C# Serialization

321190
2012년 가을학기
11/28/2012
박경신

Serialization

- **Serializaiton(직렬화)**란 객체 상태를 지속시키거나 전송할 수 있는 형식으로 변환하는 과정으로, Serialization 반대로 다시 객체로 변환하는 것을 **Deserialization** 임
- Serialization을 사용하는 이유
  - 객체의 상태를 저장소에 보존 했다가 나중에 똑같은 복사본을 다시 만들기 위하거나, 한 응용프로그램에서 다른 응용프로그램으로 객체를 전송하기 위해서 임
- 닷넷에서 제공하는 Serialization 방식
  - **Binary Serialization**
    - 형식(Type) 정확도를 유지하므로 응용프로그램의 여러 호출간에 객체 상태를 유지하는데 유용. 객체를 스트림, 디스크, 메모리, 네트워크 등으로 serialize 함.
  - **XML Serialization**
    - public 속성과 필드만 serialize하며 형식 정확도를 유지하지 않음. XML은 공개 표준이므로 웹을 통해 데이터를 공유하는 데 효과적인 방법임.

Serialization

- **Binary Serialization**
  - Serialize하고자 하는 객체의 public, private 필드와 클래스 이름을 모두 바이트의 스트림(binary)로 변환하는 방식이며, deserialize 되면 원본 객체의 정확한 복제본이 생성
  - Binary Serialization 방법
    - **기본 Serialization**
    - **클래스의 일부 멤버를 Serialize하는 선택적 Serialization**
    - **ISerializable 인터페이스를 사용한 사용자 지정 Serialization**
- **XML/SOAP Serialization**
  - XML Serialization은 객체의 public 필드와 속성 또는 메소드의 매개 변수와 반환 값을 XML 스트림으로 Serialize하는 방식으로, XML Serialization을 사용하면 저장이나 전송을 위해 직렬형식으로 변환되는 public 속성 및 필드가 있는 강력한 형식 클래스 생성
  - XML/SOAP Serialization 방법
    - **XML Serialization**
    - **SOAP인코딩된 XML 스트림으로 Serialization**

기본 Serialization

- 클래스를 **Serializable 특성**으로 표시

```
[Serializable]
public class MyObject {
    public int n1 = 0; public int n2 = 0; public string str = null;
}
```

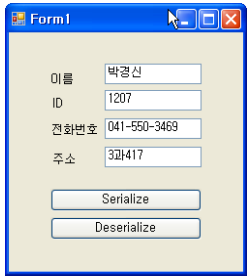
- 클래스 객체를 **Serialize하여 파일로 저장**

```
MyObject obj = new MyObject();      // 객체 생성
obj.n1 = 1; obj.n2 = 2; obj.str = "test";
FileStream stream = new FileStream("TestFile.bin", FileMode.Create, FileAccess.Write,
FileShare.None);                    // FileStream 생성
BinaryFormatter bf= new BinaryFormatter(); // BinaryFormatter 생성
bf.Serialze(stream, obj);           // Formatter에게 stream과 객체를 주고 serialize
stream.Close();                     // FileStream 닫기
```

- Serialize된 객체를 반대로 **Deserialize로 복원**

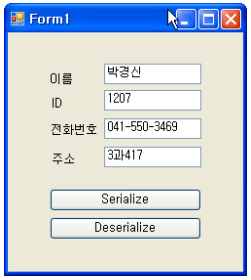
```
BinaryFormatter bf= new BinaryFormatter(); // BinaryFormatter 생성
FileStream stream = new FileStream("TestFile.bin", FileMode.Open, FileAccess.Read);
MyObject obj = (MyObject) bf.Deserialze(stream);  // Deserialize
stream.Close();                     // FileStream 닫기
```

```
[Serializable]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```




```
using System.Runtime.Serialization.Formatters.Binary;
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream(@"C:₩Person.bin", FileMode.Create,
                                   FileAccess.Write);
    BinaryFormatter bf= new BinaryFormatter();
    bf.Serialze(fs, p); // Formatter에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

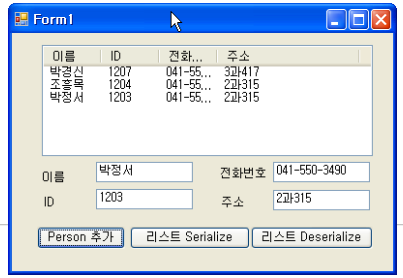
```
[Serializable]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```




```
using System.Runtime.Serialization.Formatters.Soap;
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream(@"C:₩Person.xml", FileMode.Create,
                                   FileAccess.Write);
    SoapFormatter sf= new SoapFormatter();
    sf.Serialze(fs, p); // SoapFormatter에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

```
using System.Runtime.Serialization.Formatters.Binary;
List<Person> pList = new List<Person>();
private void button1_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream(@"C:₩PersonList.bin", FileMode.Create, FileAccess.Write);
    BinaryFormatter bf= new BinaryFormatter();
    bf.Serialze(fs, pList); // BinaryFormatter에게 stream과 Person 리스트 객체를 주고 serialize
    fs.Close();
}
private void button2_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream(@"C:₩PersonList.bin", FileMode.Open, FileAccess.Read);
    BinaryFormatter bf= new BinaryFormatter();
    pList = (List<Person>)bf.Deserialze(fs); // Person 리스트 deserialize
    fs.Close();

    // 리스트뷰에 출력
    listView1.Items.Clear();
    foreach (Person p in pList) {
        listView1.Items.Add(p.ToListViewItem());
    }
}
```



