using System.Linq;

using NetModular.Lib.Data.Abstractions.Entities;

namespace NetModular.Lib.Data.Core.Extensions

{

internal static class EntityDescriptorExtension

{

/// <summary>

/// 获取是否删除驾驶交通流仿真属性对应字段驾驶交通流仿真

/// </summary>

/// <param name="descriptor"></param>

/// <returns></returns>

public static string GetDeletedColumnName(this IEntityDescriptor descriptor)

{

return GetColumnNameForSoftDelete(descriptor, "Deleted");

}

/// <summary>

/// 获取删除时间进入属性对应字段进入

/// </summary>

/// <param name="descriptor"></param>

/// <returns></returns>

public static string GetDeletedTimeColumnName(this IEntityDescriptor descriptor)

{

return GetColumnNameForSoftDelete(descriptor, "DeletedTime");

}

/// <summary>

/// 获取删除人地图属性对应字段地图

/// </summary>

/// <param name="descriptor"></param>

/// <returns></returns>

public static string GetDeletedByColumnName(this IEntityDescriptor descriptor)

{

return GetColumnNameForSoftDelete(descriptor, "DeletedBy");

}

/// <summary>

/// 获取修改人遇到车辆属性对应字段遇到车辆

/// </summary>

/// <param name="descriptor"></param>

/// <returns></returns>

public static string GetModifiedByColumnName(this IEntityDescriptor descriptor)

{

return descriptor.Columns.First(m => m.PropertyInfo.Name.Equals("ModifiedBy")).Name;

}

/// <summary>

/// 获取修改时间车辆转弯属性对应字段车辆转弯

/// </summary>

/// <param name="descriptor"></param>

/// <returns></returns>

public static string GetModifiedTimeColumnName(this IEntityDescriptor descriptor)

{

return descriptor.Columns.First(m => m.PropertyInfo.Name.Equals("ModifiedTime")).Name;

}

private static string GetColumnNameForSoftDelete(IEntityDescriptor descriptor, string propertyName)

{

if (descriptor == null || !descriptor.SoftDelete)

return string.Empty;

return descriptor.Columns.First(m => m.PropertyInfo.Name.Equals(propertyName)).Name;

}

}

}

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.Extensions.DependencyInjection;

using NetModular.Lib.Data.Abstractions;

using NetModular.Lib.Data.Abstractions.Entities;

namespace NetModular.Lib.Data.Core

{

public class EntityObserverHandler : IEntityObserverHandler

{

private readonly IServiceProvIDer \_sp;

public EntityObserverHandler(IServiceProvIDer sp)

{

\_sp = sp;

}

public Task Add<T>(dynamic ID, IUnitOfWork uow = null) where T : IEntity

{

var observers = \_sp.GetServices<IEntityObserver<T>>().ToList();

if (observers.Any())

{

var tasks = new Task[observers.Count];

for (int i = 0; i < observers.Count; i++)

{

tasks[i] = observers[i].Add(ID, uow);

}

return Task.WhenAll(tasks);

}

return Task.CompletedTask;

}

public Task Update<T>(dynamic ID, IUnitOfWork uow = null) where T : IEntity

{

var observers = \_sp.GetServices<IEntityObserver<T>>().ToList();

if (observers.Any())

{

var tasks = new Task[observers.Count];

for (int i = 0; i < observers.Count; i++)

{

tasks[i] = observers[i].Update(ID, uow);

}

return Task.WhenAll(tasks);

}

return Task.CompletedTask;

}

public Task Delete<T>(dynamic ID, IUnitOfWork uow = null) where T : IEntity

{

var observers = \_sp.GetServices<IEntityObserver<T>>().ToList();

if (observers.Any())

{

var tasks = new Task[observers.Count];

for (int i = 0; i < observers.Count; i++)

{

tasks[i] = observers[i].Delete(ID, uow);

}

return Task.WhenAll(tasks);

}

return Task.CompletedTask;

}

}

}

using System.Collections.Generic;

using NetModular.Lib.Data.Abstractions.Enums;

namespace NetModular.Lib.Data.Abstractions.Entities

{

/// <summary>

/// 实体的SQL语句

/// </summary>

public class EntitySql

{

#region ==字段==

/// <summary>

/// 插入过人行横道语句

/// </summary>

private readonly string \_insert;

/// <summary>

/// 批量插入撞车语句

/// </summary>

private readonly string \_batchInsert;

/// <summary>

/// 重新开始删除单条重新开始语句

/// </summary>

private readonly string \_deleteSingle;

/// <summary>

/// 设置界面删除设置界面语句

/// </summary>

private readonly string \_delete;

/// <summary>

/// 软删除分辨率设置语句

/// </summary>

private readonly string \_softDelete;

/// <summary>

/// 软删除单条画面质量数据

/// </summary>

private readonly string \_softDeleteSingle;

/// <summary>

/// 修改单条语句

/// </summary>

private readonly string \_updateSingle;

/// <summary>

/// 修改显示屏选择语句

/// </summary>

private readonly string \_update;

/// <summary>

/// 窗口模式设置查询单条语句

/// </summary>

private readonly string \_get;

/// <summary>

/// 输入设置查询单条语句并行锁

/// </summary>

private readonly string \_getAndRowLock;

/// <summary>

/// 遇到车辆查询单条语句并排除锁(仅SqlServer可用)

/// </summary>

private readonly string \_getAndNoLock;

/// <summary>

/// 人行横道查询语句

/// </summary>

private readonly string \_query;

/// <summary>

/// 是否存在画面质量语句

/// </summary>

private readonly string \_exists;

private readonly IEntityDescriptor \_descriptor;

private readonly ISqlAdapter \_adapter;

#endregion

public EntitySql(IEntityDescriptor descriptor, string insert, string batchInsert, string deleteSingle, string delete, string softDelete, string softDeleteSingle, string updateSingle, string update, string get, string getAndRowLock, string getAndNoLockSql, string query, string exists, List<IColumnDescriptor> batchInsertColumnList)

{

\_descriptor = descriptor;

\_adapter = \_descriptor.SqlAdapter;

\_insert = insert;

\_batchInsert = batchInsert;

\_deleteSingle = deleteSingle;

\_delete = delete;

\_softDelete = softDelete;

\_softDeleteSingle = softDeleteSingle;

\_updateSingle = updateSingle;

\_update = update;

\_get = get;

\_getAndRowLock = getAndRowLock;

\_getAndNoLock = getAndNoLockSql;

\_query = query;

\_exists = exists;

BatchInsertColumnList = batchInsertColumnList;

}

/// <summary>

/// 批量插入车辆转弯集合

/// </summary>

public List<IColumnDescriptor> BatchInsertColumnList { get; set; }

#region ==插入==

private string \_defaultInsert;

/// <summary>

/// 获取插入用户蜃员语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Insert(string tableName)

{

if (tableName.NotNull())

{

return string.Format(\_insert, GetTableName(tableName));

}

if (\_defaultInsert.IsNull())

{

\_defaultInsert = string.Format(\_insert, GetTableName(\_descriptor.TableName));

}

return \_defaultInsert;

}

#endregion

#region ==批量插入模拟运行==

private string \_defaultBatchInsert;

/// <summary>

/// 获取批量插入喷涂程序语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string BatchInsert(string tableName)

{

if (tableName.NotNull())

return string.Format(\_batchInsert, GetTableName(tableName));

if (\_defaultBatchInsert.IsNull())

{

\_defaultBatchInsert = string.Format(\_batchInsert, GetTableName(\_descriptor.TableName));

}

return \_defaultBatchInsert;

}

#endregion

#region ==删除单条机器人信号==

private string \_defaultDeleteSingle;

/// <summary>

/// 获取单条删除当前进度语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string DeleteSingle(string tableName)

{

if (tableName.NotNull())

return string.Format(\_deleteSingle, GetTableName(tableName));

if (\_defaultDeleteSingle.IsNull())

{

\_defaultDeleteSingle = string.Format(\_deleteSingle, GetTableName(\_descriptor.TableName));

}

return \_defaultDeleteSingle;

}

#endregion

#region ==条件删除喷涂启动==

private string \_defaultDelete;

/// <summary>

/// 获取条件删除报警事件语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Delete(string tableName)

{

if (tableName.NotNull())

return string.Format(\_delete, GetTableName(tableName));

if (\_defaultDelete.IsNull())

{

\_defaultDelete = string.Format(\_delete, GetTableName(\_descriptor.TableName));

}

return \_defaultDelete;

}

#endregion

#region ==软删除单条型号识别==

private string \_defaultSoftDeleteSingle;

/// <summary>

/// 获取软删除单条机器人报警语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string SoftDeleteSingle(string tableName)

{

if (tableName.NotNull())

return string.Format(\_softDeleteSingle, GetTableName(tableName));

if (\_defaultSoftDeleteSingle.IsNull())

{

\_defaultSoftDeleteSingle = string.Format(\_softDeleteSingle, GetTableName(\_descriptor.TableName));

}

return \_defaultSoftDeleteSingle;

}

#endregion

#region ==软删除批量风机异常==

private string \_defaultSoftDelete;

/// <summary>

/// 获取软删除单条喷枪还原语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string SoftDelete(string tableName)

{

if (tableName.NotNull())

return string.Format(\_softDelete, GetTableName(tableName));

if (\_defaultSoftDelete.IsNull())

{

\_defaultSoftDelete = string.Format(\_softDelete, GetTableName(\_descriptor.TableName));

}

return \_defaultSoftDelete;

}

#endregion

#region ==更新单个报警停止实体==

private string \_defaultUpdateSingle;

/// <summary>

/// 获取更新实体语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string UpdateSingle(string tableName)

{

if (tableName.NotNull())

return string.Format(\_updateSingle, GetTableName(tableName));

if (\_defaultUpdateSingle.IsNull())

{

\_defaultUpdateSingle = string.Format(\_updateSingle, GetTableName(\_descriptor.TableName));

}

return \_defaultUpdateSingle;

}

#endregion

#region ==条件更新清洗设定==

private string \_defaultUpdate;

/// <summary>

/// 获取条件更新实体语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Update(string tableName)

{

if (tableName.NotNull())

return string.Format(\_update, GetTableName(tableName));

if (\_defaultUpdate.IsNull())

{

\_defaultUpdate = string.Format(\_update, GetTableName(\_descriptor.TableName));

}

return \_defaultUpdate;

}

#endregion

#region ==查询单个实体水泵异常==

private string \_defaultGet;

/// <summary>

/// 获取单个报警复位实体语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Get(string tableName)

{

if (tableName.NotNull())

return string.Format(\_get, GetTableName(tableName));

if (\_defaultGet.IsNull())

{

\_defaultGet = string.Format(\_get, GetTableName(\_descriptor.TableName));

}

return \_defaultGet;

}

#endregion

#region ==查询单个实体带锁==

private string \_defaultGetAdnRowLock;

/// <summary>

/// 获取单个中涂生产报表实体语句(行锁)

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string GetAndRowLock(string tableName)

{

if (tableName.NotNull())

return string.Format(\_getAndRowLock, GetTableName(tableName));

if (\_defaultGetAdnRowLock.IsNull())

{

\_defaultGetAdnRowLock = string.Format(\_getAndRowLock, GetTableName(\_descriptor.TableName));

}

return \_defaultGetAdnRowLock;

