XSQL开发说明

# XSQL创建数据库对象

在项目（Web服务或桌面应用）启动时，自动判定数据库对象是否存在，当对象不存在时执行创建SQL创建对象。

方便好用的功能，只须给定一个空数据库，即能在项目启动时，构造完整个数据库。

数据库对象包括：表、视图、过程、函数、序列（Oracle）、索引、触发器、约束等数据概念的对象。

支持数据库类型如下：

1. Oracle
2. MySQL
3. SQLServer
4. DB2
5. SQLite
6. PostgreSQL

用此方法创建的数据库对象，均可通过 管理页面（<http://127.0.0.1:80/WebName/analyses>）反复重新创建（会先删除后重新创建），如下图。

管理页面需在web.xml中添加配置后，才能正常使用。配置 方法见org.hy.common.xml.plugins.analyse.AnalyseObjectServlet类中的说明。



**XML配置语法：**

1. **<sql>**
3. **<dataSourceGroup** ref="数据库连接池组的对象引用" **/>**
5. **<content>**
6. <![CDATA[
7. 数据库SQL。多个执行SQL间用;/分隔。
8. ]]>
9. **</content>**
11. **<comment>**注释说明。会在数据库对象创建时输出**</comment>**  <!-- 可选配置 -->
12. **<create>**数据库对象名称(不区分大小写)**</create>**   <!-- 此行请放在最后 -->
14. **</sql>**

**XML配置说明：**

1. <content>节点定义SQL语句，多个执行SQL语句间用分号加除号（**;/**）分隔。这样就能更好的支持数据库存储过程及函数的构造。
2. <create>节点一定要放在最后，当解析器遇到<create>节点时，就立刻执行<content>节点中定义的SQL语句，及显示<comment>节点的注释说明。
3. <create>节点中定义的数据库对象名称，会在<content>节点中定义的SQL中查找匹配，当未查找到时，会错异常，表示创建对象名称与创建SQL不匹配。
4. 请尽量在<content>节点中编写可重复数次执行的SQL语句。如，创建数据库对象A时，要向B表中插入一条配置数据，B表中也只能有一条对应数据库对象A的配置数据。此时，应当在INSERT语句前，再添加一条DELETE语句，预防性的先删除B表的配置数据后，再插入配置数据，防止重复数据。

**XML配置举例：**

1. **<sql** id="XSQL\_Create\_TActivityType"**>**
3. **<dataSourceGroup** ref="DSG\_XFlow" **/>**
5. **<content>**
6. <![CDATA[
7. CREATE TABLE TActivityType
8. (
9. activityTypeID     VARCHAR2(64)     NOT NULL
10. ,activityType       NVARCHAR2(128)   NOT NULL
11. ,infoComment        NVARCHAR2(2000)
12. ,orderNo            NUMBER(10)
13. );/
15. ALTER TABLE TActivityType ADD CONSTRAINT PK\_TActivityType PRIMARY KEY (activityTypeID);/
17. COMMENT ON TABLE  TActivityType                IS '工作流活动(节点)类型';/
18. COMMENT ON COLUMN TActivityType.activityTypeID IS '工作流活动类型ID';/
19. COMMENT ON COLUMN TActivityType.activityType   IS '工作流活动类型名称';/
20. COMMENT ON COLUMN TActivityType.infoComment    IS '备注说明';/
21. COMMENT ON COLUMN TActivityType.orderNo        IS '排列顺序';/
22. ]]>
23. **</content>**
25. **<comment>**工作流活动(节点)类型**</comment>**
26. **<create>**TActivityType**</create>**
28. **</sql>**

# XSQL创建多种数据库的对象

上面已经讲了如何创建数据库对象，但如何支持不同类型的数据库对象的创建呢？比如说，项目需要支持SQLServer、Oracle两种数据库的表创建怎么办呢？放心XSQL是支持的。

