ptc

DIGITAL TRANSFORMS PHYSICAL

用PQL进行WINDCHILL开发

DEVELOPMENT GUIDE

PTC DTAS

-

AGENDA

- Windchill数据库操作
- 常见困扰
- PQL开发环境配置
- PQL的Hello World
- 多对象(BO)查询
- 对象和表的混合查询
- 子查询
- PQL查询最新版本
- 自定义DB函数&JPQL查询
- Repository功能

常见困扰

Windchill的数据层采用的自己专有的O/R Mapping，在进行新功能开发时就必须得使用PTC的标准接口(wt.fc.*)或者JDBC进行数据库操作:

- 标准接口(wt.fc.*),能够方便的进行简单的业务查询和数据操作；但是，当查询稍微比较复杂，标准接口(QuerySpec)的代码就变得极为冗长，且难以阅读；有时候还存在参数绑定的错误；
- JDBC，能够方便的进行数据操作，并且性能优良；但是，Windchill基本上所有的业务处理都是围绕业务对象进行的，所以使用JDBC就会频繁的进行O/R转换；即麻烦又耗费性能；
- Info*Engine使用就更少了；性能低下，使用复杂，功能不完整，难以阅读，风格怪异；

PQL部署

Windchill配置： jpql-1.1.3.jar

- 复制jpql-1.1.3.jar、antlr4-4.5.3.jar到$WT_HOME/codebase/WEB-INF/lib目录。
- 注册服务：

```
xconfmanager -s
wt.services.service.2090=com.ptc.xworx.pql.XPersistenceManager/com.ptc.xworx.pql.StandardXPersistenceMan
ager -t codebase/wt.properties -p
```

- 设置缓存条目(默认500)：

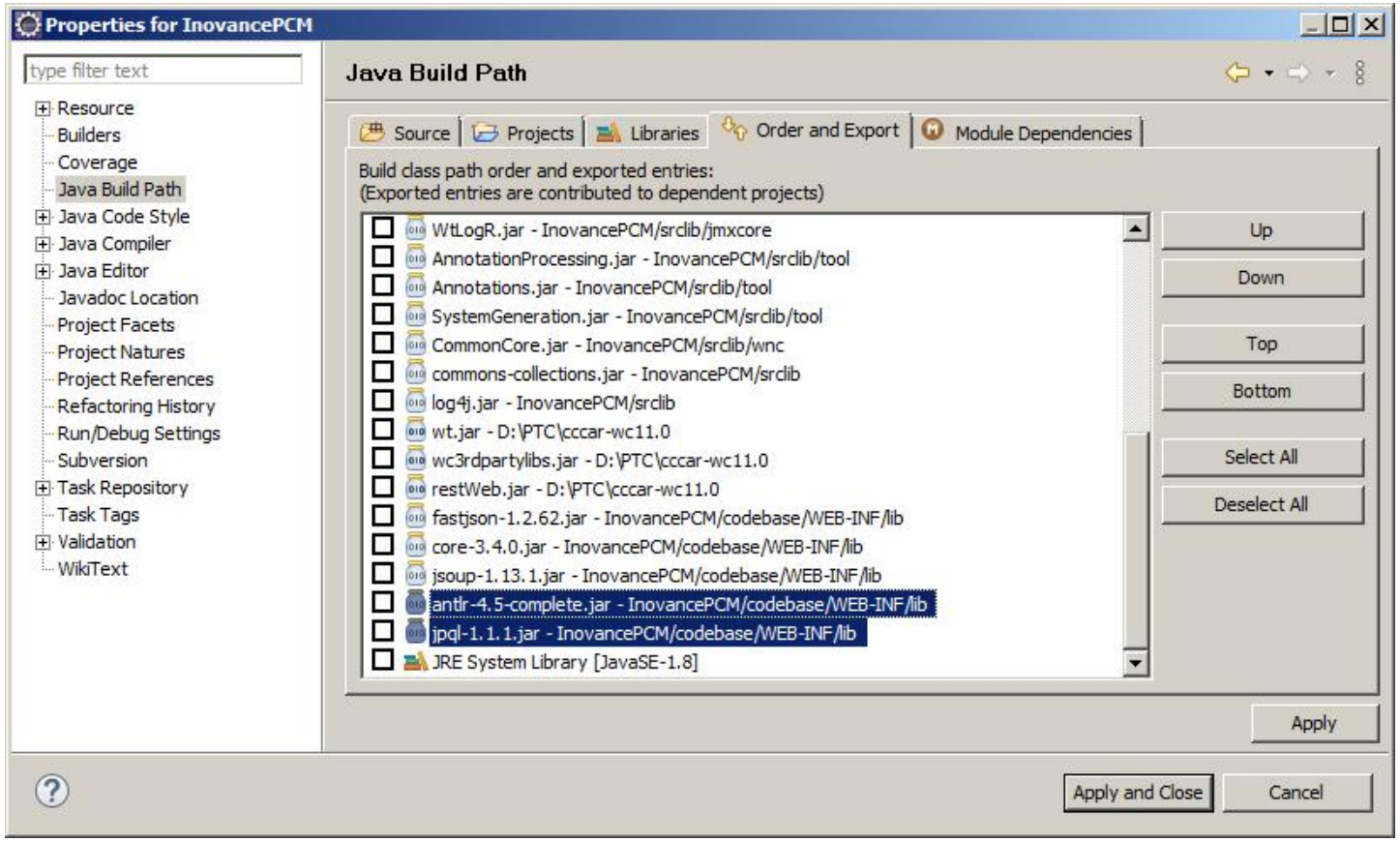
```
xconfmanager -s com.ptc.xworx.pql.cache.count=500 -t codebase/wt.properties -p
```

