LẬP TRÌNH ỨNG DỤNG MẠNG

Giảng viên: TS. Phạm Trương Hồng Ngân

Email: pthngan@ctu.edu.vn

Bộ Môn Tin Học Ứng Dụng

Khoa Công Nghệ Thông Tin và Truyền Thông

Đại Học Cần Thơ

CHƯƠNG 2
XUẤT NHẬP TRONG .NET FRAMEWORK

Thời gian: 6 tiết

NỘI DUNG

1. Stream trong .NET
2. Lớp FileStream
3. Các lớp Stream khác
4. Thao tác Stream
5. Serialization

GIỚI THIỆU

- I/O là vấn đề rất quan trọng đối với truyền thông trên mạng
- Chương này sẽ khảo sát các hoạt động I/O bên dưới
- Khảo sát vấn đề stream để phục vụ cho việc chuyển đổi các đối tượng phức tạp sang stream

STREAM TRONG .NET

STREAM TRONG .NET

- Một stream: là một đối tượng được sử dụng để truyền dữ liệu
  - Đọc stream: dữ liệu truyền từ các ***nguồn bên ngoài*** vào ứng dụng
  - Ghi stream: dữ liệu truyền từ chương trình ra ***nguồn bên ngoài.***
- **Nguồn bên ngoài:** có thể từ nhiều loại bao gồm từ máy in, đĩa cứng cho đến card mạng
  - Các tập tin
  - Đọc/ghi dữ liệu thông qua một giao thức mạng để trao đổi dữ liệu với một máy khác ở xa
  - Đọc/ghi vào một bộ đệm
  - Đọc/ghi vào một bộ nhớ

STREAM TRONG .NET (2)

- Tùy theo đối tượng tương tác mà các stream có các đặc trưng riêng
  - Chỉ cho đọc
  - Chỉ cho ghi
  - Cho phép truy cập nhẫu nhiên (cho phép thay đổi vị trí con trỏ đọc dữ liệu trong luồng - ví dụ dịch chuyển vào giữa luồng dữ liệu để đọc dữ liệu ở khoảng nào đó)
  - Chỉ truy cập tuần tự
- Thư viện .NET cung cấp lớp cơ sở System.IO.Stream để hỗ trợ làm việc đọc ghi các byte dữ liệu với các stream
- Các lớp stream kế thừa đặc thù như: FileStream, BufferStream, MemoryStream...

STREAM TRONG .NET (3)

- Các thuộc tính chung của stream

Thuộc tính	Ý nghĩa
CanRead	Cho biết stream hỗ trợ việc đọc hay không
CanWrite	Cho biết stream hỗ trợ việc ghi hay không
CanSeek	Cho biết stream hỗ trợ dịch chuyển con trỏ hay không
CanTimeout	Cho biết stream có đặt được time out
Length	Cho biết kích thước (byte) của stream
Position	Đọc hoặc thiết lập vị trí đọc (thiết lập thì stream phải hỗ trợ Seek)
ReadTimeout	Đọc hoặc thiết lập giá trị (mili giây) danh cho tác vụ đọc stream trước khi time out phát sinh
WriteTimeout	Đọc hoặc thiết lập giá trị (mili giây) danh cho tác vụ ghi stream trước khi time out phát sinh

STREAM TRONG .NET (4)

- Các phương thức chung của stream

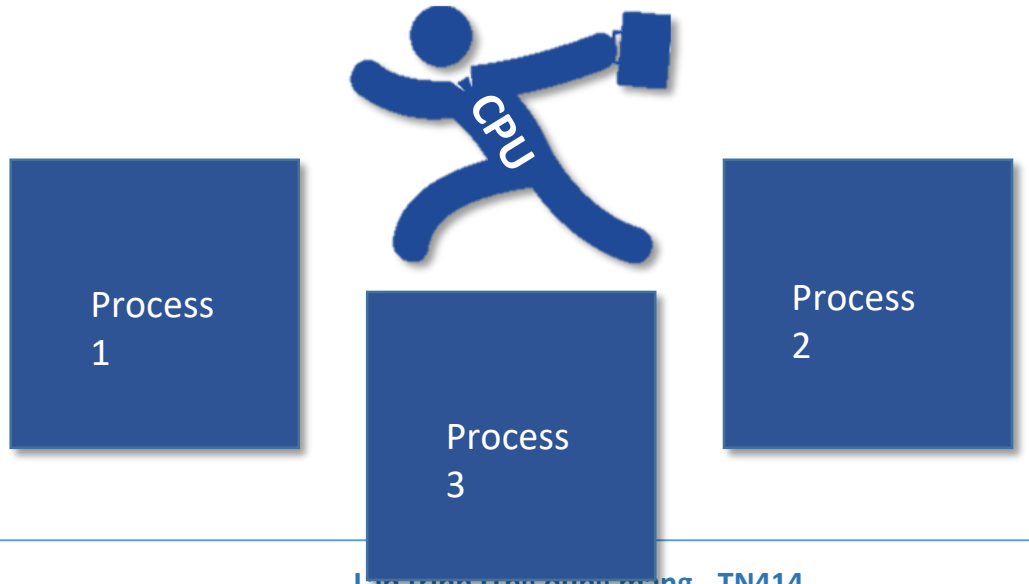
Phương Thức	Ý nghĩa
ReadByte	Đọc từng byte, trả về int (cast 1 byte) hoặc -1 nếu cuối file.
Read	Đọc một số byte, từ vị trí nào đó, kết quả đọc lưu vào mảng byte. Trả về số lượng byte đọc được, 0 nếu cuối stream.
WriteByte	Lưu 1 byte vào stream
Write	Lưu mảng bytes vào stream stream.Read(buffer, offset, count);
Seek	Thiết lập vị trí trong stream
Flush	Giải phóng các bộ đệm

