# 更新

7月18日更新了两表如何在第三张表进行关系操作

# Windchill优化

通过site.xconf文件进行优化操作：等于修改codebase中的wt.perproties文件

BackgroundServer服务窗口的打开和关闭：当然也可以关闭methodServer窗口

第一步：打开文件E:\ptc\Windchill\_10.2\Windchill \site.xconf文件添加

<Property name="wt.manager.monitor.services" overridable="true"

targetFile="codebase/wt.properties"

value="MethodServer BackgroundMethodServer"/>//这里是添加窗口操作，去掉就可以关闭

第二步.在windchill 的shell中执行xconfmanager -p命令来发布属性

其实就是更改wt.perproties文件

第三步 .删除tomcat缓存 可以不用操作的

流程绑定主物件，启动工作流后，流程状态依然显示为"未启动"的问题优化

<Property name="wt.queue.executeQueues" overridable="true"

targetFile="codebase/wt.properties"

value="true"/>

远程Eclipse的Debug支持：刚开始不支持需要自己手动配置

<Property name="wt.manager.cmd.MethodServer.debug.args" overridable="true"

targetFile="codebase/wt.properties"

value="-Xdebug -Xrunjdwp:transport=dt\_socket,address=9999,server=y,suspend=n"/>

更改method很多窗口的优化

<Property name="wt.manager.monitor.start.MethodServer" overridable="true"

targetFile="codebase/wt.properties"

value="1"/>

允许用户修改密码的功能

<Property name="wt.org.services.userPasswordChangeEnabled" overridable="true"

targetFile="codebase/wt.properties" value="true"/>

WEB-INF\log4jMethodServer.properties

需求:需监控此类的打印数据

方法:配置D:\ptc\Windchill\_10.1\Windchill\codebase\WEB-INF\log4jMethodServer.properties

在结尾加上以下,明确到com.wwl路径下的所有JAVA类

log4j.logger.coml=INFO

或者明确到此数据

log4j.logger.com.wwl =INFO

或者明确到类

log4j.logger. com.wwl.users =INFO

查看LOG日志

D:\ptc\Windchill\_10.1\Windchill\logs\mothodServer-log4j.log

Windchill\tomcat\conf\web.xml

jsp修改不用重启服务的方法：

打开Windchill\tomcat\conf\web.xml，修改里面的配置，将下面参数的值改为true

<init-param>

<param-name>development</param-name>

<param-value>true</param-value>

</init-param>

# 开发步骤

1）导包：把winchill的jar统一eclipse管理

右键工程名 🡪 add Library (添加库) 🡪 User Library 🡪 User Librarys –> new:输入名称 🡪 add External JARS

D:\ptc\Windchill\_10.2\Windchill\lib。

D:\ptc\Windchill\_10.2\Windchill\srclib 备注 tool包是建模需要的

3、D:\ptc\Windchill\_10.2\Windchill\codebase。

4、D:\ptc\Windchill\_10.2\Windchill\codebase\WEB-INF\lib，

2）导入windchill中codebase的所有内容（其实是工程中的class）

陈工师傅作法：【单build：建立。构建路径】

右键工程名build path 🡪add Class Folder 🡪Create New Folder 🡪建名称 🡪Advanced 🡪 Browse 🡪D:\ptc\Windchill\_10.2\Windchill\codebase

我的测试后可行作法：(感觉不行)

右键工build path 🡪 add External Class Folder 🡪 D:\ptc\Windchill\_10.2\Windchill\codebase

3）EclipseDebug远程调试

1、在wt.properties加入：

wt.manager.cmd.MethodServer.debug.args=-Xdebug -Xrunjdwp:transport=dt\_socket,address=9999,server=y,suspend=n

最好在E:\ptc\Windchill\_10.2\Windchill \site.xconf文件添加：

<Property name="wt.manager.cmd.MethodServer.debug.args" overridable="true"

targetFile="codebase/wt.properties"

value="-Xdebug -Xrunjdwp:transport=dt\_socket,address=9999,server=y,suspend=n"/>

然后使用命令：shell中执行xconfmanager -p命令来发布属性后就是修改了wt.properties文件

2、eclipse中进 run 🡪DebugConfigurations 🡪remote Java Application -- >Host:localhost（本地）Port：跟上面address一致：9999

测试代码

public static void main(String[] args) throws WTRuntimeException, WTException {

wt.fc.ReferenceFactory rf=new wt.fc.ReferenceFactory();

//根据数据库中的ida2a2获取到部件

WTPart part=(WTPart)rf.getReference("OR:wt.part.WTPart:138107").getObject();

// getPartNameByid(126048);

// getPartByid(126048);

// List<JSONObject> jsons=ClassificationProcessor.getSubNodeById(116794);

// System.out.println(jsons);

// HashMap attrconditon=new HashMap();

// attrconditon.put("remark", "1");

// attrconditon.put("length", "1");

// List<LYHPartAttrView> views=ClassificationUtil.searchClassPartView(116801,attrconditon);

// System.out.println(views.size());

System.out.println(part.getName());

}

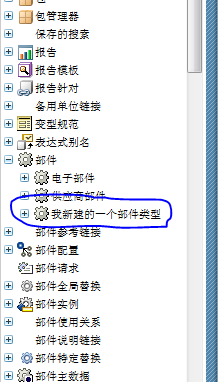
}

# Windchill操作

自定义类型和属性

WTTypeDefinition 类型表:保存所有的能创建的类型表，如：部件、文档等

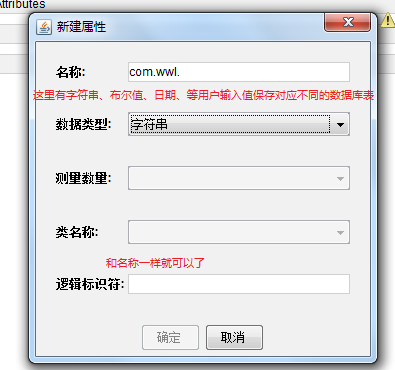
自定义类型：站点--》类型和属性管理—》 部件—》进行创建”这是我第一次创建的部件类型”--》创建完成后在产品进行创建部件会显示” 这是我第一次创建的部件类型”同时在WTTypeDefinition表里就有 内部的属性名称生成



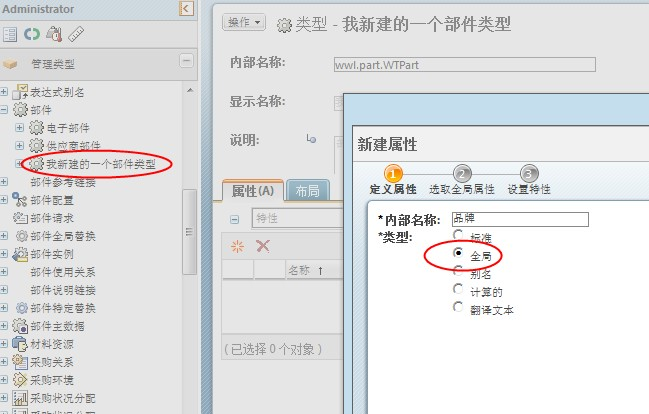
StringDefinition 属性表：所有创建部件、或文档要设置的属性都保存在此表

创建自定义属性brand并使用：

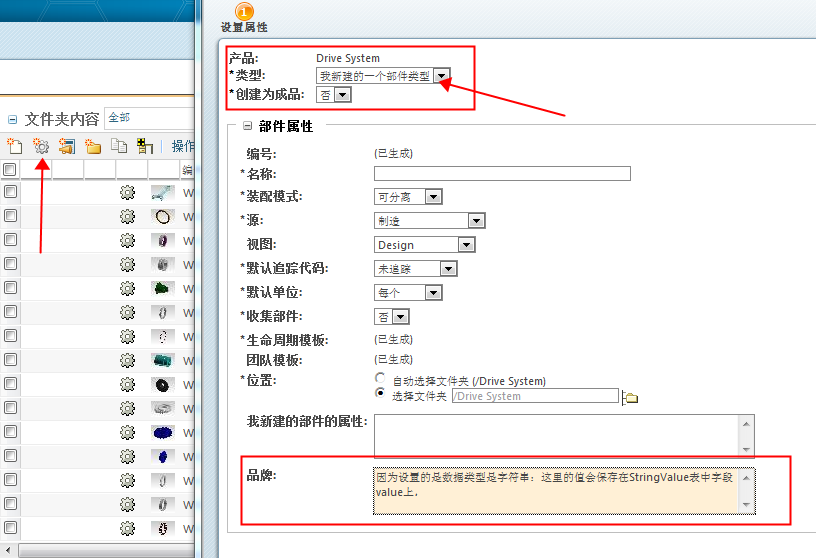
第一步：站点--》类型和属性管理—》左上角有个全局属性—》在需求属性中新建：brand，逻辑标识符写一样的brand，数据类型选择字符串，点击完成



