Сериализация

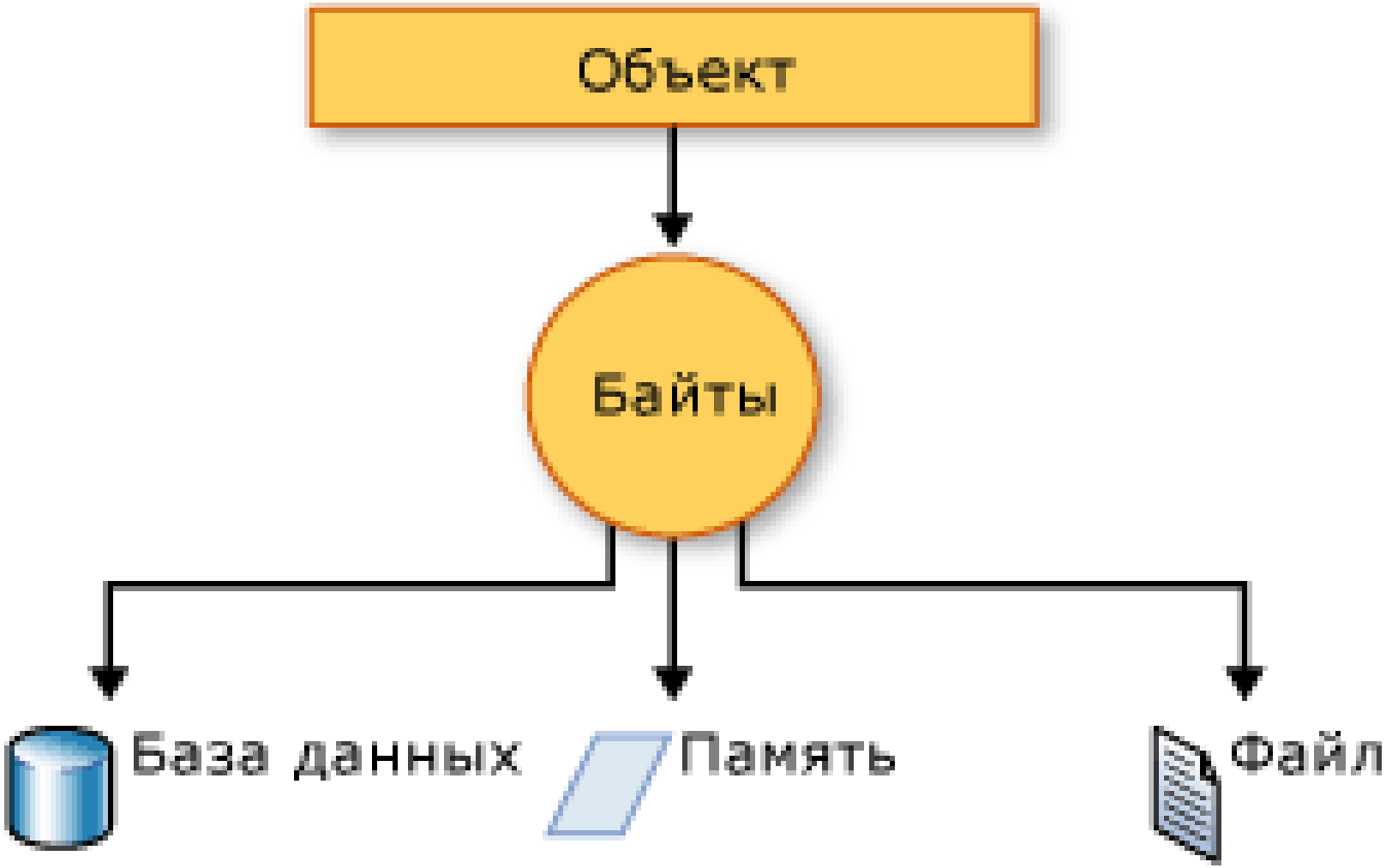
Понятие сериализации

- ***Сериализация*** – процесс преобразования структуры данных в последовательность байтов (или XML-узлов / JSON / etc.).
- ***Десереализация*** – восстановление структуры данных из последовательности байтов (или XML-узлов / JSON / etc.).

