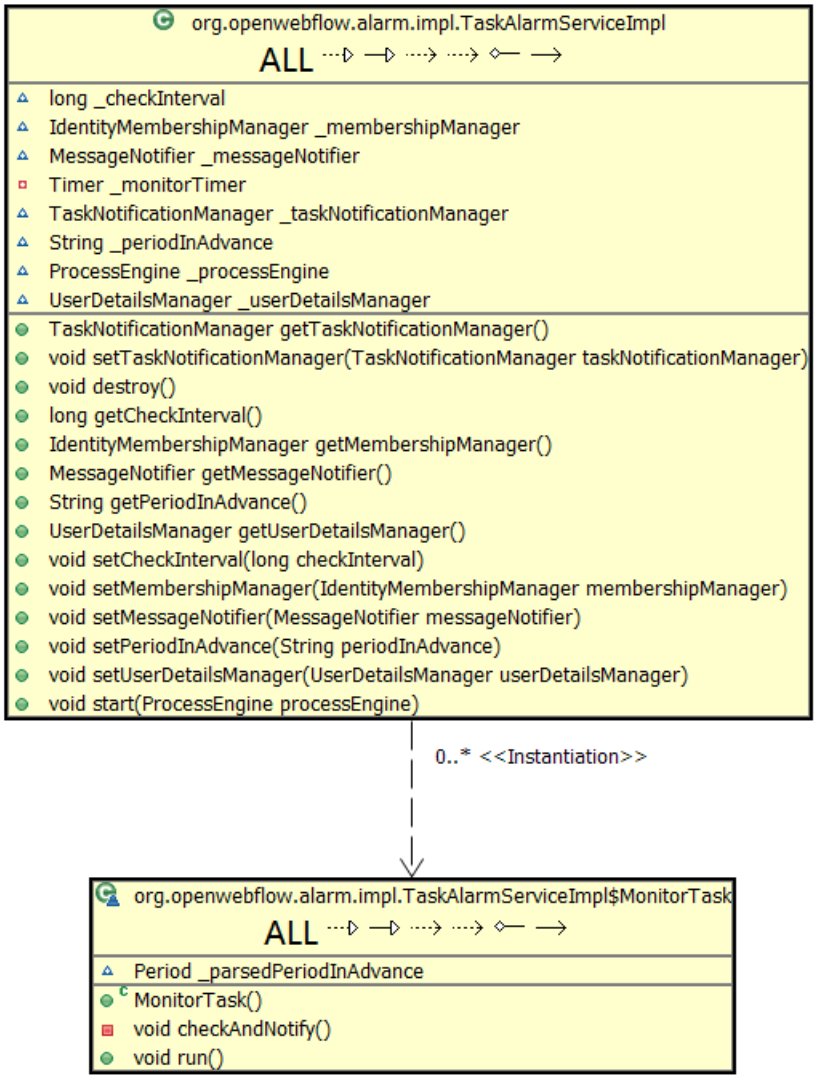
接下来，不需要任何设置，在分配任务（或者选择候选人）的时候，会自动包含被代理人。

