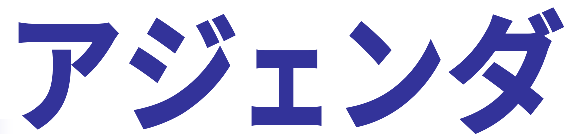
マイクロソフト株式会社
デベロッパー & プラットフォーム統括本部
エバンジェリスト
荒井 省三
mailto:shozoa@microsoft.com

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved



- IronPython 登場の背景
- IronPython のご紹介
- Visual Studio との連携
- アーキテクチャ と内部構造
- まとめ

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

.NET は 動的言語に向かない？

- *"The CLI is, by design, not friendly to dynamic languages. Prototypes were built, but ran way too slowly."* – Jon Udell, InfoWorld, Aug. 2003
  CLI **の設計は動的言語との相性が悪い。プロトタイプを作成してみたが、非常に遅かった。**
- *"The speed of the current system is so low as to render the current implementation useless for anything beyond demonstration purposes."* – ActiveState's report on Python for .NET
  作成したシステムの実行速度は、デモ以外には使えないほどに遅い。

- .NET **は動的言語向けのプラットホームには力不足？**

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

そもそも .NET Framework って何？

- .NET Framework とは Microsoft Windows 向けの新しい マネージ プログラミング モデルであり実行環境
  - マネージ：GC に管理される
  - アンマネージ：従来の Win32 API ベースのため、GC は管理できない
- マイクロソフトは Common Language Infrastructure (CLI) のコア機能を標準化している
  - ECMA と ISO 標準に CLI と C# 言語がある
    - オープン ソース 実装として MONO プロジェクトがある
  - 複数の言語向けの仮想マシン (スタック マシン)
    - Common Intermediate Language (CIL：中間言語)
    - 共通型システム (Common Type System)
    - JIT、GC、リフレクション等の標準機能
  - 8 種類の主要な言語が製品として利用できる
    - C#、VB、マネージ C++、J#、Eiffel、COBOL、RPG、Delphi
  - CLI のマイクロソフト実装が 共通言語ランタイム (Common Language Runtime – CLR -)

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Jim Hugunin 立ち上がる

- **「何故、**.NET **は動的言語として駄目なプラットホームなのか？」という短い論文を書くことを決心した**
- **が、不幸にも** Python **を** .NET **でうまく動かす方法を見つけちゃった**
- IronPython 0.2 **を** 1 **週間で書き上げて、**Python Conference 2004 **で発表するために、**pystones **でベンチマークをやったけど…**

  は、はえー

- **だから、開発の継続を決定**

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

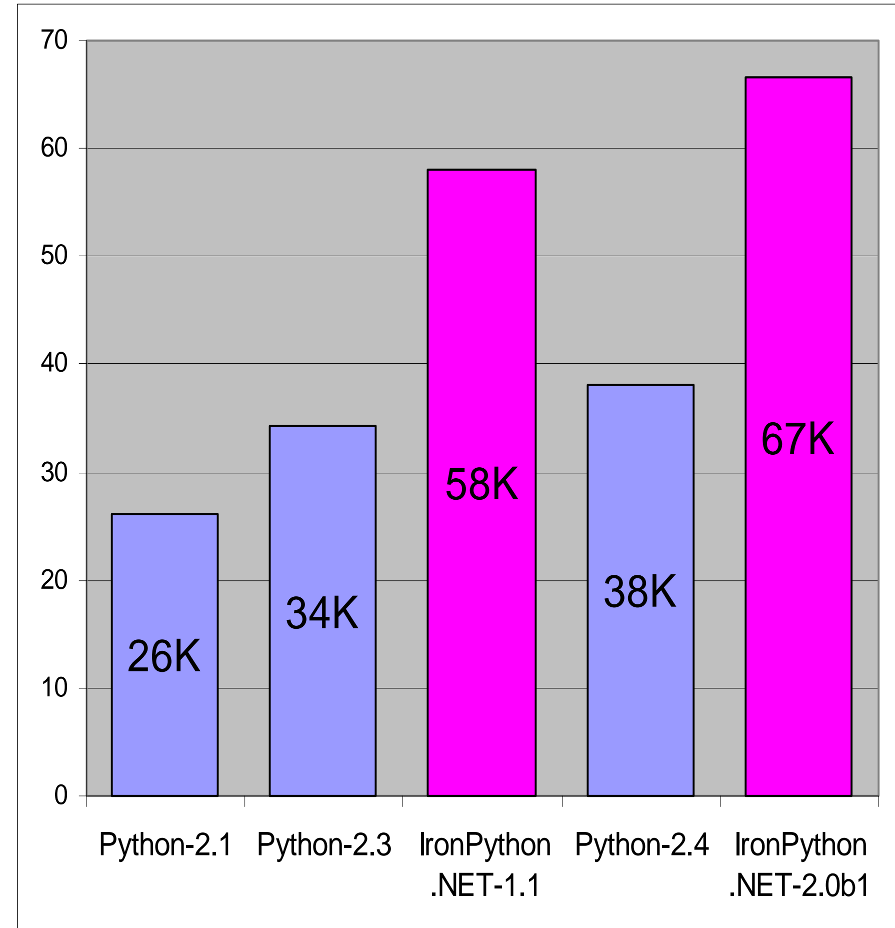
IronPython 登場

- Open Source Conference 2004 で IronPython 0.6 を Common Public License でリリース
- *“There was a meme floating around, a few years ago, that the CLR is inherently unfriendly to dynamic languages. As one of the transmitters of that meme, I’m delighted to be proved wrong.”* – Jon Udell, InfoWorld, July 2004
  「CLR が本質的に動的言語と相性が悪い」と書いた数年前のメモが一人歩きしている。そのメモの発信者の一人として、その事が間違っていると証明されて喜んでいる。
- Jim Hugunin は Redmond へ合流して、IronPython 0.7 をリリース
  - Shared Source Licensing Program としてのライセンス
  - .NET Framework 2.0 ベータ 対応
- 現在の IronPython 1.0 ベータへ至る

