わんくま同盟 大阪勉強会 #13

「ダイナミックな世界」

19:00〜20:30

Lv1くまー by 中博俊

みなさんは
ダイナミック言語(DL)や、
軽量言語(LL)を
知っていますか？

動的言語とは

- ダイナミック言語(動的言語)
  - ⇔スタティック言語(静的言語)
- 特徴
  - コンパイルを実行時に行ったり、インタプリタのようにふるまったり、がちがちに固めない。
  - 実行時にあればいいじゃないか
- 具体的には
  - Visual Basic, Java Script, Iron Python, Iron Ruby, Power Shell 等

動的言語とは

- Silverlightを知っていますか？
  - Windows, Mac OSで動く
  - IE, Mozilla, Firefox, Safariで動く
  - XAMLが動く
  - そんなアプリケーション環境

動的言語とは

- Silverlight1.1にはDLRが搭載されるんです。
  ↓
- DLRとは
  - Dynamic Language Runtimeの略で、動的言語環境
  - 動的言語の積極的サポート

- 1番の特徴は
  - 複数言語が、1つの動的言語環境で動くこと
  - Iron Pythonから、Java Scriptのコードを簡単に呼べます。

デモしましょ

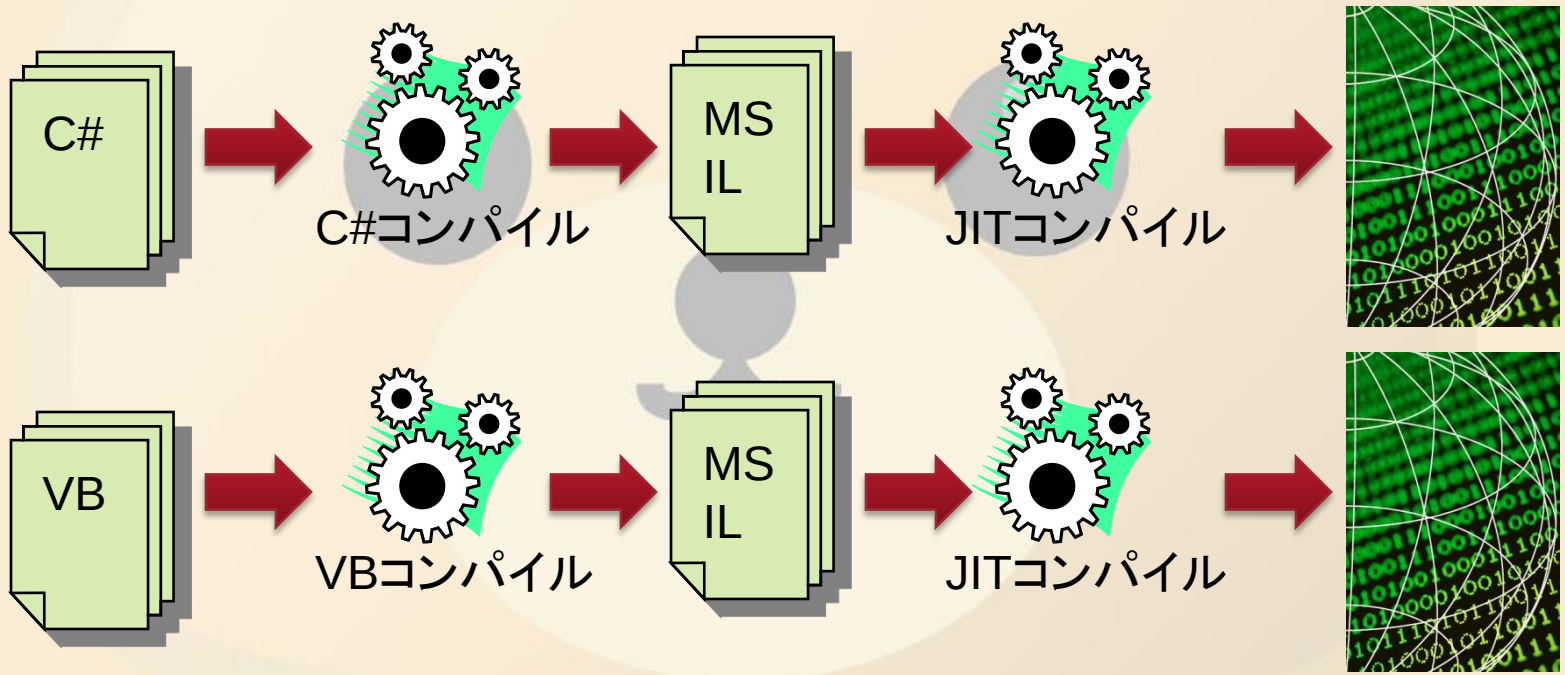
- PowerShellのデモ
- IronPythonのデモ
- 電卓のデモ

同居

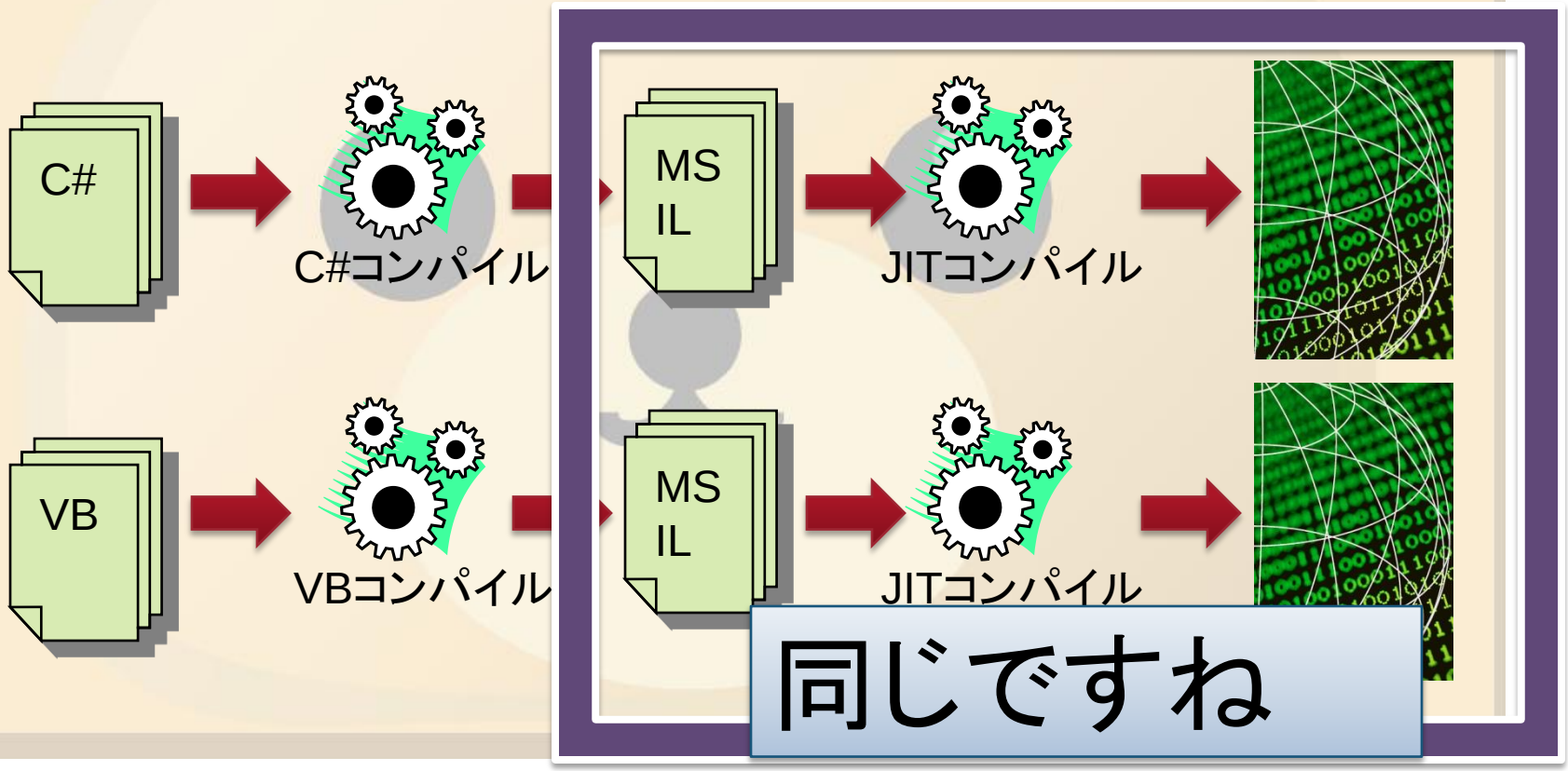
- IronPythonと、JavaScriptが同居しているということは？
- C#とVBが同居しているということとおなじ？

No

- C#ソース → C#コンパイル → MSIL → JITコンパイル → アセンブリ →実行



- C#ソース → C#コンパイル → MSIL → JITコンパイル → アセンブリ →実行



DLRとは

- スクリプトな言語であるJavascriptや、IronPythonの実行環境であるDLRとは

- 結局AST + MiniCLRである。

AST

- Abstract Syntax Tree
- 日本語では抽象構文木

- プログラムのソースを抽象化する
- ソースコードを論理的なものに変える

- 次の資料を一緒に読みましょう。

http://www.iunknown.com/2007/05/compiler_dev_la.html

John Lam on Software

Ruby, Dynamic Language Runtime, Silverlight and Microsoft

« DLR Running on Mono | Main | Wrapping up the Compiler Dev Lab »

MAY 22, 2007

Compiler Dev Lab

The DLR team is hosting this year's compiler dev lab on campus this week. One of the really cool things about the lab was meeting folks who were really interested porting

* Facebook

facebook

Name:
John Lam

Networks:
Microsoft
Seattle, WA

Email Me

* About

突然ですが

- Haskellという言語を知っていますか?