- 重启Windchill

PQL开发环境配置

PQL开发环境配置就比较简单：

- Eclipse环境，将jpql-1.1.3.jar、antlr4-4.5.3.jar中的lib目录下的jar包添加进项目；
- 其他开发环境类似；



PQL的HELLO WORLD

让我们开始二个PQL查询：

```
QueryResult qr = XPersistenceHelper.manager.query("select a from wt.part.WTPart a");
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    WTPart part = (WTPart) objs[0];
    System.out.println(part.getNumber());
}

QueryResult qr = XPersistenceHelper.manager.query("select a.WTPARTNUMBER from WTPARTMASTER a");
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    String wtpartnumber = (String) objs[0];
    System.out.println(wtpartnumber);
}
```

PQL即能够对业务对象进行查询，也能够对物理表进行查询：

- 对业务对象进行查询，FROM子句必须是业务对象的类全名(FROM wt.part.WTPart A)；如要返回对象，别名必须填入SELECT子句(SELECT A)【见上例】
- 对物理表的直接查询，没有太多要求，参照SQL的写法即可
- FROM子句的表或对象必须要有别名

多业务对象组合查询

同样JPQL也可以和SQL一样对多个表和多业务对象进行连接查询：

```
StringBuffer jpql = new StringBuffer("select b from wt.part.WTPart a, wt.part.WTPartMaster b ");
jpql.append("where a.IDA3MASTERREFERENCE=b.IDA2A2 and b.WTPARTNUMBER in('D3_0000000170', 'D3_0000000180')");
QueryResult qr = XPersistenceHelper.manager.query(jpql.toString());
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    WTPartMaster master = (WTPartMaster) objs[0];
    System.out.println(master.getNumber());
}
```

```
StringBuffer jpql = new StringBuffer("SELECT B.WTPARTNUMBER FROM WTPart A, WTPartMaster B ");
jpql.append("WHERE A.IDA3MASTERREFERENCE = B.IDA2A2 AND B.WTPARTNUMBER IN('D3_0000000170', 'D3_0000000180')");
QueryResult qr = XPersistenceHelper.manager.query(jpql.toString());
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    String wtpartnumber = (String) objs[0];
    System.out.println(wtpartnumber);
}
```