Disclaimer: в исходных кодах данной презентации отсутствует код обработки исключений (исключительно для удобочитаемости и отсутствия излишних нагромождений). Однако, это не означает, что исключения не надо обрабатывать…

SerializationException (как и многое другое) никто не отменял!

Зачем нужна сериализация?



Механизмы сериализации .NET

- контракты данных;
- двоичная;
- SOAP-сериализация;
- XML-сериализация;
- IXmlSerializable;
- JSON-сериализация.

Класс для сериализации

```
[Serializable]
public class AC_circuit    {
    public double R;    // активное сопротивление цепи
    public double L;    // индуктивность цепи
    public double C;    // емкость цепи
    public AC_circuit(double ri, double li, double ci) { // конструкт.
        R = ri; L = li; C = ci; }
    public AC_circuit() : this(0, 0, 0) { }
    public double Z(double omega)      {  // Полное сопротивление
        double sk = omega * L - 1 / (omega * C);
        return Math.Sqrt(R * R + sk * sk);
    }
    public override string ToString()      {
        return string.Format("R={0:F2}; L={1:F2}; C={2:F2}",  R, L, C);
    }
}  // AC_circuit
```

«Шаги» сериализации

(на примере хранения в файле)

1. **Создать объект класса.**
2. **Создать байтовый поток и связать его с потоком для записи (например, FileStream).**
3. **Создать объект сериализации, называемый форматером.**
4. **Используя метод Serialize() / WriteObject() объекта-форматера сохранить в потоке представление объекта.**
5. **Закрыть поток.**

Примечание: в качестве целевого хранилища можно вместо файла использовать память (MemoryStream), сеть (NetworkStream) и т.д.

Двоичная сериализация объекта

1) Пространство имен двоичной сериализации:

```
using System.Runtime.Serialization.Formatters.Binary;
```

2) Создание двоичного форматера – объекта класса:

```
BinaryFormatter binformatter = new BinaryFormatter();
```

3) Создание байтового потока (и файла):

```
FileStream fs = new FileStream("AC_circuit.bin", FileMode.Create);
```

4) Собственно сериализация – обращение к методу:

```
binformatter.Serialize(байтовый_поток, сериализуемый_объект);
```

Здесь байтовый_поток – это fs или ссылка на другой источник.

Двоичная сериализация одного объекта

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

class BinarySerExample    {
    static void Main( )    {
          AC_circuit r1 = new AC_circuit(1, 1, 1);
          BinaryFormatter binformatter = new BinaryFormatter();
          using (FileStream fs = new FileStream("AC_circuit.bin",
                                                FileMode.Create))
          {
                binformatter.Serialize(fs, r1);
          }
    }
}
```

Двоичная сериализация одного объекта

Результат сериализации в файле AC_circuit.bin:

_ яяяя_ __ CClassLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null__ _ClassLibrary1.AC_circuit_ _R_L_C ____ p? p? p?_

Схема двоичной десериализации

1) Пространство имен двоичной десериализации:

```
using System.Runtime.Serialization.Formatters.Binary;
```

2) Создать ссылку с типом десериализуемого объекта:

```
AC_circuit circuit = null;
```

3) Создать объект сериализации (форматер):

```
BinaryFormatter binformatter = new BinaryFormatter();
```

4) Создать байтовый поток и связать его с файлом (FileStream) :

```
FileStream file = new FileStream("AC_circuit.bin", FileMode.Open);
```

5) Выполнить десериализацию методом Deserialize(поток):

```
circuit = (AC_circuit)binformatter.Deserialize(file);
```

Пример двоичной десериализации

```
AC_circuit circuit = null;
using (FileStream file = new FileStream("AC_circuit.bin",
                                                    FileMode.Open))
{
        circuit = (AC_circuit)binformatter.Deserialize(file);
        Console.WriteLine("Восстановлен объект:");
        Console.WriteLine(circuit.ToString());
}
```

Результат выполнения десериализации и печати:

Восстановлен объект:

R=1,00; L=1,00; C=1,00

Шаги для XML-сериализации объекта

Пространство имен XML-сериализации:

```
using System.Xml.Serialization
```

Создание XML-форматера – объекта класса XmlSerializer :

```
XmlSerializer ser = new XmlSerializer(typeof(AC_circuit));
```

Создание байтового потока и файла:

