無料電子ブック

学習

.NET Framework

Free unaffiliated eBook created from
Stack Overflow contributors.

#.net

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: -net-framework

It is an unofficial and free .NET Framework ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official .NET Framework.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: .NET Frameworkをいめる

.NET Frameworkは、もともとマイクロソフトがしたのライブラリとランタイムです。すべての.NETプログラムは、Microsoft Intermediate LanguageMSILとばれるバイトコードにコンパイルされます。 MSILはCommon Language RuntimeCLRによってされます。

に、.NET Frameworkをサポートするさまざまなの「Hello World」のをいくつかします。「Hello World」は、デバイスに「Hello World」をするプログラムです。これは、のプログラムをするためのなをするためにされます。また、のコンパイラ、、およびがすべてしくすることをするためのテストとしてもできます。

.NETでサポートされているのリスト

バージョン

。ネット

バージョン	
1.0	2002-02-13
1.1	2003-04-24
2.0	2005117
3.0	2006-11-06
3.5	2007-11-19
3.5 SP1	2008811
4.0	2010-04-12
4.5	2012-08-15
4.5.1	20131017
4.5.2	2014-05-05
4.6	2015-07-20
4.6.1	2015-11-17
4.6.2	2016-08-02
4.7	2017-04-05

コンパクトフレームワーク

バージョン	
1.0	2000-01-01
2.0	2005-10-01
3.5	2007-11-19
3.7	2009-01-01
3.9	2013-06-01

マイクロフレームワーク

バージョン	
4.2	2011104
4.3	2012-12-04
4.4	2015-10-20

Examples

CのHello World

```
using System;

class Program
{
    // The Main() function is the first function to be executed in a program
    static void Main()
    {
        // Write the string "Hello World to the standard out
        Console.WriteLine("Hello World");
    }
}
```

`Console.WriteLine`はいくつかのオーバーロードがあります。この、 "Hello World"がパラメータであり、に "Hello World"をストリームにします。のオーバーロードは、ストリームにきむにの`.ToString`をびすことがあります。については、 .NET Frameworkのマニュアルをしてください。

.NET Fiddleでのライブデモ

C

Visual Basic .NETのHello World

```
Imports System

Module Program
    Public Sub Main()
        Console.WriteLine("Hello World")
    End Sub
End Module
```

.NET Fiddleでのライブデモ

Visual Basic .NETの

Hello World in F

```
open System

[<EntryPoint>]
let main argv =
    printfn "Hello World"
    0
```

.NET Fiddleでのライブデモ

Fの

C ++ / CLIでのHello World

```
using namespace System;

int main(array<String^>^ args)
{
    Console::WriteLine("Hello World");
}
```

PowerShellのHello World

```
Write-Host "Hello World"
```

PowerShell

Hello World inネメルル

```
System.Console.WriteLine("Hello World");
```

Hello World in Oxygene

```
namespace HelloWorld;

interface

type
  App = class
  public
    class method Main(args: array of String);
  end;

implementation

class method App.Main(args: array of String);
begin
  Console.WriteLine('Hello World');
end;

end.
```

ハロー.イン.イン.ブー

```
print "Hello World"
```

Hello World in PythonIronPython

```
print "Hello World"
```

```
import clr
from System import Console
Console.WriteLine("Hello World")
```

Hello World in IL

```
.class public auto ansi beforefieldinit Program
       extends [mscorlib]System.Object
{
  .method public hidebysig static void  Main() cil managed
  {
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr      "Hello World"
    IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b:  nop
    IL_000c:  ret
  }

  .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  ret
  }
```

```
}
```

オンラインで.NET Frameworkをいめるをむ https://riptutorial.com/ja/dot-net/topic/14/-net-frameworkをいめる

2: .NETコア

き

.NET Coreは、MicrosoftとGitHubの.NETコミュニティによってされるプラットフォームです。これはクロスプラットフォームであり、Windows、macOS、Linuxをサポートし、デバイス、クラウド、みみ/ IoTシナリオでできます。