## 4.4 任务催办

任务催办的功能与引擎相对独立，它负责针对未能及时处理的任务（根据任务的 dueDate），给出通知，包括邮件通知、短信通知等。

### 4.4.1 设计方案

OpenWebFlow 针对催办功能设计了一个 TaskAlarmService 服务，一个典型的 TaskAlarmServiceImpl 服务定义如下：




如下示出 myTaskAlarmService 的配置：

```
1.     <!-- 任务催办配置 -->
2.  <bean id="myTaskAlarmService"
   class="org.openwebflow.alarm.impl.TaskAlarmServiceImpl">
3.         <!-- 截止日期提前量 -->
4.         <property name="periodInAdvance" value="P2D" />
5.         <!-- 设置消息通知机制 -->
6.         <property name="messageNotifier">
7.             <!-- 采用邮件发送 -->
8.             <bean class="org.openwebflow.alarm.impl.MailMessageNotifier">
9.                 <property name="subjectTemplate" value="请尽快处理#{'$'}{task.name}任务
   " />
10.                 <property name="messageTemplateResource" value="${alarm.mail.template}"
   />
11.                 <property name="mailSender">
```

```
12.                     <bean class="org.openwebflow.alarm.impl.MailSender">
13.                         <property name="serverHost" value="${mail.host}" />
14.                         <property name="serverPort" value="${mail.port}" />
15.                         <property name="authUserName" value="${mail.username}" />
16.                         <property name="authPassword" value="${mail.password}" />
17.                         <property name="mailFrom" value="${mail.from}" />
18.                     </bean>
19.                 </property>
20.             </bean>
21.         </property>
22.         <property name="membershipManager" ref="myMembershipManager" />
23.         <property name="userDetailsManager" ref="myUserDetailsManager" />
24.         <property name="taskNotificationManager" ref="myTaskNotificationManager" />
25.     </bean>
```

可以看到，TaskAlarmServiceImpl 包含如下重要属性：

- periodInAdvance：任务催办时间提前量（相对于 dueDate 而言），格式采用 ISO 8601 时间段表示法（和 Activiti 的 due date 格式一样，参考 http://en.wikipedia.org/wiki/ISO_8601#Durations），如：P2D 代表提前 2 天
- messageNotifier：定义通知的方式，如上示例的是邮件发送的方式，用户可以改造成其它方式，如：短信方式，或者组合方式；
- membershipManager：成员关系管理器，TaskAlarmService 会通知到候选组里面的所有成员；
- userDetailsManager：用户详细信息管理器，TaskAlarmService 会在发送通知信息的时候用到用户相关属性，如：邮箱、手机号等；
- taskNotificationManager：任务通知信息管理；

ProcessEngine 在启动时会加载 StartTaskAlarmService，从而启动催办服务。

## 4.4.2 使用方法

欲使用任务催办功能，需要遵循如下步骤：

- 配置 myTaskNotificationManager、myUserDetailsManager 和 myMembershipManager；
- 配置 myTaskAlarmService；
- 在 activiti.cfg.core.xml 中的 processEngineConfiguration 里打开 StartTaskAlarmService（默认情况下这是打开的）；

接下来，不需要任何设置，在任务即将过期（dueDate）的时候，委托人（assignee）

或候选人就会收到催办通知。

# 4.5 任务流程控制

OpenWebFlow 提供了安全的加签（包括前加签/后加签）、自由跳转（包括前进/后）、分裂节点等功能。

## 4.5.1 设计方案

任务流程控制功能由 TaskFlowControlService 提供，相关的实现在 org.openwebflow.ctrl 包里面，一般通过 TaskFlowControlServiceFactory 获取到 TaskFlowControlService。

