微软 SqlHelper 类中文注释和使用方法

整理：飞晏博客；网址：http://www.feiyan.info/416.html。

和微软发布的 SqlHelper.cs 不同，为了方便我把里面的 SqlHelperParameterCatch.cs 单独列出来了。此外还有 OledbHelper.cs、OdbcHelper.cs 和 XMLHelper.cs 可供使用。

SQLHelper.cs 文件源代码和注释：

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Data;
using System.Data.SqlClient;
using System.Xml;
using System.Collections;
using System.Configuration;

/// <summary>
///SqlHelper 的摘要说明
/// </summary>

public sealed class SqlHelper
{
    public SqlHelper()
    {
        //
        //TODO: 在此处添加构造函数逻辑
        //
    }

    public static readonly string connectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["testConnectionString"].ConnectionString;

    #region 私有构造函数和方法

    /// <summary>
    /// 将SqlParameters参数数组（参数值）分配给SqlCommand命令
    /// 这个方法将给任何一个参数分配DBNull.Value
    /// 该操作将阻止默认值的使用（大爷的，没看懂）
    /// </summary>
    /// <param name="command">要分配SqlParameters参数的SqlCommand命令</param>
    /// <param name="commandParameters">SqlParameters参数数组</param>
    private static void AttachParameters(SqlCommand command, SqlParameter[] commandParameters)
    {
```

```
        if (command == null) throw new ArgumentNullException("command");
        if (commandParameters != null)
        {
            foreach (SqlParameter p in commandParameters)
            {
                if (p != null)
                {
                    // Check for derived output value with no value assigned
                    if ((p.Direction == ParameterDirection.InputOutput ||
                        p.Direction == ParameterDirection.Input) &&
                        (p.Value == null))
                    {
                        p.Value = DBNull.Value;
                    }
                    command.Parameters.Add(p);
                }
            }
        }
    }

    /// <summary>
    /// 将DataRow类型的列值分配到SqlParameter参数数组
    /// </summary>
    /// <param name="commandParameters">要分配的SqlParameter参数数组</param>
    /// <param name="dataRow">将要分配给存储过程参数的DataRow</param>
    private static void AssignParameterValues(SqlParameter[] commandParameters, DataRow dataRow)
    {
        if ((commandParameters == null) || (dataRow == null))
        {
            // 没有接收到数据就返回
            return;
        }

        int i = 0;
        // 设置参数值
        foreach (SqlParameter commandParameter in commandParameters)
        {
            // 检查参数名称，如果不存在，抛出一个异常
            if (commandParameter.ParameterName == null ||
                commandParameter.ParameterName.Length <= 1)
                throw new Exception(
                    string.Format(
                        "请提供参数{0},一个有效的名称{1}.",
                        i, commandParameter.ParameterName));
```

```
                // 从DataRow的表中获取为参数数组中数组名称的列的索引.
                // 如果存在和参数名称相同的列,则将列值赋给当前名称的参数.
                if (dataRow.Table.Columns.IndexOf(commandParameter.ParameterName.Substring(1)) !=
-1)
                    commandParameter.Value =
dataRow[commandParameter.ParameterName.Substring(1)];
                i++;
            }
        }

        /// <summary>
        /// 将一个对象数组分配给SqlParameter参数数组.（重载方法）
        /// </summary>
        /// <param name="commandParameters">要分配的SqlParameter参数数组</param>
        /// <param name="parameterValues">将要分配给存储过程参数的对象数组</param>
        private static void AssignParameterValues(SqlParameter[] commandParameters, object[]
parameterValues)
        {
            if ((commandParameters == null) || (parameterValues == null))
            {
                return;
            }

            // 确保对象数组个数与参数个数匹配,如果不匹配,抛出一个异常
            if (commandParameters.Length != parameterValues.Length)
            {
                throw new ArgumentException("参数值个数与参数不匹配.");
            }

            // 给参数赋值
            for (int i = 0, j = commandParameters.Length; i < j; i++)
            {
                if (parameterValues[i] is IDbDataParameter)
                {
                    IDbDataParameter paramInstance = (IDbDataParameter)parameterValues[i];
                    if (paramInstance.Value == null)
                    {
                        commandParameters[i].Value = DBNull.Value;
                    }
                    else
                    {
                        commandParameters[i].Value = paramInstance.Value;
                    }
                }
```

```
                else if (parameterValues[i] == null)
                {
                    commandParameters[i].Value = DBNull.Value;
                }
                else
                {
                    commandParameters[i].Value = parameterValues[i];
                }
            }
        }

        /// <summary>
        /// 预处理用户提供的命令,数据库连接/事务/命令类型/参数
        /// </summary>
        /// <param name="command">要处理的SqlCommand</param>
        /// <param name="connection">有效的数据库连接</param>
        /// <param name="transaction">一个有效的事务或者是null值</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本, 其它.)</param>
        /// <param name="commandText">存储过程名或都T-SQL命令文本</param>
        /// <param name="commandParameters">和命令相关联的SqlParameter参数数组,如果没有参数为
'null'</param>
        /// <param name="mustCloseConnection"><c>true</c> 如果连接是打开的,则为true,其它情况下为
false.</param>
        private static void PrepareCommand(SqlCommand command, SqlConnection connection,
SqlTransaction transaction, CommandType commandType, string commandText, SqlParameter[]
commandParameters, out bool mustCloseConnection)
        {
            if (command == null) throw new ArgumentNullException("command");
            if (commandText == null || commandText.Length == 0) throw new
ArgumentNullException("commandText");

            if (connection.State != ConnectionState.Open)
            {
                mustCloseConnection = true;
                connection.Open();
            }
            else
            {
                mustCloseConnection = false;
            }

            command.CommandTimeout = 1000 * 60 * 60; // 60分钟

            // 给命令分配一个数据库连接.
```

```
            command.Connection = connection;

            // 设置命令文本(存储过程名或SQL语句)
            command.CommandText = commandText;

            // 分配事务
            if (transaction != null)
            {
                if (transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
                command.Transaction = transaction;
            }

            // 设置命令类型.
            command.CommandType = commandType;

            // 分配命令参数
            if (commandParameters != null)
            {
                AttachParameters(command, commandParameters);
            }
            return;
        }

        #endregion 私有构造函数和方法结束


        #region ExecuteNonQuery命令

        /// <summary>
        /// 执行指定连接字符串,类型的SqlCommand.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int result = ExecuteNonQuery(connString, CommandType.StoredProcedure, "PublishOrders");
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本, 其它.)</param>
        /// <param name="commandText">存储过程名称或SQL语句</param>
        /// <returns>返回命令影响的行数</returns>
        public static int ExecuteNonQuery(string connectionString, CommandType commandType, string commandText)
        {
            // ExecuteNonQuery #1
```

```
            return ExecuteNonQuery(connectionString, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定连接字符串,类型的SqlCommand.如果没有提供参数,不返回结果
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int result = ExecuteNonQuery(connString, CommandType.StoredProcedure, "PublishOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本, 其它.)</param>
        /// <param name="commandText">存储过程名称或SQL语句</param>
        /// <param name="commandParameters">SqlParameter参数数组</param>
        /// <returns>返回命令影响的行数</returns>
        public static int ExecuteNonQuery(string connectionString, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteNonQuery #2
            if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                return ExecuteNonQuery(connection, commandType, commandText, commandParameters);
            }
        }

        /// <summary>
        /// 执行指定连接字符串的存储过程,将对象数组的值赋给存储过程参数.
        /// 此方法需要在参数缓存方法中探索参数并生成参数.
        /// </summary>
        /// <remarks>
        /// 这个方法没有提供访问输出参数和返回值.
        /// 示例:
        /// int result = ExecuteNonQuery(connString, "PublishOrders", 24, 36);
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="parameterValues">分配到存储过程输入参数的对象数组</param>
        /// <returns>返回受影响的行数</returns>
        public static int ExecuteNonQuery(string connectionString, string spName, params object[]
```

