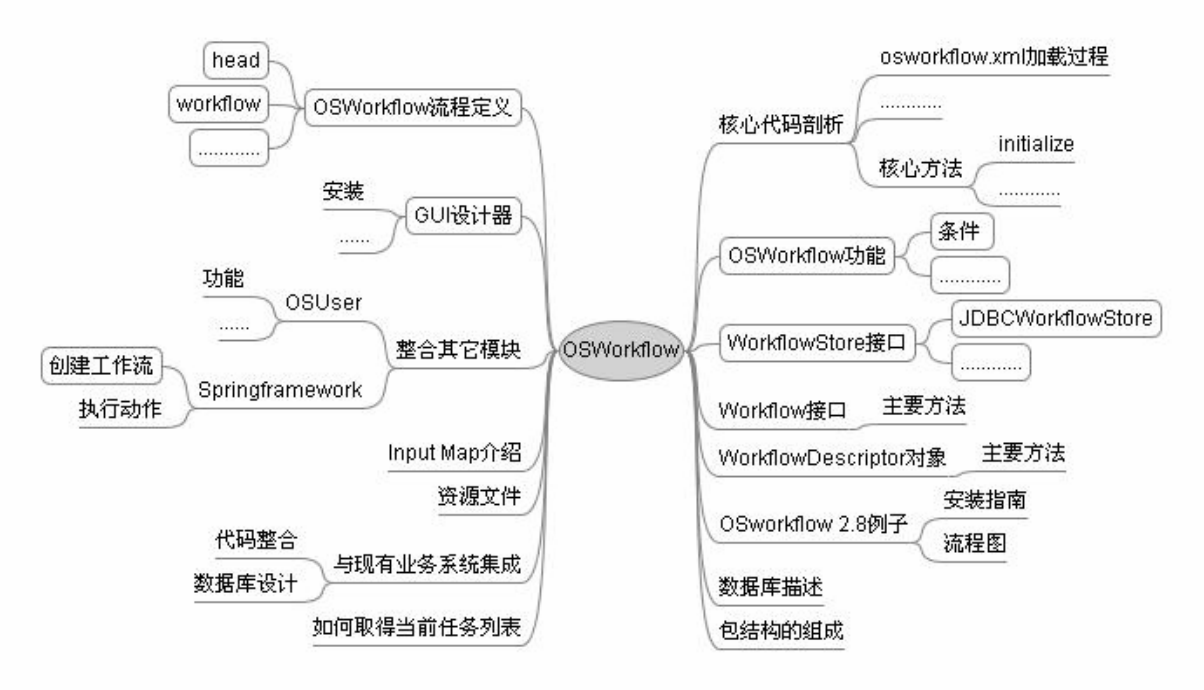
OpenDoc Series’

OSWorkflow
开发指南



V1.0

作者：陈刚

文档说明

参与人员：

作者	联络
陈刚	cucuchen520(at)yahoo.com.cn

(at) 为 email @ 符号

发布记录

版本	日期	作者	说明
1.0	2007.07.20	陈刚	编著
1.0	2007.07.20	夏昕	文档格式编排
1.0	2007.08.14	曹晓钢	编辑、校对
1.0	2007.09.13	陈克忠	审校

合作网站

本文档在合作网站 Matrix, infoq 中文站，JavaEye, CSDN, SpringSide 同步发布。

OpenDoc 版权说明

Open Doc Series 目前包括以下几份文档：

- **Spring 开发指南**
- **Hibernate 开发指南**
- **ibatis2 开发指南**
- **Webwork2 开发指南**
- **持续集成实践之 CruiseControl**
- **Using the Rake Build Language**
- **OSWorkflow 中文手册**

以上文档可从http://www.redsaga.com获取最新更新信息

简介

OSWorkflow 是 opensymphony 组织开发的一个工作流引擎，目前的版本是 2.8。OSWorkflow 用纯 Java 语言编写，并且开放源代码。它最大的特点就是极其的灵活。它面向的人群是具有技术背景的软件开发人员。OSWorkflow 不提倡用可视化工具定义流程。用户可以根据自己的实际需求，来设计出完全符合自身业务逻辑的系统，而并不需要使用复杂的代码去实现。换句话说 OSWorkflow 让我们真正解放了，使得我们从底层的代码堆中爬了出来，轻松地用一套通用的引擎机制去实现各种业务流程。OSWorkflow 提供我们所有工作流

中可能用到的元素例如：步骤（step）、条件（conditions）、循环（loops）、分支（spilts）、合并（joins）、角色（roles）、函数（function）等等。

首先我们来谈谈**步骤**：步骤是工作流中很重要的概念。如果我们把工作流比喻成一条从起点站驶向终点站的公共汽车路线，那么步骤就相当于汽车站台。而汽车有的正在排队等候进站，有的还没有进站，有的刚出站，这样就形成了所谓的“已完成”、“正在处理”、“已添加至处理队列”、“未处理”等状态。

另外一个重要的概念就是**动作**，动作就是工作流中每一步骤中″需要处理的事情″，每一个动作执行完毕以后都有一个**结果**。公共汽车停站下客就好比一个动作，动作完成以后，开向下一站，或者加油，或者返程等等就是一个结果。当然，实际上的工作流远比这辆汽车来的复杂,它涉及到的结果还包括原地踏步停留在同一步骤,或者是流转到另外的步骤中去，或者是流转到一个分支中去，或者汇集到一个合并中等。如果动作被设置成为 auto，那么只要触发器满足条件或者有来自外部的事件工作流便可自动执行。

在许多流程中，如果遇到并行处理某些事情，这就是**分支**。分支一般是指并行处理多件事情而没有先后顺序。若有一条分支进行了回退处理，整个流程都将回退。

与之相对的，**合并**就是把几条符合条件的分支聚合起来，使得事情变成″殊途同归″。这也是非常常见的流程，同时也是最复杂的一种流程。

在步骤、动作和结果中都提供了**函数**功能，函数按执行的先后时机可分为 pre-functions 和 post-functions。顾名思义，pre-functions 就是在事情发生之前执行的，而 post-functions 就是在事情发生以后执行的。

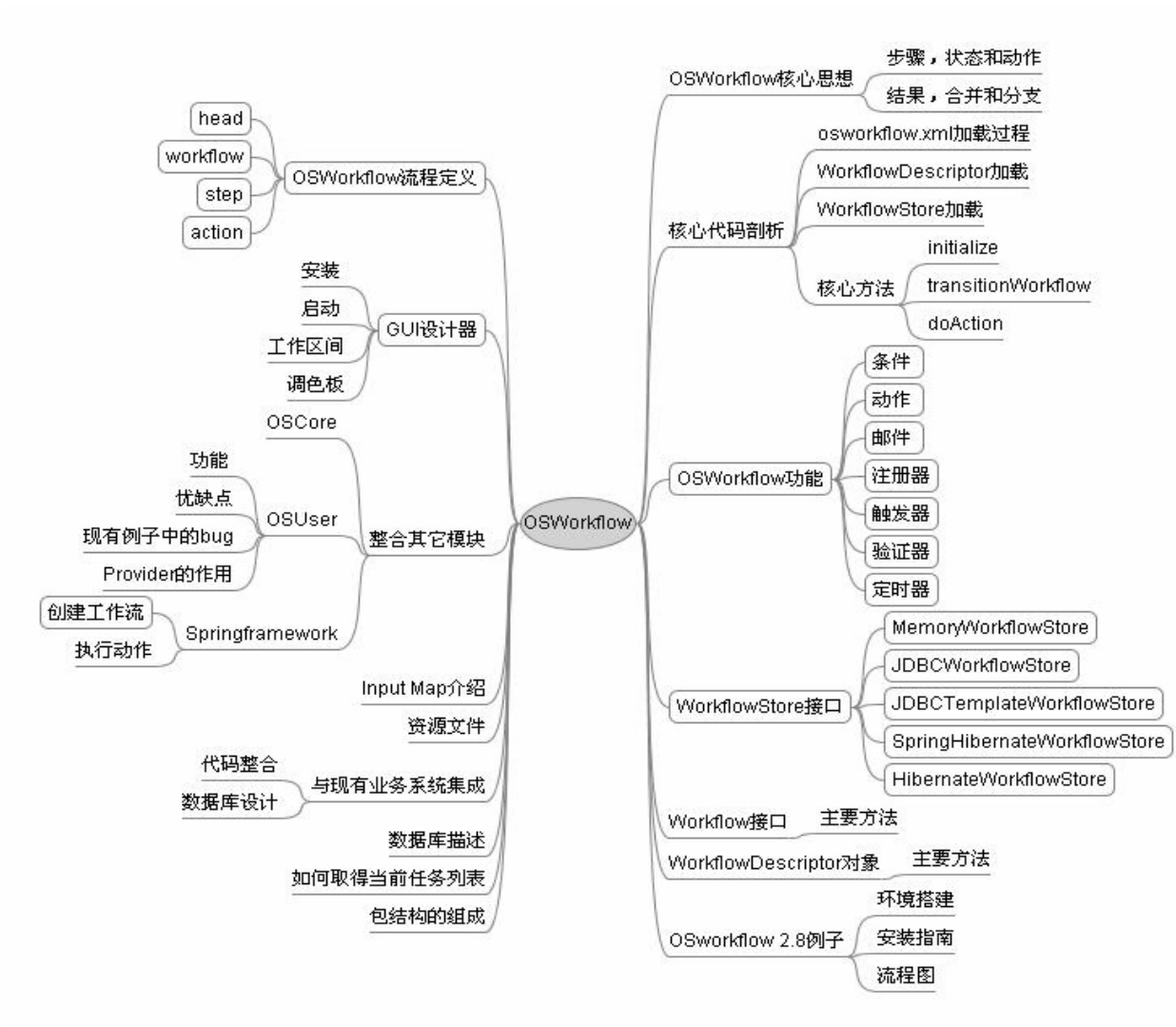
验证器是用来验证用户输入的数据是否合法的。它也可以被应用在步骤,动作或结果中。

动作的执行结果可以是有**条件**的（conditional）也可以是**无条件**的（unconditional）。对于有条件结果，可以允许有多个条件。引擎将首先检查是否有满足的条件，它会逐一进行检查，直到符合的条件被找到才能执行。如果没有一个条件被满足，那么最终引擎将产生无条件结果。

在每个步骤中调用工作流的人被称之为**调用者**（caller），而每个步骤都也会有一个**所有者**（owner），以代表在当前步骤中负责执行动作的角色或用户。

当前用户在执行当前步骤的时候，这些步骤被保留在当前表中(current)，而一旦步骤被执行完毕，引擎会马上将这个当前步骤从当前表中移到历史表中（history）。

OSWorkfow 的高级特性有发送邮件，注册器功能，通用动作和全局动作, 触发器和定时器等等，以下会一一讲解。



图： OSWorkflow 学习过程中所需掌握的各个概念的关系

跑通 OSWorkflow2.8 例子

所需 JAR 包

OSWorkflow 自身

OSWorkflow 自身(%osworkflow 解压包%\)：

osworkflow-2.8.0.jar

OSWorkflow 核心引用包

OSWorkflow 核心引用包(%osworkflow 解压包%\lib\ core)：

commons-logging.jar：必要，支持日志。
propertyset-1.4.jar：必要，支持 propertyset 的
aggregate ,cached ,memory ,jdbc ,file ,javabeans ,map ,xml 接口实现，并不支持 hibernate3，如果要支持 hibernate3，要自己写代码。这个下面再谈。
oscore-2.2.5.jar:必要，提供了一些工具等。

OSWorkflow 可选包

OSWorkflow 可选包(%osworkflow 解压包%\lib\ optional)：

bsf.jar:支持 bsf，可选。
bsh-1.2b7.jar:支持 beanshell，可选。
ehcache.jar:支持缓存，可选。
osuser-1.0-dev-2Feb05.jar:支持例子里面的用户和群组管理，在涉及到用户和群组的操作建议加上此包。

与 Spring2 联用所需包

spring2 所需的包(%spring 解压包%\dist) ：

spring.jar(version:2.05)

与 Hibernate3 联用所需包

Hibernate3 所需的包(%hibernate 解压包%\lib) ：

antlr.jar
cglib.jar
asm.jar
asm-attrs.jars
commons-collections.jar
hibernate3.jar
jta.jar
dom4j.jar
log4j.jar

WorkflowStore

配置 osworkflow 的核心之一就是配置 WorkflowStore。有多种不同的配置方法，下面逐一讲解。

MemoryWorkflowStore

在官方文档里面有现成的例子可以参照，最重要的也就是要把 persistence class 设置成为 **com.opensymphony.workflow.spi.memory.MemoryWorkflowStore**
具体来说，在\%webapp%\WEB-INF\classes 下的 osworkflow.xml 中：

```
<osworkflow>
    <persistence class="com.opensymphony.workflow.spi.memory.MemoryWorkflowStore"/>
    <factory class="com.opensymphony.workflow.loader.XMLWorkflowFactory">
        <property key="resource" value="workflows.xml" />
    </factory>
</osworkflow>
```