任务流程控制功能实现的关键思路是：*将对既有流程的各种调整映射成“对 Activity 对象的更新及克隆”*，并在必要的时候还要实现持久化和加载。

OpenWebFlow 在实现“新流程”的持久化的时候玩了一个 trick，即仅仅将“调整”的方式（活动工厂信息）做持久化，而非针对“调整之后的新版本”做持久化。譬如，我们根据 step2 活动创建一个 step21 活动，所有的信息都一样，这个时候只要持久化工厂类型（活动克隆）、模板活动 ID（step2）、新活动 ID（step21）就可以了。比较复杂的例子，是将某个活动分裂成 N 个串行的会签活动，这种情况也只需要记录模板活动 ID、新活动 ID 数组，不需要记录更多的信息。如下示出一个创建 N 个用户任务活动的例子：

```
1.  public class ChainedActivitiesCreator extends RuntimeActivityCreatorSupport implements
    RuntimeActivityCreator
2.  {
3.   public ActivityImpl[] createActivities(ProcessEngine processEngine, ProcessDefinitionEntity
    processDefinition,
4.              RuntimeActivityDefinitionEntity info)
5.   {
6.       info.setFactoryName(ChainedActivitiesCreator.class.getName());
7.       RuntimeActivityDefinitionEntityIntepreter radei = new
    RuntimeActivityDefinitionEntityIntepreter(info);
8.
9.       if (radei.getCloneActivityIds() == null)
10.      {
```

```
11.             radei.setCloneActivityIds(CollectionUtils.arrayToList(new
    String[radei.getAssignees().size()]));
12.         }
13.
14.         return createActivities(processEngine, processDefinition, info.getProcessInstanceId(),
15.             radei.getPrototypeActivityId(), radei.getNextActivityId(), radei.getAssignees(),
16.             radei.getCloneActivityIds());
17.     }
18.
19.     private ActivityImpl[] createActivities(ProcessEngine processEngine,
    ProcessDefinitionEntity processDefinition,
20.             String processInstanceId, String prototypeActivityId, String nextActivityId,
    List<String> assignees,
21.             List<String> activityIds)
22.     {
23.         ActivityImpl prototypeActivity = ProcessDefinitionUtils.getActivity(processEngine,
    processDefinition.getId(),
24.             prototypeActivityId);
25.
26.         List<ActivityImpl> activities = new ArrayList<ActivityImpl>();
27.         for (int i = 0; i < assignees.size(); i++)
28.         {
29.             if (activityIds.get(i) == null)
30.             {
31.                 String activityId = createUniqueActivityId(processInstanceId,
    prototypeActivityId);
32.                 activityIds.set(i, activityId);
33.             }
34.
35.             ActivityImpl clone = createActivity(processEngine, processDefinition,
    prototypeActivity,
36.                 activityIds.get(i), assignees.get(i));
37.             activities.add(clone);
38.         }
39.
40.         ActivityImpl nextActivity = ProcessDefinitionUtils.getActivity(processEngine,
    processDefinition.getId(),
41.             nextActivityId);
42.         createActivityChain(activities, nextActivity);
43.
44.         return activities.toArray(new ActivityImpl[0]);
45.     }
46. }
```

活动工厂信息用 RuntimeActivityDefinitionEntity 类来表示，为了方便，不同活动工厂的个性化信息存成了一个 JSON 字符串，并会在加载的时候解析成一个 Map。如下是一个节点分裂的活动工厂信息：

```
{"sequential":true,"assignees":["bluejoe","alex"],"cloneActivityId":"2520001:step2:1419823449424-8","prototypeActivityId":"step2"}
```

通过 RuntimeActivityDefinitionEntity 来调整流程，通过一系列的 creator 来表示。如：org.openwebflow.ctrl.creator.MultiInstanceActivityCreator 用来创建多实例节点。

### 4.5.2 使用方法

欲使用任务流程控制功能，需要遵循如下步骤：

- 配置 myActivityPermissionManager、myActivityDefinitionManager；
- 在 activiti.cfg.core.xml 中的 processEngineConfiguration 里打开 LoadRuntimeActivityDefinitions（默认情况下这是打开的），该设置会自动加载持久化的活动定义；
- 配置 TaskFlowControlServiceFactory；

```
<!-- 工作流流转服务对象工厂 -->
<bean class="org.openwebflow.ctrl.impl.DefaultTaskFlowControlServiceFactory" />
```

- 获取到 TaskFlowControlServiceFactory，创建 TaskFlowControlService：

```
_taskFlowControlServiceFactory = _ctx.getBean(DefaultTaskFlowControlServiceFactory.class);
ProcessInstance instance = _processEngine.getRuntimeService().startProcessInstanceByKey("test2");
TaskFlowControlService tfcs = _taskFlowControlServiceFactory.create(instance.getId());
```

- 使用 TaskFlowControlService：

```
ActivityImpl[] as = tfcs.insertTasksBefore("step2", "bluejoe", "alex");
```

