ソフトウェア開発実践セミナー

Windowsプログラミングとネットワーク

金子　勇　kaneko@ipl.t.u-tokyo.ac.jp
土村展之　tutimura@mist.t.u-tokyo.ac.jp
情報理工学系研究科　数理情報学専攻

2002年12月4日 (第8回)

全体の流れ

1. Windows開発現状
2. Webアプリ開発環境
3. Windows開発環境
4. Windowsプログラミング基礎
5. ソケットプログラミング (WinSock)

1. ソフトウェア開発現状

- インターネット現状
  - サーバ側：　　UNIXが有利
  - クライアント側：Microsoft Windowsが一般的

使ってもらえるアプリケーション

- Windows上で快適に動作する
  Windowsネイティブアプリが好まれる

Windowsアプリ開発現状

- Windowsネイティブアプリから.NETアプリに移行中
  - Windows 95系

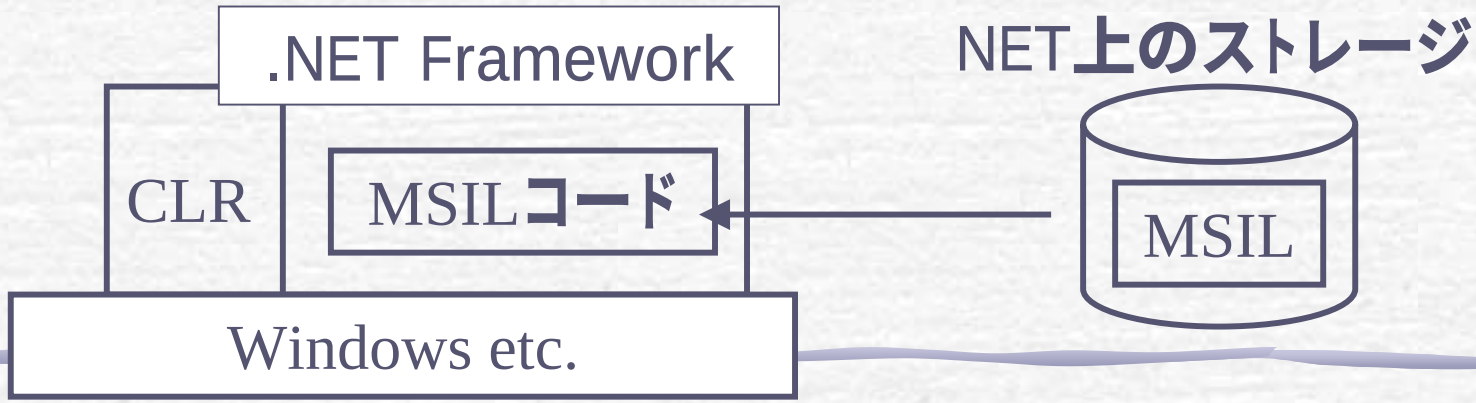
    DOS → Win3.1(Win16API) → Win95(Win32cAPI) → Me
  - Windows NT系

    Windows NT (Win32API) → XP → Windows.NET

- Win32APIを用いるWindowsネイティブアプリは無くならない

Windows.NET

- .NETはJavaに似た考えがベース (VMを基盤とする)
- .NETアプリは.NET Frameworkで動作
- MSIL(Microsoft Intermediate Language)という中間コード形式でコードを作成し CLR(Common Language Runtime)で駆動させる
- Javaと違い中間コード生成の記述言語は任意



Windows開発環境

- **統合開発環境**（IDE）**系**
  - Visual Studio（Visual C++）, Code Warrior
- RAD（Rapid Application Development）**系**
  - Visual Basic, Visual C#, Delphi, C++ Builder
- Web**アプリ開発環境**
  - IIS（Microsoft**社の**Web**サーバ**）+ASP
  - Visual InterDev

2．Webアプリ開発環境

- IIS (Internet Information Server)
  - ASP(Active Server Pages)**による動的**Web**ページ**
  - **開発環境として**Visual InterDev
    → ASP.NET（VisualStudio.NET**に統合**）

ASP (Active Server Pages)

- 動的なWebページを実現するメカニズム
- Webページ内にスクリプトを記述
  - JavaScript
  - VBScript
- IISにWebブラウザからアクセス
  → スクリプトを解釈してHTMLを出力

http://localhost/test.asp

```
<%@ Language=VBScript %>
<HTML><BODY>
<%      for i = 1 to 10
                %> <p> Hello World ! <%
                Response.Write(i)
                %> </p> <%
        next
%>
</BODY></HTML>
```