```
FileStream fs = new FileStream("AC_circuit.xml", FileMode.Create))
```

Собственно сериализация - обращение к методу:

```
форматер.Serialize(байтовый_поток, сериализуемый_объект)
```

Пример XML-сериализации

```
using System;
using System.IO;
using System.Xml.Serialization;

namespace XMLSer  {
   class XMLSerializ   {
      static void Main(string[] args)   {
         AC_circuit r1 = new AC_circuit(1, 1, 1);
         XmlSerializer ser = new XmlSerializer(typeof(AC_circuit));
         using (FileStream fs = new FileStream("AC_circuit.xml",
                                                FileMode.Create))
          {   ser.Serialize(fs, r1);   }
      }
   }
}
```

Результат XML-сериализации объекта

Результат сериализации в файле AC_circuit.xml:

```
<?xml version="1.0"?>
<AC_circuit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <R>1</R>
  <L>1</L>
  <C>1</C>
</AC_circuit>
```

XML

XML (eXtensible Markup Language – расширяемый язык разметки)

XML – средство для создания, формирования и применения языков разметки для описания всех типов приложений и документов.

XML:
1) инструментарий для хранения данных;
2) конфигурируемое транспортное средство для информации любого рода;
3) развивающийся и открытый стандарт.

XML-десериализация

```
AC_circuit circuit = null;
// Создание XML-форматера для десериализации:
XmlSerializer deser = new XmlSerializer(typeof(AC_circuit));
using (FileStream fs = new FileStream("AC_circuit.xml", FileMode.Open))
{
        circuit = (AC_circuit)deser.Deserialize(fs);
}
Console.WriteLine("Восстановлен объект:");
Console.WriteLine(circuit.ToString());
```

```
Восстановлен объект:
R=1,00; L=1,00; C=1,00
```

Двоичная сериализация массива объектов

```
using System.Runtime.Serialization.Formatters.Binary;
. . . . .
  AC_circuit [] arCircuit = { new AC_circuit(1, 1, 1),
     new AC_circuit(2, 2, 2), new AC_circuit(3, 3, 3)};
  BinaryFormatter binformatter = new BinaryFormatter();
  using (FileStream fs = new FileStream("AC_array.bin", FileMode.Create))
  {   // Выполнение сериализации массива объектов:
        binformatter.Serialize(fs, arCircuit);
  }
```

Двоичная сериализация массива объектов

Результат сериализации в файле AC_circuit.bin:

_ яяяя_ __ CClassLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null__ _ _ __ClassLibrary1.AC_circuit_ _ _ _ _ __ _ClassLibrary1.AC_circuit_ _R_L_C ____ p? p? p?__
_ @ @ @__ _ _@ _@ _@_

Двоичная десериализация массива объектов

```
using System.Runtime.Serialization.Formatters.Binary;
. . . . .
   AC_circuit [] newArCircuit = null;  // другая (новая) ссылка
  BinaryFormatter arrformatter = new BinaryFormatter();
  using (FileStream file = new FileStream("AC_array.bin",  FileMode.Open))   {
      newArCircuit = (AC_circuit[])arrformatter.Deserialize(file);
      Console.WriteLine("Восстановлен массив объектов:");
      foreach(AC_circuit ac in newArCircuit)
          Console.WriteLine(ac.ToString());
  }
```

```
Восстановлен массив объектов:
R=1,00; L=1,00; C=1,00
R=2,00; L=2,00; C=2,00
R=3,00; L=3,00; C=3,00
```

XML-сериализация массива объектов

```
AC_circuit[] arCircuit = { new AC_circuit(1, 1, 1),
        new AC_circuit(2, 2, 2), new AC_circuit(3, 3, 3) };
// Создание XML-форматера:
XmlSerializer serXML = new XmlSerializer(typeof(AC_circuit[]));
using (FileStream fs = new FileStream("AC_circuit.xml", FileMode.Create))
        serXML.Serialize(fs, arCircuit);
```

```
<?xml version="1.0"?>
<ArrayOfAC_circuit
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <AC_circuit>
  <R>1</R>
  <L>1</L>
  <C>1</C>
 </AC_circuit>
…
 <AC_circuit>
  <R>3</R>
  <L>3</L>
  <C>3</C>
 </AC_circuit>
</ArrayOfAC_circuit>
```

