.NET Framework 4.0 世代の Expression Trees

September 26th, 2009

渋木宏明(ひどり)
Microsoft MVP for C#

自己紹介

プロフィール

- 名前
  - 渋木宏明(ひどり)
- 出身地
  - 東京都
- 職業
  - フリーランスの開発者
- 技術分野
  - Visual C#, Windows.Forms

コミュニティ活動

- ホームページ
  - http://hidori.jp/
- ブログ
  - http://hidori.jp/blog/
- Twitter
  - http://twitter.com/hidori
- その他
  - Microsoft MVP for Visual C#
  - VSUG ボードリーダー「C++/CLI その他掲示板」

アジェンダ

- はじめに
- .NET Framework 3.5 SP1 世代の Expression Trees
- .NET Framework 4.0 Beta1 世代の Expression Trees
- まとめ

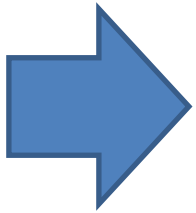
はじめに

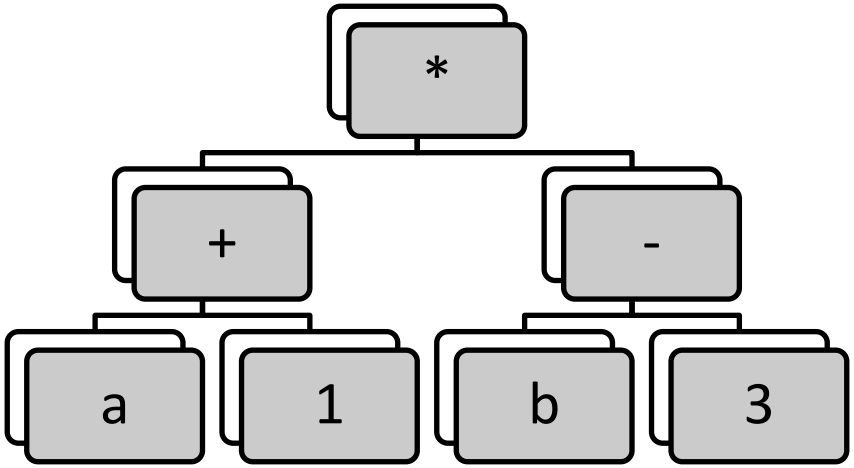
Expression Trees とは？

- **式 を木構造で表わしたモノ**
- 一般的には**「式木」**と訳されているハズ
- MSDN ライブラリでは、Expression Trees の訳語として**「式ツリー」**が採用されている

（例）式を木構造で表わす

(a + 1) * (b - 3)





※ 木構造は、演算子の優先度などを反映している。

.NET Framework 3.5SP1 世代の Expression Trees

.NET Framework 3.5SP1 における Expression Trees

- LINQ の**超重要**な基盤技術の1つ
- System.Linq.Expressions 名前空間が新設され、「式ツリー」を扱うためのクラス・列挙子などが追加された

System.Linq.Expressions 名前空間のメンバによる「式ツリー」の特徴

- 「式ツリー」の各ノードは、Epxression クラスの派生型で表わす
- 「式ツリー」は、Epxression クラスの静的メソッドを利用して構築する
- 「式ツリー」は、実行時に匿名デリゲートに変換することができる

  → **いわゆる「数学的な関数」を動的に作成することが可能**

LINQ?