JDBCWorkflowStore

第一步：web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>OSWorkflow Example App</display-name>
    <description>OSWorkflow Example App</description>
  <servlet>
        <servlet-name>SOAPWorkflow</servlet-name>
```

```
        <servlet-class>com.opensymphony.workflow.soap.SOAPWorkflowServlet</servlet-class>
  </servlet>
    <servlet-mapping>
            <servlet-name>SOAPWorkflow</servlet-name>
            <url-pattern>/soap/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
            <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

第二步：配置数据源

如果是 tomcat5.0 及以上版本，在\%tomcathome%\conf\catalina\localhost 建一个 osworkflow.xml，osworkflow.xml 里面的内容如下，请注意**红色**部分为数据源名称：

```
<?xml version="1.0" encoding="UTF-8"?>
<Context crossContext="true" displayName="Welcome to Tomcat"
docBase="E:/workspace/osworkflow/exploded" path="" >
    <Resource auth="Container" name="jdbc/oswf" type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/oswf">
      <parameter>
        <name>factory</name>
        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
      </parameter>
      <parameter>
        <name>url</name>
<value>jdbc:mysql://127.0.0.1:3306/osworkflow?useUnicode=true&amp;characterEncoding=GB
K</value>
      </parameter>
      <parameter>
        <name>password</name>
        <value>{用户实际的密码}</value>
      </parameter>
      <parameter>
        <name>maxActive</name>
        <value>10</value>
      </parameter>
      <parameter>
        <name>maxWait</name>
        <value>10000</value>
      </parameter>
      <parameter>
        <name>driverClassName</name>
        <value>com.mysql.jdbc.Driver</value>
      </parameter>
      <parameter>
```

```
        <name>username</name>
        <value>root</value>
      </parameter>
      <parameter>
        <name>maxIdle</name>
        <value>3</value>
      </parameter>
   </ResourceParams>
</Context>
```

如果是 resin，请在 web.xml 里面加入以下代码：

```
<resource-ref>
        <res-ref-name>jdbc/oswf</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <init-param driver-name="com.mysql.jdbc.Driver"/>
        <init-param url="jdbc:mysql://localhost:3306/osworkflow"/>
        <init-param user="root"/>
        <init-param password=""/>
        <init-param max-connections="20"/>
        <init-param max-idle-time="30"/>
    </resource-ref>
```

最终 web.xml 如下，红色部分为新加的数据源：

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
    <web-app>
        <display-name>OSWorkflow Example App</display-name>
        <description>OSWorkflow Example App</description>
        <servlet>
            <servlet-name>SOAPWorkflow</servlet-name>
<servlet-class>com.opensymphony.workflow.soap.SOAPWorkflowServlet</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>SOAPWorkflow</servlet-name>
            <url-pattern>/soap/*</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
        <resource-ref>
            <res-ref-name>jdbc/oswf</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <init-param driver-name="com.mysql.jdbc.Driver"/>
            <init-param url="jdbc:mysql://localhost:3306/osworkflow"/>
            <init-param user="root"/><!—数据库用户名-->
```

```
            <init-param password=""/><!—数据库密码-->
            <init-param max-connections="20"/>
            <init-param max-idle-time="30"/>
        </resource-ref>
</web-app>
```

第三步：配置 osworkflow 必要文件。

确定在\%webapp%\WEB-INF\classes\下面有以下几个文件：

example.xml:例子，不作修改。

workflows.xml:例子，不作修改。

osuser.xml,修改如下，请注意**红色**的数据源部分：

```
<opensymphony-user>
    <provider class="com.opensymphony.user.provider.jdbc.JDBCAccessProvider">
        <property name="user.table">os_user</property>
        <property name="group.table">os_group</property>
        <property name="membership.table">os_membership</property>
        <property name="user.name">username</property>
        <property name="user.password">passwordhash</property>
        <property name="group.name">groupname</property>
        <property name="membership.userName">username</property>
        <property name="membership.groupName">groupname</property>
        <property name="datasource">jdbc/oswf</property>
    </provider>
    <provider class="com.opensymphony.user.provider.jdbc.JDBCCredentialsProvider">
        <property name="user.table">os_user</property>
        <property name="group.table">os_group</property>
        <property name="membership.table">os_membership</property>
        <property name="user.name">username</property>
        <property name="user.password">passwordhash</property>
        <property name="group.name">groupname</property>
        <property name="membership.userName">username</property>
        <property name="membership.groupName">groupname</property>
        <property name="datasource">jdbc/oswf</property>
    </provider>
    <provider class="com.opensymphony.user.provider.jdbc.JDBCProfileProvider">
        <property name="user.table">os_user</property>
        <property name="group.table">os_group</property>
        <property name="membership.table">os_membership</property>
        <property name="user.name">username</property>
        <property name="user.password">passwordhash</property>
        <property name="group.name">groupname</property>
        <property name="membership.userName">username</property>
        <property name="membership.groupName">groupname</property>
        <property name="datasource">jdbc/oswf</property>
    </provider>
```

```
        <authenticator class="com.opensymphony.user.authenticator.SmartAuthenticator"/>
    </opensymphony-user>
    osworkflow.xml 修改如下：
    <osworkflow>
        <persistence class="com.opensymphony.workflow.spi.jdbc.MySQLWorkflowStore">
            <!-- For jdbc persistence, all are required. -->
            <property key="datasource" value="jdbc/oswf"/>
            <!-- mysql -->
            <property key="entry.sequence" value="SELECT max(ID)+1 FROM
OS_WFENTRY"/>
            <property key="step.sequence" value="SELECT max(ID)+1 FROM
OS_STEPIDS"/>

            <property key="entry.table" value="OS_WFENTRY"/>
            <property key="entry.id" value="ID"/>
            <property key="entry.name" value="NAME"/>
            <property key="entry.state" value="STATE"/>
            <property key="history.table" value="OS_HISTORYSTEP"/>
            <property key="current.table" value="OS_CURRENTSTEP"/>
            <property key="historyPrev.table" value="OS_HISTORYSTEP_PREV"/>
            <property key="currentPrev.table" value="OS_CURRENTSTEP_PREV"/>
            <property key="step.id" value="ID"/>
            <property key="step.entryId" value="ENTRY_ID"/>
            <property key="step.stepId" value="STEP_ID"/>
            <property key="step.actionId" value="ACTION_ID"/>
            <property key="step.owner" value="OWNER"/>
            <property key="step.caller" value="CALLER"/>
            <property key="step.startDate" value="START_DATE"/>
            <property key="step.finishDate" value="FINISH_DATE"/>
            <property key="step.dueDate" value="DUE_DATE"/>
            <property key="step.status" value="STATUS"/>
            <property key="step.previousId" value="PREVIOUS_ID"/>

            <!—mysql 特殊配置-->
            <property key="step.sequence.increment" value="INSERT INTO OS_STEPIDS
(ID) values (null)"/>
            <property key="step.sequence.retrieve" value="SELECT max(ID) FROM
OS_STEPIDS"/>
            <property key="entry.sequence.increment" value="INSERT INTO
OS_ENTRYIDS (ID) values (null)"/>
            <property key="entry.sequence.retrieve" value="SELECT max(ID) FROM
OS_ENTRYIDS"/>
        </persistence>
        <factory class="com.opensymphony.workflow.loader.XMLWorkflowFactory">
```

```
            <property key="resource" value="workflows.xml"/>
        </factory>
    </osworkflow>
```

对 osworkflow.xml 的一点说明：如果说是 mysql 数据库如上即可，如果是别的数据库，

（1）不需要 mysql 特殊配置四项，并且要把 persistence 的 class 改为 com.opensymphony.workflow.spi.jdbc.JDBCWorkflowStore

（2）如果是 oracle 数据库，则红色字体部分

```
<property key="entry.sequence" value="SELECT max(ID)+1 FROM OS_WFENTRY"/>
<property key="step.sequence" value="SELECT max(ID)+1 FROM OS_STEPIDS"/>
```

改成：

```
<property key="entry.sequence" value="SELECT SEQ_OS_WFENTRY.NEXTVAL FROM DUAL"/>
<property key="step.sequence" value="SELECT EQ_OS_CURRENTSTEPS.NEXTVAL FROM DUAL"/>
```

（3）如果是 ms sql server 数据库，则红色字体部分

```
<property key="entry.sequence" value="SELECT max(ID)+1 FROM OS_WFENTRY"/>
<property key="step.sequence" value="SELECT max(ID)+1 FROM OS_STEPIDS"/>
```

改成：

```
<property key="entry.sequence"
                    value="select count(*) + 1 from os_wfentry"/>
<property key="step.sequence"
value="select sum(c1) + 1 from (select 1 as tb, count(*) as c1
        from os_currentstep union
        select 2 as tb, count(*) as c1 from os_historystep) as TabelaFinal"/>
```

propertyset.xml 配置如下,红色部分为数据源名称：

```
<propertysets>
<propertyset name="jdbc"
class="com.opensymphony.module.propertyset.database.JDBCPropertySet">
            <arg name="datasource" value="jdbc/oswf"/>
            <arg name="table.name" value="OS_PROPERTYENTRY"/>
            <arg name="col.globalKey" value="GLOBAL_KEY"/>
            <arg name="col.itemKey" value="ITEM_KEY"/>
            <arg name="col.itemType" value="ITEM_TYPE"/>
            <arg name="col.string" value="STRING_VALUE"/>
            <arg name="col.date" value="DATE_VALUE"/>
            <arg name="col.data" value="DATA_VALUE"/>
            <arg name="col.float" value="FLOAT_VALUE"/>
            <arg name="col.number" value="NUMBER_VALUE"/>
        </propertyset>
    </propertysets>
```

第四步：将 osworkflow 解压包\src\etc\deployment\jdbc 里面相应数据库的 sql 导到库中。

SpringHibernateWorkflowStore

这种方法是联合使用 spring 2 和 Hibernate 3 的配置方法。

第一步：配置 web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>OSWorkflow Example App</display-name>
    <description>OSWorkflow Example App</description>
    <context-param>
        <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/classes/applicationContext-hibernate3.xml
            </param-value>
        </context-param>
    <listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>SOAPWorkflow</servlet-name>
<servlet-class>com.opensymphony.workflow.soap.SOAPWorkflowServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SOAPWorkflow</servlet-name>
        <url-pattern>/soap/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

第二步：配置数据源，同上。

第三步：配置 OSWorkflow 必要文件。

确定在\%webapp%\WEB-INF\classes\下面有以下几个文件：

example.xml:例子，不作修改。

workflows.xml:例子，不作修改。

osuser.xml,因为 osuser 是比较独立的模块，目前还没有支持 hibernate3，修改可以同上，这就意味着 osuser 还是要用 JDBC 存储方式。

osworkflow.xml 由于我们现在采用的是 spring+hibernate3 方式，所以便要去掉这个文件，而增加一个 spring 配置文件 applicationContext-hibernate3.xml，其配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
        "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
```

```
        <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
            <property name="jndiName">
                <value>java:comp/env/jdbc/oswf</value>
            </property>
        </bean>
        <!-- Hibernate SessionFactory -->
        <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
            <property name="dataSource">
                <ref bean="dataSource"/>
            </property>
            <property name="mappingResources">
                <list>

<value>com/opensymphony/workflow/spi/hibernate3/HibernateWorkflowEntry.hbm.xml</value>
<value>com/opensymphony/workflow/spi/hibernate3/HibernateHistoryStep.hbm.xml</value>
<value>com/opensymphony/workflow/spi/hibernate3/HibernateCurrentStep.hbm.xml</value>
<value>com/opensymphony/workflow/spi/hibernate3/HibernateCurrentStepPrev.hbm.xml</value
>
<value>com/opensymphony/workflow/spi/hibernate3/HibernateHistoryStepPrev.hbm.xml</value
>
<value>com/opensymphony/module/propertyset/hibernate3/PropertySetItem.hbm.xml</value>
                </list>
            </property>
            <property name="hibernateProperties">
                <props>
                    <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                    <prop key="hibernate.show_sql">false</prop>
                </props>
            </property>
        </bean>
        <!-- Transaction manager for a single Hibernate SessionFactory (alternative to JTA) -->
        <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
            <property name="sessionFactory">
                <ref bean="sessionFactory"/>
            </property>
        </bean>
        <!-- Transaction template for Managers, from:
    http://blog.exis.com/colin/archives/2004/07/31/concise-transaction-definitions-spring-11/ -->
        <bean id="txProxyTemplate" abstract="true"
          class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
            <property name="transactionManager" ref="transactionManager"/>
```

```
            <property name="proxyTargetClass" value="true"/>
            <property name="transactionAttributes">
                <props>
                    <prop key="do*">PROPAGATION_REQUIRED</prop>
                    <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
                </props>
            </property>
        </bean>
        <bean id="propertySetDelegate"
class="com.opensymphony.workflow.spi.hibernate3.DefaultHibernatePropertySetDelegate">
            <property name="sessionFactory">
                <ref bean="sessionFactory"/>
            </property>
        </bean>
        <bean id="workflowStore"
class="com.opensymphony.workflow.spi.hibernate3.SpringHibernateWorkflowStore">
            <property name="sessionFactory">
                <ref bean="sessionFactory"/>
            </property>
            <property name="propertySetDelegate">
                <ref bean="propertySetDelegate"/>
            </property>
        </bean>
        <bean id="workflowFactory"
class="com.opensymphony.workflow.spi.hibernate.SpringWorkflowFactory" init-method="init">
            <property name="resource">
                <value>workflows.xml</value>
            </property>
            <property name="reload">
                <value>true</value>
            </property>
        </bean>
        <bean id="osworkflowConfiguration"
class="com.opensymphony.workflow.config.SpringConfiguration">
            <property name="store">
                <ref local="workflowStore"/>
            </property>
            <property name="factory">
                <ref local="workflowFactory"/>
            </property>
        </bean>
        <bean id="workflowTypeResolver"
class="com.opensymphony.workflow.util.SpringTypeResolver">
        </bean>
```

