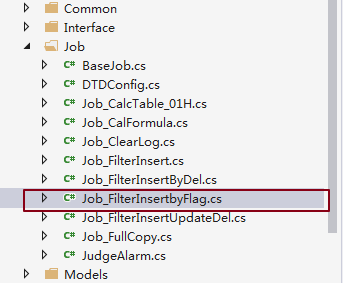
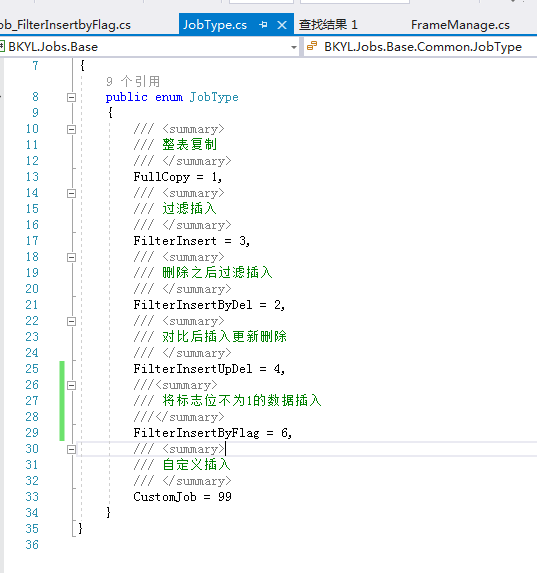
## 按标志位flag同步数据（模式选择6）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \*配置信息 |  | 备注 |  |  |
| 配置类型 | 6 | (1-删除所有数据、2-删除时间周期数据、3-直插入最新数据、99-自定义任务) |  |  |
| 删除数据周期时间 | 172800 | (以秒为单位) | 5760 |  |
| 刷新周期 | 30 | (以秒为单位) |  |  |
| / |  |  |  |  |
| \*源数据库信息 |  |  |  |  |
| 源-表名 | T\_MATERIAL\_TEAM\_YIELD |  |  |  |
| 源-主键 |  | 默认TIMESTAMP |  |  |
| 源-时间时序表时间字段 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |  |
| 源-查询过滤条件 | flag != 1 | 查表后面的语句 例如 字段a!=-9999 and 字段b=3 默认不过滤 |  |  |
| / |  |  |  |  |
| \*目标数据库信息 |  |  |  |  |
| 目标-是否含有关联触发器 | FALSE | 表内触发器是否含有对本表的操作 |  |  |
| 目标-表名 | T\_MATERIAL\_TEAM\_YIELD\_FAL |  |  |  |
| 目标-主键 |  | 默认TIMESTAMP |  |  |
| 目标-时间时序表时间字段 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |  |
| 目标-删除过滤条件 |  | 查表后面的语句 例如 字段a!=-9999 and 字段b=3 默认不过滤 |  |  |
| 插入完成后执行语句 |  | 逗号请用<%comma%> |  |  |
| / |  |  |  |  |
| \*自定义任务 |  |  |  |  |
| DLL名称 |  | 可为exe |  |  |
| 类名称 |  | 含命名空间 |  |  |
| \*列信息 |  |  |  |  |
| 源-列字段 | 源-列数据类型(date number string dump) | 目标-列字段 | 目标--列数据类型(date number string) | 列描述 |
| TEAM\_BEGIN\_TIME | date | TEAM\_BEGIN\_TIME | date | a字段有格式 |
| TEAM\_END\_TIME | date | TEAM\_END\_TIME | date |  |
| SHIFT | string | SHIFT | string |  |
| TEAM | string | TEAM | string |  |
| YIELD\_BEGIN | string | YIELD\_BEGIN | string |  |
| YIELD\_END | string | YIELD\_END | string |  |
| YIELD\_SUM | string | YIELD\_SUM | string |  |
| RUN\_TIME\_MIN | string | RUN\_TIME\_MIN | string |  |
| OPERATION\_RATE | string | OPERATION\_RATE | string |  |
| TIME\_YIELD | string | TIME\_YIELD | string |  |
| OPERATING\_FACTOR | string | OPERATING\_FACTOR | string |  |
| ID | number | ID | number |  |
| / |  |  |  |  |

1. 新增Job\_FilterInsertbyFlag.cs



1. 新增JobType.cs程序



1. 修改JobFactory.cs



1. 修改Job\_FilterInsertbyFlag.cs

获取当前日期，截取天

所有的时间都减去30天

代码如下：

using BFES.DataAccess;

using BKYL.Jobs.Base.Utils;

using System;

using System.Data;

namespace BKYL.Jobs.Base.Plugin

{

public class Job\_FilterInsertbyFlag : BaseJob

{

public override void Dispose()