```
parameterValues)
        {
            // ExecuteNonQuery #3
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果存在参数值，我们要搞清楚他们去哪里
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从探索存储过程参数(加载到缓存)并分配给存储过程参数数组.
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);

                return ExecuteNonQuery(connectionString, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                // 没有参数情况下
                return ExecuteNonQuery(connectionString, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int result = ExecuteNonQuery(conn, CommandType.StoredProcedure, "PublishOrders");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它.)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQuery(SqlConnection connection, CommandType commandType, string
commandText)
        {
            // ExecuteNonQuery #4
            return ExecuteNonQuery(connection, commandType, commandText, (SqlParameter[])null);
        }
```

```
        /// <summary>
        ///  执行指定数据库连接对象的命令
        /// </summary>
        /// <remarks>
        ///  示例:
        /// int result = ExecuteNonQuery(conn, CommandType.StoredProcedure, "PublishOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它.)</param>
        /// <param name="commandText">T存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamter参数数组</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQuery(SqlConnection connection, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteNonQuery #5
            if (connection == null) throw new ArgumentNullException("connection");

            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, connection, (SqlTransaction)null, commandType, commandText, commandParameters, out mustCloseConnection);

            int retval = cmd.ExecuteNonQuery();

            cmd.Parameters.Clear();
            if (mustCloseConnection)
                connection.Close();
            return retval;
        }

        /// <summary>
        ///  执行指定数据库连接对象的命令,将对象数组的值赋给存储过程参数.
        /// </summary>
        /// <remarks>
        ///  此方法不提供访问存储过程输出参数和返回值
        ///  示例:
        /// int result = ExecuteNonQuery(conn, "PublishOrders", 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
```

```
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQuery(SqlConnection connection, string spName, params object[]
parameterValues)
        {
            // ExecuteNonQuery #6
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);

                // 给存储过程分配参数值
                AssignParameterValues(commandParameters, parameterValues);

                return ExecuteNonQuery(connection, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return ExecuteNonQuery(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行带事务的SqlCommand.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int result = ExecuteNonQuery(trans, CommandType.StoredProcedure, "PublishOrders");
        /// </remarks>
        /// <param name="transaction">一个有效的事物</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它.)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQuery(SqlTransaction transaction, CommandType commandType, string
commandText)
        {
            // ExecuteNonQuery #7
            return ExecuteNonQuery(transaction, commandType, commandText, (SqlParameter[])null);
        }
```

```
        /// <summary>
        ///  执行带事务的SqlCommand(指定参数).
        /// </summary>
        /// <remarks>
        ///  示例:
        /// int result = ExecuteNonQuery(trans, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">一个有效的数据库事物</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它.)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamter参数数组</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQuery(SqlTransaction transaction, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteNonQuery #8
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");

            //预处理
            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, transaction.Connection, transaction, commandType, commandText, commandParameters, out mustCloseConnection);

            // 执行
            int retval = cmd.ExecuteNonQuery();

            // 清除参数集,以便再次使用
            cmd.Parameters.Clear();
            return retval;
        }

        /// <summary>
        ///  执行带事务的SqlCommand(指定参数值).
        /// </summary>
        /// <remarks>
        ///  此方法不提供访问存储过程输出参数和返回值
        ///  示例:
        /// int result = ExecuteNonQuery(conn, trans, "PublishOrders", 24, 36);
        /// </remarks>
```

```
        /// <param name="transaction">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回受影响的行数</returns>
        public static int ExecuteNonQuery(SqlTransaction transaction, string spName, params object[] parameterValues)
        {
            // ExecuteNonQuery #9
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中.
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);

                // 调用重载方法
                return ExecuteNonQuery(transaction, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                // 没有参数值
                return ExecuteNonQuery(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteNonQuery命令


        #region ExecuteDataset方法

        /// <summary>
        /// 执行指定数据库连接字符串的命令,返回DataSet.
        /// </summary>
        /// <remarks>
```

```
        /// 示例:
        /// DataSet ds = ExecuteDataset(connString, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(string connectionString, CommandType commandType, string commandText)
        {
            // ExecuteDataset #1
            return ExecuteDataset(connectionString, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库连接字符串的命令,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// DataSet ds = ExecuteDataset(connString, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型(存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamters参数数组</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(string connectionString, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteDataset #2
            if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                return ExecuteDataset(connection, commandType, commandText, commandParameters);
            }
        }

        /// <summary>
        /// 执行指定数据库连接字符串的命令,直接提供参数值,返回DataSet.
        /// </summary>
```

```
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值.
        /// 示例:
        /// DataSet ds = ExecuteDataset(connString, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(string connectionString, string spName, params object[]
parameterValues)
        {
            // ExecuteDataset #3
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中检索存储过程参数
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 给存储过程参数分配值
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteDataset(connectionString, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return ExecuteDataset(connectionString, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// DataSet ds = ExecuteDataset(conn, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
```

```
    /// <returns>返回一个包含结果集的DataSet</returns>
    public static DataSet ExecuteDataset(SqlConnection connection, CommandType commandType, string commandText)
    {
        // ExecuteDataset #4
        return ExecuteDataset(connection, commandType, commandText, (SqlParameter[])null);
    }

    /// <summary>
    /// 执行指定数据库连接对象的命令,指定存储过程参数,返回DataSet.
    /// </summary>
    /// <remarks>
    /// 示例:
    /// DataSet ds = ExecuteDataset(conn, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
    /// </remarks>
    /// <param name="connection">一个有效的数据库连接对象</param>
    /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名或T-SQL语句</param>
    /// <param name="commandParameters">SqlParamter参数数组</param>
    /// <returns>返回一个包含结果集的DataSet</returns>
    public static DataSet ExecuteDataset(SqlConnection connection, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
    {
        // ExecuteDataset #5
        if (connection == null) throw new ArgumentNullException("connection");

        // 预处理
        SqlCommand cmd = new SqlCommand();
        bool mustCloseConnection = false;
        PrepareCommand(cmd, connection, (SqlTransaction)null, commandType, commandText, commandParameters, out mustCloseConnection);

        // 创建SqlDataAdapter和DataSet.
        using (SqlDataAdapter da = new SqlDataAdapter(cmd))
        {
            DataSet ds = new DataSet();
            // 填充DataSet.
            da.Fill(ds);
            cmd.Parameters.Clear();
            if (mustCloseConnection)
                connection.Close();
            return ds;
        }
```

```
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,指定参数值,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输入参数和返回值.
        /// 示例.:
        ///  DataSet ds = ExecuteDataset(conn, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(SqlConnection connection, string spName, params object[]
parameterValues)
        {
            // ExecuteDataset #6
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 比缓存中加载存储过程参数
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);
                // 给存储过程参数分配值
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteDataset(connection, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return ExecuteDataset(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定事务的命令,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// DataSet ds = ExecuteDataset(trans, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="transaction">事务</param>
```

```
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(SqlTransaction transaction, CommandType commandType, string commandText)
        {
            // ExecuteDataset #7
            return ExecuteDataset(transaction, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定事务的命令,指定参数,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// DataSet ds = ExecuteDataset(trans, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamter参数数组</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(SqlTransaction transaction, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteDataset #8
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            // 预处理
            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, transaction.Connection, transaction, commandType, commandText, commandParameters, out mustCloseConnection);

            // 创建 DataAdapter & DataSet
            using (SqlDataAdapter da = new SqlDataAdapter(cmd))
            {
                DataSet ds = new DataSet();
                da.Fill(ds);
                cmd.Parameters.Clear();
                return ds;
            }
```

```
        }

        /// <summary>
        /// 执行指定事务的命令,指定参数值,返回DataSet.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输入参数和返回值.
        /// 示例.:
        ///  DataSet ds = ExecuteDataset(trans, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="transaction">事务</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回一个包含结果集的DataSet</returns>
        public static DataSet ExecuteDataset(SqlTransaction transaction, string spName, params object[] parameterValues)
        {
            // ExecuteDataset #9
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);
                // 给存储过程参数分配值
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteDataset(transaction, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                return ExecuteDataset(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteDataset方法