선택적 Serialization

- 클래스에서 serialize하지 않아야 할 필드를 **NonSerialized 특성**으로 표시함으로써 해당 변수가 Serialize되지 않게 함

```
[Serializable]
public class MyObject {
    public int n1 = 0;
    [NonSerialized]
    public int n2 = 0; // n2 멤버는 더 이상 serialize되지 않음
    public string str = null;
}
```

Custom Serialization

- ISerializable 인터페이스를 사용한 사용자 지정 Serialization

```
// GetObjectData 메소드와 deserialize 될 때 사용되는 특수 생성자 구현이 포함되어야 함
[Serializable]
public class MyObject  : ISerializable {
    public int n1=0;     public int n2=0;     public string str=null;
    public MyObject() { }
    protected MyObject(SerializationInfo info, StreamingContext context) {
        n1 = info.GetInt32("i");
        n2 = info.GetInt32("j");
        str = info.GetString("k");
    } // deserialize시 필요
    [SecurityPermissionAttribute(SecurityAction.Demand, SerializationFormatter = true)]
    public virtual void GetObjectData(SerializationInfo info, StreamingContext context){
        info.AddValue("i", n1);
        info.AddValue("j", n2);
        info.AddValue("k", str);
    } // serialize시 필요
}
```

```
[Serializable]
public class Person: IComparable<Person>, IEquatable<Person>, ISerializable {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;

    // 중간 생략.....

    #region ISerializable
    public Person(SerializationInfo info, StreamingContext context) {
        this.name = info.GetString("Name");
        this.id = info.GetInt32("ID");
        this.phone = info.GetString("Phone");
        this.address = info.GetString("Address");
    }
    public void GetObjectData(SerializationInfo info, StreamingContext context) {
        info.AddValue("Name", this.name);
        info.AddValue("ID", this.id);
        info.AddValue("Phone", this.phone);
        info.AddValue("Address", this.address);
    }
    #endregion
}
```

Custom Serialization

- 사용자가 Serialization을 직접 제어하는 방식으로 ISerializable 인터페이스를 사용하는 방식 외에, **Serialization 도중과 이후에 데이터를 수정하는 데 사용되는 메소드에 다음 특성을 적용**
  - OnDeserializedAttribute
  - OnDeserializing Attribute
  - OnSerializedAttribute
  - OnSerializingAttribute

XML Serialization

- XML Serialization 특징
  - XML은 공개 표준이기 때문에 XML 스트림은 플랫폼에 관계없이 필요에 따라 모든 응용프로그램에서 처리될 수 있음
  - XML serialization은 SOAP 사양과 일치하는 XML 스트림으로 객체를 serialize하는 데 사용할 수 있음
  - 객체를 serialize하거나 deserialize 하기 위해서는 **XmlSerializer** 클래스를 사용
  - Serialize된 데이터에는 데이터 자체와 클래스의 구조만 포함
  - **Public 속성 및 필드만 serialize될 수 있음 – 만약 public 이 아닌 데이터를 serialize 해야 하는 경우 binary serialization을 사용할 것**
  - 클래스는 XmlSerializer에 의해 serialize될 기본 생성자가 있어야 함
  - 메소드는 serialize 될 수 없음

XML Serialization

- XmlSerializer를 사용하여 serialize/deserialize하는 방법

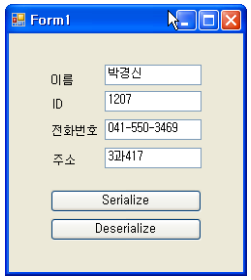
```
public class MyObject {
    public int n1=0;     private int n2=0;     public string str=null;
    public MyObject() { n1 = 1; n2 =2; str="XML serialization"; }
} // 결과물인 "obj.xml"에는 private 멤버와 메소드의 정보는 기록되지 않음
class Program {
    static void Main(string[] args) {
        MyObject obj = new MyObject();                    // MyObject 객체 생성
        // XmlSerializer 생성자에 serialize하고자 하는 객체의 타입을 전달하여 생성
        XmlSerializer xs = new XmlSerializer(typeof(MyObject));
        StreamWriter sw = new StreamWriter("obj.xml"); // stream writer 생성
        xs.Serialize(sw, obj);  // XmlSerializer에게 stream과 객체를 전달하여 serialize
        sw.Close();
        StreamReader sr = new StreamReader("obj.xml"); // stream reader 생성
        MyObject obj2 = (MyObject) xs.Deserialize(sr);    // deserialize
        Console.WriteLine("n1=" + obj2.n1 + " str=" + obj2.str);
        sr.Close();
    }
}
```

