**1. Что такое сериализация и десериализация?**

**Сериализация** — процесс преобразования объекта в формат, подходящий для хранения или передачи (например, в файл, память или сеть).  
**Десериализация** — обратный процесс восстановления объекта из сохранённого состояния.

Примеры применения:

* Хранение объектов в файлах.
* Передача данных по сети.
* Кэширование.

**2. Форматы сериализации и их структура**

**1. Binary**

* Преобразует данные в бинарный поток.
* Компактный формат, но плохо читаемый человеком.
* В .NET можно использовать BinaryFormatter (устарел) или System.Text.Json.

**2. JSON (JavaScript Object Notation)**

* Структура: пары ключ-значение, массивы объектов.
* {
* "Name": "John",
* "Age": 30
* }
* Классы: JsonSerializer (из System.Text.Json), Newtonsoft.Json.

**3. XML (Extensible Markup Language)**

* Структура: элементы и атрибуты.
* <Person>
* <Name>John</Name>
* <Age>30</Age>
* </Person>
* Классы: XmlSerializer, XDocument, XmlDocument.

**4. SOAP (Simple Object Access Protocol)**

* XML-формат для обмена сообщениями между приложениями.
* <SOAP-ENV:Envelope>
* <SOAP-ENV:Body>
* <Person>
* <Name>John</Name>
* <Age>30</Age>
* </Person>
* </SOAP-ENV:Body>
* </SOAP-ENV:Envelope>
* Класс: SoapFormatter (устарел).

**5. Protobuf (Protocol Buffers)**

* Компактный бинарный формат, используемый для высокопроизводительной передачи данных.
* Библиотека: protobuf-net.

**3. Классы пространства имен System.Xml**

* **XmlDocument**: класс DOM для работы с узлами XML.
* **XDocument**: объектная модель LINQ to XML.
* **XmlReader и XmlWriter**: для потокового чтения и записи XML.
* **XmlSerializer**: для сериализации и десериализации объектов в XML.

**4. Атрибуты для настройки XML-сериализации**

* [XmlElement]: задаёт имя элемента XML.
* [XmlAttribute]: сериализует поле или свойство как атрибут.
* [XmlIgnore]: исключает поле из сериализации.
* [XmlRoot]: задаёт имя корневого элемента.
* [XmlArray] и [XmlArrayItem]: настраивают коллекции.

Пример:

[XmlRoot("Person")]

public class Person

{

[XmlElement("FullName")]

public string Name { get; set; }

[XmlIgnore]

public int Age { get; set; }

}

**5. Отличие BinaryFormatter и SoapFormatter**

* **BinaryFormatter**:
  + Формат: бинарный.
  + Более производителен.
  + Устарел из-за уязвимостей безопасности.
* **SoapFormatter**:
  + Формат: текстовый (XML, ориентированный на совместимость).
  + Устарел и не рекомендуется.

**6. Сериализация контрактов данных**

**Контракт данных** — описание структуры данных, используемой в WCF для обмена данными.  
**Сериализация контрактов данных**: сериализация объектов с помощью атрибутов [DataContract] и [DataMember].

Пример:

[DataContract]

public class Person

{

[DataMember]

public string Name { get; set; }

[DataMember]

public int Age { get; set; }

}

Класс: DataContractSerializer.

**7. Атрибуты [OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized]**

Эти атрибуты используются для выполнения кода до/после процесса сериализации или десериализации.

Пример:

[Serializable]

public class Person

{

public string Name { get; set; }

[OnSerializing]

private void OnSerializingMethod(StreamingContext context)

{

Console.WriteLine("Сериализация начата.");

}

[OnDeserialized]

private void OnDeserializedMethod(StreamingContext context)

{

Console.WriteLine("Десериализация завершена.");

}

}

**8. Что такое XPath? Пример**

**XPath** — язык запросов для выбора узлов в XML-документах.

Пример:

var xmlDoc = new XmlDocument();

xmlDoc.Load("example.xml");

var node = xmlDoc.SelectSingleNode("//Person/Name");

Console.WriteLine(node?.InnerText);

Запрос //Person/Name выбирает элемент <Name> внутри <Person>.

**9. Возможности LINQ to XML**

* Создание, изменение, и удаление XML-узлов.
* Выполнение запросов к XML-документам.

Пример:

using System.Xml.Linq;

var xml = new XDocument(

new XElement("People",

new XElement("Person",

new XElement("Name", "John"),

new XElement("Age", 30)

)

)

);

xml.Save("people.xml");

// Запрос

var names = from person in xml.Descendants("Person")

where (int)person.Element("Age") > 20

select person.Element("Name")?.Value;

foreach (var name in names)

{

Console.WriteLine(name);

}

LINQ to XML удобен для работы с данными в функциональном стиле.