Сериализация и десериализация

Подготовил: Чеботарев А.В.

Сериализация и десериализации

- **Сериализация** представляет процесс преобразования какого-либо объекта в поток байтов.
- **Десериализация** представляет процесс восстановления из потока байтов ранее сохраненный объект.

Атрибуты

- Атрибуты обеспечивают эффективный способ связывания метаданных или декларативной информации с кодом (сборками, типами, методами, свойствами и т. д.).

Атрибут Serializable

- 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;
    }
}
```

Атрибут **NonSerialized**

При необходимости не сохранять элемент используют атрибут **NonSerialized**

.

```
public int Year { get; set; }

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

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

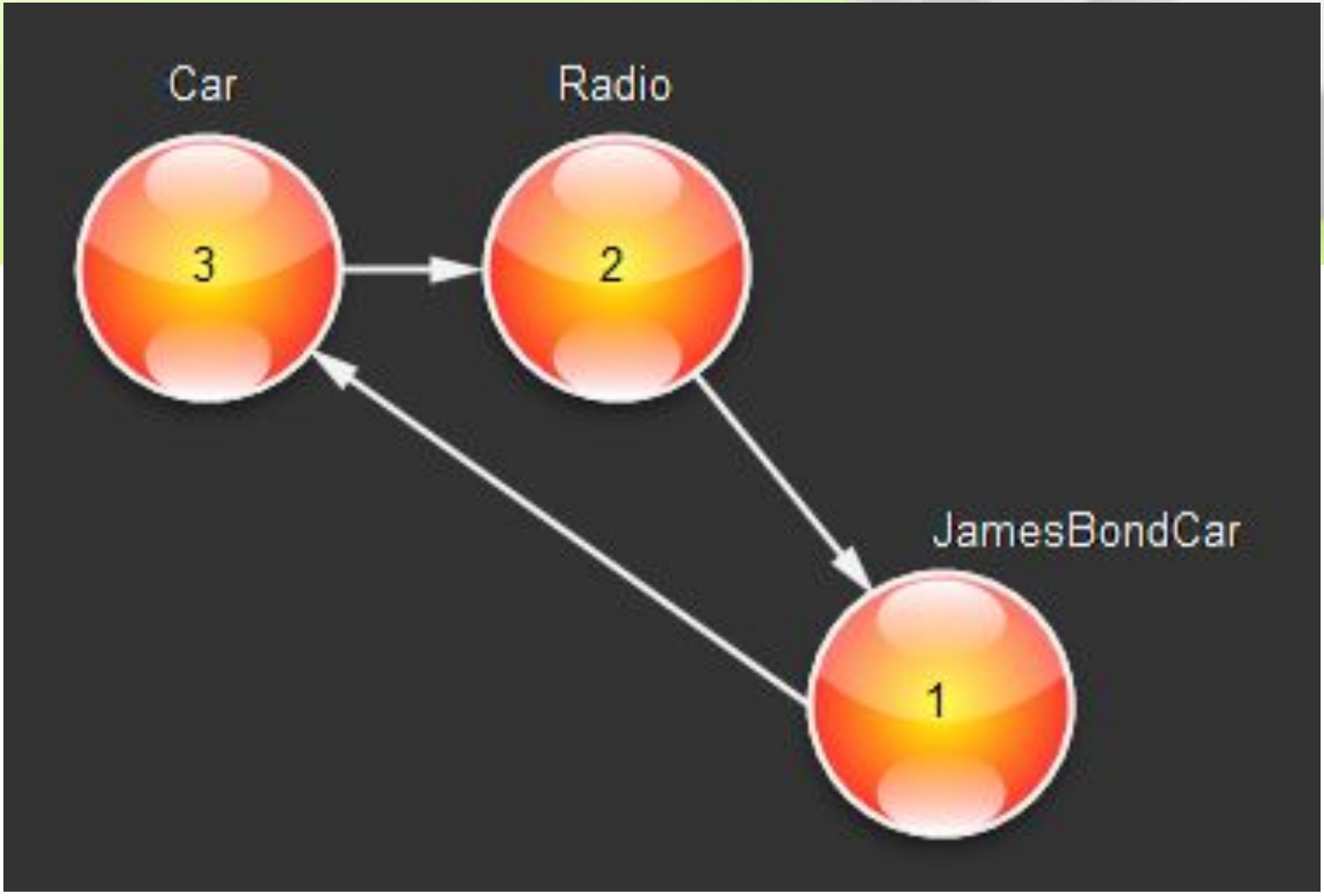
Граф объектов

- Среда CLR учитывает все связанные объекты, чтобы гарантировать корректное сохранение данных. Этот набор связанных объектов называется **графом объектов**.

Граф объектов

- Каждый объект в графе получает уникальное числовое значение. Имейте в виду, что числа, назначенные объектам в графе, являются произвольными и не имеют никакого значения для внешнего мира.

Граф объектов



Формат сериализации

- B.NET можно использовать следующие форматы сериализации:
- бинарный
- SOAP
- xml
- JSON

Дополнительная информация о данных

- *XML (Extensible Markup Language) - это новый SGML-производный язык разметки документов, позволяющий структурировать информацию разного типа, используя для этого произвольный набор инструкций.*

Дополнительная информация о данных

- В SOAP-посланиях передаются данные самых разных типов: числа, даты, строки символов, массивы, структуры. Определение типов этих данных выполняется, как обычно, в схемах XML. Схема может быть записана любым способом, но чаще всего применяется язык XSD

Дополнительная информация о данных

- В SOAP-посланиях передаются данные самых разных типов: числа, даты, строки символов, массивы, структуры. Определение типов этих данных выполняется, как обычно, в схемах XML. Схема может быть записана любым способом, но чаще всего применяется язык XSD

Формат сериализации

- Для каждого формата предусмотрен свой класс: для сериализации в бинарный формат - класс **BinaryFormatter**, для формата SOAP - класс **SoapFormatter**, для xml - **XmlSerializer**, для json - **DataContractJsonSerializer**.
- Эти классы расположены в **using** System.Runtime.Serialization.Formatters.<тип сериализации>

Интерфейс сериализации

```
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);
}
```

- XMLSerializer не реализует данный интерфейс

Точность данных

- Когда используется тип BinaryFormatter, он сохраняет не только данные полей объектов из графа, но также полное квалифицированное имя каждого типа и полное имя определяющей его сборки (имя, версия, маркер общедоступного ключа и культура).

Примеры программ(Binary)

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

namespace Serialization
{
    [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)
```

Примеры программ(Binary)

```
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);
}
```

Примеры программ(Binary)

- Использование массива данных

```
Person person1 = new Person("Tom", 29);
Person person2 = new Person("Bill", 25);
// массив для сериализации
Person[] people = new Person[] { person1, person2 };

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))
{
    // сериализуем весь массив people
    formatter.Serialize(fs, people);

    Console.WriteLine("Объект сериализован");
}

// десериализация
using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))
{
    foreach (Person p in deserilizePeople)
    {
        Console.WriteLine("Имя: {0} --- Возраст: {1}", p.Name, p.Age);
    }
```

Примеры программ(Soap)

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

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

Примеры программ(XML)

```
// 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");
}
```

«Тонкая настройка» сериализации

Тип данных	Значение
ISerializable	Этот интерфейс может быть реализован на типе [Serializable] для управления его сериализацией и десериализацией
ObjectIDGenerator	Этот тип генерирует идентификаторы для членов графа объектов
[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