        #region ExecuteReader 数据阅读器
        /// <summary>
```

```
        /// 枚举,标识数据库连接是由SqlHelper提供还是由调用者提供
        /// </summary>
        private enum SqlConnectionOwnership
        {
            /// <summary>由SqlHelper提供连接</summary>
            Internal,
            /// <summary>由调用者提供连接</summary>
            External
        }

        /// <summary>
        /// 执行指定数据库连接对象的数据阅读器.
        /// </summary>
        /// <remarks>
        /// 如果是SqlHelper打开连接,当连接关闭DataReader也将关闭.
        /// 如果是调用都打开连接,DataReader由调用都管理.
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="transaction">一个有效的事务,或者为 'null'</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <param name="commandParameters">SqlParameters参数数组,如果没有参数则为'null'</param>
        /// <param name="connectionOwnership">标识数据库连接对象是由调用者提供还是由SqlHelper提供</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        private static SqlDataReader ExecuteReader(SqlConnection connection, SqlTransaction transaction, CommandType commandType, string commandText, SqlParameter[] commandParameters, SqlConnectionOwnership connectionOwnership)
        {
            // ExecuteReader #1
            if (connection == null) throw new ArgumentNullException("connection");
            bool mustCloseConnection = false;
            // 创建命令
            SqlCommand cmd = new SqlCommand();
            try
            {
                PrepareCommand(cmd, connection, transaction, commandType, commandText, commandParameters, out mustCloseConnection);
                // 创建数据阅读器
                SqlDataReader dataReader;
                if (connectionOwnership == SqlConnectionOwnership.External)
                {
                    dataReader = cmd.ExecuteReader();
                }
```

```
            else
            {
                dataReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
            }

            //  清除参数,以便再次使用..
            // HACK: There is a problem here, the output parameter values are fletched
            // when the reader is closed, so if the parameters are detached from the command
            // then the SqlReader can磕  set its values.
            // When this happen, the parameters can磕  be used again in other command.
            bool canClear = true;
            foreach (SqlParameter commandParameter in cmd.Parameters)
            {
                if (commandParameter.Direction != ParameterDirection.Input)
                    canClear = false;
            }

            if (canClear)
            {
                cmd.Parameters.Clear();
            }
            return dataReader;
        }

        catch
        {
            if (mustCloseConnection)
                connection.Close();
            throw;
        }
    }

    /// <summary>
    ///  执行指定数据库连接字符串的数据阅读器.
    /// </summary>
    /// <remarks>
    ///  示例:
    /// SqlDataReader dr = ExecuteReader(connString, CommandType.StoredProcedure, "GetOrders");
    /// </remarks>
    /// <param name="connectionString">一个有效的数据库连接字符串</param>
    /// <param name="commandType">命令类型  (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名或T-SQL语句</param>
    /// <returns>返回包含结果集的SqlDataReader</returns>
    public static SqlDataReader ExecuteReader(string connectionString, CommandType commandType,
```

```
string commandText)
        {
            // ExecuteReader #2
            return ExecuteReader(connectionString, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库连接字符串的数据阅读器,指定参数.
        /// </summary>
        /// <remarks>
        /// 示例:
        ///   SqlDataReader dr = ExecuteReader(connString, CommandType.StoredProcedure, "GetOrders",
new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamter参数数组(new SqlParameter("@prodid",
24))</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(string connectionString, CommandType commandType,
string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteReader #3
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            SqlConnection connection = null;
            try
            {
                connection = new SqlConnection(connectionString);
                connection.Open();
                return ExecuteReader(connection, null, commandType, commandText, commandParameters,
SqlConnectionOwnership.Internal);
            }
            catch
            {
                if (connection != null) connection.Close();
                throw;
            }
        }

        /// <summary>
        /// 执行指定数据库连接字符串的数据阅读器,指定参数值.
        /// </summary>
```

```
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// SqlDataReader dr = ExecuteReader(connString, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(string connectionString, string spName, params object[] parameterValues)
        {
            // ExecuteReader #4
            if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteReader(connectionString, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                return ExecuteReader(connectionString, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的数据阅读器.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// SqlDataReader dr = ExecuteReader(conn, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(SqlConnection connection, CommandType commandType, string commandText)
        {
```

```
            // ExecuteReader #5
            return ExecuteReader(connection, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// [调用者方式]执行指定数据库连接对象的数据阅读器,指定参数.
        /// </summary>
        /// <remarks>
        ///  示例:
        /// SqlDataReader dr = ExecuteReader(conn, CommandType.StoredProcedure, "GetOrders", new
SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名或T-SQL语句</param>
        /// <param name="commandParameters">SqlParamter参数数组</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(SqlConnection connection, CommandType commandType,
string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteReader #6
            return ExecuteReader(connection, (SqlTransaction)null, commandType, commandText,
commandParameters, SqlConnectionOwnership.External);
        }

        /// <summary>
        /// [调用者方式]执行指定数据库连接对象的数据阅读器,指定参数值.
        /// </summary>
        /// <remarks>
        ///  此方法不提供访问存储过程输出参数和返回值参数.
        ///  示例:
        /// SqlDataReader dr = ExecuteReader(conn, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">T存储过程名</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(SqlConnection connection, string spName, params object[]
parameterValues)
        {
            // ExecuteReader #7
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            if ((parameterValues != null) && (parameterValues.Length > 0))
```

```
            {
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteReader(connection, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return ExecuteReader(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// [调用者方式]执行指定数据库事务的数据阅读器,指定参数值.
        /// </summary>
        /// <remarks>
        ///  示例:
        /// SqlDataReader dr = ExecuteReader(trans, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型  (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(SqlTransaction transaction, CommandType commandType,
string commandText)
        {
            // ExecuteReader #8
            return ExecuteReader(transaction, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// [调用者方式]执行指定数据库事务的数据阅读器,指定参数.
        /// </summary>
        /// <remarks>
        ///  示例:
        ///    SqlDataReader dr = ExecuteReader(trans, CommandType.StoredProcedure, "GetOrders", new
SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型  (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
```

```
        public static SqlDataReader ExecuteReader(SqlTransaction transaction, CommandType commandType,
string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteReader #9
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The
transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            return ExecuteReader(transaction.Connection, transaction, commandType, commandText,
commandParameters, SqlConnectionOwnership.External);
        }

        /// <summary>
        /// [调用者方式]执行指定数据库事务的数据阅读器,指定参数值.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// SqlDataReader dr = ExecuteReader(trans, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReader(SqlTransaction transaction, string spName, params object[]
parameterValues)
        {
            // ExecuteReader #10
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The
transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);
                AssignParameterValues(commandParameters, parameterValues);
                return ExecuteReader(transaction, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                // 没有参数值
                return ExecuteReader(transaction, CommandType.StoredProcedure, spName);
```

```
        }
    }

    #endregion ExecuteReader  数据阅读器


    #region ExecuteScalar  返回结果集中的第一行第一列

    /// <summary>
    ///  执行指定数据库连接字符串的命令,返回结果集中的第一行第一列.
    /// </summary>
    /// <remarks>
    ///  示例:
    /// int orderCount = (int)ExecuteScalar(connString, CommandType.StoredProcedure,
"GetOrderCount");
    /// </remarks>
    /// <param name="connectionString">一个有效的数据库连接字符串</param>
    /// <param name="commandType">命令类型  (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名称或T-SQL语句</param>
    /// <returns>返回结果集中的第一行第一列</returns>
    public static object ExecuteScalar(string connectionString, CommandType commandType, string
commandText)
    {
        // ExecuteScalar #1
        //  执行参数为空的方法
        return ExecuteScalar(connectionString, commandType, commandText, (SqlParameter[])null);
    }

    /// <summary>
    ///  执行指定数据库连接字符串的命令,指定参数,返回结果集中的第一行第一列.
    /// </summary>
    /// <remarks>
    ///  示例:
    /// int orderCount = (int)ExecuteScalar(connString, CommandType.StoredProcedure, "GetOrderCount",
new SqlParameter("@prodid", 24));
    /// </remarks>
    /// <param name="connectionString">一个有效的数据库连接字符串</param>
    /// <param name="commandType">命令类型  (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名称或T-SQL语句</param>
    /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
    /// <returns>返回结果集中的第一行第一列</returns>
    public static object ExecuteScalar(string connectionString, CommandType commandType, string
commandText, params SqlParameter[] commandParameters)
    {
```