## 4.6 模型文件导入

默认情况下，在 activiti.cfg.core.xml 中的 processEngineConfiguration 里会打开 ImportDefinedProcessModels：

```
<!-- 启动催办管理器 -->
<bean class="org.openwebflow.cfg.StartTaskAlarmService">
    <property name="taskAlarmService" ref="myTaskAlarmService" />
    <property name="runOnStartup" value="false" />
</bean>
```

该配置自动加载指定路径下（${model.dir}）的 BPMN 模型文件（以.bpmn 为后缀）至 Activiti 的 repository 库。

如果需要手动加载，可以使用 ProcessEngineTool 或者 ModelUtils 类。

# 5. 使用管理器接口实现自定义扩展

OpenWebFlow 需要用户提供 6 类管理器的接口，它们分别是：

- RuntimeActivityDefinitionManager：负责获取活动的定义信息，用以支持运行时期的新建活动
- ActivityPermissionManager：负责获取活动的权限设置信息
- TaskNotificationManager：负责存取任务催办通知信息
- DelegationManager：负责获取用户的代理信息
- UserDetailsManager：负责获取用户的信息（包括 E-mail、昵称、手机号等），主要用以发送催办通知
- IdentityMembershipManager：负责获取用户组成员关系，获取某用户的候选任务队列时，需要通过用户名获取到用户组

除了这些 Manager 之外，用户会发现 OpenWebFlow 还提供了一系列的 ManagerEx 接口：

- ActivityPermissionManagerEx：负责保存活动的权限设置信息
- TaskNotificationManagerEx：负责保存任务催办通知信息
- DelegationManagerEx：负责保存用户的代理信息

- UserDetailsManagerEx：负责保存用户的信息
- IdentityMembershipManagerEx：负责保存用户组成员关系

可以简单的认为，Manager 接口主要用以信息读取（read），ManagerEx 接口主要用以信息写入（write），***注意使用 OpenWebFlow 引擎时，ManagerEx 不是必须要提供相应实现的！***OpenWebFlow 引擎的所有操作只会调用到 Manager 而非 ManagerEx，提供 ManagerEx 的唯一用处是为了测试（如果没有写入，读取返回的永远是空白，测试就无法正常进行了）。

# 5.1 活动定义管理

客户程序往往需要在运行时候调整某个工作流的流程，如：让活动 step5 执行完之后跳转至 step2，这样的操作需要创建一个新的路径，为了保证后续流程的正常执行（特别是应用重启之后），这样的路径需要保存和加载。

## 5.1.1 RuntimeActivityDefinitionManager

RuntimeActivityDefinitionManager 包含如下方法：

| 方法摘要 | |
|---|---|
| java.util.List<RuntimeActivityDefinitionEntity> | **list**()<br>获取所有的活动定义信息，引擎会在启动的时候加载这些活动定义并进行注册 |
| void | **removeAll**()<br>删除所有活动定义 |
| void | **save**(RuntimeActivityDefinitionEntity entity)<br>新增一条活动定义的信息 |

## 5.1.2 RuntimeActivityDefinitionEntity

RuntimeActivityDefinitionEntity 对应于一条活动的定义信息：

| 方法摘要 | |
|---|---|
| void | deserializeProperties() |

| | |
|---|---|
| | 反序列化 PropertiesText 到 Map |
| java.lang.String | getFactoryName()<br>获取工厂名 |
| java.lang.String | getProcessDefinitionId()<br>获取流程定义的 ID |
| java.lang.String | getProcessInstanceId()<br>获取流程实例的 ID |
| java.lang.String | getPropertiesText()<br>获取 PropertiesText，它是一个 JSON 字符串 |
| <T> T | getProperty(java.lang.String name)<br>获取指定的属性值 |
| void | serializeProperties()<br>序列化 Map 至 PropertiesText |
| void | setFactoryName(java.lang.String factoryName)<br>设置工厂名 |
| void | setProcessDefinitionId(java.lang.String processDefinitionId)<br>设置流程定义 ID |
| void | setProcessInstanceId(java.lang.String processInstanceId)<br>设置流程实例 ID |
| void | setPropertiesText(java.lang.String propertiesText)<br>设置 PropertiesText |
| <T> void | setProperty(java.lang.String name, T value) |