第二步：在类型和属性管理界面选择部件-》创建新属性—》内部名称：‘品牌’必须选择全局—》选择你自定义的brand下一步直接点击完成



第三步：在产品中进行找到你为此类型设定的属性进行创建就可以看到自己设定的属性值了



StringValue保存属性值 保存定义为字符串属性用户输入值的表

自定义了属性‘brand’为字符串后，用户在为此‘品牌’的属性写值也会保存到此表的字段value上

如果属性值是int那么是保存在IntegerValue这个字段上的，同理布尔值就是在BooleanValue表上等

部件和类型表中的关系：默认部件：ida2a2:1815

关系WTTypeDefinition维护着所有类型的表包括自定义

WtPart表中字段：branchIdA2typeDefinitionRefe维护了CcontrolBranch【控制部门】表中的主键

**WtPart 表的字段：ida2typedefinitionreference字段维护着WTTypeDefinition主键id**

CcontrolBranch表字段IDA3B5维护了WTTypeDefinitionMaster表中的主键

WTTypeDefinition表中字段ida3masterreference维护着WTTypeDefinitionMaster表中的主键

WTTypeDefinition 表中的字段branchiditerationinfo维护着WtPart 表中字段branchIdA2typeDefinitionRefe

查询类型中的所有部件：

第一步把类型的名字进行查询获得WTTypeDefinition 字段中branchiditerationinfo数据

第二步：查询部件中字段branchIdA2typeDefinitionRefe值用第一步获取的，就可以查询所有此类型部件

部件和属性表中的关系：

关系 StringDefinition此表维护了字符串创建的属性名称

字符串属性StringValue表中字段：IDA3A4维护了wtpart表的主键

字符串属性StringValue表中IDA3A6维护了StringDefinition【字符串定义】表中的主键

查询所有使用了此字符串类型的部件（不是值哦）

第一步 通过类型名称查询到StringDefinition中主键值

第二步 用此值查询到StringValue中的 IDA3A4值

第三步 因为IDA3A4的值就是wtpart的主键id就可以查询到所有使用了此属性的部件

通过自定义名称为brand的值：查询所有的部件

第一步 获取brand类型中的值：如它是字符串的属性它StringValue表中字段value进行查询获取

第二步 拿获取到IDA3A4来数据进行查询部件即可

# CRUD

原始连接数据库

MethodContext mc=MethodContext.getContext();

WTConnection c=(WTConnection) mc.getConnection();

## 多对多关联操作

windchill都可以说是多对多关系，一个表维护着另两张表的关系

wtpart与wtpartmaster是多对一关系：wtpart是多:它有版本问题

陈工创建的模型表和文档表进行关联操作

@RequestMapping("/test3")

public void save(String lyhnumber,String wtdnumber ) throws WTException, WTPropertyVetoException {

LYHPart part=null; 陈工创建的模型表

WTDocumentMaster wdm=null; //需要关联的文档表

QuerySpec qs = new QuerySpec(LYHPart.class);

SearchCondition sc = new

SearchCondition(LYHPart.class,LYHPart.PARTNAME,SearchCondition.EQUAL,lyhnumber);

qs.appendWhere(sc, new int[]{0});

QueryResult qr=PersistenceHelper.manager.find(qs);

part=(LYHPart) qr.nextElement();

QuerySpec qs2 = new QuerySpec(WTDocumentMaster.class);

SearchCondition sc2 = new

SearchCondition(WTDocumentMaster.class,WTDocumentMaster.NUMBER,SearchCondition.EQUAL,wtdnumber);

qs2.appendWhere(sc2, new int[]{0});

QueryResult qr2=PersistenceHelper.manager.find(qs2);

wdm=(WTDocumentMaster) qr2.nextElement();

LYHPartDocLink ldl = new LYHPartDocLink();//这张表维护上面两张表的关系

ldl.setDoc(wdm);//set查出来的对象

ldl.setPart(part);//set set查出来的对象

PersistenceHelper.manager.save(ldl);

}

在创建的模型操作中可以进行表中联系：如LYHPartDocLink此表创建模型

ackage ext.lyric.classification;

import com.ptc.windchill.annotations.metadata.GenAsBinaryLink;

import com.ptc.windchill.annotations.metadata.GeneratedProperty;

import com.ptc.windchill.annotations.metadata.GeneratedRole;

import com.ptc.windchill.annotations.metadata.PropertyConstraints;

import wt.doc.WTDocumentMaster;

import wt.fc.ObjectToObjectLink;

import wt.util.WTException;

@GenAsBinaryLink(

superClass=ObjectToObjectLink.class,

roleA=@GeneratedRole(name="part",type=LYHPart.class), 进行LYHPart表维护

roleB=@GeneratedRole(name="doc",type=WTDocumentMaster.class), 进行WTDoc表维护

properties={

@GeneratedProperty(

name="creatorid",

type=long.class,

constraints=@PropertyConstraints(required=true)),

@GeneratedProperty(

name="modifierid",

type=long.class,

constraints=@PropertyConstraints(required=true))

}

)

public class LYHPartDocLink extends \_LYHPartDocLink{

static final long serialVersionUID=1;

public static LYHPartDocLink newLYHPartLink(final LYHPart part,final WTDocumentMaster doc)throws WTException **这里写法又不一样了**

{

final LYHPartDocLink instance=new LYHPartDocLink();

instance.initialize(part,doc);

return instance;

}

}

## 查询

基本查询

通过number或、name查询表数据

Wt.query.QuerySpec qs = new Wt.query.QuerySpec(StringDefinition.class);

wt.query.SearchCondition sc= new

wt.query.SearchCondition(StringDefinition.class,StringDefinition.NAME,SearchCondition.EQUAL,"brand");

qs.appendWhere(sc,new int[]{0});

QueryResult qr=wt.fc.PersistenceHelper.manager.find((StatementSpec)qs);

System.out.println(qs);

while(qr.hasMoreElements()){

StringDefinition sd=(StringDefinition) qr.nextElement();

//获取表中的ida2a2的查询方法

System.out.println(sd.getPersistInfo().getObjectIdentifier().getId());//获取ida2a2

//获取表中的idA3A4的查询方法

System.out.println(sd.getIBAHolderReference().getObjectId().getId());//获取idA3A4

//获取表中的ida3a6的查询方法

System.out.println(sd.getDefinitionReference().getObjectId().getId());//获取idA3A6

System.out.println(ida2a2);

}

1、获取codebase目录下的wt.properties文件:可以用来获取路径等

WTProperties properties = WTProperties.*getLocalProperties*();

返回Windchill安装目录，例："D:\ptc\Windchill\_10.1\Windchill"

String wtHomePath = properties.getProperty("wt.home");

返回Windchill 服务地址，例："plmlocal.szpnt.com.cn"

注意：经测试修改了此内容会导致服务启动不了

String hostName = properties.getProperty("wt.rmi.server.hostname");

2、获得当前登录用户的用户名

WTPrincipal p = SessionHelper.getPrincipal();第一种

WTUser user=(WTUser) SessionHelper.manager.getPrincipal();//第二种

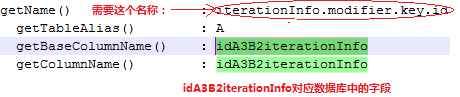
Long id=user.getPersistInfo().getObjectIdentifier().getId(); //获取数据库的id

WTUser user = (WTUser)p;

System.out.println(user.getName());//打印：wenwenliang

3、根据用户Id查询：此用户创建的文档等

第一步 （每个类中几乎都有关联wt.org.WTPart 用户类）shell命令：inforeport wt.epm.EPMDocument，此时会在E:\ptc\Windchill\_10.2\Windchill\temp目录下生成：epm.EPMDocument.out文件，通过第三方软件打开：搜索数据库找到对应的名称IDA3B2ITERATIONINFO，看图



第二步 把获得名称放到条件查询中

SearchCondition(WTDocument.class,"iterationInfo.modifier.key.id",SearchCondition.EQUAL,id);

//获取当前用户

WTUser user=(WTUser) SessionHelper.manager.getPrincipal();

Long id=user.getPersistInfo().getObjectIdentifier().getId();

System.out.println(id);

//查询开始阶段

QuerySpec q = new QuerySpec(EPMDocument.class);

SearchCondition qs = new

SearchCondition(EPMDocument.class,"iterationInfo.modifier.key.id",SearchCondition.EQUAL,id);

q.appendWhere(qs,new int[] {0});

System.out.println(q); //获取SQL语句

QueryResult qr = wt.fc.PersistenceHelper.manager.find((wt.pds.StatementSpec)q);

System.out.println(qr.size()); //获取总条数但感觉挺慢的

while(qr.hasMoreElements()){

EPMDocument ed =(EPMDocument) qr.nextElement();

System.out.println(ed.getName());

}

4、获取文档的类型

TypedUtilityServiceHelper.service.getExternalTypeIdentifier(wtobj)陈工给的

TypeIdentifierUtility.getTypeIdentifier(document) 自己找的

if(obj instanceof WTDocument) {

WTDocument wd = (WTDocument) obj;

//获取文档类型

String documentType=TypedUtilityServiceHelper.service.getExternalTypeIdentifier(wd); if(documentType.indexOf("cn.com")<0) {

return;

}

int dindex=documentType.lastIndexOf("|");

String documentTypeName=documentType.substring(dindex+1);

//获取properties文件中不需要判断名称的文档类型

Properties properties = PropertiesUtil.getDocumentTypeConfig();

String docTypeName=properties.getProperty("notNeedCheckNameType");

if(docTypeName==null)

docTypeName="";