IIS+ASPの特徴

- 動的なWebページ記述が容易
- 高速な動作
- SQLを用いたDBアクセス（ODBC経由）が容易
- あまり複雑になるとHTML記述が雑多
- イントラネット向けには良い

3．Windows開発環境

- Microsoft製品(VisualStudio, 6.0→.NET)
  - VisualBasic (RAD, BASIC)
  - VisualC# (RAD, C#)
  - VisualC++ (IDE, C++)
- Borland製品
  - Delphi (RAD, Object Pascal)
  - C++Builder (RAD, C++)
  - Kylix (DelphiのUNIX版)

Windows上での開発環境選択

- **速度重視なアプリ、ゲーム、市販アプリ**
  → Visual C++, Code Warrior
- **市販のコンポーネントを多用**
  → Visual Basic
- **プログラミング入門**
  → Java, Visual C#
- **開発効率重視で比較的複雑なGUIのアプリ作成**
  → Delphi, C++Builder, Visual C#

Microsoft Visual Basic

- Windows初期から存在する代表的なRAD開発環境
- 市販の豊富なコンポーネントが利用可能
- 受注によるシステム開発で多用
- 中間コード方式（.NETだとCLR動作）
- 実行時ランタイムが必要
- 参考文献が非常に多い
- VB6.0→VB.NETで大きく変わった

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Private components As System.ComponentModel.Icontainer
    Public Sub New()
        MyBase.New()
        InitializeComponent()
    End Sub
    Private components As System.ComponentModel.IContainer
        components = New System.ComponentModel.Container()
        Me.Text = "Form1"
    End Sub
End Class
```

Microsoft Visual C#

- VisualStudio.NETで新登場
- C#はJavaに良く似た言語
- Javaからの移行が容易
- CLRで駆動、速度は比較的遅い
- 今後、他開発環境からの移行が進むと予測される
- .NET Frameworkがインストールされていないと生成されたアプリが動作しない

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
public class Form1 : System.Windows.Forms.Form
{
        private System.ComponentModel.Container components = null;
        public Form1() { InitializeComponent(); }
        private void InitializeComponent()
        {
                this.components = new System.ComponentModel.Container();
                this.Size = new System.Drawing.Size(300,300);
                this.Text = "Form1";
        }
        static void Main() { Application.Run(new Form1()); }
}
```

Boland Delphi, C++Builder

- それぞれObjectPascal, C++ベースのRAD環境
- Windowsネイティブコードを生成するため
  比較的高速なアプリケーションを生成できる
- RAD環境であるため開発効率が非常に良い
- 専用のクラスライブラリを用いる (VCL)
- ネット上で多くのコンポーネントが提供
- 他にランタイムの必要の無い.exeが作成できる

C++Builder実演

- コンポーネントをコンテナに置くだけでGUIコード記述
  (2Way-Tool方式)
- コンポーネントの対応イベント毎にコードを記述
- オブジェクト指向
  - 独自のクラスライブラリ (VCL)
  - 各オブジェクトのメンバにアクセス→ 属性を変更やメソッド起動
  - 2Way-Tool方式により動的にプロパティ変更

```
//---------------------------------------------------------------------------
void __fastcall TForm1::OpenButtonClick(TObject *Sender)
{
        if (OpenDialog1->Execute()) {
                ListBox1->Items->LoadFromFile(OpenDialog1->FileName);
        }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ListBoxClick(TObject *Sender)
{
        Edit1->Text = ListBox1->Items->Strings[ListBox1->ItemIndex];
}
//---------------------------------------------------------------------------
```

Microsoft Visual C++

- WindowsAPIを直接用いる低レベルAPI開発向き
- .NETでは唯一Windowsネイティブアプリを作成可能
- 他にランタイムの必要の無い.exeが作成できる
- 生成されるアプリは最も動作が高速
- MSILコードも生成可能(Managed C++)
- 市販のパッケージソフトの多くがこれで作成されている
- Visualと付いているがRAD環境ではない (MFC他を使用)
- 小規模、複雑なGUIを持つアプリ開発には非効率

Visual C++実演

- クラスライブラリを用いてアプリ構築 (MFC)
- 雛形を自動生成してコードを追加
  (フレームワーク手法)
- ダイアログなどのリソースは別にリソースエディタで編集
- フレームワーク手法:
  - オブジェクト指向の応用
  - 定型的なコード部を抽象クラスとして記述
  - ユーザは独自の具象クラスを継承で生成して定型の差分だけ記述