- 对对象进行查询时，SELECT子句中既可以是对象别名(返回整个对象)，也可以是对象在数据库表中的字段名
- 对表进行查询，SELECT子句中必须时表的字段名

对象和表的混合查询

JPQL能够方便的进行对象和表的混合查询，比QuerySpec进行混合查询简单得多：

```
StringBuffer jpql = new StringBuffer("SELECT A, B.NAME FROM wt.part.WTPart A, WTPartMaster B ");
jpql.append("WHERE A.IDA3MASTERREFERENCE = B.IDA2A2 AND B.WTPARTNUMBER IN('D3_0000000170', 'D3_0000000180')");
QueryResult qr = JPersistenceHelper.manager.query(jpql.toString());
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    WTPart part = (WTPart) objs[0];
    String name = (String) objs[1];
    System.out.println(part.getNumber() + "  -  " + name);
}
```

```
D3_0000000170  -  Tube, 1/2 IN ID X 3/4 IN OD
D3_0000000180  -  Wiring Harness, Power Output
```

子查询

JPQL可以在FROM子句中使用子查询：

```
StringBuffer jpql = new StringBuffer();
jpql.append("SELECT A, B, C.CHARACTERISTIC, 'WWW.GOOGLE.COM' GOOG");
jpql.append("  FROM wt.part.WTPart A,");
jpql.append("       wt.part.WTPartMaster B,");
jpql.append("       (SELECT C.NUMBER, C.CHARACTERISTIC FROM CV_TABLE C WHERE C.NUMBER IN('XXX1TO', 'XXX2TO')) C");
jpql.append(" WHERE A.IDA3MASTERREFERENCE = B.IDA2A2");
jpql.append("   AND A.LATESTITERATIONINFO = 1");
jpql.append("   AND B.WTPARTNUMBER = C.NUMBER");
jpql.append("   AND C.NUMBER IN('XXX1TO')");
jpql.append(" ORDER BY C.CHARACTERISTIC ASC");
```

- 子查询的SELECT子句中必须时字段名，不能够是对象；
- 子查询的FROM子句可以是对象名也可以是表名，但是SELECT子句必须时字段；

自定义DB函数&PQL对象查询

Windchill的OOTB功能的限制，QuerySpec只能够使用有限的数量的DB函数，在JPQL进行对象查询时也只能够使用Windchill中定义的函数；若需要使用特殊功能的DB函数就需要在userDefinedFunctions.xml中定义，例如：

使用ROW_NUMBER()查询最新创建的10个WTPartMaster对象：

```
public static void main(String[] args) throws WTPropertyVetoException, WTException {
    StringBuffer jpql = new StringBuffer();
    jpql.append("select c from wt.part.WTPartMaster c where c.IDA2A2 in (");
    jpql.append("    select b.IDA2A2 from (");
    jpql.append("        select a.IDA2A2, ROW_NUMBER() over (order by a.CREATESTAMPA2 desc) ROWN from WTPARTMASTER a) b");
    jpql.append("     where b.ROWN <= 10)");

    QueryResult qr = XPersistenceHelper.manager.query(jpql.toString());
    while (qr.hasMoreElements()) {
        Object[] objs = (Object[]) qr.nextElement();
        WTPartMaster master = (WTPartMaster) objs[0];
        System.out.println(master);
    }
}
```

因为ROW_NUMBER()在Windchill中不是默认支持，所以需要在$WT_HOME/db/userDefinedFunctions.xml定义：

```
<Function name="ROW_NUMBER" minArgs="0" maxArgs="0" type="java.lang.String">
    <FunctionMapping datastore="Oracle" name="ROW_NUMBER()"/>
    <FunctionMapping datastore="SQLServer" name="ROW_NUMBER()"/>
    <FunctionMapping datastore="PostgreSQL" name="ROW_NUMBER()"/>
</Function>
```