String[] arrayName=docTypeName.split(",");

boolean flag=true;

for(String TybeName : arrayName) {

if(documentTypeName.equals(TybeName)) {//找到：不需要验证名称的类型

flag=false;

break;

}

}

5 自定义属性的查询

IBAUtil partIba = new IBAUtil(part);

String imageNum = ibaUtil.getIBAValue("figureno"); //获取自定义属性名称值

EPMBuildRule：3D图纸相关

SubFolder：文件夹

EPMDocument：3D图纸

WTDocument：文档

IdentityChangeHistory

ChangeIssue

WTChangeRequest2：变更请求

HolderToContent：对象和它附件的关系

WTPart：部件

ControlBranch

IteratedFolderMemberLink

新创建文档

第一次：WTDocumentMaster

第二次：wt.vc.ControlBranch

第三次：wt.doc.WTDocument 只进事件***PRE\_STORE***

第四次：wt.folder.IteratedFolderMemberLink

第五次：wt.lifecycle.LifeCycleHistory

第六次：wt.lifecycle.ObjectHistory

第七次：wt.doc.WTDocument

第八次：RecentUpdate

## windchill查询条件

升序和降序排序

LYHPartDocLink doc=(LYHPartDocLink) qr.nextElement();

QuerySpec qs2= new QuerySpec(WTDocument.class);

SearchCondition sc2 =

new SearchCondition(WTDocument.class,"masterReference.key.id",SearchCondition.EQUAL,

doc.getRoleBObject().getPersistInfo().getObjectIdentifier().getId());//通过主键id查询

//按降序来排序：参数 要查询的对象，需要排序字段名称，true是按升序，

qs2.appendOrderBy(WTDocument.class, "thePersistInfo.modifyStamp", true);

qs2.appendWhere(sc2,new int[]{0});

QueryResult qr2=PersistenceHelper.manager.find((StatementSpec)qs2)

# 建模

Windchill10.x建一个简单的模型（和Oracle数据库表一一对应）

第一步 编辑一个java类(此类可以说就是实体类和数据库字段一一对应):Users

注意:User是关键字不让创建

按下面格式进行创建实体类和数据库字段对应（下面的代码刚开始都会报错，因为还没有父类）

import com.ptc.windchill.annotations.metadata.GenAsPersistable;

import com.ptc.windchill.annotations.metadata.GeneratedProperty;

import com.ptc.windchill.annotations.metadata.PropertyConstraints;

import wt.fc.InvalidAttributeException;

import wt.fc.WTObject;

import wt.inf.container.WTContained;

import wt.util.WTException;

@GenAsPersistable(

superClass = WTObject.class,

interfaces ={WTContained.class},

properties ={

第一步：创建属性和数据库一一对应

@GeneratedProperty(

name="userName",//属性/字段名

type= String.class,//属性类型

constraints = @PropertyConstraints(required = true)),//都要等于true

@GeneratedProperty(

name="gender",

type=String.class,

constraints = @PropertyConstraints(required = true)),

@GeneratedProperty(

name="sex",

type=Integer.class,

constraints = @PropertyConstraints(required = true)) })

第二步 ：继承还没有的类（用shell命令ant生成）

public class Users extends \_Users{ //此类还没有的

static final long serialVersionUID =1;

public static Users newUser() throws WTException{ //单例模式

final Users instance=new Users();

instance.initialize(); //父类中的方法初始化

return instance;

}

//重写父类中的方法

public void checkAttributes() throws InvalidAttributeException{

super.checkAttributes();

}

}

第二步 生成父类对象

把第一步写好的类放置：E:\ptc\Windchill\_10.2\Windchill\src目录下（包名都要）

如你的类是在com.wwl.Users那路径就是：E:\ptc\Windchill\_10.2\Windchill\src\com\wwl\Users.java

打开windchill自带shell命令窗口：ant -f bin/tools.xml class -Dclass.includes=com\wwl\Users.java

一旦命令完：此时会在：

1. E:\ptc\Windchill\_10.2\Windchill\src\_gen\目录下生成相同全路径名的父类源文件\_Users.java
2. E:\ptc\Windchill\_10.2\Windchill\codebase\com\wwl\目录下有3个二进制文件\_Users.class、Users.class、Users.ClassInfo.ser
3. E:\ptc\Windchill\_10.2\Windchill\codebase\目录下两个文件：

descendentRegistry.properties文件中增加了两个内容：wt.fc.WTObject=com.wwl.test

wt.inf.container.WTContained=com.wwl.test

modelRegistry.properties文件中增加一个内容：com.wwl=test

第三步 生成数据库表

在shell命令窗口输入：ant -f bin/tools.xml sql\_script -Dgen.input=com.wwl.\_User(生成的父类没有后缀名)

此时会在E:\ptc\Windchill\_10.2\Windchill\db\sql3\目录下生成相同的（文件夹）

第四步 导入sql语句

找到E:\ptc\Windchill\_10.2\Windchill\db\sql3\com\wwl\create\_User\_Table.sql,在oracle登录账户环境下命令：@D:/a.sql

# 工作流的开发

工作流中获取路由事件的值

//从页面获取到路由受影响的的值是通过下面方式：单选框radion中name赋值的

工作流中获取HTTPServletRequest对象

需要静态方法，工作流调用，

下面这里设置NmCommandBean对象即可，windchill封装了

public static FormResult complete(NmCommandBean paramNmCommandBean)

HTTPServletRequest request =paramNmCommandBean.getRequest();//获取request对象

获取工作流中的状态、流程名字、流程节点名称

作用：开启那个流程模板都是根据流程名字来驱动的

走流程也是根据流程节点名称来判定走动的

获取workitem状态:后续打开的工作流是完成状态，

ReferenceFactory rf=new ReferenceFactory();

WorkItem wItem=(WorkItem)rf.getReference(request.getParameter("oid")).getObject();

int state =wItem.getStatus().getOrder();

pageContext.setAttribute("status", state);

} catch (WTException e) {

e.printStackTrace();

//获取流程模板的名字

WfProcess wfprocess = null;

if (workitem != null) {

WfAssignedActivity wfassignedactivity = getWfAssignedActivityByWorkItem(wItem);

if (wfassignedactivity != null) {

wfprocess = wfassignedactivity.getParentProcess();

}

}

String processTemplateName = process.getTemplate().getName() ; //获取驱动工作流的名字

//获取流程中节点的名字

WfAssignedActivity wfassignedactivity = null;

if (wItem!= null) {

wfassignedactivity = (WfAssignedActivity) wItem.getSource()

.getObject();

}

String activityName = wfassignedactivity.getName() //获取流程节点名字

用途

通过wtdocument开启工作流

public void startCreatePartProcess(Map<String, String> paramMap) {//获得网页外购物料申请单所有值

WfProcess wfprocess = null;

// 创建一个wtdocument

try {

String folerOid = paramMap.get("folderOid");//获取文件夹id

String jsonData = paramMap.get("data"); //获取数据

Folder foler = (Folder) CommonConstant.getObjectByOid(folerOid);

//自定义的方法返回file对象：指定文件的路径

File excelFile = ExcelUtil.createFileByPartApply(jsonData, foler.getName());

WTContainer con = (WTContainer) foler.getContainer();

if (foler instanceof WTContainer) {

con = (WTContainer) foler;

} else {

con = (WTContainer) foler.getContainer();

}

JSONArray jsonArray = new JSONArray(jsonData);

JSONObject jsonObj = jsonArray.getJSONObject(0);

WTLibrary wtl = CommonConstant.getLibrary(DOC\_LIBRIRY);

WTDocument wtdoc = WTDocumentUtil.createDocument(excelFile,

jsonObj.getString("name"), DOC\_FOLDER\_PATH, wtl,

PART\_APPLY\_TYPE);

// //启动流程开始

WTContainerRef containerRef = wtdoc.getContainerReference();

wtdoc = (WTDocument) PersistenceHelper.manager.refresh(wtdoc);

TeamReference teamRef = wtdoc.getTeamId();

wfprocess = WFUtil.startProcess(PART\_APPLY\_PROCESSNAME,

containerRef, teamRef);

ProcessData processData = wfprocess.getContext();

processData.setValue("primaryBusinessObject", wtdoc);// 保存主業務對象.

wfprocess = wfprocess.start(processData, true,

WTContainerRef.newWTContainerRef(con));

} catch (IOException e) {

e.printStackTrace();

} catch (WTException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

判断附件中的excle文件是否有值

第一种 在工作流中操作

第一步 路由选择事件（自定义）或转变（系统自带的）

已分配活动中节点：路由选择事件-》类型（有条件的）—》自定义名字如：通过

--》右边写静态方法—》插入变量primaryBusinessObject

ext.lyric.part.StandardPartService.decideSecondclassification(primaryBusinessObject)

第二步 StandardPartService类中的静态方法

public static void decideSecondclassification(WTObject pbo) throws WTException, PropertyVetoException, IOException{

//在第一步的变量他的对象就是WTObject对象，进行强转为启动工作流对象的WTdocument

WTDocument wd = (WTDocument)pbo;

//获取文档中的附件excle文件方法

ContentHolder holder= ContentHelper.service.getContents(wd);