STREAM TRONG .NET

- Hai stream quan trọng: networkStream và fileStream (networkStream sẽ học ở chương 4)
- Hai cách dùng stream: đồng bộ và bất đồng bộ
- Khi dùng đồng bộ: luồng (thread) tương ứng sẽ tạm ngưng đến khi tác vụ hoàn thành hoặc lỗi
- Khi dùng bất đồng bộ: luồng (thread) tương ứng sẽ ngay tức thì quay về phương thức gọi nó và bất cứ lúc nào tác vụ hoàn thành sẽ có dấu hiệu chỉ thị, hoặc lỗi xảy ra

STREAM TRONG .NET

- Kiểu chương trình "treo" để chờ tác vụ hoàn thành không "thân thiện" cho lắm, do đó phương thức gọi đồng bộ phải dùng một luồng riêng
- Bằng cách dùng các luồng và phương thức gọi bất đồng bộ làm cho có cảm giác máy tính có thể làm được nhiều việc cùng lúc



LỚP FILESTREAM

- Lớp FileStream tạo ra các đối tượng để đọc và ghi dữ liệu ra file
  - Do stream là tài nguyên không quản lý bởi graphic control, nên cần đưa nó vào cấu trúc using để tự động gọi giải phóng tài nguyên (Dispose) khi hết khối lệnh

```
string filepath = "/home/data/data.txt";
using (var stream = new FileStream(path:filepath,
mode:FileMode.Open, access: FileAccess.Read,
share: FileShare.Read))
{
    // code sử dụng stream (System.IO.Stream)
}
```

LỚP FILESTREAM

- Để tạo ra một stream file, cần 4 thông tin:
  - path: đường dẫn đến file
  - mode: kiểu liệt kê FileMode, các chế độ mở tập tin:
    - FileMode.CreateNew tạo file mới
    - FileMode.Create tạo mới, nếu file đang có bị ghi đè
    - FileMode.Open mở file đang tồn tại
    - FileMode.OpenOrCreate mở file đang tồn tại, tạo mới nếu không có
    - FileMode.Truncate mở file đang tồn tại và làm rỗng file
    - FileMode.Append mở file đang tồn tại và tới cuối file, hoặc tạo mới

LỚP FILESTREAM

- access: kiểu liệt kê FileAccess, cho biết muốn truy cập file như thế nào
  - FileAccess.Read chỉ đọc
  - FileAccess.Write chỉ ghi
  - FileAccess.ReadWrite đọc và ghi
- share: kiểu liệt kê FileShare, cho phép thiết lập chia sẻ truy cập file
  - FileShare.None không chia sẻ - tiến trình khác truy cập file sẽ lỗi cho đến khi tiến trình mở file đóng nó lại.
  - FileShare.Read cho tiến trình khác mở đọc file.
  - FileShare.Write cho tiến trình khác mở ghi file.
  - FileShare.ReadWrite cho tiến trình khác mở đọc ghi file.
  - FileShare.Delete cho tiến trình khác xóa file.

LỚP FILESTREAM

- Ngoài ra, lớp File cũng hỗ trợ tạo FileStream
  - File.OpenRead(filename) tạo stream để đọc
    File.OpenWrite(filename) tạo stream để ghi
- Lấy thông tin encoding của file Text
  - Khi đọc các file text (không phải file nhị phân), đầu tiên cần xác định encoding (UTF8, Unicode, UTF32 ...)
  - Thông tin về encoding được lưu ở vài byte đầu tiên của file nó gọi là BOM - Preamble
  - Giá trị của khoảng 5 byte đầu tiên xác định được encoding

LỚP FILESTREAM-XÁC ĐỊNH ENCODING

```
using System;
using System.IO;
using System.Text;
namespace Stream_FileStream {
  public class UtilsEncoding {
        public static Encoding GetEncoding (Stream stream) {
         //hàm xác định encoding bằng 5 byte đầu tiên
       byte[] BOMBytes = new byte[4];
         // mảng chứa 4 byte để làm bộ nhớ lưu byte đọc được
       int offset = 0;
        // vị trí (index) trong buffer - nơi ghi byte đầu tiên đọc
được
       int count = 4; // đọc 4 byte
       int numberbyte = stream.Read (BOMBytes, offset, count);
         // bắt đầu đọc 4 đầu tiên lưu vào buffer
         if (BOMBytes[0] == 0xfe && BOMBytes[1] == 0xff) {
         stream.Seek (2, SeekOrigin.Begin);
         // Di chuyển về vị trí bắt đầu của dữ liệu (đã trừ BOM)
         return Encoding.BigEndianUnicode;
       }
```