ORDER BY

PQL能够方便的对查询结果进行排序：

```
StringBuffer jpql = new StringBuffer();
jpql.append("SELECT C FROM wt.part.WTPartMaster C WHERE C.IDA2A2 IN (");
jpql.append("    SELECT B.IDA2A2 FROM (");
jpql.append("        SELECT A.IDA2A2, ROW_NUMBER() OVER (ORDER BY A.WTPARTNUMBER DESC) ROWN FROM WTPARTMASTER A) B");
jpql.append("     WHERE B.ROWN <= 10)");
jpql.append(" ORDER BY C.IDA2A2 DESC");
QueryResult qr = XPersistenceHelper.manager.query(jpql.toString());
while (qr.hasMoreElements()) {
    Object[] objs = (Object[]) qr.nextElement();
    WTPartMaster master = (WTPartMaster) objs[0];
    System.out.println(master + "           " + master.getNumber() + "    " + master.getName());
}
```

```
wt.part.WTPartMaster:70835          WCDS000740    01-52115.prt
wt.part.WTPartMaster:70827          WCDS000758    01-51258b.prt
wt.part.WTPartMaster:70823          WCDS000760    oil_pan.prt
wt.part.WTPartMaster:70761          WCDS000766    01-51257.prt
wt.part.WTPartMaster:70749          WCDS000752    01-512118a.prt
wt.part.WTPartMaster:70730          WCDS000765    01-51251b.prt
wt.part.WTPartMaster:70695          WCDS000746    01-51216_5.prt
wt.part.WTPartMaster:70670          WCDS000748    01-512138.asm
wt.part.WTPartMaster:70654          WCDS000759    01-52103.prt
wt.part.WTPartMaster:70581          WCDS000749    01-2_cam_intake.prt
```

REPOSITORY

PQL访问数据库带来了多的方便性，但任然存在着一些问题：

- PQL开发中，大量的工作量是用java去构建进行pql脚本，java&pql代码混合在一块；
- PQL查询数据库的java代码调用方式基本都是类似的；
- PQL的返回类型是QueryResult包装的Object[]数组，需要代码去处理返回数据；

基于上述原因，为此引入了Repository功能来对pql进行优化：

- PQL被集中定义的在接口PqaRepository的方法注解@Query中；
- 接口PqaRepository中的方法不需要去实现，可以直接去访问返回JPQL数据；
- Repository根据接口方法中定义的返回类型，直接转换QueryResult为返回对象；

REPOSITORY

基本使用

自定义Repository接口必须扩展[extends]于：com.ptc.xworx.pql.repo.PqaRepository;

- 接口方法需要有@Query注解，注解的value属性中指定pql;

- 接口方法返回类型通常是QueryResult，如果返回类型是其他类型，Repository会尝试去进行类型转换；

例如：如果右边例子的定义如下：

```
@Query("select a from wt.doc.WTDocumentMaster a")
WTDocumentMaster queryWTDocumentMaster();
```

返回的数据就是从查询结果中挑选第一条返回；

WTDocRepository.java

```java
package com.xxx.doc;

import com.ptc.xworx.pql.repo.PqaRepository;
import com.ptc.xworx.pql.repo.Query;

import wt.fc.QueryResult;

public interface WTDocRepository extends PqaRepository {
    @Query("SELECT A FROM wt.doc.WTDocumentMaster A")
    QueryResult queryWTDocumentMaster();
}
```

- Repository的访问都是通过Proxy进行执行的，在调用Repository接口方法之前必须使用XPersistenceHelper建立代理（示例代码见下图）

```java
public static void main(String[] args) throws WTException {
    WTDocRepository repository = XPersistenceHelper.getRepositoryObject(WTDocRepository.class);
    QueryResult qr = repository.queryWTDocumentMaster();
    while (qr.hasMoreElements()) {
        Object[] array = (Object[]) qr.nextElement();
        System.out.println(array);
    }
}
```

REPOSITORY

使用@Param传参

自定义Repository接口方法的可以传递参数给pql，Repository会根据接口方法传递的参数进行pql查询

- 方法参数必须加上@Name注解来定义参数的名称；例如：@Param("number")
- @Query中的pql可以使用@Name注解的参数，":" + Param注解值；（见下图红框）

```
public interface WTPartRepository extends PqaRepository {
    @Query("select a from wt.part.WTPartMaster a where a.WTPartNumber = :number")
    WTPartMaster getWTPartMaster(@Param("number") String number);

    @Query("select a from wt.part.WTPart a where a.LATESTITERATIONINFO=1 and a.IDA3MASTERREFERENCE=:master.persistInfo.objectIdentifier.id order by a.IDA2A2 desc")
    QueryResult getVersionsByNumber(@Param(value = "master", handle = "getWTPartMaster") String number);

    @Query("select a from wt.part.WTPart a where a.[master>number]=:number order by a.IDA2A2 desc")
    QueryResult getIterationsByNumber(@Param("number") String number);
}
```

