基于 Remoting 技术的三层架构设计

FangRonghua

2009.02.22

目　录

1 名词及简介

1.1 三层架构

三层架构，是在客户/服务之间加入了一个"中间层"，也叫组件层，本设计中为应用服务程序。

引入中间层，提供了易于访问、易于管理的方法，可以将多种应用服务封装部署于应用服务器，增强了应用程序可用性、安全性、封装复用性、可扩展性和可移置性，从而实现了高效、安全、稳定的企业级系统应用。

1.2 分层描述三层架构

本设计中，将应用程序的数据访问、合法性校验等工作放在中间层进行处理。客户端不直接与数据库进行交互。中间层提供 Remoting 服务接口，客户端通过 Remoting 技术与中间层建立连接，再经由中间层与数据库进行交互。数据通过中间层的中转无疑是降低了效率，但是它脱离于界面与数据库的完美封装，使得它的缺点不值得一提。

微软的 DNA 架构定义了三个层：表示层（presentation）、业务层（business）、和数据存储层（data access）。

也有人分七层：界面外观层、界面规则层、业务接口层、业务逻辑层、实体层、数据访问层、数据存储层。

本设计中，采用的名称为：表示层、业务层、持久层。

- 表示层：负责显示信息及从系统外部得到输入。
- 业务层：完成业务逻辑。业务层知道如何对用户输入进行处理，能够应用业务规则完成用户所需的业务，但它不知道数据如何读取和保存。
- 持久层：负责用户信息的持久化。持久层完全不知道业务，只专注于数据存储和读取。

1.3 NET Remoting 技术简介

NET Remoting 是.NET 平台上允许存在于不同应用程序域中的对象相互知晓对方并进行通讯的基础设施。调用对象被称为客户端，而被调用对象则被称为服务器或者服务器对象。它是.NET 平台上实现分布式对象系统的框架。

Remoting 的优点：

- 性能： 如果调优.Net Remoting 的性能，那么它的性能非常好，速度接近 DCOM。
- 可扩展：.Net Remoting 可供你选择传输通道类型(如 Http,Tcp)和格式类型（如 Binary,Soap)。
- 可配置：可以通过配置文件配置应用程序。
- CLR 和 CTS 的好处：由于.NET Remoting 是基于.NET 框架的，所以他拥有 Common Type System(CTS) 和 Common Language Runtime(CLR)所拥有的易于使用和功能强大的特点。
- 互用性(Interoperability)：.NET Remoting 支持开发标准（Http,SOAP,WSDL,XML).