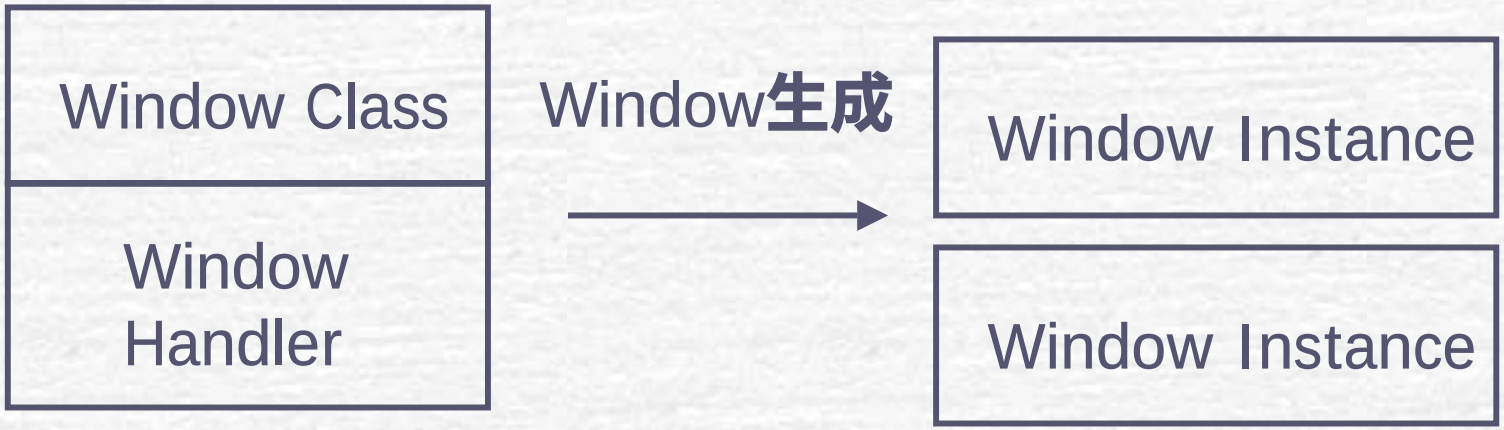
付録1 (MFCプログラム)

4．Windowsアプリの基本

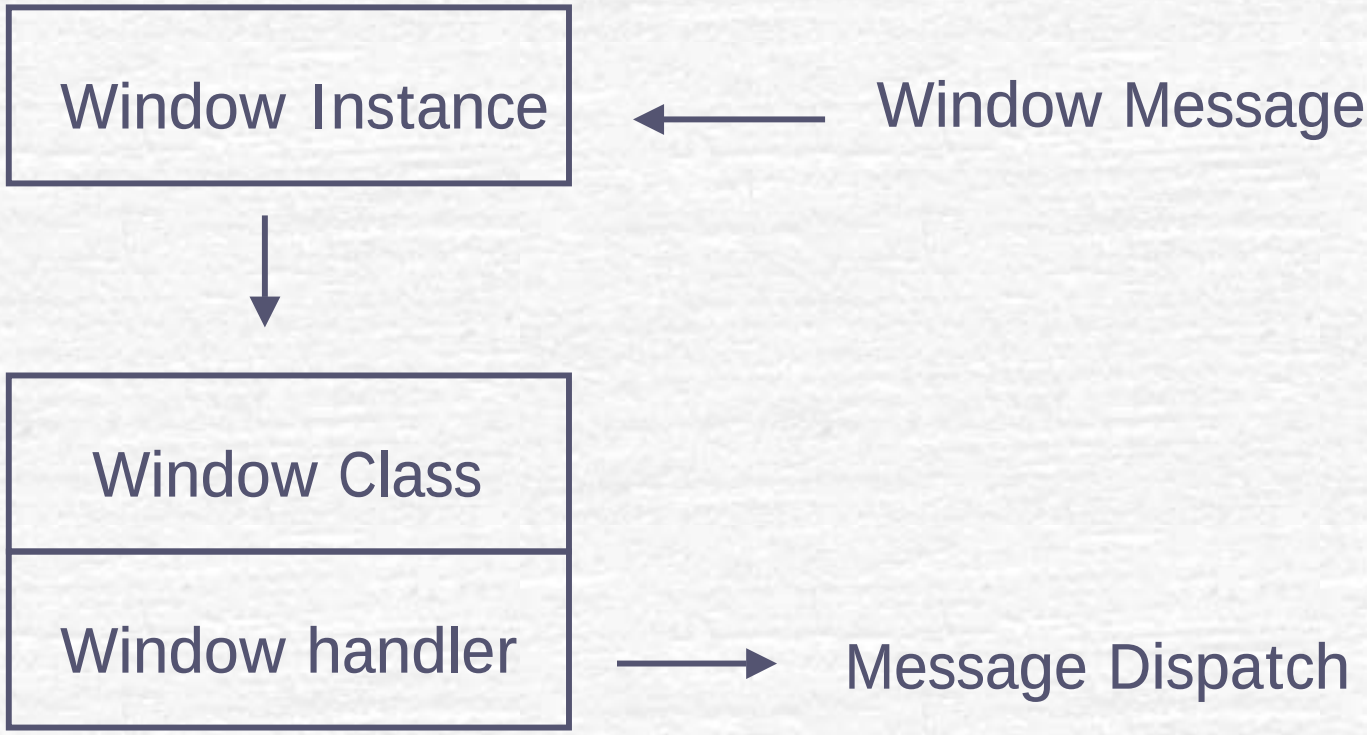
- Windowsプログラムは WinMain関数から開始
- ウインドウの種類毎に Window Class作成
- Window Class毎に Window Handler登録
- それぞれの Window Handlerでメッセージ処理