**XML配置语法：**

1. **<sql>**
3. **<dataSourceGroup** ref="DSG\_My" **/>** <!-- 数据库连接池组的对象引用 -->
5. **<content** if="ORACLE == DSG\_My.getDbProductType"**>**
6. <![CDATA[
7. Oracle数据库的SQL语句。多个执行SQL间用;/分隔。
8. ]]>
9. **</content>**
11. **<content** if="SQLSERVER == DSG\_My.getDbProductType"**>**
12. <![CDATA[
13. SQLServer数据库的SQL语句。多个执行SQL间用;/分隔。
14. ]]>
15. **</content>**
17. **<comment>**注释说明。会在数据库对象创建时输出**</comment>**  <!-- 可选配置 -->
18. **<create>**数据库对象名称(不区分大小写)**</create>**   <!-- 此行请放在最后 -->
20. **</sql>**

**XML配置说明：**

1. if关键字等式右侧的Fel表达式为真时，解析器才解析if所在的xml节点。
2. 通过<dataSourceGroup>节点引用或定义的数据库连接池组对象，来判定数据库类型。解析器再根据两个<content>节点定义的解析条件，判定条件真假，加载 不同运行时情况下对应的执行SQL语句。
3. getDbProductType()支持的数据库类型及类型名称，可在org.hy.common.db.DataSourceGroup类中找到定义。

**XML配置举例：**

1. **<sql** id="XSQL\_Create\_TActivityType"**>**
3. **<dataSourceGroup** ref="DSG\_XFlow" **/>**
5. **<content** if="ORACLE == DSG\_XFlow.getDbProductType"**>**
6. <![CDATA[
7. CREATE TABLE TActivityType
8. (
9. activityTypeID     VARCHAR2(64)     NOT NULL
10. ,activityType       NVARCHAR2(128)   NOT NULL
11. ,infoComment        NVARCHAR2(2000)
12. ,orderNo            NUMBER(10)
13. );/
15. ALTER TABLE TActivityType ADD CONSTRAINT PK\_TActivityType PRIMARY KEY (activityTypeID);/
17. COMMENT ON TABLE  TActivityType                 IS '工作流活动(节点)类型';/
18. COMMENT ON COLUMN TActivityType.activityTypeID  IS '工作流活动类型ID';/
19. COMMENT ON COLUMN TActivityType.activityType    IS '工作流活动类型名称';/
20. COMMENT ON COLUMN TActivityType.infoComment     IS '备注说明';/
21. COMMENT ON COLUMN TActivityType.orderNo         IS '排列顺序';/
22. ]]>
23. **</content>**
25. **<content** if="SQLSERVER == DSG\_XFlow.getDbProductType"**>**
26. <![CDATA[
27. CREATE TABLE TActivityType
28. (
29. activityTypeID     VARCHAR(64)     NOT NULL
30. ,activityType       NVARCHAR(128)   NOT NULL
31. ,infoComment        NVARCHAR(2000)
32. ,orderNo            INT
33. );/
35. ALTER TABLE TActivityType ADD CONSTRAINT PK\_TActivityType PRIMARY KEY (activityTypeID);/
37. EXEC P\_TableComment  TActivityType                 ,'工作流活动(节点)类型';/
38. EXEC P\_ColumnComment TActivityType ,activityTypeID ,'工作流活动类型ID';/
39. EXEC P\_ColumnComment TActivityType ,activityType   ,'工作流活动类型名称';/
40. EXEC P\_ColumnComment TActivityType ,infoComment    ,'备注说明';/
41. EXEC P\_ColumnComment TActivityType ,orderNo        ,'排列顺序';/
42. ]]>
43. **</content>**
45. **<comment>**工作流活动(节点)类型**</comment>**
46. **<create>**TActivityType**</create>**
48. **</sql>**

注：P\_TableComment和P\_ColumnComment是设置表或表字段注释的两个存储过程，因SQLServer数据库添加注释的命令太过于复杂而特意编写的。这两个存储过程也是用XSQL配置构建的。

# XSQL创建多种数据库的对象并初始化数据

上面已经讲了如何创建多种数据库的对象。又如何方便的初始化表中的数据呢？不同数据库的INSERT语句格式是一样，上面针对数据库类型分开编写对应SQL语句，难道INSERT语句也是重复的写两边吗？答案，当然是不用的。

