

MERA



Сериализация

- ✓ Что такое сериализация и зачем она нужна
- ✓ Общая схема сериализации/десериализации
- ✓ Модуль форматирования
- ✓ Требования к сериализуемым типам
- ✓ Управление сериализацией с помощью настраиваемых атрибутов

- процесс преобразования объекта или графа связанных объектов в поток байтов

Обратный процесс называется десериализацией (deserialization)

- Состояние приложения (граф объектов) можно легко сохранить в файле на диске или в базе данных и восстановить при следующем запуске приложения.
- Набор объектов можно скопировать в буфер и вставить в то же или в другое приложение. Используется в приложениях Windows Forms и Windows Presentation Foundation (WPF)

- Можно клонировать набор объектов и сохранить как «резервную копию», пока пользователь работает с «основным» набором объектов.
- Набор объектов можно легко передать по сети в процесс, запущенный на другой машине

- Это набор особым образом связанных между собой объектов
- Когда объект сохраняется в потоке, все ассоциированные с ним данные (т.е. данные базового класса и содержащиеся в нем объекты) также автоматически сериализуются. Это обеспечивается путем создания графа объектов

Практический пример сериализации/десериализации



```
private static void SerializeBinaryFormat(Object objectGraph,
    string fileName)
{
    using (Stream fStream = new FileStream(fileName,
        FileMode.Create, FileAccess.Write, FileShare.None))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        // Заставляем модуль форматирования сериализовать
        // объекты в поток ввода-вывода
        formatter.Serialize(fStream, objectGraph);
    }
}
```

Практический пример сериализации/десериализации



```
private static Object DeserializeBinaryFormat(string
    fileName)
{
    // Задание форматирования при сериализации
    BinaryFormatter formatter = new BinaryFormatter();
    using (Stream fStream = File.OpenRead(fileName))
    {
        // Заставляем модуль форматирования десериализовать
        // объекты из потока ввода-вывода
        return formatter.Deserialize(fStream);
    }
}
```


Практический пример сериализации/десериализации



```
static void Main(string[] args)
{
    var objectGraph = new List<String> {
        "Jeff", "Kristin", "Aidan", "Grant" };
    SerializeBinaryFormat(objectGraph, "serialized.dat");
    objectGraph = null;
    // Десериализация объектов и проверка их работоспособности
    objectGraph =
        (List<String>)DeserializeBinaryFormat("serialized.dat");
    foreach (var s in objectGraph)
        Console.WriteLine(s);
}
```

- модуль форматирования (**BinaryFormatter** и **SOAPFormatter**)
 - **System.IO.Stream** – поток ввода-вывода*
- * граф объектов может быть сохранен в любом типе, производном от `System. IO. Stream`

- тип (он реализует интерфейс `System.Runtime.Serialization.IFormatter`), умеющий сериализовать и десериализовать граф объектов в поток типа `System.IO.Stream`

Интерфейс `IFormatter` реализуют `BinaryFormatter` и `SoapFormatter`

`BinaryFormatter` сохраняет объект в двоичном формате, а также полное имя объекта и полное имя определяющей его сборки (имя, версию, открытый ключ, culture)

`SoapFormatter` сериализует объект в формате SOAP, также указывая полную информацию о типе

Интерфейс

System.Runtime.Serialization.IFormatter



```
public interface IFormatter
{
    void Serialize(Stream serializationStream, object Graph);
    Object Deserialize(Stream serializationStream)
        ...
}
```

- ✓ По умолчанию сериализация типа не допускается

```
internal struct Point { public Int32 x, y; }
private static void OptInSerialization() {
    Point pt = new Point { x = 1, y = 2 };
    using (var stream = new MemoryStream()) {
        new BinaryFormatter().Serialize(stream, pt);
        // исключение SerializationException
    }
}
```

- ✓ Сериализовать можно только объекты классов (структур, перечислений, делегатов), имеющих атрибут `[Serializable]`

```
[Serializable]
```

```
internal struct Point { public Int32 x, y; }
```

- ✓ Если производный класс имеет атрибут [Serializable], базовый класс также должен иметь этот атрибут. Все типы в графе объекта также должны иметь атрибут [Serializable]

- Для сериализации в SOAP используйте **SoapFormatter**
- Протокол SOAP (Simple Object Access Protocol — простой протокол доступа к объектам) определяет стандартный процесс вызова методов в независимой от платформы и операционной системы манере.
- **SoapFormatter** не поддерживает сериализацию обобщений
- Чтобы использовать **SoapFormatter**, добавьте в проект сборку **System.Runtime.Serialization.Formatters.Soap.dll**

- Для сериализации в XML используйте `System.Xml.Serialization.XmlSerializer`
- `XmlSerializer` не требует наличия атрибута **[Serializable]**
- `XmlSerializer` сериализует только открытые поля и свойства объектов
- `XmlSerializer` требует наличия у сериализуемых объектов конструктора по умолчанию

Сериализация в формате XML. Пример

```
public class Radio
{
    public double [] stationPresets;
    public string radioID = "XF-552RR6";
}
public class Car
{
    public Radio theRadio = new Radio();
    public bool isHatchBack;
}
public class JamesBondCar : Car
{
    public bool canFly;
    public bool canSubmerge;
}
```