</beans>

由配置文件可以看出和官方或者一般的 spring+hibernate3 配置方式有很大的不同。在数据库中增加了和 JDBCWorkflowStore 数据库相同的两张表 os_current_step_prev 还有 os_history_step_prev.另外 os_propertyentry 表结构和 JDBCWorkflowStore 中的 os_propertyentry 表结构有所不同，对比如下：

JDBCWorkflowStore. os_propertyentry

```
create table OS_PROPERTYENTRY
(
    GLOBAL_KEY varchar(250) NOT NULL,
    ITEM_KEY varchar(250) NOT NULL,
    ITEM_TYPE tinyint,
    STRING_VALUE varchar(255),
    DATE_VALUE datetime,
    DATA_VALUE blob,
    FLOAT_VALUE float,
    NUMBER_VALUE numeric,
    primary key (GLOBAL_KEY, ITEM_KEY)
)TYPE=InnoDB;
```

SpringHibernateWorkflowStore. os_propertyentry

```
CREATE TABLE `os_propertyentry` (
  `entity_name` varchar(125) NOT NULL,
  `entity_id` bigint(20) NOT NULL,
  `entity_key` varchar(255) NOT NULL,
  `key_type` int(11) default NULL,
  `boolean_val` int(1) default '0',
  `double_val` double default NULL,
  `string_val` varchar(255) default NULL,
  `long_val` bigint(20) default NULL,
  `int_val` int(11) default NULL,
  `date_val` date default NULL,
  PRIMARY KEY  (`entity_name`,`entity_id`,`entity_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

os_entryids 表和 os_stepids 表都不曾用到,故可以删除。

程序也有极大不同，增加了 com\opensymphony\module\propertyset\hibernate3 对 hibernate3 的 propertyset 支持；修改了 com\opensymphony\workflow\spi\hibernate3 中的 java 文件特别是 AbstractHibernateWorkflowStore.java

*.hbm.xml 文件都要加上 default-lazy= "false"，另外必须说明 HibernateHistoryStep.hbm.xml 里面的生成 ID 的 class 要设置成为 assigned，因为它的 ID 其实就是从 HibernateCurrentStep.hbm.xml 里面移过来的，所以不能自增长。

另外请读者注意：如果用的是非mysql数据库，第一次运行前你可以在 applicationContext-hibernate3.xml文件中sessionFactory bean定义中<prop key="hibernate.show_sql">false</prop>的后面加上一个<prop

key="hibernate.hbm2ddl.auto">**create-drop**</prop>，这样可以不用创建除用户和群组之外的*.hbm.xml表，在启动服务的时候系统会自动为*.hbm.xml创建所对应的表。不过注意要在创建之后把它注释掉，不然每次启动服务的时候都会重新创建一次，并且上次的数据也会因此而丢失。

propertyset.xml 中增加对 hibernate3 的支持，配置如下：

```
<propertysets>
<propertyset name="hibernate3"
class="com.opensymphony.module.propertyset.hibernate3.HibernatePropertySet"/>
</propertysets>
```

详细代码请登陆我的blog下载。下载地址：http://cucuchen520.javaeye.com/

JDBCTemplateWorkflowStore

如果大家在使用 spring 时，使用 JDBCTempate 存储数据，不妨采用这种方式，在我的示例代码中集成了此功能。注：此方式所用数据库和 jdbc 方式完全相同。

第一步：配置 web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>OSWorkflow Example App</display-name>
    <description>OSWorkflow Example App</description>
    <context-param>
        <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/classes/applicationContext-jdbc.xml
            </param-value>
        </context-param>
    <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>SOAPWorkflow</servlet-name>
<servlet-class>com.opensymphony.workflow.soap.SOAPWorkflowServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SOAPWorkflow</servlet-name>
        <url-pattern>/soap/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

第二步：配置数据源，同上。

第三步：配置 osworkflow 必要文件。

确定在\%webapp%\WEB-INF\classes\下面有以下几个文件：

example.xml:例子，不作修改。

workflows.xml:例子，不作修改。

osuser.xml,因为 osuser 是比较独立的模块，目前还没有支持 spring jdbcTempate，修改可以同上，这就意味着 osuser 还是要用普通 JDBC 存储方式。

osworkflow.xml 由于我们现在采用的是 jdbcTempate，所以便要去掉这个文件，而增加一个 spring 配置文件 applicationContext-jdbc.xml，其配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
        "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName">
            <value>java:comp/env/jdbc/oswf</value>
        </property>
    </bean>
    <!--
    <bean id="jdbcTemplateWorkflowStore"
          class="com.opensymphony.workflow.spi.jdbc.JDBCTemplateWorkflowStore">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
        <property name="jdbcTemplateProperties">
            <props>
                <prop key="history.table">OS_HISTORYSTEP</prop>
                <prop key="historyPrev.table">OS_HISTORYSTEP_PREV</prop>
                <prop key="current.table">OS_CURRENTSTEP</prop>
                <prop key="currentPrev.table">OS_CURRENTSTEP_PREV</prop>
                <prop key="step.id">ID</prop>
                <prop key="step.entryId">ENTRY_ID</prop>
                <prop key="step.stepId">STEP_ID</prop>
                <prop key="step.actionId">ACTION_ID</prop>
                <prop key="step.owner">OWNER</prop>
                <prop key="step.caller">CALLER</prop>
                <prop key="step.startDate">START_DATE</prop>
                <prop key="step.finishDate">FINISH_DATE</prop>
                <prop key="step.dueDate">DUE_DATE</prop>
                <prop key="step.status">STATUS</prop>
                <prop key="step.previousId">PREVIOUS_ID</prop>
                <!--oracle-->
                <property key="entry.sequence" value="SELECT
SEQ_OS_WFENTRY.NEXTVAL FROM DUAL"/>
```

```
                    <property key="step.sequence" value="SELECT
EQ_OS_CURRENTSTEPS.NEXTVAL FROM DUAL"/>
                        <prop key="entry.table">OS_WFENTRY</prop>
                        <prop key="entry.id">ID</prop>
                        <prop key="entry.name">NAME</prop>
                        <prop key="entry.state">STATE</prop>
                    </props>
                </property>
            </bean>
            -->
```

jdbcTemplateWorkflowStore 这个 bean 是对非 mysql 数据库的支持，里面的 step.sequence 和 entry.sequence 的配置根据不同数据库而配置不同的值，它们的值和 jdbcWorkflowStore 中的完全一样，以上为 oracle 配置。

```
            <bean id="mysqlTemplateWorkflowStore"

class="com.opensymphony.workflow.spi.jdbc.MySQLTemplateWorkflowStore">
                <property name="dataSource">
                    <ref bean="dataSource"/>
                </property>
                <property name="propertySetDelegate">
                    <ref bean="propertySetDelegate"/>
                </property>
                <property name="jdbcTemplateProperties">
                    <props>
                        <prop key="history.table">OS_HISTORYSTEP</prop>
                        <prop key="historyPrev.table">OS_HISTORYSTEP_PREV</prop>
                        <prop key="current.table">OS_CURRENTSTEP</prop>
                        <prop key="currentPrev.table">OS_CURRENTSTEP_PREV</prop>
                        <prop key="step.id">ID</prop>
                        <prop key="step.entryId">ENTRY_ID</prop>
                        <prop key="step.stepId">STEP_ID</prop>
                        <prop key="step.actionId">ACTION_ID</prop>
                        <prop key="step.owner">OWNER</prop>
                        <prop key="step.caller">CALLER</prop>
                        <prop key="step.startDate">START_DATE</prop>
                        <prop key="step.finishDate">FINISH_DATE</prop>
                        <prop key="step.dueDate">DUE_DATE</prop>
                        <prop key="step.status">STATUS</prop>
                        <prop key="step.previousId">PREVIOUS_ID</prop>
    <!--mysql-->
                        <prop key="step.sequence">SELECT max(ID)+1 FROM
OS_STEPIDS</prop>
                        <prop key="entry.sequence">SELECT max(ID)+1 FROM
```

```
OS_WFENTRY</prop>
                        <prop key="entry.table">OS_WFENTRY</prop>
                        <prop key="entry.id">ID</prop>
                        <prop key="entry.name">NAME</prop>
                        <prop key="entry.state">STATE</prop>
    <!—新加四项-->
                        <prop key="step.sequence.increment">INSERT INTO OS_STEPIDS (ID)
values (null)</prop>
                        <prop key="step.sequence.retrieve">SELECT max(ID) FROM
OS_STEPIDS</prop>
                        <prop key="entry.sequence.increment">INSERT INTO OS_ENTRYIDS
(ID) values (null)</prop>
                        <prop key="entry.sequence.retrieve">SELECT max(ID) FROM
OS_ENTRYIDS</prop>
                    </props>
                </property>
            </bean>
```

mysqlTemplateWorkflowStore 是专门为 mysql 数据库定制的一个 bean，大家可以看出 step.sequence 和 entry.sequence 与 jdbcTemplateWorkflowStore 中的差别，另外新加了四项配置。其实大家用心思考一下就会知道，以上两个 bean 里面所有的配置都是从 JDBCWorkflowStore 的 osworkflow.xml 中 copy 而来，可以看出完全没有作任何的改变。

```
            <bean id="propertySetDelegate"
class="com.opensymphony.workflow.spi.jdbc.DefaultJDBCTemplatePropertySetDelegate">
                <property name="dataSource">
                    <ref bean="dataSource"/>
                </property>
            </bean>
            <bean id="workflowFactory"
class="com.opensymphony.workflow.spi.hibernate.SpringWorkflowFactory" init-method="init">
                <property name="resource">
                    <value>workflows.xml</value>
                </property>
                <property name="reload">
                    <value>true</value>
                </property>
            </bean>
            <bean id="osworkflowConfiguration"
class="com.opensymphony.workflow.config.SpringConfiguration">
                <property name="store">
    <!—如果是 mysql 调用 mysqlTemplateWorkflowStore ， 如果不是则调用
jdbcTemplateWorkflowStore -->
                    <ref local="mysqlTemplateWorkflowStore"/>
                </property>
```

```
            <property name="factory">
                <ref local="workflowFactory"/>
            </property>
        </bean>
        <bean id="workflowTypeResolver"
class="com.opensymphony.workflow.util.SpringTypeResolver">
        </bean>
    </beans>
```