这时就要用到XSQL的 “应用级触发器”功能了。具体功能不在此赘述，请详见org.hy.common.xml.XSQLTrigger类中的说明。

**XML配置语法：**

1. **<sql>**
3. **<dataSourceGroup** ref="DSG\_My" **/>** <!-- 数据库连接池组的对象引用 -->
5. **<content** if="ORACLE == DSG\_My.getDbProductType"**>**
6. <![CDATA[
7. Oracle数据库的SQL语句。多个执行SQL间用;/分隔。
8. ]]>
9. **</content>**
11. **<content** if="SQLSERVER == DSG\_My.getDbProductType"**>**
12. <![CDATA[
13. SQLServer数据库的SQL语句。多个执行SQL间用;/分隔。
14. ]]>
15. **</content>**
17. <trigger>
18. <create ref="另一个XSQL对象的引用"/> <!-- 可在此XSQL对象中执行相同的SQL语句 -->
19. <create ref="支持多个应用级触发器"/>
20. </trigger>
22. **<comment>**注释说明。会在数据库对象创建时输出**</comment>**  <!-- 可选配置 -->
23. **<create>**数据库对象名称(不区分大小写)**</create>**   <!-- 此行请放在最后 -->
25. **</sql>**

**XML配置说明：**

1. <trigger>节点将在本节点XSQL执行后执行，类似于数据库的After触发器。
2. <trigger>节点中可以添加一个或多个<create>节点。支持多个应用级触发器。
3. <trigger>节点中的<create>节点的数据库可以与本XSQL节点的数据库不一样，即支持跨库操作。

**XML配置举例：**

1. **<sql** id="XSQL\_Create\_TActivityType\_Init\_Insert"**>**
3. **<dataSourceGroup** ref="DSG\_XFlow" **/>**
5. **<content>**
6. <![CDATA[
7. INSERT  INTO TActivityType VALUES('AT001' ,'开始' ,1);/
8. INSERT  INTO TActivityType VALUES('AT999' ,'结束' ,999);/
9. ]]>
10. **</content>**
12. **<comment>**工作流活动(节点)类型表的初始化数据**</comment>**
14. **</sql>**


18. **<sql** id="XSQL\_Create\_TActivityType"**>**
20. **<dataSourceGroup** ref="DSG\_XFlow" **/>**
22. **<content** if="ORACLE == DSG\_XFlow.getDbProductType"**>**
23. <![CDATA[
24. CREATE TABLE TActivityType
25. (
26. activityTypeID     VARCHAR2(64)     NOT NULL
27. ,activityType       NVARCHAR2(128)   NOT NULL
28. ,infoComment        NVARCHAR2(2000)
29. ,orderNo            NUMBER(10)
30. );/
32. ALTER TABLE TActivityType ADD CONSTRAINT PK\_TActivityType PRIMARY KEY (activityTypeID);/
34. COMMENT ON TABLE  TActivityType                 IS '工作流活动(节点)类型';/
35. COMMENT ON COLUMN TActivityType.activityTypeID  IS '工作流活动类型ID';/
36. COMMENT ON COLUMN TActivityType.activityType    IS '工作流活动类型名称';/
37. COMMENT ON COLUMN TActivityType.infoComment     IS '备注说明';/
38. COMMENT ON COLUMN TActivityType.orderNo         IS '排列顺序';/
39. ]]>
40. **</content>**
42. **<content** if="SQLSERVER == DSG\_XFlow.getDbProductType"**>**
43. <![CDATA[
44. CREATE TABLE TActivityType
45. (
46. activityTypeID     VARCHAR(64)     NOT NULL
47. ,activityType       NVARCHAR(128)   NOT NULL
48. ,infoComment        NVARCHAR(2000)
49. ,orderNo            INT
50. );/
52. ALTER TABLE TActivityType ADD CONSTRAINT PK\_TActivityType PRIMARY KEY (activityTypeID);/
54. EXEC P\_TableComment  TActivityType                 ,'工作流活动(节点)类型';/
55. EXEC P\_ColumnComment TActivityType ,activityTypeID ,'工作流活动类型ID';/
56. EXEC P\_ColumnComment TActivityType ,activityType   ,'工作流活动类型名称';/
57. EXEC P\_ColumnComment TActivityType ,infoComment    ,'备注说明';/
58. EXEC P\_ColumnComment TActivityType ,orderNo        ,'排列顺序';/
59. ]]>
60. **</content>**
62. **<trigger>**
63. **<create** ref="XSQL\_Create\_TActivityType\_Init\_Insert"**/>**
64. **</trigger>**
66. **<comment>**工作流活动(节点)类型**</comment>**
67. **<create>**TActivityType**</create>**
69. **</sql>**