QueryResult item =

ContentHelper.service.getContentsByRole(( FormatContentHolder)holder,ContentRoleType.PRIMARY);

if(item.hasMoreElements()) {

ApplicationData appdata = (ApplicationData) item.nextElement();

if (appdata != null) {

//String fileName=appdata.getFileName();//获取文件名

//获取io流：这里面就有所有的附件内容

InputStream in = ContentServerHelper.service

.findContentStream(appdata);

//excle中jar包方法对象，把流放进去

XSSFWorkbook workbook = new XSSFWorkbook(in);

Sheet sheet1 = (Sheet) workbook.getSheetAt(0);

Row row = sheet1.getRow(1);//从0开始的获取excle第二行数据

Cell cell=row.getCell(13);//从0开始获取excle第二行第14列的数据

String str=cell.getStringCellValue();

if( str==null ||"".equals(str)){

throw new WTException("物料群组不能为空！");

};

}

}

第二种 进行配置actions，然后类需要继承

第一步 E:\ptc\Windchill\_10.2\Windchill\codebase\config\actions\ custom-actions.xml进行配置

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE listofactions SYSTEM "actions.dtd">

<listofactions>

//配置<objecttype>标签

<objecttype name="work" resourceBundle="wt.workflow.work.WorkItem">

<action name="complete" enabledwhensuspended="true" aiax="page">

//自定义的全路径类名、方法名

<command class="ext.lyric.workflow.WorkflowUtil" method="complete"/>

</action>

</objecttype>

</listofactions>

第二步 在ext.lyric.workflow.WorkflowUtil类继承RemoteAccess包下建静态方法complete

public static FormResult complete(NmCommandBean paramNmCommandBean)

throws Exception {

FormResult fr = null;

// check 流程中备注信息

checkComments(paramNmCommandBean);

// 检查流程对象是否处于检出状态

System.out.println("==========================================checkComments");

String workItemOid = paramNmCommandBean.getRequest()

.getParameter("oid");

WorkItem workitem = (WorkItem) UtilManager

.getPersistableByOid(workItemOid);

WfAssignedActivity wfassignedactivity = getWfAssignedActivityByWorkItem(workitem);

WfProcess process = getWFProcessByWorkItem(workitem);

WTObject primaryBusinessObj = (WTObject) process.getContext().getValue(

"primaryBusinessObject");

boolean falg = completeActivity2(paramNmCommandBean);

checkObjectState(paramNmCommandBean, primaryBusinessObj, falg);

System.out.println("==========================================checkObjectState");

// if("PBOM审批流".equalsIgnoreCase(process.getTemplate().getName()) &&

// "机加辅料添加".equals(wfassignedactivity.getName())){

// checkPBOMDocument(primaryBusinessObj);

// }

// 外购物料信息变更申请审批流程

String user = workitem.getRole().getDisplay();

String processTemplateName = process.getTemplate().getName() ;

String activityName = wfassignedactivity.getName() ;

// hm 20170104

if ("外购物料申请审批流程".equals(processTemplateName)) {

int k = activityName.indexOf("批准");

int m = activityName.indexOf("物料拦截");

if ((k > -1|| m > -1) && !falg) {

WorkflowCheckUtil.checkParyApply(primaryBusinessObj,

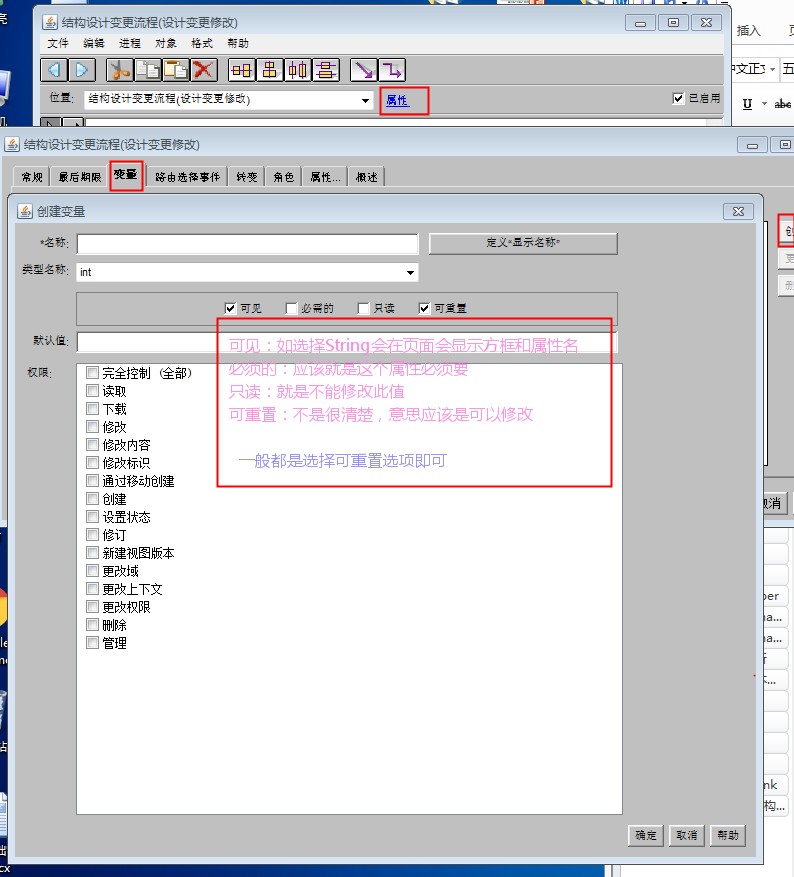
activityName);

}

}

工作流全局变量的设置和赋值

第一步 设置工作流全局变量：工作流模板首页🡪属性—>变量—>创建



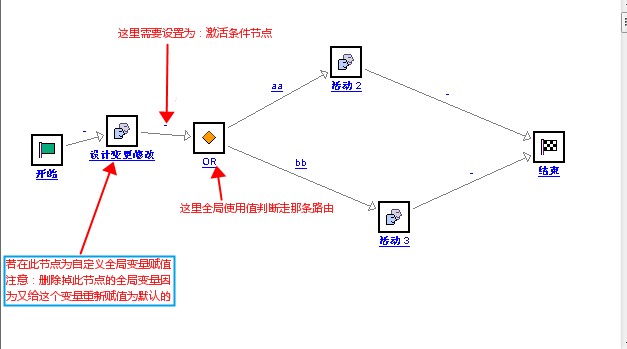
第二步 在需要的节点进行操作：如画一个条件路由器在这里进行操作

需要注意的事项

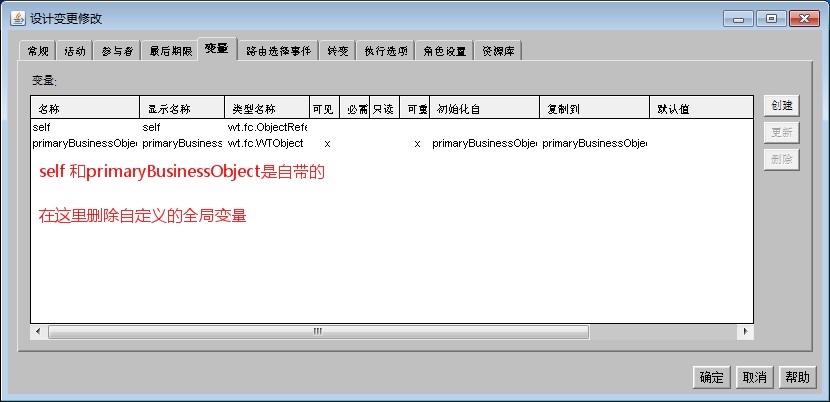
1、在一个节点为自定义的全局变量赋值，需要在下一个节点才能获取值

2、在要经过的节点如若不用全局变量需删除每个节点的自定义变量名称，不然会导致重新为此全局变量赋值操作

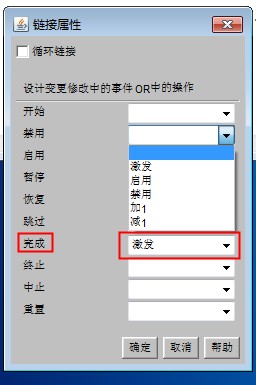
3、使用了条件的节点需要相对应的进行设置为“激活”如点击完成任务的 完成 在箭头有提示的点击在完成进行设置为激活才会触发条件节点



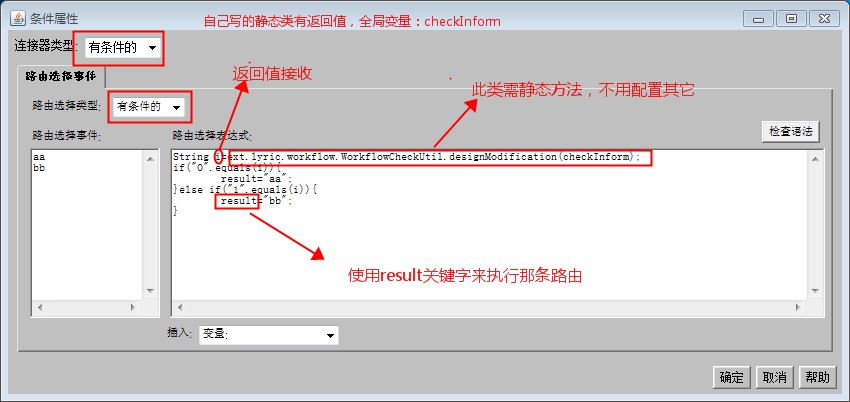
活动节点中删除变量



激活条件节点图



条件属性的操作



第三步 代码部分操作：

为全局变量进行赋值：前提需要在工作流模板有设置了全局变量

String workItemOid = request.getParameter("oid");

