# Разработка .NET приложений

Лекция X

Вызов неуправляемого кода - P/Invoke

# Взаимодействие с неуправляемым кодом

- Взаимодействие с **API** операционной системы
- Взаимодействие с нативным **API** других компонентов
- Взаимодействие с **COM** компонентами (для **Windows**)

- P/Invoke - **platform invoke** - технология, позволяющая обращаться из управляемого кода
  - к функциям в неуправляемых библиотеках
  - к неуправляемым структурам
  - Задавать/передавать **callback** в неуправляемых библиотеках

- C++/CLI - позволяет создать оболочку для нативного кода C++ и делают его доступными из программ .NET. Позволяет управляемым и неуправляемым конструкциям совместно существовать и взаимодействовать в одной сборке и даже в одном файле

# Задание вызываемой функции

- Пространства имен: **System** и **System.Runtime.InteropServices**

```
[DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);
```

- DllImport сообщает **CLR**, что требуется загрузить неуправляемую библиотеку DLL (в данном случае user32.dll)
- Метод должен иметь точно такое же имя и такую же сигнатуру, что и неуправляемый метод
- **extern** - сообщает **CLR**, что это внешний метод и что при его вызове **CLR** должна найти его в библиотеке DLL, указанной в атрибуте DllImport.

- Для **Mac OS**

```
[DllImport("libSystem.dylib")]
private static extern int getpid();
```

- Для **Linux**

```
[DllImport("libc.so.6")]
private static extern int getpid();
```

# Вызов заданной функции

- Как вызов обычного .NET метода
- Windows

```
[DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption,
uint uType);

public static void Main(string[] args)
{
    MessageBox(IntPtr.Zero, "Native message box", "Hello", 0);
}
```

- Mac OS

```
[DllImport("libSystem.dylib")]
private static extern int getpid();

public static void Main(string[] args)
{.
    int pid = getpid();
     Console.WriteLine(pid);
}
```

- Linux

```
[DllImport("libc.so.6")]
private static extern int getpid();

public static void Main(string[] args)
{
    int pid = getpid();
    Console.WriteLine(pid);
}
```

# Маршалинг типов

- Маршалинг — это процесс преобразования типов при переходе от управляемого кода к машинному и обратно
- Необходимость в маршалинге вызвана различием типов в управляемом и неуправляемом коде

- Маршалинг простых типов

| Тип .NET | Нативный тип |
|---|---|
| byte | uint8_t |
| sbyte | int8_t |
| short | int16_t |
| ushort | uint16_t |
| int | int32_t |
| uint | uint32_t |
| long | int64_t |
| ulong | uint64_t |
| char | Тип char или char16_t в зависимости от кодировки CharSet P/Invoke или структуры. |
| string | Тип char* или char16_t* в зависимости от кодировки CharSet P/Invoke или структуры.. |
| System.IntPtr | intptr_t |
| System.UIntPtr | uintptr_t |
| Типы указателей .NET (за исключением void*) | void* |
| Тип, производный от System.Runtime.InteropServices.SafeHandle | void* |
| Тип, производный от System.Runtime.InteropServices.CriticalHandle | void* |
| bool | Тип Win32 BOOL |
| decimal | Структура COM DECIMAL |
| Делегат .NET | Нативный указатель на функцию |
| System.DateTime | Тип Win32 DATE |
| System.Guid | Тип Win32 GUID |

# Маршалинг char, string и StringBuilder

- Зависит от значения, заданного в поле CharSet

  - P/Invoke

```
[DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);
```

  - Для структуры (для всех полей типа **string** и **char**)