LỚP FILESTREAM-XÁC ĐỊNH ENCODING

```
      if (BOMBytes[0] == 0xff && BOMBytes[1] == 0xfe) {
         stream.Seek (2, SeekOrigin.Begin);
           // Di chuyển về vị trí bắt đầu của dữ liệu (đã trừ BOM)
         return Encoding.Unicode;
       }

       if (BOMBytes[0] == 0xef && BOMBytes[1] == 0xbb && BOMBytes[2]
== 0xbf) {
         stream.Seek (3, SeekOrigin.Begin);
         return Encoding.UTF8;
       }
         if (BOMBytes[0] == 0x2b && BOMBytes[1] == 0x2f &&
BOMBytes[2] == 0x76) {
         stream.Seek (3, SeekOrigin.Begin);
         return Encoding.UTF7;
       }
```

LỚP FILESTREAM-XÁC ĐỊNH ENCODING

```
        if (BOMBytes[0] == 0xff && BOMBytes[1] == 0xfe && BOMBytes[2]
== 0 && BOMBytes[3] == 0) {
          stream.Seek (4, SeekOrigin.Begin);
          return Encoding.UTF32;
        }
        if (BOMBytes[0] == 0 && BOMBytes[1] == 0 && BOMBytes[2] ==
0xfe && BOMBytes[3] == 0xff) {
          stream.Seek (4, SeekOrigin.Begin);
          return Encoding.GetEncoding (12001);
        }
        stream.Seek (0, SeekOrigin.Begin);
        return Encoding.Default;
      }
    }
}
```

LỚP FILESTREAM-XÁC ĐỊNH ENCODING

- ASCII 7 bit: 7 bit biểu diễn các ký tự, đủ biểu diễn bảng chữ cái tiếng Anh (in hoa, thường, số ký tự đặc biệt)
- ASCII 1 byte: 1 byte biểu diễn 1 ký tự.
- UTF-16: 2 byte biểu diễn 1 ký tự
- UTF-32: 4 byte biểu diễn 1 ký tự
- UTF-8: dùng biến để xác định bao nhiêu byte cho mỗi ký tự cụ thể, Mỗi ký tự có thể từ 1 - 6 byte

LỚP FILESTREAM-GHI/ĐỌC TEXT FILE

- Ghi tập tin văn bản (tạo mới, ghi đè)
  - Ghi các bytes BOM, lấy mảng bytes BOM bằng cách gọi encoding.GetPreamble()
  - Encoding và chuyển chuỗi thành mảng bytes
  - Lưu ra stream bằng Write
- Đọc tập tin văn bản
  - Xác định Encoding của file text
  - Đọc từng khối byte vào buffer (mảng byte)
  - Thực hiện Encoding để xác định chuỗi
- Copy file
  - Tạo 2 stream, một để đọc - một để lưu

LỚP FILESTREAM-GHI TEXT FILE

```
string filepath = "/mycode/2.txt";
using (var stream = new FileStream( path:filepath, mode:
FileMode.Create, access: FileAccess.Write, share: FileShare.None))
{
    //Write BOM - UTF8
    Encoding encoding = Encoding.UTF8;
    byte[] bom = encoding.GetPreamble();
    stream.Write(bom, 0, bom.Length);
     string s1 = "Xin chào các bạn sinh viên CTU! \n";
    string s2 = "Ví dụ - ghi file text bằng stream";
     // Encode chuỗi - lưu vào mảng bytes
    byte[] buffer = encoding.GetBytes(s1);
    stream.Write(buffer, 0, buffer.Length);  // lưu vào stream
     buffer = encoding.GetBytes(s2);
    stream.Write(buffer, 0, buffer.Length);  // lưu vào stream
}
```

LỚP FILESTREAM-ĐỌC TEXT FILE

```
string filepath = "/mycode/1.txt"; int SIZEBUFFER = 256;
using (var stream = new FileStream( path:filepath, mode:
FileMode.Open, access: FileAccess.ReadWrite, share:FileShare.Read))
{
    Encoding encoding = GetEncoding(stream);
    Console.WriteLine(encoding.ToString());
    byte[] buffer = new byte[SIZEBUFFER];
    bool endread = false;
    do
    {   int numberRead = stream.Read(buffer, 0, SIZEBUFFER);
        if (numberRead == 0) endread = true;
        if (numberRead < SIZEBUFFER)
        {Array.Clear(buffer, numberRead, SIZEBUFFER - numberRead);
        }
        string s = encoding.GetString(buffer, 0, numberRead);
        Console.WriteLine(s);
    } while (!endread);
}
```

