var expensiveSofas = xDoc.Descendants("Sofa")

.Where(s => decimal.TryParse(s.Element("Price")?.Value, out decimal price) && price > 20000)

.ToList();

// Выводим найденные дорогие диваны

foreach (var sofa in expensiveSofas)

{

Console.WriteLine($"Название: {sofa.Element("Name")?.Value}, Цена: {sofa.Element("Price")?.Value}");

}

1. Что такое сериализация, десериализация?

**Сериализация** - процесс преобразования объектов или связанных объектов в поток байт (диск, память, сеть)

**Десериализация** — это процесс восстановления объекта из ранее сериализованного формата. То есть из байтов, строки или другого формата, в котором объект был сохранен, восстанавливается сам объект.

1. Какие существуют форматы сериализации? Поясните структуру для каждого формата. Какие классы для работы с ними существуют в .NET?
   1. бинарный –BinaryFormatter

В этом формате объект сериализуется в поток байтов, который можно сохранить в файл

Классы .NET: `BinaryFormatter` (устаревший), `DataContractSerializer`.

* 1. SOAP –SoapFormatter

- Структура: представляет данные в формате XML, оптимизированном для работы с веб-сервисами.

Классы .NET: `SoapFormatter` (устаревший).

* 1. xml – XmlSerializer

- Структура: данные представлены в виде элементов с тегами.

 Объекты представлены в виде иерархической структуры.

 Каждый объект или его свойства записываются как элементы XML.

`XmlSerializer`, `XmlDocument`.

* 1. JSON – DataContractJsonSerializer

Структура: данные хранятся в формате ключ-значение.

`JsonSerializer`

1. Какие классы существуют в пространстве имен System.Xml?

\*\*`XmlDocument`\*\*: позволяет работать с XML в виде дерева (парсинг, модификация).

- \*\*`XmlReader` / `XmlWriter`\*\*: чтение и запись XML в потоках.

- \*\*`XmlSerializer`\*\*: сериализация объектов в XML и обратно.

- \*\*`XPathNavigator`\*\*: позволяет выполнять XPath-запросы для поиска данных в XML.

- \*\*`XmlSchema`\*\*: работа с XML-схемами (валидация данных).

XmlElement - Представляет элемент XML-документа.

1. Какие атрибуты используются для настройки XML сериализации?

\*\*`[XmlRoot]`\*\*: задает корневой элемент XML.

`[XmlElement]`\*\*: задает имя XML-элемента для свойства или поля.

`[XmlAttribute]`\*\*: преобразует свойство в XML-атрибут.

\*\*`[XmlIgnore]`\*\*: исключает свойство из сериализации.

1. В чем отличие BinaryFormatter или SoapFormatter?

BinaryFormatter:

сериализует состояние объекта в поток, используя двоичный формат.

SoapFormatter:

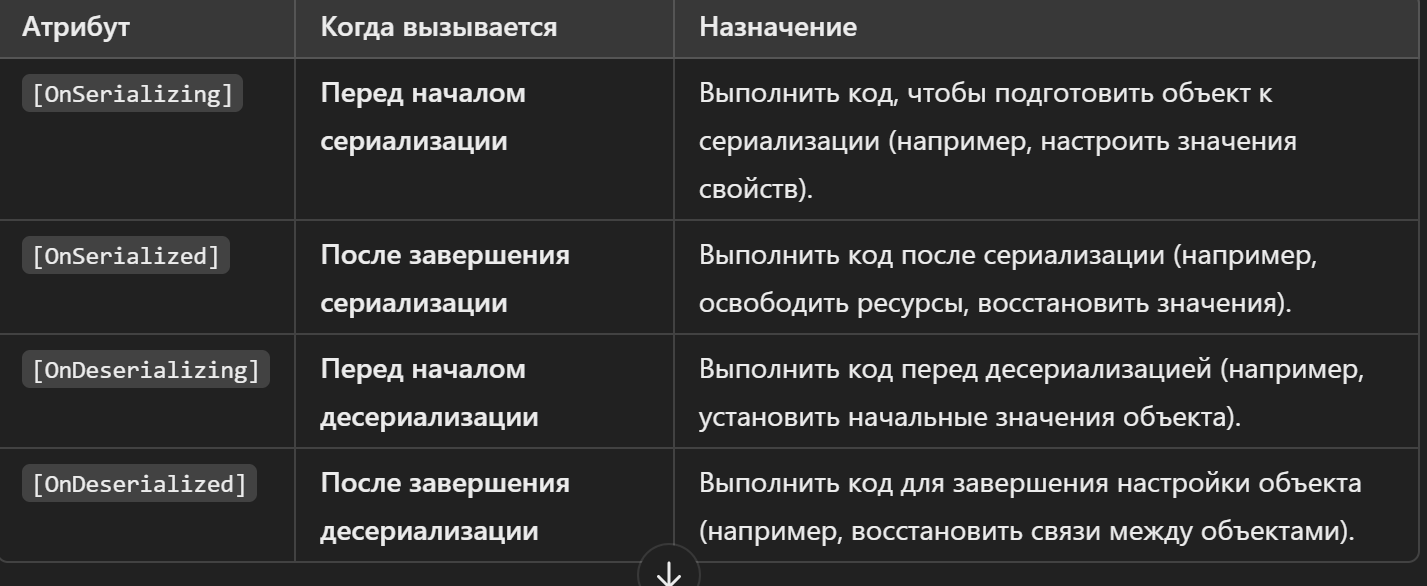
сохраняет состояние объекта в виде сообщения SOAP (стандартный XML-формат для передачи и приема сообщений от веб-служб).