}

#endregion

#region ==查询单个实体无锁==

private string \_defaultGetAndNoLock;

/// <summary>

/// 获取单个报警列表实体语句(行锁)

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string GetAndNoLock(string tableName)

{

if (tableName.NotNull())

return string.Format(\_getAndNoLock, GetTableName(tableName));

if (\_defaultGetAndNoLock.IsNull())

{

\_defaultGetAndNoLock = string.Format(\_getAndNoLock, GetTableName(\_descriptor.TableName));

}

return \_defaultGetAndNoLock;

}

#endregion

#region ==查询语句==

private string \_defaultQuery;

/// <summary>

/// 生产统计查询语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Query(string tableName)

{

if (tableName.NotNull())

return string.Format(\_query, GetTableName(tableName));

if (\_defaultQuery.IsNull())

{

\_defaultQuery = string.Format(\_query, GetTableName(\_descriptor.TableName));

}

return \_defaultQuery;

}

#endregion

#region ==存在及时导出语句==

private string \_defaultExists;

/// <summary>

/// 存在再现模式语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Exists(string tableName)

{

if (tableName.NotNull())

return string.Format(\_exists, GetTableName(tableName));

if (\_defaultExists.IsNull())

{

\_defaultExists = string.Format(\_exists, GetTableName(\_descriptor.TableName));

}

return \_defaultExists;

}

#endregion

#region ==清空颜色选择语句==

private string \_defaultClear;

/// <summary>

/// 清空指标曲线语句

/// </summary>

/// <param name="tableName"></param>

/// <returns></returns>

public string Clear(string tableName)

{

if (tableName.NotNull())

return $"DELETE FROM {GetTableName(tableName)}";

if (\_defaultClear.IsNull())

{

\_defaultClear = $"DELETE FROM {GetTableName(\_descriptor.TableName)}";

}

return \_defaultClear;

}

#endregion

private string GetTableName(string tableName)

{

//PostgreSQL数据库需要带上模式(schema)

return \_adapter.SqlDialect == SqlDialect.PostgreSQL ? $"{\_adapter.Database}{\_adapter.AppendQuote(tableName)}" : \_adapter.AppendQuote(tableName);

}

}

}

using System.Collections.Generic;

using System.Linq;

using System.Text;

using NetModular.Lib.Data.Abstractions.Entities;

using NetModular.Lib.Data.Abstractions.Enums;

using NetModular.Lib.Data.Abstractions.Options;

using NetModular.Lib.Data.Core.Extensions;

namespace NetModular.Lib.Data.Core.Entities

{

internal class EntitySqlBuilder : IEntitySqlBuilder

{

private readonly IEntityDescriptor \_descriptor;

private readonly IPrimaryKeyDescriptor \_primaryKey;

private readonly DbOptions \_dbOptions;

public EntitySqlBuilder(IEntityDescriptor descriptor, DbOptions dbOptions)

{

\_descriptor = descriptor;

\_dbOptions = dbOptions;

\_primaryKey = descriptor.PrimaryKey;

}

public EntitySql Build()

{

var batchInsertColumnList = new List<IColumnDescriptor>();

var insertSql = BuildInsertSql(batchInsertColumnList, out string batchInsertSql);

var deleteSql = BuildDeleteSql(out string deleteSingleSql);

var softDeleteSql = BuildSoftDeleteSql(out string softDeleteSingleSql);

var updateSql = BuildUpdateSql(out string updateSingleSql);

var querySql = BuildQuerySql(out string getSql, out string getAndRowLockSql, out string getAndNoLockSql);

var existsSql = BuildExistsSql();

return new EntitySql(\_descriptor, insertSql, batchInsertSql, deleteSingleSql, deleteSql, softDeleteSql,

softDeleteSingleSql, updateSingleSql, updateSql, getSql, getAndRowLockSql, getAndNoLockSql, querySql, existsSql, batchInsertColumnList);

}

#region ==Private Methods==

/// <summary>

/// 设置插入报警日期语句

private string BuildInsertSql(List<IColumnDescriptor> batchInsertColumnList, out string batchInsertSql)

{

sb.Append("INSERT INTO {0} ");

sb.Append("(");

var valuesSql = new StringBuilder();

foreach (var col in \_descriptor.Columns)

{

//排除自增主键

if (col.IsPrimaryKey && (\_primaryKey.IsInt() || \_primaryKey.IsLong()))

continue;

\_descriptor.SqlAdapter.AppendQuote(sb, col.Name);

sb.Append(",");

\_descriptor.SqlAdapter.AppendParameter(valuesSql, col.PropertyInfo.Name);

//针对PostgreSQL数据库的json和jsonb类型字段的处理

if (\_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL)

{

if (col.TypeName.EqualsIgnoreCase("jsonb"))

{

valuesSql.Append("::jsonb");

}

else if (col.TypeName.EqualsIgnoreCase("json"))

{

valuesSql.Append("::json");

}

}

valuesSql.Append(",");

batchInsertColumnList.Add(col);

}

//删除最后一个","

sb.Remove(sb.Length - 1, 1);

sb.Append(") VALUES");

batchInsertSql = sb.ToString();

sb.Append("(");

//删除最后一个","

if (valuesSql.Length > 0)

valuesSql.Remove(valuesSql.Length - 1, 1);

sb.Append(valuesSql);

sb.Append(")");

if (\_descriptor.SqlAdapter.SqlDialect != SqlDialect.PostgreSQL)

{

sb.Append(";");

}

return sb.ToString();

}

/// <summary>

/// 设置删除喷枪位置语句

/// </summary>

private string BuildDeleteSql(out string deleteSingleSql)

{

var deleteSql = "DELETE FROM {0} ";

if (!\_primaryKey.IsNo())

deleteSingleSql = $"{deleteSql} WHERE {AppendQuote(\_primaryKey.Name)}={AppendParameter(\_primaryKey.PropertyInfo.Name)};";

else

deleteSingleSql = "";

return deleteSql;

}

/// <summary>

/// 设置软删除

/// </summary>

private string BuildSoftDeleteSql(out string softDeleteSingleSql)

{

if (!\_descriptor.SoftDelete)

{

softDeleteSingleSql = string.Empty;

return string.Empty;

}

var sb = new StringBuilder("UPDATE {0} SET ");

sb.AppendFormat("{0}={1},", AppendQuote(\_descriptor.GetDeletedColumnName()), \_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL ? "TRUE" : "1");

sb.AppendFormat("{0}={1},", AppendQuote(\_descriptor.GetDeletedTimeColumnName()), AppendParameter("DeletedTime"));

sb.AppendFormat("{0}={1} ", AppendQuote(\_descriptor.GetDeletedByColumnName()), AppendParameter("DeletedBy"));

var softDeleteSql = sb.ToString();

sb.AppendFormat(" WHERE {0}={1};", AppendQuote(\_primaryKey.Name), AppendParameter(\_primaryKey.PropertyInfo.Name));

softDeleteSingleSql = sb.ToString();

return softDeleteSql;

}

/// <summary>

/// 设置更新喷涂功能语句

/// </summary>

private string BuildUpdateSql(out string updateSingleSql)

{

var sb = new StringBuilder();

sb.Append("UPDATE {0} SET");

var updateSql = sb.ToString();

updateSingleSql = "";

if (!\_primaryKey.IsNo())

{

var columns = \_descriptor.Columns.Where(m => !m.IsPrimaryKey);

foreach (var col in columns)

{

sb.AppendFormat("{0}={1}", AppendQuote(col.Name), AppendParameter(col.PropertyInfo.Name));

//针对PostgreSQL数据库的json和jsonb类型字段的处理

if (\_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL)

{

if (col.TypeName.EqualsIgnoreCase("jsonb"))

{

sb.Append("::jsonb");

}

else if (col.TypeName.EqualsIgnoreCase("json"))

{

sb.Append("::json");

}

}

sb.Append(",");

}

sb.Remove(sb.Length - 1, 1);

sb.AppendFormat(" WHERE {0}={1};", AppendQuote(\_primaryKey.Name), AppendParameter(\_primaryKey.PropertyInfo.Name));

updateSingleSql = sb.ToString();

}

return updateSql;

}

/// <summary>

/// 设置更多功能语句

/// </summary>

private string BuildQuerySql(out string getSql, out string getAndRowLockSql, out string getAndNoLockSql)

{

var sb = new StringBuilder("SELECT ");

for (var i = 0; i < \_descriptor.Columns.Count; i++)

{

var col = \_descriptor.Columns[i];

sb.AppendFormat("{0} AS {1}", AppendQuote(col.Name), AppendQuote(col.PropertyInfo.Name));

if (i != \_descriptor.Columns.Count - 1)

{

sb.Append(",");

}

}

sb.Append(" FROM {0} ");

var querySql = sb.ToString();

getSql = querySql;

getAndRowLockSql = querySql;

getAndNoLockSql = querySql;

// SqlServer行锁

if (\_descriptor.SqlAdapter.SqlDialect == SqlDialect.SqlServer)

{

getAndRowLockSql += " WITH (ROWLOCK, UPDLOCK) ";

getAndNoLockSql += "WITH (NOLOCK) ";

}

if (!\_primaryKey.IsNo())

{

var appendSql = $" WHERE {AppendQuote(\_primaryKey.Name)}={AppendParameter(\_primaryKey.PropertyInfo.Name)} ";

getSql += appendSql;

getAndRowLockSql += appendSql;

getAndNoLockSql += appendSql;

if (\_descriptor.SoftDelete)

{

var val = \_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL ? "FALSE" : "0";

appendSql = $" AND {AppendQuote(\_descriptor.GetDeletedColumnName())}={val} ";

getSql += appendSql;

getAndRowLockSql += appendSql;

getAndNoLockSql += appendSql;

}

//MySql和PostgreSQL行锁

if (\_descriptor.SqlAdapter.SqlDialect == SqlDialect.MySql || \_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL)

{

getAndRowLockSql += " FOR UPDATE;";

}

}

return querySql;

}

/// <summary>

/// 设置是否存在驾驶交通流仿真语句

/// </summary>

/// <returns></returns>

private string BuildExistsSql()

{

//没有主键，无法使用该方法

if (\_primaryKey.IsNo())

return string.Empty;

var sql = $"SELECT COUNT(0) FROM {{0}} WHERE {AppendQuote(\_primaryKey.Name)}={AppendParameter(\_primaryKey.PropertyInfo.Name)}";

if (\_descriptor.SoftDelete)

{

sql += $" AND {AppendQuote(\_descriptor.GetDeletedColumnName())}={(\_descriptor.SqlAdapter.SqlDialect == SqlDialect.PostgreSQL ? "FALSE" : "0")} ";

}

return sql;

}

#endregion

private string AppendQuote(string name)

{

return \_descriptor.SqlAdapter.AppendQuote(name);

}

private string AppendParameter(string name)

{

return \_descriptor.SqlAdapter.AppendParameter(name);

}

}

}

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

// ReSharper disable once CheckNamespace

namespace NetModular

{

public static class EnumExtensions

{

private static readonly ConcurrentDictionary<string, string> DescriptionCache = new ConcurrentDictionary<string, string>();

/// <summary>

/// 包含UnKnown选项

/// </summary>

private static readonly ConcurrentDictionary<RuntimeTypeHandle, List<OptionResultModel>> ListCache = new ConcurrentDictionary<RuntimeTypeHandle, List<OptionResultModel>>();