```
<AC_circuit>
  <R>2</R>
  <L>2</L>
  <C>2</C>
 </AC_circuit>
```

XML-десериализация массива объектов

```
using System.Xml.Serialization;
. . . . .
 AC_circuit[] newArCircuit = null;
 // Создание XML-форматера для десериализации:
 XmlSerializer deserXML = new XmlSerializer(typeof(AC_circuit[]));
 using (FileStream fs = new FileStream("AC_circuit.xml", FileMode.Open))    {
          newArCircuit = (AC_circuit[])deserXML.Deserialize(fs);
 }
 Console.WriteLine("Восстановлен массив объектов:");
 Console.WriteLine(newArCircuit.ToString());
 foreach (AC_circuit ac in newArCircuit)
          Console.WriteLine(ac.ToString());
```

XML-десериализация массива объектов

Результаты выполнения программы десериализации:

Восстановлен объект:

ClassLibrary1.AC_circuit[]

R=1,00; L=1,00; C=1,00

R=2,00; L=2,00; C=2,00

R=3,00; L=3,00; C=3,00

XML-сериализация обобщенного списка

```
static void Main( ) {
   List<AC_circuit> list = new List<AC_circuit>();
   list.Add(new AC_circuit(1, 1, 1));
   list.Add(new AC_circuit(2, 2, 2));
   list.Add(new AC_circuit(3, 3, 3));
   XmlSerializer serXML = new XmlSerializer(typeof(List<AC_circuit>));
   using (FileStream fs = new
                          FileStream("CircuitList.xml",FileMode.Create))
   {
                  serXML.Serialize(fs, list);
   }
```

```
<?xml version="1.0"?>
<ArrayOfAC_circuit
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <AC_circuit>
  <R>1</R>
  <L>1</L>
  <C>1</C>
 </AC_circuit>
…
 <AC_circuit>
  <R>3</R>
  <L>3</L>
  <C>3</C>
 </AC_circuit>
</ArrayOfAC_circuit>
```

```
<AC_circuit>
  <R>2</R>
  <L>2</L>
  <C>2</C>
 </AC_circuit>
```

XML-десериализация обобщенного списка

```
List<AC_circuit> newList = null;
XmlSerializer deserXML = new
        XmlSerializer(typeof(List<AC_circuit>));
using (FileStream fs = new FileStream("CircuitList.xml",
                                      FileMode.Open))
 {
        newList = (List<AC_circuit>)deserXML.Deserialize(fs);
 }
 Console.WriteLine("Восстановлен список (XML):");
 Console.WriteLine(newList.ToString());
 foreach (AC_circuit ac in newList)
        Console.WriteLine(ac.ToString());
```

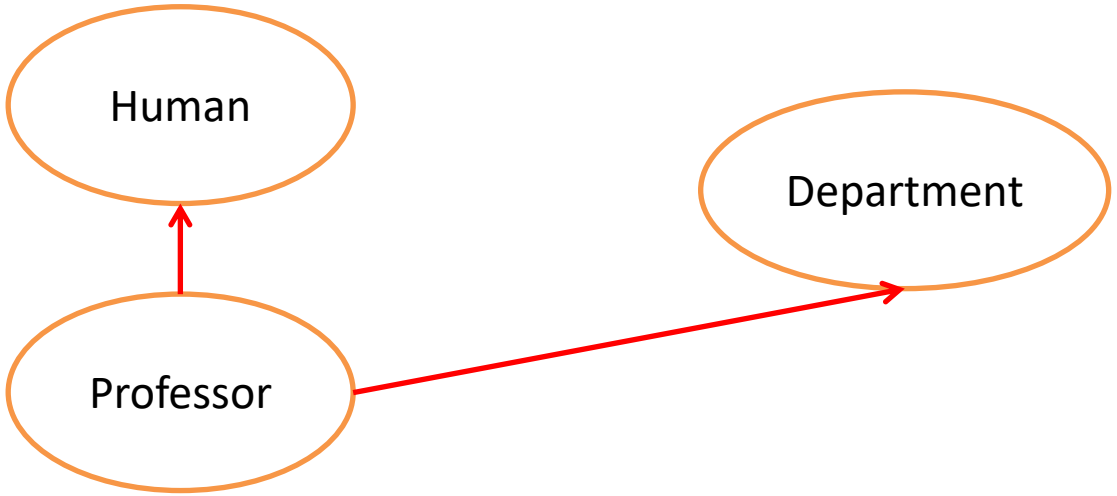
XML-десериализация списка объектов

Результаты выполнения программы десериализации:

Восстановлен объект:
System.Collections.Generic.List`1[ClassLibrary1.AC_circuit]
R=1,00; L=1,00; C=1,00
R=2,00; L=2,00; C=2,00
R=3,00; L=3,00; C=3,00

Граф объектов

Граф объектов – участвующий в процедуре сериализации набор взаимосвязанных объектов.



Класс «точка на плоскости»