1. Что такое сериализация контрактов данных, контракт данных?

Контракт данных – это тип (класс или структура), объект которого описывает информационный фрагмент (открытые поля и свойства)

Это процесс преобразования объекта, соответствующего контракту данных, в формат, который можно передать по сети

1. Где и для чего используются атрибуты [OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized]?



1. Что такое XPath? Приведите пример.

\*XPath\*\* — это язык для навигации по XML-документу. Позволяет выбирать узлы, элементы и атрибуты по запросам.

9. Какие возможности дает LINQ to Xml. Приведите примеры

LINQ to XML предоставляет гибкий и удобный способ работы с XML-документами. Это средство позволяет:

1. **Создавать** XML-документы.
2. **Загружать** XML-документы из файлов, строк или потоков.
3. **Изменять** структуру XML: добавлять, удалять, обновлять элементы и атрибуты.
4. **Запрашивать** данные из XML-документов с помощью выражений LINQ.
5. **Сохранять** XML-документы в файл или строку.

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization.Formatters.Soap;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Xml;

using System.Xml.Linq;

using System.Xml.Serialization;

namespace FurnitureStore

{

// Интерфейс сборки

public interface IAssemble

{

void Assemble();

void ShowInfo();

}

// Интерфейс сериализации

public interface ISerializer

{

void Serialize<T>(T obj, string filePath); // Метод для сериализации объекта типа T в файл по указанному пути (filePath). Объект obj будет преобразован в поток байтов и записан в файл.

T Deserialize<T>(string filePath); //Метод для десериализации объекта из файла по указанному пути (filePath). Файл будет считан, и его содержимое будет преобразовано обратно в объект типа T.

}

public class BinarySerializer : ISerializer

{

public void Serialize<T>(T obj, string filePath)

{

var formatter = new BinaryFormatter(); //это класс в .NET, который используется для сериализации объектов в бинарный формат

using (var stream = new FileStream(filePath, FileMode.Create)) //// получаем поток, куда будем записывать сериализованный объект.filePath: Это путь к файлу, в который мы будем записывать сериализованный объект. Если файл существует, его содержимое будет перезаписано.

// Если файл не существует, он будет создан.

{

//Внутри блока using вызывается метод Serialize у BinaryFormatter, который берет объект и записывает его в поток stream, то есть в файл.

formatter.Serialize(stream, obj); //Когда вызывается formatter.Serialize, объект obj преобразуется в последовательность байтов, которые записываются в файл через поток stream.

}

}

public T Deserialize<T>(string filePath)

{

var formatter = new BinaryFormatter(); //это класс, который умеет преобразовывать данные из бинарного формата обратно в объект.

using (var stream = new FileStream(filePath, FileMode.Open)) //Указывает, что операционная система должна открыть существующий файл.

{

return (T)formatter.Deserialize(stream);//Метод formatter.Deserialize(stream) читает данные из файла (потока) и преобразует их в объект. (T) — это приведение объекта к типу T, указанному при вызове метода.