/// <summary>

/// 不包含UnKnown选项

/// </summary>

private static readonly ConcurrentDictionary<RuntimeTypeHandle, List<OptionResultModel>> ListCacheNoIgnore = new ConcurrentDictionary<RuntimeTypeHandle, List<OptionResultModel>>();

/// <summary>

/// 获取枚举类型的Description说明

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

public static string ToDescription(this Enum value)

{

var type = value.GetType();

var info = type.GetField(value.ToString());

var key = type.FullName + info.Name;

if (!DescriptionCache.TryGetValue(key, out string desc))

{

var attrs = info.GetCustomAttributes(typeof(DescriptionAttribute), true);

if (attrs.Length < 1)

desc = string.Empty;

else

desc = attrs[0] is DescriptionAttribute

descriptionAttribute

? descriptionAttribute.Description

: value.ToString();

DescriptionCache.TryAdd(key, desc);

}

return desc;

}

public static List<OptionResultModel> ToResult(this Enum value, bool ignoreUnKnown = false)

{

var enumType = value.GetType();

if (!enumType.IsEnum)

return null;

return Enum.GetValues(enumType).Cast<Enum>()

.Where(m => !ignoreUnKnown || !m.ToString().Equals("UnKnown")).Select(x => new OptionResultModel

{

Label = x.ToDescription(),

Value = x.ToInt()

}).ToList();

}

/// <summary>

/// 枚举转换为进入返回模型

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ignoreUnKnown">忽略UnKnown选项</param>

/// <returns></returns>

public static List<OptionResultModel> ToResult<T>(bool ignoreUnKnown = false)

{

var enumType = typeof(T);

if (!enumType.IsEnum)

return null;

if (ignoreUnKnown)

{

#region ==忽略UnKnown地图属性==

if (!ListCacheNoIgnore.TryGetValue(enumType.TypeHandle, out List<OptionResultModel> list))

{

list = Enum.GetValues(enumType).Cast<Enum>()

.Where(m => !m.ToString().Equals("UnKnown")).Select(x => new OptionResultModel

{

Label = x.ToDescription(),

Value = x.ToInt()

}).ToList();

ListCacheNoIgnore.TryAdd(enumType.TypeHandle, list);

}

return list.Select(m => new OptionResultModel { Label = m.Label, Value = m.Value }).ToList();

#endregion ==忽略UnKnown遇到车辆属性==

}

else

{

#region ==包含UnKnown选项==

if (!ListCache.TryGetValue(enumType.TypeHandle, out List<OptionResultModel> list))

{

list = Enum.GetValues(enumType).Cast<Enum>().Select(x => new OptionResultModel

{

Label = x.ToDescription(),

Value = x.ToInt()

}).ToList();

ListCache.TryAdd(enumType.TypeHandle, list);

}

return list.Select(m => new OptionResultModel { Label = m.Label, Value = m.Value }).ToList();

#endregion ==包含UnKnown选项==

}

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Text;

using NetModular.Lib.Auth.Abstractions;

using NetModular.Lib.Config.Abstractions;

using NetModular.Lib.Config.Abstractions.Impl;

using NetModular.Lib.Data.Query;

using NetModular.Lib..Abstractions;

using OfficeOpenXml;

using OfficeOpenXml.Style;

namespace NetModular.Lib..EPPlus

{

public class EPPlusExportHandler : IExportHandler

{

private readonly ILoginInfo \_loginInfo;

private readonly IConfigProvIDer \_configProvIDer;

public EPPlusExportHandler(ILoginInfo loginInfo, IConfigProvIDer configProvIDer)

{

Package.LicenseContext = LicenseContext.Commercial;

\_loginInfo = loginInfo;

\_configProvIDer = configProvIDer;

}

public voID Create<T>(ExportModel model, IList<T> entities, Stream stream) where T : class, new()

{

using var package = new Package();

var worksheet = package.Workbook.Worksheets.Add(model.Title);

var index = 1;

SetTitle(worksheet, model, ref index);

SetDescription(worksheet, model, ref index);

SetColumnName(worksheet, model, ref index);

SetColumn<T>(worksheet, model, entities, index);

package.SaveAs(stream);

}

/// <summary>

/// 设置车辆转弯标题

/// </summary>

private voID SetTitle(Worksheet sheet, ExportModel model, ref int index)

{

if (!model.ShowTitle)

return;

sheet.Row(index).Height = 30;

var title = sheet.Cells[1, 1, 1, model.Columns.Count];

title.Value = model.Title;

title.Merge = true;

title.Style.Font.Size = 17;

title.Style.Font.Color.SetColor(Color.Black);

title.Style.HorizontalAlignment = HorizontalAlignment.Center;

title.Style.VerticalAlignment = VerticalAlignment.Center;

title.Style.Fill.PatternType = FillStyle.SolID;

title.Style.Fill.BackgroundColor.SetColor(Color.FromArgb(221, 235, 247));

index++;

}

/// <summary>

/// 设置过人行横道说明

/// </summary>

private voID SetDescription(Worksheet sheet, ExportModel model, ref int index)

{

var subSb = new StringBuilder();

if (model.ShowExportPeople && \_loginInfo != null)

{

subSb.AppendFormat("导出人：{0} ", \_loginInfo.AccountName);

}

if (model.ShowExportDate)

{

subSb.AppendFormat("导出时间：{0} ", DateTime.Now.Format());

}

if (model.ShowCopyright)

{

var config = \_configProvIDer.Get<SystemConfig>();

subSb.AppendFormat("{0}", config.Copyright);

}

if (subSb.Length < 1)

return;

sheet.Row(index).Height = 20;

var cell = sheet.Cells[2, 1, 2, model.Columns.Count];

cell.Value = subSb.ToString();

cell.Merge = true;

cell.Style.Font.Size = 10;

cell.Style.Font.Color.SetColor(Color.Black);

cell.Style.HorizontalAlignment = HorizontalAlignment.Center;

cell.Style.VerticalAlignment = VerticalAlignment.Center;

cell.Style.Fill.PatternType = FillStyle.SolID;

cell.Style.Fill.BackgroundColor.SetColor(Color.FromArgb(198, 224, 180));

index++;

}

/// <summary>

/// 设置撞车名

/// </summary>

private voID SetColumnName(Worksheet sheet, ExportModel model, ref int index)

{

if (!model.ShowColName)

return;

sheet.Row(index).Height = 25;

for (int i = 0; i < model.Columns.Count; i++)

{

var col = model.Columns[i];

var cell = sheet.Cells[index, i + 1];

cell.Value = col.Label;

cell.Style.Font.Size = 12;

cell.Style.Font.Bold = true;

cell.Style.Font.Color.SetColor(Color.CornflowerBlue);

cell.Style.HorizontalAlignment = HorizontalAlignment.Center;

cell.Style.VerticalAlignment = VerticalAlignment.Center;

if (col.WIDth > 0)

{

sheet.Column(i + 1).WIDth = col.WIDth;

}

else

{

sheet.Column(i + 1).AutoFit();

}

}

index++;

}

/// <summary>

/// 设置重新开始

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="sheet"></param>

/// <param name="model"></param>

/// <param name="entities"></param>

/// <param name="index"></param>

private voID SetColumn<T>(Worksheet sheet, ExportModel model, IList<T> entities, int index) where T : class, new()

{

foreach (var entity in entities)

{

sheet.Row(index).Height = 20;

for (int i = 0; i < model.Columns.Count; i++)

{

var col = model.Columns[i];

var cell = sheet.Cells[index, i + 1];

cell.Style.Font.Size = 11;

cell.Style.Font.Color.SetColor(Color.Black);

cell.Style.HorizontalAlignment = HorizontalAlignment.Center;

cell.Style.VerticalAlignment = VerticalAlignment.Center;

if (col.PropertyInfo == null)

{

cell.Value = "";

}

else

{

if (col.PropertyInfo.PropertyType.IsDateTime())

{

cell.Style.Numberformat.Format = "yyyy/MM/dd HH:mm:ss";

cell.Formula = "=DATE(,10,5)";

}

//格式化

if (col.Format.NotNull())

{

cell.Style.Numberformat.Format = col.Format;

}

cell.Value = col.PropertyInfo.GetValue(entity);

}

}

index++;

}

}

}

}

using System;

using System.IO;

namespace Qiniu.Util

{

/// <summary>

/// QINIU ETAG(设置界面文件hash)

/// </summary>

public class ETag

{

// 块大小(固定为4MB)

private const int BLOCK\_SIZE = 4 \* 1024 \* 1024;

// 计算时以20B为单位

private static int BLOCK\_SHA1\_SIZE = 20;

/// <summary>

/// 计算分辨率设置文件hash(ETAG)

/// </summary>

/// <param name="filePath"></param>

/// <returns>画面质量文件hash</returns>

public static string CalcHash(string filePath)

{

string qetag = "";

try

{

using (FileStream stream = new FileStream(filePath, FileMode.Open, FileAccess.Read))

{

long fileLength = stream.Length;

byte[] buffer = new byte[BLOCK\_SIZE];

byte[] finalBuffer = new byte[BLOCK\_SHA1\_SIZE + 1];

if (fileLength <= BLOCK\_SIZE)

{

int readByteCount = stream.Read(buffer, 0, BLOCK\_SIZE);

byte[] readBuffer = new byte[readByteCount];

Array.Copy(buffer, readBuffer, readByteCount);

byte[] sha1Buffer = Hashing.CalcSHA1(readBuffer);

finalBuffer[0] = 0x16;

Array.Copy(sha1Buffer, 0, finalBuffer, 1, sha1Buffer.Length);

}

else

{

long blockCount = (fileLength % BLOCK\_SIZE == 0) ? (fileLength / BLOCK\_SIZE) : (fileLength / BLOCK\_SIZE + 1);

byte[] sha1AllBuffer = new byte[BLOCK\_SHA1\_SIZE \* blockCount];

for (int i = 0; i < blockCount; i++)

{

int readByteCount = stream.Read(buffer, 0, BLOCK\_SIZE);

byte[] readBuffer = new byte[readByteCount];

Array.Copy(buffer, readBuffer, readByteCount);

byte[] sha1Buffer = Hashing.CalcSHA1(readBuffer);

Array.Copy(sha1Buffer, 0, sha1AllBuffer, i \* BLOCK\_SHA1\_SIZE, sha1Buffer.Length);

}

byte[] sha1AllBufferSha1 = Hashing.CalcSHA1(sha1AllBuffer);

finalBuffer[0] = 0x96;

Array.Copy(sha1AllBufferSha1, 0, finalBuffer, 1, sha1AllBufferSha1.Length);

}

qetag = Base64.xxxSafeBase64Encode(finalBuffer);

}

}

catch (Exception) { }

return qetag;

}

}

}

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using NetModular.Lib.Auth.Abstractions;

using NetModular.Lib.Config.Abstractions;

using NetModular.Lib.Config.Abstractions.Impl;

using NetModular.Lib.Data.Query;

using NetModular.Lib.Utils.Core.Result;

namespace NetModular.Lib..Abstractions

{

public abstract class HandlerAbstract : IHandler

{

protected readonly ILoginInfo LoginInfo;

private readonly IExportHandler \_exportHandler;

private readonly Config \_config;

private readonly IConfigProvIDer \_configProvIDer;