注：应用级触发器执行的XSQL对象XSQL\_Create\_TActivityType\_Init\_Insert应在触发源的XSQL对象XSQL\_Create\_TActivityType之前被解析构造出来。

# XSQL执行多个SQL语句

XSQL被Java调用一次，有能力顺次向数据库发起多个SQL语句的执行命令。通常来说，这些SQL语句均为非查询语句的DDL、DML、DCL、TCL语句等。

每个SQL语句以分号加除号 **;/** 结尾，表示一个完整SQL语句的结束。

为什么使用 **;/** 两个符号组成的标记表示结尾呢？原因是，只有这样才能更好的支持数据库存储过程及函数等复杂对象的构造。

Java在调用此类XSQL时，应使用XSQL.execute()系列的方法执行。此方法只返回是否执行成功。

执行某一个SQL语句异常后，其后的SQL语句将不再执行。

**XML配置举例：**

详见“[XSQL创建数据库对象](#_XSQL创建数据库对象)”

# XSQL常规List查询

最常规的列表查询。

**XML配置语法：**

1. **<sqls>**
3. **<sql>**
5. **<dataSourceGroup** ref="数据库连接池组的对象引用" **/>**
7. **<content>**
8. <![CDATA[
9. SELECT \* FROM 表名
10. ]]>
11. **</content>**
13. **<result>**
14. **<row>**行数据对应的Java对象**</row>**
15. **<cfill>**setter(colValue)**</cfill>**  <!-- 将字段填充到Java对象的方法 -->
16. **</result>**
18. **</sql>**
20. **</sqls>**

**XML配置说明：**

1. <dataSourceGroup>节点定义操作哪个数据库。是数据库连接池组的对象。
2. <content>节点定义执行SQL语句。用<![CDATA[ ... ]]>它括住SQL语句是为了防止SQL语句中出现特殊字符时，XML解析器无法识别。
3. <result>节点定义查询结果集如何映射为Java对象。所以，此节点只出现在查询的XSQL配置中。
4. <result><row>节点定义数据库表的每行数据映射为Java的哪个对象。其值为Java类的全路径。
5. <result><cfill>节点定义数据库表字段以哪种方法映射为Java对象的属性值。其关键字有以下几种：
   1. row：表示行级对象
   2. row.xxx：中间有个点。xxx表示Java对象的成员属性名称，需定义成员属性的getter()方法才能使用。row.xxx即表示支持面向对象。xxx也应当是数据库查询SQL语句的查询字段之一。
   3. rowNo：表示行号
   4. colNo：表示列号
   5. colName：表示列名称
   6. colValue：表示列上的数值
   7. setter：表示用Java对象的成员属性对应的Setter()方法填充Java对象实例。**setter(colValue)**是固定写法之一。
   8. 自定义字段级填充方法。
      1. 如，<row>节点的行级对象被定义为java.util.HashMap时，<cfill>节点配置为：<cfill>**put(colName ,colValue)**</cfill>。即使用HashMap对象的put方法填充实例。
      2. 如，<row>节点的行级对象被定义为java.util.ArrayList时，<cfile>节点配置为：<cfill>**add(colValue)**</cfill>。即使用ArrayList对象的add方法填充实例。
      3. <cfill>定义的方法名是动态可变的、可自定义的（setter、put、add），可随外界变化而变化，千变万化XSQL都能映射。
6. <result><table>节点定义查询结果集（表级对象）被映射为Java的哪个对象。其值为Java类的全路径。默认值为java.util.ArrayList，所以上面语法中可以不配置。
7. <result><fill>节点定义行级对象填充到表级对象的填充方法名。与<cfill>节点类似，不再赘述了。默认值为**add(row)**，与<result><table>相互配对。