{

}

public override void InitTask()

{

}

//public override void RunTask(DateTime currentTime)

//{

public override void RunTask(DateTime currentTime)

{

string log = "";

DataTable dt = null;

try

{

object S\_Max;

object T\_Max;

decimal S\_Max\_Int = 0;

decimal T\_Max\_Int = 0;

DateTime S\_Max\_Date = new DateTime();

DateTime T\_Max\_Date = new DateTime();

bool IsDateTimeType = true;

#region 查询最大时间

using (IDataBase iDataBase = DalFactory.GreateIDataBase(S\_DBSource))

{

try

{

string sql = string.Format("select max({0}) from {1} alias", tableConfig.S\_TableSequential, tableConfig.S\_TableName);

if (!string.IsNullOrEmpty(tableConfig.S\_Filter))

{

sql += " where " + tableConfig.S\_Filter;

}

S\_Max = iDataBase.GetScalar(sql);

}

catch (Exception ee)

{

throw ee;

}

}

using (IDataBase iDataBase = DalFactory.GreateIDataBase(T\_DBSource))

{

try

{

T\_Max = DateTime.Now.AddDays(-30).Date.ToString();

//T\_Max = iDataBase.GetScalar(string.Format("select max({0}) from {1} ", tableConfig.T\_TableSequential, tableConfig.T\_TableName));

}

catch (Exception ex)

{

throw ex;

}

}

#endregion

#region 类型判断

Type S\_Maxtype = S\_Max.GetType();

if (S\_Maxtype == typeof(DateTime))

{

S\_Max\_Date = S\_Max.ToDate();

T\_Max\_Date = T\_Max.ToDate();

//T\_Max\_Date = S\_Max\_Date.AddDays(-1);

if (S\_Max\_Date <= T\_Max\_Date)

{

return;

}

}

else if (S\_Maxtype == typeof(System.Decimal) || S\_Maxtype == typeof(System.Double) || S\_Maxtype == typeof(System.Int32) || S\_Maxtype == typeof(int) || S\_Maxtype == typeof(System.Single) || S\_Maxtype == typeof(float))

{

S\_Max\_Int = S\_Max.ToDecimal();

T\_Max\_Int = T\_Max.ToDecimal();

if (S\_Max\_Int <= T\_Max\_Int)

{

return;

}

IsDateTimeType = false;

}

else if (S\_Maxtype == typeof(System.String))

{

throw new Exception("源-时间时序表时间字段是字符串类型，如果是时间请转换时间类型！！" + S\_Maxtype.Name + " 值Value:" + S\_Max);

}

else if (S\_Maxtype == typeof(DBNull))

{

return;

}

else

{

throw new Exception("缺失最大值匹配的类型！！" + S\_Maxtype.Name + " 值Value:" + S\_Max);

}

#endregion

#region 查询数据

using (IDataBase iDataBase = DalFactory.GreateIDataBase(S\_DBSource))

{

string sql = null;

try

{

if (IsDateTimeType)

{

sql = string.Format(tableConfig.SelectSql + " where {0}>={1} and {0}<{2} ", tableConfig.S\_TableSequential

, DataBaseFactory.ConvertTimeString(T\_Max\_Date.AddSeconds(1), S\_DBSource.DBType), DataBaseFactory.ConvertTimeString(S\_Max\_Date.AddSeconds(1), S\_DBSource.DBType));

}

else

{

sql = string.Format(tableConfig.SelectSql + " where {0}>{1} and {0}<={2} ", tableConfig.S\_TableSequential

, T\_Max\_Int, S\_Max\_Int);

}

if (!string.IsNullOrEmpty(tableConfig.S\_Filter))

{

sql += " and " + tableConfig.S\_Filter;

}

dt = iDataBase.GetDataTable(sql);

//转换中文乱码

dt = RemoveDump(dt);

}

catch (Exception ex)

{

if (ex.Message != null && ex.Message.Contains("\*&\*&"))

{

throw ex;

}

else

{

throw new Exception(sql + "语句错误！");

}

}

}

#endregion

#region 插入数据

bool issuccess = false;

using (IDataBase iDataBase = DalFactory.GreateIDataBase(T\_DBSource))

{

try

{

iDataBase.BeginTran();

issuccess = iDataBase.BulkInsert(tableConfig.T\_TableName, Columns, Columns, dt, tableConfig.IsExistTri);

iDataBase.CommitTran();

log += string.Format("{0}表插入成功 {1}条!", tableConfig.T\_TableName, dt.Rows.Count);

UpdateFlag(iDataBase, issuccess, tableConfig.UpdateSql, ref log);

}

catch (Exception ex)

{

iDataBase.RollbackTran();

issuccess = false;

throw ex;