WorkItem workitem = (WorkItem) UtilManager.getPersistableByOid(workItemOid);

WfProcess process = WorkflowUtil.getWFProcessByWorkItem(workitem);

ProcessData processdata = process.getContext();

//全局变量名称：checkInform

processdata.setValue("checkInform", request.getParameter("checkInform"));

process.setContext(processdata);

process = (WfProcess) PersistenceHelper.manager.modify(process);

process = (WfProcess) PersistenceHelper.manager.refresh(process);

# Windchill类

## WTPart

public class wt.part.WTPart extends wt.part.\_WTPart(开头有底斜杠)

此类是部件类

普通方法

getContainerName()//获取查询到此对象的：最近产品页面的名称：摩托车

part.getName()：获取此部件的名称

getAuthoringLanguage()：获取语言编码

getConceptualClassname()：打印全类路径名：wt.part.WTPart

getCreatorEMail()：获得创照者的邮件：[531688079@qq.com](mailto:531688079@qq.com)

getCreatorFullName()：创造者的名字：温文亮

getCreatorName()：创造者账号：wenwenliang

getFlexTypeIdPath()：暂时未知：我测试打印null

getFolderPath()：打印路径把：/Default/摩托车配件/轮子/轮子（1）

part.getLocation()：获取路径不带名称：Default/摩托车配件/轮子

getIdentity()：获取到3个数据：0000000021, 轮子（1）, 1 (Design)

getLifeCycleName()：获取到的是英文不知啥意思：Two Phase Development

getOrganizationName()：获取到组织名：wwc

getOrganizationUniqueIdentifier()：也是获取到唯一标识符组织名：wwc

getType()：获取类型：部件

getViewName()：获取版本中的英文：Design

### 文档的类型

（1）WTDocumetManster ：是文档的主要信息，一个文档只有一条记录；

（2）WTDocument：是文档小版本记录，每一个文档小版本都有一条记录；

备注：

（1）文档大版本记录在WTDocument,通过getBranchIdentifier()方法可以获取到大版本的ID；

（2）如果文件已建立很多版本版序，当master改变时将影响所有的版本版序；

（3）文档帮助类为WTDocumentHelper, 实现类为WTDocumentHelper.service

## QuerySpec

【查询 说明书】

public class QuerySpec extends Object implements Externalizable（此类也是标识符）

这个类表示一个关于Windchill对象的查询。该查询是根据类、搜索condtions等构造的。当PersistenceManager使用时，该类根据其当前属性值构造一个SQL查询语句。

注意，此实现不同步。如果mulitple线程修改或执行相同的实例，那么访问必须在外部同步。或者，可以通过clone()方法复制实例，以便每个线程访问它自己的实例。

构造方法(Constructor)

QuerySpec()

创建规范的对象。

QuerySpec(Class queryClass)

创建规范对象并标识要搜索的基类。

QuerySpec(Class baseClass1, Class baseClass2)

创建规范对象并标识要搜索的基类。

Method(方法)

1、addClassList(Class newClass, boolean isSelectable)：将类添加到类列表

参数：类，true：将类包含在select中

-----------------------------------------------------------------------------------------

2、appendAnd()在where子句中添加一个和操作符

-----------------------------------------------------------------------------------------

3、int appendClassList(Class newClass,boolean isSelectable)将一个新类追加到类列表。

参数:

可选择-指示是否将类包含在SELECT中。

返回:

在类列表中的int位置。

抛出:

QueryException

-----------------------------------------------------------------------------------------

4、appendCloseParen()：在where子句中添加一个闭括号。

-----------------------------------------------------------------------------------------

5、int appendFrom(TableExpression a\_tableExpression)：

在From子句中追加一个表表达式。

参数是一个TableExpression类型的

-----------------------------------------------------------------------------------------

**6、**[**appendGroupBy**](file:///D:\ptc\Windchill_10.2\Windchill\codebase\wt\clients\library\api\wt\query\QuerySpec.html#appendGroupBy(wt.query.ColumnExpression, int[], boolean))([**ColumnExpression**](file:///D:\ptc\Windchill_10.2\Windchill\codebase\wt\clients\library\api\wt\query\ColumnExpression.html) a\_column, int[] a\_fromIndicies,

boolean a\_appendSelect)

将列表达式添加到GROUP BY子句中。

7、appendHaving(WhereExpression a\_where,int[]a\_fromIndicies)

将where表达式附加到HAVING子句。

8、appendHint(Staring a\_hint)

该方法附加了一个datastore特定的查询优化提示。

9、appendJoin(int a\_linkIndex, String a\_role, int a\_targetIndex)

将链接索引指定的链接类的联接条件附加到目标索引指定的目标类。

10、appendJoin(int a\_linkIndex, String a\_role, Persistable a\_source)

将链接索引指定的链接类的连接条件附加到该角色的源对象。

11、appendNot()

在where子句中添加一个NOT运算符。

12、appendOpenParen()

在where子句中添加一个开括号。

13、appendOr()

在where子句中添加一个或操作符。

14、appendOrderBy(OrderBy a\_orderBy,int[]a\_fromIndicies)

该方法将OrderBy表达式附加到查询的OrderBy子句。

15、appendSelect(ColumnExpression a\_column, boolean a\_selectOnly)

将列表达式追加到select子句。

16、appendSelect(columnexa\_column, int[] a\_fromIndicies, boolean a\_selectOnly)

将列表达式追加到select子句。

17、appendSelectAttribute(String a\_attributeName, int a\_classIndex, boolean a\_selectOnly)

将class属性附加到select子句。

18、appendSelectReference(ColumnExpression a\_className, ColumnExpression a\_id, int a\_classNameFromIndex, int a\_idFromIndex，布尔a\_fullObject)

将类引用附加到select子句。

19、appendSelectReference(字符串a\_attributeName, int a\_classIndex, boolean a\_fullObject)

将类引用附加到select子句。

20、appendWhere(WhereExpression a\_whereExpression,int[]a\_fromIndicies)

将表达式附加到WHERE子

21、getBindParameterAt(int a\_index)

返回指定偏移量的绑定参数值。

22、getBindParameterCount()

获取属性的值:bindParameterCount;指示此查询的编号绑定参数。

23、getConditionCount()

获取属性的值:conditionCount;表示该查询附加的搜索条件的数量

24、getDepthThreshold()

获取属性的值:depthThreshold;子类组合数的阈值。

25、getFromClause()

获取扮演角色的关联的对象:from子句。

26、getResultIndex(int a\_classIndex)

根据类列表的当前可选设置返回类索引的结果索引。

27、getResultIndicies()

根据当前可选设置，返回类列表中每个类的结果索引数组。

28、getSelect(int position)

返回类列表中指定位置的可选择属性。

29、isAdvancedQueryEnabled()

获取属性的值:advancedQueryEnabled;指示是否启用高级查询功能。

30、isDistinct()

获取属性的值:distinct;指示查询是否返回不同的结果。

31、isUseBind()

获取属性的值:useBind;指示是否在附加搜索条件时使用绑定参数。

32、setAdvancedQueryEnabled(布尔a\_AdvancedQueryEnabled)

设置属性的值:advancedQueryEnabled;指示是否启用高级查询功能

33、setDepthThreshold(int a\_DepthThreshold)

设置属性的值:depthThreshold;子类组合数的阈值。

34、setDistinct(布尔a\_Distinct)

设置属性的值:distinct;指示查询是否返回不同的结果。

35、setQueryLimit(int a\_queryLimit)

设置该查询返回的结果的查询限制。

36、setSelect(int位置,布尔isSelectable)

设置类列表中指定位置的可选属性。

37、setUseBind(布尔a\_UseBind)

设置属性的值:useBind;指示是否在附加搜索条件时使用绑定参数。

## [SearchCondition](file:///D:\ptc\Windchill_10.2\Windchill\codebase\wt\clients\library\api\wt\query\SearchCondition.html)

这个类表示查询的搜索条件。当附加到QuerySpec时，值将在SQL WHERE子句中使用。

Public class SearchCondition extends Object implenebts WhereEcpression,Extemalizable

## wt.fc

### PersistenceHelper

public class PersistenceHelper extends Object implements Serializable

PersistenceHelper【持久性助手】

重要：此类的全部方法和变量都是静态的

这个类包含与PersistenceManager关联的实用方法。该方法构建用于导航关联的查询规范。

静态成员变量

static PersistenceManager manager：获得PersistemceManager接口类

method方法

static NavigateSpec buildNavigateSpec(Object source, String role, Class linkClass, boolean onlyOtherSide)

该方法构建用于导航关联的查询规范。该方法构建用于导航关联的查询规范。

static Timestamp getCreateStamp(Persistable obj)

为给定的持久化对象检索创建的时间戳。

static Timestamp getModifyStamp(Persistable obj)

检索给定的持久化对象的修改时间戳。

static ObjectIdentifier getObjectIdentifier(Persistable obj)

