方案一、用视图覆盖同名的ACT\_ID\_系列表

放弃使用系列表：ACT\_ID\_，创建同名的视图。

### 1.删除已创建的ACT\_ID\_\*表

创建视图必须删除引擎自动创建的ACT\_ID\_\*表，否则不能创建视图。

### 2.创建视图：

* ACT\_ID\_GROUP
* ACT\_ID\_INFO
* ACT\_ID\_MEMBERSHIP
* ACT\_ID\_USER

创建的视图要保证数据类型一致，例如用户的ACT\_ID\_MEMBERSHIP表的两个字段都是字符型，一般系统中都是用NUMBER作为用户、角色的主键类型，所以创建视图的时候要把数字类型转换为字符型。

### 3.修改引擎默认配置

在引擎配置中设置属性**dbIdentityUsed**为**false**即可。

代码如下

DROP VIEW IF EXISTS act\_id\_membership;

DROP VIEW IF EXISTS act\_id\_user;

DROP VIEW IF EXISTS act\_id\_group;

CREATE or replace VIEW act\_id\_user(ID\_,REV\_,FIRST\_,LAST\_,EMAIL\_,PWD\_,PICTURE\_ID\_ ,CONSTRAINT id\_pk PRIMARY KEY(ID\_) RELY DISABLE NOVALIDATE) AS

SELECT

TO\_CHAR(au.login\_name) AS ID\_,

0 AS REV\_,

to\_char(au.user\_name\_cn) AS FIRST\_,

'' AS LAST\_,

to\_char(au.email) AS EMAIL\_,

to\_char(au.PASSWORD) AS PWD\_,

'' AS PICTURE\_ID\_

FROM base\_user au;

CREATE VIEW act\_id\_group

AS

SELECT ar.name AS ID\_,

NULL AS REV\_,

ar.name AS NAME\_,

'assignment' AS TYPE\_

FROM s\_role ar;

CREATE VIEW act\_id\_membership

AS

SELECT (SELECT u.username FROM s\_user u WHERE u.id=ur.user\_id) AS USER\_ID\_,

(SELECT r.name FROM s\_role r WHERE r.id=ur.role\_id) AS GROUP\_ID\_

FROM s\_user\_role ur;

在spring.xml里添加：

1. <bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">

    ...

    <property name="dbIdentityUsed" value="false">

    ...

</property></bean>

## 方案二、调用IdentifyService接口完成同步

## 参考<http://www.kafeitu.me/activiti/2012/04/23/synchronize-or-redesign-user-and-role-for-activiti.html>

该方案任然使用的act\_id\_\*表，不过同步数据到用户自定表

## 方案三：自定义SessionFactory（项目使用）

## 使用mybits作为持久层。

## 第一步：在application.xml中导入 ：

## <import resource="applicationContext-activiti.xml" />

的activiti的配置文件，其中activiti的关键配置如下：

<bean id=*"processEngineConfiguration"* class=*"org.activiti.spring.SpringProcessEngineConfiguration"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"transactionManager"* ref=*"transactionManager"* />

<property name=*"databaseSchemaUpdate"* value=*"true"* />

<property name=*"jobExecutorActivate"* value=*"true"* />

。。。

<!-- 自定义表单字段类型 ,自己的项目里加上这个类然后指向他,这里先使用activiti中自己的用户角色表 -->

<property name=*"customSessionFactories"*>

<list>

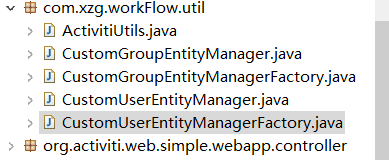
<bean class=*"com.xzg.workFlow.util.CustomUserEntityManagerFactory"*/>

<bean class=*"com.xzg.workFlow.util.CustomGroupEntityManagerFactory"*/>

</list> </property>

</bean>

其中这两个类就是覆盖activiti调用用户角的接口，通过重写他们来使用自定义的用户角色。



其中两个工程类是用来覆盖原来activiti自己的manager管理类

主要代码如下：

//用户管理工厂类

**public** **class** CustomUserEntityManagerFactory **implements** SessionFactory {

@Resource

**private** CustomUserEntityManager customUserEntityManager;

**public** Class<?> getSessionType() {

**return** UserIdentityManager.**class**;

}

**public** Session openSession() {

**return** customUserEntityManager;

}

//组管理工厂类

**public** **class** CustomGroupEntityManagerFactory **implements** SessionFactory {

@Resource

**private** CustomGroupEntityManager customGroupEntityManager;

**public** Class<?> getSessionType() {

// **TODO** Auto-generated method stub

**return** GroupIdentityManager.**class**;

}

**public** Session openSession() {

**return** customGroupEntityManager;

}

接下来就是使用自定义的管理类，但是自定义的管理类必须要继承activiti相应的接口，这是为了避免activiti工作流在调用api的时候和自定义不同。

主要代码如下：

//使用自定义用户角色一般只需覆盖findUserById和findGroupsByUser方法

@Service

**public** **class** CustomUserEntityManager **extends** UserEntityManager{

@Resource(name="userManager")

**private** UserMapper mapper;

@Override

**public** UserEntity findUserById(**final** String userCode) {

**if** (userCode == **null**)

**return** **null**;

**try** {

UserEntity userEntity = **null**;

com.xzg.domain.User bUser = mapper.getUserById(Long.*valueOf*(userCode));

userEntity = ActivitiUtils.*toActivitiUser*(bUser);

**return** userEntity;

} **catch** ( Exception e) {

e.printStackTrace();

}

**return** **null** ;

}

其他方法视情况自定。其中UserMapper是dao层。

userEntity = ActivitiUtils.*toActivitiUser*(bUser);

这里的ActivitiUtils是一个共有转化类，目的是为了将activiti自身的用户和自定义的用户转化。同理：

//使用自定义用户角色一般只需覆盖findGroupById和findGroupsByUser方法

@Service

**public** **class** CustomGroupEntityManager **extends** GroupEntityManager

不在赘述。

以下是ActivitiUtils的主要方法：

/\*\*

\* @author hasee

\* @TIME 2016年12月27日

\* 注意类的隐藏和实例创建

\*/

public class ActivitiUtils {

public static UserEntity toActivitiUser(User bUser){

UserEntity userEntity = new UserEntity();

userEntity.setId(bUser.getUserId().toString());

userEntity.setFirstName(bUser.getUserName());

userEntity.setLastName(bUser.getUserName());

userEntity.setPassword(bUser.getPassword());

userEntity.setEmail(bUser.getEmail());

userEntity.setRevision(1);

return userEntity;

}

public static GroupEntity toActivitiGroup(Group bGroup){

GroupEntity groupEntity = new GroupEntity();

groupEntity.setRevision(1);

groupEntity.setType("assignment");

groupEntity.setId(bGroup.getRoleId());

groupEntity.setName(bGroup.getRoleName() );

return groupEntity;

}

public static List<org.activiti.engine.identity.Group> toActivitiGroups(List<Group> bGroups){

List<org.activiti.engine.identity.Group> groupEntitys = new ArrayList<org.activiti.engine.identity.Group>();

for (Group bGroup : bGroups) {

GroupEntity groupEntity = toActivitiGroup(bGroup);

groupEntitys.add(groupEntity);

}

return groupEntitys;

}

}

这样就达到了自定义用户和角色的功能了，使用部分如下：