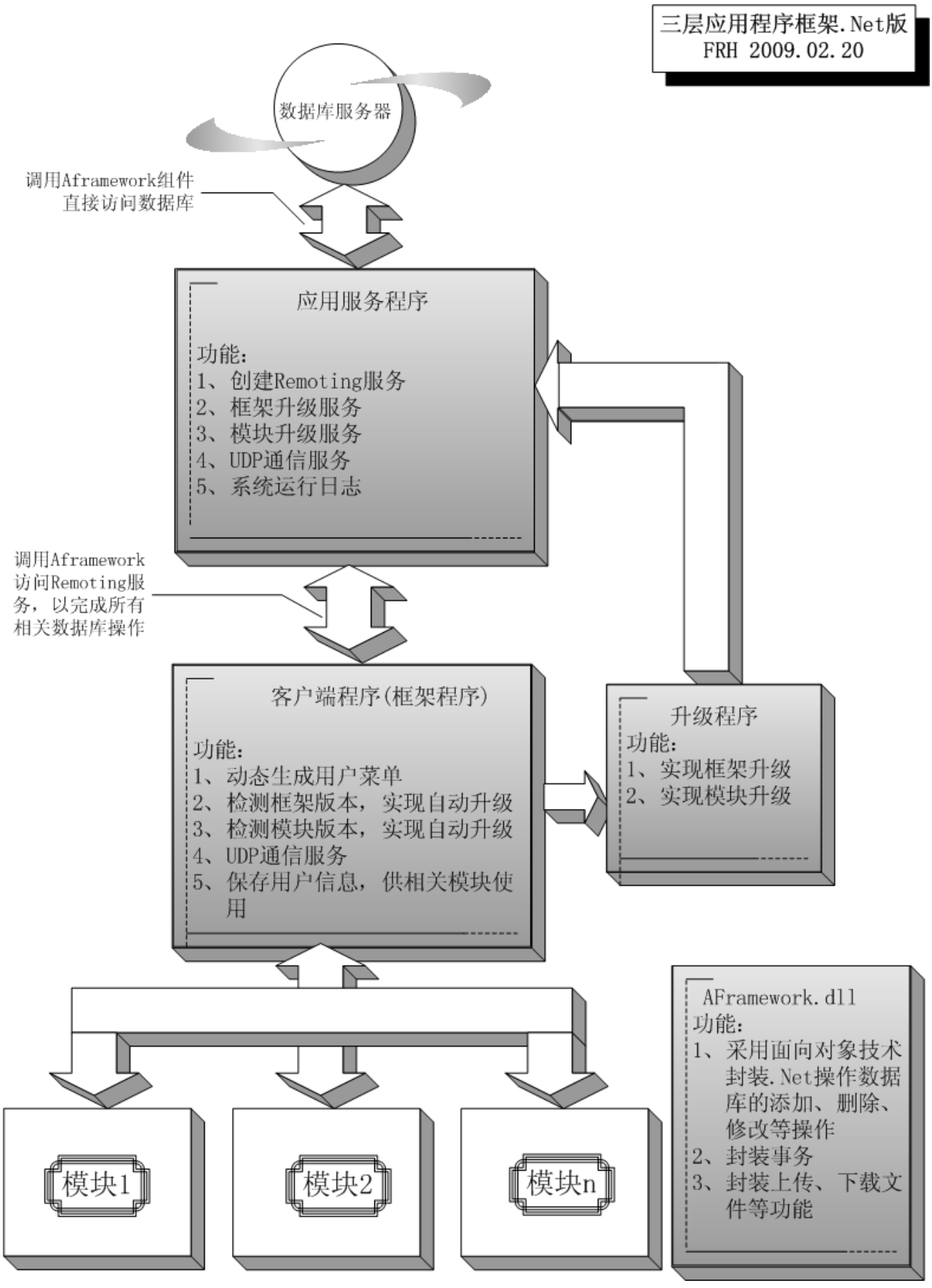
- 安全性
- 生命周期管理

1.4 Remoting 和 Webservice 有什么区别

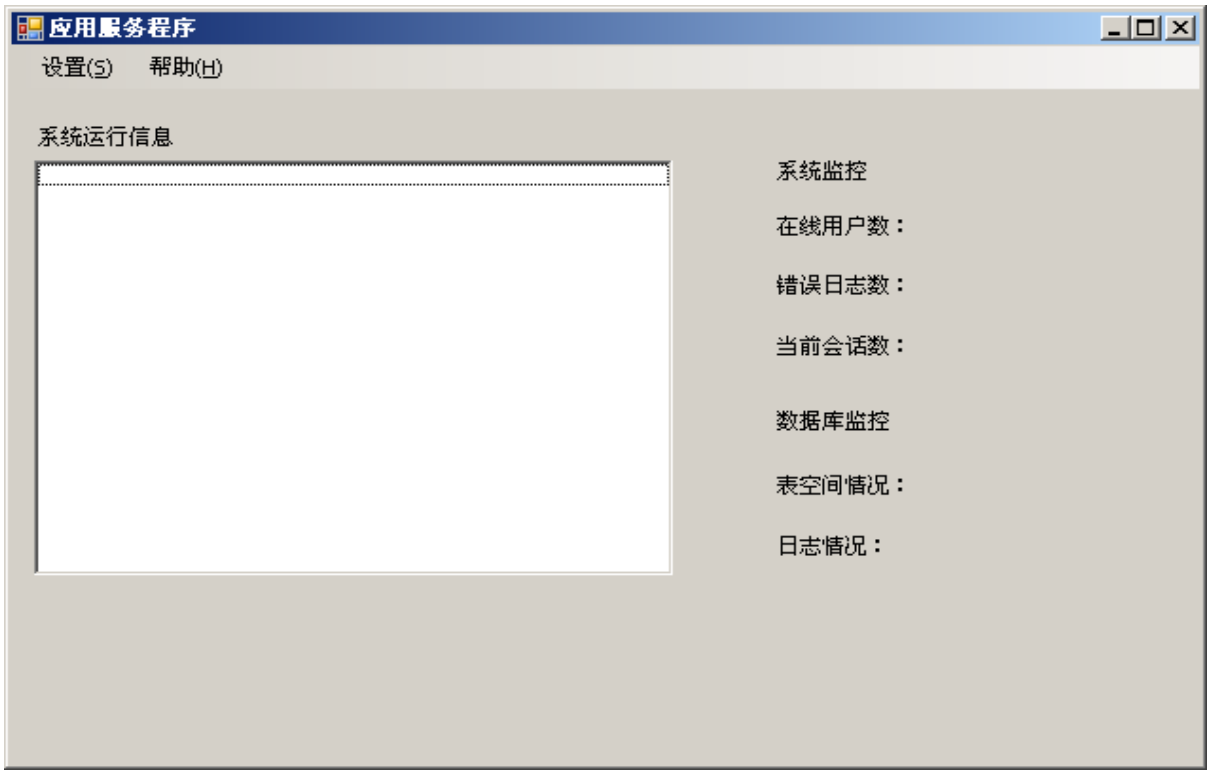
- Remoting 是 MarshByReference 的，可以传变量的引用，直接对服务器对象操作。速度快，适合 intranet。
webservice 是 MarshByValue 的，必须传对象的值。速度慢，可以过 FIREWALL,配置比较简单，适合 internet。
- 一般来说，Remoting 是和平台相关的，需要客户和服务器都是.NET，但可配置特性比较好，可以自定义协议。web service 可以做到跨平台通信，但必须采用 SOAP 协议。