检索给定的持久化对象的对象标识符

static boolean isDeleted(Persistable obj)

如果指定的持久化对象标记为delet，则返回true

static boolean isEquivalent(Persistable obj1, Persistable obj2)

如果两个给定对象是相等的持久化对象，则返回true。

static boolean isPersistent(Object obj)

返回true，如果给定对象是稳定的，并且已经保存在数据库中。

static boolean isPersistent(Persistable obj)

如果给定的持久化对象保存在数据存储中，则返回true。

static QueryResult navigate(Object source, String role, Class linkClass, boolean onlyOtherSide)

检索与specfy源对象(s)角色和链接类相关的对象。

### PersistenceManager

@RemoteInterface **有一个注解**

public interface PersistenceManager 接口类

PersistenceManager【持久性管理器】

PersistenceManager接口识别应用程序用来管理其业务对象的持久状态的方法集。虽然该接口声明的所有方法都在服务器上执行，但是它们通过一个助手类访问客户机应用程序。看到PersistenceHelper。

Methods方法

删除

Persistable delete(Persistable obj)

从数据存储中移除给定的持久化对象。返回一个标识接口Persistable

WTSet delete(WTSet a\_bjects)

从数据存储中移除指定的持久化对象。返回一个标识接口类WTSet

修改

Persistable modify(Persistable obj)

在数据存储中更新给定的持久化对象。

Persistable modify(Persistable obj, String attrName, ObjectMappable objAttr)

使用ObjectMappable属性中指定的值更新datastore中给定的持久化对象。

-------------------------------------------------------------------------------------------------------

WTCollection modify(WTCollection a\_objects)

在数据存储中更新指定的持久化对象。

WTCollection modify(WTCollection a\_objects, WTCollectionExceptionHandler a\_handler)

在数据存储中更新指定的持久化对象。

保存

Persistable save(Persistable obj)

如果给定对象是持久的，则调用modify方法，否则save方法将调用存储方法。

WTCollection save(WTCollection objects)

调用已经持久化的对象上的修改方法，否则调用存储方法。

Persistable store(Persistable obj)

在datastor中存储指定的持久化对象。e.

WTCollection store(WTCollection a\_objects)

在数据存储中存储指定的持久化对象集合。

WTCollection store(WTCollection a\_objects, WTCollectionExceptionHandler a\_handler)

在数据存储中存储指定的持久化对象集合。

查询

QueryResult find(Class targetLinkClass, ObjectIdentifier obj1Oid, String obj1Role,

ObjectIdentifier obj2Oid)

检索存在于两个持久化对象之间的所有链接对象，并给出它们的对象标识符。

QueryResult find(Class targetLinkClass, Persistable obj1, String obj1Role, Persistable obj2)

检索存在于两个给定的持久化对象之间的任何和所有链接对象。

QueryResult find(StatementSpec a\_statementSpec)

根据指定的语句从数据存储中检索持久性对象。

ResultProcessor find(StatementSpec a\_statementSpec, ResultProcessor a\_resultProcessor)

从数据存储中检索指定语句的持久性对象。返回ResultProcessor

-----------------------------------------------------------------------------------------------------------------------------

String getCurrentSequence(Class a\_class)

给定序列类，返回序列的数量。

String getCurrentSequence(Class a\_class, String AttributeName)

给定列序列的类作为必需的输入，以及列序列的属性名称，返回当前序列。

InputStream getLob(LobLocator lob)

返回与给定Lob定位器关联的Lob的输入流。

String getNextSequence(Class a\_class)

给定序列类作为输入，返回下一个值。

String getNextSequence(Class a\_class, String attributeName)

给定列序列的类作为必需的输入，以及列序列的属性名称，返回下一个序列。

Long getSequenceIncrement(Class a\_class)

给定序列类，返回当前序列。

Long getSequenceIncrement(Class a\_class, String AttributeName)

给定列序列的类作为必需的输入，以及列序列的属性名称，返回序列的减量。

QueryResult navigate(Persistable obj, String role, Class linkClass)

检索与指定的持久化对象相关的对象，并提供一个角色和链接类。

QueryResult navigate(Persistable obj, String role, Class linkClass, boolean onlyOtherSide)

检索与指定的持久化对象相关的对象，并提供一个角色和链接类。

QueryResult navigate(Persistable obj, String role, QuerySpec criteria)

检索与给定的持久化对象相关的对象，给定角色，关联名称和选择标准。

QueryResult navigate(Persistable obj, String role, QuerySpec criteria, boolean onlyOtherSide)

检索与给定的持久化对象相关的对象，给定角色，关联名称和选择标准。

Persistable refresh(ObjectIdentifier objId)

给定对象标识符，从数据库检索一个持久化对象。

Persistable refresh(ObjectIdentifier objId, boolean lock)

给定对象标识符，从数据库检索一个持久化对象。

Persistable refresh(Persistable obj)

从数据库检索给定的持久化对象，以恢复其状态。

Persistable refresh(Persistable obj, boolean fullRefresh)

从数据库检索给定的持久化对象，以恢复其状态。

Persistable refresh(Persistable obj, boolean fullRefresh, boolean inPlace)

从数据库检索给定的持久化对象，以恢复其状态。

Persistable refresh(Persistable obj, boolean fullRefresh, boolean inPlace, boolean lock)

从数据库检索给定的持久化对象，以恢复其状态。

### PersistenceManagerEvent

用于表示在PersistenceManager中发生的事件。该事件同时支持单个和多个对象事件目标。对于大多数操作，事件目标要么是单一的持久化，要么是持久性的WTCollection。

CLEANUP\_LINK单个对象事件包含源持久化对象作为目标，清理的链接在关联链接属性中填充。CLEANUP\_LINK多个对象事件包含一个WTKeyedMap作为目标。该映射中的键集包含源对象，关联的值是链接对象。注意，WTKeyedMap中的对象可能不会被夸大，因此在可能的情况下应该将它们作为引用。

COPY\_LINK单个对象事件包含原始的链接对象作为目标，新的链接在关联链接属性中填充。COPY\_LINK多个对象事件包含一个WTKeyedMap作为目标。该映射中的键集包含原始的链接对象，关联的值是新的链接对象。注意，该事件的WTKeyedMap中的对象将被夸大。

static String copy\_link

事件用于通知侦听器一个链接对象已被复制。

static String insert

该事件用于通知侦听器，该对象已插入到数据存储中。

static String post\_delete

事件用于通知侦听器一个对象已被删除。

static String post\_modify

事件用于通知侦听器对象已被修改。

static String post\_store

事件用于通知侦听器一个新对象已经创建。

static String pre\_delete

该事件用于通知侦听器一个对象即将被删除。

static String pre\_insert

该事件用于通知侦听器，该对象将被插入到数据存储中。

static String pre\_modify

该事件用于通知侦听器一个对象即将被修改。

static String pre\_remove

该事件用于通知侦听器一个对象即将从数据存储中删除。

static String pre\_store

该事件用于通知侦听器一个新对象即将被创建：即将保存

static String prepare\_for\_modification

事件用于通知侦听器一个对象要准备修改。

static String prepare\_for\_view

事件用于通知侦听器一个对象是为视图准备的。

静态字符串 REMOVE

事件用于通知侦听器对象已从数据存储中删除。

静态字符串 UPDATE

事件用于通知侦听器，该对象已在数据存储中更新。

/\*\*

\* 部件创建、图纸创建相关监听操作

\*

\*/

public class StandardListenerService extends StandardManager implements ListenerService, Serializable {

private static final long serialVersionUID = 1L;

private static final String CLASSNAME = StandardListenerService.class.getName();

private KeyedEventListener listener;

@Override

public String getConceptualClassname() {

return CLASSNAME;

}

public static StandardListenerService newStandardListenerService() throws WTException {

StandardListenerService instance = new StandardListenerService();

instance.initialize();

return instance;

}

@Override

protected void performStartupProcess() throws ManagerException {

listener = new ListenerEventListener(this.getConceptualClassname());

// wt.ownership.OwnershipServiceEvent/POST\_CHANGEOWNER

getManagerService().addEventListener(listener, PersistenceManagerEvent.generateEventKey(PersistenceManagerEvent.PRE\_STORE));

getManagerService().addEventListener(listener, PersistenceManagerEvent.generateEventKey(PersistenceManagerEvent.POST\_STORE));

getManagerService().addEventListener(listener, WorkInProgressServiceEvent.generateEventKey(WorkInProgressServiceEvent.POST\_CHECKIN));

getManagerService().addEventListener(listener, OwnershipServiceEvent.generateEventKey(OwnershipServiceEvent.POST\_CHANGEOWNER));

}

class ListenerEventListener extends ServiceEventListenerAdapter {

public ListenerEventListener(String manager\_name) {

super(manager\_name);

}

@Override

public void notifyVetoableEvent(Object eve) throws Exception {

if (!(eve instanceof KeyedEvent))

return;

KeyedEvent eventObject = (KeyedEvent) eve;

if (eventObject instanceof PersistenceManagerEvent) {

PersistenceManagerEvent event = (PersistenceManagerEvent) eve;

if (event.getEventKey().contains(PersistenceManagerEvent.PRE\_STORE)) {//即将被创建对象

processProStoreAction(event);

return ;