```
            // ExecuteScalar #2
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            // 创建并打开数据库连接对象,操作完成释放对象.
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                // 调用指定数据库连接字符串重载方法.
                return ExecuteScalar(connection, commandType, commandText, commandParameters);
            }
        }

        /// <summary>
        /// 执行指定数据库连接字符串的命令,指定参数值,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(connString, "GetOrderCount", 24, 36);
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(string connectionString, string spName, params object[]
parameterValues)
        {
            // ExecuteScalar #3
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);
                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);
                // 调用重载方法
                return ExecuteScalar(connectionString, CommandType.StoredProcedure, spName,
commandParameters);
            }
```

```
            else
            {
                // 没有参数值
                return ExecuteScalar(connectionString, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(conn, CommandType.StoredProcedure, "GetOrderCount");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlConnection connection, CommandType commandType, string commandText)
        {
            // ExecuteScalar #4
            // 执行参数为空的方法
            return ExecuteScalar(connection, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,指定参数,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(conn, CommandType.StoredProcedure, "GetOrderCount", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlConnection connection, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteScalar #5
            if (connection == null) throw new ArgumentNullException("connection");
```

```
            // 创建SqlCommand命令,并进行预处理
            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, connection, (SqlTransaction)null, commandType, commandText,
commandParameters, out mustCloseConnection);
            // 执行SqlCommand命令,并返回结果.
            object retval = cmd.ExecuteScalar();
            // 清除参数,以便再次使用.
            cmd.Parameters.Clear();
            if (mustCloseConnection)
                connection.Close();
            return retval;
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,指定参数值,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(conn, "GetOrderCount", 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlConnection connection, string spName, params object[]
parameterValues)
        {
            // ExecuteScalar #6
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);
                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);
                // 调用重载方法
                return ExecuteScalar(connection, CommandType.StoredProcedure, spName,
commandParameters);
```

```
            }
            else
            {
                // 没有参数值
                return ExecuteScalar(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库事务的命令,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(trans, CommandType.StoredProcedure, "GetOrderCount");
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlTransaction transaction, CommandType commandType, string commandText)
        {
            // ExecuteScalar #7
            // 执行参数为空的方法
            return ExecuteScalar(transaction, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库事务的命令,指定参数,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(trans, CommandType.StoredProcedure, "GetOrderCount", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlTransaction transaction, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteScalar #8
```

```
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");

            // 创建SqlCommand命令,并进行预处理
            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, transaction.Connection, transaction, commandType, commandText, commandParameters, out mustCloseConnection);

            // 执行SqlCommand命令,并返回结果.
            object retval = cmd.ExecuteScalar();

            // 清除参数,以便再次使用.
            cmd.Parameters.Clear();
            return retval;
        }

        /// <summary>
        /// 执行指定数据库事务的命令,指定参数值,返回结果集中的第一行第一列.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// int orderCount = (int)ExecuteScalar(trans, "GetOrderCount", 24, 36);
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalar(SqlTransaction transaction, string spName, params object[] parameterValues)
        {
            // ExecuteScalar #9
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);
```

```
                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);

                // 调用重载方法
                return ExecuteScalar(transaction, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                // 没有参数值
                return ExecuteScalar(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteScalar


        #region ExecuteXmlReader XML阅读器

        /// <summary>
        /// 执行指定数据库连接对象的SqlCommand命令,并产生一个XmlReader对象做为结果集返回.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// XmlReader r = ExecuteXmlReader(conn, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句 using "FOR XML AUTO"</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReader(SqlConnection connection, CommandType commandType,
string commandText)
        {
            // ExecuteXmlReader #1
            // 执行参数为空的方法
            return ExecuteXmlReader(connection, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库连接对象的SqlCommand命令,并产生一个XmlReader对象做为结果集返回,指
定参数.
        /// </summary>
        /// <remarks>
```

```
        /// 示例:
        /// XmlReader r = ExecuteXmlReader(conn, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句 using "FOR XML AUTO"</param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReader(SqlConnection connection, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteXmlReader #2
            if (connection == null) throw new ArgumentNullException("connection");
            bool mustCloseConnection = false;

            // 创建SqlCommand命令,并进行预处理
            SqlCommand cmd = new SqlCommand();
            try
            {
                PrepareCommand(cmd, connection, (SqlTransaction)null, commandType, commandText, commandParameters, out mustCloseConnection);

                // 执行命令
                XmlReader retval = cmd.ExecuteXmlReader();

                // 清除参数,以便再次使用.
                cmd.Parameters.Clear();
                return retval;
            }
            catch
            {
                if (mustCloseConnection)
                    connection.Close();
                throw;
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的SqlCommand命令,并产生一个XmlReader对象做为结果集返回,指定参数值.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
```

```
        /// 示例:
        /// XmlReader r = ExecuteXmlReader(conn, "GetOrders", 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称 using "FOR XML AUTO"</param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReader(SqlConnection connection, string spName, params object[] parameterValues)
        {
            // ExecuteXmlReader #3
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connection, spName);

                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);

                // 调用重载方法
                return ExecuteXmlReader(connection, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                // 没有参数值
                return ExecuteXmlReader(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定数据库事务的SqlCommand命令,并产生一个XmlReader对象做为结果集返回.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// XmlReader r = ExecuteXmlReader(trans, CommandType.StoredProcedure, "GetOrders");
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
```

```
        /// <param name="commandText">存储过程名称或T-SQL语句 using "FOR XML AUTO"</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReader(SqlTransaction transaction, CommandType commandType, string commandText)
        {
            // ExecuteXmlReader #4
            // 执行参数为空的方法
            return ExecuteXmlReader(transaction, commandType, commandText, (SqlParameter[])null);
        }

        /// <summary>
        /// 执行指定数据库事务的SqlCommand命令,并产生一个XmlReader对象做为结果集返回,指定参数.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// XmlReader r = ExecuteXmlReader(trans, CommandType.StoredProcedure, "GetOrders", new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句 using "FOR XML AUTO"</param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReader(SqlTransaction transaction, CommandType commandType, string commandText, params SqlParameter[] commandParameters)
        {
            // ExecuteXmlReader #5
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");

            // 创建SqlCommand命令,并进行预处理
            SqlCommand cmd = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(cmd, transaction.Connection, transaction, commandType, commandText, commandParameters, out mustCloseConnection);

            // 执行命令
            XmlReader retval = cmd.ExecuteXmlReader();

            // 清除参数,以便再次使用.
            cmd.Parameters.Clear();
            return retval;
```

```
    }

    /// <summary>
    /// 执行指定数据库事务的SqlCommand命令,并产生一个XmlReader对象做为结果集返回,指定参数值.
    /// </summary>
    /// <remarks>
    /// 此方法不提供访问存储过程输出参数和返回值参数.
    /// 示例:
    /// XmlReader r = ExecuteXmlReader(trans, "GetOrders", 24, 36);
    /// </remarks>
    /// <param name="transaction">一个有效的连接事务</param>
    /// <param name="spName">存储过程名称</param>
    /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
    /// <returns>返回一个包含结果集的DataSet.</returns>
    public static XmlReader ExecuteXmlReader(SqlTransaction transaction, string spName, params object[] parameterValues)
    {
        // ExecuteXmlReader #6
        if (transaction == null) throw new ArgumentNullException("transaction");
        if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

        // 如果有参数值
        if ((parameterValues != null) && (parameterValues.Length > 0))
        {
            // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

            // 给存储过程参数赋值
            AssignParameterValues(commandParameters, parameterValues);

            // 调用重载方法
            return ExecuteXmlReader(transaction, CommandType.StoredProcedure, spName, commandParameters);
        }
        else
        {
            // 没有参数值
            return ExecuteXmlReader(transaction, CommandType.StoredProcedure, spName);
        }
```