.NETコアをえるときは、フレキシブルな、クロスプラットフォーム、コマンドラインツール、オープンソースなど、のことをえてください。

もうつのきなは、たとえそれがオープンソースであっても、マイクロソフトはにそれをサポートしているということです。

それでは、.NET Coreにはツール、ローカルサービス、テキストベースのゲームになのアプリケーションモデルコンソールアプリケーションがまれています。そののアプリケーションモデルは.NETコアのにされており、のようなをしています。

- ASP.NETコア
- Windows 10ユニバーサルWindowsプラットフォームUWP
- Xamarin.Forms

また、.NET Coreは.NET Standard Libraryをしているため、.NET Standard Librariesをサポートしています。

.NETライブラリは、が.NETでできるしたの.NET APIをするAPIです。 .NETでは、.NETライブラリにし、.NETライブラリをターゲットとするライブラリをサポートするために、このをするがあります。

Examples

なコンソールアプリケーション

```
public class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("\nWhat is your name? ");
        var name = Console.ReadLine();
        var date = DateTime.Now;
        Console.WriteLine("\nHello, {0}, on {1:d} at {1:t}", name, date);
        Console.Write("\nPress any key to exit...");
        Console.ReadKey(true);
    }
}
```

オンラインで.NETコアをむ https://riptutorial.com/ja/dot-net/topic/9059/-netコア

3: .NETフレームワークをした

き

このトピックは、.NETフレームワークでタスクライブラリをするマルチコアプログラミングについてです。タスクライブラリをすると、がめるコードをし、なコアのでることができます。したがって、ソフトウェアがアップグレードでアップグレードされることをにすることができます。

Examples

パラレルは、データパラレルをするために、タスクライブラリとともにされています。データとは、ソース・コレクションまたはのにしてじがしてつまりしてされるシナリオをします。 .NETは、Parallel.ForおよびParallel.Foreachをしてデータのをするしいをします。

```
//Sequential version

foreach (var item in sourcecollection){

Process(item);

}

// Parallel equivalent

Parallel.foreach(sourcecollection, item => Process(item));
```

のParallel.ForEachは、のコアをしており、じようにパフォーマンスをさせます。

オンラインで.NETフレームワークをしたをむ https://riptutorial.com/ja/dot-net/topic/8085/-netフレームワークをした

4: ADO.NET

き

ADOActiveX Data Objects.Netは、Microsoftがするツールであり、SQL Server、Oracle、XMLなどのデータソースにコンポーネントをしてアクセスします。 .Netフロントエンドアプリケーションは、なをつADO.Netをしてデータソースにすると、データの、、およびをうことができます。

ADO.Netは、コネクションレスのアーキテクチャをします。はセッションでするはないため、データベースとのやりりはなです。

Parameters.AddWithValueをして**SQL**を**Parameters.AddWithValue**するの AddWithValueはしていではありません。このでは、されたものからデータのタイプをすることにします。これにより、によってをできなくなることがあります。 char / varchar する「n」をvarchar やdateなどのSQL Serverのデータには、する.NETデータがないことにしてください。そのようなは、 わりにしいデータのAddをするがあります 。

Examples

SQLをコマンドとしてする

```
// Uses Windows authentication. Replace the Trusted_Connection parameter with
// User Id=...;Password=...; to use SQL Server authentication instead. You may
// want to find the appropriate connection string for your server.
string connectionString =
@"Server=myServer\myInstance;Database=myDataBase;Trusted_Connection=True;"

string sql = "INSERT INTO myTable (myDateTimeField, myIntField) " +
    "VALUES (@someDateTime, @someInt);";

// Most ADO.NET objects are disposable and, thus, require the using keyword.
using (var connection = new SqlConnection(connectionString))
using (var command = new SqlCommand(sql, connection))
{
    // Use parameters instead of string concatenation to add user-supplied
    // values to avoid SQL injection and formatting issues. Explicitly supply datatype.

    // System.Data.SqlDbType is an enumeration. See Note1
    command.Parameters.Add("@someDateTime", SqlDbType.DateTime).Value = myDateTimeVariable;
    command.Parameters.Add("@someInt", SqlDbType.Int).Value = myInt32Variable;

    // Execute the SQL statement. Use ExecuteScalar and ExecuteReader instead
    // for query that return results (or see the more specific examples, once
    // those have been added).

    connection.Open();
    command.ExecuteNonQuery();
}
```