}

}

}

public class SoapSerializer : ISerializer

{

public void Serialize<T>(T obj, string filePath)

{

var formatter = new SoapFormatter();

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

{

formatter.Serialize(stream, obj);

}

}

public T Deserialize<T>(string filePath)

{

var formatter = new SoapFormatter();

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

{

return (T)formatter.Deserialize(stream);

}

}

}

public class JsonSerializer : ISerializer

{

public void Serialize<T>(T obj, string filePath)

{

var options = new JsonSerializerOptions { WriteIndented = true }; //Это класс из библиотеки System.Text.Json, который предоставляет настройки для работы с JSON-сериализацией и десериализацией.

//Используется для управления форматированием JSON

var json = System.Text.Json.JsonSerializer.Serialize(obj, options);//JsonSerializer — это класс, который находится в пространстве имён System.Text.Json. Позволяет записывать JSON с отступами для лучшей читаемости

File.WriteAllText(filePath, json); //File.WriteAllText записывает текстовые данные в файл.

}

public T Deserialize<T>(string filePath)

{

var json = File.ReadAllText(filePath);

return System.Text.Json.JsonSerializer.Deserialize<T>(json);//то метод, который преобразует строку JSON (содержимое переменной json) обратно в объект типа T.

}

}

public class XmlSerializerWrapper : ISerializer

{

public void Serialize<T>(T obj, string filePath)

{

var serializer = new XmlSerializer(typeof(T)); //// передаем в конструктор тип класса

//// получаем поток, куда будем записывать сериализованный объект

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

{

serializer.Serialize(stream, obj);

}

}

public T Deserialize<T>(string filePath)

{

var serializer = new XmlSerializer(typeof(T));

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

{

return (T)serializer.Deserialize(stream); //Метод Deserialize объекта XmlSerializer читает XML-данные из потока stream и преобразует их обратно в объект типа T.

}

}

}

[Serializable]

public abstract class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

// Этот элемент не будет сериализован

[NonSerialized]

[XmlIgnore]

public string Material ;// Применён атрибут [NonSerialized]

protected Product(string name, decimal price)

{

Name = name;

Price = price;

}

public abstract void DisplayInfo();

public abstract void ShowInfo();

}

[Serializable]

public abstract class Furniture : Product, IAssemble

{

public string Color { get; set; }

protected Furniture(string name, decimal price, string material, string color)

: base(name, price)

{

Material = material; // Значение будет записано, но поле не сериализуется

Color = color;

}

public virtual void Assemble()

{

Console.WriteLine($"Сборка {Name} из {Material} цвета {Color}.");

}

public override void ShowInfo()

{

Console.WriteLine($"Общая информация - Мебель: {Name}, Материал: {Material}, Цвет: {Color}, Цена: {Price} руб.");

}

}

[Serializable]

public class Sofa : Furniture

{

public int SeatCount { get; set; }

public Sofa() : base("Диван", 0, "", "Цвет") { }

public Sofa(decimal price, string material, string color, int seatCount)

: base("Диван", price, material, color)

{

SeatCount = seatCount;

}

public override void DisplayInfo()

{

Console.WriteLine($"{Name}: {Material}, {Color}, Кол-во мест: {SeatCount}, Цена: {Price} руб.");

}

}

class Program

{

static void Main(string[] args)

{

// Коллекция объектов

var sofas = new List<Sofa>

{

new Sofa(19999, "Дерево", "Синий", 3),

new Sofa(15999, "Металл", "Красный", 2),

new Sofa(24999, "Ткань", "Зеленый", 4)

};

// Преобразуем коллекцию в массив для SOAP

var sofaArray = sofas.ToArray();

// Сериализаторы

ISerializer binarySerializer = new BinarySerializer();

ISerializer soapSerializer = new SoapSerializer();

ISerializer jsonSerializer = new JsonSerializer();

ISerializer xmlSerializer = new XmlSerializerWrapper(); // XML теперь работает с List<Sofa>

// Binary Serialization

binarySerializer.Serialize(sofas, "sofas\_binary.dat");

Console.WriteLine("Коллекция объектов сериализована в Binary формате.");

// Binary Deserialization

var binaryDeserializedSofas = binarySerializer.Deserialize<List<Sofa>>("sofas\_binary.dat");

Console.WriteLine("Коллекция объектов десериализована из Binary формата:");

foreach (var sofa in binaryDeserializedSofas)

{

sofa.ShowInfo();

}

// JSON Serialization

jsonSerializer.Serialize(sofas, "sofas.json");

Console.WriteLine("Коллекция объектов сериализована в JSON формате.");

