Serialization and Deserialization

Сериализация представляет собой процесс преобразования объектов поток байтов с целью сохранения его в памяти, в базе данных или в файле.

Основное назначение сериализации—сохранить состояние объекта для того, что бы иметь возможность воссоздать его в случае необходимости.

Процесс обратный сериализации называется десериализацией

Атрибуты

`Serializable` – отмечает элемент для сериализации.

`NonSerializable` – сериализация для элемента выполняться не будет.

`OptionalField` –используется для безопасной сериализации измененных элементов

Интерфейс

Методы интерфейса `IDeserializationCallback`:

`Void OnDeserialization(objectsender)` – вызывается после того, как завершился процесс десериализации

Чтобы объект определенного класса можно было сериализовать, надо этот класс пометить атрибутом **Serializable**

```
[Serializable]
class Person
{
    public string Name { get; set; }
    public int Year { get; set; }

    public Person(string name, int year)
    {
        Name = name;
        Year = year;
    }

}
```

Если мы не хотим, чтобы какой-то член класса сериализовался, то мы его помечаем атрибутом **NonSerialized**:

```
[Serializable]
class Person
{
    public string Name { get; set; }
    public int Year { get; set; }

    [NonSerialized]
    public string AccNumber { get; set; }

    public Person(string name, int year)
    {
        Name = name;
        Year = year;
    }

}
```

При наследовании подобного класса, следует учитывать, что атрибут Serializable автоматически не наследуется. И если мы хотим, чтобы производный класс также мог бы быть сериализован, то опять же мы применяем к нему атрибут

Формат сериализации

Хотя сериализация представляет собой преобразование объекта в некоторый набор байтов, но в действительности только бинарным форматом она не ограничивается. Итак, в .NET можно использовать следующие форматы:

- бинарный
- SOAP
- xml
- JSON

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

```
public interface IFormatter
{
    SerializationBinder Binder { get; set;}
    StreamingContext Context { get; set;}
    ISurrogateSelector SurrogateSelector  { get; set;}
    object Deserialize (Stream serializationStream);
    void Serialize (Stream serializationStream, object graph);

}
```

Хотя классы BinaryFormatter и SoapFormatter по-разному реализуют данный интерфейс, но общий функционал будет тот же: для сериализации будет использоваться метод `Serialize`, который в качестве параметров принимает поток, куда помещает сериализованные данные (например, бинарный файл), и объект, который надо сериализовать. А для десериализации будет применяться метод `Deserialize`, который в качестве параметра принимает поток с сериализованными данными.

Класс XmlSerializer не реализует интерфейс IFormatter и по функциональности в целом несколько отличается от BinaryFormatter и SoapFormatter, но и он также предоставляет для сериализации метод Serialize, а для десериализации Deserialize. И в этом плане очень легко при необходимости перейти от одного способа сериализации к другому

XML Serialization

Подходит для сериализации открытых типов и членов типов.

Позволяет сериализовать только отдельные объекты.

Для выполнения XML сериализации не обязательно использовать атрибут `Serializable`

Для того, чтобы сериализоватьобъект в формате XML необходимо выполнитьследующие действия

1.Объявить класс как открытый.

2.Объявить все члены класса, которые необходимо сериализовать, как открытые.

3.Создать конструктор не принимающий параметров.

```
// класс и его члены объявлены как public
[Serializable]
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    // стандартный конструктор без параметров
    public Person()
    { }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}
class Program
{
    static void Main(string[] args)
    {
        // объект для сериализации
        Person person = new Person("Tom", 29);
        Console.WriteLine("Объект создан");

        // передаем в конструктор тип класса
        XmlSerializer formatter = new XmlSerializer(typeof(Person));

        // получаем поток, куда будем записывать сериализованный объект
        using (FileStream fs = new FileStream("persons.xml",
FileMode.OpenOrCreate))
        {
            formatter.Serialize(fs, person);

            Console.WriteLine("Объект сериализован");
        }

        // десериализация
        using (FileStream fs = new FileStream("persons.xml",
FileMode.OpenOrCreate))
        {
            Person newPerson = (Person)formatter.Deserialize(fs);
```

```
                Console.WriteLine("Объект десериализован");
                Console.WriteLine("Имя: {0} --- Возраст: {1}", newPerson.Name,
newPerson.Age);
            }

            Console.ReadLine();
        }

    }
```

SoapFormatter

SOAP протокол основан на XML.