1 MSFT SQL Serverのバリエーションの[SqlDbType]をしてください。

2 MySQLのバリエーションについては、 [MySqlDbType]をしてください。

ベストプラクティス - SQLの

```
public void SaveNewEmployee(Employee newEmployee)
{
    // best practice - wrap all database connections in a using block so they are always
closed & disposed even in the event of an Exception
    // best practice - retrieve the connection string by name from the app.config or
web.config (depending on the application type) (note, this requires an assembly reference to
System.configuration)
    using(SqlConnection con = new
SqlConnection(System.Configuration.ConfigurationManager.ConnectionStrings["MyConnectionName"].Connectio

    {
        // best practice - use column names in your INSERT statement so you are not dependent
on the sql schema column order
        // best practice - always use parameters to avoid sql injection attacks and errors if
malformed text is used like including a single quote which is the sql equivalent of escaping
or starting a string (varchar/nvarchar)
        // best practice - give your parameters meaningful names just like you do variables in
your code
        using(SqlCommand sc = new SqlCommand("INSERT INTO employee (FirstName, LastName,
DateOfBirth /*etc*/) VALUES (@firstName, @lastName, @dateOfBirth /*etc*/)", con))
        {
            // best practice - always specify the database data type of the column you are
using
            // best practice - check for valid values in your code and/or use a database
constraint, if inserting NULL then use System.DbNull.Value
            sc.Parameters.Add(new SqlParameter("@firstName", SqlDbType.VarChar, 200){Value =
newEmployee.FirstName ?? (object) System.DBNull.Value});
            sc.Parameters.Add(new SqlParameter("@lastName", SqlDbType.VarChar, 200){Value =
newEmployee.LastName ?? (object) System.DBNull.Value});

            // best practice - always use the correct types when specifying your parameters,
Value is assigned to a DateTime instance and not a string representation of a Date
            sc.Parameters.Add(new SqlParameter("@dateOfBirth", SqlDbType.Date){ Value =
newEmployee.DateOfBirth });

            // best practice - open your connection as late as possible unless you need to
verify that the database connection is valid and wont fail and the proceeding code execution
takes a long time (not the case here)
            con.Open();
            sc.ExecuteNonQuery();
        }

        // the end of the using block will close and dispose the SqlConnection
        // best practice - end the using block as soon as possible to release the database
connection
    }
}

// supporting class used as parameter for example
public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
```

```
    public DateTime DateOfBirth { get; set; }
}
```

ADO.NETでののベストプラクティス

- はのでをくことです。プロシ―ジャのがこれをえると、オブジェクトをプ―ルにすようにをじます。のプ―ルのサイズ= 100.プ―ルにより、SQL Serverへののパフォ―マンスがします。 SQL Serverでのプ―リング
- のブロックのすべてのデ―タベ―スをラップして、がしてもにじられてされるようにします。 ステ―トメントののについては、 usingステ―トメントCリファレンスをしてください。
- app.configまたはweb.configからをでしますアプリケ―ションのによってなります
  - これには、 `System.configuration`へのアセンブリがです
  - ファイルののは、「 とファイル 」をしてください。
- のパラメ―タはにする
  - SQLインジェクションをける
  - のエスケ―プまたはvarchar / nvarcharのsqlにするをむようなのテキストをすると、エラ―をできます。
  - デ―タベ―ス･プロバイダに、すべてのデ―タベ―ス･プロバイダでサポ―トされていないせをさせることで、がします。
- パラメ―タをする
  - SQLパラメ―タのとサイズのは、//ののなです
  - コ―ドのとじように、Sqlパラメ―タののあるをけてください
  - しているのデ―タベ―スのデ―タをします。これによりったパラメ―タがされないため、しないがするがあります。
  - コマンドにすに、したパラメ―タをします「 ガ―ベッジ･イン、ガ―ベッジ･アウト 」というように。できるだけくスタックにってくるをする
  - たとえば、DateTimeのをりてず、のDateTimeインスタンスをパラメ―タのにするなど、パラメ―タをりてるときにしいをします
  - パラメ―タのサイズをします。これは、パラメ―タのとサイズがする、SQL Serverはをできるためです。 MAXに-1をする
  - AddWithValueメソッドをしないでください。なは、にじてパラメ―タタイプやﾉスケ―ルをするのをれてしまうことです。については、にAddWithValueのをめることができますかをしてください。
- デ―タベ―スをする
  - できるだけくをいて、できるだけくじてください。これは、リソ―スをするのなガイドラインです
  - デ―タベ―スインスタンスをしないでくださいシングルトンホストに`SqlConnection`のインスタンスがある。にじてコ―ドでにしいデ―タベ―スインスタンスをしてから、びしコ―ドをし、したら「 」します。そのはのとおりです。
    - ほとんどのデ―タベ―スプロバイダにはらかのプ―リングがあり、しいをするのはです
    - コ―ドがのスレッドでをすると、のエラ―もなくなります。

