[**C# 操作 XML 数据库类型、Oracle XMLType**](http://www.cnblogs.com/flashelf/archive/2008/09/02/1281670.html)

  最近练习一下 Oracle11g XML 类型的数据、看看这东西到底怎么样

xml 类型很久就有了一直没有关注，有时间正好看看；

这次学习要做的事情

1、设计一个C# 类来生成 XML Schema (XML 架构)

先设计类，然后由类生成表可能是有很多人蒙昧以求的；正好和 ORM 相反

最主要的 XML 架构可以用来约束数据库中的；XML 的有效性

2、基于上一步的 XML Schema 我们创建一个数据表、并包含这个和上边 C# 类对应的的 XML 类型；

3、插入一些数据；

4、改变 C# 的类重新生成 Schema 在更新数据库中的 Schema

这步假设业务变更看看，更改如何进行，是否方便等

5、用 C# 写一个网页，显示这些数据；

6、阶段性总结

好开始：

**一、设计一个C# 类来生成 XML Schema类代码：**

类如下

namespace Model  
{  
      
    /// <summary>  
    /// 电话  
    /// </summary>  
    public class Phone  
    {  
        string \_code;  
        /// <summary>  
        /// 电话号  
        /// </summary>  
        public string Code  
        {  
            get { return \_code; }  
            set { \_code = value; }  
        }  
        PhoneType \_type;  
        public PhoneType Type  
        {  
            get { return \_type; }  
            set { \_type = value; }  
        }          
          
    }  
    /// <summary>  
    /// 电话类型  
    /// </summary>  
    public enum PhoneType  
    {  
        [XmlEnum(Name = "未知")] //XML 序列化用的名称  
        Unknown,  
        [XmlEnum(Name = "移动")]  
        Mobile,  
        [XmlEnum(Name = "固定")]  
        Fixed,  
          
          
          
    }  
    /// <summary>  
    /// 电话集合  
    /// </summary>  
    [XmlRoot("Phones")]  
    public class Phones : List<Phone>  
    {  
        public void Add(string code, PhoneType type)  
        {  
            base.Add(new Phone() { Code = code, Type = type });  
        }  
          
    }  
}

3个类：电话、电话类型（枚举）、电话类型集合；

以 Phones 类生成 Schema  ，用 .net sdk 的 xsd.exe 或自己写代码都可以生成的、我就不多说了；

Schema  这东西如果纯手写也是劳动量巨大的；谢谢.net 为我们提供这个功能吧；

**Schema 如下 ：**

<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="Phones" nillable="true" type="ArrayOfPhone" />  
  <xs:complexType name="ArrayOfPhone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Phone" nillable="true" type="Phone" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:complexType name="Phone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="1" name="Code" type="xs:string" />  
      <xs:element minOccurs="1" maxOccurs="1" name="Type" type="PhoneType" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:simpleType name="PhoneType">  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="未知" />  
      <xs:enumeration value="移动" />  
      <xs:enumeration value="固定" />  
    </xs:restriction>  
  </xs:simpleType>  
</xs:schema>

**二、我们创建一个数据表、并包含这个和上边 C# 类对应的的 XML 类型；**

**2.1 注册架构：**

BEGIN  
    -- 注册架构  
  DBMS\_XMLSCHEMA.registerschema(schemaurl => 'http://www.OracleDemo.com/Phones.xsd',  
                                schemadoc => '<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="Phones" nillable="true" type="ArrayOfPhone" />  
  <xs:complexType name="ArrayOfPhone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Phone" nillable="true" type="Phone" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:complexType name="Phone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="1" name="Code" type="xs:string" />  
      <xs:element minOccurs="1" maxOccurs="1" name="Type" type="PhoneType" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:simpleType name="PhoneType">  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="未知" />  
      <xs:enumeration value="移动" />  
      <xs:enumeration value="固定" />  
    </xs:restriction>  
  </xs:simpleType>  
</xs:schema>',  
                                  
