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

**Сериализация** (в программировании) — процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция **десериализации** (структуризации) — восстановление начального состояния структуры данных из битовой последовательности.

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

В .NET можно использовать следующие форматы:

* **бинарный** (Двоичный (**бинарный**) **файл** — в широком смысле: последовательность произвольных байтов.)
* **SOAP** (SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body).)
* **xml** (**xml** — это текстовый файл в **формате XML** (от англ. eXtensible Markup Language) — расширяемый язык разметки, он описывает документ и частично описывает поведение программ, читающих **XML**-документы. **Формат** языка разрабатывался с расчетом применения широкого его в Интернете.)
* **JSON** (JSON-текст представляет собой (в закодированном виде) одну из двух структур: Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.)

Для каждого формата предусмотрен свой класс: для сериализации в бинарный формат - класс **BinaryFormatter**, для формата **SOAP** - класс **SoapFormatter**, для **xml** - **XmlSerializer**, для **json** - **DataContractJsonSerializer**.

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

В нем определен ряд классов, которые позволяют манипулировать xml-документом:

* **XmlNode**: представляет узел xml. В качестве узла может использоваться весь документ, так и отдельный элемент
* **XmlDocument**: представляет весь xml-документ
* **XmlElement**: представляет отдельный элемент. Наследуется от класса XmlNode
* **XmlAttribute**: представляет атрибут элемента
* **XmlText**: представляет значение элемента в виде текста, то есть тот текст, который находится в элементе между его открывающим и закрывающим тегами
* **XmlComment**: представляет комментарий в xml
* **XmlNodeList**: используется для работы со списком узлов

1. **Какие атрибуты используются для настройки XML сериализации?**

Ключевым классом, который позволяет манипулировать содержимым xml, является **XmlNode**, поэтому рассмотрим некоторые его основные методы и свойства:

* Свойство **Attributes** возвращает объект XmlAttributeCollection, который представляет коллекцию атрибутов
* Свойство **ChildNodes** возвращает коллекцию дочерних узлов для данного узла
* Свойство **HasChildNodes** возвращает true, если текущий узел имеет дочерние узлы
* Свойство **FirstChild** возвращает первый дочерний узел
* Свойство **LastChild** возвращает последний дочерний узел
* Свойство **InnerText** возвращает текстовое значение узла
* Свойство **InnerXml** возвращает всю внутреннюю разметку xml узла
* Свойство **Name** возвращает название узла. Например, <user> - значение свойства Name равно "user"
* Свойство **ParentNode** возвращает родительский узел у текущего узла

1. **В чем отличие BinaryFormatter или SoapFormatter?**

Протокол **SOAP** (Simple Object Access Protocol — простой протокол доступа к объектам) определяет стандартный процесс вызова методов в независящей от платформы и операционной системы манере.

1. **Что такое сериализация контрактов данных, контракт данных?**

**Контракт данных** – это тип (класс или структура), объект которого описывает информационный фрагмент. Подразумевается, что этот фрагмент может быть сохранён, а затем восстановлен. Работу с контрактами данных можно рассматривать как один из механизмов сериализации.

Для сериализации контракта данных используются классы:

* DataContractSerializer ‑ сериализует контракт;
* NetDataContractSerializer ‑ сериализует данные и тип контракта;
* DataContractJsonSerializer ‑ сериализует контракта в формате JSON.

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

|  |  |
| --- | --- |
| **[OnDeserialized]** | Этот атрибут позволяет указать метод, который будет вызван немедленно после десериализации объекта |
| **[OnDeserializing]** | Этот атрибут позволяет указать метод, который будет вызван перед процессом десериализации |
| **[OnSerialized]** | Этот атрибут позволяет указать метод, который будет вызван немедленно после того, как объект сериализован |
| **[OnSerializing]** | Этот атрибут позволяет указать метод, который будет вызван перед процессом сериализации |

1. **Что такое XPath? Приведите пример.**

**XPath** (XML Path Language) — язык запросов к элементам XML-документа.

XDocument xdoc = XDocument.Load(@"..\CharactersList\_xml.xml");

var items = from i in xdoc.Element("ArrayOfCharacters").Elements("Characters")

where Convert.ToInt32(i.Element("Attack").Value) > 100

select i;

1. **Какие возможности дает LINQ to Xml. Приведите примеры**

Чтобы начать работу с имеющимся **xml**-файлом, надо сначала загрузить его с помощью статического метода **XDocument.Load(),** в который передается путь к файлу.

Поскольку **xml** хранит иерархически выстроенные элементы, то и для доступа к элементам надо идти начиная с высшего уровня в этой иерархии и далее вниз.

Значение простых элементов, которые содержат один текст, можно получить с помощью свойства **Value**

Сочетая операторы Linq и LINQ to XML можно довольно просто извлечь из документа данные и затем обработать их. Например, имеется следующий класс:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Phone  {  public string Name { get; set; }  public string Price { get; set; }  } |

Создадим на основании данных в xml объекты этого класса:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | XDocument xdoc = XDocument.Load("phones.xml");  var items = from xe in xdoc.Element("phones").Elements("phone")  where xe.Element("company").Value=="Samsung"  select new Phone  {  Name = xe.Attribute("name").Value,  Price = xe.Element("price").Value  };  foreach (var item in items)  Console.WriteLine($"{item.Name} - {item.Price}"); |