ベンダーのクラスをするためのインターフェースの

```
var providerName = "System.Data.SqlClient";    //Oracle.ManagedDataAccess.Client, IBM.Data.DB2
var connectionString = "{your-connection-string}";
//you will probably get the above two values in the ConnectionStringSettings object from
.config file

var factory = DbProviderFactories.GetFactory(providerName);
using(var connection = factory.CreateConnection()) {    //IDbConnection
    connection.ConnectionString = connectionString;
    connection.Open();

    using(var command = connection.CreateCommand()) {    //IDbCommand
        command.CommandText = "{query}";

        using(var reader = command.ExecuteReader()) {    //IDataReader
            while(reader.Read()) {
                ...
            }
        }
    }
}
```

オンラインでADO.NETをむ https://riptutorial.com/ja/dot-net/topic/3589/ado-net

5: ASP.NETのスマートなをしたASP.NET MVCのグローバリゼーション

ASP.NETページのスマートな

このアプローチのは、コントローラやのクラスを.resxファイルからをするコードでさせるがないことです。 `HttpModule`は.poファイルのをしてられたテキストをきえます。がつかった、`HttpModule`はをきえます。がつからないは、がされ、されていないのテキストがページにされます。

.poファイルは、アプリケーションのをするためのなであるため、それらをするためにできるいくつかのアプリケーションがあります。 .poファイルをのユーザーにして、をすることはです。

Examples

なとセットアップ

1. MVCプロジェクトにI18Nナゲットパッケージをします。
2. web.configで、 `<httpModules>` `<modules>`セクションまたは`<modules>`セクションに `i18n.LocalizingModule`をします。

```
<!-- IIS 6 -->
<httpModules>
  <add name="i18n.LocalizingModule" type="i18n.LocalizingModule, i18n" />
</httpModules>

<!-- IIS 7 -->
<system.webServer>
  <modules>
    <add name="i18n.LocalizingModule" type="i18n.LocalizingModule, i18n" />
  </modules>
</system.webServer>
```

3. "locale"というのフォルダをサイトのルートにします。サポートするカルチャーのサブフォルダをします。たとえば、 `/locale/fr/`ます。
4. カルチャーのフォルダに`messages.po`というのテキストファイルをし`messages.po`。
5. テストので、 `messages.po`ファイルにのテキストをします。

```
#: Translation test
msgid "Hello, world!"
msgstr "Bonjour le monde!"
```

6. プロジェクトにするテキストをすコントローラをします。

```
using System.Web.Mvc;
```

```
namespace I18nDemo.Controllers
{
    public class DefaultController : Controller
    {
        public ActionResult Index()
        {
            // Text inside [[[triple brackets]]] must precisely match
            // the msgid in your .po file.
            return Content("[[[Hello, world!]]]");
        }
    }
}
```

7. MVCアプリケーションをし、 http// localhost[yourportnumber] / defaultなどのコントローラアクションにするルートをします 。
   URLがデフォルトのカルチャをするようにされていることをしてください。
   http// localhost[yourportnumber] / en / default 。
8. URLに`/en/`を`/fr/` またはあなたがしたときえます。ページにはされたテキストがされます。
9. ブラウザのをしてのカルチャをし、 `/default`してください。カルチャーをするようにURLがされ、されたテキストがされることをしてください。
10. web.configでは、ユーザーが`locale`フォルダをできないようにハンドラをします。

```
<!-- IIS 6 -->
<system.web>
  <httpHandlers>
    <add path="*" verb="*" type="System.Web.HttpNotFoundHandler"/>
  </httpHandlers>
</system.web>

<!-- IIS 7 -->
<system.webServer>
  <handlers>
    <remove name="BlockViewHandler"/>
   <add name="BlockViewHandler" path="*" verb="*" preCondition="integratedMode"
type="System.Web.HttpNotFoundHandler"/>
  </handlers>
</system.webServer>
```