2 设计图

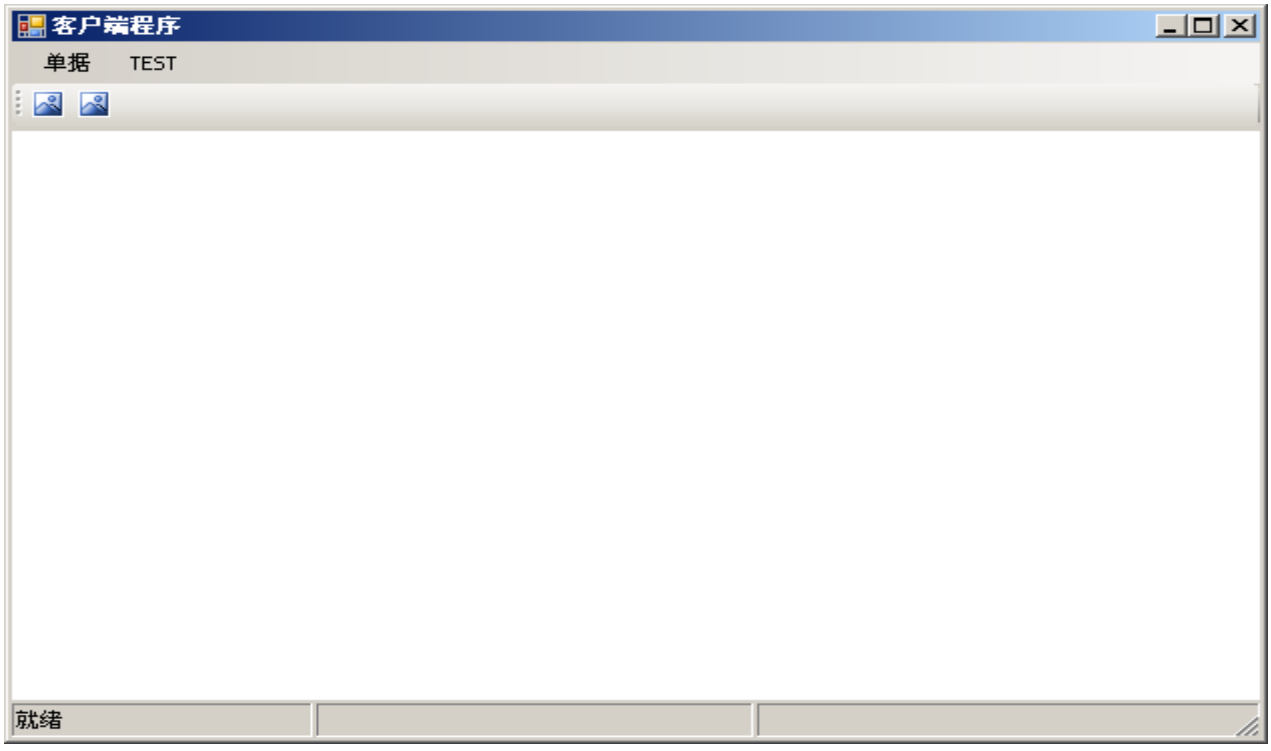
2.1 框架总体设计图


三层应用程序框架.Net版
FRH 2009.02.20
数据库服务器
调用Aframework组件
直接访问数据库
应用服务程序
功能:
1、创建Remoting服务
2、框架升级服务
3、模块升级服务
4、UDP通信服务
5、系统运行日志
调用Aframework
访问Remoting服
务，以完成所有
相关数据库操作
客户端程序(框架程序)
功能:
1、动态生成用户菜单
2、检测框架版本，实现自动升级
3、检测模块版本，实现自动升级
4、UDP通信服务
5、保存用户信息，供相关模块使
用
升级程序
功能:
1、实现框架升级
2、实现模块升级
模块1
模块2
模块n
AFramework.dll
功能:
1、采用面向对象技术
封装.Net操作数据
库的添加、删除、
修改等操作
2、封装事务
3、封装上传、下载文
件等功能

2.2 应用服务程序设计图



2.3 客户端程序设计图



2.3.1 客户端应用程序说明

- 每个模块放在单独的文件夹下，文件名、路径等信息需要配置在数据库中。
- 根据用户权限动态生成菜单，动态加载模块。
- 界面设计：

客户端程序窗口 1024*743 （显示任务栏）
菜单高度为 1016*24
工具栏高度 1016*25
状态栏高度 1016*22
模块高度应为 743 - 71 = 672
模块宽高为：1024*672 效果比较好
●模块窗口加载到客户端程序中间白色区域，加载时将去掉模块窗口的标题栏。

3 AFramework 程序集

文件名：AFramework.dll
AFramework 是持久层的一种实现，封装了应用程序常用的数据库操作。运用该组件时，主要运用两个类：PersistenceManager、QueryStatement。

3.1 PersistenceManager 类包含的方法：

3.1.1 新增记录

命名空间：AFramework
类名：PersistenceManager

方法	Int Insert(string sql)
名称	新增记录 (sql语句版)
说明	创建一个新的Oralce连接，执行该sql语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Insert(string sql,int sessionID)
名称	新增记录 (sql语句+事务版)
说明	使用指定的会话连接，执行该sql语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Insert(object model)
名称	新增记录 (DataModel版)
说明	创建一个新的Oralce连接，根据model自动构造插入语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Insert(object model, int sessionID)
名称	新增记录 (DataModel+事务版)
说明	使用指定的会话连接，根据model自动构造插入语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