```
    }

    #endregion ExecuteXmlReader 阅读器结束


    #region FillDataset 填充数据集

    /// <summary>
    /// 执行指定数据库连接字符串的命令,映射数据表并填充数据集.
    /// </summary>
    /// <remarks>
    /// 示例:
    /// FillDataset(connString, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"});
    /// </remarks>
    /// <param name="connectionString">一个有效的数据库连接字符串</param>
    /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名称或T-SQL语句</param>
    /// <param name="dataSet">要填充结果集的DataSet实例</param>
    /// <param name="tableNames">表映射的数据表数组
    /// 用户定义的表名 (可有是实际的表名.)</param>
    public static void FillDataset(string connectionString, CommandType commandType, string
commandText, DataSet dataSet, string[] tableNames)
    {
        // FillDataset #1
        if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
        if (dataSet == null) throw new ArgumentNullException("dataSet");

        // 创建并打开数据库连接对象,操作完成释放对象.
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // 调用指定数据库连接字符串重载方法.
            FillDataset(connection, commandType, commandText, dataSet, tableNames);
        }
    }

    /// <summary>
    /// 执行指定数据库连接字符串的命令,映射数据表并填充数据集.指定命令参数.
    /// </summary>
    /// <remarks>
    /// 示例:
    /// FillDataset(connString, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"},
```

```
new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="dataSet">分配给命令的SqlParamter参数数组</param>
        /// <param name="tableNames">要填充结果集的DataSet实例</param>
        /// <param name="commandParameters">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        public static void FillDataset(string connectionString, CommandType commandType,
            string commandText, DataSet dataSet, string[] tableNames,
            params SqlParameter[] commandParameters)
        {
            // FillDataset #2
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (dataSet == null) throw new ArgumentNullException("dataSet");

            // 创建并打开数据库连接对象,操作完成释放对象.
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                // 调用指定数据库连接字符串重载方法.
                FillDataset(connection, commandType, commandText, dataSet, tableNames,
commandParameters);
            }
        }

        /// <summary>
        /// 执行指定数据库连接字符串的命令,映射数据表并填充数据集,指定存储过程参数值.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// FillDataset(connString, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"},
24);
        /// </remarks>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
```

```
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        public static void FillDataset(string connectionString, string spName, DataSet dataSet, string[]
tableNames, params object[] parameterValues)
        {
            // FillDataset #3
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (dataSet == null) throw new ArgumentNullException("dataSet");
            // 创建并打开数据库连接对象,操作完成释放对象.
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                // 调用指定数据库连接字符串重载方法.
                FillDataset(connection, spName, dataSet, tableNames, parameterValues);
            }
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,映射数据表并填充数据集.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// FillDataset(conn, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"});
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        public static void FillDataset(SqlConnection connection, CommandType commandType, string
commandText, DataSet dataSet, string[] tableNames)
        {
            // FillDataset #4
            FillDataset(connection, commandType, commandText, dataSet, tableNames, null);
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,映射数据表并填充数据集,指定参数.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// FillDataset(conn, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"}, new
```

```
SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        public static void FillDataset(SqlConnection connection, CommandType commandType, string
commandText, DataSet dataSet, string[] tableNames, params SqlParameter[] commandParameters)
        {
            // FillDataset #5
            FillDataset(connection, null, commandType, commandText, dataSet, tableNames,
commandParameters);
        }

        /// <summary>
        /// 执行指定数据库连接对象的命令,映射数据表并填充数据集,指定存储过程参数值.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// FillDataset(conn, "GetOrders", ds, new string[] {"orders"}, 24, 36);
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        public static void FillDataset(SqlConnection connection, string spName, DataSet dataSet, string[]
tableNames, params object[] parameterValues)
        {
            // FillDataset #6
            if (connection == null) throw new ArgumentNullException("connection");
            if (dataSet == null) throw new ArgumentNullException("dataSet");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
```

```
            // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connection, spName);

            // 给存储过程参数赋值
            AssignParameterValues(commandParameters, parameterValues);

            // 调用重载方法
            FillDataset(connection, CommandType.StoredProcedure, spName, dataSet, tableNames, commandParameters);
        }
        else
        {
            // 没有参数值
            FillDataset(connection, CommandType.StoredProcedure, spName, dataSet, tableNames);
        }
    }

    /// <summary>
    /// 执行指定数据库事务的命令,映射数据表并填充数据集.
    /// </summary>
    /// <remarks>
    /// 示例:
    /// FillDataset(trans, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"});
    /// </remarks>
    /// <param name="transaction">一个有效的连接事务</param>
    /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
    /// <param name="commandText">存储过程名称或T-SQL语句</param>
    /// <param name="dataSet">要填充结果集的DataSet实例</param>
    /// <param name="tableNames">表映射的数据表数组
    /// 用户定义的表名 (可有是实际的表名.)
    /// </param>
    public static void FillDataset(SqlTransaction transaction, CommandType commandType, string commandText, DataSet dataSet, string[] tableNames)
    {
        // FillDataset #7
        FillDataset(transaction, commandType, commandText, dataSet, tableNames, null);
    }

    /// <summary>
    /// 执行指定数据库事务的命令,映射数据表并填充数据集,指定参数.
    /// </summary>
    /// <remarks>
```

```
        /// 示例:
        /// FillDataset(trans, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"}, new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        public static void FillDataset(SqlTransaction transaction, CommandType commandType, string commandText, DataSet dataSet, string[] tableNames, params SqlParameter[] commandParameters)
        {
            // FillDataset #8
            FillDataset(transaction.Connection, transaction, commandType, commandText, dataSet, tableNames, commandParameters);
        }

        /// <summary>
        /// 执行指定数据库事务的命令,映射数据表并填充数据集,指定存储过程参数值.
        /// </summary>
        /// <remarks>
        /// 此方法不提供访问存储过程输出参数和返回值参数.
        /// 示例:
        /// FillDataset(trans, "GetOrders", ds, new string[]{"orders"}, 24, 36);
        /// </remarks>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        /// <param name="parameterValues">分配给存储过程输入参数的对象数组</param>
        public static void FillDataset(SqlTransaction transaction, string spName, DataSet dataSet, string[] tableNames, params object[] parameterValues)
        {
            // FillDataset #9
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (dataSet == null) throw new ArgumentNullException("dataSet");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
```

```
            // 如果有参数值
            if ((parameterValues != null) && (parameterValues.Length > 0))
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                // 给存储过程参数赋值
                AssignParameterValues(commandParameters, parameterValues);

                // 调用重载方法
                FillDataset(transaction, CommandType.StoredProcedure, spName, dataSet, tableNames, commandParameters);
            }
            else
            {
                // 没有参数值
                FillDataset(transaction, CommandType.StoredProcedure, spName, dataSet, tableNames);
            }
        }

        /// <summary>
        /// [私有方法][内部调用]执行指定数据库连接对象/事务的命令,映射数据表并填充数据集,DataSet/TableNames/SqlParameters.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// FillDataset(conn, trans, CommandType.StoredProcedure, "GetOrders", ds, new string[] {"orders"}, new SqlParameter("@prodid", 24));
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="transaction">一个有效的连接事务</param>
        /// <param name="commandType">命令类型 (存储过程,命令文本或其它)</param>
        /// <param name="commandText">存储过程名称或T-SQL语句</param>
        /// <param name="dataSet">要填充结果集的DataSet实例</param>
        /// <param name="tableNames">表映射的数据表数组
        /// 用户定义的表名 (可有是实际的表名.)
        /// </param>
        /// <param name="commandParameters">分配给命令的SqlParamter参数数组</param>
        private static void FillDataset(SqlConnection connection, SqlTransaction transaction, CommandType commandType,
            string commandText, DataSet dataSet, string[] tableNames, params SqlParameter[]
```

```
commandParameters)
        {
            // FillDataset #10
            if (connection == null) throw new ArgumentNullException("connection");
            if (dataSet == null) throw new ArgumentNullException("dataSet");
            // 创建SqlCommand命令,并进行预处理
            SqlCommand command = new SqlCommand();
            bool mustCloseConnection = false;
            PrepareCommand(command, connection, transaction, commandType, commandText,
commandParameters, out mustCloseConnection);