- 接口方法的参数可以对象，@Query中可以通过“:master.persistInfo.objectIdentifier.id”方式来访使用参数对象（见上图绿框）；

REPOSITORY

使用方法传参

使用PQL进行复杂的复合查询时，QL代码就会很长，难以看懂；为此引入@Param handle的功能来解决这个问题：

- 将复合查询中的子查询拆分成单独的方法，通过@Param handle对子查询方法的结果进行引用；
- 复杂特殊参数预处理，通过@Param handle调用静态方法，@Query使用handle的返回结果；

```
public interface WTPartRepository extends PqaRepository {
    @Query("select a from wt.part.WTPartMaster a where a.WTPartNumber = :number")
    WTPartMaster getWTPartMaster(@Param("number") String number);

    @Query("select a from wt.part.WTPart a where a.LATESTITERATIONINFO=1 and a.IDA3MASTERREFERENCE=:master.persistInfo.objectIdentifier.id order by a.IDA2A2 desc")
    QueryResult getVersionsByNumber(@Param(value = "master", handle = "getWTPartMaster") String number);

    @Query("select a from wt.part.WTPart a where a.[master>number]=:number order by a.IDA2A2 desc")
    QueryResult getIterationsByNumber(@Param("number") String number);
}
```

REPOSITORY

使用占位符

PQL支持使用占位符的方式去动态构建QuerySpec：

- PQL中的占位符必须以“@”作为前缀进行定义(见下图)；
- 占位符必须使用在@Query中；
- 占位符在转换成QuerySpec之前会被@Param替换；

```
@Query("select a from wt.part.WTPart a where a.latestiterationinfo = 1 and " //
        + "a.oneOffVersionInfo.identifier.oneOffVersionId=null and " //
        + "a.branchida2typedefinitionrefe=:typeId and a.@field=:value and " //
        + "a.versionsortida2versioninfo in(select MAX(b.versionsortida2versioninfo) from wt.part.WTPart b " //
        + "where a.ida3masterreference=b.ida3masterreference and b.oneoffversionida2oneoffversi=null)")
QueryResult queryWTPartByAttribute(@Param("field") String field, @Param("value") Object value, @Param("typeId") long typeId);
```

REPOSITORY

PQL设置QuerySpec参数

PQL支持直接去设置QuerySpec的参数：

- 设计QuerySpec中的参数以“$”作为前缀进行定义(见下图)；
- 占位符必须使用在@Query中；
- 占位符在转换成QuerySpec之时，QuerySpec的方法会被调用；
  - 下图中的$DescendantQuery=false将调用QuerySpec的setDescendantQuery()方法；

```
@Query("select a from wt.part.WTPart a where $DescendantQuery=false and a.latestiterationinfo = 1 and " //
        + "a.oneOffVersionInfo.identifier.oneOffVersionId=null and a.branchida2typedefinitionrefe=:typeId and " //
        + "a.@field=:value and a.versionsortida2versioninfo in " //
        + "(select MAX(b.versionsortida2versioninfo) from wt.part.WTPart b where " //
        + "$DescendantQuery=false and a.ida3masterreference=b.ida3masterreference and b.oneoffversionida2oneoffversi=null)")
QueryResult getWTPartByAttribute(@Param("field") String field, @Param("value") Object value, @Param("typeId") long typeId);
```

PERMISSION

PQL支持带权限查询和忽略权限查询：

- 忽略权限查询：
  - XPersistenceHelper.manager.query();
- 不忽略权限查询：
  - XPersistenceHelper.manager.find();
- @Query默认是带权限查询：
  - @Query("select a from wt.part.WTPartMaster a")
- @Query忽略权限查询：
  - @Query(value="select a from wt.part.WTPartMaster a", permission=false)

ptc

DIGITAL TRANSFORMS PHYSICAL

THANK YOU

ptc.com