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Pystones/second

- 伝統的なベンチマーク
  - Python での dhrystone
  - Lib/test/pystone.py
  - Python で出荷されるベンチマークのみ
- 簡単だけど重要なこと
  - 200行までのコード
  - Python の基本的な Ops
  - OO では無い
- IronPython は早い
  - CPython 2.3 よりも .NET 1.1 で 1.7 倍
  - CPython 2.4 よりも .NET 2.0 β で 1.8倍



Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

IronPython のご紹介

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

.NET のパワーを Python に

- .NET ライブラリへのアクセスをシームレスに
  - マイクロソフト プラットホームの基盤であるライブラリ
  - 既存ライブラリの多くをラップ
  - 新しい OSS ライブラリを同じように書き始めている
  - ライブラリ プールの共有化
- .NET Framework 上で構築
  - CLR が提供する様々なサービス (JIT、GC、 etc)
  - 進行中の実験結果の実現の容易さ
  - どうして、 VM 開発への要望を出さないのか
- 複数の言語との統合
  - デバッガ、スタックトレース、プロファイラ、... との統合
  - 他の言語との相互利用

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

IronPython の基本的な使い方

```
>>> 2 + 2
4
>>> print "Hello World!"
Hello World!
>>> for i in range(3) : print i
0
1
2
>>> x = 10
>>> print x
10
```

```
>>> def add(a, b) :
...     return a + b
...
>>> add(3, 2)
5
>>> add("Iron", "Python")
IronPython
```

```
#組み込み関数
>>> dir()
['__builtins__', '__doc__', '__name__']
```

```
#組み込みモジュール
>>> import sys
>>> sys.version
>>> sys.executable
```

.NET ライブラリの使い方

```
>>> import System                    #標準ライブラリは importで使用できる
>>> dir(System.Environment)
['_CommandLine', ………, 'OSVersion', ……]
>>> System.Environment.OSVersion
Microsoft Windows NT 5.1.2600 Service Pack 2
>>> System.Environment.CommandLine
'"C:¥¥IronPython-1.0-Beta3.0¥¥IronPythonConsole.exe"'
```

```
#グローバル ネームスペースへのインポート
>>> from System.Math import *
>>> dir()
['Abs', 'Acos', 'Asin', 'Atan', 'BigNul', 'Ceiling', 'Cos',
'Cosh', DivRem', 'E', 'Equals', 'Exp', 'Floor',
'GetHashCode', 'GetType', 'IEEERemainder', 'Log', 'Log10',
'Max', 'Min', 'PI', 'Pow', 'Round', 'Sign', 'Sin', 'Sinh',
'Sqrt', 'System', 'Tan', 'Tanh', 'ToString', 'Truncate', '_',
'__builtins__', '__doc__', '__name__']
>>> Sin(PI/2)
1.0
```

.NET ライブラリのロード (1/2)

```
#.NET 標準ライブラリのロード
>>> import clr                                    #初期のベータから変更された
>>> clr.AddReference("System.Xml")
>>> from System.Xml import *
>>> d = XmlDocument()
>>> d.Load("Releases.xml")
>>> n = d.SelectNodes("//@Version")
>>> for e in n: print e.Value
0.7
……
```

```
#.NET で開発されたライブラリのロード
>>> import clr
>>> import nt
>>> clr.Path.append(nt.getcwd())                 #パス設定
>>> clr.AddReferenceToFile("Mapack.dll")     #AddReferenceも可
>>> from Mapack import *
>>> dir()
['CholeslyDecomposition', 'EigenvalueDecomposition',
'LuDecomposition', 'Matrix', 'QrDecomposition',
'SinguarValueDecomposition', '_', '__builtins__', '__doc__',
'__name__', 'sys']
```

.NET ライブラリのロード (2/2)

```
>>> m = Matrix() #Matrix クラスのインスタンス生成
Traceback (most recent call last):
    at <shell>
ValueError: Bad args for the method <constructor# for Mapack.Matrix>
>>> print Matrix.__new__.__doc__
__new__(int, int)
__new__(int, int, float)
__new__(System.Double[][])
>>> m = Matrix(2, 2, 1.2) #.NET クラスではカスタム インデクサをサポート
>>> n = Matrix(2,1)                    #
>>> n[0,0] = 4
>>> dir(m)                 #Matrix は オーバーロード オペレータを持つ
>>> m * n                  #Pythonは __add__,__mul__,…オペレータで表現
4.8
0

>>> n.Transpose() * m
4.8 0

>>> n + -n
0
0
```

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

.NET ライブラリを活用する 1/2

```
#.NET 標準ライブラリを活用する
>>> import System        #ネームスペース System
>>> from System.Collections import *
>>> h = Hashtable()     #Hashtable インスタンスを作成
>>> h["a"] = "IronPython"       #要素を追加
>>> h["b"] = "Tutorial"
>>> h["a"]
IronPython
>>> for e in h: print e.Key, ":", e.Value    #列挙
a : IronPython
b : Tutorial
>>> l = ArrayList([1,2,3])     #ArrayListの初期化(リスト)
>>> for i in l: print i
1
2
3
>>> s = Stack((1,2,3))         #Stackの初期化(タプル)
>>> while s.Count: s.Pop()
3
2
1
```

.NET ライブラリを活用する 2/2

```
#.NET 標準ライブラリを活用する ジェネリック
#.NET 標準ライブラリが提供する型をあたかも Python の型のように扱う
>>> from System.Collections.Generic import *
>>> l = List[str]()                    #文字型のListコレクションの作成
>>> l.Add("Hello")                     #要素を追加
>>> l.Add("Hi")
>>> l.Add(3)
Traceback (most recent call last):
    at <shell>
ValueEror: Bad args for the mothod <mothod# Add on …>
>>> for e in l: print e                #列挙
Hello
Hi
```