                                local     => TRUE,  
                                gentypes  => TRUE,  
                                genbean   => FALSE,  
                                gentables => TRUE);  
  
END;  
--会建立 xml 描述的【Oracle自定义类型】  
-- 如果用Oracle 工具查看 Types 下会出现一些如 Phone\*\*\*\_T,phone\*\*\*\_coll,ArrayOfPhone\*\*\*\_T 类似名称的 【Oracle自定义类型】  
-- 结构就和xml schema 是一样  
-- gentables => TRUE 还会建立一些表；  
-- 如 create table Phones721\_TAB of SYS.XMLTYPE --物理表  
-- create table SYS\_NTyIVemDaJQXqHZgjqYv+haQ== of Phone711\_T --自定义类型表

**2.2 创建表**

CREATE TABLE XML\_USER\_INFO (  
 NPK integer,  
 USER\_NAME NVARCHAR2(**50**),  
 Phones XMLType,  
 primary key (NPK)  
)  
XMLTYPE COLUMN Phones STORE AS OBJECT RELATIONAL          --以对象关系方式建立，而不是二进制  
XMLSCHEMA "http://www.OracleDemo.com/Phones.xsd"  
ELEMENT "Phones"  
VARRAY Phones.XMLDATA."Phone" STORE AS TABLE XML\_USER\_INFO\_XMLNT01 --将xml中 Phones/Phone 定义为一个数组嵌套表  
/  
-- 返回:成功  
-- 这时 oracle 还会建立一个 XML\_USER\_INFO\_XMLNT01 的【嵌套表】

我建立了一个叫 XML\_USER\_INFO 的表，这个东西假设为一个用户信息表（真正的用户信息不可能这么少列的）

Oracle 建立XMLType 时可以指定以二进制或对象关系方式建立xml 类型，

我这里选择 STORE AS OBJECT RELATIONAL 据说可以提高性能，等待考证

下一面我要给 Phones/Phone 下加一些约束所以将 Phone 定义为一个数组嵌套表；说白了就是定义一个表里面都放置 Phone 类型

**2.3 定义约束**

ALTER TABLE XML\_USER\_INFO\_XMLNT01

       ADD constraint PK\_XML\_USER\_INFO\_XMLNT01 primary key (NESTED\_TABLE\_ID, "Code")

就是每个人的电话号码不能重复； 数据库里的东西要是没有约束是很郁闷的，这里也试验一下这个问题；

constraint PK\_XML\_USER\_INFO\_XMLNT01 可以省略，因为有名字比较容易从异常中看出到底是什么列出错，我一般都会加上这个

除非一个表就一个主键；

 遗憾的是集合类型的元素、不能加外键(就是说如果是 Phones 的属性是可以加外键的 Phone属性，如电话号码，类型什么的就不行，必须和数据库表一行能对应上的才可以加外键否则只能用 schema 约束)；

 外键的例子以后在说吧；

**2.4 本步骤总结**

比较满意、虽然SQL 代码很多不过，schema 是类生成的、能够节省一些设计时间，不过学习成本是必须的；

XMLType 的 schema 并不是必须的，不过没有约束，关系对数据库来说，时间长了是不好维护的尤其是后来人；

**三、插入一些数据**

**3.1 插入数据**

INSERT INTO XML\_USER\_INFO VALUES (**1**,'用户1'  
  
    ,XMLType('<Phones xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http:www.w3.org/2001/XMLSchema">  
  
 <Phone>  
  
    <Code>13940588000</Code>  
  
    <Type>移动</Type>  
  
 </Phone>  
  
 <Phone>  
  
    <Code>024-22222222-1</Code>  
  
    <Type>固定</Type>  
  
 </Phone>  
  
 <Phone>  
  
    <Code>8788888</Code>  
  
    <Type>未知</Type>  
  
 </Phone>  
  
</Phones>'  
  
    )  
  
)  
  