LỚP FILESTREAM-COPY TEXT FILE

```
string filepath_src = "/mycode/1.txt";
string filepath_des = "/mycode/3.txt";
int SIZEBUFFER = 5;   // tăng bộ đệm đọc sẽ nhanh hơn
using (var streamwrite = File.OpenWrite(filepath_des))
using (var streamread = File.OpenRead(filepath_src))
{
    byte[] buffer = new byte[SIZEBUFFER];
    bool endread = false;
    do
    {
        int numberRead = streamread.Read(buffer, 0, SIZEBUFFER);
        if (numberRead == 0) endread = true;
        else {
            streamwrite.Write(buffer, 0, numberRead);
        }
    } while (!endread);
}
```

CÁC LỚP STREAM KHÁC

- Network Stream (Sẽ học ở chương 4)
- Crypto Stream (Xem trong giáo trình)
- Binary Stream
- Text Stream

BINARY VÀ TEXT STREAMS

- Plain text là dạng thức phổ biến dùng trong các stream để con người dễ đọc và soạn thảo. Tương lai sẽ thay bằng XML
- Đặc tính chung của plain text là mỗi đơn vị thông tin được kết thúc với mã phím enter (tổ hợp hai mã UTF8 là 10 và 13 trong C# hay vbCrLf trong VB.NET)

STREAMREADER

```
private void btnRead_Click(object sender, System.EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.ShowDialog();
    FileStream fs = new
    FileStream(ofd.FileName,
    FileMode.OpenOrCreate);
    StreamReader sr = new StreamReader(fs);
    int lineCount = 0;
    while (sr.ReadLine() != null)
    {
        lineCount++;
    }
    fs.Close();
    MessageBox.Show("There are " +
    lineCount + " lines in " + ofd.FileName);
}
```

STREAMREADER

Phương thức /thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của StreamReader
Peek	Trả về ký tự kế tiếp, hoặc giá trị -1 nếu đến cuối stream
Read	Đọc ký tự kế tiếp hoặc một tập các ký tự từ input stream
ReadBlock	Đọc các ký tự từ stream hiện hành và ghi dữ liệu vào bộ đệm, bắt đầu tại vị trí chỉ định
ReadLine	Đọc một dòng ký tự từ stream hiện hành và trả về dưới dạng string
ReadToEnd	Đọc từ vị trí hiện hành đến cuối stream

BINARYWRITER

```
private void btnWrite_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.ShowDialog();
    FileStream fs = new
    FileStream(sfd.FileName, FileMode.CreateNew);
    BinaryWriter bw = new BinaryWriter(fs);
    int[] myArray = new int[1000];
    for (int i = 0; i < 1000; i++)
    {
        myArray[i] = i;
        bw.Write(myArray[i]);
    }
    bw.Close();
}
```

BINARYWRITER

Phương thức /thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của BinaryWriter
Close	Đóng BinaryWriter hiện hành và stream liên quan
Seek	Định vị trí con trỏ trên stream hiện hành
Write	Ghi giá trị vào stream hiện hành
Write7BitEncodedInt	Ghi giá trị số nguyên 32 bit (dạng nén) vào stream hiện hành

SERIALIZATION

NỘI DUNG

- Giới thiệu
- Kỹ thuật chuyển đổi với BinaryFormatter
- Kỹ thuật chuyển đổi với XmlSerializer
- Custom Serialization: Kỹ thuật chuyển đổi lớp đối tượng thông qua lớp giao tiếp ISerializable

GIỚI THIỆU

- Nhiều ứng dụng cần lưu trữ và trao đổi dữ liệu được lưu trong các đối tượng với nhau
- Serialization (chuyển đổi) : là tiến trình biến đổi và tái tạo các đối tượng để chúng có thể được lưu trữ và trao đổi giữa các ứng dụng.
- .NET framework cung cấp nhiều kỹ thuật chuyển đổi để đơn giản hóa tác vụ này

BINARYFORMATTER

BINARYFORMATTER

- Serialize
- Deserialize

BINARYFORMATTER - SERIALIZE

- Tiến trình chuyển một đối tượng thành chuỗi tuần tự các byte để có thể lưu trữ hoặc trao đổi.
- Các bước thực hiện
  - Tạo đối tượng Stream lưu kết quả chuyển đổi
  - Tạo đối tượng BinaryFormatter
  - Gọi phương thức BinaryFormatter.Serialize để chuyển đổi, lưu kết quả vào Stream

BINARYFORMATTER - SERIALIZE

- Demo serialize

```
//tạo đối tượng lưu kết quả chuyển đổi
string data = "This must be store in a file";
FileStream fs = new
FileStream("SerializedString.Data",
FileMode.Create);
//tạo đối tượng BinaryFormatter
BinaryFormatter bf = new BinaryFormatter();
//chuyển đổi và lưu kết quả
bf.Serialize(fs,data);
fs.Close();
```