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Windows.Forms を活用する

```
#チュートリアルの winforms.py を利用する
>>> import winforms
>>> from System.Windows.Forms import *
>>> from System.Drawing import *
>>> f = Form()                  #Formのインスタンス作成
>>> f.Show()                    #Formの表示
>>> f.Text = "WinForms from IronPython"    #Formのタイトル
>>> def click(*args): print args   #イベントハンドラ定義
>>> f.Click += click                        #イベントハンドラの追加
>>> (System.Windows.Forms.Form, Text: Win Form from IronPython,
<MouseEventArgs object at 0x000000000000002B>)
>>> f.click -= clicl                        #イベントハンドラの削除
>>> dir(MouseEventArgs)
['Button', 'Clicks', 'Delta', 'Empty', 'Equals', 'Finalize',
'GetHashCode', 'GetType', 'Location', 'MemberwiseClone', 'ToString',
'X', 'Y', '__class__', '__init__', '__module__', '__new__']
>>> def click(f, a):        #イベントハンドラ定義 f:フォーム、a:MouseEventArgs
...       l = Label(Text = "Hello") # Text = は プロパティ構文
...       l.Location = a.Location
...       f.Controls.Add(l)
>>> f.Click += click
>>> for i in f.Controls: i.Font = Font("Arial", 18)
>>> for i in f.Controls: i.Text = "Hi"
>>> f.Controls.Clear()
```

イベントハンドラを活用する

```
#.NET の FileSystemWatcher のイベントをPythonで作成する
>>> from System.IO import *
>>> w = FileSystemWatcher()
>>> dir(w)
>>> dir(FileSystemEventArgs)
['ChangeType', 'Equals', 'FullPath', 'GetHashCode', 'GetType', 'Name',
'ToString', '__new__']
>>> w.Path = "." #カレント ディレクトリを監視
>>> def handle(w, a):      #イベントハンドラ定義 w:obj、a:FileSystemEventArgs
...       Print a.ChangeType, a.FullPath
>>> w.Changed += handle    #変更イベントハンドラ
>>> w.Created += handle    #作成イベントハンドラ
>>> w.Deleted += handle    #削除イベントハンドラ
>>> w.EnableRaisingEvents = True   #イベントを発生させる
Created .¥New Text Document.txt
……
>>> w.Changed -= handle    #イベントハンドラの削除
>>> w.Created -= handle
>>> w.Deleted -= handle
```

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

COM 相互運用 1/2

```
#.NET の COM 相互運用アセンブリを作成する（ .NET Framework SDK の世界）
C:¥IronPython¥Tutorial>tlbimp %SystemRoot%¥msagent¥agentsvr.exe
Microsoft (R) .NET Framework Type Library to Assembly Converter
2.0.50727.42
Copyright (C) Microsoft Corporation.  All rights reserved.
Type library imported to AgentServerObjects.dll
```

```
#.NET の COM 相互運用アセンブリを活用する
>>> import nt
>>> import clr
>>> clr.Path.append(nt.getcwd())
>>> clr.AddReferenceToFile("AgentServerObjects.dll")
>>> from AgentServerObjects import *
>>> dir()
['AgentServer', 'AgentServerClass', 'IAgent',
'IAgentAudioOutputProperties', 'IAgentAudioOutputPropertiesEx',
'IAgentBalloon', 'IAgentBalloonEx', 'IAgentCharacter',
'IAgentCharacterEx', 'IAgentCommand', 'IAgentCommandEx',
'IAgentCommandWindow', 'IAgentCommands', 'IAgentCommandsEx',
'IAgentEx', 'IAgentNotifySink', 'IAgentNotifySinkEx',
'IAgentPropertySheet', 'IAgentSpeechInputProperties',
'IAgentUserInput', '_', '__builtins__', '__dict__', '__doc__',
'__name__', 'clr', 'site']
```

COM 相互運用 2/2

```
>>> a = AgentServerClass()    #AgentServer のインスタンス作成
>>> dir(a)
['CreateObjRef', 'Equals', 'Finalize', 'GetCharacter', 'GetHashCode',
'GetLifetimeService', 'GetSuspended', 'GetType', 'InitializeLifetimeService',
'Load', 'MemberwiseClone', 'Register', 'ToString', 'Unload', 'Unregister',
'__class__', '__init__', '__module__', '__new__']
>>> a.Load.__doc__
'(void, int, int) Load(object vLoadKey)¥r¥n'
>>> a.Load("Merlin.acs")      # Merlin ウイザード キャラクタの読み込み
>>> cid = _[0]                          # キャラクタ ID
>>> c = a.GetCharacter(cid)   # キャラクタの取得
>>> c.Show(0)                           # キャラクタの表示
>>> c.Think("IronPython Tutorial")      # キャラクタの考え中。。。
>>> for n in c.GetAnimationNames(): print n
...
RestPose
Blink
...
>>> for n in c.GetAnimationNames():globals()[n]=lambda name=n:c.Play(name)
          #グローバル ネームスペースに アニメーションを定義
>>> Congratulate()            #アニメーション の組み込みメソッド
>>> c.StopAll(0)              #アニメーションの停止
>>> c.Hide(0)                           #キャラクタの非表示
```

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Visual Studio との連携

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

.NET アプリケーションから使う

```
#.NET の C# 言語から使う
using System;
using IronPython.Hosting;

public class Eval {
   public static void Main(string[] args)
   {
        string input = "";
        while(true)
        {
           Console.WriteLine("何か入力して下さい");
           input = Console.ReadLine();
           if (input == "")  break;
           PythonEngine pe = new PythonEngine();
           Console.WriteLine(pe.Evalute(input));
        }
    }
```