SOAP 인코딩된 XML Serialization

- SOAP 인코딩된 XML Serialize하기 위해서는, 새로운 SoapReflectionImporter를 만들고 serialize된 클래스의 형식으로 ImportTypeMapping 메소드를 호출하여 XmlTypeMapping을 만든뒤 XmlSerializer 생성자 매개변수에 전달

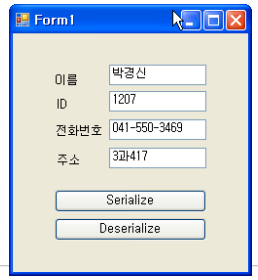
```
class Program {
    static void Main(string[] args) {
        MyObject obj = new MyObject();                    // MyObject 객체 생성
        XmlTypeMapping xtm =
            new SoapReflectionImporter().ImportTypeMapping(typeof(MyObject));
        XmlSerializer xs = new XmlSerializer(xtm);
        StreamWriter sw = new StreamWriter("obj.xml"); // stream writer 생성
        xs.Serialize(sw, obj);  // XmlSerializer에게 stream과 객체를 전달하여 serialize
        sw.Close();
    }
}
```

```
[XmlType(Namespace="urn:Person")]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    [XmlAttribute(AttributeName="Name")]
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```




```
using System.Xml;
Using System.Xml.Serialization;
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream(@"C:₩Person.xml", FileMode.Create, FileAccess.Write);
    XmlSerializer xs= new XmlSerializer(typeof(Person));
    xs.Serialze(fs, p); // XmlSerializer에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

```
[XmlType(Namespace="urn:Person")]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    [XmlAttribute(AttributeName="Name")]
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략
```




```
using System.Xml;
Using System.Xml.Serialization;
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream(@"C:₩Person.xml", FileMode.Create, FileAccess.Write);
    XmlTypeMapping xtm= new
        SoapReflectionImporter().ImportTypeMapping(typeof(Person));
    XmlSerializer xs= new XmlSerializer(xtm);
    xs.Serialze(fs, p); // XmlSerializer에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

XML

- XML 기본
  - Element와 Contents로 구성
  - Element는 문서의 구조를 정의하는 요소. 시작과 끝 태그(Tag)를 사용하여 표시
  - Contents는 실제 데이터
  - XML 파서(Parser)가 필요함
    - www.w3.org/XML에서 정의한 구문규칙을 사용
    - 대표적인 파서로 ms XML Core Services(MSXML)
  - XML 문서는 선택적으로 문서의 구조를 정의하는 DTD (Document Type Definition) 또는 스키마 (Schema)를 참조가능

XML 구성요소

- XML 구성요소
  - Element – 마크업 태그와 그 안에 포함된 내용
  - Root Element – 문서 내 모든 Element와 내용을 포함하고 있는 XML 문서의 요소
  - Attribute – Element에 포함되어 추가적인 정보를 제공
  - Entity – 텍스트, Binary, 비 ASCII 문자를 저장하는 데 사용
  - Processing Instruction – 전체 문서나 문서의 일부를 처리하는 응용프로그램과 연결해주는 명령어
  - Comment (주석) – XML 프로세서가 해석하지 않는 설명문
  - CDATA Section – 모든 문자를 마크업이나 태그로 인식하지않고 일반 문자로 인식할 수 있는 표기법. 즉, 특수한 문자를 일반 텍스트로 인식하도록 하는 표기법

XML Element

- public class Person {

```
[XmlElement("Name")]
public string Name   {
   get;   set;
}
[XmlElement("ID")]
public int ID   {
   get;      set;
} ..... }
```

- **<Person>**
  <Name>Park **</Name>**
  <ID>1207**</ID>**
  <Phone> 000-111-2222**</Phone>**
  <Address>단국대학교**</Address>**
  </Person>

XML Attribute

- public class Person {

```
[XmlAttribute(AttributeName="Name")]
public string Name   {
   get;         set;
}
[XmlAttribute(AttributeName="ID")]
public int ID   {
   get;      set;
} ..... }
```

- **<Person Name=“**Park**” ID=“**1207**” Phone=“**000-111-2222**” Address=“**단국대학교**”/>**

XML 구성요소

- 처리명령 (Processing Instruction)
  - <? 와 ?> 사이에 표현되고 특정 응용프로그램에 대해 처리할 정보나 명력을 가리키는 역할
  - **<?xml:stylesheet type="text/xsl" href="메모.xsl"?>**
    - XML 의 스타일 쉬트를 지정하고, 타입은 test/xsl이며, 소스는 메모.xsl에 있다는 의미
- 주석 (Comment)
  - <!- - 와 - -> 사이에 정의되고 주석처리로 사용
  - **<!-- <Person Name="**Park" **ID="**1207" **Phone="**000-111-2222" **Address="**단국대학교"**/>-->**
- CDATA 섹션 (CDATA Section)
  - <![CDATA[와]]> 사이에 정의되고 그 안의 내용을 문자열로 사용
  - **<![CDATA[<Person Name="**Park" **ID="**1207" **Phone="**000-111-2222" **Address="**단국대학교"**/>]]>**