```
[Serializable]
public class Point   {
   public double X { get; set; }
   public double Y { get; set; }
   public Point(double a, double b) { X = a; Y = b; }
   // Расстояние между точками:
   public double Distance(Point ps)  {
      double dx = X - ps.X;
      double dy = Y - ps.Y;
      return Math.Sqrt(dx * dx + dy * dy);
   }
}  // class Point
```

Класс «окружность» (композиция с точкой)

```
[Serializable]
public class Circle {   // !!!! композиция классов
        public Point center;    // центр круга
        double rad;             // радиус круга
        public Circle(double xc, double yc, double rad)   {
            center = new Point(xc, yc);
            this.rad = rad;
        }   // Circle( )
        public override string ToString()     {
          string format = "xc={0:g5},\tyc={1:g5},\tRad={2:g5}";
          string res = string.Format(format, center.X, center.Y, rad);
            return res;
        }   // ToString( )
        public double Rad { get { return rad; } } // Радиус круга
}   // class Circle
```

Двоичная сериализация графа объектов

```
class BinCircle    {
   static void Main()   {
      Circle cir = new Circle(1, 1, 1);
      BinaryFormatter binformatter = new BinaryFormatter();
      using (FileStream fs = new FileStream("Circle.bin", FileMode.Create))
      {   // Выполнение сериализации:
         binformatter.Serialize(fs, cir);
      }
   }
}
```

Операторы двоичной десериализации графа объектов

```
Circle circle = null;
using (FileStream file = new FileStream("Circle.bin", FileMode.Open))
{
    circle = (Circle)binformatter.Deserialize(file);
    Console.WriteLine("Восстановлен объект:");
    Console.WriteLine(circle.ToString());
}
```

Восстановлен объект:
xc=1, yc=1, Rad=1

Конструкторы умолчания

```
public Point() : this(0,0) {}
public Circle() : this(0,0,0) { }
```

XML-сериализация/десериализация возможна только при наличии конструкторов по умолчанию!

XML-сериализация

```
using System;
using System.IO;
using System.Xml.Serialization;

class XML_circle    {
  static void Main( )   {
    Circle cir = new Circle(1, 1, 1);
    XmlSerializer ser = new XmlSerializer(typeof(Circle));
    using (FileStream fxml = new FileStream("Circle.xml", FileMode.Create))
          ser.Serialize(fxml, cir);    // Выполнение сериализации
  }
}
```

Результат XML-сериализации

```
<?xml version="1.0"?>
<Circle xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <center>
    <X>1</X>
    <Y>1</Y>
  </center>
  <Rad>1</Rad>
</Circle>
```

XML-десериализация

```
 XmlSerializer ser = new XmlSerializer(typeof(Circle));
. . . . .
Circle circle = null;
using (FileStream file = new FileStream("Circle.xml", FileMode.Open))
   {
      circle = (Circle)ser.Deserialize(file);
      Console.WriteLine("Восстановлен объект:");
      Console.WriteLine(circle.ToString());
   }
```

Именование XML-элементов и XML-атрибутов

По умолчанию имена XML-элементов – это имена полей.

[XmlElement ("Новое_имя_XML-элемента")]

[XmlAttribute("*Имя_атрибута*")]

[XmlArray("*Имя_массива*")]

[XmlArray("*Имя_массива*")], [XmlArrayItem("*Имя_элемента*")]

XML-форматер

- Сохраняет значения только открытых (public) полей;
- Работает только с открытыми (public) классами;
- Игнорирует атрибут *NonSerialized;*
- Требует наличия конструктора с пустым списком параметров (конструктора по умолчанию) для всех классов, экземпляры которых присутствуют в графе объектов.

Атрибуты для двоичной сериализации

Двоичная сериализация:

- Атрибуты
- Реализация интерфейса **ISerializable**

[Serializable] – в объявлении типа;
[NonSerialized] – в объявлении игнорируемых полей.

Атрибуты для методов:
[OnSerializing] - перед сериализацией;
[OnSerialized] – после сериализации;

[OnDeserializing] - перед десериализацией;
[OnDeserialized] – после десериализации.

Атрибуты двоичной сериализации. Пример