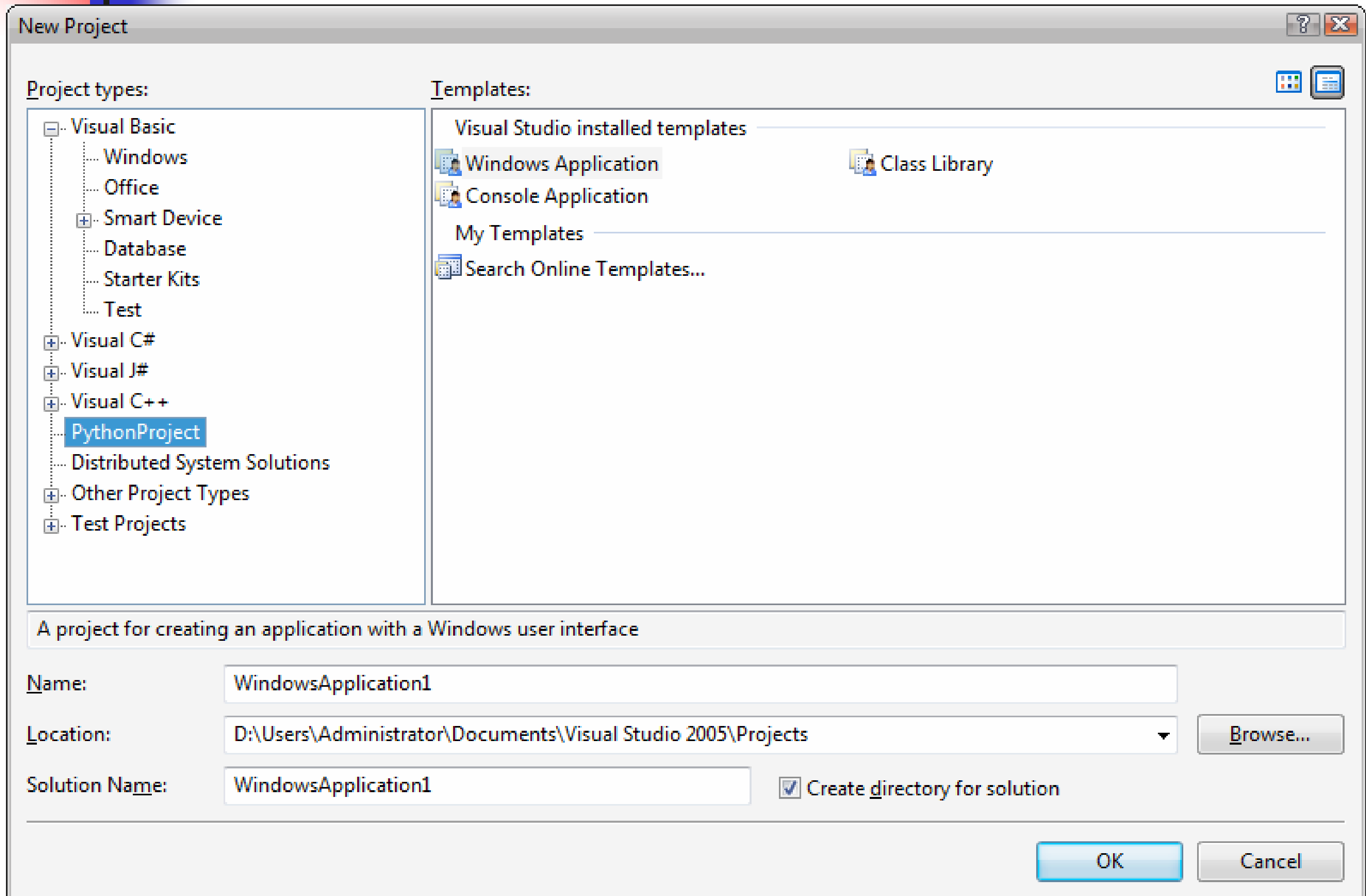
Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Visual Studio を Python で拡張 1/3

- Visual Stduio 2005 SDK の統合サンプル
  - 言語テンプレートを提供
    - Python コンソール アプリケーション
    - Python Windows アプリケーション
    - Python クラス ライブラリ
- まだ、完成度は高くない
- Python スクリプトのデバッグが可能
- IronPython コンパイラー サービスを使うことで .NET アセンブリにビルドする
  - 実行時に スクリプト ファイルが不要
  - スクリプトの実行よりも高速に動作