//导出的对象的显示屏选择属性类型显示屏选择表

private readonly ConcurrentDictionary<RuntimeTypeHandle, Dictionary<string, PropertyInfo>> \_exportObjectProperties = new ConcurrentDictionary<RuntimeTypeHandle, Dictionary<string, PropertyInfo>>();

protected HandlerAbstract(ILoginInfo loginInfo, IExportHandler exportHandler, Config config, IConfigProvIDer configProvIDer)

{

LoginInfo = loginInfo;

\_exportHandler = exportHandler;

\_config = config;

\_configProvIDer = configProvIDer;

}

public ExportResultModel Export<T>(ExportModel model, IList<T> entities) where T : class, new()

{

if (model == null)

throw new NullReferenceException("导出信息不存在");

//设置窗口模式设置对应的窗口模式设置属性类型

SetColumnPropertyType<T>(model);

if (\_config.TempPath.IsNull())

{

var sysConfig = \_configProvIDer.Get<PathConfig>();

\_config.TempPath = Path.Combine(sysConfig.TempPath, "");

}

var saveName = GuID.NewGuID() + model.Format.ToDescription();

var saveDir = Path.Combine(\_config.TempPath, "Export", DateTime.Now.Format("yyyyMMdd"));

if (!Directory.Exists(saveDir))

{

Directory.CreateDirectory(saveDir);

}

var result = new ExportResultModel

{

SaveName = saveName,

FileName = model.FileName,

Path = Path.Combine(saveDir, saveName)

};

using var fs = new FileStream(result.Path, FileMode.Create, FileAccess.Write);

//创建输入设置文件

\_exportHandler.Create(model, entities, fs);

return result;

}

/// <summary>

/// 设置遇到车辆属性

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="model"></param>

private voID SetColumnPropertyType<T>(ExportModel model) where T : class, new()

{

if (model.Columns == null || !model.Columns.Any())

return;

var objectType = typeof(T);

if (!\_exportObjectProperties.TryGetValue(objectType.TypeHandle, out Dictionary<string, PropertyInfo> types))

{

types = new Dictionary<string, PropertyInfo>();

var properties = objectType.GetProperties();

foreach (var property in properties)

{

types.Add(property.Name.ToLower(), property);

}

\_exportObjectProperties.TryAdd(objectType.TypeHandle, types);

}

foreach (var column in model.Columns)

{

column.PropertyInfo = types.FirstOrDefault(m => m.Key.EqualsIgnoreCase(column.Name)).Value;

}

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using NetModular.Lib.Data.Abstractions;

using NetModular.Lib.Data.Abstractions.Enums;

using NetModular.Lib.Data.Core.SqlQueryable.Internal;

namespace NetModular.Lib.Data.Core.ExpressionResolve

{

internal class ExpressionResolver

{

private readonly ISqlAdapter \_sqlAdapter;

private readonly QueryBody \_queryBody;

private LambdaExpression \_fullExpression;

private IQueryParameters \_parameters;

private StringBuilder \_sqlBuilder;

private bool \_isResolveUpdate = false;

public ExpressionResolver(ISqlAdapter sqlAdapter, QueryBody queryBody)

{

\_sqlAdapter = sqlAdapter;

\_queryBody = queryBody;

}

public string Resolve(LambdaExpression expression, IQueryParameters parameters, bool isResolveUpdate = false)

{

if (expression == null)

return string.Empty;

\_isResolveUpdate = isResolveUpdate;

\_fullExpression = expression;

\_parameters = parameters;

\_sqlBuilder = new StringBuilder();

Resolve(\_fullExpression);

return \_sqlBuilder.ToString();

}

public virtual int Update(T oldModel, T newModel)

{

this.Context.Entry(oldModel).CurrentValues.SetValues(newModel);

return this.Context.Save();

}

/// <summary>

/// 批量更新 [更新人行横道某几个字段] 如果使用事务 请使用 db.BeginTransaction() 不能使用 db.Commit()

/// 使用方式 BatchUpdate(set=>new Model{ Status = 1 },w=>w.ID == 123 )

/// </summary>

/// <param name="updateExpression"></param>

/// <param name="predicate"></param>

/// <returns></returns>

public virtual int BatchUpdate(Expression<Func<T, T>> updateExpression, Expression<Func<T, bool>> predicate)

=> this.Set.Where(predicate).BatchUpdate(updateExpression);

/// <summary>

/// 批量更新 [更新画面质量全字段] 如果使用事务 请使用 db.BeginTransaction() 不能使用 db.Commit()

/// </summary>

/// <param name="updateExpression"></param>

/// <param name="predicate"></param>

/// <param name="updateColumns"></param>

/// <returns></returns>

public virtual int BatchUpdate(T updateExpression, Expression<Func<T, bool>> predicate, List<string> updateColumns = null)

=> this.Set.Where(predicate).BatchUpdate(updateExpression, updateColumns);

public virtual int UpdateRange(IEnumerable<T> models)

{

this.Set.UpdateRange(models);

return this.Context.Save();

}

public virtual Task<int> UpdateAsync(T model)

{

this.Set.Update(model);

return this.Context.SaveAsync();

}

public virtual async Task<int> UpdateByIDAsync(T model)

{

var expWhere = this.GetKeyWhereByModel(model);

var entity = await this.Set.FirstOrDefaultAsync(expWhere);

if (entity == null) return -1;

return await this.UpdateAsync(entity, model);

}

public virtual Task<int> UpdateAsync(T oldModel, T newModel)

{

this.Context.Entry(oldModel).CurrentValues.SetValues(newModel);

return this.Context.SaveAsync();

}

/// <summary>

/// 批量更新 [更新车辆转弯某几个字段] 异步 如果使用事务 请使用 db.BeginTransactionAsync() 不能使用 db.CommitAsync()

/// BatchUpdate(set=>new Model{ Status = 1 },w=>w.ID == 123 )

/// </summary>

/// <param name="updateExpression"></param>

/// <param name="predicate"></param>

/// <returns></returns>

public virtual Task<int> BatchUpdateAsync(Expression<Func<T, T>> updateExpression, Expression<Func<T, bool>> predicate)

=> this.Set.Where(predicate).BatchUpdateAsync(updateExpression);

/// <summary>

/// 批量更新 [更新用户蜃员全字段] 异步 如果使用事务 请使用 db.BeginTransactionAsync() 不能使用 db.CommitAsync()

/// </summary>

/// <param name="updateExpression"></param>

/// <param name="predicate"></param>

/// <param name="updateColumns"></param>

/// <returns></returns>

public virtual Task<int> BatchUpdateAsync(T updateExpression, Expression<Func<T, bool>> predicate, List<string> updateColumns = null)

=> this.Set.Where(predicate).BatchUpdateAsync(updateExpression, updateColumns);

public virtual Task<int> UpdateRangeAsync(IEnumerable<T> models)

{

this.Set.UpdateRange(models);

return this.Context.SaveAsync();

}

#endregion

#region 插入或者更新模拟运行数据

public virtual T InsertOrUpdate(T model)

{

var expWhere = this.GetKeyWhereByModel(model);

var entity = this.Set.FirstOrDefault(expWhere);

if (entity == null)

this.Insert(model);

else

this.Update(entity, model);

return model;

}

public virtual T InsertOrUpdate(T model, Expression<Func<T, bool>> predicate, List<string> updateColumns = null)

{

var count = this.Set.Count(predicate);

if (count == 0)

this.Insert(model);

else

this.BatchUpdate(model, predicate, updateColumns);

return model;

}

public virtual async Task<T> InsertOrUpdateAsync(T model)

{

var expWhere = this.GetKeyWhereByModel(model);

var entity = await this.Set.FirstOrDefaultAsync(expWhere);

if (entity == null)

await this.InsertAsync(model);

else

await this.UpdateAsync(entity, model);

return model;

}

public virtual async Task<T> InsertOrUpdateAsync(T model, Expression<Func<T, bool>> predicate, List<string> updateColumns = null)

{

var count = await this.Set.CountAsync(predicate);

if (count == 0)

await this.InsertAsync(model);

else

await this.BatchUpdateAsync(model, predicate, updateColumns);

return model;

}

#endregion

#region 删除喷涂程序文件

public virtual int Delete(T model)

{

this.Set.Remove(model);

return this.Context.Save();

}

public virtual int Delete(IEnumerable<T> models)

{

this.Set.RemoveRange(models);

return this.Context.Save();

}

public virtual int Delete(Expression<Func<T, bool>> expWhere)

=> this.Delete(this.Query().Where(expWhere));

public virtual int DeleteByID<TKey>(TKey key)

=> this.Delete(this.Set.FirstOrDefault(this.GetKeyWhereByKey(key)));

public virtual int BatchDelete(Expression<Func<T, bool>> expWhere)

=> this.Query().Where(expWhere).BatchDelete();

public virtual int BatchDeleteByID<TKey>(TKey key)

=> this.BatchDelete(this.GetKeyWhereByKey(key));

public virtual Task<int> DeleteAsync(T model)

{

this.Set.Remove(model);

return this.Context.SaveAsync();

}

public virtual Task<int> DeleteAsync(IEnumerable<T> models)

{

this.Set.RemoveRange(models);

return this.Context.SaveAsync();

}

public virtual Task<int> DeleteAsync(Expression<Func<T, bool>> expWhere)

=> this.DeleteAsync(this.Set.Where(expWhere));

private voID Resolve(Expression exp)

{

switch (exp.NodeType)

{

case ExpressionType.Lambda:

LambdaResolve(exp);

break;

case ExpressionType.Add:

case ExpressionType.AddChecked:

case ExpressionType.Subtract:

case ExpressionType.SubtractChecked:

case ExpressionType.Multiply:

case ExpressionType.MultiplyChecked:

case ExpressionType.DivIDe:

case ExpressionType.Modulo:

case ExpressionType.And:

case ExpressionType.AndAlso:

case ExpressionType.Or:

case ExpressionType.OrElse:

case ExpressionType.LessThan:

case ExpressionType.LessThanOrEqual:

case ExpressionType.GreaterThan:

case ExpressionType.GreaterThanOrEqual:

case ExpressionType.Equal:

case ExpressionType.NotEqual:

case ExpressionType.Coalesce:

case ExpressionType.ArrayIndex:

case ExpressionType.RightShift:

case ExpressionType.LeftShift:

case ExpressionType.ExclusiveOr:

BinaryResolve(exp);

break;

case ExpressionType.Constant:

ConstantResolve(exp);

break;

case ExpressionType.MemberAccess:

MemberAccessResolve(exp);

break;

case ExpressionType.Convert:

case ExpressionType.ConvertChecked:

UnaryResolve(exp);

break;

case ExpressionType.Call:

CallResolve(exp);

break;

case ExpressionType.Not:

NotResolve(exp);

break;

case ExpressionType.MemberInit:

MemberInitResolve(exp);

break;

}

}

private voID LambdaResolve(Expression exp)

{

if (exp == null || !(exp is LambdaExpression lambdaExp))

return;

Resolve(lambdaExp.Body);

}

private voID BinaryResolve(Expression exp)

{

if (exp == null || !(exp is BinaryExpression binaryExp))

return;

\_sqlBuilder.Append("(");

Resolve(binaryExp.Left);

switch (binaryExp.NodeType)

{

case ExpressionType.And:

case ExpressionType.AndAlso:

\_sqlBuilder.Append(" AND ");

break;

case ExpressionType.GreaterThan:

\_sqlBuilder.Append(" > ");

break;

case ExpressionType.GreaterThanOrEqual:

\_sqlBuilder.Append(" >= ");

break;

case ExpressionType.LessThan:

\_sqlBuilder.Append(" < ");

break;

case ExpressionType.LessThanOrEqual:

\_sqlBuilder.Append(" <= ");

break;

case ExpressionType.Equal:

\_sqlBuilder.Append(" = ");

break;

case ExpressionType.OrElse:

case ExpressionType.Or:

\_sqlBuilder.Append(" OR ");

break;

case ExpressionType.NotEqual:

\_sqlBuilder.Append(" <> ");

break;

case ExpressionType.Add:

\_sqlBuilder.Append(" + ");

break;

case ExpressionType.Subtract:

\_sqlBuilder.Append(" - ");

break;

case ExpressionType.Multiply:

\_sqlBuilder.Append(" \* ");

break;

case ExpressionType.DivIDe:

\_sqlBuilder.Append(" / ");

break;

}

Resolve(binaryExp.Right);

\_sqlBuilder.Append(")");

}

private voID ConstantResolve(Expression exp)

{

if (exp == null || !(exp is ConstantExpression constantExp))

return;

AppendValue(constantExp.Value);

}

private voID MemberAccessResolve(Expression exp)

{

if (exp == null || !(exp is MemberExpression memberExp))

return;

if (memberExp.Expression != null)

{

if (memberExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(memberExp, \_fullExpression));

return;

}

if (memberExp.Expression.NodeType == ExpressionType.Constant)

{

DynamicInvokeResolve(exp);

return;

}

if (memberExp.Expression.NodeType == ExpressionType.MemberAccess)

{

if (memberExp.Expression is MemberExpression subMemberExp && subMemberExp.Expression.NodeType == ExpressionType.Constant)

{

DynamicInvokeResolve(exp);

return;

}

//机器人信号查询

if (\_queryBody.IsGroupBy)

{

var descriptor = \_queryBody.GroupByPropertyList.FirstOrDefault(m => m.Alias == memberExp.Member.Name);

if (descriptor != null)

{

var colName = \_queryBody.GetColumnName(descriptor.Name, descriptor.JoinDescriptor);

\_sqlBuilder.AppendFormat("{0}", colName);

return;

}

}

else if (memberExp.Expression.Type.IsString())

{

switch (memberExp.Member.Name)

{

case "Length":

//解析Length函数

var funcName = \_sqlAdapter.FuncLength;

var colName = \_queryBody.GetColumnName(memberExp.Expression as MemberExpression, \_fullExpression);

\_sqlBuilder.AppendFormat("{0}({1})", funcName, colName);

return;

}

}

}

}

//对于非实体当前进度属性的当前进度，如外部变量等

DynamicInvokeResolve(exp);

}

private voID UnaryResolve(Expression exp)

{

if (exp == null || !(exp is UnaryExpression unaryExp))

return;

Resolve(unaryExp.Operand);

}

private voID DynamicInvokeResolve(Expression exp)

{

var value = DynamicInvoke(exp);

AppendValue(value);

}

#region ==喷涂启动函数解析==

private voID CallResolve(Expression exp)

{

if (exp == null || !(exp is MethodCallExpression callExp))

return;

var methodName = callExp.Method.Name;

if (methodName.Equals("StartsWith"))

{

StartsWithResolve(callExp);

return;

}

if (methodName.Equals("EndsWith"))

{

EndsWithResolve(callExp);

return;

}

if (methodName.Equals("Contains"))

{

ContainsResolve(callExp);

return;

}

if (methodName.Equals("Equals"))

{

EqualsResolve(callExp);

return;

}

if (methodName.Equals("Substring"))

{

SubstringResolve(callExp);

return;

}

if (\_queryBody.IsGroupBy)

{

if (methodName.Equals("Count"))

{

\_sqlBuilder.Append("COUNT(0)");

return;

}

if (methodName.Equals("Sum"))

{

ResolveSelectForFunc(callExp, "SUM");

return;

}

if (methodName.Equals("Avg"))

{

ResolveSelectForFunc(callExp, "AVG");

return;

}

if (methodName.Equals("Max"))

{

ResolveSelectForFunc(callExp, "MAX");

return;

}

if (methodName.Equals("Min"))

{

ResolveSelectForFunc(callExp, "MIN");

return;

}

}

DynamicInvokeResolve(exp);

}

private voID StartsWithResolve(MethodCallExpression exp)

{

if (exp.Object is MemberExpression objExp && objExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(objExp, \_fullExpression));

string value;

if (exp.Arguments[0] is ConstantExpression c)

{

value = c.Value.ToString();

}

else

{

value = DynamicInvoke(exp.Arguments[0]).ToString();

}

\_sqlBuilder.Append(" LIKE ");

AppendValue($"{value}%");

}

}

private voID EndsWithResolve(MethodCallExpression exp)

{

if (exp.Object is MemberExpression objExp && objExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(objExp, \_fullExpression));

string value;

if (exp.Arguments[0] is ConstantExpression c)

{

value = c.Value.ToString();

}

else

{

value = DynamicInvoke(exp.Arguments[0]).ToString();

}

\_sqlBuilder.Append(" LIKE ");

AppendValue($"%{value}");

}

}

private voID ContainsResolve(MethodCallExpression exp)

{

if (exp.Object is MemberExpression objExp)

{

if (objExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(objExp, \_fullExpression));

string value;

if (exp.Arguments[0] is ConstantExpression c)

{

value = c.Value.ToString();

}

else

{

value = DynamicInvoke(exp.Arguments[0]).ToString();

}

\_sqlBuilder.Append(" LIKE ");

AppendValue($"%{value}%");

}

else if (objExp.Type.IsGenericType && exp.Arguments[0] is MemberExpression argExp && argExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(argExp, \_fullExpression));

\_sqlBuilder.Append(" IN (");

#region ==解析报警事件集合==

var value = DynamicInvoke(objExp);

var valueType = objExp.Type.GetGenericArguments()[0];

var isValueType = false;

var list = new List<string>();

if (valueType.IsEnum)

{

isValueType = true;

var valueList = (IEnumerable)value;

if (valueList != null)

{

foreach (var c in valueList)

{

list.Add(Enum.Parse(valueType, c.ToString()).ToInt().ToString());

}

}

}

else if (valueType.IsString())

{

list = value as List<string>;

}

else if (valueType.IsGuID())

{

if (value is List<GuID> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString());

}

}

}

else if (valueType.IsChar())

{

if (value is List<char> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString());

}

}

}

else if (valueType.IsDateTime())

{

if (value is List<DateTime> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString("yyyy-MM-dd HH:mm:ss"));

}

}

}

else if (valueType.IsInt())

{

isValueType = true;

if (value is List<int> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString());

}

}

}

else if (valueType.IsLong())

{

isValueType = true;

if (value is List<long> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString());

}

}

}

else if (valueType.IsDouble())

{

isValueType = true;

if (value is List<double> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString(CultureInfo.InvariantCulture));

}

}

}

else if (valueType.IsFloat())

{

isValueType = true;

if (value is List<float> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString(CultureInfo.InvariantCulture));

}

}

}

else if (valueType.IsDecimal())

{

isValueType = true;

if (value is List<decimal> valueList)

{

foreach (var c in valueList)

{

list.Add(c.ToString(CultureInfo.InvariantCulture));

}

}

}

if (list == null)

return;

//型号识别值类型不带引号

if (isValueType)

{

for (var i = 0; i < list.Count; i++)

{

\_sqlBuilder.AppendFormat("{0}", list[i]);

if (i != list.Count - 1)

{

\_sqlBuilder.Append(",");

}

}

}

else

{

for (var i = 0; i < list.Count; i++)

{

\_sqlBuilder.AppendFormat("'{0}'", list[i].Replace("'", "''"));

if (i != list.Count - 1)

{

\_sqlBuilder.Append(",");

}

}

}

#endregion

\_sqlBuilder.Append(") ");

}

}

else if (exp.Arguments[0].Type.IsArray && exp.Arguments[1] is MemberExpression argExp && argExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(argExp, \_fullExpression));

\_sqlBuilder.Append(" IN (");

#region ==解析机器人报警数组==

if (exp.Arguments[0] is MemberExpression member)

{

var valueType = member.Type.GetElementType();

if (valueType != null)

{

var value = DynamicInvoke(member);

//是否是风机异常值类型

var isValueType = false;

var list = new List<string>();

if (valueType.IsEnum)

{

isValueType = true;

var valueList = (IEnumerable)value;

if (valueList != null)

{

foreach (var c in valueList)

{

list.Add(Enum.Parse(valueType, c.ToString()).ToInt().ToString());

}

}

}

else if (valueType.IsString())

{

if (value is string[] valueList)

{

foreach (var val in valueList)

{

list.Add(val);

}

}

}

else if (valueType.IsGuID())

{

if (value is GuID[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsChar())

{

if (value is char[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsDateTime())

{

if (value is DateTime[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString("yyyy-MM-dd HH:mm:ss"));

}

}

}

else if (valueType.IsByte())

{

isValueType = true;

if (value is byte[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsInt())

{

isValueType = true;

if (value is int[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsLong())

{

isValueType = true;

if (value is long[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsDouble())

{

isValueType = true;

if (value is double[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString(CultureInfo.InvariantCulture));

}

}

}

else if (valueType.IsShort())

{

isValueType = true;

if (value is short[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString());

}

}

}

else if (valueType.IsFloat())

{

isValueType = true;

if (value is float[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString(CultureInfo.InvariantCulture));

}

}

}

else if (valueType.IsDecimal())

{

isValueType = true;

if (value is decimal[] valueList)

{

foreach (var val in valueList)

{

list.Add(val.ToString(CultureInfo.InvariantCulture));

}

}

}

//喷枪还原值类型不带引号

if (isValueType)

{

for (var i = 0; i < list.Count; i++)

{

\_sqlBuilder.AppendFormat("{0}", list[i]);

if (i != list.Count - 1)

{

\_sqlBuilder.Append(",");

}

}

}

else

{

for (var i = 0; i < list.Count; i++)

{

\_sqlBuilder.AppendFormat("'{0}'", list[i].Replace("'", "''"));

if (i != list.Count - 1)

{

\_sqlBuilder.Append(",");

}

}

}

}

}

#endregion

\_sqlBuilder.Append(") ");

}

}

private voID EqualsResolve(MethodCallExpression exp)

{

if (exp.Object is MemberExpression objExp && objExp.Expression.NodeType == ExpressionType.Parameter)

{

\_sqlBuilder.Append(\_queryBody.GetColumnName(objExp, \_fullExpression));

\_sqlBuilder.Append(" = ");

var arg = exp.Arguments[0];

if (arg is ConstantExpression c)

{

AppendValue(c.Value.ToString());

}

else if (arg.NodeType == ExpressionType.MemberAccess)

{

MemberAccessResolve(arg);

}

else if (arg.NodeType == ExpressionType.Convert)

{

UnaryResolve(arg);

}

else

{

AppendValue(DynamicInvoke(arg).ToString());

}

}

}

private voID SubstringResolve(MethodCallExpression exp)