```
[Serializable]
class Person    {  // Версия 1
       public string name;
}

[Serializable]
class Person    {  //  Версия 2
      public string name;
      [OptionalField (VersionAdded = 2)]
      public DateTime DateOfBirth;
}
```

Конфигурирование классов

[Serializable]

```
[Serializable]
class Student
```

[NonSerialized]

```
[NonSerialized]
private string Surname;
```

При использовании:

- Атрибут **Serializable** не наследуется.
- Атрибут ***NonSerialized <u>наследуется</u>.***

Интерфейс IXmlSerializable

Позволяет выполнить сериализацию/десериализацию типа в ручном режиме.

```
public System.Xml.Schema.XmlSchema GetSchema ();

public void WriteXml (System.Xml.XmlWriter writer);

public void ReadXml (System.Xml.XmlReader reader);
```

SOAP-сериализация объектов

```
[Serializable]
class Student
{
        public string Name { get; set; }
        public int Age { get; set; }

        public Student(string name, int age)
        {
            Name = name;
            Age = age;
        }
}
```

SOAP-сериализация

```
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

static void MainSerSOAP()
{
    Student[] st = new Student[] { new Student("Vic", 18),
                new Student("Alex", 20) };

    // создаем объект SoapFormatter
    SoapFormatter formatter = new SoapFormatter();
    // получаем поток, куда будем записывать сериализованный объект
    using (FileStream fs = new FileStream("students.soap", FileMode.OpenOrCreate))
    {
        formatter.Serialize(fs, st);

        Console.WriteLine("Объект сериализован");
    }
}
```

*Nuget: SoapFormatter (SoapFormatter and related types for .NET Standard)

SOAP-десериализация

```
static void MainDeserSOAP()
{
    using (FileStream fs = new FileStream("students.soap",
        FileMode.OpenOrCreate))
    {
        // создаем объект SoapFormatter
        SoapFormatter formatter = new SoapFormatter();
        Student[] st = (Student[])formatter.Deserialize(fs);

        Console.WriteLine("Объект десериализован");
        Console.WriteLine("Восстановлен массив Person (SOAP):");
        foreach (var s in st)
        {
            Console.WriteLine("Имя: {0}, Возраст: {1}", s.Name, s.Age);
        }
    }
}
```

JSON – формат обмена данными

JSON (**J**ava**S**cript **O**bject **N**otation) - простой текстовый формат обмена данными, основанный на JavaScript.

Несмотря на происхождение от JavaScript считается независимым от языка (поддержка JSON есть на всех платформах).

Человекочитаемый! Удобен для чтения и написания как человеком, так и компьютером.

www.json.org, RFC 8259

(рассматривается классика, т.е. без учета json5.org!)

JSON. Типы данных. Строки.

Строка:

- Последовательность из 0+ символов Юникода;
- Заключается в двойные кавычки, например **"это строка"**;
- поддерживает эскейп-последовательности, начинающиеся с бэкслеша (****). Например: \t, \r, \n и т.д. Поддерживается \uXXXX формат, где X – шестнадцатеричная цифра. "\u0068ello" == "hello"

JSON. Типы данных. Числа.

Числа:

- Целые: **0, -5, 36**;
- Вещественные: **3.14, 456.567**;
- Экспоненциальный (научный) формат: **0.314e1, 0.456567E3**;
- Нет поддержки NaN или Infinity (вместо этого исп. null);
- Нет поддержки восьмеричной, шестнадцатеричной и т.д. системы счисления.

JSON. Типы данных. Логический.

Два значения: **true** / **false**

JSON. null

null – значение, обозначающее пустоту (или отсутствие значения).
Примечание: не рекомендуется использовать **null**, если требуется вернуть пустую коллекцию, используйте **[]**.

Составные типы данных **объекты** и **массивы** на следующих слайдах.

JSON. Все важное на одном слайде.

Данные в формате JSON:

- JavaScript-объекты **{ ... }** (множество пар **"имя" : значение**);
- массивы **[...]**;
- значения одного из типов:
  - строки в двойных кавычках;
  - число (целое или вещественное);
  - логическое значение (**true** или **false**);
  - **null**.

JSON. Примеры.

1) Пустой объект:

{ }

2) Объект с одной парой имя /значение:

{ "lastName" : "Ivanov" }

3) Объект с несколькими парами имя /значение:

{ "firstName" : "Ivan", "lastName" : "Ivanov" }

JSON. Массивы.

Массив – упорядоченная коллекция значений.

Правила:

- Начинается с символа **[**
- Заканчивается символом **]**
- Содержит набор значений, разделенных запятыми)

Доступ к элементу массива осуществляется по его индексу (начинается с нуля) с использованием операции **[]**.

Длина массива доступна в свойстве **length**.

JSON. Пример объекта с массивом