- 統合言語クエリ(Language Integrated Query)
- VB, C# などのソースコード中に、SQL 似の構文でデータ操作を記述するための仕組み
- 「LINQ プロバイダ」が提供されている様々なデータソースを、ほぼ同じような記述で操作することができる
- 標準で以下のデータソースが利用可能（ .NET Framework 3.5SP1 リリース当初）
  - LINQ to Object（一般的なデータクラスを操作）
  - LINQ to SQL（SQL Server 上のデータを操作）
  - LINQ to XML（XML 文書のデータを操作）

こんな風にデータ操作

```
// LINQ によるクエリ
var query = from x in table
            where x.Age > 30
            select x;

// クエリ結果を表示
foreach (var person in query)
{
   Console.Out.WriteLine(person.Name);
}
```

こんな風に検索することが出来る

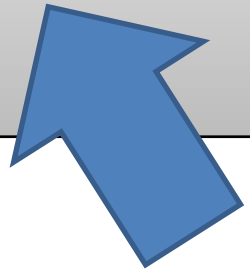
```
var query =
```

LINQ の基盤技術

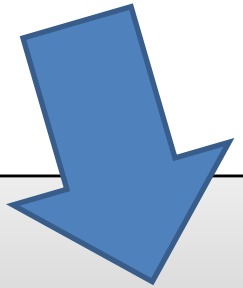
クエリ構文	• SQL 似の構文でデータ操作を記述
拡張メソッド	• 既存の型に対してメソッドを追加（見掛け上）
匿名型	• 型宣言なしでデータクラスを使用
暗黙的に型指定される ローカル変数	• var キーワードによる、型名の記述を省いた変数宣言
ラムダ式	• 名前の無い計算式を記述 • ラムダ式を書いただけでは、計算は実行されない
式ツリー	• **条件式などの内部表現**

LINQ のココが式ツリー

```
var query = from x in table
        where 30 > x.Age
        select x;
```