/

**3.2 测试约束有效性（在Type改小灵通）**

INSERT INTO XML\_USER\_INFO VALUES (**2**,'用户2'  
    ,XMLType('<Phones xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http:www.w3.org/2001/XMLSchema">  
  <Phone>  
    <Code>13940588000</Code>  
    <Type>移动</Type>  
  </Phone>  
  <Phone>  
    <Code>13940588000</Code>  
    <Type>固定</Type>  
  </Phone>  
  <Phone>  
    <Code>8788888</Code>  
    <Type>小灵通</Type>  
  </Phone>  
</Phones>'  
    )  
)  
-- 返回错误：ORA-31038: enumeration 值无效: "小灵通"  
-- 证明 PhoneType 的 XMLSchema 约束是有效的（废话）；  
/

….插入我就不多写了都这模样

注意：大家可以看出xmltype这里的数据，就是 Phones 对象xml序列化后的的样子

也就是说我们可以比较方便把类直接插入数据库，如果用平面表这里要执行 1\*n次的 insert

代码例子我就不提供了先，因为太简单了 XMLType( :XMLString ) 然后给对象序列化了

给到 :XMLString 参数里就可以了；

**3.3 本步骤总结**

如果开发的话，可以节省一些语句；尤其适合那种，子表数据一次插入很少的情况

而且用 select 读取的时候，应该可以直接反序列化为对象

能省去往实体类赋值的代码量；

**四、改变 C# 的类重新生成 Schema 在更新数据库中的 Schema**

假设我们的业务更改：那个业务也不可能不变的对吧，所以测试是否容易修改也是必要的

假设，【电话】加入了说明属性、【电话类型】加入了小灵通；集合类没更改；

**4.1 类的更改**

public class Phone  
{  
    string \_code;  
  
    public string Code  
    {  
        get { return \_code; }  
        set { \_code = value; }  
    }  
    PhoneType \_type;  
    public PhoneType Type  
    {  
        get { return \_type; }  
        set { \_type = value; }  
    }  
      
    //新加入   
**string \_make;  
    public string Make  
    {  
        get { return \_make; }  
        set { \_make = value; }  
    }**  
      
}  
  
  
public enum PhoneType  
{  
    [XmlEnum(Name = "未知")]  
    Unknown,  
    [XmlEnum(Name = "移动")]  
    Mobile,  
    [XmlEnum(Name = "固定")]  
    Fixed,  
    **[XmlEnum(Name = "小灵通")] //新加入  
    PHS**      
      
}

**4.2 重新生成 xsd**

方法在上面说过了

**4.3 数据库更改**

declare   
   -- 旧Schema  
  oldSchemaDoc nvarchar2(**2000**) := '<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="Phones" nillable="true" type="ArrayOfPhone" />  
  <xs:complexType name="ArrayOfPhone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Phone" nillable="true" type="Phone" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:complexType name="Phone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="1" name="Code" type="xs:string" />  
      <xs:element minOccurs="1" maxOccurs="1" name="Type" type="PhoneType" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:simpleType name="PhoneType">  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="未知" />  
      <xs:enumeration value="移动" />  
      <xs:enumeration value="固定" />  
    </xs:restriction>  
  </xs:simpleType>  
</xs:schema>';  
 -- 新Schema  
  newSchemaDoc nvarchar2(**2000**) := '<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="Phones" nillable="true" type="ArrayOfPhone" />  
  <xs:complexType name="ArrayOfPhone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Phone" nillable="true" type="Phone" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:complexType name="Phone">  
    <xs:sequence>  
      <xs:element minOccurs="0" maxOccurs="1" name="Code" type="xs:string" />  
      <xs:element minOccurs="1" maxOccurs="1" name="Type" type="PhoneType" />  
      <xs:element minOccurs="0" maxOccurs="1" name="Make" type="xs:string" />  
    </xs:sequence>  
  </xs:complexType>  
  <xs:simpleType name="PhoneType">  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="未知" />  
      <xs:enumeration value="移动" />  
      <xs:enumeration value="固定" />  
      <xs:enumeration value="小灵通" />  
    </xs:restriction>  
  </xs:simpleType>  
</xs:schema>';  
 -- 变量存储 xmldiff xml差异结果  
 diffXMLDoc clob;  
   