propertyset.xml 中增加对 jdbcTempate 的支持，配置如下：

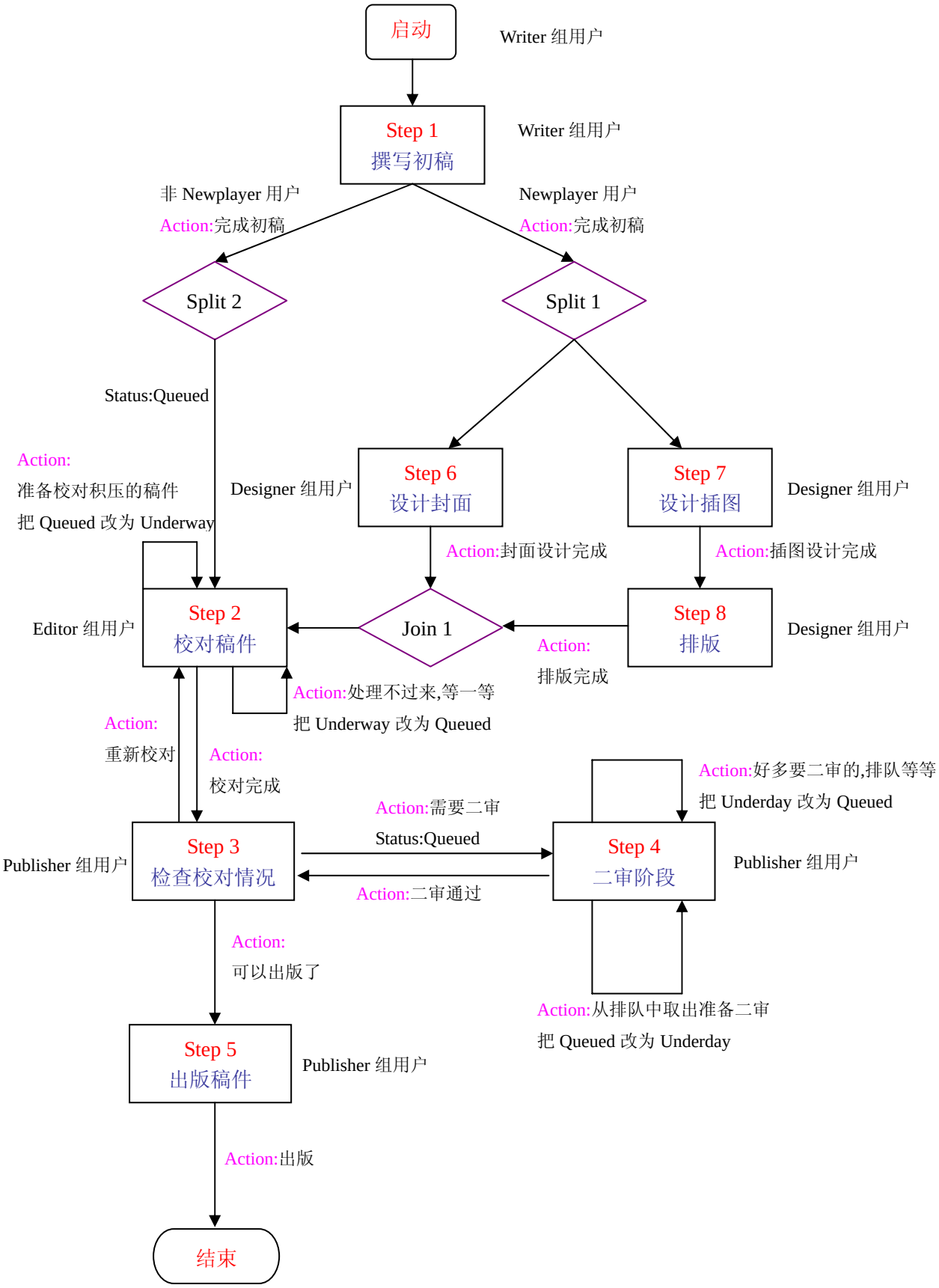
```
    <propertysets>
    <propertyset name="hibernate3"
class="com.opensymphony.module.propertyset.hibernate3.HibernatePropertySet"/>
    </propertysets>
    <propertyset name="jdbcTemplate"
class="com.opensymphony.module.propertyset.database.JDBCTemplatePropertySet">
            <arg name="table.name" value="OS_PROPERTYENTRY"/>
            <arg name="col.globalKey" value="GLOBAL_KEY"/>
            <arg name="col.itemKey" value="ITEM_KEY"/>
            <arg name="col.itemType" value="ITEM_TYPE"/>
            <arg name="col.string" value="STRING_VALUE"/>
            <arg name="col.date" value="DATE_VALUE"/>
            <arg name="col.data" value="DATA_VALUE"/>
            <arg name="col.float" value="FLOAT_VALUE"/>
            <arg name="col.number" value="NUMBER_VALUE"/>
        </propertyset>
```

详细代码请登陆我的blog下载。下载地址：http://cucuchen520.javaeye.com/

HibernateWorkflowStore

这种方式我是不推荐使用的，因为它需要自己控制 Hibernate Session，而且官方的例子到目前为止我一直没有配置成功过，如果想用这种方式，必须要花一番苦功夫改写源代码☹

例子流程图


启动
Writer 组用户
Step 1
撰写初稿
Writer 组用户
非 Newplayer 用户
Action:完成初稿
Newplayer 用户
Action:完成初稿
Split 2
Split 1
Status:Queued
Action:
准备校对积压的稿件
把 Queued 改为 Underway
Designer 组用户
Step 6
设计封面
Step 7
设计插图
Designer 组用户
Action:封面设计完成
Action:插图设计完成
Editor 组用户
Step 2
校对稿件
Join 1
Step 8
排版
Designer 组用户
Action:
排版完成
Action:处理不过来,等一等
把 Underway 改为 Queued
Action:
重新校对
Action:
校对完成
Action:好多要二审的,排队等等
把 Underday 改为 Queued
Action:需要二审
Status:Queued
Publisher 组用户
Step 3
检查校对情况
Step 4
二审阶段
Publisher 组用户
Action:二审通过
Action:
可以出版了
Action:从排队中取出准备二审
把 Queued 改为 Underday
Step 5
出版稿件
Publisher 组用户
Action:出版
结束

与 Spring 联用的 OSWorkflow 工作流

创建工作流

下面我们来简单介绍一下如何在 spring（或还有 hibernate）环境下，使用 OSWorkflow 的 API 来创建一个新的工作流。首先应取得 Spring 中的 ApplicationContext, 其次是得到 Spring Configuration, 将 Configuration 设置到 Workflow 对象中去。下面是示例代码：

```
ApplicationContext cxt =
WebApplicationContextUtils.getWebApplicationContext(this.getServletConfig().getServletContext());
Configuration conf = (SpringConfiguration) cxt.getBean("osworkflowConfiguration");
Workflow wf = new BasicWorkflow(username);
wf.setConfiguration(conf);
Map inputs = new HashMap();
inputs.put("docTitle", request.getParameter("title"));
wf.initialize("workflowName", 1, inputs);
```

注意：一般情况下，你应该将 wf 定义为 Workflow 接口，而不应该是 BasicWorkflow。

执行动作

和创建工作流类似，在和 Spring 绑定后，执行一个动作也非常简单，代码如下：

```
ApplicationContext cxt =
WebApplicationContextUtils.getWebApplicationContext(this.getServletConfig().getServletContext());
Configuration conf = (SpringConfiguration) cxt.getBean("osworkflowConfiguration");
Workflow wf = new BasicWorkflow(username);
wf.setConfiguration(conf);
HashMap inputs = new HashMap();
inputs.put("docTitle", request.getParameter("title"));
long id = Long.parseLong(request.getParameter("workflowId"));
wf.doAction(id, 1, inputs);
```

以上的代码中，都是显式获取 Spring 配置文件中的 osworkflowConfiguration bean 的，没有利用 spring 自身的注射功能。更符合 spring 理念的方法是将 workflow 配置成为 spring 的 bean，并将 osworkflowConfiguration 注射到 workflow 中去。

调用接口中的参数和方法详解

Input Map

workflow 的 initialize、doAction，还有 wf.getAvailableActions(id, map)方法都有一个 HashMap 类型的参数，是用来传递工作流中所需要的数据或者对象的,其有效范围只包括当前步骤或者下一步的 pre-function。可应用在 function,condition,validator 等当中。

Workflow 接口里面的主要方法

getAvailableActions(id, null)	得到当前工作流实例所有有效动作. 第一个参数:工作流实例 ID 第二个参数:要传递的对象
getCurrentSteps(id)	根据工作流实例 ID 得到所有当前步骤.
getEntryState(id)	根据工作流实例 ID 得到所有当前实例状态.
getHistorySteps(id)	根据工作流实例 ID 得到所有历史步骤.
getPropertySet(id)	根据工作流实例 ID 得到 PropertySet 对象.
getSecurityPermissions(id)	根据工作流实例 ID 得到当前状态权限列表.
getWorkflowDescriptor(id)	根据工作流实例 ID 得到工作流 XML 描述文件对象.
getWorkflowName(id)	根据工作流实例 ID 得到当前实例名称.
getWorkflowNames()	得到所有的工作流程名称

WorkflowDescriptor 对象里面的主要方法

getAction(id)	根据动作 ID 得到当前动作描述信息. 有三种类型：初始化动作；全局动作；步骤中的一般动作。
getCommonActions()	得到通用动作列表.
getGlobalActions()	得到全局动作列表. 3
getGlobalConditions()	得到全局条件列表.
getInitialAction(id)	根据初始动作 ID 得到初始动作描述信息.
getInitialActions()	得到所有初始动作描述信息.
getJoin(id)	根据 Join ID 得到相应 Join 描述信息.
getJoins()	得到所有 Join 描述信息.
getName()	得到工作流程描述文件名称.
getRegisters()	得到所有的注册器.
getSplit(id)	根据 Split ID 得到相应 Split 描述信息.
getSplits()	得到所有 Split 描述信息.
getStep(id)	根据 Step ID 得到相应 Step 描述信息.
getSteps()	得到所有 Step 描述信息.

getTriggerFunction(id)	根据 TriggerFunction ID 得到相应 TriggerFunction 描述信息.
getTriggerFunctions()	得到所有 TriggerFunction 描述信息.

OSUser 详解

OSUser 几大功能

- ✓ 对用户数据进行维护
- ✓ 对群组数据进行维护
- ✓ 对用户所属群组进行维护
- ✓ 提供用户安全验证
- ✓ 提供对 propertyset 数据进行增加删除的支持

OSUser 的优点

灵活性：只需要配置一个 osuser.xml 文件就可以轻松搞定所有与用户和权限相关的逻辑。
通用性：OSUser 的设计规范实用于所有的业务系统。

OSUser 的缺点

目前只支持 JDBC 而不支持风靡全球的 hibernate 等；可扩展性比较差，比如说 os_user 只有两个字段，如果想要在数据库表中加上用户的年龄就比较麻烦。如果想完全做到用 hibernate 操作用户模块，建议大家摈弃 OSUser，完全由自己重写这一块的代码。

OSUser 现有例子中的 bug

如果开发者使用的是 SpringHibernateStore 的存储方式,那么在前台 jsp 中的 manager 模块将会抛出异常，原因是因为在 OSUser 中引用了 jdbc propertyset 模块，但是 hibernate propertyset 中所用的 os_propertyentry 表和 jdbc 中所用的 os_propertyentry 表有很大的不同而导致如 ITEM_KEY 找不到这样的异常发生。目前暂时还没有解决的办法，我建议大家重新写一个对 propertyset 进行维护的模块。
还有一个bug就是在http://localhost/default.jsp中创建用户和群组时用户名称和群组名称不能重名，这个非常要命，一旦重名便抛出com.opensymphony.user.DuplicateEntityException: group test already exists
这样的异常，这使得很多朋友看着莫名其妙。追究其原因，是由于 OSUser 开发组在开始的时候，写重名的判断是查找所有的用户和群组进行比对，而不是只查找用户或者只查找群组，用户和群组都被 put 到 credentialsProviders 这个 List 中去了。所以大家在使用的过程中还是要多加注意。

Provider 的作用

com.opensymphony.user.provider.jdbc.JDBCAccessProvider
对群组进行维护，其中的方法有：

- 增加一个用户到群组中；
- 创建一个新的群组；
- 判断群组是否存在；
- 判断用户是否在一个群组中；
- 群组列表；
- 列出有某个用户的所有群组列表；
- 根据群组名列出所有用户；
- 删除一个群组；
- 将一个用户从某个群组中移除。

com.opensymphony.user.provider.jdbc.JDBCCredentialsProvider
对用户进行维护，其中的方法有：

- 根据传入的用户名和密码进行安全认证；
- 根据传入的用户名和密码修改密码；
- 创建一个新的用户；
- 判断用户是否存在；
- 用户列表；
- 删除一个用户。

com.opensymphony.user.provider.jdbc.JDBCProfileProvider
里面主要的方法是取得对 PropertySet 的操作接口；还有一个方法是根据传入的用户名判断他(她)是否存在于用户表中。