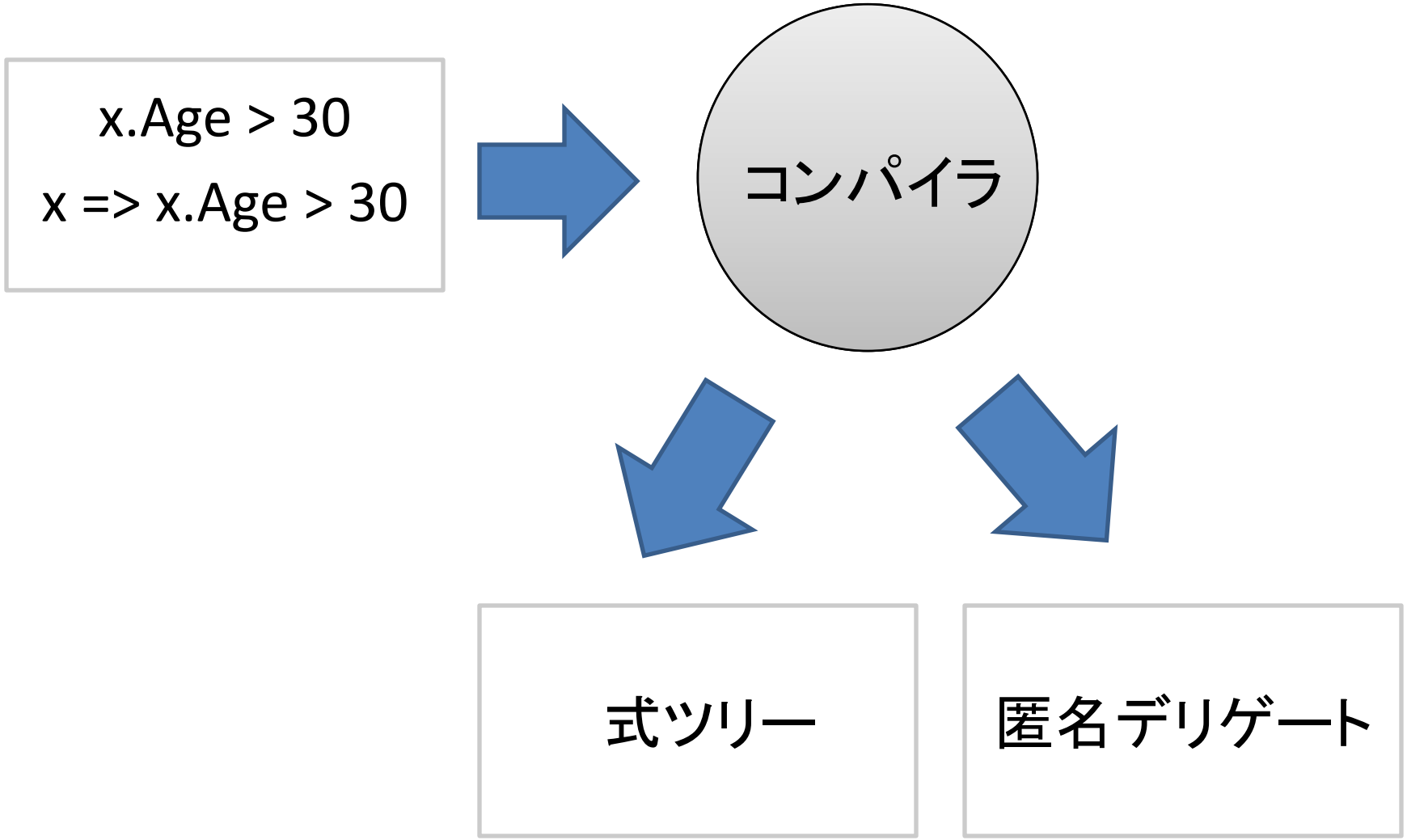
ココ



ココとか

```
var query = table
        .Where(x => 30 > x.Age);
```

条件式・選択式、ラムダ式
→ 式ツリー or 匿名デリゲート



式ツリーの応用

- 条件式を動的に作成、クエリを実行
  - チェックボックス、コンボボックスなどで与えられる複合条件から式木を作成して、クエリを実行
- 計算式を動的に作成、実行
  - 業務系ではあまりそういうニーズは無い？
- 式木そのものを処理対象とする
  - 岩永（ufcpp）さんのホームページで紹介されている「式木を微分」のサンプル http://ufcpp.net/study/csharp/sp3_expressionsample.html

こんなデータがあるとして…

```
/// <summary>
///「名前」と「年齢」を格納するデータクラス
/// </summary>
class Person
{
        public string Name { get; set; }
        public int Age { get; set; }
}

// テストデータ
var table = new[]
{
    new Person { Name = "ailight", Age = 30},
    new Person { Name = "hidori", Age = 29},
    new Person { Name = "kazuk", Age = 35},
};
```

条件式を動的に作成

```
// ラムダ式 ‘_ => _.Age > 30’ と等価な式ツリーを作成
static Expression<Func<Person, bool>> BuildExpression()
{
   // ラムダ式のパラメータ '_‘
   var param = Expression.Parameter(typeof(Person), "_");

   // 比較式 '_.Age > 30‘
   var left = Expression.Property(param, "Age");
   var right = Expression.Constant(30);

   var body = Expression.GreaterThan(left, right);

   // ラムダ式を返す
   return (Expression<Func<Person, bool>>)Expression.Lambda(body, new[] { param });
}
```

動的に作成した条件式でクエリを実行

```
// ラムダ式を動的に生成
var lambda = BuildExpression();

// ラムダ式から匿名デリゲートを作成
var del = lambda.Compile();

// クエリに、動的に作成された条件式を与える
// LINQ to SQL の場合は直接 lambda を与える
var query = table.Where(del);

// クエリ結果を表示
foreach (var person in query)
{
    Console.Out.WriteLine(person.Name);
}
```

計算式を動的に作成

```
// ラムダ式 ‘x => x * x’ と等価な式ツリーを作成
static Expression<Func<int,int>> BuildExpression()
{
    // ラムダ式のパラメータ 'x‘
    var param = Expression.Parameter(typeof(int), "x");

    // 乗算式 'x * x‘
    var body = Expression.Multiply(param, param);

    // ラムダ式を返す
    return (Expression<Func<int, int>>)Expression.Lambda(body, new[] { param });
}
```