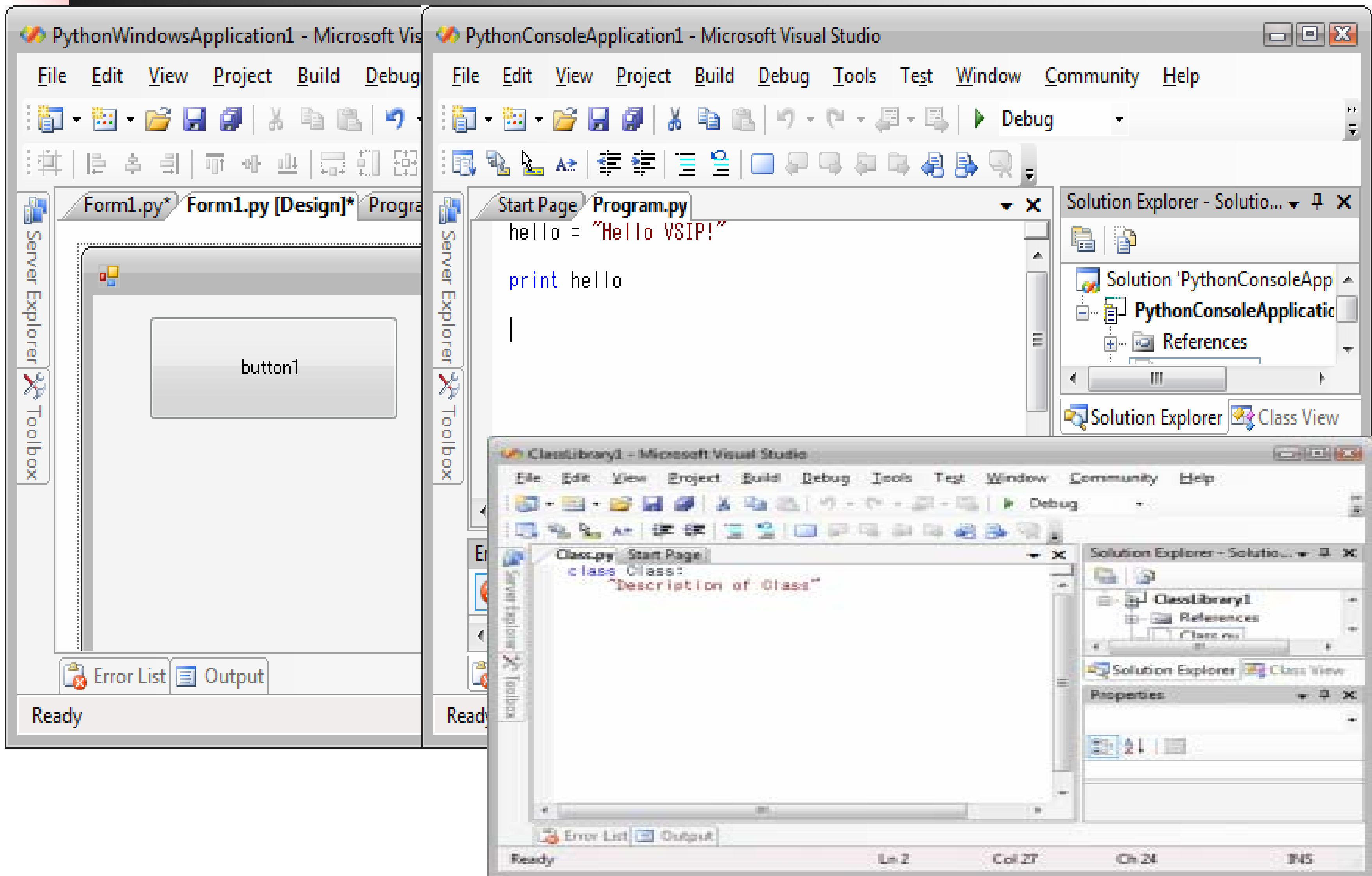
Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Python プロジェクト 2/3



Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Python プロジェクト 3/3

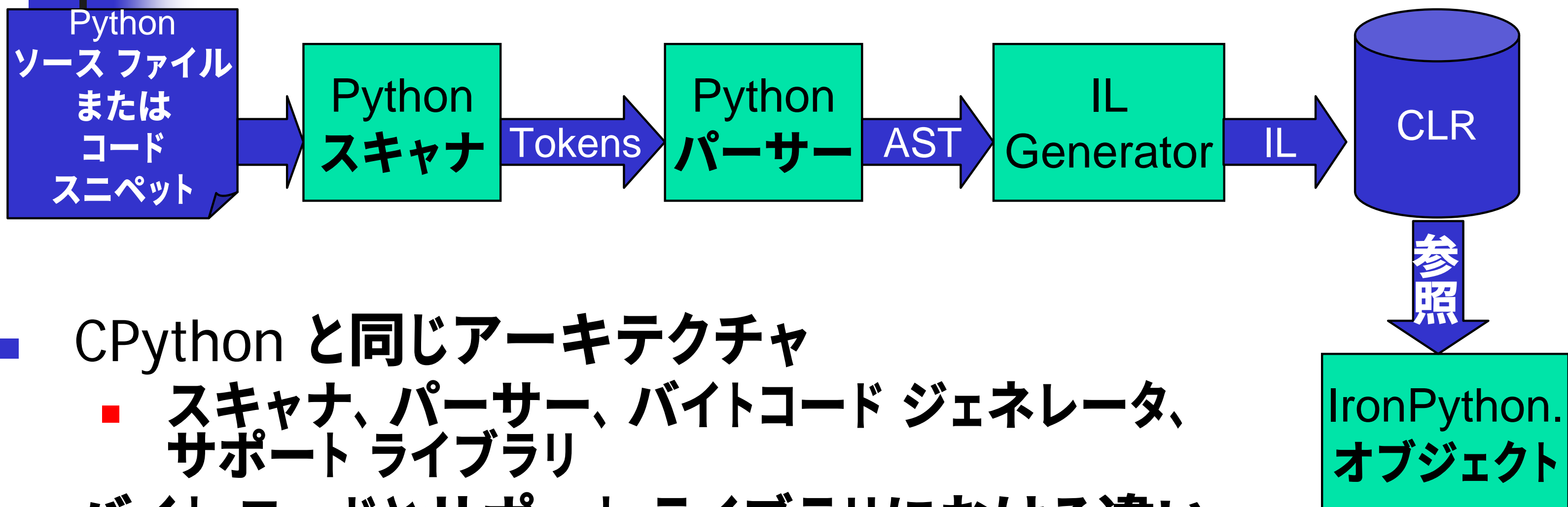


Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

アーキテクチャ と 内部構造

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

IronPython アーキテクチャ



- CPython と同じアーキテクチャ
  - スキャナ、パーサー、バイトコード ジェネレータ、サポート ライブラリ
- バイト コードとサポート ライブラリにおける違い
  - バイト コードは CLR 向けの IL – Native コードを生成するようなもの
  - ライブラリは、 C の代わりに C# で書かれている
- コンパイルは 事前か動的が可能
  - Exe や dll を生成するか、実行時に動的にロード
- IronPython は C# で書かれている
  - Jim は 3 種類のプロトタイプを作成してみた
  - 設計パターンを理解してから、 C# へと移植した