3.1.2 更新记录

命名空间：AFramework
类名：PersistenceManager

方法	Int Update(string sql)
名称	更新记录 (sql语句版)
说明	创建一个新的Oralce连接，执行该sql语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Update(string sql,int sessionID)
名称	更新记录 (sql语句+事务版)

说明	使用指定的会话连接，执行该sql语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Update(object model, QueryStatement query)
名称	更新记录 (DataModel版)
说明	创建一个新的Oralce连接，根据model及query自动构造更新语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Update(object model, QueryStatement query, int sessionID)
名称	更新记录 (DataModel+事务版)
说明	使用指定的会话连接，根据model及query自动构造更新语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

3.1.3 删除记录

命名空间：AFramework
类名：PersistenceManager

方法	Int Delete(string sql)
名称	删除记录 (sql语句版)
说明	创建一个新的Oralce连接，执行该sql语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Delete(string sql, int sessionID)
名称	删除记录 (sql语句+事务版)
说明	使用指定的会话连接，执行该sql语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Delete(QueryStatement query)
名称	删除记录 (DataModel版)
说明	创建一个新的Oralce连接，根据query自动构造删除语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Int Delete(QueryStatement query, int sessionID)
名称	删除记录 (DataModel+事务版)

说明	使用指定的会话连接，根据query自动构造删除语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

3.1.4 单值查询

命名空间：AFramework
类名：PersistenceManager

方法	Object GetSingleValue(string sql)
名称	单值查询 (sql语句版)
说明	创建一个新的Oralce连接，执行该sql语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Object GetSingleValue(string sql, int sessionID)
名称	单值查询 (sql语句+事务版)
说明	使用指定的会话连接，执行该sql语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

方法	Object GetSingleValue(QueryStatement query)
名称	单值查询 (DataModel版)
说明	创建一个新的Oralce连接，根据query自动构造查询语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	Object GetSingleValue(QueryStatement query, int sessionID)
名称	单值查询 (DataModel+事务版)
说明	使用指定的会话连接，根据query自动构造查询语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

3.1.5 取 DataSet

命名空间：AFramework
类名：PersistenceManager

方法	DataSet GetDataSet(string sql)
名称	取DataSet (sql语句版)
说明	创建一个新的Oralce连接，执行该sql语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	DataSet GetDataSet(string sql,int sessionID)
名称	取DataSet (sql语句+事务版)
说明	使用指定的会话连接，执行该sql语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

方法	DataSet GetDataSet(QueryStatement query)
名称	取DataSet (DataModel版)
说明	创建一个新的Oralce连接，根据query自动构造查询语句，完成操作后自动提交，并关闭连接。遇到异常则抛出异常并自动回滚。

方法	DataSet GetDataSet(QueryStatement query, int sessionID)
名称	取DataSet (DataModel+事务版)
说明	使用指定的会话连接，根据query自动构造查询语句，完成操作后不提交，不关闭连接。遇到异常则抛出异常并自动回滚。

3.1.6 事务管理

命名空间：AFramework
类名：PersistenceManager

方法	Int Session_OpenSession()
名称	创建并打开会话
说明	连接Oracle，创建并打开会话。创建成功则返回SessioinID，在事务管理时用到该ID，以标识该会话。创建失败返回-1。

方法	bool Session_BeginTransaction(int sessionID)
名称	开启指定会话的事务
说明	开启成功，返回true；若失败，返回false。

方法	bool Session_CommitTransaction(int sessionID)

名称	提交指定会话的事务
说明	提交成功，返回true；若失败，返回false。

方法	bool Session_RollbackTransaction(int sessionID)
名称	回滚指定会话的事务
说明	回滚成功，返回true；若失败，返回false。 服务器端操作数据发生异常时自动回滚。客户端调用本方法时与服务器自动回滚不会冲突。