OSWorkflow 包的描述

说明:所有包均以 com.opensymphony.workflow 开头

	定义了 **Workflow,Condition,FunctionProvider,Register,Validator** 等接口, 还定义了一些常用的 **Exception** 和很重要的 **JoinNodes** 对象等
basic	只有两个对象: **BasicWorkflow**, 未实现事务回滚; **BasicWorkflowContext**:上下文对象的 **Basic** 实现方式
config	里面主要是读取 **XML** 配置文件的一些常用类
ejb	**Workflow** 的 **EJB** 实现方式, 主要对象是 **EJBWorkflow** 和 **EJBWorkflowContext**
loader	在工作流初始化时必须要加载的一些类. 主要有 **WorkflowFactory** 接口及 **XMLWorkflowFactory** 实现类, 还有 **WorkflowDescriptor** 等描述 **XML** 配置文件的实体对象等
ofbiz	**Workflow** 的 **ofbiz** 实现方式, 只有两个对象分别是 **OfbizWorkflow** 和 **OfbizWorkflowContext**
query	**OSWorkflow** 查询分析器, 完成各种单一或者嵌套或者组合查询. 最主要的对象应该是 **WorkflowExpressionQuery**
soap	**Workflow** 的 **soap** 实现方式
spi	与后台打交道的关于 **workflow** 存储的一些类或者对象 有 **WorkflowEntry** 接口, 此接口返回一个工作流实例对象; **WorkflowStore** 接口非常重要, 里面定义的全部都是工作流的存储及查询方法; **JDBCWorkflowStore** 实现 **SimpleWorkflowEntry**; **hibernate** 或者 **spring+hibernate** 实现 **HibernateWorkflowEntry**; **Step** 接口被 **SimpleStep** 或者 **HibernateStep** 调用; **SimpleStep** 是 **JDBCWorkflowStore** 实现方式; **HibernateStep** 是 **HibernateWorkflowStore** 或者 **SpringHibernateWorkflow** 的实现方式
spi.ejb	**EJB** 形式的存储实现方式
spi.hibernate	**Hibernate2** 或者 **spring+hibernate2** 形式的存储实现方式
spi.hibernate3	**Hibernate3** 或者 **spring+hibernate3** 形式的存储实现方式, 目前这种是主流方式，代码我有重大修改，详细敬请下载我的源代码；另外请大家注意：**hibernate** 不支持复合查询
spi.jdbc	**Jdbc** 形式的存储实现方式，非常常见但又非常经典的方式！另外我扩展了一个 **springTemplate** 实现方式, 详细请大家看文

	档的 **JDBCTemplateWorkflowStore** 部分及相应源代码
spi.memory	**Memory** 形式的存储实现方式，例子里面默认就是用的这种方式. 个人认为只可作示例用，不推荐在项目中使用
spi.ofbiz	**Ofbiz** 形式的存储实现方式，非常少见的方式，略
spi.ojb	**org.apache.ojb** 形式的存储实现方式
spi.prevayler	**org.prevayler** 形式的存储实现方式
timer	定时器，用来处理工作流中的定时任务
util	主要是扩展 **OSWorkflow** 用以实现对 **WebWork.jndi,jmail，springframework,Log4j，OSUser** 等等的支持
util.beanshell	**Beanshell** 形式的 **condition,function,registor** 和 **validator** 实现方式
util.bsf	bsf 形式的 **condition,function,registor** 和 **validator** 实现方式
util.ejb	**EJB** 的本地与远程两种形式的 **condition,function,registor** 和 **validator** 实现方式
util.jndi	**jndi** 形式的 **condition,function,registor** 和 **validator** 实现方式

OSWorkflow 数据库的描述

JDBCWorkflowStore 和 JDBCTemplateWorkflowStore 这两种方式所用的表结构完全相同。现将所要用到的表结构开列如下并对每一张表每一字段进行详细说明。在此我不对 hibernate 表结构进行说明，因为有一点点的不同，请自行到我网站上下载源代码进行对比。
注：以下表结构全部以 mysql5.0 为例。

os_currentstep

```
CREATE TABLE `os_currentstep` (
  `ID` bigint(20) NOT NULL,
  `ENTRY_ID` bigint(20) default NULL,
  `STEP_ID` int(11) default NULL,
  `ACTION_ID` int(11) default NULL,
  `OWNER` varchar(35) default NULL,
  `START_DATE` datetime default NULL,
  `FINISH_DATE` datetime default NULL,
```

```
  `DUE_DATE` datetime default NULL,
  `STATUS` varchar(40) default NULL,
  `CALLER` varchar(35) default NULL,
  PRIMARY KEY  (`ID`),
  KEY `ENTRY_ID` (`ENTRY_ID`),
  KEY `OWNER` (`OWNER`),
  KEY `CALLER` (`CALLER`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `os_currentstep`
  ADD FOREIGN KEY (`ENTRY_ID`) REFERENCES `os_wfentry` (`ID`),
  ADD FOREIGN KEY (`OWNER`) REFERENCES `os_user` (`USERNAME`),
  ADD FOREIGN KEY (`CALLER`) REFERENCES `os_user` (`USERNAME`);
```

ID：当前步骤的 ID 号。
ENTRY_ID：工作流程实例 ID，与 os_wfentry 的 ID 相关联。
STEP_ID：当前步骤 ID。
ACTION_ID：当前动作 ID。
OWNER：当前状态下流程所有者，与 os_user 用户 ID 关联。
START_DATE：当前流程开始时间。
FINISH_DATE：当前流程结束时间。
DUE_DATE：好像并没有起什么作用☹(((
STATUS：流程当前状态。
CALLER：当前流程调用者，与 os_user 用户 ID 关联。

os_currentstep_prev

```
CREATE TABLE `os_currentstep_prev` (
  `ID` bigint(20) NOT NULL,
  `PREVIOUS_ID` bigint(20) NOT NULL,
  PRIMARY KEY  (`ID`,`PREVIOUS_ID`),
  KEY `ID` (`ID`),
  KEY `PREVIOUS_ID` (`PREVIOUS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `os_currentstep_prev`
  ADD FOREIGN KEY (`ID`) REFERENCES `os_currentstep` (`ID`),
  ADD FOREIGN KEY (`PREVIOUS_ID`) REFERENCES `os_historystep` (`ID`);
```

ID：当前步骤 ID，与 os_currentstep 的 ID 作关联。
PREVIOUS_ID：当前步骤之前步骤 ID，与 os_historystep 的 ID 作关联。

os_historystep

```
CREATE TABLE `os_historystep` (
   `ID` bigint(20) NOT NULL,
   `ENTRY_ID` bigint(20) default NULL,
   `STEP_ID` int(11) default NULL,
   `ACTION_ID` int(11) default NULL,
   `OWNER` varchar(35) default NULL,
   `START_DATE` datetime default NULL,
   `FINISH_DATE` datetime default NULL,
   `DUE_DATE` datetime default NULL,
   `STATUS` varchar(40) default NULL,
   `CALLER` varchar(35) default NULL,
   PRIMARY KEY    (`ID`),
   KEY `ENTRY_ID` (`ENTRY_ID`),
   KEY `OWNER` (`OWNER`),
   KEY `CALLER` (`CALLER`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `os_historystep`
   ADD FOREIGN KEY (`ENTRY_ID`) REFERENCES `os_wfentry` (`ID`),
   ADD FOREIGN KEY (`OWNER`) REFERENCES `os_user` (`USERNAME`),
   ADD FOREIGN KEY (`CALLER`) REFERENCES `os_user` (`USERNAME`);
```

ID：历史步骤的 ID 号。
ENTRY_ID：工作流程实例 ID，与 os_wfentry 的 ID 相关联。
STEP_ID：历史步骤 ID。
ACTION_ID：历史动作 ID。
OWNER：历史状态下流程所有者，与 os_user 用户 ID 关联。
START_DATE：历史流程开始时间。
FINISH_DATE：历史流程结束时间。
DUE_DATE：好像并没有起什么作用☹(((
STATUS：流程历史状态。
CALLER：历史流程调用者，与 os_user 用户 ID 关联。

os_historystep_prev

```
CREATE TABLE `os_historystep_prev` (
   `ID` bigint(20) NOT NULL,
   `PREVIOUS_ID` bigint(20) NOT NULL,
   PRIMARY KEY    (`ID`,`PREVIOUS_ID`),
   KEY `ID` (`ID`),
   KEY `PREVIOUS_ID` (`PREVIOUS_ID`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `os_historystep_prev`
   ADD FOREIGN KEY (`ID`) REFERENCES `os_historystep` (`ID`),
   ADD FOREIGN KEY (`PREVIOUS_ID`) REFERENCES `os_historystep` (`ID`);
```

ID：历史步骤 ID，与 os_historystep 的 ID 作关联。
PREVIOUS_ID：历史步骤之前步骤 ID，与 os_historystep 的 ID 作关联。

os_wfentry

```
CREATE TABLE `os_wfentry` (
   `ID` bigint(20) NOT NULL,
   `NAME` varchar(60) default NULL,
   `STATE` int(11) default NULL,
   PRIMARY KEY  (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

ID：流程的 ID 号。
NAME：流程名称。
STATE：流程状态。有五种：CREATED = 0;ACTIVATED = 1;SUSPENDED = 2;KILLED = 3;COMPLETED = 4;UNKNOWN = -1。

os_entryids

```
CREATE TABLE `os_entryids` (
   `ID` bigint(20) NOT NULL auto_increment,
   PRIMARY KEY  (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

说明：hibernate 无此表。
ID：由数据库自动生成的 workflow_entry_id。供 os_wfentry 表用。

os_stepids

```
CREATE TABLE `os_stepids` (
   `ID` bigint(20) NOT NULL auto_increment,
   PRIMARY KEY  (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

说明：hibernate 无此表。
ID：由数据库自动生成的 step_id。供 os__currentstep 表或 os_historytstep 表使用。

os_propertyentry

```
CREATE TABLE `os_propertyentry` (
```

```
  `GLOBAL_KEY` varchar(250) NOT NULL,
  `ITEM_KEY` varchar(250) NOT NULL,
  `ITEM_TYPE` tinyint(4) default NULL,
  `STRING_VALUE` varchar(255) default NULL,
  `DATE_VALUE` datetime default NULL,
  `DATA_VALUE` blob,
  `FLOAT_VALUE` float default NULL,
  `NUMBER_VALUE` bigint(10) default NULL,
  PRIMARY KEY  (`GLOBAL_KEY`,`ITEM_KEY`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

说明：hibernate 的 os_ propertyentry 表与此表有点差别。
GLOBAL_KEY：全局变量名，如果为 JDBCWorkflowStore 即为 osff_加上流程实例 ID；如果为 JDBCTemplateWorkflowStore 即为 osff_temp_加上流程实例 ID；
ITEM_KEY：局部变量名，通常是指真正要在程序中调用的”key”。
ITEM_TYPE：以数字标明其 property 的类型。如 5 是 String 型。
DATE_VALUE：如果是 Date 类型的数据，存入这个字段里。
DATA_VALUE：如果是 Data 类型的数据，存入这个字段里。
FLOAT_VALUE：如果是 Float 类型的数据，存入这个字段里。
NUMBER_VALUE：如果是 Number 类型的数据，存入这个字段里。

os_user

```
CREATE TABLE `os_user` (
  `USERNAME` varchar(100) NOT NULL,
  `PASSWORDHASH` mediumtext,
  PRIMARY KEY  (`USERNAME`),
  KEY `USERNAME` (`USERNAME`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

USERNAME：用户名，主键。
PASSWORDHASH：用户密码。

os_group

```
CREATE TABLE `os_group` (
  `GROUPNAME` varchar(20) NOT NULL,
  PRIMARY KEY  (`GROUPNAME`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

GROUPNAME：群组名，主键

os_membership

```
CREATE TABLE `os_membership` (
```

```
  `USERNAME` varchar(20) NOT NULL,
  `GROUPNAME` varchar(20) NOT NULL,
  PRIMARY KEY  (`USERNAME`,`GROUPNAME`),
  KEY `USERNAME` (`USERNAME`),
  KEY `GROUPNAME` (`GROUPNAME`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

USERNAME：用户名，主键。
GROUPNAME：群组名，主键

```
ALTER TABLE `os_membership`
  ADD FOREIGN KEY (`USERNAME`) REFERENCES `os_user` (`USERNAME`),
  ADD FOREIGN KEY (`GROUPNAME`) REFERENCES `os_group` (`GROUPNAME`);
```

以上三张表与 OSWorkflow 核心没有多大的关系，这是标准而经典的权限模式，用户和群组是多对多的关系，我不想做深入讨论。尤其要注意的是：如果不用这三张表，osuser.xml 可以不用配置，但在 XML 流程定义文件里就不能再引用 **com.opensymphony.workflow.util.OSUserGroupCondition** 了，切记切记！
有人要问，如果不用那用户和权限怎么办？我想这恰巧就是 OSWorkflow 的灵活之处，建立自己的用户和权限表就 OK 了，想怎么建都成啊。呵呵……
Hibernate 的数据库我就不多说了，大同小异，不过和以上的表还是有点差别的。自己琢磨吧☺
一点补充：在我的网站http://cucuchen520.javaeye.com/
上面的 osworkflow2.8.rar 里面有全套的代码及数据库 script 文件和配置文件。详情请您自行下载。

OSWorkflow 核心代码剖析

osworkflow.xml 加载过程

1) 在 AbatractWorkflow 中的 getConfiguration()方法调用了 Configuration 中的 load(null)方法
2) Configuration 中的 load(null)方法调用了 getInputStream()方法。

getInputStream()方法里面加载 osworkflow.xml 的顺序可由源代码看出端倪，而方法的注释已经说得再清楚不过了，我就不再赘述了。

```
/**
     * Load the default configuration from the current context classloader.
     * The search order is:
     * <li>Specified URL</li>
     * <li>osworkflow.xml</li>
     * <li>/osworkflow.xml</li>
     * <li>META-INF/osworkflow.xml</li>
     * <li>/META-INF/osworkflow.xml</li>
     */
    protected InputStream getInputStream(URL url) {
```