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

IronPython コード 構造

Src		Iron Python ソース コード
IronMath		数学ライブラリ
IronPython		IronPython のメイン モジュール
	Compiler	AST(Tokenizer、Parser、CodeGen)
	Hosting	ホスティング インターフェース、PythonEngine, Console
	Modules	ビルトイン モジュール (sys, nt, __builtins__, time, …)
	Runtime	実行エンジン
	CodeDom	CodeDom：オブジェクトグラフをコンパイルしたり、ソースコードをコンパイル
IronPythonConsole		コンソール (Hosting API を利用)
IronPythonTest		サポート ライブラリのテスト用
Scripts		スクリプト
	Tests	IronPython テスト用

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Factorial 関数をコンパイル

```
def factorial(n):
    if n == 1: return 1
    return n * factorial(n-1)
```

0 LOAD_FAST 0(n)	IL_0003: ldarg.0
3 LOAD_CONST 1(1)	IL_0004: ldsfld object __main__::c0PST0400005
6 COMPARE_OP 2(==)	IL_0009: call object [IronPython]IronPython.Runtime.Ops::Equal(object,object)
9 JUMP_IF_FALSE 8(to 20)	IL_000e: call bool [IronPython]IronPython.Runtime.Ops::IsTrue(object) IL_0013: brfalse IL_002b
12 POP_TOP	**結果をスタックへ入れ、抜ける**
13 LOAD_CONST 1(1)	IL_001b: ldsfld object __main__::c0PST04000005
16 RETURN_VALUE	IL_007d: ret

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Factorial 関数を C# 表現へ

```
public static object factorial$f0(object n)
 {     int num1;
       object obj1;
       try
      {     num1 = 2;
            if (Ops.IsTrue(Ops.Equal(n, __main__.c$0)))
            {     num1 = 2;
                   return __main__.c$0;   }
            num1 = 3;
            Ops.CheckInitialized(__main__.factorial);
            return Ops.Multiply(n, Ops.CallWithContext(
                       __main__.myModule__py, __main__factorial,
                       Ops.Subtract(n, __main__.c$0)));
             obj1 = null;
       }
       catch
       {     Ops.UpdateTraceBack(__main__.myModule__py,
                                          "factorial", "factorial.py", num1); }
       return obj1;   }
//コンストラクタで「c$0」を「1」に初期化
```

キー クラス – Ops

- Runtime¥Ops.cs の Ops クラスで Python オペレータを実装している
  - Modules¥operetor.cs から Ops クラスを呼び出す
- オペレータ
  - a + b … Ops.Add(a, b)
  - a += b … Ops.InPlaceAdd(a, b)
  - a < b … Ops.LessThan(a, b)
- アトリビュート アクセス
  - a(b) … Ops.Call(a, b)
  - a.b() … Ops.Call(Ops.GetAttr(a, “b”))
- 言語構造のサポート
  - print a … Ops.Print(a)
  - raise e … Ops.Raise(e, null, null)

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

タイプ システム

- DynamicType – Python 型の抽象基本クラス
    - Python の “Ops” 命令セットをサポートするためのメソッド定義
  - OldClass – 本来の Python クラス
  - PythonType ー .NET のクラスを継承した Python クラス
    - RefrectedType ー .NET クラスをインポート
    - UserType ー .NET クラスを継承した Python クラス
      - >>> s = “ py”
      - >>> s.strip() #Python スタイル
      - >>> s = “ py”
      - >>> s.Trim() #何が起きているのか？
- .NET クラスの継承をサポート
  - ISuperDynamicType
    - .NET クラスを継承した Python クラスのインスタンスは、このインターフェースを実装
    - .NET クラスを継承したクラスにおいても、動的なセマンティックスをサポート

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

インターフェース

- 関数呼び出し
  - ICallable ... 配列パラメータを持つ呼び出し
  - IFastCallable ... 4個以内のパラメータ向けの高速呼び出し
  - IFancyCallable ... 名前つき引数をサポート
- アトリビュート アクセス
  - ICustomAttributes
- クラスは ICallable インターフェースを実装している

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

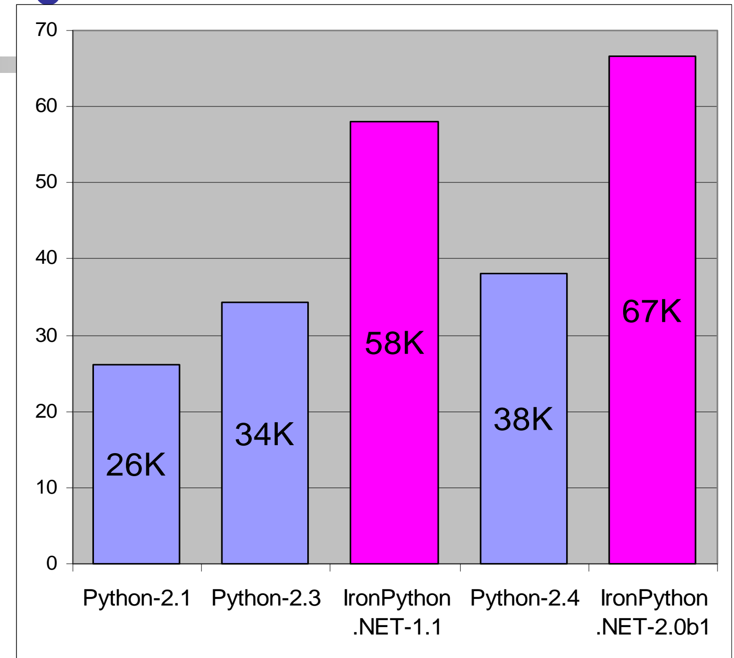
何故、パフォーマンスが良いのか

- 可能な限り CLR ネイティブな構造を活用
  - 面白いほど マシン コードが最適化されてる
  - が、これを Python セマンティックスと一致させるために使わなければならない
- 多くのケース向けの 最適なパスを作成する
  - 開発時は C# コードを生成するように Python スクリプトを使う
  - Reflection.Emit を実行時の IL 生成に使用する
  - 幾つかのパスは 手書きの価値を持つかもしれない
  - 何が早いパスに値するかを理解することが挑戦だ！
- 完全な動的な実行のために以下を考慮する
  - 一般的でないものに対する多目的のサポート
  - Python の完全な動的セマンティックスを扱う

