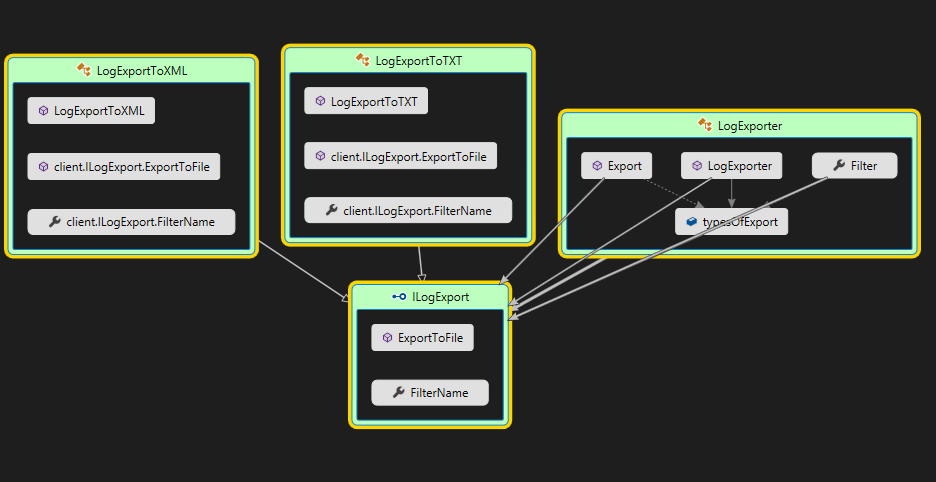
Паттерны

1. Паттерн мост



* 1. Основной код реализации

Файл Exports.cs

public interface ILogExport

{

/// <summary>

/// Строка имени фильтра в диалоге сохранения окна.

/// </summary>

string FilterName { get; }

/// <summary>

/// Прозводит экспорт в необходимый формат

/// </summary>

/// <param name="path">Путь куда будет сохраняться файл.</param>

/// <param name="messages">Сообщения,которые были экспортированны.</param>

void ExportToFile(string path, IEnumerable<Message> messages);

}

/// <summary>

/// Реализация экспорта в текстовый формат

/// </summary>

class LogExportToTXT : ILogExport

{

string ILogExport.FilterName

{

get { return "Текстовый файл |\*.txt"; }

}

void ILogExport.ExportToFile(string path, IEnumerable<Message> messages)

{

try

{

var file = File.CreateText(path);

foreach (var msg in messages)

{

file.WriteLine(msg.Stamp + " " + msg.ClientInfo.Name + " " + msg.Level + " " + msg.Text);

}

file.Close();

}

finally

{

}

}

}

/// <summary>

/// Реализация экспорта в XML формат

/// </summary>

class LogExportToXML : ILogExport

{

string ILogExport.FilterName

{

get { return "XML Файл |\*.xml"; }

}

void ILogExport.ExportToFile(string path, IEnumerable<Message> messages)

{

try

{

var file = File.CreateText(path);

var ser = new System.Xml.Serialization.XmlSerializer(typeof(Message));

foreach (var msg in messages)

{

ser.Serialize(file, msg);

}

file.Close();

}

finally

{

}

}

} }

/// <summary>

/// Мост организующий экспорт в любой из доступных форматов.

/// </summary>

public class LogExporter

{

private List<ILogExport> typesOfExport;

/// <summary>

///

/// </summary>

/// <param name="types"> Список ссылок на объекты, реализовывающие экспорт</param>

public LogExporter(List<ILogExport> types)

{

typesOfExport = types;

}

/// <summary>

/// Список фильтров,которые появятся в диалоге сохранения файла.

/// </summary>

public string Filter

{

get

{

string s = "";

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

{

if (i + 1 == typesOfExport.Count)

{

s += typesOfExport[i].FilterName;

}

else

{

s += typesOfExport[i].FilterName + "|";

}

}

s = s.PadLeft(s.Length - 1);

return s;

}

}

/// <summary>

/// Производит экспорт сообщений в указанный формат

/// </summary>

/// <param name="path">Имя файла(полный путь) куда будет произведен экспорт</param>

/// <param name="index">Индекс фильтраэ, который был применен при сохранении</param>

/// <param name="messages"> Сообщения,которые будут экспортированны</param>

public void Export(string path, int index, IEnumerable<Message> messages)

{

typesOfExport[index].ExportToFile(path, messages);

}

}

* 1. Код использования

Файл Form1.cs

private void exportButton\_Click(object sender, EventArgs e)

{

var saveFile = new SaveFileDialog();

var exportTypes = new List<ILogExport> { new LogExportToTXT(), new LogExportToXML() };

var exporter = new LogExporter(exportTypes);

saveFile.Filter = exporter.Filter;

var dialogResult = saveFile.ShowDialog();

if (dialogResult == DialogResult.OK)