`SoapFormatter` представляет наиболее эффективный способ для сериализации объектов, которые будут передаваться по сети или десериализоваться на другой платформе.

`SoapFormatter` не поддерживает совместимость по сериализации между версиями .NET Framework

BinaryFormatter

`BinaryFormatter` представляет собой наиболее эффективный способ сериализации.

Обеспечивает совместимость между версиями .NET Framework

Ограничение: сериализация и десериализация должны выполняться только .NET приложениями

Методы интерфейса `ISerializable`:

> `Void GetObjectData(SerializationInfoinfo, StreamingContextcontext)` – помещает данные, необходимые для корректной сериализацииобъекта в объект класса `SerializationInfo`

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

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

class Program
{
    static void Main(string[] args)
    {
        // объект для сериализации
        Person person = new Person("Tom", 29);
        Console.WriteLine("Объект создан");

        // создаем объект BinaryFormatter
        BinaryFormatter formatter = new BinaryFormatter();
        // получаем поток, куда будем записывать сериализованный объект
        using (FileStream fs = new FileStream("people.dat",
FileMode.OpenOrCreate))
        {
            formatter.Serialize(fs, person);

            Console.WriteLine("Объект сериализован");
        }

        // десериализация из файла people.dat
        using (FileStream fs = new FileStream("people.dat",
FileMode.OpenOrCreate))
        {
            Person newPerson = (Person)formatter.Deserialize(fs);

            Console.WriteLine("Объект десериализован");
            Console.WriteLine("Имя: {0} --- Возраст: {1}", newPerson.Name,
newPerson.Age);
```

```
            }

            Console.ReadLine();
        }
    }
```

Сериализация в JSON. DataContractJsonSerializer

Для сериализации объектов в формат JSON в пространстве System.Runtime.Serialization.Json определен класс`DataContractJsonSerializer`. Чтобы задействовать этот класс, в проект надо добавить сборку *System.Runtime.Serialization.dll*. Для записи объектов в json-файл в этом классе имеется метод `WriteObject()`, а для чтения ранее сериализованных объектов - метод `ReadObject()`

```
using System;
using System.IO;
using System.Runtime.Serialization.Json;
using System.Runtime.Serialization;

namespace Serialization
{
    [DataContract]
    public class Person
    {
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public int Age { get; set; }
        [DataMember]
        public Company Company { get; set; }

        public Person()
        { }

        public Person(string name, int age, Company comp)
        {
            Name = name;
            Age = age;
            Company = comp;
        }
    }

    public class Company
    {
        public string Name { get; set; }

        public Company() { }

        public Company(string name)
        {
            Name = name;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Person person1 = new Person("Tom", 29, new Company("Microsoft"));
            Person person2 = new Person("Bill", 25, new Company("Apple"));
            Person[] people = new Person[] { person1, person2 };

            DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(Person[]));

            using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))
            {
                jsonFormatter.WriteObject(fs, people);
            }

            using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))
            {
                Person[] newpeople = (Person[])jsonFormatter.ReadObject(fs);

                foreach (Person p in newpeople)
```

```
                {
                    Console.WriteLine("Имя: {0} --- Возраст: {1} --- Компания: {2}", p.Name, p.Age,
p.Company.Name);
                }
            }
            Console.ReadLine();
        }
    }

}
```

```
[{
    "Age":29,
    "Company":{"Name":"Microsoft"},
    "Name":"Tom"
},{
    "Age":25,
    "Company":{"Name":"Apple"},
    "Name":"Bill"
}]
```