BINARYFORMATTER - DESERIALIZE

- Tiến trình chuyển chuỗi tuần tự các byte thu được từ quá trình serialize thành đối tượng ban đầu
- Các bước thực hiện
  - Tạo Stream đọc kết quả quá trình serialize
  - Tạo đối tượng BinaryFormatter
  - Tạo đối tượng lưu dữ liệu sau chuyển đổi
  - Gọi phương thức BinaryFormatter.Deserialize để chuyển đổi lại và ép kiểu phù hợp với kiểu của đối tượng ban đầu

BINARYFORMATTER - DESERIALIZE

- Demo deserialize

```
//Tạo stream đọc kết quả thu được từ quá trình
serialize
FileStream fs = new
FileStream("SerializedString.Data",
FileMode.Create);
//Tạo đối tượng  BinaryFormatter
BinaryFormatter bf = new BinaryFormatter();
//tạo đối tượng lưu kết quả chuyển đổi
string data ="";
//chuyển đổi và lưu kết quả
data = (string)bf.Deserialize(fs);
fs.Close();
Console.WriteLine(data);
```

BINARYFORMATTER – SERIALIZABLE CLASS

- Thêm thuộc tính Serializable vào lớp cần chuyển đổi, .NET framework sẽ tự động serialize.
- Có thể kiểm soát quá trình serialize của các lớp để tăng hiệu quả / đáp ứng các yêu cầu của ứng dụng.

```
[Serializable]
public class ShoppingCartItem
{
    public int productId;
    public decimal price;
    public int quantity;
    public decimal total;
}
```

BINARYFORMATTER – SERIALIZABLE CLASS

- Vô hiệu hóa chuyển đổi các thành phần của lớp
  - Dùng cho những giá trị tạm, thuộc tính tính toán
  - Thêm thuộc tính NonSerialized trước khai báo
  - Thành phần NonSerialized không được khởi tạo khi deserialize.

BINARYFORMATTER – SERIALIZABLE CLASS

```
[Serializable]
public class ShoppingCartItem
{
    public int productId;
    public decimal price;
    public int quantity;
    [NonSerialized] public decimal total;
    public ShoppingCartItem(int _procId, decimal
_price, int _quan)
    {
    productId = _procId;
    price = _price;
    quantity = _quan;
    total = price + quantity;
    }
}
```

BINARYFORMATTER – SERIALIZABLE CLASS

- Tự động khởi tạo các thành phần NonSerialized khi deserialize:
  - Thực thi interface IDeserializationCallback
  - Thực thi phương thức IDeserializationCallback.OnDeserialization

BINARYFORMATTER – SERIALIZABLE CLASS

```
[Serializable]
public class ShoppingCartItem: IDeserializationCallback
{
    …
    [NonSerialized] public decimal total;
    public ShoppingCartItem2(int _procId, decimal _price,
int _quan)
    {
        …
        total = price + quantity;
    }
    void IDeserializationCallback.OnDeserialization(object
sender)
    {
        total = price * quantity;
    }
}
```

BINARYFORMATTER – SERIALIZABLE CLASS

- Tương thích phiên bản
  - Phát sinh ngoại lệ khi deserialize đối tượng được serialize ở phiên bản trước của ứng dụng
    - Thêm thành phần mới vào lớp, deserialize đối tượng được serialize trước đó mà không có thành phần mới
  - Giải pháp:
    - Thực thi custom serialization
    - Thêm thuộc tính OptionalField trước thành phần mới có thể gây không tương thích phiên bản
  - Thành phần OptionalField không được khởi tạo khi deserialize

BINARYFORMATTER – SERIALIZABLE CLASS

```
[Serializable]
public class ShoppingCartItem
{
    public int productId;
    public decimal price;
    public int quantity;
    [NonSerialized] public decimal total;
    [OptionalField] public bool taxable;
}
```

BINARYFORMATTER – SERIALIZABLE CLASS

- Lưu ý khi xử lý tương thích phiên bản
  - Không bỏ thành phần serialize
  - Không áp dụng thuộc tính NonSerialized cho thành phần không áp dụng thuộc tính này ở phiên bản trước
  - Không đổi tên/kiểu của thành phần serialize
  - Áp dụng OptionalField khi khi thêm mới thành phần serialize
  - Khi bỏ thuộc tính NonSerialized cho thành phần không áp dụng thuộc tính này ở phiên bản trước, dùng OptionalField
  - Với các thành phần OptionalField, thực thi IDeserializationCallback để khởi tạo giá trị ban đầu

BINARYFORMATTER – SERIALIZABLE CLASS

- Chọn định dạng chuyển đổi
  - .NET Framework cung cấp 2 phương thức định dạng dữ liệu chuyển đổi:
    - BinaryFormatter: định dạng hiệu quả nhất để serialize các đối tượng sẽ chỉ được sử dụng bởi các ứng dụng .NET
    - SoapFormatter:
      - Định dạng XML, là cách thức đáng tin cậy để serialize các đối tượng được trao đổi trên môi trường mạng(được sử dụng cho các ứng dụng ngoài .NET)
      - Có khả năng vượt tường lửa tốt hơn BinaryFormatter