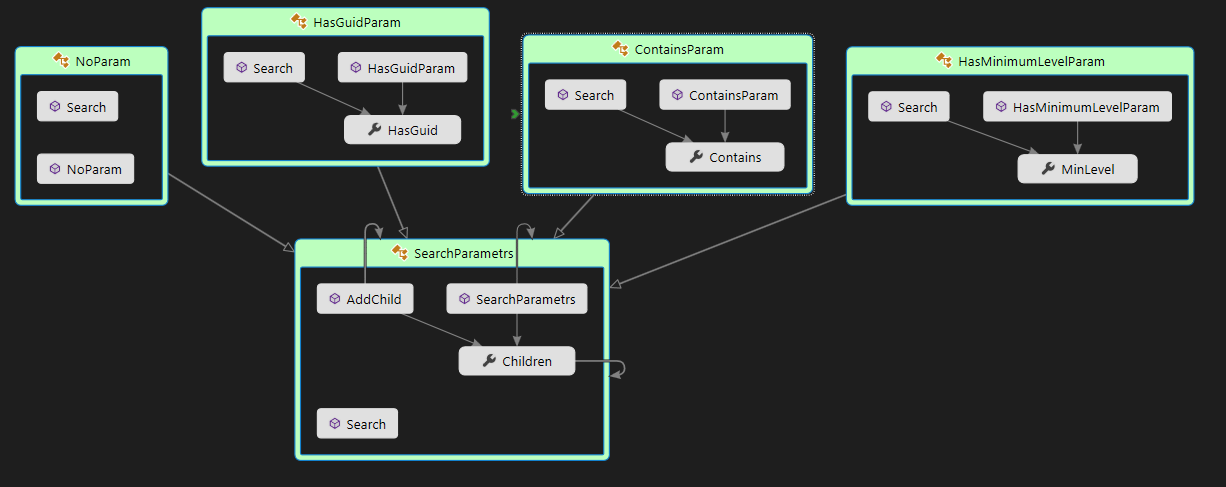
{

exporter.Export(saveFile.FileName, saveFile.FilterIndex - 1, (List<lib.Models.Message>)MessagesView.DataSource);

}

}

1. Паттерн «Компоновщик»



* 1. Основной код реализации

Файл SearchParametrsTree.cs

/// <summary>

/// Описывает параметры,которые могут быть применены для фильтрации IEnumerable Model.Message

/// </summary>

public abstract class SearchParametrs

{

/// <summary>

/// Список детей параметра, каждый каждый уровень вниз ,добавляет логическое и, вширь - логическое или

/// </summary>

public List<SearchParametrs> Children { get; private set; }

public SearchParametrs()

{

Children = new List<SearchParametrs>();

}

//public SearchParametrs Parent;

/// <summary>

/// Добавляет дочеренее древо параметров

/// </summary>

/// <param name="child">Новое дочернее древо параметров</param>

public void AddChild(SearchParametrs child)

{

//child.Parent = this;

Children.Add(child);

}

public abstract IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs);

}

public class HasGuidParam:SearchParametrs

{

/// <summary>

/// Имеет указанный Guid

/// </summary>

public Guid? HasGuid { get; set; }

public HasGuidParam(Guid guid)

{

HasGuid = guid;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.ClientInfo.Guid == HasGuid.ToString() select m;

}

}

public class HasMinimumLevelParam:SearchParametrs

{

/// <summary>

/// Минимальный уровень

/// </summary>

public int MinLevel { get; set; }

public HasMinimumLevelParam(int level)

{

MinLevel= level;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.Level >= MinLevel select m;

}

}

public class NoParam:SearchParametrs

{

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return raw\_msgs;

}

}

public class ContainsParam:SearchParametrs

{

/// <summary>

/// Содержит подстроку

/// </summary>

public string Contains { get; set; }

public ContainsParam(string str)

{

Contains = str;

}

public override IEnumerable<Models.Message> Search(IEnumerable<Models.Message> raw\_msgs)

{

return from m in raw\_msgs where m.Text.Contains(Contains) select m;

}

}

1. Паттерн Модуль таблицы

В данной реализации модулем таблицы является атрибут объект db класса WinHistory.

Файл WinHistory.cs

public class WinHistory

{

private static WinHistoryDbContext db = new WinHistoryDbContext();

...

}

Через данную сущность мы получаем доступ ко всем таблицам бд . Разделение области определения структуры таблицы происходит в классах неймспейса lib.Models

1. Паттерн Data Mapper

Как и в случае с паттерном Модуль таблицы , Entity Framework реализует данный паттерн, изолируя объекты от бд.

Пример использования:

Файл WinHistory.cs

/// <summary>

///

/// </summary>

/// <param name="guid">Уникальный guid программы</param>

/// <returns>Возвращает информацию о программе клиенте </returns>

public ClientInfo ReciveClientInfo(Guid guid)

{

return (from clients in db.Clients where clients.Guid == guid.ToString() select clients).Single();

}

Здесь объект db.Clients позволяет обращаться с таблицей данных Clients как с объектом.

1. Паттерн «Шлюз к записи данных»

Возвращаемый объект ClientInfo предыдущего примера является ярким примером шлоюза к записи данных. Мы можем обращаться к данным записи в базе, как к обычному объекту, используя данный паттерн.