```
        InputStream is = null;

        if (url != null) {
            try {
                is = url.openStream();
            } catch (Exception ex) {
            }
        }

        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("osworkflow.xml");
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("/osworkflow.xml");
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("META-INF/osworkflow.xml");
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("/META-INF/osworkflow.xml");
            } catch (Exception e) {
            }
        }

        return is;
    }
```

3) 在 load(null)方法中调用流程描述文件的工厂并进行实例化并将其初始化。代码如下：

```
factory = (WorkflowFactory) ClassLoaderUtil.loadClass(clazz, getClass()).newInstance();
```

```
                    Properties properties = new Properties();
                    List props = XMLUtil.getChildElements(factoryElement, "property");

                    for (int i = 0; i < props.size(); i++) {
                        Element e = (Element) props.get(i);
                        properties.setProperty(e.getAttribute("key"),
e.getAttribute("value"));
                    }

                    factory.init(properties);
                    factory.initDone();
```

4) 我们再来看看在 factory 的 initDone()方法中到底干了些什么。其实主要是获取到 osworkflows.xml 的名字并进行解析，代码如下：

```
String name = getProperties().getProperty("resource", "workflows.xml");
        InputStream is = getInputStream(name);
```

5) factory 中的 getInputStream(name)方法代码如下，可以看出它寻找 osworkflow.xml 的顺序。

```
/**
     * Load the workflow config file from the current context classloader.
     * The search order is:
     * <li>Specified URL</li>
     * <li>&lt;name&gt;</li>
     * <li>/&lt;name&gt;</li>
     * <li>META-INF/&lt;name&gt;</li>
     * <li>/META-INF/&lt;name&gt;</li>
     */
    protected InputStream getInputStream(String name) {
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
        InputStream is = null;

        if ((name != null) && (name.indexOf(":/") > -1)) {
            try {
                is = new URL(name).openStream();
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream(name);
            } catch (Exception e) {
            }
        }
```

```
        if (is == null) {
            try {
                is = classLoader.getResourceAsStream('/' + name);
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("META-INF/" + name);
            } catch (Exception e) {
            }
        }

        if (is == null) {
            try {
                is = classLoader.getResourceAsStream("/META-INF/" + name);
            } catch (Exception e) {
            }
        }

        return is;
    }
```

6) initDone()在找到流程定义文件之后，一个非常重要的任务便是往 workflows(HashMap）中加入以流程名称为参数的静态内部类 WorkflowConfig(其实也就是 workflows.xml 中配置的 workflow，它有三个参数：name，type，loaction)。这个 WorkflowConfig 将会为下面 WorkflowDescriptor 对象的加载作好充分地准备。

WorkflowDescriptor 对象加载过程

1) 首先在 Workflow 接口中 getWorkflowDescriptor(String workflowName)方法调用 Configuration 对象中的 getWorkflow(workflowName)方法。
2) Configuration 对象调用 factory 中的 getWorkflow(name)方法。
3) factory 中的 getWorkflow(name)方法调用同类中的 getWorkflow(name, true)方法。
4) getWorkflow(name, true)方法首先用 name 参数取得 WorkflowConfig 对象，这个 WorkflowConfig 对象在 factory 初始化的时候已经被生成了。紧接着调用 loadWorkflow(c, validate)方法去解析 XML 并最终生成 WorkflowDescriptor 对象。

WorkflowStore 对象加载过程

1) 首先 AbatractWorkflow 中的 getPersistence()方法调用 Configuration 对象中的 getWorkflowStore()方法。

2) 如果 store 为空，从预先用 load(URL url)加载的 persistenceClass 中取得实现类，然后再创建新的实例并初始化参数，最后返回 WorkflowStore 对象。详细代码如下：

```
public WorkflowStore getWorkflowStore() throws StoreException {
        if (store == null) {
            String clazz = getPersistence();

            try {
                store = (WorkflowStore) Class.forName(clazz).newInstance();
            } catch (Exception ex) {
                throw new StoreException("Error creating store", ex);
            }

            store.init(getPersistenceArgs());
        }

        return store;
    }
```

Workflow 接口中的核心方法

initialize 方法

初始化方法主要完成了以下几个功能

- 得到 WorkflowStore 实现类，利用里面的 createEntry 方法创建一个 WorkflowEntry 对象。
- 执行 populateTransientMap 方法，将 context(WorkflowContext),entry(WorkflowEntry),store(WorkflowStore), configuration(Configuration), descriptor(WorkflowDescriptor)装进 transientVars；将当前要执行的 actionId 和 currentSteps 装进 transientVars；将所有 XML 中配置的 register 装进 transientVars。
- 根据 restrict-to 里面配置的条件来判断是否能初始化工作流，如果不能则回滚并抛出 InvalidRoleException 异常。
- 执行 transitionWorkflow 方法传递工作流，这个 transitionWorkflow 方法是 Workflow 中重中之重的方法，以下会进行详细讲解。
- 返回当前工作流程实例 ID。

transitionWorkflow 方法

transitionWorkflow 方法是工作流最最核心的方法，它主要是完成以下功能

- 利用 getCurrentStep 方法取得当前步骤：如果只有一个有效当前步骤，直接返回；如果有多个有效当前步骤，返回符合条件的第一个。
- 调用动作验证器来验证 transientVars 里面的变量。
- 执行当前步骤中的所有 post-function。

- 执行当前动作中的所有 pre-function。
- 检查当前动作中的所有有条件结果，如果有符合条件的，验证 transientVars 里面的变量并执行有条件结果里面的 pre-function；如果动作里面没有一个有条件结果，执行无条件结果，验证 transientVars 里面的变量并执行无条件结果里面的 pre-function。
- 如果程序进入到一个 split 中：1）验证 transientVars 里面的变量；2）执行 split 中的 pre-functions；3）如果动作中的 finish 不等于 true，执行 split 中所有的 result；4)结束当前步骤并将其移至历史步骤中去，创建新的步骤并执行新步骤中的 pre-function；5)执行 split 中的 post-function。
- 如果程序进入到一个 join 中：1) 结束当前步骤并将其移至历史步骤中去；2)将刚结束的步骤和在 join 中的当前步骤还有历史步骤加到 joinSteps 集合中并产生 JoinNodes 对象，将此对象 put 到 transientVars 里；3）检查 join 条件；4）执行 join 有条件结果中的 validator；5)执行 join 有条件结果中的 pre-function；6)如果当前步骤不在历史步骤里面，把它移到历史步骤里面去；7) 如果刚刚结束的当前动作中的 finish 不等于 true，创建新的步骤并执行新步骤中的 pre-function；8）执行 join 中的 post-function。
- 如果程序进入到另一个 step 中：结束当前步骤并将其移至历史步骤中去，创建新的步骤并执行新步骤中的 pre-function。
- 如果动作里面有符合条件的有条件结果，执行有条件结果里面的 post-function；如果动作里面没有一个有条件结果，则执行无条件结果里面的 post-function。
- 执行动作里面的 post-function。
- 如果动作一开始是一个初始状态，将设置 ACTIVATED 标识；如果动作在 XML 里面有完成状态的标识，将设置 COMPLETED 标识。
- 执行有效的自动动作(auto action)。
- 最后返回流程是否完成的布尔值：如果流程实例已经完结，返回 true；否则返回 false。

doAction 方法

- 判断工作流程实例的状态，如果状态不为 ACTIVATED(1)，直接返回。
- 利用 findCurrentSteps 方法得到当前所有步骤列表。
- 执行 populateTransientMap 方法，将 context(WorkflowContext),entry(WorkflowEntry),store(WorkflowStore), configuration(Configuration), descriptor(WorkflowDescriptor)装进 transientVars；将当前要执行的 actionId 和 currentSteps 装进 transientVars；将所有 XML 中配置的 register 装进 transientVars。
- 检查全局动作(Global Action)和当前步骤里面所有动作的有效性。如果有无效动作，直接抛出 InvalidActionException 异常。
- 执行 transitionWorkflow 方法传递工作流，如果捕获到 WorkflowException，抛出异常并回滚。
- 如果动作中没有显式地标明 finish 的状态为 true，那么这时要执行 checkImplicitFinish 方法，查找当前步骤中是否还有有效动作，如果没有一个有效动作，则直接调用 completeEntry 方法结束流程并将流程的状态设置成为 COMPLETED(4)。

如何与现有系统集成

有很多网友问我如何将 OSWorkflow 集成到现有的系统(常见的如 OA)中去，其实这是一件非常容易的事情，在我翻译的《OSWorkflow 中文手册》工作流程思想章节<和抽象实体的集成>中有所提及，在此我更详细的解释如下：

在初始化一个新的工作流时，必须要在你的 Service 层执行以下方法：

```
public long doInitialize(String un, String title, String content) throws Exception {
        Workflow wf = new BasicWorkflow(un);
        long wf_id = -1;
        try {
            wf_id = wf.initialize("example", 100, null);
        }
        catch (Exception e) {
            throw e;
        }
        return wf_id;
    }
```

这时候要加入自己的业务逻辑代码，例如：

```
workflowDAO.saveDocumentation(wf_id, title, content);
```

os_doc 有三个字段： wf_id（非常重要，绑定工作流 ID，主键）；title(文档标题),要从创建工作流的前台 newdoc.jsp 中传过来；content（文档内容）也要从创建工作流的前台 newdoc.jsp 中传过来。

newdoc.jsp 中的 form 表单提供三个参数：用户名；文档标题和文档内容；另加一个提交按钮提到后台处理，代码如下：

```
<%@ page contentType="text/html; charset=GB2312" %>
<%
    String un = request.getParameter("un");
%>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="/css/style.css">
</head>

<body>
<table width="621" height="248" cellpadding="0" cellspacing="0" bordercolor="#0000FF" border="1">
        <form name="form1" method="post" action="newworkflow.jsp">
            <input type="hidden" name="un" value="<%=un%>">
            <tr>
                <th height="18" colspan="2">
                    <div align="left">创建文档</div>
                </th>
            </tr>
```

```
                <tr>
                    <td width="189">文档标题：</td>
                    <td width="422" height="18"><input type="text" name="title"></td>
                </tr>
                <tr>
                    <td>文档内容：</td>
                    <td height="35"><textarea name="content" cols="50"
rows="8"></textarea></td>
                </tr>
                <tr>
                    <td height="18" colspan="2">
                        <div align="center"><input type="submit" value="提交"></div>
                    </td>
                </tr>
            </form>

        </table>
        <jsp:include page="nav.jsp" flush="true">
            <jsp:param name="un" value="<%=un%>"/>
        </jsp:include>
        </body>
        </html>
```

后台代码应该如下：

Serive 部分：

```
public long doInitialize(String un, String title, String content) throws Exception {
        Workflow wf = new BasicWorkflow(un);
        long wf_id = -1;
        try {
            wf_id = wf.initialize("example", 100, null);
            this.workflowDAO.saveDocumentation(wf_id, title, content);
        }
        catch (Exception e) {
            throw e;
        }
        return wf_id;
    }
```

由以上可以看出：它初始化了一个流程，并且将从前台传过来的文档进行了保存操作。

DAO 部分：

```
public void saveDocumentation(long wf_id, String title, String content) throws
        DataAccessException {
        this.getJdbcTemplate().update("insert into os_doc (wf_id,title,content) values
```

```
                    (?,?,?)",
                    new Object[]{wf_id, title, content});
        }
```

在 Action 中的代码如下：

```
String un = request.getParameter("un");
        String title = request.getParameter("title");
        String content = request.getParameter("content");

        ApplicationContext cxt =
WebApplicationContextUtils.getWebApplicationContext(this.getServletConfig().getServletContex
t());
        WorkflowService workflowService = (WorkflowService)
cxt.getBean("workflowService");



        boolean bSuccess = false;
        long id = 0;
        try {
            id = workflowService.doInitialize(un, title, content);
            bSuccess = true;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
```

可以看出，它主要是接收参数，然后调用 Service 中的 doInitialize 方法初始化工作流。

在进行文档传输的时候，集成的方法和上面类似，如下：

test.jsp 中的 form 表单提供五个参数：用户名，流程 ID，动作 ID，文档标题和文档内容,**具体表单代码如下：**

```
<table width="621" height="248" cellpadding="0" cellspacing="0" bordercolor="#0000FF"
border="1">
        <form name="form1" method="post" action="test.jsp">
            <input type="hidden" name="un" value="<%=un%>">
            <input type="hidden" name="id" value="<%=id%>">
            <input type="hidden" name="do" value="" id="do">
            <tr>
                <th height="18" colspan="2">
                    <div align="left">修改文档</div>
                </th>
            </tr>
            <tr>
                <td width="189">文档标题：</td>
```