            // 执行命令
            using (SqlDataAdapter dataAdapter = new SqlDataAdapter(command))
            {
                // 追加表映射
                if (tableNames != null && tableNames.Length > 0)
                {
                    string tableName = "Table";
                    for (int index = 0; index < tableNames.Length; index++)
                    {
                        if (tableNames[index] == null || tableNames[index].Length == 0) throw new
ArgumentException("The tableNames parameter must contain a list of tables, a value was provided as null
or empty string.", "tableNames");
                        dataAdapter.TableMappings.Add(tableName, tableNames[index]);
                        tableName += (index + 1).ToString();
                    }
                }

                // 填充数据集使用默认表名称
                dataAdapter.Fill(dataSet);

                // 清除参数,以便再次使用.
                command.Parameters.Clear();
            }
            if (mustCloseConnection)
                connection.Close();
        }

        #endregion FillDataset 填充数据集


        #region UpdateDataset 更新数据集

        /// <summary>
```

```
        /// 执行数据集更新到数据库,指定inserted, updated, or deleted命令.
        /// </summary>
        /// <remarks>
        /// 示例:
        /// UpdateDataset(conn, insertCommand, deleteCommand, updateCommand, dataSet, "Order");
        /// </remarks>
        /// <param name="insertCommand">[追加记录]一个有效的T-SQL语句或存储过程</param>
        /// <param name="deleteCommand">[删除记录]一个有效的T-SQL语句或存储过程</param>
        /// <param name="updateCommand">[更新记录]一个有效的T-SQL语句或存储过程</param>
        /// <param name="dataSet">要更新到数据库的DataSet</param>
        /// <param name="tableName">要更新到数据库的DataTable</param>
        public static void UpdateDataset(SqlCommand insertCommand, SqlCommand deleteCommand,
SqlCommand updateCommand, DataSet dataSet, string tableName)
        {
            if (insertCommand == null) throw new ArgumentNullException("insertCommand");
            if (deleteCommand == null) throw new ArgumentNullException("deleteCommand");
            if (updateCommand == null) throw new ArgumentNullException("updateCommand");
            if (tableName == null || tableName.Length == 0) throw new
ArgumentNullException("tableName");

            // 创建SqlDataAdapter,当操作完成后释放.
            using (SqlDataAdapter dataAdapter = new SqlDataAdapter())
            {
                // 设置数据适配器命令
                dataAdapter.UpdateCommand = updateCommand;
                dataAdapter.InsertCommand = insertCommand;
                dataAdapter.DeleteCommand = deleteCommand;

                // 更新数据集改变到数据库
                dataAdapter.Update(dataSet, tableName);

                // 提交所有改变到数据集.
                dataSet.AcceptChanges();
            }
        }

        #endregion UpdateDataset 更新数据集


        #region CreateCommand 创建一条SqlCommand命令

        /// <summary>
        /// 创建SqlCommand命令,指定数据库连接对象,存储过程名和参数.
        /// </summary>
```

```
        /// <remarks>
        ///  示例:
        ///    SqlCommand command = CreateCommand(conn, "AddCustomer", "CustomerID",
"CustomerName");
        /// </remarks>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="sourceColumns">源表的列名称数组</param>
        /// <returns>返回SqlCommand命令</returns>
        public static SqlCommand CreateCommand(SqlConnection connection, string spName, params string[]
sourceColumns)
        {
            // CreateCommand #1
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            //  创建命令
            SqlCommand cmd = new SqlCommand(spName, connection);
            cmd.CommandType = CommandType.StoredProcedure;

            //  如果有参数值
            if ((sourceColumns != null) && (sourceColumns.Length > 0))
            {
                //  从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);

                //  将源表的列到映射到DataSet命令中.
                for (int index = 0; index < sourceColumns.Length; index++)
                    commandParameters[index].SourceColumn = sourceColumns[index];

                // Attach the discovered parameters to the SqlCommand object
                AttachParameters(cmd, commandParameters);
            }
            return cmd;
        }

        #endregion CreateCommand  创建一条SqlCommand命令


        #region ExecuteNonQueryTypedParams  类型化参数(DataRow)

        /// <summary>
```

```
        /// 执行指定连接数据库连接字符串的存储过程,使用DataRow做为参数值,返回受影响的行数.
        /// </summary>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQueryTypedParams(String connectionString, String spName, DataRow
dataRow)
        {
            // ExecuteNonQueryTypedParams #1
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteNonQuery(connectionString, CommandType.StoredProcedure,
spName, commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteNonQuery(connectionString, CommandType.StoredProcedure,
spName);
            }
        }

        /// <summary>
        /// 执行指定连接数据库连接对象的存储过程,使用DataRow做为参数值,返回受影响的行数.
        /// </summary>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回影响的行数</returns>
        public static int ExecuteNonQueryTypedParams(SqlConnection connection, String spName, DataRow
dataRow)
```

```
    {
        // ExecuteNonQueryTypedParams #2
        if (connection == null) throw new ArgumentNullException("connection");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
        //  如果row有值,存储过程必须初始化.
        if (dataRow != null && dataRow.ItemArray.Length > 0)
        {
            //  从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connection, spName);

            //  分配参数值
            AssignParameterValues(commandParameters, dataRow);

            return SqlHelper.ExecuteNonQuery(connection, CommandType.StoredProcedure, spName, commandParameters);
        }
        else
        {
            return SqlHelper.ExecuteNonQuery(connection, CommandType.StoredProcedure, spName);
        }
    }



    /// <summary>
    ///  执行指定连接数据库事物的存储过程,使用DataRow做为参数值,返回受影响的行数.
    /// </summary>
    /// <param name="transaction">一个有效的连接事务  object</param>
    /// <param name="spName">存储过程名称</param>
    /// <param name="dataRow">使用DataRow作为参数值</param>
    /// <returns>返回影响的行数</returns>
    public static int ExecuteNonQueryTypedParams(SqlTransaction transaction, String spName, DataRow dataRow)
    {
        // ExecuteNonQueryTypedParams #3
        if (transaction == null) throw new ArgumentNullException("transaction");
        if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
        // Sf the row has values, the store procedure parameters must be initialized
        if (dataRow != null && dataRow.ItemArray.Length > 0)
        {
```

```
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);
                return SqlHelper.ExecuteNonQuery(transaction, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteNonQuery(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteNonQueryTypedParams 类型化参数(DataRow)


        #region ExecuteDatasetTypedParams 类型化参数(DataRow)

        /// <summary>
        /// 执行指定连接数据库连接字符串的存储过程,使用DataRow做为参数值,返回DataSet.
        /// </summary>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回一个包含结果集的DataSet.</returns>
        public static DataSet ExecuteDatasetTypedParams(string connectionString, String spName, DataRow dataRow)
        {
            // ExecuteDatasetTypedParams #1
            if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");
            //如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 分配参数值
```

```
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteDataset(connectionString, CommandType.StoredProcedure,
spName, commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteDataset(connectionString, CommandType.StoredProcedure,
spName);
            }
        }

        /// <summary>
        /// 执行指定连接数据库连接对象的存储过程,使用DataRow做为参数值,返回DataSet.
        /// </summary>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回一个包含结果集的DataSet.</returns>
        public static DataSet ExecuteDatasetTypedParams(SqlConnection connection, String spName,
DataRow dataRow)
        {
            // ExecuteDatasetTypedParams #2
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteDataset(connection, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteDataset(connection, CommandType.StoredProcedure, spName);
            }
```

```
        }

        /// <summary>
        ///  执行指定连接数据库事务的存储过程,使用DataRow做为参数值,返回DataSet.
        /// </summary>
        /// <param name="transaction">一个有效的连接事务  object</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回一个包含结果集的DataSet.</returns>
        public static DataSet ExecuteDatasetTypedParams(SqlTransaction transaction, String spName,
DataRow dataRow)
        {
            // ExecuteDatasetTypedParams #3
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The
transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            //  如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                //  从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                //  分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteDataset(transaction, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteDataset(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteDatasetTypedParams  类型化参数(DataRow)