XMLSERIALIZER

XMLSERIALIZER

- Serialize
- Deserialize

XMLSERIALIZER

- XML : định dạng tài liệu văn bản chuẩn cho việc lưu trữ và trao đổi thông tin
- .NET Framework cung cấp nhiều thư viện hỗ trợ đọc, ghi file XML, chuyển đổi các đối tượng sang định dạng XML và ngược lại.
- Sử dụng XML Serialization vì:
  - Khả năng giao tiếp rộng
  - Thân thiện với người dùng, dễ dàng đọc và hiệu chỉnh
  - Khả năng tương thích phiên bản cao

XMLSERIALIZER

- Hạn chế của XML Serialization
  - Chỉ có thể chuyển đổi các dữ liệu public
  - Không thể chuyển đổi đối tượng đồ thị, biểu đồ

XMLSERIALIZER - SERIALIZE

- Các bước thực hiện
  - Tạo đối tượng Stream/ TextWriter/ XmlWriter để lưu kết quả chuyển đổi
  - Tạo đối tượng XmlSerializer với kiểu của đối tượng cần chuyển đổi
  - Gọi phương thức XmlSerializer.Serialize để chuyển đổi và lưu kết quả

XMLSERIALIZER - SERIALIZE

- Demo serialize

```
//tạo đối tượng lưu kết quả chuyển đổi
FileStream fs = new
FileStream("SerializedDate.xml",
FileMode.Create);
//tạo đối tượng XmlSerializer
XmlSerializer xs = new
XmlSerializer(typeof(DateTime));
//chuyển đổi và lưu kết quả
xs.Serialize(fs, System.DateTime.Now);
fs.Close();
```

XMLSERIALIZER - DESERIALIZE

- Các bước thực hiện
  - Tạo đối tượng Stream/ TextWriter/ XmlWriter để đọc kết quả chuyển đổi của quá trình serialize
  - Tạo đối tượng XmlSerializer với kiểu của đối tượng cần chuyển đổi
  - Gọi phương thức XmlSerializer.Deserialize để tái tạo đối tượng ban đầu, ép kiểu dữ liệu cho phù hợp

XMLSERIALIZER - DESERIALIZE

- Demo deserialize

```
//Tạo stream đọc kết quả thu được từ quá trình
serialize
FileStream fs = new
FileStream("SerializedDate.xml", FileMode.Open);
//Tạo đối tượng XmlSerializer
XmlSerializer xs = new
XmlSerializer(typeof(DateTime));
//Thực thi tái tạo đối tượng
DateTime time = (DateTime)xs.Deserialize(fs);
fs.Close();
Console.WriteLine("Day: "+time.DayOfWeek +",Time: "
+time.TimeOfDay.ToString());
```

XMLSERIALIZER – SERIALIZABLE CLASS

- Tạo lớp có thể serialize
  - Khi chuyển đổi các lớp đáp ứng yêu cầu Xml serialization nhưng không có bất kỳ thuộc tính Xml Serialization nào, .NET sẽ dùng định dạng mặc định hiện có để đáp ứng yêu cầu của nhiều người dùng.
  - Tên của Xml element : phụ thuộc vào tên lớp và tên thành phần
  - Mỗi thành phần được chuyển đổi thành một Xml element riêng biệt.

XMLSERIALIZER - SERIALIZABLE CLASS

- Tạo lớp có thể chuyển đổi

Serialize

```
public class ShoppingCartItem
{
    public int productId;
    public decimal price;
    public int quantity;
    public decimal total;
    public ShoppingCartItem()
    {}
}
```

```
<?xml version="1.0" ?>
<ShoppingCartItem>
<productId>100</productId>
<price>12.0</price>
<quantity>5</quantity>
</ShoppingCartItem>
```

XMLSERIALIZER - SERIALIZABLE CLASS

- Tạo lớp có thể serialize
  - Nếu chỉ tạo tài liệu XML mô tả lớp, kết quả quá trình chuyển đổi được coi là đủ
  - Nếu muốn tạo tài liệu XML đáp ứng những yêu cầu cụ thể: cần can thiệp vào quá trình chuyển đổi tài liệu XML có định dạng theo yêu cầu.

XMLSERIALIZER - SERIALIZABLE CLASS

- Tạo lớp có thể chuyển đổi

```
[XmlRoot("CartItem")]
public class ShoppingCartItem
{
    [XmlAttribute] public int productId;
    public decimal price;
    public int quantity;
    [XmlIgnore] public decimal total;
    public ShoppingCartItem(){}
}
```

Serialize

```
<?xml version="1.0" ?>
<CartItem productId="100">
<price>12.0</price>
<quantity>5</quantity>
</CartItem>
```

XMLSERIALIZER – SERIALIZE DATASET

- Chuyển đổi 1 đối tượng DataSet

```
private void SerializeDataSet(string filename)
{
    XmlSerializer ser = new XmlSerializer(typeof(DataSet));
    DataSet ds = new DataSet("myDataSet");
    DataTable dt = new DataTable("table1");
    dt.Columns.Add(new DataColumn("thing"));
    ds.Tables.Add(dt);
    for (int i = 0; i < 5; i++)
    {
        DataRow r = dt.NewRow();
        r[0] ="Thing "+i;
        dt.Rows.Add(r);
    }
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, ds);
    writer.Close();
}
```