 -- url  
 v\_schema\_url nvarchar2(**255**) :=  'http://www.OracleDemo.com/Phones.xsd';  
begin   
   --生成差异结果  
  select xmldiff(xmltype(oldSchemaDoc),xmltype(newSchemaDoc)).getClobVal() into diffXMLDoc from dual;  
  -- 11g 新增加的原地更改函数，性能比原来那个 DBMS\_XMLSCHEMA.copyEvolve  
  -- 概念 执行原地 XML 模式演变 http://www.oracle.com/technology/global/cn/obe/11gr1\_db/datamgmt/xmldb2\_a/xmldb2\_a.htm  
  DBMS\_XMLSCHEMA.inPlaceEvolve( v\_schema\_url ,  xmltype(diffXMLDoc));   
end;

流程就是拿 旧的Schema 和 新的Schema 比较生成一个差异结果  select xmldiff...这里，然后调用DBMS\_XMLSCHEMA.inPlaceEvolve

更新 Schema 很简单函数就用到两个，代码没多少就是2个Schema 占地方；

**4.4 在插入一条数据看看**

--- 在插入一条  
INSERT INTO XML\_USER\_INFO VALUES (**3**,'用户3'  
    ,XMLType('<Phones xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <Phone>  
    <Code>13930588003</Code>  
    <Type>移动</Type>      
  </Phone>  
  <Phone>  
    <Code>024-32222223-1</Code>  
    <Type>固定</Type>  
    <Make>公司</Make>  
  </Phone>  
  <Phone>  
    <Code>024-3788883</Code>  
    <Type>小灵通</Type>  
    <Make>公司</Make>  
  </Phone>  
</Phones>'  
    )  
)  
/

**4.5 本步骤总结**

更改还是比较方便的；数据库更改在所难免可以快速的更改才是王道；不过例子比较片面，大待大规模测试

**五、用 C# 写一个网页，显示这些数据；**

**5.1 显示效果**

显示效果如下（砢碜点啊）

|  |  |  |
| --- | --- | --- |
| **NPK** | **用户名** | **电话信息** |
| 1 | 用户1 | 【13940588000;移动;】、【024-22222222-1;固定;】、【12345678;未知;】 |
| 2 | 用户2 | 【13920588002;移动;】、【024-22222222-1;固定;】、【2788882;未知;】 |
| 3 | 用户3 | 【024-32222223-1;固定;公司】、【024-3788883;小灵通;公司】、【13930588003;移动;】 |
| 4 | 用户4 | 【13940588004;移动;个人】、【024-42222224-1;固定;公司】、【024-4788884;小灵通;公司】、【024-4788844;固定;个人】 |
| 5 | 用户5 | 【13950588005;移动;个人】、【024-52222225-1;固定;公司】、【024-5788885;小灵通;公司】、【024-5788845;固定;个人】 |

**5.2 代码**

//用户信息（XML\_USER\_INFO 是一样的）

public class UserInfo  
    {  
        int \_NPK;  
  
        public int NPK  
        {  
            get { return \_NPK; }  
            set { \_NPK = value; }  
        }  
        string \_userName;  
        public string UserName  
        {  
            get { return \_userName; }  
            set { \_userName = value; }  
        }  
          
        Phones \_phones;  
        public Phones Phones  
        {  
            get { return \_phones; }  
            set { \_phones = value; }  
        }  
  
    }

//数据操作代码 （很简单就是，把数据全差出来）

public class UserInfoDAL  
{  
    public static List<UserInfo> GetUserInfos()  
    {  
        //我本身是想用 ODP.net 不过为了让大家看懂就先用 ms 的 OracleClient  
        //其实  ODP.net 操作 oracle 更方便点;直接支持 xmlType 的不用转换  
          