        #region ExecuteReaderTypedParams  类型化参数(DataRow)

        /// <summary>
```

```
        /// 执行指定连接数据库连接字符串的存储过程,使用DataRow做为参数值,返回DataReader.
        /// </summary>
        /// <param name="connectionString">一个有效的数据库连接字符串</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReaderTypedParams(String connectionString, String spName,
DataRow dataRow)
        {
            // ExecuteReaderTypedParams #1
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteReader(connectionString, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteReader(connectionString, CommandType.StoredProcedure,
spName);
            }
        }

        /// <summary>
        /// 执行指定连接数据库连接对象的存储过程,使用DataRow做为参数值,返回DataReader.
        /// </summary>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回包含结果集的SqlDataReader</returns>
        public static SqlDataReader ExecuteReaderTypedParams(SqlConnection connection, String spName,
DataRow dataRow)
```

```
    {
        // ExecuteReaderTypedParams #2
        if (connection == null) throw new ArgumentNullException("connection");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

        // 如果row有值,存储过程必须初始化.
        if (dataRow != null && dataRow.ItemArray.Length > 0)
        {
            // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connection, spName);

            // 分配参数值
            AssignParameterValues(commandParameters, dataRow);

            return SqlHelper.ExecuteReader(connection, CommandType.StoredProcedure, spName, commandParameters);
        }
        else
        {
            return SqlHelper.ExecuteReader(connection, CommandType.StoredProcedure, spName);
        }
    }

    /// <summary>
    /// 执行指定连接数据库事物的存储过程,使用DataRow做为参数值,返回DataReader.
    /// </summary>
    /// <param name="transaction">一个有效的连接事务 object</param>
    /// <param name="spName">存储过程名称</param>
    /// <param name="dataRow">使用DataRow作为参数值</param>
    /// <returns>返回包含结果集的SqlDataReader</returns>
    public static SqlDataReader ExecuteReaderTypedParams(SqlTransaction transaction, String spName, DataRow dataRow)
    {
        // ExecuteReaderTypedParams #3
        if (transaction == null) throw new ArgumentNullException("transaction");
        if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

        // 如果row有值,存储过程必须初始化.
        if (dataRow != null && dataRow.ItemArray.Length > 0)
        {
```

```
            // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

            // 分配参数值
            AssignParameterValues(commandParameters, dataRow);

            return SqlHelper.ExecuteReader(transaction, CommandType.StoredProcedure, spName, commandParameters);
        }
        else
        {
            return SqlHelper.ExecuteReader(transaction, CommandType.StoredProcedure, spName);
        }
    }

    #endregion ExecuteReaderTypedParams 类型化参数(DataRow)


    #region ExecuteScalarTypedParams 类型化参数(DataRow)

    /// <summary>
    /// 执行指定连接数据库连接字符串的存储过程,使用DataRow做为参数值,返回结果集中的第一行第一列.
    /// </summary>
    /// <param name="connectionString">一个有效的数据库连接字符串</param>
    /// <param name="spName">存储过程名称</param>
    /// <param name="dataRow">使用DataRow作为参数值</param>
    /// <returns>返回结果集中的第一行第一列</returns>
    public static object ExecuteScalarTypedParams(String connectionString, String spName, DataRow dataRow)
    {
        // ExecuteScalarTypedParams #1
        if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

        // 如果row有值,存储过程必须初始化.
        if (dataRow != null && dataRow.ItemArray.Length > 0)
        {
            // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
            SqlParameter[] commandParameters =
```

```
SqlHelperParameterCache.GetSpParameterSet(connectionString, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteScalar(connectionString, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteScalar(connectionString, CommandType.StoredProcedure,
spName);
            }
        }

        /// <summary>
        /// 执行指定连接数据库连接对象的存储过程,使用DataRow做为参数值,返回结果集中的第一行第
一列.
        /// </summary>
        /// <param name="connection">一个有效的数据库连接对象</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalarTypedParams(SqlConnection connection, String spName, DataRow
dataRow)
        {
            // ExecuteScalarTypedParams #2
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(connection, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteScalar(connection, CommandType.StoredProcedure, spName,
commandParameters);
            }
```

```
            else
            {
                return SqlHelper.ExecuteScalar(connection, CommandType.StoredProcedure, spName);
            }
        }

        /// <summary>
        /// 执行指定连接数据库事务的存储过程,使用DataRow做为参数值,返回结果集中的第一行第一列.
        /// </summary>
        /// <param name="transaction">一个有效的连接事务 object</param>
        /// <param name="spName">存储过程名称</param>
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回结果集中的第一行第一列</returns>
        public static object ExecuteScalarTypedParams(SqlTransaction transaction, String spName, DataRow dataRow)
        {
            // ExecuteScalarTypedParams #3
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
                SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteScalar(transaction, CommandType.StoredProcedure, spName, commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteScalar(transaction, CommandType.StoredProcedure, spName);
            }
        }

        #endregion ExecuteScalarTypedParams 类型化参数(DataRow)
```

```
#region ExecuteXmlReaderTypedParams  类型化参数(DataRow)

/// <summary>
/// 执行指定连接数据库连接对象的存储过程,使用DataRow做为参数值,返回XmlReader类型的结果集.
/// </summary>
/// <param name="connection">一个有效的数据库连接对象</param>
/// <param name="spName">存储过程名称</param>
/// <param name="dataRow">使用DataRow作为参数值</param>
/// <returns>返回XmlReader结果集对象.</returns>
public static XmlReader ExecuteXmlReaderTypedParams(SqlConnection connection, String spName, DataRow dataRow)
{
    // ExecuteXmlReaderTypedParams #1
    if (connection == null) throw new ArgumentNullException("connection");
    if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

    // 如果row有值,存储过程必须初始化.
    if (dataRow != null && dataRow.ItemArray.Length > 0)
    {
        // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到缓存中. ()
        SqlParameter[] commandParameters = SqlHelperParameterCache.GetSpParameterSet(connection, spName);

        // 分配参数值
        AssignParameterValues(commandParameters, dataRow);

        return SqlHelper.ExecuteXmlReader(connection, CommandType.StoredProcedure, spName, commandParameters);
    }
    else
    {
        return SqlHelper.ExecuteXmlReader(connection, CommandType.StoredProcedure, spName);
    }
}

/// <summary>
/// 执行指定连接数据库事务的存储过程,使用DataRow做为参数值,返回XmlReader类型的结果集.
/// </summary>
/// <param name="transaction">一个有效的连接事务  object</param>
/// <param name="spName">存储过程名称</param>
```

```
        /// <param name="dataRow">使用DataRow作为参数值</param>
        /// <returns>返回XmlReader结果集对象.</returns>
        public static XmlReader ExecuteXmlReaderTypedParams(SqlTransaction transaction, String spName,
DataRow dataRow)
        {
            // ExecuteXmlReaderTypedParams #2
            if (transaction == null) throw new ArgumentNullException("transaction");
            if (transaction != null && transaction.Connection == null) throw new ArgumentException("The
transaction was rollbacked or commited, please provide an open transaction.", "transaction");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            // 如果row有值,存储过程必须初始化.
            if (dataRow != null && dataRow.ItemArray.Length > 0)
            {
                // 从缓存中加载存储过程参数,如果缓存中不存在则从数据库中检索参数信息并加载到
缓存中. ()
                SqlParameter[] commandParameters =
SqlHelperParameterCache.GetSpParameterSet(transaction.Connection, spName);

                // 分配参数值
                AssignParameterValues(commandParameters, dataRow);

                return SqlHelper.ExecuteXmlReader(transaction, CommandType.StoredProcedure, spName,
commandParameters);
            }
            else
            {
                return SqlHelper.ExecuteXmlReader(transaction, CommandType.StoredProcedure, spName);
            }