- 純粋関数型言語
- 手続きは書けない

- それよりそもそも関数型言語ってなに？

関数型言語とは

- 状態をもたないこと(再入可能)
  - Function add(x,y) { return x+y;}
  - このメソッドは状態をもたないために、同じ値1,3を入れれば必ず4が出てくる
- 関数も型

- 単純に言うとこういうこと

カリー化

- Haskellでは関数は1つの引数しかとれません。(戻り値も1つ)
- 先ほどのaddを考えてみましょう。

- Add(1,2)は以下のように分解できます。
- Number1(){return 1;}
  Add2(Number1){return Number1() + 2;}

C#でのデモ

カリー化のメリット

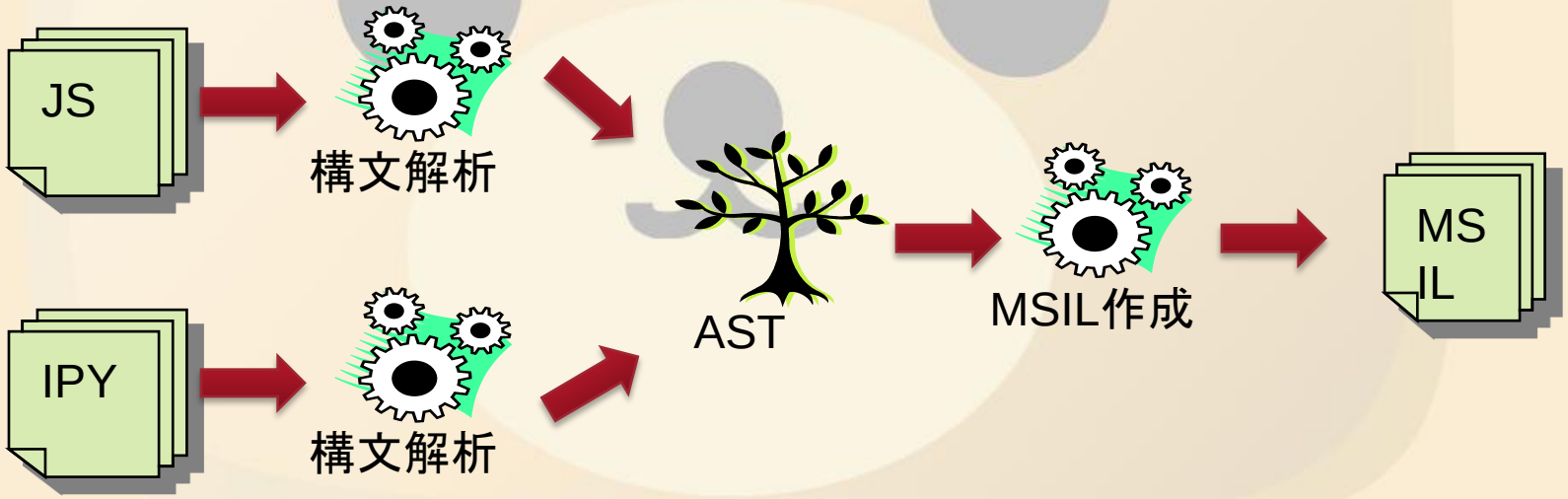
- 物事を単純化できる
- 戻り値は関数なので、使う瞬間に評価(実行)することができる。(遅延評価)

- 閑話休題

- 前の資料に戻る

- C#のコンパイルも実際には

- DLRの構文解析は



LinqのExpression Treeって

- Linqのクエリ式はExpression Treeに展開される。(場合が多い)

- http://blogs.wankuma.com/shuujin/archive/2007/09/25/97854.aspx

- var query =
  from address in adventureWorks.Address
  where address.City == "Bothell“
  select address.AddressID;
- IQueryable<int> query = adventureWorks.Address.

  Where<Address>(Expression.Lambda<Func<Address, bool>>(
  Expression.Equal(Expression.Property(CS$0$0000 =
  Expression.Parameter(typeof(Address), "address"), (MethodInfo)
  methodof(Address.get_City)), Expression.Constant("Bothell",
  typeof(string)), false, (MethodInfo) methodof(string.op_Equality)), new
  ParameterExpression[] { CS$0$0000 })).

  Select<Address, int>(Expression.Lambda<Func<Address, int>>(
  Expression.Property(CS$0$0000 = Expression.Parameter(typeof(Address),
  "address"), (MethodInfo) methodof(Address.get_AddressID)), new
  ParameterExpression[] { CS$0$0000 }));

- なぜこんなに属性などを付加しているのか？
- データソースに応じて
- **動的に**
- 処理を振り分けられるように
- 論理構造を表現するのにTreeが一番ふさわしいから。でもやっぱり違う技術です

- DLRと言いながら、C#とか、Haskellとかいろんなところのいろんなエッセンスを紹介しました。
- 今後もまだまだプログラミング言語は進化しそうです、実行環境は進展しそうです。
- こんなダイナミックな世界に没頭できて幸せです。(^^

- ありがとうございました。