# 5.2 活动权限管理

## 5.2.1 ActivityPermissionManager

活动权限的管理接口为 ActivityPermissionManager，它的定义如下：

| 方法摘要 |
|---|

| | |
|---|---|
| ActivityPermissionEntity | load(java.lang.String processDefinitionId, java.lang.String taskDefinitionKey, boolean addOrRemove)<br>获取指定活动的权限定义信息 |

### 5.2.2 ActivityPermissionEntity

非常简单，它只需要一个方法，该方法返回一个 ActivityPermissionEntity，该实体的定义如下：

| 方法摘要 | |
|---|---|
| java.lang.String | getAssignee()<br>获取直接授权人 |
| java.lang.String[] | getGrantedGroupIds()<br>获取候选组列表 |
| java.lang.String[] | getGrantedUserIds()<br>获取候选用户列表 |

### 5.2.3 ActivityPermissionManagerEx

ActivityPermissionManagerEx 实现对活动权限表的“写”操作：

| 方法摘要 | |
|---|---|
| void | removeAll()<br>删除所有权限定义信息 |
| void | save(java.lang.String processDefId, java.lang.String taskDefinitionKey, java.lang.String assignee, java.lang.String[] candidateGroupIds, java.lang.String[] candidateUserIds)<br>保存一条权限定义信息 |

## 5.3 任务催办通知管理

### 5.3.1 TaskNotificationManager

TaskNotificationManager 负责读取和设置任务催办的状态，该接口也很简单：

| 方法摘要 | |
|---|---|
| boolean | isNotified(java.lang.String taskId)<br>判断指定任务是否通知过 |
| void | setNotified(java.lang.String taskId)<br>设置指定任务的通知状态 |

可以理解成 TaskNotificationManager 维护了一个队列，记录了每个任务的通知状态（已通知为 true，未通知为 false）。

### 5.3.2 TaskNotificationManagerEx

TaskNotificationManagerEx 实现了对通知记录的“写”操作：

| 方法摘要 | |
|---|---|
| void | removeAll()<br>删除所有通知记录 |

## 5.4 用户代理关系管理

### 5.4.1 DelegationManager

DelegationManager 用以维护用户之间的代理关系。接口包含 2 个方法：

| 方法摘要 | |
|---|---|
| java.lang.String[] | getDelegates(java.lang.String delegated)<br>获取指定用户的代理人列表 |
| java.util.List<DelegationEntity> | listDelegationEntities()<br>获取所有的代理信息列表，引擎会在启动的时候加载 |

### 5.4.2 DelegationEntity

DelegationEntity 描述一条代理关系：

| 方法摘要 | |
|---|---|
| java.lang.String | getDelegate()<br>获取当前代理记录的代理人 |
| java.lang.String | getDelegated()<br>获取当前代理记录的被代理人 |

### 5.4.3 DelegationManagerEx

DelegationManagerEx 用以实现对代理记录的“写”操作：

| 方法摘要 | |
|---|---|
| void | removeAll()<br>删除所有代理信息 |
| void | saveDelegation(java.lang.String delegated,<br>java.lang.String delegate)<br>保存一条代理信息 |

## 5.5 用户详细信息管理

### 5.5.1 UserDetailsManager

UserDetailsManager 负责获取用户的信息（包括 E-mail、昵称、手机号等），主要用以发送催办通知。UserDetailsManager 接口包含的方法如下：

| 方法摘要 | |
|---|---|
| UserDetailsEntity | findUserDetails(java.lang.String userId)<br>根据用户名获取用户详细信息 |