        }

        #endregion ExecuteXmlReaderTypedParams 类型化参数(DataRow)
}
```

SqlHelperParameterCatch.cs 文件源代码和注释：

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Data.SqlClient;
using System.Collections;
using System.Data;
```

```
/// <summary>
///SqlHelperParameterCache 的摘要说明
/// </summary>
public sealed class SqlHelperParameterCache
{
    #region private methods, variables, and constructors

    //Since this class provides only static methods, make the default constructor private to prevent
    //instances from being created with "new SqlHelperParameterCache()"
    private SqlHelperParameterCache() { }

    private static Hashtable paramCache = Hashtable.Synchronized(new Hashtable());

    /// <summary>
    /// Resolve at run time the appropriate set of SqlParameters for a stored procedure
    /// </summary>
    /// <param name="connection">A valid SqlConnection object</param>
    /// <param name="spName">The name of the stored procedure</param>
    /// <param name="includeReturnValueParameter">Whether or not to include their return value
parameter</param>
    /// <returns>The parameter array discovered.</returns>
    private static SqlParameter[] DiscoverSpParameterSet(SqlConnection connection, string spName, bool
includeReturnValueParameter)
    {
        if (connection == null) throw new ArgumentNullException("connection");
        if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

        SqlCommand cmd = new SqlCommand(spName, connection);
        cmd.CommandType = CommandType.StoredProcedure;

        connection.Open();
        SqlCommandBuilder.DeriveParameters(cmd);
        connection.Close();

        if (!includeReturnValueParameter)
        {
            cmd.Parameters.RemoveAt(0);
        }

        SqlParameter[] discoveredParameters = new SqlParameter[cmd.Parameters.Count];

        cmd.Parameters.CopyTo(discoveredParameters, 0);

        // Init the parameters with a DBNull value
```

```
            foreach (SqlParameter discoveredParameter in discoveredParameters)
            {
                discoveredParameter.Value = DBNull.Value;
            }
            return discoveredParameters;
        }

        /// <summary>
        /// Deep copy of cached SqlParameter array
        /// </summary>
        /// <param name="originalParameters"></param>
        /// <returns></returns>
        private static SqlParameter[] CloneParameters(SqlParameter[] originalParameters)
        {
            SqlParameter[] clonedParameters = new SqlParameter[originalParameters.Length];

            for (int i = 0, j = originalParameters.Length; i < j; i++)
            {
                clonedParameters[i] = (SqlParameter)((ICloneable)originalParameters[i]).Clone();
            }

            return clonedParameters;
        }

        #endregion private methods, variables, and constructors

        #region caching functions

        /// <summary>
        /// Add parameter array to the cache
        /// </summary>
        /// <param name="connectionString">A valid connection string for a SqlConnection</param>
        /// <param name="commandText">The stored procedure name or T-SQL command</param>
        /// <param name="commandParameters">An array of SqlParamters to be cached</param>
        public static void CacheParameterSet(string connectionString, string commandText, params
SqlParameter[] commandParameters)
        {
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (commandText == null || commandText.Length == 0) throw new
ArgumentNullException("commandText");

            string hashKey = connectionString + ":" + commandText;
```

```
            paramCache[hashKey] = commandParameters;
        }

        /// <summary>
        /// Retrieve a parameter array from the cache
        /// </summary>
        /// <param name="connectionString">A valid connection string for a SqlConnection</param>
        /// <param name="commandText">The stored procedure name or T-SQL command</param>
        /// <returns>An array of SqlParamters</returns>
        public static SqlParameter[] GetCachedParameterSet(string connectionString, string commandText)
        {
            if (connectionString == null || connectionString.Length == 0) throw new
ArgumentNullException("connectionString");
            if (commandText == null || commandText.Length == 0) throw new
ArgumentNullException("commandText");

            string hashKey = connectionString + ":" + commandText;

            SqlParameter[] cachedParameters = paramCache[hashKey] as SqlParameter[];
            if (cachedParameters == null)
            {
                return null;
            }
            else
            {
                return CloneParameters(cachedParameters);
            }
        }

        #endregion caching functions

        #region Parameter Discovery Functions

        /// <summary>
        /// Retrieves the set of SqlParameters appropriate for the stored procedure
        /// </summary>
        /// <remarks>
        /// This method will query the database for this information, and then store it in a cache for future
requests.
        /// </remarks>
        /// <param name="connectionString">A valid connection string for a SqlConnection</param>
        /// <param name="spName">The name of the stored procedure</param>
        /// <returns>An array of SqlParameters</returns>
        public static SqlParameter[] GetSpParameterSet(string connectionString, string spName)
```

```
        {
            return GetSpParameterSet(connectionString, spName, false);
        }

        /// <summary>
        /// Retrieves the set of SqlParameters appropriate for the stored procedure
        /// </summary>
        /// <remarks>
        /// This method will query the database for this information, and then store it in a cache for future requests.
        /// </remarks>
        /// <param name="connectionString">A valid connection string for a SqlConnection</param>
        /// <param name="spName">The name of the stored procedure</param>
        /// <param name="includeReturnValueParameter">A bool value indicating whether the return value parameter should be included in the results</param>
        /// <returns>An array of SqlParameters</returns>
        public static SqlParameter[] GetSpParameterSet(string connectionString, string spName, bool includeReturnValueParameter)
        {
            if (connectionString == null || connectionString.Length == 0) throw new ArgumentNullException("connectionString");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                return GetSpParameterSetInternal(connection, spName, includeReturnValueParameter);
            }
        }

        /// <summary>
        /// Retrieves the set of SqlParameters appropriate for the stored procedure
        /// </summary>
        /// <remarks>
        /// This method will query the database for this information, and then store it in a cache for future requests.
        /// </remarks>
        /// <param name="connection">A valid SqlConnection object</param>
        /// <param name="spName">The name of the stored procedure</param>
        /// <returns>An array of SqlParameters</returns>
        internal static SqlParameter[] GetSpParameterSet(SqlConnection connection, string spName)
        {
            return GetSpParameterSet(connection, spName, false);
        }
```

```
        /// <summary>
        /// Retrieves the set of SqlParameters appropriate for the stored procedure
        /// </summary>
        /// <remarks>
        /// This method will query the database for this information, and then store it in a cache for future
requests.
        /// </remarks>
        /// <param name="connection">A valid SqlConnection object</param>
        /// <param name="spName">The name of the stored procedure</param>
        /// <param name="includeReturnValueParameter">A bool value indicating whether the return value
parameter should be included in the results</param>
        /// <returns>An array of SqlParameters</returns>
        internal static SqlParameter[] GetSpParameterSet(SqlConnection connection, string spName, bool
includeReturnValueParameter)
        {
            if (connection == null) throw new ArgumentNullException("connection");
            using (SqlConnection clonedConnection = (SqlConnection)((ICloneable)connection).Clone())
            {
                return GetSpParameterSetInternal(clonedConnection, spName,
includeReturnValueParameter);
            }
        }

        /// <summary>
        /// Retrieves the set of SqlParameters appropriate for the stored procedure
        /// </summary>
        /// <param name="connection">A valid SqlConnection object</param>
        /// <param name="spName">The name of the stored procedure</param>
        /// <param name="includeReturnValueParameter">A bool value indicating whether the return value
parameter should be included in the results</param>
        /// <returns>An array of SqlParameters</returns>
        private static SqlParameter[] GetSpParameterSetInternal(SqlConnection connection, string spName,
bool includeReturnValueParameter)
        {
            if (connection == null) throw new ArgumentNullException("connection");
            if (spName == null || spName.Length == 0) throw new ArgumentNullException("spName");

            string hashKey = connection.ConnectionString + ":" + spName + (includeReturnValueParameter ?
":include ReturnValue Parameter" : "");

            SqlParameter[] cachedParameters;

            cachedParameters = paramCache[hashKey] as SqlParameter[];
            if (cachedParameters == null)
```

```
            {
                SqlParameter[] spParameters = DiscoverSpParameterSet(connection, spName,
includeReturnValueParameter);
                paramCache[hashKey] = spParameters;
                cachedParameters = spParameters;
            }

            return CloneParameters(cachedParameters);
        }

        #endregion Parameter Discovery Functions
}
```