動的に作成した計算式を実行

```
// ラムダ式を動的に生成
var lambda = BuildExpression();

// ラムダ式から匿名デリゲートを作成
var del = lambda.Compile();

// 計算式を実行
var result = del(7);

// 計算結果を表示
Console.Out.WriteLine(result);
```

.NET Framework 4.0 世代の Expression Trees

.NET Framework 4.0 における Expression Trees

- **DLR** の**超重要**な基盤技術の1つ
- System.Linq.Expressions 名前空間のメンバが追加・拡張され、**「構文木（Abstraction Syntax Trees）」を扱うことができるようになった**
  （後方互換性は保たれている）

DLR?

- 動的言語ランタイム（Dynamic Language Runtime）
- **各言語でバラバラに実装されている構文木やデータ型を統合、.NET での動的言語実装を強力に支援**
- 現在、DLR を利用して実装された、以下の動的言語が CodePlex で公開されている
  - IronPython
  - IronRuby

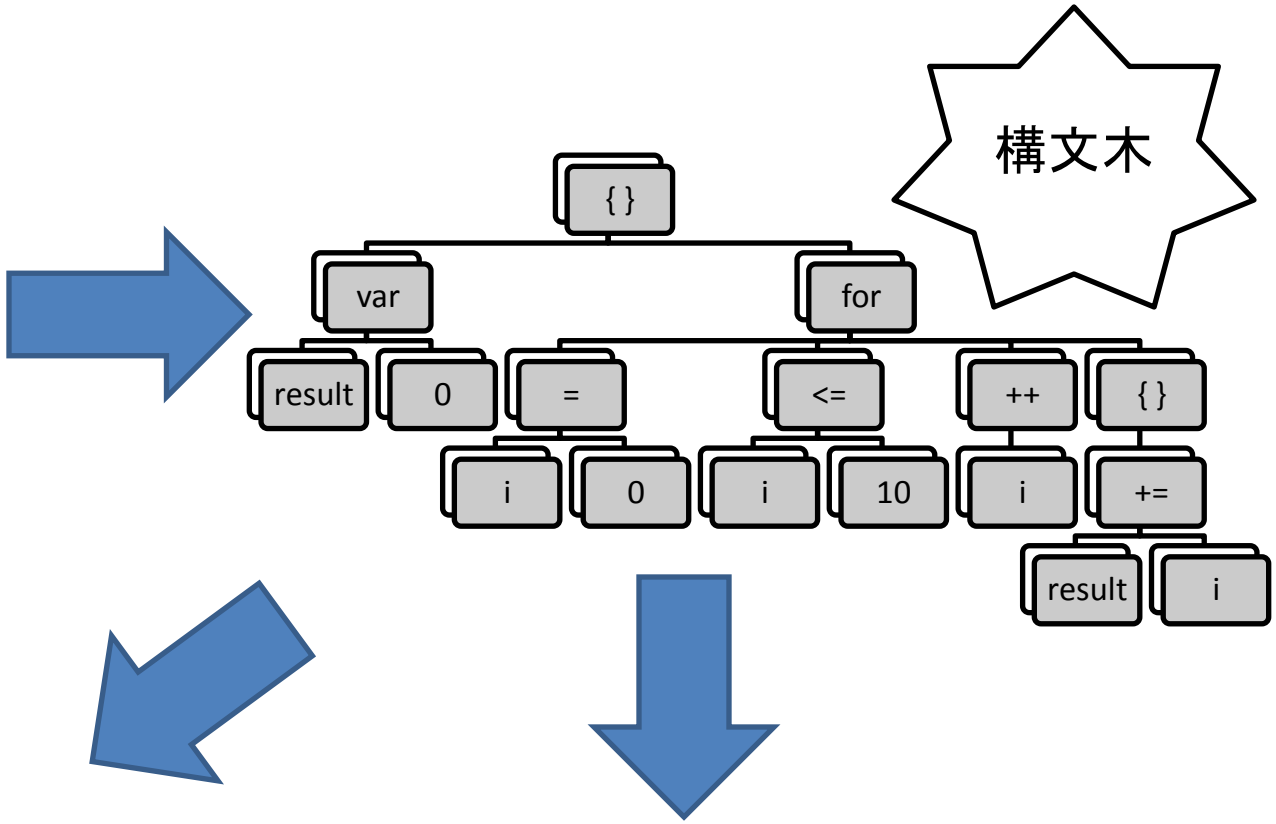
DLR の基盤技術

動的オブジェクト型	• 実行時にプロパティやメソッドを追加・削除することが可能 • 名前によるメンバアクセスを簡素化
名前管理	• 各言語間の名前（変数名、クラス名などなど）の相互運用を助ける辞書的な機構
式ツリー（拡張版）	• **条件式などの内部表現** • **条件判断やループなどの制御構造が追加された**

※ 正式な発表がまだ行われていないので、上図はかなりアバウトなものです。

「よくある」
プログラミング言語の処理フロー

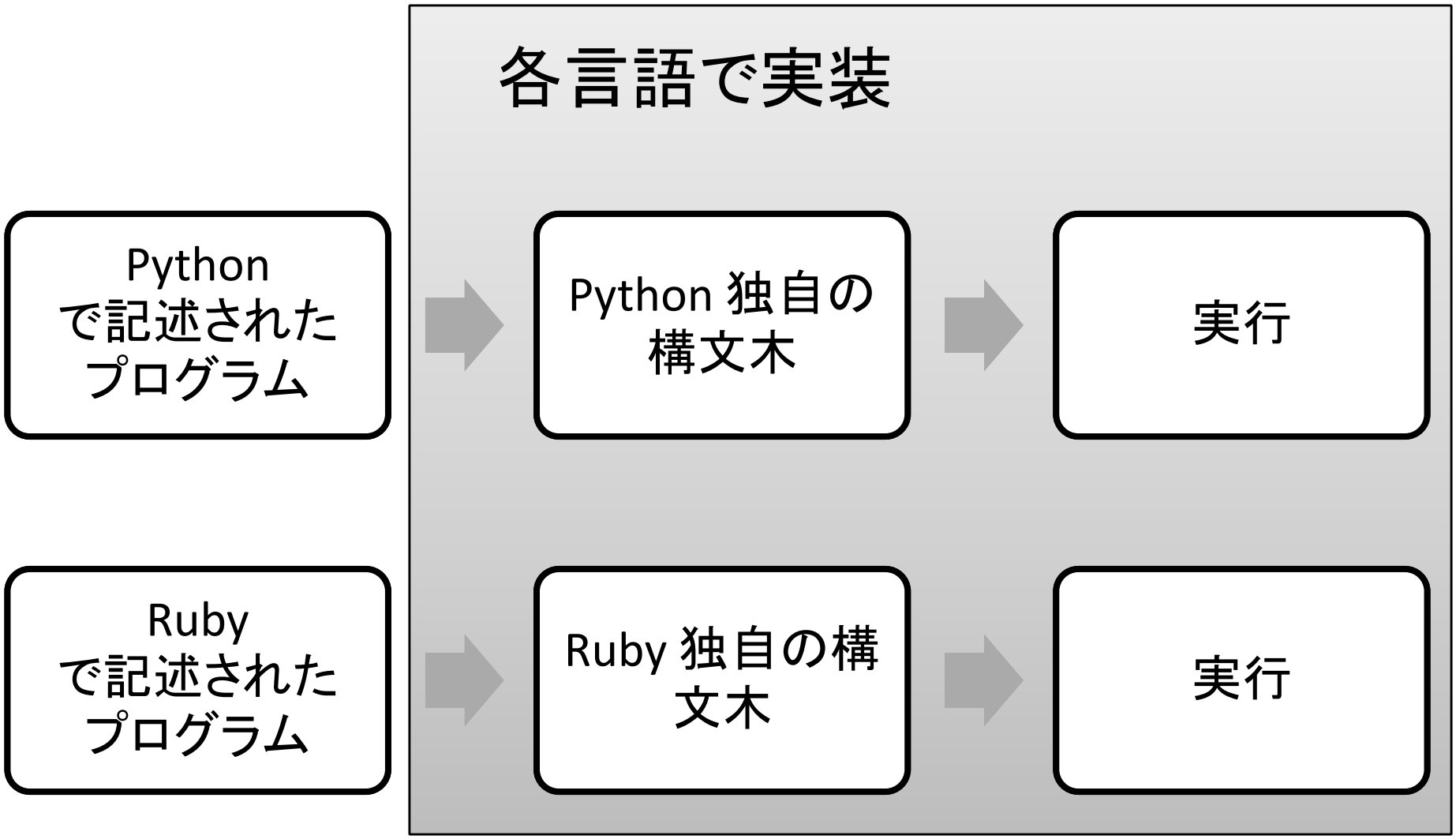
```
var result = 0;

for (i = 1; i <= x; i++)
{
   result += i;
}
```