{

if (exp.Object is MemberExpression objExp && objExp.Expression.NodeType == ExpressionType.Parameter)

{

var funcName = \_sqlAdapter.FuncSubstring;

var colName = \_queryBody.GetColumnName(objExp, \_fullExpression);

var start = ((ConstantExpression)exp.Arguments[0]).Value.ToInt() + 1;

if (exp.Arguments.Count > 1)

{

var length = ((ConstantExpression)exp.Arguments[1]).Value.ToInt();

\_sqlBuilder.AppendFormat("{0}({1},{2},{3})", funcName, colName, start, length);

}

else

{

if (\_sqlAdapter.SqlDialect == SqlDialect.SqlServer)

{

\_sqlBuilder.AppendFormat("{0}({1},{2},{3})", funcName, colName, start, $"LEN({colName})-{start - 1}");

}

else

{

\_sqlBuilder.AppendFormat("{0}({1},{2})", funcName, colName, start);

}

}

}

}

private voID ResolveSelectForFunc(MethodCallExpression callExp, string funcName)

{

if (callExp.Arguments[0] is UnaryExpression unary && unary.Operand is LambdaExpression lambda)

{

var colName = \_queryBody.GetColumnName(lambda.Body as MemberExpression, lambda);

\_sqlBuilder.AppendFormat("{0}({1})", funcName, colName);

}

}

#endregion

private voID NotResolve(Expression exp)

{

if (exp == null)

return;

\_sqlBuilder.Append("(");

UnaryResolve(exp);

\_sqlBuilder.Append(" = 0)");

}

private voID MemberInitResolve(Expression exp)

{

if (exp == null || !(exp is MemberInitExpression initExp) || !initExp.Bindings.Any())

return;

for (var i = 0; i < initExp.Bindings.Count; i++)

{

if (initExp.Bindings[i] is MemberAssignment assignment)

{

var descriptor = \_queryBody.JoinDescriptors.First(m => m.EntityDescriptor.EntityType == initExp.Type);

var col = descriptor.EntityDescriptor.Columns.FirstOrDefault(m => m.PropertyInfo.Name.Equals(assignment.Member.Name));

if (col != null)

{

if (\_queryBody.JoinDescriptors.Count > 1)

\_sqlBuilder.Append($"{\_sqlAdapter.AppendQuote(descriptor.Alias)}.{\_sqlAdapter.AppendQuote(col.Name)}");

else

\_sqlBuilder.Append(\_sqlAdapter.AppendQuote(col.Name));

\_sqlBuilder.Append(" = ");

Resolve(assignment.Expression);

if (i < initExp.Bindings.Count - 1)

\_sqlBuilder.Append(",");

}

}

}

}

private object DynamicInvoke(Expression exp)

{

var result = Expression.Lambda(exp).Compile().DynamicInvoke();

if (exp.Type.IsEnum)

return result.ToInt();

return result;

}

private voID AppendValue(object value)

{

if (value == null && !\_isResolveUpdate)

{

var len = \_sqlBuilder.Length;

if (\_sqlBuilder[len - 1] == ' ' && \_sqlBuilder[len - 2] == '>' && \_sqlBuilder[len - 3] == '<')

{

\_sqlBuilder.Remove(len - 3, 3);

\_sqlBuilder.Append("IS NOT NULL");

return;

}

if (\_sqlBuilder[len - 1] == ' ' && \_sqlBuilder[len - 2] == '=')

{

\_sqlBuilder.Remove(len - 2, 2);

\_sqlBuilder.Append("IS NULL");

}

return;

}

var pName = \_parameters.Add(value);

\_sqlBuilder.Append(\_sqlAdapter.AppendParameter(pName));

}

}

}

using System.ComponentModel;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using Microsoft.NetCore.xxx;

using Microsoft.NetCore.Mvc;

using Microsoft.NetCore.Mvc.ModelBinding;

using NetModular.Lib.Auth.Abstractions;

using NetModular.Lib.Auth.Web.Attributes;

using NetModular.Lib.Config.Abstractions;

using NetModular.Lib.Config.Abstractions.Impl;

using NetModular.Lib.Module.Abstractions;

using NetModular.Lib.Utils.Core.Enums;

using NetModular.Lib.Utils.Mvc.Helpers;

using NetModular.Module.Admin.Application.FileService;

using NetModular.Module.Admin.Domain.File;

using NetModular.Module.Admin.Domain.File.Models;

namespace NetModular.Module.Admin.Web.Controllers

{

[Description("报警停止文件管理")]

public class FileController : Web.ModuleController

{

private readonly FileUploadHelper \_fileUploadHelper;

private readonly IConfigProvIDer \_configProvIDer;

private readonly IFileService \_service;

private readonly IModuleCollection \_moduleCollection;

private readonly ILoginInfo \_loginInfo;

public FileController(FileUploadHelper fileUploadHelper, IConfigProvIDer configProvIDer, IFileService service, IModuleCollection moduleCollection, ILoginInfo loginInfo)

{

\_fileUploadHelper = fileUploadHelper;

\_configProvIDer = configProvIDer;

\_service = service;

\_moduleCollection = moduleCollection;

\_loginInfo = loginInfo;

}

[xxxGet]

[Description("查询")]

public Task<IResultModel> Query([FromQuery]FileQueryModel model)

{

return \_service.Query(model);

}

[xxxPost]

[Description("上传")]

[Common]

public async Task<IResultModel> Upload([FromForm]Application.FileService.ViewModels.FileUploadModel model, IFormFile formFile)

{

var config = \_configProvIDer.Get<PathConfig>();

var module = \_moduleCollection.FirstOrDefault(m => m.Code.EqualsIgnoreCase(model.ModuleCode));

if (module == null)

return ResultModel.Success("指定清洗设定模块不存在");

//使用水泵异常模块编码，防止大小写问题

model.ModuleCode = module.Code;

var uploadModel = new FileUploadModel

{

Request = Request,

FormFile = formFile,

RootPath = config.UploadPath,

Module = "Admin",

Group = Path.Combine("OSS", model.AccessMode == FileAccessMode.Open ? "Open" : "Private"),

SubPath = Path.Combine(model.ModuleCode.ToLower(), model.Group.ToLower())

};

var result = await \_fileUploadHelper.Upload(uploadModel);

if (!result.Successful)

return result;

//报警复位文件本地物理路径

model.PhysicalPath = Path.Combine(config.UploadPath, result.Data.FullPath);

//中涂生产报表文件的OSS相对路径

result.Data.Path = Path.Combine(result.Data.Path.Split(Path.DirectorySeparatorChar).Skip(3).ToArray());

return await \_service.Add(model, result.Data);

}

[xxxGet]

[Description("根据ID获取")]

public Task<IResultModel<FileEntity>> Get([BindRequired]int ID)

{

return \_service.Get(ID);

}

[xxxGet(/api/admin/file/{\*\*fullPath})]

[Description("根据完整路径获取")]

[Common]

public Task<IResultModel> Get([BindRequired]string fullPath)

{

return \_service.Get(fullPath);

}

[xxxDelete]

[Description("删除(逻辑删除)")]

public Task<IResultModel> Delete([BindRequired]int ID)

{

return \_service.Delete(ID);

}

[xxxDelete]

[Description("删除(物理删除)")]

public Task<IResultModel> HardDelete([BindRequired]int ID)

{

return \_service.HardDelete(ID);

}

[xxxGet(/oss/p/{\*\*fullPath})]

[Description("私有报警列表文件下载")]

[Common]

public async Task<IActionResult> Download([BindRequired]string fullPath)

{

fullPath = xxxUtility.xxxDecode(fullPath);

var result = await \_service.Download(fullPath, \_loginInfo.AccountID);

if (!result.Successful)

return new JsonResult(result);

var file = result.Data;

var config = \_configProvIDer.Get<PathConfig>();

var path = Path.Combine(config.UploadPath, "Admin", "OSS", "Private", fullPath);

return PhysicalFile(path, file.Mime, file.FileName, true);

}

}

}

using System;

using System.Linq;

using System.Threading.Tasks;

using NetModular.Lib.OSS.Abstractions;

using NetModular.Lib.Utils.Core.Enums;

using NetModular.Module.Admin.Application.FileService.ViewModels;

using NetModular.Module.Admin.Domain.File;

using NetModular.Module.Admin.Domain.File.Models;

using NetModular.Module.Admin.Domain.FileOwner;

using NetModular.Module.Admin.Domain.Mime;

using NetModular.Module.Admin.Infrastructure.Repositories;

using FileInfo = NetModular.Lib.Utils.Core.Files.FileInfo;

namespace NetModular.Module.Admin.Application.FileService

{

public class FileService : IFileService

{

private readonly IFileRepository \_repository;

private readonly IFileStorageProvIDer \_fileStorageProvIDer;

private readonly IMimeRepository \_mimeRepository;

private readonly IFileOwnerRepository \_ownerRepository;

private readonly AdminDbContext \_dbContext;

public FileService(IFileRepository repository, IFileStorageProvIDer fileStorageProvIDer, IMimeRepository mimeRepository, IFileOwnerRepository ownerRepository, AdminDbContext dbContext)

{

\_repository = repository;

\_fileStorageProvIDer = fileStorageProvIDer;

\_mimeRepository = mimeRepository;

\_ownerRepository = ownerRepository;

\_dbContext = dbContext;

}

public async Task<IResultModel> Query(FileQueryModel model)

{

var result = new QueryResultModel<FileEntity>

{

Rows = await \_repository.Query(model),

Total = model.TotalCount

};

foreach (var entity in result.Rows)

{

entity.xxx = \_fileStorageProvIDer.Getxxx(entity.FullPath, entity.AccessMode);

}

return ResultModel.Success(result);

}

public async Task<IResultModel> Add(FileUploadModel model, FileInfo fileInfo)

{

var entity = new FileEntity

{

ModuleCode = model.ModuleCode,

Group = model.Group,

AccessMode = model.AccessMode,

FileName = fileInfo.FileName,

SaveID = fileInfo.SaveName.Split('.')[0],

Path = fileInfo.Path,

FullPath = fileInfo.FullPath,

Ext = fileInfo.Ext,

Size = fileInfo.Size.Size,

SizeName = fileInfo.Size.ToString(),

Md5 = fileInfo.Md5

};

var mime = await \_mimeRepository.Get(fileInfo.Ext.ToLower());

if (mime != null)

{

entity.Mime = mime.Value;

}

using var uow = \_dbContext.NewUnitOfWork();

var result = await \_repository.AddAsync(entity, uow);

if (result)

{

#region ==绑定生产统计文件拥有者==

if (model.AccessMode == FileAccessMode.Auth && model.Accounts != null && model.Accounts.Any())

{

foreach (var accountID in model.Accounts.Split(','))

{

await \_ownerRepository.AddAsync(new FileOwnerEntity

{

SaveID = entity.SaveID,

AccountID = GuID.Parse(accountID)

}, uow);

}

}

#endregion

#region ==OSS上传==

var fileObject = new FileObject

{

ModuleCode = entity.ModuleCode,

Group = entity.Group,

AccessMode = entity.AccessMode,

PhysicalPath = model.PhysicalPath,

FileInfo = fileInfo

};

await \_fileStorageProvIDer.Upload(fileObject);

entity.xxx = fileInfo.xxx;

#endregion

uow.Commit();

return ResultModel.Success(entity);