```
                    <td width="422" height="18"><input type="text" name="title"
value="<%=vo.getTitle()==null?"":vo.getTitle()%>"></td>
                </tr>
                <tr>
                    <td>文档内容：</td>
                    <td height="35"><textarea name="content" cols="50"
rows="8"><%=vo.getContent()==null?"":vo.getContent()%></textarea></td>
                </tr>
            </form>

    </table>
```

其中 title，content 如何得来？原来，在前转到 test.jsp 之前，要在 Action 中预置好文档的有关信息供 test.jsp 中 form 表单调用。

在 Action 有以下代码：

```
    ApplicationContext cxt =
            WebApplicationContextUtils.getWebApplicationContext(this.getServletConfig().
            getServletContext());
    WorkflowService workflowService = (WorkflowService)cxt.getBean("workflowService");
    DocumentationVO vo = workflowService.getDocByWorkflowId(id);
```

Service 中的代码：

```
    public DocumentationVO getDocByWorkflowId(long wf_id) throws DataAccessException {
            return this.workflowDAO.getDocByWorkflowId(wf_id);
        }
```

DAO 中的代码：

```
    public DocumentationVO getDocByWorkflowId(long wf_id) throws DataAccessException {
            List list =
                    this.getJdbcTemplate().queryForList("select * from os_doc where wf_id
= ?", new Object[]{wf_id});
            DocumentationVO vo = new DocumentationVO();
            if (list != null && !list.isEmpty()) {
                Map map = (Map) list.get(0);
                vo.setWf_id(wf_id);
                vo.setTitle((String) map.get("title"));
                vo.setContent((String) map.get("content"));
            }
            return vo;
        }
```

DocumentationVO 代码如下：

```
    package com.opensymphony.workflow.vo;

    /**
     * @author chris.chen
```

```
 */
public class DocumentationVO {
    private long wf_id;
    private String title;
    private String content;

    public long getWf_id() {
        return wf_id;
    }

    public void setWf_id(long wf_id) {
        this.wf_id = wf_id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
}
```

这时如果执行了某个动作按钮，将提交数据到 Action 中进行处理，Action 将执行以下代码：

```
//动作 ID
String doString = request.getParameter("do");
if (doString != null && !doString.equals("")) {
        int action = Integer.parseInt(doString);
        workflowService.doAction(wf, id, action, title, content);
    }
//此时要重新从数据库中获得 DocumentationVO 对象，返回到 test.jsp 中显示的才是修改过后的数据。
    DocumentationVO vo = workflowService.getDocByWorkflowId(id);
```

而此时 Service 中到底做了些什么事情呢？代码如下：

```
public void doAction(Workflow wf, long wf_id, int action_id, String title, String content)
```

```
throws
                Exception {
        try {
                wf.doAction(wf_id, action_id, null);
                this.workflowDAO.updateDocumentation(wf_id, title, content);
        }
        catch (Exception e) {
                throw e;
        }
    }
```

由以上可以看出：它执行了一个动作，并且将从前台传过来的文档进行了更新操作。

DAO 中的代码如下：

```
public void updateDocumentation(long wf_id, String title, String content) throws
DataAccessException {
        this.getJdbcTemplate().update("update os_doc set title =?,content =?where wf_id
=?",
                new Object[]{title, content, wf_id});
    }
```

上述代码实现了如何绑定文档并进行操作，但一个完整的 OA 系统是有流程审批意见的，**如何加上审批意见呢？**

首先，要在数据库中建一张名为 os_doc_opinion 的表，表里面有六个字段：id(主键，自增长)，entry_id（流程实例编号），action_id（动作编号），caller（调用者），opinion（流程审批意见），opinion_time(审批时间)。表结构如下(mysql5.0 为例)：

```
CREATE TABLE `os_doc_opinion` (
  `ID` bigint(20) NOT NULL auto_increment,
  `ENTRY_ID` bigint(20) default NULL,
  `ACTION_ID` int(11) default NULL,
  `CALLER` varchar(35) default NULL,
  `OPINION` text,
  `OPINION_TIME` datetime default NULL,
  PRIMARY KEY  (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

接着，在前台 test.jsp 的提交表单中加上审批意见：

```
<tr>
  <td>审批意见：</td>
  <td height="35"><textarea name="opinion" cols="60" rows="8"></textarea></td>
</tr>
```

并且要把当前调用者也要传到后台去，那么要在 form 表单中将其加入到 hidden 域中，如下：

```
<input type="hidden" name="un" value="<%=un%>">
```

这时候，后台的 doAction 这个 service 也要作一些改变：

```
//多了两个参数：un(当前调用者)和 opinion(流程审批意见)。
public void doAction(Workflow wf, long wf_id, int action_id, String un, String title, String content, String opinion) throws
        Exception {
    try {
//执行工作流的传递
        wf.doAction(wf_id, action_id, null);
//对文档进行更新操作
        this.workflowDAO.updateDocumentation(wf_id, title, content);
//加入了对流程审批意见进行操作的方法
//将流程实例 ID，动作 ID，调用者，意见保存到 os_doc_opinion 中去
        this.workflowDAO.saveDocumentationOpinion(wf_id, action_id, un, opinion);
    }
    catch (Exception e) {
        throw e;
    }
}
```

最后，可以在前台的 test.jsp 中，根据工作流程 ID 去调用与之相关的所有审批意见列表：

```
List opinions = workflowService.getDocumentationOpinions(id);
        if (opinions != null && !opinions.isEmpty()) {
            for (int i = 0; i < opinions.size(); i++) {
                DocumentationOpinionVO dovo = (DocumentationOpinionVO) opinions.get(i);
                out.println("动作：" + wd.getAction(dovo.getAction_id()).getName() + "(" + dovo.getAction_id() + ")");
                out.println("调用者：" + dovo.getCaller());
                out.println("审批意见：" + dovo.getOpinion());
                out.println("审批时间:" + dovo.getOpinion_time());
                out.println("<br>");
            }
        }
```

由以上示例可以看出，绑定现有系统决非难事。如果大家觉得文档写得比较抽象，想继续弄懂其原理，可以到http://cucuchen520.javaeye.com去下载osworkflow2.8_bundle2.rar全套绑定现有系统的代码。

当前调用者如何取得任务列表

这个问题看起来很简单但却一直困惑着广大开发者，解决办法就是：取得所有流程实例集合

以后，再执行 getAvailableActions 方法。
由于所有实例都放在 os_wfentry 这张表中。那么，得到实例 ID 的 SQL 为：
SELECT ID FROM OS_WFENTRY WHERE STATE=1 AND NAME = 'example'（假设流程名称为 example）
取出流程实例 ID 集合以后，再调用 Workflow 接口中的 getAvailableActions(id,null)方法即可取得当前调用者所有任务(有效动作)列表。

OSWorkflow 高级功能

全局条件

global-conditons 的作用范围是动作：包括一般动作，全局动作和被调用的通用动作。它和 global-actions 类似，一旦被定义，就将被所有的动作隐式地调用。对于全局性的东西，我建议大家小心谨慎地运用，免得带来不必要的麻烦。

全局动作

global-actions 是全局动作，它定义在 initial-actions 标签和 steps 标签的中间，一旦被定义，就将被所有的步骤隐式地调用。它最典型的应用就是终止当前正在运行的流程实例，XML 配置如下：

```
<global-actions>
        <action id="1000" name="终止流程" finish="true">
            <results>
                <unconditional-result old-status="Finished" status="Underway" step="-1"/>
            </results>
        </action>
</global-actions>
```

通用动作

common-actions 定义在 global-actions 的后面，它不会被所有步骤隐式的调用到，步骤里应该显式地调用它，它和一般 action 的位置没有先后顺序，可以放在其前，也可放在其后，也可放在中间。

```
<common-actions>
        <action id="1001" name="发布通知">
            <results>
                <unconditional-result old-status="Finished" status="Underway" step="-1">
                    <post-functions>
                        <function type="beanshell">
                            <arg name="script">
                                System.out.println("发布通知。。。");
```

```
                                    </arg>
                                </function>
                            </post-functions>
                        </unconditional-result>
                    </results>
                </action>
            </common-actions>
……..
<actions>
  <common-action id="1001"/>
  ……
</actions>
```

自动动作

只要满足条件，动作将会自动执行(auto action)。如果有两个自动动作都符合条件，优先执行配置在前面的自动动作；如果说上下两个相连的步骤里都有符合条件的自动动作，且上一步骤里面自动动作的结果指向下一步骤，将连续执行；如果自动动作执行以后步骤不发生改变，可以手动执行其它未执行的非自动动作；如果自动动作执行以后步骤发生改变，将直接跳到结果里所指向的相应步骤。

以下是一个典型的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow PUBLIC "-//OpenSymphony Group//DTD OSWorkflow 2.6//EN" "http://www.opensymphony.com/osworkflow/workflow_2_6.dtd">
<workflow>
    <initial-actions>
        <action id="100" name="Start Workflow">
            <pre-functions>
                <function type="class">
                    <arg name="class.name">
                        com.opensymphony.workflow.util.Caller
                    </arg>
                </function>
            </pre-functions>
            <results>
                <unconditional-result old-status="Finished" status="Underway" step="1"
                  owner="${caller}"/>
            </results>
        </action>
    </initial-actions>
    <steps>
        <step id="1" name="First Step">
            <actions>
                <action id="1" name="The first action">
```

```
                    <results>
                        <unconditional-result old-status="Finished" status="Queued"
                         step="2"/>
                    </results>
                </action>
                <action id="2" name="The second action" auto="true">
                    <restrict-to>
                        <conditions type="AND">
                            <condition type="class">
                                <arg name="status">Underway</arg>
                                <arg name="class.name">
                                    com.opensymphony.workflow.util.StatusCondition
                                </arg>
                            </condition>
                        </conditions>
                    </restrict-to>
                    <results>
                        <unconditional-result old-status="Finished" status="Queued"
                          step="1"/>
                    </results>
                </action>
            </actions>
        </step>
        <step id="2" name="Sencond Step">
            <actions>
                <action id="3" name="The third action">
                    <restrict-to>
                        <conditions type="AND">
                            <condition type="class">
                                <arg name="status">Queued</arg>
                                <arg name="class.name">
                                    com.opensymphony.workflow.util.StatusCondition
                                </arg>
                            </condition>
                        </conditions>
                    </restrict-to>
                    <results>
                        <unconditional-result old-status="Finished" status="Finished"
                          step="2"/>
                    </results>
                </action>
            </actions>
        </step>
    </steps>
```

```
</workflow>
```

发送邮件

OSWorkflow 本身在工具函数里面内置了 SendEmail 的功能，不过它这个没有实现认证接口，比较完善的邮件系统必须要经过认证。所以我也扩展了一下它。注意：必须要在 lib 包里面加入提供 jmail 支持的包：activation.jar 和 mail.jar。
具体代码如下，只要是我修改的都加有注释。

```
/*
 * Copyright (c) 2002-2003 by OpenSymphony
 * All rights reserved.
 */
package com.opensymphony.workflow.util;

import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.workflow.FunctionProvider;
import com.opensymphony.workflow.config.Configuration;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.*;


/**
 * Sends an email to a group of users. The following arguments are expected:
 * <p/>
 * <ul>
 * <li>to - comma seperated list of email addresses</li>
 * <li>from - single email address</li>
 * <li>subject - the message subject</li>
 * <li>cc - comma seperated list of email addresses (optional)</li>
 * <li>message - the message body</li>
 * <li>smtpHost - the SMTP host that will relay the message</li>
 * <li>parseVariables - if 'true', then variables of the form ${} in subject,
 * message, to, and cc fields will be parsed</li>
 * </ul>
 *
 * @author <a href="mailto:plightbo@hotmail.com">Pat Lightbody</a>
```

```
 */
public class OSSendEmail implements FunctionProvider {
    //~ Static fields/initializers /////////////////////////////////////////////

    private static final Log log = LogFactory.getLog(SendEmail.class);

    //~ Methods ////////////////////////////////////////////////////////////////

    public void execute(Map transientVars, Map args, PropertySet ps) {
        String to = (String) args.get("to");
        String from = (String) args.get("from");
        String subject = (String) args.get("subject");
        String cc = (String) args.get("cc");
        String m = (String) args.get("message");
        String smtpHost = (String) args.get("smtpHost");
        boolean parseVariables = "true".equals(args.get("parseVariables"));
        Configuration config = (Configuration) transientVars.get("configuration");

        //add by chris 在这里增加了两个用于用户验证的参数
        String username = (String) args.get("username");
        String password = (String) args.get("password");
        try {
            Properties props = new Properties();
            props.put("mail.smtp.host", smtpHost);
            //add by chris 增加认证
            props.put("mail.smtp.auth", "true");

            Session sendMailSession = Session.getInstance(props, new
MyAuthenticator(username, password));
            Transport transport = sendMailSession.getTransport("smtp");
            Message message = new MimeMessage(sendMailSession);

            message.setFrom(new InternetAddress(from));