CUSTOM SERIALIZATION

CUSTOM SERIALIZATION

- Kỹ thuật chuyển đổi lớp đối tượng thông qua lớp giao tiếp ISerializable
- Tiến trình điều khiển việc chuyển đổi và tái tạo đối tượng, đảm bảo tương thích phiên bản

CUSTOM SERIALIZATION

- Thực thi custom serialization
  - Serialize trong .NET rất uyển chuyển và có thể tùy biến để đáp ứng yêu cầu phát triển ứng dụng.
  - Có thể override quá trình serialize xây dựng sẵn trong .NET bằng cách thực thi interface ISerializable và khai báo thuộc tính Serializable cho lớp.
  - Thực thi interface ISerializable sẽ gọi
    - **Phương thức GetObjectData trong quá trình serialize**
    - **Phương thức khởi tạo đặc biệt trong quá trình deserialize**

CUSTOM SERIALIZATION-DEMO

```
[Serializable]
public class ShoppingCartItem : ISerializable
{
    public Int32 productId;
    public decimal price;
    public Int32 quantity;
    [NonSerialized] public decimal total;
    public ShoppingCartItem(Int32 _productId,
    decimal _price, int _quantity)
    {
        productId = _productId;
        price = _price;
        quantity = _quantity;
        total = price * quantity;
    }
```

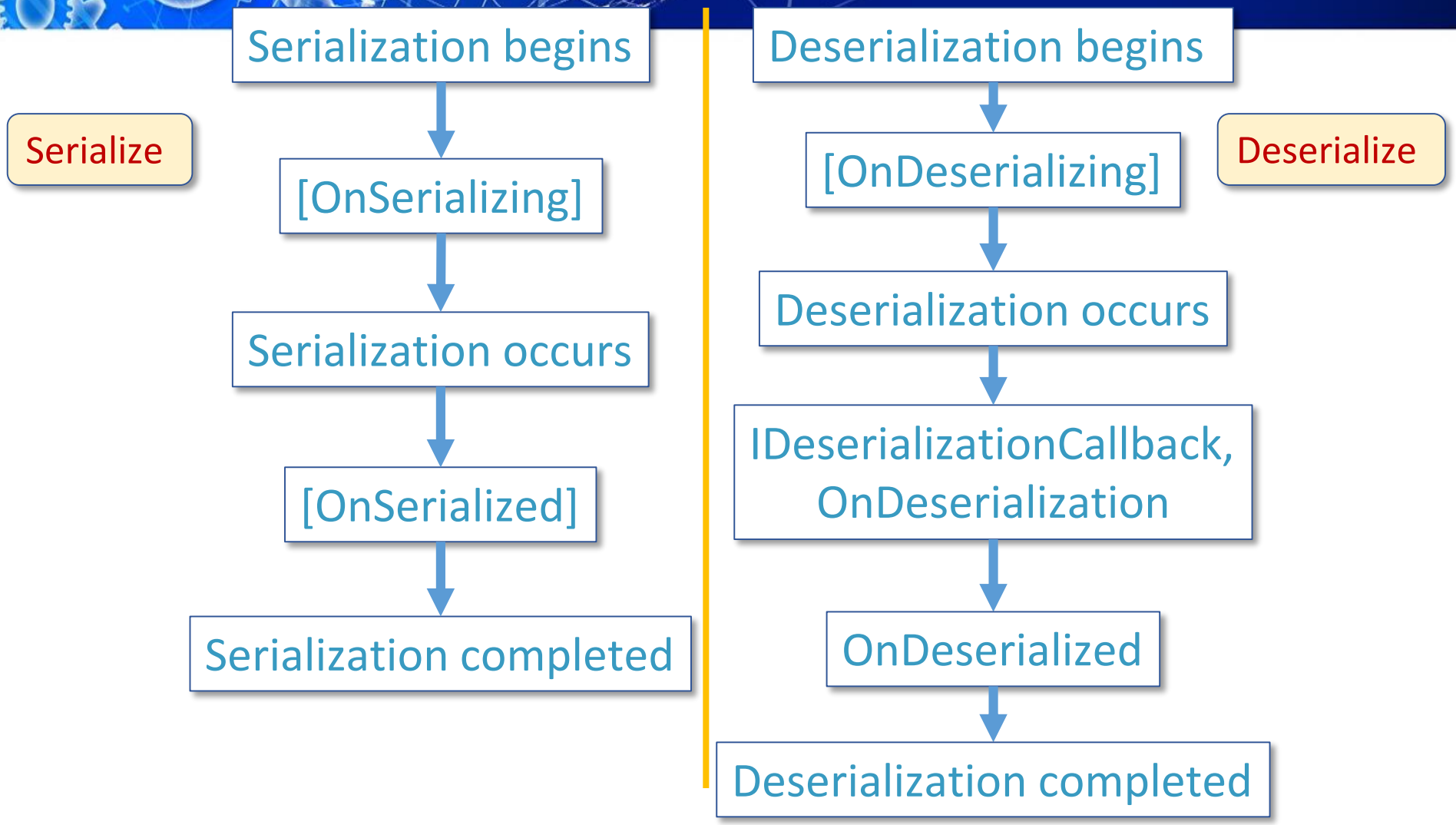
CUSTOM SERIALIZATION-DEMO

```
protected ShoppingCartItem(SerializationInfo info,
StreamingContext context)
    {
        productId = info.GetInt32("Product ID");
        price = info.GetDecimal("Price");
        quantity = info.GetInt32("Quantity");
        total = price * quantity;
    }
    [SecurityPermissionAttribute(SecurityAction.Demand,
SerializationFormatter = true)]
    public virtual void
    GetObjectData(SerializationInfo
    info, StreamingContext context)
    {
        info.AddValue("Product ID", productId);
        info.AddValue("Price", price);
        info.AddValue("Quantity", quantity);
    }
}
```