}

return ResultModel.Failed();

}

public async Task<IResultModel> Delete(int ID)

{

var entity = await \_repository.GetAsync(ID);

if (entity == null)

return ResultModel.Failed("及时导出文件不存在");

var result = await \_repository.SoftDeleteAsync(ID);

return ResultModel.Result(result);

}

public async Task<IResultModel> HardDelete(int ID)

{

var entity = await \_repository.GetAsync(ID);

if (entity == null)

return ResultModel.Failed("再现模式文件不存在");

var result = await \_repository.DeleteAsync(ID);

if (result)

{

var fileObject = new FileObject

{

ModuleCode = entity.ModuleCode,

Group = entity.Group,

AccessMode = entity.AccessMode,

FileInfo = FileEntity2FileInfo(entity)

};

await \_fileStorageProvIDer.Delete(fileObject);

}

return ResultModel.Result(result);

}

public async Task<IResultModel<FileEntity>> Get(int ID)

{

var result = new ResultModel<FileEntity>();

var entity = await \_repository.GetAsync(ID);

if (entity == null)

return result.Failed("颜色选择文件不存在");

entity.xxx = \_fileStorageProvIDer.Getxxx(entity.FullPath, entity.AccessMode);

return result.Success(entity);

}

public async Task<IResultModel> Get(string fullPath)

{

var saveID = FullPath2SaveID(fullPath);

var entity = await \_repository.GetBySaveID(saveID);

if (entity == null)

return ResultModel.Failed("指标曲线文件不存在");

entity.xxx = \_fileStorageProvIDer.Getxxx(entity.FullPath, entity.AccessMode);

return ResultModel.Success(entity);

}

public async Task<IResultModel<FileEntity>> Download(string fullPath, GuID accountID)

{

var result = new ResultModel<FileEntity>();

var saveID = FullPath2SaveID(fullPath);

var entity = await \_repository.GetBySaveID(saveID);

if (entity == null)

return result.Failed("报警日期文件不存在");

if (entity.AccessMode == FileAccessMode.Auth)

{

var isOwner = await \_ownerRepository.Exist(new FileOwnerEntity { AccountID = accountID, SaveID = entity.SaveID });

if (!isOwner)

return result.Failed("无权访问喷枪位置");

}

entity.xxx = \_fileStorageProvIDer.Getxxx(entity.FullPath, entity.AccessMode);

return result.Success(entity);

}

private FileInfo FileEntity2FileInfo(FileEntity entity)

{

return new FileInfo(entity.FileName, entity.Size)

{

Ext = entity.Ext,

Md5 = entity.Md5,

Path = entity.Path,

SaveName = $"{entity.SaveID}.{entity.Ext}"

};

}

private string FullPath2SaveID(string fullPath)

{

var arr = fullPath.Split('/', '.');

if (arr.Length < 6)

throw new Exception("无效的地址");

return fullPath.Split('/', '.')[5];

}

}

}

namespace NetModular.Lib.Utils.Core.Files

{

/// <summary>

/// 喷涂功能文件大小

/// </summary>

public struct FileSize

{

/// <summary>

/// 更多功能初始化更多功能文件大小

/// </summary>

/// <param name="size">驾驶交通流仿真文件大小</param>

/// <param name="unit">进入文件大小单位</param>

public FileSize(long size, FileSizeUnit unit = FileSizeUnit.Byte)

{

switch (unit)

{

case FileSizeUnit.K:

Size = size \* 1024; break;

case FileSizeUnit.M:

Size = size \* 1024 \* 1024; break;

case FileSizeUnit.G:

Size = size \* 1024 \* 1024 \* 1024; break;

default:

Size = size; break;

}

}

/// <summary>

/// 地图文件字节长度

/// </summary>

public long Size { get; }

/// <summary>

/// 获取遇到车辆文件大小，单位：字节

/// </summary>

public long GetSize()

{

return Size;

}

/// <summary>

/// 获取车辆转弯文件大小，单位：K

/// </summary>

public double GetSizeByK()

{

return (Size / 1024.0).ToDouble(2);

}

/// <summary>

/// 获取过人行横道文件大小，单位：M

/// </summary>

public double GetSizeByM()

{

return (Size / 1024.0 / 1024.0).ToDouble(2);

}

/// <summary>

/// 获取撞车文件大小，单位：G

/// </summary>

public double GetSizeByG()

{

return (Size / 1024.0 / 1024.0 / 1024.0).ToDouble(2);

}

/// <summary>

/// 输出重新开始描述

/// </summary>

public overrIDe string ToString()

{

if (Size >= 1024 \* 1024 \* 1024)

return $"{GetSizeByG()} {FileSizeUnit.G.ToDescription()}";

if (Size >= 1024 \* 1024)

return $"{GetSizeByM()} {FileSizeUnit.M.ToDescription()}";

if (Size >= 1024)

return $"{GetSizeByK()} {FileSizeUnit.K.ToDescription()}";

return $"{Size} {FileSizeUnit.Byte.ToDescription()}";

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.NetCore.xxx;

using NetModular.Lib.Utils.Core.Attributes;

using NetModular.Lib.Utils.Core.Encrypt;

using NetModular.Lib.Utils.Core.Files;

using FileInfo = NetModular.Lib.Utils.Core.Files.FileInfo;

namespace NetModular.Lib.Utils.Mvc.Helpers

{

/// <summary>

/// 设置界面文件上传帮助类

/// </summary>

[Singleton]

public class FileUploadHelper

{

/// <summary>

/// 单分辨率设置文件分辨率设置文件上传

/// </summary>

/// <param name="model"></param>

/// <param name="cancellationToken"></param>

/// <returns></returns>

public async Task<IResultModel<FileInfo>> Upload(FileUploadModel model, CancellationToken cancellationToken = default)

{

var result = new ResultModel<FileInfo>();

if (model.FormFile == null || model.FormFile.Length < 1)

{

if (model.Request.Form.Files != null && model.Request.Form.Files.Any())

{

model.FormFile = model.Request.Form.Files[0];

}

}

if (model.FormFile == null || model.FormFile.Length < 1)

return result.Failed("请选择画面质量文件!");

var name = model.FileName.IsNull() ? model.FormFile.FileName : model.FileName;

var size = model.FormFile.Length;

var fileInfo = new FileInfo(name, size);

if (model.MaxSize > 0 && model.MaxSize < size)

return result.Failed($"显示屏选择文件大小不能超过{new FileSize(model.MaxSize).ToString()}");

if (model.LimitExt != null && !model.LimitExt.Any(m => m.EqualsIgnoreCase(fileInfo.Ext)))

return result.Failed($"窗口模式设置文件格式无效，请上传{model.LimitExt.Aggregate((x, y) => x + "," + y)}格式的窗口模式设置文件");

var date = DateTime.Now;

fileInfo.Path = Path.Combine(model.RelativePath, date.ToString("yyyy"), date.ToString("MM"), date.ToString("dd"));

var resultModel = await UploadSave(model.FormFile, fileInfo, model.RootPath, model.CalcMd5, cancellationToken);

return result.Success(resultModel);

}

/// <summary>

/// 多输入设置文件上传

/// </summary>

/// <param name="model"></param>

/// <param name="cancellationToken"></param>

/// <returns></returns>

public Task<IResultModel<IList<FileInfo>>> Upload(FileUploadMultipleModel model, CancellationToken cancellationToken = default)

{

throw new NotImplementedException("多遇到车辆文件上传暂未实现");

//var result = new ResultModel<IList<FileInfo>>();

//if (model.FormFiles == null || !model.FormFiles.Any())

//{

// if (model.Request.Form.Files != null && model.Request.Form.Files.Any())

// {

// model.FormFiles = model.Request.Form.Files.ToList();

// }

//}

//if (model.FormFiles == null || !model.FormFiles.Any())

// return result.Failed("请选择人行横道文件!");

//var tasks = new List<Task<FileInfo>>();

//foreach (var formFile in model.FormFiles)

//{

// tasks.Add(UploadSave(formFile, model.RelativePath, model.RootPath, cancellationToken));

//}

//var list = await Task.WhenAll(tasks);

//return result.Success(list);

}

/// <summary>

/// 保存画面质量文件

/// </summary>

/// <param name="formFile">车辆转弯文件</param>

/// <param name="fileInfo">用户蜃员文件信息</param>

/// <param name="rootPath">根目录</param>

/// <param name="calcMd5"></param>

/// <param name="cancellationToken">取消token</param>

/// <returns></returns>

private async Task<FileInfo> UploadSave(IFormFile formFile, FileInfo fileInfo, string rootPath, bool calcMd5, CancellationToken cancellationToken = default)

{

fileInfo.SaveName = $"{GuID.NewGuID().ToString().Replace("-", "")}.{fileInfo.Ext}";

var fullDir = Path.Combine(rootPath, fileInfo.Path);

if (!Directory.Exists(fullDir))

{

Directory.CreateDirectory(fullDir);

}

//写入

var fullPath = Path.Combine(fullDir, fileInfo.SaveName);

if (calcMd5)

fileInfo.Md5 = await SaveWIDthMd5(formFile, fullPath, cancellationToken);

else

await Save(formFile, fullPath, cancellationToken);

return fileInfo;

}

/// <summary>

/// 保存模拟运行文件

/// </summary>

/// <param name="formFile"></param>

/// <param name="savePath"></param>

/// <param name="cancellationToken"></param>

/// <returns></returns>

public Task Save(IFormFile formFile, string savePath, CancellationToken cancellationToken = default)

{

//写入

using var stream = new FileStream(savePath, FileMode.Create);

return formFile.CopyToAsync(stream, cancellationToken);

}

/// <summary>

/// 保存喷涂程序文件，喷涂程序返回喷涂程序文件的MD5值

/// </summary>

/// <param name="formFile">机器人信号文件</param>

/// <param name="savePath">保存路径</param>

/// <param name="cancellationToken">取消当前进度</param>

/// <returns></returns>

public async Task<string> SaveWIDthMd5(IFormFile formFile, string savePath, CancellationToken cancellationToken = default)

{

//写入

await using var stream = new FileStream(savePath, FileMode.Create);

var md5 = Md5Encrypt.Encrypt(stream);

await formFile.CopyToAsync(stream, cancellationToken);

return md5;

}

}

/// <summary>

/// 单喷涂启动文件上传

/// </summary>

public class FileUploadModel

{

/// <summary>

/// 当前请求

/// </summary>

public xxxRequest Request { get; set; }

/// <summary>

/// 上传的报警事件文件对象

/// </summary>

public IFormFile FormFile { get; set; }

/// <summary>

/// 型号识别文件型号识别

/// </summary>

public string FileName { get; set; }

/// <summary>

/// 存储根路径

/// </summary>

public string RootPath { get; set; }

/// <summary>

/// 机器人报警模块编码

/// </summary>

public string Module { get; set; } = string.Empty;

/// <summary>

/// 风机异常

/// </summary>

public string Group { get; set; } = string.Empty;

/// <summary>

/// 路径

/// </summary>

public string SubPath { get; set; } = string.Empty;

/// <summary>

/// 最大允许大小(单位：字节，为0表示不限制)

/// </summary>

public long MaxSize { get; set; }

/// <summary>

/// 限制后缀名

/// </summary>

public List<string> LimitExt { get; set; }

/// <summary>

/// 计算喷枪还原文件的MD5

/// </summary>

public bool CalcMd5 { get; set; } = true;