// JSON Deserialization

var jsonDeserializedSofas = jsonSerializer.Deserialize<List<Sofa>>("sofas.json");

Console.WriteLine("Коллекция объектов десериализована из JSON формата:");

foreach (var sofa in jsonDeserializedSofas)

{

sofa.ShowInfo();

}

// XML Serialization

xmlSerializer.Serialize(sofas, "sofas.xml");

Console.WriteLine("Коллекция объектов сериализована в XML формате.");

// XML Deserialization

var xmlDeserializedSofas = xmlSerializer.Deserialize<List<Sofa>>("sofas.xml");

Console.WriteLine("Коллекция объектов десериализована из XML формата:");

foreach (var sofa in xmlDeserializedSofas)

{

sofa.ShowInfo();

}

// SOAP Serialization

soapSerializer.Serialize(sofaArray, "sofas\_soap.xml");

Console.WriteLine("Коллекция объектов сериализована в SOAP формате.");

// SOAP Deserialization

var soapDeserializedSofas = soapSerializer.Deserialize<Sofa[]>("sofas\_soap.xml");

Console.WriteLine("Коллекция объектов десериализована из SOAP формата:");

foreach (var sofa in soapDeserializedSofas)

{

sofa.ShowInfo();

}

// Сериализация в XML с помощью Linq to XML

var xDoc = new XDocument( //Он используется для создания, модификации, сериализации и загрузки XML-документов в .NET.

new XElement("Sofas",

from sofa in sofas //перебирает коллекцию объектов sofas (список диванов). Для каждого объекта дивана создается новый элемент XML <Sofa>,

select new XElement("Sofa",

//В этом примере создается элемент <Name>, в котором содержится текстовое значение "Sofa".

new XElement("Name", sofa.Name), //XElement — это класс из библиотеки LINQ to XML в .NET, который представляет собой отдельный элемент в XML-документе.

new XElement("Price", sofa.Price),

new XElement("Material", sofa.Material),

new XElement("Color", sofa.Color),

new XElement("SeatCount", sofa.SeatCount)

)

)

);

// XPath представляет язык запросов в XML. Он позволяет выбирать элементы, соответствующие определенному селектору.

string newXmlPath = "new.xml";

xDoc.Save(newXmlPath);// метод Save используется для сохранения объекта XDocument (в данном случае, переменной xDoc, которая содержит ваш XML-документ) в файл по пути

Console.WriteLine($"новый XML сохранен в {newXmlPath}");

var wooSofas = xDoc.Descendants("Sofa")//Возвращает отфильтрованную коллекцию элементов-потомков для этого документа

.Where(s => s.Element("Material")?.Value == "Дерево")//Для каждого элемента <Sofa> проверяется, что значение его дочернего элемента <Material> равно строке "Дерево".

.ToList();

// Выводим найденные диваны

foreach (var sofa in wooSofas)

{

Console.WriteLine($"Название: {sofa.Element("Name")?.Value}, Материал: {sofa.Element("Material")?.Value}, Цена: {sofa.Element("Price")?.Value}");

}

// Работа с XPath

var xmlDoc = new XmlDocument(); // создается новый экземпляр класса XmlDocument, который предоставляет методы и свойства для работы с XML-документом.

xmlDoc.Load("new.xml"); //Метод Load загружает содержимое XML-файла, указанного в качестве параметра, и загружает его в память

// XPath запрос для поиска всех объектов Sofa

XmlNodeList sofaNodes = xmlDoc.SelectNodes("//Sofa");// В данном случае запрос //Sofa ищет все элементы <Sofa>, которые присутствуют в документе

Console.WriteLine("\nВсе диваны в XML:");

foreach (XmlNode node in sofaNodes)

{

Console.WriteLine($"Название: {node["Name"].InnerText}, Материал: {node["Material"].InnerText}");//это свойство, позволяющее задавать или получать текстовое содержимое элемента

}

// XPath запрос для поиска соф, у которых материал "Дерево"

XmlNodeList woodSofas = xmlDoc.SelectNodes("//Sofa[Material='Дерево']"); //запрос с использованием XPath для выбора всех узлов <Sofa>, у которых дочерний элемент <Material> содержит значение "Дерево".

Console.WriteLine("\nДиваны с материалом 'Дерево':");

foreach (XmlNode node in woodSofas)

{

Console.WriteLine($"Название: {node["Name"].InnerText}, Материал: {node["Material"].InnerText}");

}

}

}

}