方法	bool Session_CloseSession(int sessionID)
名称	关闭会话
说明	关闭成功，返回true；若失败，返回false。

3.1.7 执行存储过程

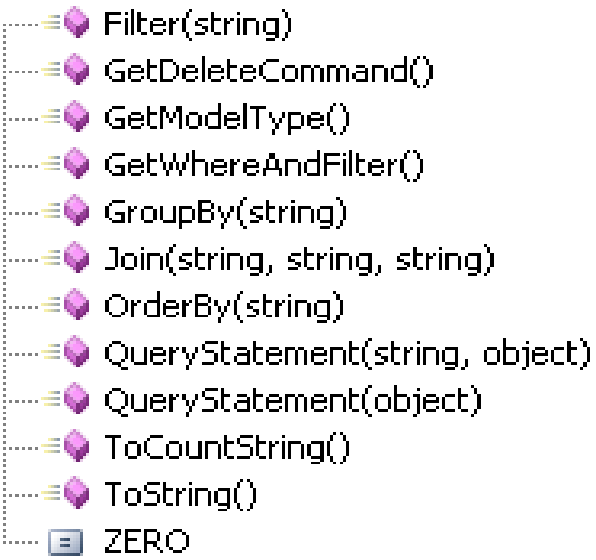
方法	String ExeStoredProcedure(string procedureName,OracleParameter[] pars)
名称	执行存储过程
说明	若需要返回值则需要在parameters中定义一个Direction为Output或InputOutput的参数，该参数必须为字符串类型。 若parameters长度为0，则认为无输入输出参数，只调用该存储过程。 执行过程出错将抛出异常。

3.1.8 Remoting 设置

方法	void SetRemotingServerIP(string ip)
名称	设置Remoting服务器IP地址
说明	如：127.0.0.1

方法	void SetRemotingServerPort(string port)
名称	设置Remoting服务器的服务端口
说明	如：6868 本设计采用http通道传输，可设置为80。

3.2 QueryStatement 类所包含的方法：



4 实体

实体即表对应的类，在本设计中也称为 DataModel。
每一张都生成一个类。在建立表之后，将运用工具生成全部表的实体。
开发过程中，表和实体的修改是同步的。

4.1 实体示例

```
using System;

namespace AFW_Test
{
    /// <summary>
    ///用户表
    /// </summary>
    [Serializable()]
    public class AFW_User
    {

        System.Decimal user_id;
```

```
string logon_name;

string user_name;

string password;

string sex;

public AFW_User()
{
}

/// <summary>
///
/// </summary>
public virtual System.Decimal USER_ID
{
    get
    {
        return this.user_id;
    }
    set
    {
        this.user_id=value;
    }
}

/// <summary>
///
/// </summary>
public virtual string LOGON_NAME
{
    get
    {
        return this.logon_name;
    }
    set
    {
        this.logon_name=value;
    }
}

/// <summary>
```

```
    ///
    /// </summary>
    public virtual string USER_NAME
    {
        get
        {
            return this.user_name;
        }
        set
        {
            this.user_name=value;
        }
    }

    /// <summary>
    ///
    /// </summary>
    public virtual string PASSWORD
    {
        get
        {
            return this.password;
        }
        set
        {
            this.password=value;
        }
    }

    /// <summary>
    ///
    /// </summary>
    public virtual string SEX
    {
        get
        {
            return this.sex;
        }
        set
        {
            this.sex=value;
        }
    }
}
```

```
}
```