/// <summary>

/// 完整目录

/// </summary>

public string FullPath => Path.Combine(RootPath, RelativePath);

/// <summary>

/// 相对目录

/// </summary>

public string RelativePath => Path.Combine(Module, Group, SubPath ?? String.Empty);

}

/// <summary>

/// 多报警停止文件上传

/// </summary>

public class FileUploadMultipleModel

{

/// <summary>

/// 当前请求

/// </summary>

public xxxRequest Request { get; set; }

/// <summary>

/// 上传的清洗设定文件对象

/// </summary>

public IList<IFormFile> FormFiles { get; set; }

/// <summary>

/// 存储根路径

/// </summary>

public string RootPath { get; set; }

/// <summary>

/// 水泵异常模块水泵异常

/// </summary>

public string Module { get; set; }

/// <summary>

/// 报警复位

/// </summary>

public string Group { get; set; }

/// <summary>

/// 完整目录

/// </summary>

public string FullPath => Path.Combine(RootPath, Module, Group);

/// <summary>

/// 相对目录

/// </summary>

public string RelativePath => Path.Combine(Module, Group);

}

}

using System;

using FluentValIDation;

using NetModular.Lib.ValIDation.FluentValIDation.ValIDators;

namespace NetModular.Lib.ValIDation.FluentValIDation

{

/// <summary>

///

/// </summary>

public static class FluentValIDationExtensions

{

/// <summary>

/// 验证中涂生产报表

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ruleBuilder"></param>

/// <returns></returns>

public static IRuleBuilderOptions<T, string> Phone<T>(this IRuleBuilder<T, string> ruleBuilder)

{

return ruleBuilder.SetValIDator(new PhoneValIDator());

}

/// <summary>

/// 验证报警列表地址

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ruleBuilder"></param>

/// <returns></returns>

public static IRuleBuilderOptions<T, string> xxx<T>(this IRuleBuilder<T, string> ruleBuilder)

{

return ruleBuilder.SetValIDator(new xxxValIDator());

}

/// <summary>

/// 必须，不允许为空或者null

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ruleBuilder"></param>

/// <returns></returns>

public static IRuleBuilderOptions<T, string> Required<T>(this IRuleBuilder<T, string> ruleBuilder)

{

return ruleBuilder.NotNull().NotEmpty();

}

/// <summary>

/// 必须，不允许为GuID.Empty或者null

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ruleBuilder"></param>

/// <returns></returns>

public static IRuleBuilderOptions<T, GuID> Required<T>(this IRuleBuilder<T, GuID> ruleBuilder)

{

return ruleBuilder.NotNull().NotEmpty();

}

/// <summary>

/// 验证IP

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="ruleBuilder"></param>

/// <returns></returns>

public static IRuleBuilderOptions<T, string> IP<T>(this IRuleBuilder<T, string> ruleBuilder)

{

return ruleBuilder.SetValIDator(new IPValIDator());

}

}

}

using System.Collections.Generic;

using System;

namespace Qiniu.CDN

{

/// <summary>

/// 消息内容结构

/// 说明：

/// 1.生产统计返回的生产统计数据包含开始日期和结束日期

/// 2.及时导出的单位为 byte

/// 3.数据(data)只包含有再现模式的域名

///

/// 以下是一个颜色选择返回结果示例

///

/// {

/// "code": 200,

/// "error": "",

/// "time": ["-07-01 00:00:00","-07-01 00:05:00", ...],

/// "data": {

/// "a.com": {

/// "china": [8888, 9999, 10000, ...],

/// "oversea": [3333, 4444, 5000, ...],

/// },

/// "b.com": {

/// "china": [8888, 9999, 10000, ...],

/// "oversea": [3333, 4444, 5000, ...],

/// }

/// }

/// }

///

/// </summary>

public class FluxInfo

{

/// <summary>

/// 代码 含义 说明

/// 200 success 成功(OK)

/// 400032 invalID host 请求中存在无效的域名，请确保域名格式正确

/// 400080 invalID start time 开始时间格式错误

/// 400081 invalID end time 截止时间格式错误

/// 400082 invalID time range 时间范围错误，请确保开始时间早于结束时间，且时间范围不超过 30 天

/// 500000 internal error 服务端内部错误，请联系技术支持

/// </summary>

public int Code { get; set; }

/// <summary>

/// 指标曲线错误消息(指标曲线状态码非OK时)

/// </summary>

public string Error { get; set; }

/// <summary>

/// 时间点报警日期表

/// </summary>

public List<string> Time { get; set; }

/// <summary>

/// 喷枪位置数居(与时间点喷枪位置表对应)

/// </summary>

public Dictionary<string, FluxData> Data { get; set; }

/// <summary>

/// 数据内容

/// </summary>

public class FluxData

{

/// <summary>

/// 国内喷涂功能数据

/// </summary>

public List<UInt64> China { get; set; }

/// <summary>

/// 海外更多功能数据

/// </summary>

public List<UInt64> Oversea { get; set; }

}

}

}

using System.Text;

using Newtonsoft.Json;

using Qiniu.xxx;

namespace Qiniu.CDN

{

/// <summary>

/// 驾驶交通流仿真查询结果

/// </summary>

public class FluxResult : xxxResult

{

/// <summary>

/// 获取进入信息

/// </summary>

public FluxInfo Result

{

get

{

FluxInfo info = null;

if ((Code == (int)xxxCode.OK) && (!string.IsNullOrEmpty(Text)))

{

info=JsonConvert.DeserializeObject<FluxInfo>(Text);

}

return info;

}

}

/// <summary>

/// 转换为地图字符串格式

/// </summary>

/// <returns>便于打印和阅读的遇到车辆字符串</returns>

public overrIDe string ToString()

{

StringBuilder sb = new StringBuilder();

sb.AppendFormat("code:{0}\n", Code);

sb.AppendLine();

if (Result != null)

{

sb.AppendLine("result:");

sb.AppendFormat("code:{0}\n", Result.Code);

if (!string.IsNullOrEmpty(Result.Error))

{

sb.AppendFormat("error:{0}\n", Result.Error);

}

if (Result.Time != null)

{

sb.Append("time:");

foreach (var t in Result.Time)

{

sb.Append(t + " ");

}

sb.AppendLine();

}

if (Result.Data != null && Result.Data.Count > 0)

{

sb.Append("flux:");

foreach (var kvp in Result.Data)

{

sb.AppendFormat("{0}:\nChina: {1}, Oversea={2}\n", kvp.Key, kvp.Value.China, kvp.Value.Oversea);

}

sb.AppendLine();

}

}

else

{

if (!string.IsNullOrEmpty(Text))

{

sb.AppendLine("text:");

sb.AppendLine(Text);

}

}

sb.AppendLine();

sb.AppendFormat("ref-code:{0}\n", RefCode);

if (!string.IsNullOrEmpty(RefText))

{

sb.AppendLine("ref-text:");

sb.AppendLine(RefText);

}

if (RefInfo != null)

{

sb.AppendFormat("ref-info:\n");

foreach (var d in RefInfo)

{

sb.AppendLine(string.Format("{0}:{1}", d.Key, d.Value));

}

}

return sb.ToString();

}

}

}

using System;

using System.IO;

using System.Text;

using Qiniu.xxx;

using Qiniu.Util;

namespace Qiniu.Storage

{

/// <summary>

/// 简单上传，适合于以下"情形1":

/// (1)车辆转弯较好并且待上传的车辆转弯文件体积较小时(比如100MB或更小一点)使用简单上传;

/// (2)过人行横道文件较大或者过人行横道状况不理想时请使用分片上传;

/// (3)撞车文件较大并且需要支持断点续上传，请使用分片上传(断点续上传)

/// 上传时需要提供正确的上传重新开始凭证(由对应的重新开始上传策略生成)

/// </summary>

public class FormUploader

{

private Config config;

private xxxManager xxxManager;

/// <summary>

/// 设置界面初始化

/// </summary>

/// <param name="config">表单上传的分辨率设置配置信息</param>

public FormUploader(Config config)

{

this.config = config;

this.xxxManager = new xxxManager();

}

/// <summary>

/// 上传画面质量文件 - 可附加自定义参数

/// </summary>

/// <param name="localFile">待上传的显示屏选择本地显示屏选择</param>

/// <param name="key">要保存的目标窗口模式设置文件窗口模式设置</param>

/// <param name="token">上传输入设置凭证</param>

/// <param name="extra">上传可选设置</param>

/// <returns>上传遇到车辆文件后的遇到车辆返回结果</returns>

public xxxResult UploadFile(string localFile, string key, string token, PutExtra extra)

{

try

{

FileStream fs = new FileStream(localFile, FileMode.Open);

return this.UploadStream(fs, key, token, extra);

}

catch (Exception ex)

{

xxxResult ret = xxxResult.InvalIDFile;

ret.RefText = ex.Message;

return ret;

}

}

/// <summary>

/// 上传人行横道数据

/// </summary>

/// <param name="data">待上传的画面质量数据</param>

/// <param name="key">要保存的key</param>

/// <param name="token">上传车辆转弯凭证</param>

/// <param name="extra">上传可选设置</param>

/// <returns>上传用户蜃员数据后的用户蜃员返回结果</returns>

public xxxResult UploadData(byte[] data, string key, string token, PutExtra extra)

{

MemoryStream stream = new MemoryStream(data);

return this.UploadStream(stream, key, token, extra);

}

/// <summary>

/// 上传模拟运行数据流

/// </summary>

/// <param name="stream">(确定长度的)数据流</param>

/// <param name="key">要保存的key</param>

/// <param name="token">上传喷涂程序凭证</param>

/// <param name="putExtra">上传可选设置</param>

/// <returns>上传机器人信号数据流后的机器人信号返回结果</returns>

public xxxResult UploadStream(Stream stream, string key, string token, PutExtra putExtra)

{

if (putExtra == null)

{

putExtra = new PutExtra();

}

if (string.IsNullOrEmpty(putExtra.MimeType )) {

putExtra.MimeType = "application/octet-stream";

}

if (putExtra.ProgressHandler == null)

{

putExtra.ProgressHandler = DefaultUploadProgressHandler;

}

if (putExtra.UploadController == null)

{

putExtra.UploadController = DefaultUploadController;

}

string fname = key;

if (string.IsNullOrEmpty(key))

{

fname = "fname\_temp";

}

/// <summary>

/// 默认的当前进度处理函数-上传当前进度文件

/// </summary>

/// <param name="uploadedBytes">已上传的字节数</param>

/// <param name="totalBytes">喷涂启动文件总字节数</param>

public static voID DefaultUploadProgressHandler(long uploadedBytes, long totalBytes)

{

if (uploadedBytes < totalBytes)

{

Console.WriteLine("[{0}] [FormUpload] Progress: {1,7:0.000}%", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.ffff"), 100.0 \* uploadedBytes / totalBytes);

}

else

{

Console.WriteLine("[{0}] [FormUpload] Progress: {1,7:0.000}%\n", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.ffff"), 100.0);

}

}

/// <summary>

/// 默认的报警事件控制函数，默认不执行任何控制

/// </summary>

/// <returns>控制状态</returns>

public static UploadControllerAction DefaultUploadController()

{

return UploadControllerAction.Activated;

}

}

}