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet. Ansi)]
public struct DefaultString
{
   public string str;
}
```

| Значение CharSet | Windows | Mono в Unix (.NET Core 3.0+) |
|---|---|---|
| Ansi | char (системная кодовая страница Windows (ANSI) по умолчанию) | char (UTF-8) |
| Unicode | wchar_t (UTF-16) (LPWSTR ) | char16_t (UTF-16) |
| Auto | wchar_t (UTF-16) (LPSTR ) | char (UTF-8) |

# Маршалинг string

- Если нужно настроить маршалиг конкретной строки в структуре, то можно использовать атрибут **MarshalAs**

- Для задания ANSI
  - public struct AnsiString
  - {
  - [MarshalAs(UnmanagedType.LPStr)]
  - public string str;
  - }

- Для задания **UTF-16**
  - public struct UnicodeString
  - {
  - [MarshalAs(UnmanagedType.LPWStr)]
  - public string str;
  - }

- Для задания **UTF-8**
  - public struct UTF8String
  - {
  - [MarshalAs(UnmanagedType.LPUTF8Str)]
  - public string str;
  - }

- Для **COM API** может быть полезно маршалировать строку как **BSTR**
  **[MarshalAs(UnmanagedType.BStr)]**
- Для API на базе WinRT может быть полезно маршалировать строку как **HSTRING**
  **[MarshalAs(UnmanagedType.HString)]**
- Если API требует передавать строку на месте в структуре (также указывается размер строки)
  **[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4)]**

# Параметры атрибута DllImport

| Параметр | Значение по умолчанию | Рекомендация | Подробнее |
|---|---|---|---|
| PreserveSig | true | Оставьте значение по умолчанию | Если для параметра явно задано значение false, в результате сбоя вызова кидается исключение |
| SetLastError | false | в зависимости от API | Установите true, если для получения значения в API используется GetLastError и Marshal.GetLastWin32Error. Если API устанавливает условие, указывающее на ошибку, перед тем как выполнить другие вызовы, получите информацию об ошибке, чтобы избежать ее непреднамеренной перезаписи. |
| CharSet | CharSet.None с откатом на CharSet.Ansi | Если в определении есть строки или символы, используйте CharSet.Unicode или CharSet.Ansi в явном виде | Указывает на маршалинг строк.<br>Также влияет на поиск нужной функции при задании ExactSpelling в false. |
| ExactSpelling | false | true | Если задать true, можно немного повысить производительность — среда выполнения не будет искать другие имена функций с суффиксом "A" или "W" в зависимости от значения параметра CharSet ("A" для CharSet.Ansi и "W" для CharSet.Unicode). |

# Передача классов и структур

- CLR управляет физическим размещением полей данных класса или структуры в управляемой памяти
- Если необходимо передать тип в неуправляемый код, нужно использовать StructLayoutAttribute
- Атрибут StructLayoutAttribute - позволяет управлять физическим размещением полей данных класса или структуры в памяти
  - LayoutKind - управляет макетом объекта при его экспорте в неуправляемый код
    - Auto - CLR автоматически выбирает размещение для членов объекта в неуправляемой памяти. Доступ к объектам с LayoutKind.Auto не может быть предоставлен вне управляемого кода. Попытка выполнить такую операцию вызовет исключение. Значение по умолчанию для классов
    - Sequential - члены объекта располагаются последовательно, в порядке своего появления при экспортировании в неуправляемую память. Члены располагаются в соответствии с компоновкой, заданной в Pack, и могут быть несмежными. Значение по умолчанию для структур
    - Explicit - точное положение каждого члена объекта в неуправляемой памяти управляется явно в соответствии с настройкой поля Pack. Каждый член должен использовать атрибут FieldOffsetAttribute для указания положения этого поля внутри типа.
  - Pack - управляет выравниванием полей данных для класса или структуры в памяти
  - Size - указывает абсолютный размер класса или структуры

# Передача классов и структур

```
enum Bool
{
  False = 0,
  True
};

[StructLayout(LayoutKind.Sequential)]
public struct Point
{
  public int x;
  public int y;
}

[StructLayout(LayoutKind.Explicit)]
public struct Rect
{
  [FieldOffset(0)] public int left;
  [FieldOffset(4)] public int top;
  [FieldOffset(8)] public int right;
  [FieldOffset(12)] public int bottom;
}
```

```
[StructLayout(LayoutKind.Explicit, Size=16, CharSet=CharSet.Ansi)]
public class SystemTime
{
    [FieldOffset(0)]  public ushort wYear;
    [FieldOffset(2)]  public ushort wMonth;
    [FieldOffset(4)]  public ushort wDayOfWeek;
    [FieldOffset(6)]  public ushort wDay;
    [FieldOffset(8)]  public ushort wHour;
    [FieldOffset(10)] public ushort wMinute;
    [FieldOffset(12)] public ushort wSecond;
    [FieldOffset(14)] public ushort wMilliseconds;
}

internal static class NativeMethods
{
  [DllImport("kernel32.dll")]
  internal static extern void GetSystemTime([MarshalAs(UnmanagedType.LPStruct)]MySystemTime st);
};

public static void Main() {
SystemTime sysTime = new SystemTime();
    NativeMethods.GetSystemTime(sysTime);
    Console.WriteLine("The System time is {0}/{1}/{2} {3}:{4}:{5}", sysTime.wDay,
          sysTime.wMonth, sysTime.wYear, sysTime.wHour, sysTime.wMinute, sysTime.wSecond);
}
```

# Маршалинг bool полей

- По умолчанию логические значения маршалируются как собственное 4-байтное значение Win32 BOOL