}

if (eventObject.getEventType().equals(PersistenceManagerEvent.POST\_STORE)) {//已经创建对象

updatePartIBAFromEPM(event);

processCreateFolderAction(event);

try {

processCheckEPMdoc(event);

} catch (RemoteException e) {

e.printStackTrace();

}

processCheckWTDocument(event);

processEditFloderNameAction(event);

//处理问题报告的创建

processChangeIssueCreate(event);

//处理工艺文档

updatePartUserLink(event);

updatePartAttrByCheckin(event);

// processChangeObject(event);

putEventToScheduleQueue(event);

//保存文档

pubEventToSave(event);

return ;

}

}

# Windchill中的事件监听

以下是Windchill中可监听的部分事件类型，一般是按功能划分：

Wt.fc.PersistenceManagerEvent

Wt.vc.VersionControlServiceEvent

Wt.vc.wip.WorkLnProgressServiceEvent

Wt.ecents.summart.ModifyContentSummartEvent

以PersistenceManagerEvent为例。所有可监听的事件在该类的静态常量中，定义如下：

Java代码

Public static final String PRE\_STORE = “PRE\_STORE”

Public static final String POST\_STORE = “POST\_STORE”

Public static final String PRE\_MODIFY = “PRE\_MODIFY

Public static final String POST\_MODIFY = “POST\_MODIFY”

Public static final String PREPARE\_FOR\_MODIFICSTION = “PREPARE\_FOR\_MODIFICSTION”

Public static final String PREPARE\_FOR\_VIEW = “PREPARE\_FOR\_VIEW”

Public static final String UPDATE = “UPDATE”

Public static final String PRE\_REMOVE = “PRE\_REMOVE”

Public static final String REMOVE = “REMOVE”

Public static final String PRE\_DELETE = “PRE\_DELETE”

Public static final String POST\_DELETE = “POST\_DELETE”

Public static final String CLEANUP\_LINK = “CLEANUP\_LINK

Public static final String COPY\_LINK = “COPY\_LINK”

所有的事件监听原理相同，我们以删除事件为例（对应的事件类型为

PersistenceManagerEvent.POST\_DELETE）,假定存在名字为postdelete的文档对象，如果对此文档进行删除操作，则在提交确认的时候阻止并进行信息提示，实现该监听的方法如下：

第一步：创建一个空的接口ListenerService，这个必须的

package test;

/\*\*

\* 空接口，什么也不做，单这是必须的

\* <P>Description:</P>

\* @author wenwenliang

\*/

public interface ListenerService {

}

第二步：创建具体的实现类，这个类需要继承wt.services.StandardManager类并实现我们定义的空接口ListenerService，

**package** test;

**import** java.io.Serializable;

**import** test.ListenerService;

**import** wt.events.KeyedEvent;

**import** wt.events.KeyedEventListener;

**import** wt.fc.PersistenceManagerEvent;

**import** wt.services.ManagerException;

**import** wt.services.ServiceEventListenerAdapter;

**import** wt.services.StandardManager;

**import** wt.util.WTException;

**public** **class** TestService **extends** StandardManager **implements** ListenerService,Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

//获取此类的全路径名

**private** **static** **final** String ***CLASSNAME***=TestService.**class**.getName();

**private** KeyedEventListener listener;

//获取此类成员变量的：全路径名

//重写父类StandardManager中的方法

@Override

**public** String getConceptualClassname() {

**return** ***CLASSNAME***;

}

//差不多是把此类给StandardManager初始化，然后返回此类

**public** **static** TestService newTestService() **throws** WTException {

TestService instance=**new** TestService();

instance.initialize();//此类是StandardManager中的初始化的意思

**return** instance;

}

/\*\*

\* 添加需要监听的事件

\*/

@Override//重写父类StandardManager中的方法

**protected** **void** performStartupProcess() **throws** ManagerException {

//传入 类的全路径名

listener = **new** WCListenerEventListener(**this**.getConceptualClassname());

//添加需要监听的事件，可以添加多个，这里添加的是删除事件 POST\_DELETE和修改事件 POST\_MODIFY

getManagerService().addEventListener(listener, PersistenceManagerEvent

.*generateEventKey*(PersistenceManagerEvent.***POST\_DELETE***));

getManagerService().addEventListener(listener, PersistenceManagerEvent

.*generateEventKey*(PersistenceManagerEvent.***POST\_MODIFY***));

}

/\*\*

\* 定义内部类，用来处理相应的事件

\*/

**class** WCListenerEventListener **extends** ServiceEventListenerAdapter{

//继承的类没有无参数的构造方法这里必须构造

//隐式超级构造函数ServiceEventListenerAdapter()对默认构造函数没有定义。必须定义显式构造函数

**public** WCListenerEventListener(String manager\_name) {

**super**(manager\_name);

}

//重写wt.services.ServiceEventListenerAdapter中的方法

@Override

**public** **void** notifyVetoableEvent(Object eve) **throws** Exception {

**if**(!(eve **instanceof** KeyedEvent)) {//KeyedEvent一种事件

**return**;

}

//获取当前触发的事件对象

KeyedEvent event =(KeyedEvent) eve;

//获取当前被操作的持久化对象，如不见，文档，容器等

Object target =event.getEventTarget();

String eventType = event.getEventType();

}

//还没写完，看https://wenku.baidu.com/view/7593c56d1ed9ad51f01df2b4.html

}

}

# Windchill常用类及基本操作

部件及相关操作

对象

Wt.part.WTPart

Wt.part.WTPartMaster

部件操作

查询部件

Public WTPart getObjectByName(String partName)

Throws WTException,InstantiationException,

文档、图档

变更

升级

容器

文件夹

生命周期

工作流

项目

软属性

权限

查询

Windchill学习网页

<https://wenku.baidu.com/view/eeb1cbb10b4c2e3f56276330.html（windchill常用类及基本操作）百度文库共55>页左右吧，不错

# Windchill新建角色

打开 Windchill 外壳程序并运行 enumcustomize

单击浏览并打开 WT\_HOME/codebase/wt/project/RoleRB.RB.ser

添加自定义角色

单击保存，然后继续编译资源文件 wt.project.RoleRB

使用 ant -f codebase/MakeJar.xml 命令重新编译jar文件

停止 Tomcat （仅Windchill 9.0， 9.1需执行此步）和 Windchill

删除 TOMCAT\work（Windchill9.0， 9.1）或 WT\_HOME/tomcat/instances 目录下的内容，清空 Tomcat  缓存

重启 Tomcat 和 Windchill

# oracle数据库表

### WTPartMaster

部件Master

部件表（26个字段）【WT Part Master：重量部分主】

字段名 字段描述 关系描述

Ida2a2 唯一标识（主id） 主键

Classnamea2a2 wt.part.WTPartMaster 该表对应的类

Name 部件名称

Wtpartnumber 部件编号

Createstampa2 创建时间

Modifyatampa2 更新时间

UpdateCounta2

Updatestampa2 更新时间

Description 描述

Ida3containerreference 对应表（PDMLinkProduct 中 ida2a2） 外键（PDMLinkProduct）

classnamekeycontainerreferen wt.pdmlink.PDMI.inkPRoduct(容器) 该外键表对应类

Series wt.series.HarvardSeries

ida3organizationreference 对应表（WTOrganization 中 ida2a2） 外键（WTOrganization）

classnamekeyorganizationrefe wt.org.WTOrganization(组织) 该外键表对应类

还有一些是自动时间字段等（具体再了解吧）

### WTPart

部件表【WT Part：重量的一部分】

Ida2a2 唯一标识 主键

Classnamea2a2 wt.part.WTPart 该表对应的类

Modifystampa2 更新时间

Createstampa2 创建时间

Markfordeletea2

Updatestampa2 更新时间

Ida3containerreference 对应表（PDMLinkProduct 中 ida2a2） 外键（PDMLinkProduct）

classnamekeycontainerreferen wt.pdmlink.PDMI.inkPRoduct(容器) 该外键表对应类

statecheckoutinfo 检入,检出状态 值为c/i检入，c/o检出

ida3a2checkoutinfo

classnamekeya2checkoutinfo

contractnumber

ida3domainref 对应表（AdministrativeDomain中 ida2a2）外键（AdministrativeDomain）

classnamekeydomainref wt.admin.AdministrativeDomain 该外键表对应类

ida3a2folderinginfo 对应表（Cabinet中 ida2a2） 外键cabinet

classnamekeya2folderinginfo wt.folder.Cabinet 该外键表对应类

ida3b2folderinginfo` 对应表（SubFolder中 ida2a2） 外键（SubFokler）

classnamekeyb2folderinginfo wt.folder.SubFolder 该外键表对应类

Nameb2folderinginfo 文件夹名

ida3d2iterationinfo 对应表（WTUser 中ida2a2） 外键（WTUser）

classnamekeyd2iterationinfo wt.org.WTUser:

ida3b2iterationinfo 对应表（WTUser 中ida2a2） 外键（WTUser）

classnamekeyb2iterationinfo wt.org.WTUser 跟上面一模一样的好奇怪

iterationida2iterationinfo 小版本 值（1.2.3….）

latestiterarioninfo 是否为最新版本 值为（0/1）,1即为最新版本

ida3masterreference 对应WTPartMaster中 ida2a2 外键WTPartMaster

classnamekeymasterreferenc wt.part.WTPartMaster 该外键表对应类

ida3a2state 对应表（LifeCycleTemplate 中 idida2a2）外键LifeCycleTemplate

classnamekeya2state wt.lifecycle.lifeCycleTemplate 该外键表对应类（生命周期模板）

statestate 生命周期状态

ida3view 对应表（View 中ida2a2） 外键 View

classnamekeyview wt.vc.views.View 该外键对应类（视图）

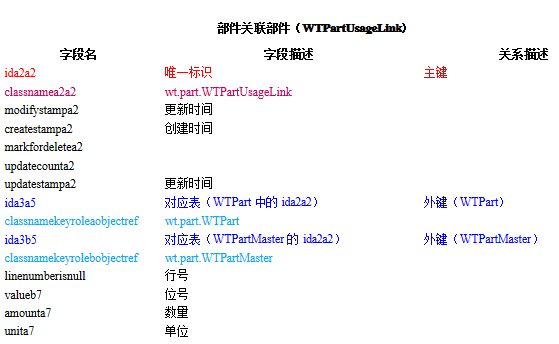
ida2typedefinitionreference 对应表（WTTypeDefinition 中 ida2a2） 外键（WTTypeDefinition）