CUSTOM SERIALIZATION

- Các sự kiện trong Serialization
  - Serializing: phát sinh trước khi serialize bắt đầu
    - **Thêm thuộc tính OnSerializing trước phương thức**
  - Serialized: phát sinh sau khi serialize hoàn tất
    - **Thêm thuộc tính OnSerialized trước phương thức**
  - Deserializing: phát sinh trước khi deserialize bắt đầu
    - **Thêm thuộc tính OnDeserializing trước phương thức**
  - Deserialized: phát sinh sau khi serialize kết thúc
    - **Thêm thuộc tính OnDeserialized trước phương thức**

CUSTOM SERIALIZATION - EVENT



CUSTOM SERIALIZATION

- **Các sự kiện trong Serialization**
  - Các sự kiện này là cách tốt nhất và dễ dàng nhất để điều khiển tiến trình chuyển đổi
    - Không can thiệp vào serialization stream
    - Cho phép hiệu chỉnh đối tượng trước và sau serialize
  - Yêu cầu cho các phương thức xử lý các sự kiện
    - **Có tham số là đối tượng StreamingContext**
    - **Không trả về kết quả**

CUSTOM SERIALIZATION-DEMO

```
[Serializable]
public class ShoppingCartItem : ISerializable
{
    public Int32 productId;
    public decimal price;
    public Int32 quantity;
    public decimal total;
    public void GetObjectData(SerializationInfo info,
StreamingContext context)
    {throw new NotImplementedException();}
    [OnSerializing] void CalculateTotal(StreamingContext sc)
    {total = price * quantity;}
    [OnDeserializing] void CheckTotal(StreamingContext sc)
    {
        if (total == 0)
        {CalculateTotal(sc);}
    }
}
```

CUSTOM SERIALIZATION

- **Thay đổi ngữ cảnh chuyển đổi**
  - Khi serialize 1 đối tượng: không cần quan tâm đích đến
  - Trong vài trường hợp: serialize và deserialize sẽ khác nhau phụ thuộc vào đích đến
  - Cấu trúc StreamingContext cung cấp thông tin đích đến của đối tượng được serialize cho lớp xử lý thực thi interface ISerializable
    - StreamingContext có 2 thuộc tính:
      - **Context: tham chiếu đến đối tượng chứa thông tin ngữ cảnh**
      - **state: 1 tập cờ hiệu chỉ ra nguồn đích của đối tượng đang serialize và deserialize**

CUSTOM SERIALIZATION

- **Các cờ hiệu của thuộc tính State**
  - StreamingContextStates. CrossProcess
  - StreamingContextStates. CrossMachine
  - StreamingContextStates. File
  - StreamingContextStates. Persistence
  - StreamingContextStates. Remoting
  - StreamingContextStates. Other
  - StreamingContextStates. Clone
  - StreamingContextStates. CrossAppDomain
  - StreamingContextStates. All

CUSTOM SERIALIZATION

FIELDS	
All	Specifies that the serialized data can be transmitted to or received from any of the other contexts.
Clone	Specifies that the object graph is being cloned. Users can assume that the cloned graph will continue to exist within the same process and be safe to access handles or other references to unmanaged resources.
CrossAppDomain	Specifies that the source or destination context is a different AppDomain. (For a description of AppDomains, see Application Domains).
CrossMachine	Specifies that the source or destination context is a different computer.
CrossProcess	Specifies that the source or destination context is a different process on the same computer.

CUSTOM SERIALIZATION

FIELDS	
File	Specifies that the source or destination context is a file. Users can assume that files will last longer than the process that created them and not serialize objects in such a way that deserialization will require accessing any data from the current process.
Other	Specifies that the serialization context is unknown.
Persistence	Specifies that the source or destination context is a persisted store, which could include databases, files, or other backing stores. Users can assume that persisted data will last longer than the process that created the data and not serialize objects so that deserialization will require accessing any data from the current process.
Remoting	Specifies that the data is remoted to a context in an unknown location. Users cannot make any assumptions whether this is on the same computer.