```
public struct WinBool
{
   public bool b;
}
public struct WinBool
{
   [MarshalAs(UnmanagedType.Bool)]   public bool b;
}
```

- UnmanagedType.U1 или UnmanagedType.I1 указывает **CLR**, что поле нужно маршалировать как 1-байтный нативный тип bool

```
public struct CBool
{
   [MarshalAs(UnmanagedType.U1)]    public bool b;
}
```

- В Windows значение UnmanagedType.VariantBool задает маршалинг **bool** поля в 2-байтное значение VARIANT_BOOL:

```
public struct VariantBool
{
   [MarshalAs(UnmanagedType.VariantBool)]    public bool b;
}
```

# Непреобразуемые типы данных

- Непреобразуемые типы — это типы данных с одинаковым представлением на битовом уровне в управляемом и нитивном коде. Для маршалинга в нативный код и из него эти типы не обязательно преобразовывать в другой формат, что повышает производительность. Поэтому им следует отдавать предпочтение.
  Если непреобразуемые типы передаются по ссылке, маршалер просто закрепляет их вместо копирования в промежуточный буфер (классы изначально передаются по ссылке, структуры передаются по ссылке при использовании атрибутов ref или out).

- Непреобразуемые типы данных:
  - byte, sbyte, short, ushort, int, uint, long, ulong, single, double
  - невложенные одномерные массивы непреобразуемых типов (например, int[]);
  - структуры и классы с фиксированной структурой, содержащие только непреобразуемые значения, например типы полей. Чтобы обеспечить фиксированную структуру, необходимо указать [StructLayout(LayoutKind.Sequential)] или [StructLayout(LayoutKind.Explicit)];

- Преобразуемые типы данных:
  - bool

- Типы данных, преобразуемость которых зависит от места использования
  - char, string
  - char является непреобразуемым в одномерном массиве или, если он является частью другого типа, явным образом помечается [StructLayout] с CharSet = CharSet.Unicode.
  - string является непреобразуемым, если не содержится в другом типе, и передается как аргумент с атрибутом [MarshalAs(UnmanagedType.LPWStr)] или [DllImport] имеет набор CharSet = CharSet.Unicode.

- Рекомендуется по возможности сделать свои структуры данных непреобразуемыми.

# Явный Pin объектов

- GCHandle позволяет закреплять управляемый объект и получать нативный указатель на него.

```
GCHandle handle = GCHandle.Alloc(obj, GCHandleType.Pinned);
IntPtr ptr = handle.AddrOfPinnedObject();
...
handle.Free();
```

- Передача ссылки на управляемый объект через нативный код и обратно на управляемый код, обычно с обратным вызов

```
GCHandle handle = GCHandle.Alloc(obj);
SomeNativeEnumerator(callbackDelegate, GCHandle.ToIntPtr(handle));

// In the callback
GCHandle handle = GCHandle.FromIntPtr(param);
object managedObject = handle.Target;

// After the last callback
handle.Free()
```

- GCHandle необходимо явно освобождать, чтобы избежать утечек памяти

# Задание callback

- Необходимо определить делегат с такой же сигнатурой, которая требуется во внешнем методе

```
private delegate bool EnumWC(IntPtr hwnd, IntPtr lParam);
```

- При задании вешней функции использовать этот делегат

```
[DllImport("user32.dll")]
private static extern int EnumWindows(EnumWC lpEnumFunc, IntPtr lParam);
```

- Вызов нативной функции с передачей делегата

```
EnumWindows(new EnumWC(OutputWindow), IntPtr.Zero);
```

где

```
private static bool OutputWindow(IntPtr hwnd, IntPtr lParam)
{
   Console.WriteLine(hwnd.ToInt64());     return true;
}
```

- Управляемый код в **OutputWindow** будет вызван для каждого окна в системе из нативной функции **EnumWindows**
- Аналогично и для других ОС