        //GETSTRINGVAL 是 Oracle xmlType 类的一个函数，以string 形式返回xml 对象数据  
        const string SQL = "SELECT t.NPK,t.USER\_NAME, t.PHONES.GETSTRINGVAL() as  PHONES FROM XML\_USER\_INFO t order by 1";  
  
        List<UserInfo> models = new List<UserInfo>();  
        string connstring = System.Configuration.ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;  
  
          
        using (OracleConnection conn = new OracleConnection(connstring))  
        {  
            conn.Open();  
            using (OracleCommand cmd = new OracleCommand(SQL, conn))  
            {  
                  
                using (OracleDataReader reader = cmd.ExecuteReader())  
                {  
                    while (reader.Read())  
                    {  
                        DataTable dt = reader.GetSchemaTable();  
                          
                        int npk = Convert.ToInt32(reader["NPK"]);   
                          
                        string userName = reader["USER\_NAME"].ToString();  
                          
                        //读取 XML 类型  
                        string phonesXml = reader["PHONES"].ToString();  
                          
                        //这里是我自己扩展的全局的 string 对象的方法,就是把 string 通过XML反序列化为对象实例  
                        Phones ps = phonesXml.XMLToObject<Phones>();      
                          
                        models.Add(new UserInfo() { NPK = npk, UserName=userName, Phones = ps });  
                    }  
                }  
            }  
        }  
  
        return models;  
  
    }  
}

//网页的代码

<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="ObjectDataSource1">  
    <Columns>  
        <asp:BoundField DataField="NPK" HeaderText="NPK" SortExpression="NPK" />  
        <asp:BoundField DataField="UserName" HeaderText="用户名" SortExpression="UserName" />  
        <asp:BoundField DataField="Phones" HeaderText="电话信息" SortExpression="Phones" />  
    </Columns>  
</asp:GridView>  
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"   
    SelectMethod="GetUserInfos" TypeName="WebApplication1.UserInfoDAL">  
</asp:ObjectDataSource>

-- 可能有人会问；你的网页【**电话信息**】那列怎么显示成那个样子的？

-- 其实很简单；重写 Phones 类的 ToString() 想怎么显示都可以的，我这里就不贴代码了

-- 如果有人需要说一声，因为不属于本片博文的技术讨论范围就不贴上了；

**5.3 本步骤总结**

读取操作也是很方便的，比如在做分页的时候，就比连接2个表（用户表、电话表）要方便的多

而且返回数据，可以直接转换为对象更加方便了；不过数据量大的子表就不太适合了这种方式

**6、总结**

目前看来

XMLType 比起平面表更加类似 class 类

因为.net 可以直接通过类生成Schema 在生成 XMLType 数据库表，能节省一些数据表的设计时间,改完对象直接更改数据库就可以了

遗憾：C# 视乎生成不了Schema 的很多约束属性如限制字段长度的约束

还有集合内的类型，加不了外键比如本来我要是把 电话类型放到一个平面表里

就加不上外键了；

处理那些树形的数据比较适合；

如一个主记录、关联n个子记录的情况

查询插入都是比较方便的，直接可以序列化或反序列化为对象或xml 能节省些代码；

遗憾：XML 序列化可能慢点，不过可以使操作更明晰，如果直接操作 xml 的话太头痛了代码也乱

终于写完了，贴了这么多代码，园子里用Oracle 的不多，可能有人看不懂，不过问我就好了

我可以解答你，大家共同学习进步吧！

不过本人公司基本都是玩 oracle 没有时间去玩 sqlserver 抱歉了先

如果有时间在做个 sqlserver 的例子吧！

或者出一个 sqlserver 和 oracle 的 xml 类型对比性测试，如果有人愿意和我做这次试验，我可以提供 oracle 的测试机和代码

我对sqlserver 不是很熟悉的，已经n年没用了，如果我自己出对比测试，怕糟蹋了 sqlserver

我家的计算机是 AMD6000 +、4G内存 这种测试应该是没啥问题的，跑些不是超级大的应用还是可以的；

很晚了，错字明天在挑吧 呵呵；