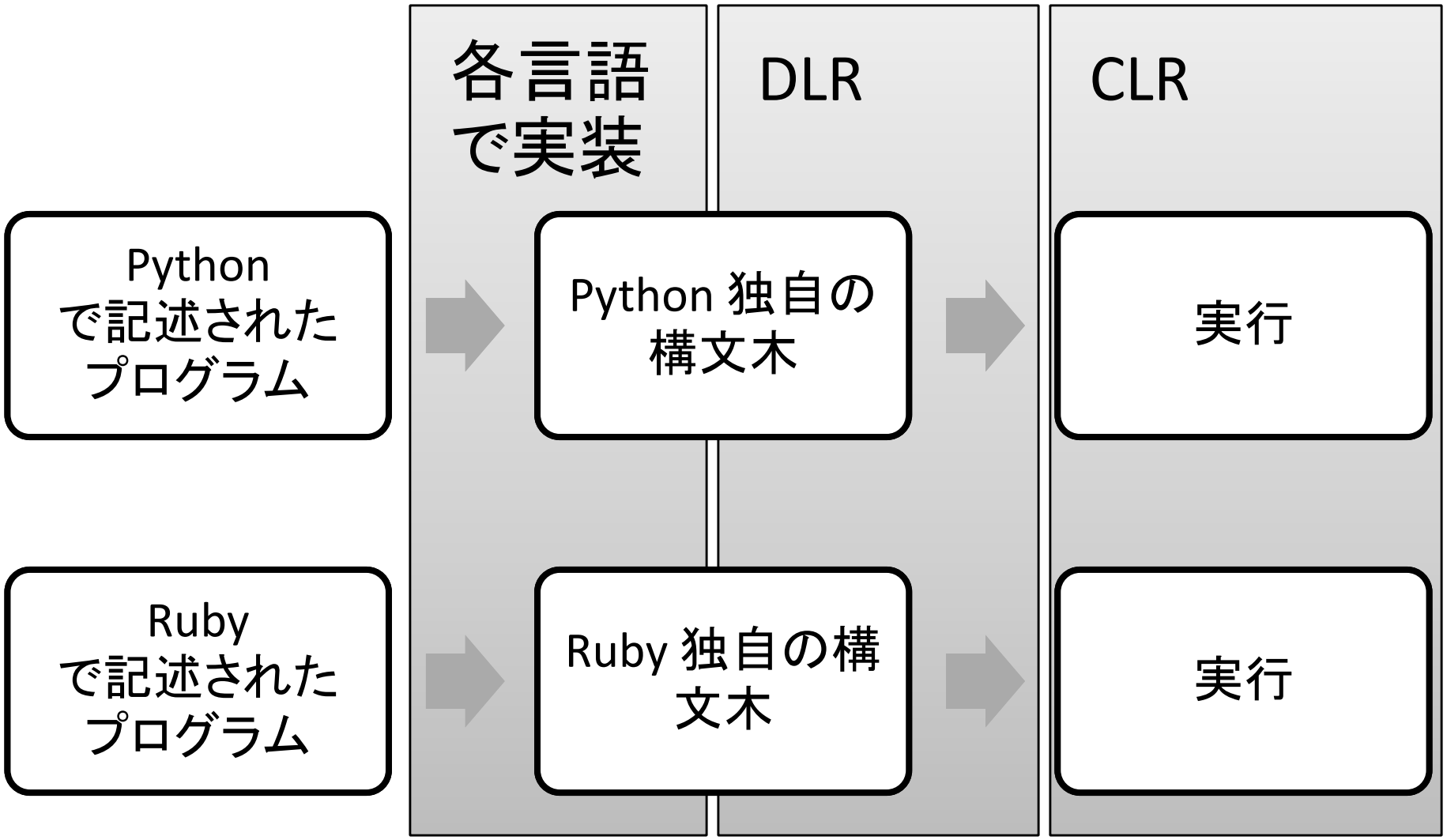
その場で解釈実行
→動的言語

実行形式を生成
→ 静的言語（コンパイラ）

DLR 登場以前


各言語で実装
Python
で記述された
プログラム
Python 独自の
構文木
実行
Ruby
で記述された
プログラム
Ruby 独自の構
文木
実行

DLR 登場以後



※ 少し大げさに表現しています。

System.Linq.Expressions 名前空間に対する拡張

- 条件判断やループなど、制御構造を表わすための Expression クラス派生型が追加された
- System.Linq.Expressions 名前空間配下のクラス数：２１個→３７個（１６個増）
- 「構文木」は、実行時に匿名デリゲートに変換することができる

→制御構造を含んだ、いわゆる「メソッド」を動的に作成することが可能

構文木（拡張された式ツリー）の応用

- 制御構造を含む計算式を動的に作成、実行
  - 再帰ではマズイような場合？
- おれおれスクリプト言語の作成基盤
  - 他言語との相互運用を考えなければ、DLR のすべての機能を使わなくてもおk？
- アプリケーションの「ふるまい」を動的に変更
  - ちょっと漠然としてる？？

デモ

おれおれスクリプト言語的なモノを作ってみた。

まとめ

渋木宏明(ひどり)の結論

- .NET Framework 4.0 世代の Expression Trees は、メタプログラミングのための強力なツール
- DLR の下請けとして埋もれさせるのはモッタイナイ！
- アプリケーション上層で頻繁に使うものではないが、「ハマった」時の効果は絶大なものがある(かもしれない)
- Visual Studio 2010 Beta2 リリースの噂が囁かれているので、余裕のある人は是非お試しを

リソース

- MSDN ライブラリ
  - System.Linq.Expressions 名前空間 (3.5SP1)
    http://msdn.microsoft.com/ja-jp/library/system.linq.expressions.aspx
  - System.Linq.Expressions 名前空間 (4.0Beta1)
    http://msdn.microsoft.com/en-us/library/system.linq.expressions(VS.100).aspx
- 岩永(ufcpp)さんのブログ
  - 式木
    http://ufcpp.net/study/csharp/sp3_expression.html
- 湯川（NyaRuRu）さんのブログ
  - 全てが式になる，全てが木になる，全てが式木になる
    http://d.hatena.ne.jp/NyaRuRu/20071230/p1

そろそろ起きる時間ですよ (^o^)

Q&A

fin.

ご静聴ありがとうございました