            Set toSet = new HashSet();
            VariableResolver variableResolver = config.getVariableResolver();
            StringTokenizer st = new StringTokenizer(parseVariables ?
variableResolver.translateVariables(to, transientVars, ps).toString() : to, ", ");

            while (st.hasMoreTokens()) {
                String user = st.nextToken();
                toSet.add(new InternetAddress(user));
            }
```

```
                message.setRecipients(Message.RecipientType.TO,          (InternetAddress[])
toSet.toArray(new InternetAddress[toSet.size()]));

                Set ccSet = null;

                if (cc != null) {
                    ccSet = new HashSet();

                    if (parseVariables) {
                        cc = variableResolver.translateVariables(cc, transientVars, ps).toString();
                    }

                    st = new StringTokenizer(cc, '', '');

                    while (st.hasMoreTokens()) {
                        String user = st.nextToken();
                        ccSet.add(new InternetAddress(user));
                    }
                }

                if ((ccSet != null) && (ccSet.size() > 0)) {
                    message.setRecipients(Message.RecipientType.CC,          (InternetAddress[])
ccSet.toArray(new InternetAddress[ccSet.size()]));
                }

                message.setSubject(parseVariables   ?   variableResolver.translateVariables(subject,
transientVars, ps).toString() : subject);
                message.setSentDate(new Date());
                message.setText(parseVariables        ?        variableResolver.translateVariables(m,
transientVars, ps).toString() : m);
                message.saveChanges();

                transport.connect();
                //transport.sendMessage(message, message.getAllRecipients());
                //add by chris    将发送邮件的方法作了改变如下
                transport.send(message);
                transport.close();
            } catch (MessagingException e) {
                log.error("Error sending email:", e);
            }
        }

        /**
         * 增加了一个认证内部类
```

```
 */
class MyAuthenticator extends javax.mail.Authenticator {

    private String user;
    private String password;

    public MyAuthenticator() {

    }

    public MyAuthenticator(String user, String password) {
        this.user = user;
        this.password = password;
    }

    protected PasswordAuthentication getPasswordAuthentication() {

        return new PasswordAuthentication(user, password);
    }
}
}
```

在 XML 里面的定义如下，在这里我定义在 common-actions 里面：

```
<common-actions>
    <action id="1001" name="发送邮件">
        <results>
            <unconditional-result old-status="Finished" status="Underway" step="-1">
                <post-functions>
                    <function type="class">
                        <arg name="class.name">
                            com.opensymphony.workflow.util.OSSendEmail
                        </arg>
                        <arg name="from">j2eemail@163.com</arg>
                        <arg name="to">cucuchen520@163.com</arg>
                        <arg name="cc">cucuchen520@yahoo.com.cn</arg>
                        <arg name="subject">test email</arg>
                        <arg name="message">email content</arg>
                        <arg name="smtpHost">smtp.163.com</arg>
                        <arg name="parseVariables">false</arg>
                        <arg name="usernusame">j2eemail</arg>
                        <arg name="password">000000</arg>
                    </function>
                </post-functions>
```

```
                </unconditional-result>
            </results>
        </action>
    </common-actions>
```

参数解释：

from:发件箱
to:收件箱
cc:抄送，可以为空
subject:主题
message:内容
smtpHost:smtp 邮件服务器
parseVariables:是否支持参数变量，如果为 true 表示支持
username:发件者用户名
password:发件者密码

注册器

注册器最典型的应用是用于记录日志，这个也在工具包里面内置了，配置如下：

```
<registers>
        <register type="class" variable-name="log">
            <arg name="class.name">com.opensymphony.workflow.util.LogRegister</arg>
            <arg name="addInstanceId">true</arg>
            <arg name="Category">test</arg>
        </register>
</registers>
```

addInstanceId 是否在日志中加上流程实例 ID，可选，默认为 false。
Category 是分类名称，可选。
可以在 function 或 condition 里面直接调用注册器，如下：

```
<function type="beanshell">
        <arg name="script">
            transientVars.get("log").info("Initiate Work");
        </arg>
</function>
```

触发器

申明一个触发器非常简单，只需要在 initial-actions 的上面定义一个触发器函数便可以了，举例如下：

```
<trigger-functions>
        <trigger-function id="10">
            <function type="beanshell">
                <arg name="script">
```

```
                    propertySet.setString("testTrigger", "blahblah");
                </arg>
            </function>
        </trigger-function>
</trigger-functions>
```

触发器需要使用 Workflow 对象里面执行触发器的方法才能够被触发，调用方法如下：

```
Workflow wf = new BasicWorkflow(username);
    wf.setConfiguration(conf);
    long id = Long.parseLong(request.getParameter("id"));
    try {
        wf.executeTriggerFunction(id, 10);
        out.println("execute ok!");
    } catch (Exception e) {
        e.printStackTrace();
    }
```

executeTriggerFunction()方法的第一个参数为实例 id,第二个参数为触发器 id。

定时器

定时器与触发器关系非常密切。注意此功能需要 quartz 的支持，要把 osworkflow 解压包 %osworkflow %\lib\ optional \quartz.jar 加到 lib 里面去。(提示：这个 quartz.jar 为 1.3 版，在 org.quartz 里面没有 quartz.properties，这时需要在 classes 目录下面手工加一个 quartz.properties，如果将 quartz.jar 升级到高版本则不用。)

可以在初始化动作里面申明一个 sheduler，方法如下：

```
<initial-actions>
        <action id="1" name="Start Workflow">
            <pre-functions>
                <function type="class">
                    <arg name="class.name">
                        com.opensymphony.workflow.util.ScheduleJob
                    </arg>
                    <arg name="jobName">Scheduler Test</arg>
                    <arg name="triggerName">SchedulerTestTrigger</arg>
                    <arg name="triggerId">10</arg>
                    <arg name="schedulerStart">true</arg>
                    <arg name="local">true</arg>
                    <arg name="cronExpression">0/5 * * * * ?</arg>
                </function>

                <function type="class">
                    <arg name="class.name">
                        com.opensymphony.workflow.util.Caller
                     </arg>
                </function>
```

```
            </pre-functions>
            <results>
                <unconditional-result old-status="Finished" status="Underway" step="1"
                 owner="${caller}"/>
            </results>
        </action>
    </initial-actions>
```

同时要在 initial-actions 的上面加上触发器，代码与上面触发器小节里面的代码完全一样。其实定时器真正执行的是配置在里面的 trigger，此例执行 triigerId 为 10 的触发器。

参数解释：

jobName：任务名称

triggerName：触发器名称

triggerId：触发器 ID

schedulerStart：是否开启定时器。如果为 false，当执行了这个 function 也不会运行

local：是否为本地服务

cronExpression：定时器的定时规则，本例为每五秒执行一次。详情请查阅 quartz 文档

WEB-INF/classes 目录里面 quartz.properties 的配置如下：

```
#============================================================================
# Configure Main Scheduler Properties
#============================================================================
org.quartz.scheduler.instanceName=QuartzScheduler
org.quartz.scheduler.instanceId=AUTO
#============================================================================
# Configure ThreadPool
#============================================================================
org.quartz.threadPool.class=org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount=5
org.quartz.threadPool.threadPriority=5
#============================================================================
# Configure JobStore
#============================================================================
org.quartz.jobStore.misfireThreshold=60000
org.quartz.jobStore.class =org.quartz.simpl.RAMJobStore
```

您可以根据自己的实际需要进行更改，不过 quartz 相对来说用得比较少，所以还需要朋友们多多实践，并多多查阅 quartz 官方文档☺

验证器

验证器的作用是验证 Input Map，详见 Input Map 章节。可以在 step,或者 result, unconditional-result 的里面加入验证器，位置是在 pre-functions 的前面，我们可以自定义一个验证器，其配置举例如下：

```
<action id="1" name="Initial Work">
```

```
                <restrict-to>
                    <conditions>
                        <condition type="class">
                            <arg name="class.name">
                                com.opensymphony.workflow.util.StatusCondition
                            </arg>
                            <arg name="status">Queued</arg>
                        </condition>
                    </conditions>
                </restrict-to>
                <validators>
                    <validator type="class">
                        <arg name="class.name">
                            com.simple.validators.TitleValidator
                        </arg>
                    </validator>
                </validators>
                <pre-functions>
                    <function type="class">
                        <arg name="class.name">
                            com.opensymphony.workflow.util.Caller
                        </arg>
                        <arg name="stepId">1</arg>
                    </function>
                    <function type="beanshell">
                        <arg name="script">
                            transientVars.get("log").info("Initiate Work");
                        </arg>
                    </function>
                </pre-functions>
                <results>
                    <unconditional-result old-status="Finished" status="Prepared"
                     step="1" owner="${caller}"/>
                </results>
                <post-functions>
                    <function type="beanshell">
                        <arg name="script">
                            propertySet.setString("title",
                            (String)transientVars.get("title"));
                        </arg>
                    </function>
                </post-functions>
            </action>
```

请注意 TitleValidator 验证器，它的作用是验证输入的 title 是否合法，下面是输入部分代码：

```
Map inputs = new HashMap();
inputs.put("docTitle", request.getParameter("title"));
wf.initialize("workflowName", 1, inputs);
```

而在 TitleValidator 这个类里面，实现真正的验证功能，注意一定要实现 Validator 接口：

```
package com.simple.validators;
import java.util.Map;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.workflow.InvalidInputException;
import com.opensymphony.workflow.Validator;
import com.opensymphony.workflow.WorkflowException;
public class TitleValidator implements Validator
{
   public void validate(Map transientVars, Map args, PropertySet ps)
           throws InvalidInputException, WorkflowException
   {
      String title = (String)transientVars.get("docTitle");
      if(title == null)
         throw new InvalidInputException("标题不能为空!");
      if(title.length() > 30)
         throw new InvalidInputException("标题长度不能大于 30！");
   }
}
```

流程描述定义规范

根据官方 DTD 的描述：

head

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow PUBLIC "-//OpenSymphony Group//DTD OSWorkflow 2.8//EN"
"http://www.opensymphony.com/osworkflow/workflow_2_8.dtd">
```

workfow

```
<!ELEMENT workflow (meta*, registers?, trigger-functions?, global-conditions?, initial-actions,
global-actions?, common-actions?, steps, splits?, joins?)>
```

以上说明了在 workflow 根节点所有的子节点。
*代表零个或者一个或者多个。
?代表零个或者一个。
既没有*也没有?代表有且只有一个。

请严格参照 DTD 的顺序书写 XML。其排列顺序必须为：
meta, registers, trigger-functions, global-conditions, initial-actions, global-actions, common-actions, steps, splits, joins

step

<!ELEMENT step (meta*, pre-functions?, external-permissions?, actions?, post-functions?)>
<!ATTLIST step
id CDATA #REQUIRED
name CDATA #REQUIRED
>
规则同上，请严格参照 DTD 的顺序书写 XML。

action

<!ELEMENT action (meta*, restrict-to? , validators?, pre-functions?, results, post-functions?)>
<!ATTLIST action
id CDATA #REQUIRED
name CDATA #REQUIRED
view CDATA #IMPLIED
auto (TRUE | FALSE | true | false) #IMPLIED
finish (TRUE | FALSE | true | false) #IMPLIED
>
规则同上，请严格参照 DTD 的顺序书写 XML。
更详细的规则请参阅我翻译的《OSWorkflow 中文手册》第 6.1 章节。

使用 GUI 设计器

下载最新的 osworkflow-2.8.0.zip，将 lib/designer 目录里面的 froms.jar,foxtrot.jar,jgraph.jar,looks.jar,syntax.jar 共五个 jar 拷贝到和 designer.jar 相同目录，双击 designer.jar 便可运行。注意：如果双击运行的是 win RAR，那么请在 designer.jar 上面右键->属性->常规->更改，将推荐的程序更改成为 Java(TM) 2 Platform Standard Edition binary 即可。

流程配置资源

官方提供的下载地址：
https://osworkflow.dev.java.net/source/browse/osworkflow/src/test/samples/
这里面分了多种情况详细列举了所有可能出现的流程配置方法，是个宝贵的资源，请大家务必去下载使用！

后记

最后，由于 OSWorkfow 的中文文档过于稀少且官方文档有些已经过时， 所以笔者才写了这份文档供大家参考和学习。初次学习工作流的朋友可能会感觉到 OSWorkfow 的概念过于抽象，这就需要不断摸索，经常实践并研究它的源代码，才能真正意义上掌握它并取得长足进步。

在此之前我写了一份名为《OSWorkflow 中文文档》的文档，那是一本自己经验和官方文档的”合辑”，为了完全区分官方和自己的工作，也为了解决版权问题，我在此便把两者完全地独立开了，并各自新增了不少内容。这份文档是原作，是对翻译的官方文档的有力补充。如果大家想看我翻译的中文版官方文档，请下载《OSWorkflow 中文手册》。

由于本人的水平有限，如有疏漏，恳请大家批评指正！本人联系方式如下：

MSN:cucuchen520@hotmail.com
Email:cucuchen520@yahoo.com.cn