**XML配置举例：**

1. **<sqls>**
3. **<sql** id="XSQL\_XFlow\_ActivityType\_Query"**>**
5. **<dataSourceGroup** ref="DSG\_XFlow" **/>**
7. **<content>**
8. <![CDATA[
9. SELECT  A.activityTypeID
10. ,A.activityType
11. ,A.infoComment
12. ,A.orderNo
13. FROM  TActivityType  A
14. ORDER  BY A.orderNo
15. ]]>
16. **</content>**
18. **<result>**
19. **<row>**org.hy.xflow.engine.bean.ActivityType**</row>**
20. **<cfill>**setter(colValue)**</cfill>**
21. **</result>**
23. **</sql>**
25. **</sqls>**

# XSQL查询结果为Map结构的查询

查询结果为Map结果的查询。 相信你能理解是什么意思。常用于数据字典的查询。因为Map结构可以通过key快速定位value。

**XML配置语法：**

1. **<sqls>**
3. **<sql>**
5. **<dataSourceGroup** ref="数据库连接池组的对象引用" **/>**
7. **<content>**
8. <![CDATA[
9. SELECT \* FROM 表名
10. ]]>
11. **</content>**
13. **<result>**
14. **<table>**java.util.HashMap**</row>**        <!-- 表级的对象类型 -->
15. **<fill>**put(row.成员属性名 ,row)**</fill>** <!-- 行级对象填充到表级对象的填充方法 -->
16. **<row>**行数据对应的Java对象**</row>**
17. **<cfill>**setter(colValue)**</cfill>**  <!-- 用什么方法将每个字段填充到Java对象 -->
18. **</result>**
20. **</sql>**
22. **</sqls>**

**XML配置说明：**

1. 语法中多数参数已在“[XSQL常规List查询](#_XSQL常规List查询)”阐述过，这里只做补充说明。
2. <table>节点定义查询结果集（表级对象）被映射为Java的哪个对象。其值为Java类的全路径。定义它为java.util.HashMap的结构来存储数据库查询结果集。当然，换成java.util.Hashtable也是可以的。
3. <fill><fill>节点定义行级对象填充到表级对象的填充方法名。定义它使用HashMap.put()方法将每一行数据存放在表级对象中，即存储在HashMap中。
4. row.成员属性名：成员属性名为Java对象的成员属性名称，需定义成员属性的getter()方法才能使用。Java对象是<row>节点定义的。成员属性名，也应当是数据库查询SQL语句的查询字段之一。

**XML配置举例：**

1. **<sql** id="XSQL\_MessageKey\_Query\_SysID"**>**
3. **<dataSourceGroup** ref="DSG\_LogWeb" **/>**
5. **<content>**
6. <![CDATA[
7. SELECT  A.sysid
8. ,A.msgKey
9. FROM  TMessageKey  A
10. WHERE  A.sysid IS NOT NULL
11. AND  A.sid   IS NULL
12. ORDER  BY A.sysid
13. ]]>
14. **</content>**
16. **<result>**
17. **<table>**java.util.HashMap**</table>**
18. **<fill>**put(row.sysid ,row.msgKey)**</fill>**
19. **<row>**org.hy.log.model.MessageKey**</row>**
20. **<cfill>**setter(colValue)**</cfill>**
21. **</result>**
23. **</sql>**

注：上面XSQL返回的Java结构为HashMap<String ,MessageKey>，其HashMap.key为Messagekey.getSysId()方法的值。

# XSQL一对一复合型查询

Java对象A的成员属性类型为Java对象B，对象A与B是一对一关系，对象A包含B。生成这样的数据结构的XSQL查询。