# R流・C#3.0 LINQとラムダ式で遊ぼう

2008年06月21日

R・田中一郎

http://blogs.wankuma.com/rti/


Microsoft MVP for Development Tools - Visual C#

(April 2007 - March 2009)

# 自己紹介

会員番号:34

名前:R･田中一郎

所在:栃木県

年齢:18才

職業:

主に業務用
システムの開発

2005年11月
Microsoft Visual Studio .NET デビュー。
この頃から R.Tanaka.Ichiro と名のりネットでアクティブに活動を始める。

2006年02月
C# を学び始める。理想的な言語に感動。尊敬する方々の影響も大きい。

2006年09月
わんくま同盟加盟
ある事件がきっかけで、中さんから声をかけていただき加盟。

2006年11月
MSC2006 にて R･田中一郎として始めて人前に姿を晒す。

2007年04月
Microsoft MVP for Visual Developer - Visual C# を受賞。

2007年06月
わんくま同盟勉強会にてスピーカーデビュー。

2008年04月
Microsoft MVP for Development Tools - Visual C# を受賞。

2008年05月
70-526,70-536 試験をパス。
Microsoft Certified Technology Specialist for .Net Framework 2.0: Windows Applications 資格取得。

# アジェンダ

- はじめに
- ラムダ式とは？
- ラムダ式で遊んでみる
- LINQに欠かせないラムダ式
- 最後に

本セッションの目的

ラムダ式やLINQで遊ぶことで、

ラムダ式やLINQを理解して、

使いこなせるようになろう

# ラムダ式とは？

- 匿名メソッドの記法

```
func<int, int, int> f  = (x, y) => x + y;
```

- 式ツリー型に代入できる

```
Expression<func<int, int, int>> e = (x, y) => x + y;
```

# ラムダ式とは？　－　匿名メソッドの記法について

通常のメソッド

```
private int Method(int x, int y) { return x + y; }
```

匿名メソッド

```
Func<int, int, int> method =
    delegate(int x, int y) { return x + y; }
```

ラムダ式

```
Func<int, int, int> method =
    (int x, int y) => { return x + y; }
Func<int, int, int> method = (x, y) => x + y;
```

# ラムダ式とは？　－　ラムダ式を使ってみる

```
private void Plus()     { this.Calculate((a, b) => a + b); }
private void Subtract() { this.Calculate((a, b) => a - b); }
private void Multiply() { this.Calculate((a, b) => a * b); }
private void Divide()   { this.Calculate((a, b) => a / b); }

private void Calculate(Func<int, int, int> func) {
    foreach(var x in this.Collection) {
        x.Result = func(x.Value1, x.Value2);
        Console.WriteLine(x.Result);
    }
}
```

# ラムダ式とは？　ー　式ツリーについて

```
public void Plus() {
    Expression<Func<int, int, int>> e = (a, b) => a + b;

    var bin = (BinaryExpression) e.Body;
    var v1 = (ParameterExpression) bin.Left;
    var v2 = (ParameterExpression) bin.Right;
    Console.WriteLine(e);    // (a + b)
    Console.WriteLine(v1);  // a
    Console.WriteLine(v2);   // b
    this.Calculate(ex.Compile());
}
```

# ラムダ式で遊ぼう！ー　Override

Override を使った場合

```
class Base {
    public virtual int Method(int a, int b) {}
}
class HogeHoge {
    public override int Method(int a, int b) {
        return a + b;
    }
}
```

ラムダ式を使って同じことを実現してみる

```
class Base {
    public Func<int, int, int> Method;
}
class HogeHoge {
    public HogeHoge() {
        this.Method = (a, b) => a + b;
    }
}
```

# ラムダ式で遊ぼう！　ー　Template Method パターン

```
public void Method(Action methodB) {
    this.MethodA();
     methodB();
    this.MethodC();
}
```

# ラムダ式で遊ぼう！　－　Observer パターン

```
public class Subject : INotifyPropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;
    private int _Value = 0;
    public int Value {
        get { return this._Value; }
        set {
            if (this._Value == value) return;
            this._Value = value;
            if (PropertyChanged == null) return;
            PropertyChanged(this, new PropertyChangedEventArgs(“Value”));
        }
    }
}
public class Observer {
    public Observer() {
        var subject = new Subject();
        subject.PropertyChanged += (sender, e) => MessageBox(“Changed!”);
    }
}
```

# ラムダ式で遊ぼう！　－　Strategyパターン

```
(this.IsAdd ?
        new Action(this.AddData) :
        new Action(this.UpdateData))();


(this.IsAdd ?
        new Action(() => {
                ・・・ 追加処理}) :
        new Action(() => {
                ・・・ 更新処理}) ();
```

# ラムダ式で遊ぼう！　ー　コレクションに格納する

```
var calc = new Dictionary<string, <Func<int, int, int>>() {
    “Plus”      ,new Func<int, int, int>((a, b) => a + b),
    “Subtract  ,new Func<int, int, int>((a, b) => a - b),
    “Multiply “,new Func<int, int, int>((a, b) => a * b),
    “Divide”    ,new Func<int, int, int>((a, b) => a / b),
}

foreach(var x in this.Collection) {
        x.Result = calc[“Plus”](x.Value1, x.Value2);
        Console.WriteLine(x.Result);
}
```