付録2（Win32APIプログラム）

Window ClassとWindow Handler



Window Messageの Dispatch



Windows Message

- イベントの発生を表すメッセージ
- ウィンドウ毎に受信キューを持つ
- 種類毎にMessage ID (WM_・・・) を持つ
- イベント発生時には種類と引数二つが渡される
  - Firstメッセージ(wParam, 符号無し32ビット整数)
  - Lastメッセージ(lParam , 符号無し32ビット整数)

- システムで使っていないMessage IDを
  USER側で自由な用途に使用可

代表的なWindowメッセージ

- WM_LBUTTONDOWN (マウスイベント)
- WM_KEYDOWN (キーイベント)
- WM_CREATE (ウインドウ作成)
- WM_CLOSE (ウインドウ状態変化)
- WM_INITDIALOG (ダイアログ初期化)
- WM_PAINT (再描画指示)
- WM_COMMAND (メニューなどでのコマンド選択)

Window Handler内部でswicth処理を行う

5．Windowsネットアプリ開発

- 各開発環境に依存
- WinSockをカプセル化したクラスライブラリ
  （MFC→CSocket、Delphi→TSocket）
- 内部でのネットワーク処理はWinSock

- 結局最後はWinSock API直叩き

WinSock

- UNIX SocketをWindowsで使うためのAPI群
  - socket()
  - bind ()
  - listen ()
  - accept ()
  - connect ()
  - read()
  - write()
  - closesocket()
- 関数の利用法はUNIXとまったく同じ

Windowsでの注意点

- ウインドウシステム上で動作する
  - プロセス（スレッド）がロックする
    → アプリケーションが操作不能

  - スレッドをロックさせない必要
    → 一つの処理は数msec以内で処理

ロックする可能性のあるWinSock関数

- read() → **受信時にデータが無い**
- write() → **相手の受信バッファが限界**
- accept → **クライアントが接続するまで**
- connect → **サーバーに接続するまで**

スレッドをロックさせない方法

- 通信スレッドをGUIスレッドと分離
  - GUIメッセージ処理とソケット処理スレッド
- 同期I/O (ブロッキング関数)
  - UNIX同様、Select関数でポーリング
- 非同期I/O（ノンブロッキング関数)
  - Windowsメッセージを用いる

WinSock非同期I/O

- UNIX互換の関数はブロッキング動作
- 変化が生じたらWindowsMessage送信

→非同期I/O

- Windowsプログラミングとの相性が良い
- WSAで始まる関数群

WSA系WinSock関数

- WSAStartup (初期化)
- WSACleanup (終了)
- WSAAsyncSelect (非同期select)

- WSAAsyncSelectでソケットイベント発生をWindows Message送信に置き換える

WSAAsyncSelect

- **WSAAsyncSelect** (s, hWnd, wMsg, lEvent)
  - SOCKET s　(**ソケット識別子**)
  - HWND hWnd　(Window**ハンドル**)
  - unsigned int wMsg (**ユーザメッセージ**ID)
  - long lEvent (**監視イベント**)

非同期acceptの例

```
WSAAsyncSelect ( s, hWnd, WM_USER + 1,  FD_ACCEPT);
bind (acceptSocket, &sin, sizeof(sin));
listen (acceptSocket, 32);
   ’
   ’
LRESULT CALLBACK WndProc
    (HWND hwnd, UINT wMessage, WPARAM wParam, LPARAM lParam)
{
    switch (wMessage) {
    case WM_USER + 1:  // クライアント接続でメッセージ受信
            newSocket = accept (acceptSocket, ’’’
```

参考文献

[1] **インターネットのための**Winsock**プログラミング**, ISBN4-7741-0371-3, DaveRoberts**著**, **撲 易道訳**, **技術評論社**