```
{
        "employeeId": 1234567,
        "name": "Ivanov Ivan",
        "hireDate": "2022-01-10",
        "location": "Moscow, RU",
        "hasDrivingLicence": false,
        "childrenAges": [ 3, 9 ]
}
```

JSON vs XML. Сравнение документов.

```
{ "Manufacturer":"ADN",
  "Name":"Steel",
  "Price": 100.0 }
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
      <Manufacturer>ADN</Manufacturer>
      <Name>Steel</Name>
      <Price>100.0</Price>
</root>
```

Сравнение JSON и XML

Общие черты:

• Текстовые форматы;

• Само описываемые (человекочитаемые);

• Иерархические (значение может являться объектом или массивом (JSON) ИЛИ узлом (XML).

Отличия JSON от XML:

- JSON легковеснее (быстрее), чем XML;
- В JSON значения типизируемые (в XML – строки)
- В JSON меньше синтаксиса и нет семантики (схем).
- JSON документ сразу доступен для JavaScript (eval()).

Недостатки JSON (по сравнению с XML)

- Отсутствие пространств имен;
- Нет валидации по схеме (нет DTD / XML Schema, но есть jsonlint.com, json-schema.org).

JSON. Когда использовать?

- Передача данных на сервер (от сервера), особенно в web-приложениях.
- Выполнение асинхронных AJAX-вызовов без перезагрузки страниц в web-приложениях.
- Работа с базами данных (особенно документно-ориентированными).
- Сохранение данных в локальном хранилище (сериализация).

Контракты данных

Контракт данных - формальное соглашение между службой и клиентом, абстрактно описывающее данные, обмен которыми происходит. Это значит, что для взаимодействия клиент и служба не обязаны совместно использовать одни и те же типы, достаточно совместно использовать одни и те же контракты данных. Контракт данных для каждого параметра и возвращаемого типа четко определяет, какие данные сериализуются (превращаются в XML/JSON/etc…) для обмена.

Появились как часть Windows Communication Foundation (WCF).

Атрибуты контрактов данных

- [DataContract] – для сериализуемых типов;
- [DataMember] – для членов сериализуемых типов;
- [EnumMember] – для членов перечислений;

https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/feature-details/types-supported-by-the-data-contract-serializer

JSON-сериализация через контракты данных

Сборка:

System.Runtime.Serialization.dll

Пространство имен:

using System.Runtime.Serialization.Json;

Класс определяет объект-форматер:

DataContractJsonSerializer

Методы форматера:

WriteObject() - сериализация

ReadObject() - десериализация

JSON-сериализация и наследование

```
[DataContract, KnownTypeAttribute(typeof(Professor))]      // XmlInclude(typeof(Professor))
public class Human {
        [DataMember]
        public string Name { get; set; }
        public Human(string name) { Name = name; }
}

[DataContract]
public class Professor : Human {
        public Professor(string name) : base(name) { }
}

DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(Professor));

using (FileStream fs = new FileStream("doc.json", FileMode.Create))
        serializer.WriteObject(fs, new Professor("Ivanov"));

Professor prof;
using (FileStream fs = new FileStream("doc.json ", FileMode.Open))
        prof = deser.ReadObject(fs) as Professor;
```