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

CLR 技術は ？

- DynamicMethod
  - 軽量なコード生成 (LCG)
  - 言語実装向けに便利な API
- System.Reflection.Emit ネームスペース
- デリゲート パフォーマンス
  - デリゲートは CLR の大きなメリット
  - パフォーマンスが大幅に向上
- 実行環境に埋め込まれた ジェネリック
  - 完全に動的な リフレクションをサポート
  - 複数言語との親和性において追加された新しい機能
- リフレクション パフォーマンス
  - .NET 1.1 から 2.0 で劇的に向上
    - 実装が COM から マネージに変更になった



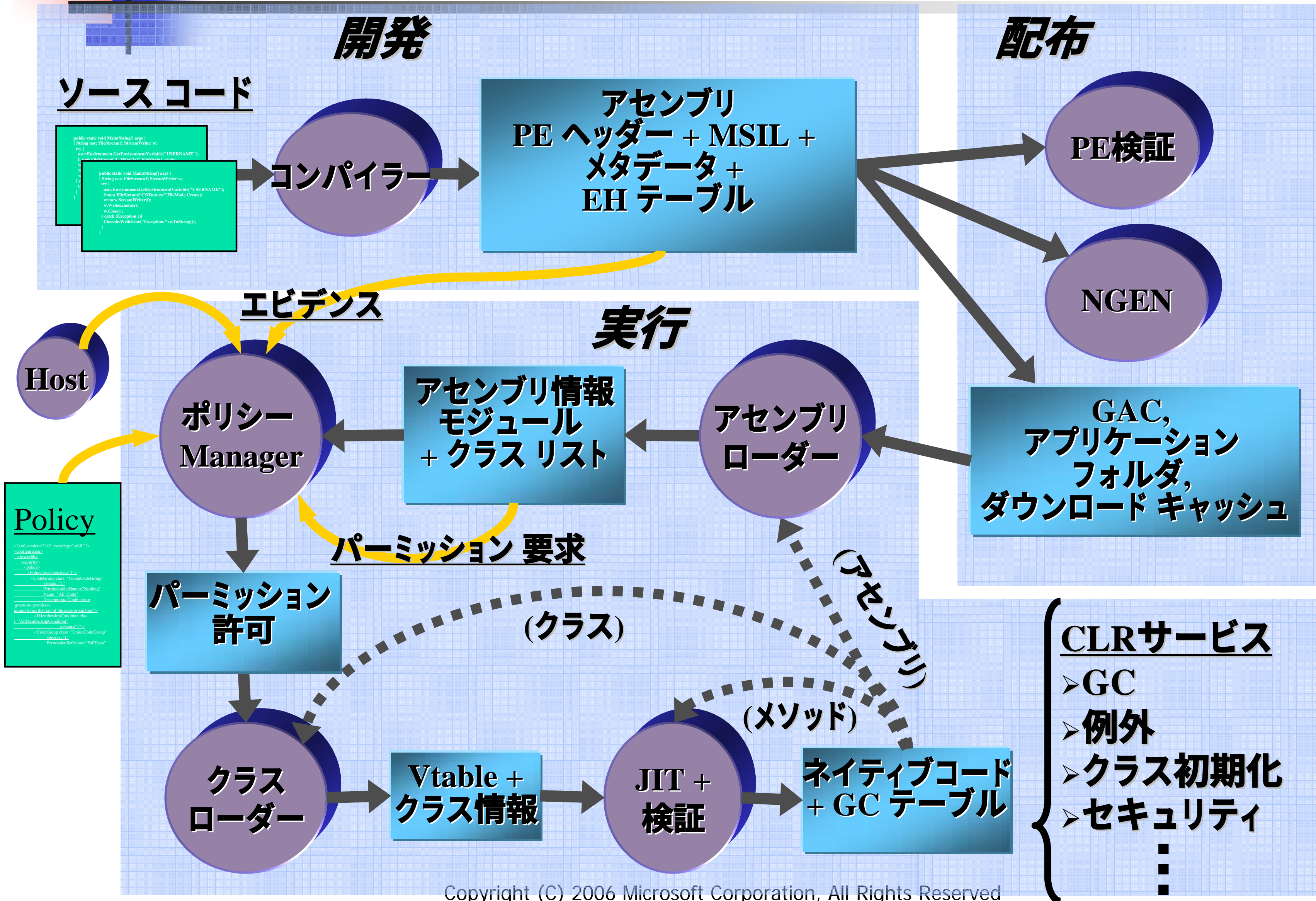
Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

LightWeight Code Generator

- System.Reflection.Emit とは
  - IL を生成するための API を提供する
  - アセンブリ を生成する手順
    - アセンブリを定義 (AssemblyBuilder)
    - モジュールの定義 (ModuleBuilder)
    - クラスの型を定義 (TypeBuilder、 TypeAttributes)
    - フィールド定義 (FieldBuilder)
    - コンストラクタの定義 (ConstructorBuilder、CunstructorInfo)
    - コンストラクタ コードの定義 (ILGenerator)
    - メソッドの定義 (MethodBuilder、ParameterInfo)
    - メソッド コードの定義 (ILGenerator)
- DynamicMethod とは
  - .NET 2.0 で提供された軽量な IL 生成 API
  - IL の生成手順
    - 動的メソッドの定義 (DynamicMethod、MethodInfo)
    - メソッド コードの定義 (ILGenerator)
  - AssemblyBuilder との違いは
    - 実行中のアセンブリの中に スタティックなメソッドを作成する
    - このため、アセンブリやモジュール、クラス定義などが不要