5 业务层

业务层代码先由工具生成各表的文件，开发过程中在此基础上添加及修改方法。

5.1 业务层代码示例

```
using System;
using System.Data;
using AFW_Test;
using AFramework;

namespace AFW_Test.DataAccess
{
    /// <summary>
    ///用户表
    /// </summary>
    public class AFWUserManager
    {
        public AFWUserManager()
        {
        }

        /// <summary>
        /// 查询全部数据
        /// </summary>
        public DataSet SelectAllAFWUser()
        {
            AFW_User model = new AFW_User();
            QueryStatement query = new QueryStatement(model);
            return PersistenceManager.GetDataSet(query);
        }

        /// <summary>
        /// 取指定主键编码的 DataSet
        /// 若根据面向对象设计 则应该取指定主键编码的实体
        /// 但考虑实际运用中 DataSet 操作更简便 故取之
        /// </summary>
        public DataSet GetAFWUserByID(int user_id)
```

```
        {
            AFW_User model = new AFW_User();
            model.USER_ID = user_id;
            QueryStatement query = new QueryStatement(model);
            return PersistenceManager.GetDataSet(query);
        }

        /// <summary>
        /// 新增一行记录
        /// </summary>
        public int AddAFWUser(AFW_User afw_user)
        {
            return PersistenceManager.Insert(afw_user);
        }

        /// <summary>
        /// 更新一行记录
        /// </summary>
        public int UpdateAFWUser(AFW_User afw_user)
        {
            AFW_User model = new AFW_User();
            model.USER_ID = afw_user.USER_ID;
            QueryStatement query = new QueryStatement(model);
            return PersistenceManager.Update(afw_user,query);
        }

        /// <summary>
        /// 删除指定的一行记录
        /// </summary>
        public int DeleteAFWUserByID(int user_id)
        {
            AFW_User model = new AFW_User();
            model.USER_ID = user_id;
            QueryStatement query = new QueryStatement(model);
            return PersistenceManager.Delete(query);
        }
    }
}
```

6 测试代码

以下为程序控制事务的代码示例

```
        private void btnTest_Click(object sender, EventArgs e)
        {
            for (int i = 1; i < 101; i++)
            {
                Thread t = new Thread(new ThreadStart(Thread_Test));
                t.Start();

                Thread t2 = new Thread(new ThreadStart(Thread_Test));
                t2.Start();

                Thread t3 = new Thread(new ThreadStart(Thread_Test));
                t3.Start();

                Thread t4 = new Thread(new ThreadStart(Thread_Test));
                t4.Start();

                Thread t5 = new Thread(new ThreadStart(Thread_Test));
                t5.Start();
            }
        }

        private void Thread_Test()
        {
            int id = intID ++;

            try
            {
                int sessionID = PersistenceManager.Session_OpenSession();
                PersistenceManager.Session_BeginTransaction(sessionID);

                AFW_Test_One one = new AFW_Test_One();

                //ID1值要赋给表二、表三，所以不由触发器生成。
                one.ID1 = PersistenceManager.GetSequenceNextVal("AFW_TEST_ONE_seq");

                one.COL2 = "这是字段2";
                one.COL3 = "这是字段3";
                one.COL4 = "这是字段4";
                one.COL5 = id.ToString();

                PersistenceManager.Insert(one, sessionID);

                AFW_Test_Two two = new AFW_Test_Two();
                two.ID2 = PersistenceManager.GetSequenceNextVal("AFW_TEST_TWO_seq");
                two.ID1 = one.ID1;
                two.COL3 = "这是字段3";
                two.COL4 = "这是字段4";
                two.COL5 = id.ToString();

                PersistenceManager.Insert(two, sessionID);
```

```
                AFW_Test_Three three = new AFW_Test_Three();
                //three.ID3 = id;       //由触发器生成
                three.ID1 = one.ID1;
                three.ID2 = two.ID2;
                three.COL4 = "这是字段4";
                three.COL5 = id.ToString();

                PersistenceManager.Insert(three, sessionID);

                PersistenceManager.Session_CommitTransaction(sessionID);
                PersistenceManager.Session_CloseSession(sessionID);

                Console.WriteLine("id=" + id.ToString() + " 新增数据成功! ");
            }
            catch (Exception ex)
            {
                PersistenceManager.Session_RollbackTransaction(sessionID);
                Console.WriteLine("id="+ id.ToString() + " 新增数据失败 " +ex.Message)
            }
        }
    }
}
```

运行结果正常。

联系方式：QQ652182937

… END …