### 5.5.2 UserDetailsEntity

UserDetailsEntity 用以描述用户详细信息，注意它没有强制要求提供诸如 getName()这样的方法，而是提供了一个 getProperty(String name)方法：

| 方法摘要 | |
|---|---|
| <T> T | getProperty(java.lang.String name)<br>获取指定属性的值 |
| java.lang.String[] | getPropertyNames()<br>获取所有的属性名 |
| java.lang.String | getUserId()<br>获取用户的 ID |
| <T> void | setProperty(java.lang.String name, T value)<br>设置指定属性的值 |

UserDetailsEntity 同时提供了几个字符串常量：

| 字段摘要 | |
|---|---|
| static java.lang.String | STRING_PROPERTY_EMAIL<br>EMAIL 属性名 |
| static java.lang.String | STRING_PROPERTY_MOBILE_PHONE_NUMBER<br>手机号码属性名 |
| static java.lang.String | STRING_PROPERTY_NICK_NAME<br>昵称属性名 |
| static java.lang.String | STRING_PROPERTY_USER_ID<br>用户 ID 属性名 |

### 5.5.3 UserDetailsManagerEx

UserDetailsManagerEx 用以实现用户信息的“写”操作：

| 方法摘要 | |
|---|---|
| void | removeAll()<br>删除所有用户信息 |
| void | saveUserDetails(UserDetailsEntity userDetails)<br>保存某个用户的信息 |

## 5.6 用户组成员关系管理

### 5.6.1 IdentityMembershipManager

IdentityMembershipManager 负责获取用户组成员关系，简单点，就是获取指定用户所在的组的 ID 列表，以及指定组内的成员 ID 列表。

| 方法摘要 | |
|---|---|
| java.util.List<java.lang.String> | findGroupIdsByUser(java.lang.String userId)<br>获取指定的用户所在的组 ID 列表 |
| java.util.List<java.lang.String> | findUserIdsByGroup(java.lang.String groupId)<br>获取指定组的成员用户 ID 列表 |

### 5.6.2 IdentityMembershipManagerEx

IdentityMembershipManagerEx 提供了对成员关系信息的“写”操作：

| 方法摘要 | |
|---|---|
| void | removeAll()<br>删除所有成员关系 |
| void | saveMembership(java.lang.String userId,<br>java.lang.String groupId)<br>保存成员关系 |

# 6. 其他帮助

本文档的下载地址为：https://github.com/bluejoe2008/openwebflow/tree/master/doc，可通过该地址及时查阅最新版本。

如果用户需要查阅 OpenWebFlow 的 Java API，可以参考 javadoc（https://github.com/bluejoe2008/openwebflow/tree/master/openwebflow-core/doc/javadoc）。

另外可以关注 Wiki（https://github.com/bluejoe2008/openwebflow/wiki），

提交话题（https://github.com/bluejoe2008/openwebflow/issues），以及与作者bluejoe2008@gmail.com 直接联系。

# 7. Activiti 的 BUG 及对策

目前作者发现 Activiti 框架存在 2 个 bug，主要表现在：

第一个 bug 是在 BaseBpmnJsonConverter 将 BPMN 模型转存为 JSON 格式的时候，会忽略对 true 布尔值的输出，这个 bug 会造成 JsonConverterUtil.getPropertyValueAsBoolean()获取到 false 值（因为此时的判断标准变成 Yes 或 No）。该 bug 的报告地址：https://github.com/Activiti/Activiti/pull/464#event-204722250

另外一个 bug 是在 BPMN 文件加载的时候，当本地字符集为非 UTF-8（如：GB2312）时，会出现模型加载的错误。该 bug 的报告地址：https://github.com/Activiti/Activiti/pull/486#event-220121880

目前作者已经针对以上两个 bug 提交了 bugfix 并被 master 版本合并。不过考虑到版本稳定性问题，OpenWebFlow 最新版本还是采用了其他方法来避免了如上 bug 的发生。