Сериализация в формате XML. Пример

```
static void SaveAsXmlFormat(object objGraph, string fileName)
{
    // Сохранить объект в файле CarData.xml в формате XML.
    XmlSerializer xmlFormat = new
        XmlSerializer(typeof(JamesBondCar),
            new Type[] { typeof(Radio), typeof(Car) });
    using (Stream fStream = new FileStream(fileName,
        FileMode.Create, FileAccess.Write, FileShare.None))
    {
        xmlFormat.Serialize(fStream, objGraph);
        Console.WriteLine("=> Saved car in XML format!");
    }
}
```

Сериализация в формате XML. Пример

```
static object LoadFromXmlFormat(string fileName)
{
    XmlSerializer xmlFormat = new
        XmlSerializer(typeof(JamesBondCar), new Type[] {
            typeof(Radio), typeof(Car) });
    using (Stream fStream = new FileStream(fileName,
        FileMode.Open, FileAccess.Read, FileShare.None))
    {
        return xmlFormat.Deserialize(fStream);
    }
}
```

Проблема:

Мне не нужно сериализовать ВСЕ поля сериализуемого типа

Причины:

- ✓ Поле содержит информацию, становящуюся недействительной после десериализации (Например, дескрипторы объектов ядра Windows)
- ✓ Поля содержат легко обновляемую информацию

Решение:

Использовать настраиваемый атрибут
System.NonSerializedAttribute

Управление сериализацией

Настраиваемый атрибут

```
[Serializable]
```

```
internal class Circle {  
    private Double m_radius;  
    [NonSerialized]  
    private Double m_area;  
    public Circle(Double radius) {  
        m_radius = radius;  
        m_area = Math.PI * m_radius * m_radius;  
    }  
}
```

Проблема:

Нужно изменить значения поля во время десериализации

```
static void Main(string[] args)
{
    Circle = new Circle();
    // выполнили сериализацию
    // выполнили десериализацию
    // проблема: m_area = 0 :-(
}
```

Решение:

Использовать настраиваемый атрибут

System.Runtime.Serialization.OnDeserializedAttribute

Управление сериализацией . Настраиваемый атрибут `System.Runtime.Serialization.OnDeserializedAttribute`

```
[Serializable]
internal class Circle
{
    private Double m_radius;
    [NonSerialized]
    private Double m_area;
    public Circle(Double radius)
    {
        m_radius = radius;
        m_area = Math.PI * m_radius * m_radius;
    }
    [OnDeserialized]
    private void OnDeserialized(StreamingContext context)
    {
        m_area = Math.PI * m_radius * m_radius;
    }
}
```


Управление сериализацией

Настраиваемые атрибуты

```
[OnDeserializing]
```

```
private void OnDeserializing(StreamingContext context) {  
    // Присвоение полям значений по умолчанию в новой версии типа  
}
```

```
[OnDeserialized]
```

```
private void OnDeserialized(StreamingContext context) {  
    // Инициализация временного состояния полей  
    sum = x + y;  
}
```

```
[OnSerializing]
```

```
private void OnSerializing(StreamingContext context) {  
    // Модификация состояния перед сериализацией  
}
```

```
[OnSerialized]
```

```
private void OnSerialized(StreamingContext context) {  
    // Восстановление любого состояния после сериализации  
}
```

- ✓ Для каждого сериализуемого объекта, кроме его состояния (значения сериализуемых полей), сохраняется полное имя сборки и информация о версии сборки
- ✓ При десериализации объекта загружается сборка и метаданные типа
- ✓ Если сборка, в которой находится сериализуемый тип, не имеет строгого имени (strong name), форматтеры полностью игнорируют информацию о версии
- ✓ Для сборок со строгим именем при сериализации и десериализации версии сборок с типами должны быть согласованы

Проблема:

Нужно десериализовать объект в более старую версию, где может не быть полей, определенных в сериализованном объекте

```
// старая версия типа
[Serializable]
public class Address
{
    string Street;
    string City;
}
```

```
// новая версия типа
[Serializable]
public class Address
{
    string Street;
    string City;
    string CountryField
}
```

Решение:

Использовать настраиваемый атрибут

System.Runtime.Serialization.OptionalFieldAttribute

```
// новая версия типа
[Serializable]
public class Address
{
    string Street;
    string City;
    [OptionalField]
    string CountryField
}
```

- Модуль форматирования
- **System.IO.Stream** – поток ввода-вывода
- **Настраиваемые атрибуты**

Интерфейс ISerializable

Реализация интерфейса **ISerializable** позволяет вмешаться в процесс управления сериализацией

```
public interface ISerializable
{
    void GetObjectData(SerializationInfo info,
        StreamingContext context);
}
```

Задание “Книжная картотека”

Реализуйте редактор книжных карточек. Каждая книжная карточка должна содержать информацию об одной книге: название, список авторов и год издания.

Программа должна показывать список книг (названий), а также детальную информацию о выбранной книге в списке: название, список авторов и год издания.

Программа должна позволять ввести данные о новой книге.

Данные о книгах должны сохраняться и впоследствии загружаться из файла. Реализуйте сериализацию и десериализацию в двоичном формате, форматах SOAP и XML

Реализуйте класс Прямоугольник.

Пусть полями класса будут длины двух его сторон и площадь.

Сделайте площадь несериализуемым полем.

Обеспечьте вычисление площади при десериализации объекта.

Q & A

MERA