# LINQで遊ぼう！ー　LINQ to Objects(1)

```
public Form1() {
    InitializeComponent();
    var c = new[] {
            new { Code = 51 , Name = “ぽぴ王子”, Age = 18 },
            new { Code = 34 , Name = “R・田中一郎”, Age = 18 },
            new { Code = 111, Name = "IIJIMAS”. Age = 20 }};
    var q1 =
        from x in c
        where x.Age == 18
        select new { Code = x.Code, Name = x.Name };
    var q2 = c
        .Where(x => x.Age == 18)
        .Select(x => new { Code = x.Code, Name = x.Name });
    var q3 = c
        .条件(x => x.Age == 18)
        .選択(x => new { Code = x.Code, Name = x.Name });
    listBox1.Binding(q1.ToArray(), "Code", "Name");
    listBox2.Binding(q2.ToArray(), "Code", "Name");
    listBox3.Binding(q3.ToArray(), "Code", "Name");
}
```

# LINQで遊ぼう！ー　LINQ to Objects(3)

```
static void Binding(
      this ListControl  x, object dataSource, string valueMember, string displayMember)
{
    x.DataSource = dataSource;
    x.ValueMember = valueMember;
    x.DisplayMember = displayMember;
}

public static IEnumerable<T> 条件<T>(this IEnumerable<T> value, Func<T, bool> func) {
    var r = new List<T>();
    foreach(var x in value) if (func(x)) r.Add(x);
    return r;
}

public static IEnumerable<TR>選択<TS, TR>(this IEnumerable<TS> value, Func<TS, TR> func) {
    var r = new List<TR>();
    foreach(var x in value) r.Add(func(x));
    return r;
}
```

# LINQで遊ぼう！ー　LINQ to HogeHoge(1)

```
using System.Windows.Forms;
using R.Tanaka.Ichiro.Linq.HogeHoge;
//using System.Linq;

namespace WindowsFormsApplication1 {
    public partial class LINQtoHogeHogeForm : Form {
        public LINQtoHogeHogeForm() {
            InitializeComponent();
            using (var dt = new MemberDataSet.MemberDataTable()) {
                var q = from p in dt where p.Age == 18
                    select new { p.Code, p.Name };
                this.grid.DataSource = q.ToList();
            }
        }
    }
}
```

# LINQで遊ぼう！ー　LINQ to HogeHoge(2)

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Data.Common;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace R.Tanaka.Ichiro.Linq.HogeHoge {
    public static class LinqToHogeHoge {
        public static IEnumerable<T> Where<T>(
             this IEnumerable<T> value, Expression<Func<T, bool>> lambdaExpression)
        {
            var dataTable = value as DataTable;
            var sql = GetSqlText(dataTable.TableName, lambdaExpression.Body);
            FillBySqlText(dataTable, sql);
            return dataTable.Rows.Cast<T>();
        }

        // インチキ SQL TEXT 生成ロジック
        // 実際は、各ノードのタイプに応じて再帰しながら条件式を組み立てるか、
        // ソースのプロバイダのプロバイダの CreateQuery() メソッドを使う
        private static string GetSqlText(string tableName, Expression expressionBody) {
            return String.Format("SELECT * FROM {0} WHERE {1}", tableName, GetWhere(expressionBody));
        }
```

# LINQで遊ぼう！ー　LINQ to HogeHoge(2)

```
private static string GetWhere(Expression node) {
  var bin = node       as BinaryExpression;
  var v1 = bin.Left    as MemberExpression;
  var v2 = bin.Right   as ConstantExpression;
  var op =
          bin.NodeType == ExpressionType.Equal              ? "=" :
          bin.NodeType == ExpressionType.GreaterThan        ? ">" :
          bin.NodeType == ExpressionType.LessThan           ? "<" : "";
          //  ・・・・・続く
  var v = v2.Value.ToString();
  if (v2.Type.IsClass) v = "¥'" + v + "¥'";
  return v1.Member.Name  + op + v;
}

private static void FillBySqlText(DataTable dataTable, string sqlText) {
      using (var ad = new OleDbDataAdapter()) {
          var map = new DataTableMapping();
          map.SourceTable = "Table";
          map.DataSetTable = dataTable.TableName;
          foreach(DataColumn x in dataTable.Columns)
              map.ColumnMappings.Add(x.ColumnName, x.ColumnName);
          ad.TableMappings.Add(map);
          var cs = global::WindowsFormsApplication1.Properties.Settings.Default.MemberConnectionString;
          using (var cn = new OleDbConnection(cs)) {
              cn.Open();
```

# LINQで遊ぼう！ー　LINQ to HogeHoge(2)

```
                using (var cd = new OleDbCommand()) {
                        cd.Connection                = cn;
                        cd.CommandText               = sqlText;
                        cd.CommandType               = CommandType.Text;
                        ad.SelectCommand = cd;
                        dataTable.Clear();
                        ad.Fill(dataTable);
                }
                cn.Close();
            }
        }
    }
    public static IEnumerable<TR> Select<TS, TR>(
            this IEnumerable<TS> value, Func<TS, TR> func) {
            var r = new List<TR>();
            foreach(var x in value) r.Add(func(x));
            return r;
    }

    public static List<T> ToList<T>(this IEnumerable<T> value) {
            var list = new List<T>();
            foreach(var x in value) list.Add(x);
            return list;
    }
}
```