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

CLR の実行イメージ －参考－



Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

まとめ

- IronPython は Python 2.4 と互換が目標
- CLS 機能の全てをサポート予定
  - .NET クラスライブラリとのシームレスな連携
- IronPython のスレッドは シングル スレッド
  - Windows アプリケーションは メッセージ ポンプが必要だから
  - -X:MTA オプションでマルチスレッド
    - でも、PythonEngine がマルチスレッドで生成されるだけ
- COM 相互運用もできる
  - が、.NET の COM 相互運用を利用
  - つまり、CLR の提供する機能をシームレスに使えることの応用でしかない
- IronPython は .NET Framewrok を有効活用している
  - .NET Framework の進化は IronPython にもメリットをもたらす
- IronPython は プラットホームの進化を忘れない

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

情報入手先

- http://workspace.gotdontet.com/ironpython/
- メーリングリスト
  - http://lists.ironpython.com/listinfo.cgi/users-ironpython.com
- http://ironpython.com/
- IronPython Team Wiki
  - http://channel9.msdn.com/wiki/default.aspx/IronPython.HomePage
- Visual Studio 2005 SDK
  - http://affiliate.vsipmembers.com/
  - Visual Studio Industry Partners Program への登録が必要 (無料)
- IronPython: A fast Python Implementation for .NET and Mono
  - http://www.python.org/pycon/dc2004/papers/9/

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

日本語環境への対応状況

- PythonEngine は C# で開発されているため日本語を扱うことは問題なし
- IronPythonConsole では IME 切り替えのキーを受け付けない
- スクリプト ファイル で日本語を扱うと化ける
  - スクリプト ファイルの読み込みのエンコーディングが ASCII 固定となっており、SJIS には対応していないため
  - UNICODE には対応済み (Beta4から対応。BOM が基本 ー UTF-8/16)
  - Compiler¥Parser.cs の FromFile メソッド のエンコーディングを修正すれば SJIS を扱うことは可能
    - 但し、広範なテストを受けて実証されていない
    - 本質的には、ファイルのエンコードを自動識別してエンコーディングを決定するのがベストと考えられる

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

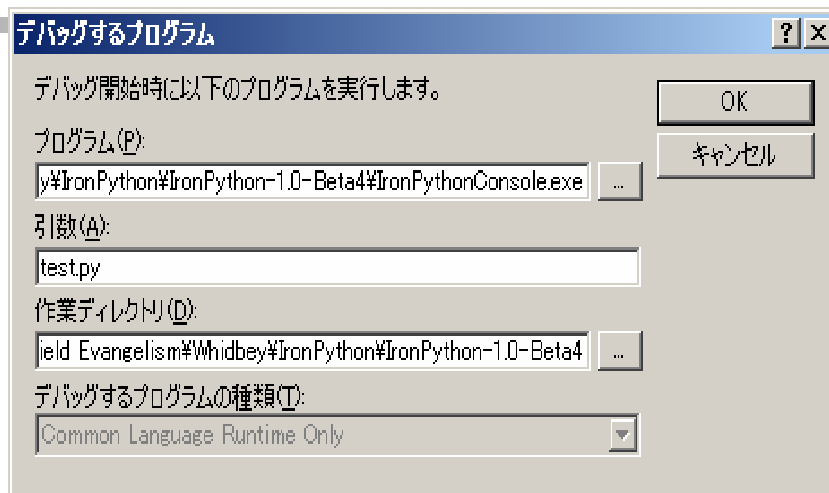
Python スクリプトの 事前コンパイル

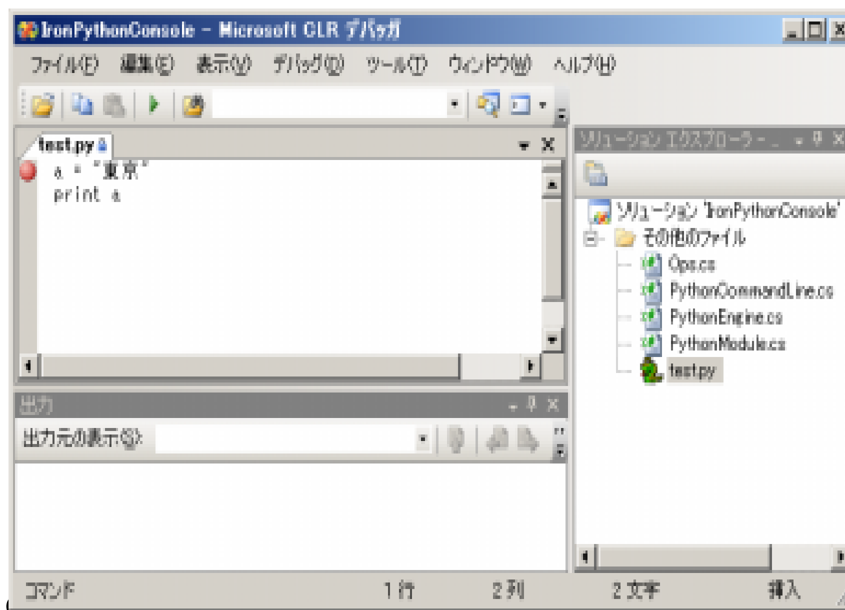
- -X:SaveAssemblies オプションを使用する
- Factorial.py の例では
  - 「IronPythonConsole.exe –X:SaveAssemblies factorial.py」でアセンブリを出力
- 配布対象が、IronPython ランタイムと出力したアセンブリだけになる
  - スクリプト ファイルを実行時に解析するオーバヘッドが無くなる
  - 実行速度の改善に役立つ

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved

Python スクリプトのデバック

- .NET Framework SDK の CLR デバッガを使用する
- デバックするプログラムを指定する
- ファイルメニューから、スクリプトファイルを開く
- ブレークポイントを作成してデバック





Copyright (C) 2006 Microsoft

Microsoft®

Your potential. Our passion.™

Copyright (C) 2006 Microsoft Corporation, All Rights Reserved