### WTPartUsageLink

部件关联部件



select \* from wtpartmaster pmt where pmt.ida2a2 in(

select pul.ida3b5 from wtpart pt,wtpartmaster ptm,wtpartusagelink pul where

pt.ida3masterreference=ptm.ida2a2

and pul.ida3a5=pt.ida2a2 and pt.latestiterationinfo=1 and ptm.wtpartnumb

er='GC000040'

)

Wtchangeactivity2 -->ECA

Wtchangeorder2 --->ECN

Wtchangerequest2 --->ECR

ChangeRecordIfc --->记录ECA与产生对象之间的关系

AffectedActivityData ---->记录ECA与受影响对象之间的关系

IncludedInIfc --->记录ECN和ECA之间的关系

RelevantRequestData --->ECR和受影响对象之间的关系

WTPart 部件一小部分：像小版本，关联很多表

Wtpartmaster 部件重要部分：像部件名称，编号等

WTDocument

Wtdocumentmaster

EPMDocument Epmdocumentmaster Epmbuildrule Epmbuildhistory Epmdescribelink --->ProE相关

WfProcess Wfprocesstemplate -- 流程实例 流程模板

WorkItem -- 任务

Wfassignedactivity Wfassignedactivitytemplate --活动对象 活动模板

Wfexpressionrobot Wfexprrobottemplate -- 表达式对象 表达式模板

Promotionnotice Promotiontarget --升级请求对象表 升级请求对象和升级对象关系表

Wtpartusagelink --部件使用关系表

WTPartReferenceLink:部件与参考文档之间的关系

WTPartDescribeLink:部件与描述文档之间的关系

WTDocumentUsageLink:文档结构关系

WTDocumentDependencyLink:文档之间的参考关系

WTUser 用户表

WTGroup 群组

Ballots

select \* from MembershipLink --上下文团队关系角色成员关系

PDMLInkProduct -- 产品

Wtlibrary --存储库

Project2 --项目

-- 软属性部件相关表

StringDefinition StringValue

FloatDefinition FloatValue

RefenceDefintion

select \* from WTTypeDefinition -- 类型表

LocationChangeRecord 移动历史记录表

IdentityChangeHistory 重命名历史记录表

MadeFromLink 另存为对象之间的关系

LifeCycleHistory 成熟度历史记录

**自己测试过的**

**开启工作流模板会在这四张表建立一条记录：**

**WorkItem:工作项：此表主要记录了OID唯一标识**

**workItemLink**【工作项的链接】

**WfProcess：**流程实例 开启了工作流模板的记录 ：**查看此模板记录，有开启的话：删除不了工作模板。**

**WfAssignmentEventAudit【作业活动的审计】**这个有记录但是还想没什么用

**创建了模板会在此两个表建立记录：**

WFPROCESSTEMPLATEMASTER【流程模板的管理】记录工作流模板名称等内容

**WFProcessTemplate：【工作流模板】：有小版本啊记录，和部件的记录一样的**

WFProcessRequestER:【流程请求者】

**Oracle插入记录查询**

select \* from v$sql

# windchill10.1的目录结构及使用

**Windchill环境变量设置**

**JAVA\_HOME** windchill使用的java SDk所在目录，如果使用PTC提供的java SDK，则为：ptc\Windchill\_10.1\java

**WT\_HOME**

Windchill 应用程序所在目录，一盘为C:\ptc\windchill\_10.1\Windchill

**ANT\_HOME**

Windchill使用的ANT应用程序所在的目录，默认为%WT\_HOME%\ant

**Windchill目录**

**Bin –** 包含批处理脚本，如Tools.bat,Windchill.exe等

**Codebase** 【代码库】– 包含Windchill运行时所需要的文件（后文详细介绍）

**Db** – 包含数据库配置文件和Sql脚本

**installer**- 包含安装、更新时产生的日志文件

**Lib** – 包含windchill客户端需要使用的java Beans

**LoadFiles/loadXmlFiles** – 包含系统初始化数据文件

**Logs** – 包含Windchill运行时的日志文件

**Src** – 后文详细介绍

**Taskeditor【task editor：任务编辑器】** –IE\*TASK编辑器

**Task【任务、工作】** – IE\*TASK脚本文件

**Tomcat** = 内置的TOMCAT所在目录

**Temp** - Windchill 默认的临时文件所在目录

**Utilities** – 包含多个工具（如Windchill配置助手）

**Codebase目录**

**Com\ptc和wt-Windchill**运行时所调用的类文件

**Config** – Windchill运行时使用的配置文件

**Netmarkets和wtcore** – Windchill运行时使用的页面文件（JSp、Jacascript、css以及图像等）

**Templates** – Windchill template技术使用的HTML文件

**Codebase 目录下主要包含下述类型的文件（web的url：根目录）**

扩展名 说明 扩展名 说明

.class java 类文件 .xml xml文件

.jsp/.jspf jsp文件 <包名>Resource.class 资源文件

.js/.jsfrag javascript脚本 <包名>ModeIRB.RB.ser 模型资源文件

.css/.xsi 样式文件 <枚举类型名>RB.RB.ser 枚举资源文件

.gif/.png 图片 .html 静态页面文件

**Src**目录主要包含了一些多国语言化得资源文件

**-.rbInfo**  - 为枚举类型的多国语言化资源文件

**.java**  - 为Windchill UI 多国语言化得资源文件

**重要的.properties文件**

**Wt.properties** 包含主要的Java配置项和Windchill系统配置项

**Service.properties** 包含Windchill动态代理模式的配置项

**Db.properties** 包含数据库的配置信息

**使用Eclipse进行开发**

下载Eclipser并安装插件

Eclipse IDE for java EE DEVElopers 版本

使用下述插件帮助开发

Aptana plug-in for Eclipse

（<http://download.aptana.com/tools/studio/plugin/install/studio/>）

fileSync Plug-in for Eclipse(<http://andrei.gmxhome.de/eclipse/>)

SQLExplorer P;ug –in for Eclipse(<http://eclipsesql.sourceforge.net/>)

Subclipse Plug – In for Eclipse(<http://subclipse.tigris.org/update_1.8.x/>)

**创建Windchill项目**

在Windchill Shell中执行tools eclipse\_project

**导入Windchill项目到Eclipse**

1. 打开Eclipse，并将工作区切换至X：\ptc\Windchill\_10.1\eclipse
2. 在Eclipse选择File 🡪Import… 🡪General 🡪 Existing Projects into Workspace
3. 选择cust\_Windchill\_src,并点击Finish
4. 将项目文件的编码修改为utf – 8

**重新配置类路径**

1. 将工作区目录（一般是X:\ptc\windchill\_10.1\eclipse）的只读属性取消
2. 移除类路径中的codebase目录项
3. 使用下述ant脚本打包codebase目录下的.class文件，并添加到项目类路径中

<project name=”makeCCjars”

default=”makeCCJars”

basedir=”X:/ptc/Windchill\_10.1/Windchill/codebase”>

<target name=”makeCCjars”>

<jar destfile=”X:/ptc/Windchill\_10.1/Windchill/x22codebase.jar”

Basedir=” X:/ptc/Windchill\_10.1/Windchill/codebase” excludes=”\*\*/ext/\*”

Includes=”\*\*/\*.class” />

**示例程序**

编写HelloWindchill应用程序

Public class HElloWindchill implements RemoteAccess{

Public static void main(String[] args) throws RemoteException,IncocationTargetException{

RemoteMethodServer rms= RemoteMethodServer.getDefault();

Rms.invoke(“sayHello”,HelloWindchill.class.getName(),null, new Class[]{},new Object[] {});

}

Public static void satHello() throws WTException{

WTPrincipal loginUser=SessionHelper.manager.getPrincipal();

System.out.println(“Hello,”+loginUser.getName());

1、Windchill10.2目录下的：Windchill\codebase是web上下文（url根路径），

2、（新装）C:\Windows\System32\drivers\etc\hosts文件中部分内容：

127.0.0.1 localhost

# ::1 localhost

10.0.8.15 mail.liyuanheng.com