}

}

#endregion

//置标志位

using (IDataBase iDataBase = DalFactory.GreateIDataBase(S\_DBSource))

{

string sql = null;

try

{

if (IsDateTimeType)

{

sql = string.Format("update {0} ", tableConfig.S\_TableName);

sql += string.Format(tableConfig.UpdateSql + " set flag='1' where {0}>={1} and {0}<{2} ", tableConfig.S\_TableSequential

, DataBaseFactory.ConvertTimeString(T\_Max\_Date.AddSeconds(1), S\_DBSource.DBType), DataBaseFactory.ConvertTimeString(S\_Max\_Date.AddSeconds(1), S\_DBSource.DBType));

}

else

{

sql = string.Format("update {0} ", tableConfig.S\_TableName);

sql = string.Format(tableConfig.UpdateSql + " set flag='1' where {0}>{1} and {0}<={2} ", tableConfig.S\_TableSequential

, T\_Max\_Int, S\_Max\_Int);

}

if (!string.IsNullOrEmpty(tableConfig.S\_Filter))

{

sql += " and " + tableConfig.S\_Filter;

}

iDataBase.ExecuteCommand(sql);

}

catch (Exception ex)

{

if (ex.Message != null && ex.Message.Contains("\*&\*&"))

{

throw ex;

}

else

{

throw new Exception(sql + "语句错误！");

}

}

}

Outputlog(log);

}

catch (Exception ee)

{

throw ee;

}

finally

{

if (dt != null)

{

dt.Clear();

dt.Dispose();

dt = null;

}

log = null;

}

}

public override void RunTaskException(DateTime currentTime, Exception exception)

{

GlobalObject.RichTextErrorLog.AppendTextByAsync("任务进程(" + currentTime.ToString("yyyy-MM-dd :HH:mm:ss") + ")：" + tableConfig.FolderName + "->" + tableConfig.FileName + "\r\n " + exception.Message, System.Drawing.Color.Red);

Writelog(exception.Message + exception.StackTrace);

}

}

}

## 源表更新、修改、删除（模式选4）

注：源表与目标表需要设置主键

两表主键对比，源表更新、删除、修改同时作用于目标表（相同操作，即更新、删除、修改）

|  |  |  |  |
| --- | --- | --- | --- |
| \*配置信息 |  | 备注 |  |
| 配置类型 | 4 | (1-删除所有数据、2-删除时间周期数据、3-直插入最新数据、99-自定义任务) |  |
| 删除数据周期时间 | 432000 | (以秒为单位) | 5760 |
| 刷新周期 | 30 | (以秒为单位) |  |
| / |  |  |  |
| \*源数据库信息 |  |  |  |
| 源-表名 | T\_MATERIAL\_TEAM\_YIELD |  |  |
| 源-主键 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |
| 源-时间时序表时间字段 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |
| 源-查询过滤条件 |  | 查表后面的语句 例如 字段a!=-9999 and 字段b=3 默认不过滤 |  |
| / |  |  |  |
| \*目标数据库信息 |  |  |  |
| 目标-是否含有关联触发器 | FALSE | 表内触发器是否含有对本表的操作 |  |
| 目标-表名 | T\_MATERIAL\_TEAM\_YIELD\_FAL |  |  |
| 目标-主键 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |
| 目标-时间时序表时间字段 | TEAM\_BEGIN\_TIME | 默认TIMESTAMP |  |
| 目标-删除过滤条件 |  | 查表后面的语句 例如 字段a!=-9999 and 字段b=3 默认不过滤 |  |
| 插入完成后执行语句 |  | 逗号请用<%comma%> |  |
| / |  |  |  |
| \*自定义任务 |  |  |  |
| DLL名称 |  | 可为exe |  |
| 类名称 |  | 含命名空间 |  |
| \*列信息 |  |  |  |
| 源-列字段 | 源-列数据类型(date number string dump) | 目标-列字段 | 目标--列数据类型(date number string) |
| TEAM\_BEGIN\_TIME | date | TEAM\_BEGIN\_TIME | date |
| TEAM\_END\_TIME | date | TEAM\_END\_TIME | date |
| SHIFT | string | SHIFT | string |
| TEAM | string | TEAM | string |
| YIELD\_BEGIN | string | YIELD\_BEGIN | string |
| YIELD\_END | string | YIELD\_END | string |
| YIELD\_SUM | string | YIELD\_SUM | string |
| RUN\_TIME\_MIN | string | RUN\_TIME\_MIN | string |
| OPERATION\_RATE | string | OPERATION\_RATE | string |
| TIME\_YIELD | string | TIME\_YIELD | string |
| OPERATING\_FACTOR | string | OPERATING\_FACTOR | string |
| ID | number | ID | number |
| / |  |  |  |