オンラインでASP.NETのスマートなをしたASP.NET MVCのグローバリゼーションをむ
https://riptutorial.com/ja/dot-net/topic/5086/asp-netのスマートなをしたasp-net-mvcのグローバリゼーション

6: CでSHA1をする

き

このプロジェクトでは、SHA1ハッシュをするをています。えば、からハッシュをし、SHA1ハッシュをするです。 git hubのソース https : //github.com/mahdiabasi/SHA1Tool

Examples

ファイルのSHA1チェックサムをする

まず、System.Security.CryptographyとSystem.IOをプロジェクトにします

```
public string GetSha1Hash(string filePath)
    {
        using (FileStream fs = File.OpenRead(filePath))
        {
            SHA1 sha = new SHA1Managed();
            return BitConverter.ToString(sha.ComputeHash(fs));
        }
    }
```

オンラインでCでSHA1をするをむ https://riptutorial.com/ja/dot-net/topic/9457/c-でsha1をする

7: CでSHA1をする

き

このプロジェクトでは、SHA1ハッシュをするをています。えば、からハッシュをし、SHA1ハッシュをするです。

githubのソースcompelete https : //github.com/mahdiabasi/SHA1Tool

Examples

ファイルのSHA1チェックサムをする

まずSystem.Security.Cryptographyをプロジェクトにします

```
public string GetSha1Hash(string filePath)
    {
        using (FileStream fs = File.OpenRead(filePath))
        {
            SHA1 sha = new SHA1Managed();
            return BitConverter.ToString(sha.ComputeHash(fs));
        }
    }
```

テキストのハッシュをする

```
  public static string TextToHash(string text)
    {
        var sh = SHA1.Create();
        var hash = new StringBuilder();
        byte[] bytes = Encoding.UTF8.GetBytes(text);
        byte[] b = sh.ComputeHash(bytes);
        foreach (byte a in b)
        {
            var h = a.ToString("x2");
            hash.Append(h);
        }
        return hash.ToString();
    }
```

オンラインでCでSHA1をするをむ https://riptutorial.com/ja/dot-net/topic/9458/c-でsha1をする

8: CLR

Examples

ランタイムの

Common Language RuntimeCLRは、マシンで、.NET Frameworkのです。をむ

- **Common Intermediate Language** してCILまたはILとばれるなバイトコーードは、
- マシンコーードをするJust-In-Timeコンパイラ
- メモリをするトレースガベージコレクタ
- AppDomainsとばれるサブプロセスのサポート
- なコーードとレベルのによるセキュリティメカニズム

CLRでされるコーードは、されていないコーードとばれるCLRはネイティブコーードのでされるコーードとするために、マネージコーードとばれます。コーードとコーードのをするさまざまなメカニズムがあります。

オンラインでCLRをむ https://riptutorial.com/ja/dot-net/topic/3942/clr

9: DateTimeの

Examples

ParseExact

```
var dateString = "2015-11-24";

var date = DateTime.ParseExact(dateString, "yyyy-MM-dd", null);
Console.WriteLine(date);
```

2015112412:00:00 AM

`CultureInfo.CurrentCulture`を3のパラメ―タとしてすことは、 `null`をすこととじことにしてください。または、のにすることができます。

フォ―マット

は、にするのにすることができます

```
var date = DateTime.ParseExact("24|201511", "dd|yyyyMM", null);
Console.WriteLine(date);
```

2015112412:00:00 AM

ではないはすべてリテラルとしてわれます

```
var date = DateTime.ParseExact("2015|11|24", "yyyy|MM|dd", null);
Console.WriteLine(date);
```

2015112412:00:00 AM

の

```
var date = DateTime.ParseExact("2015-01-24 11:11:30", "yyyy-mm-dd hh:MM:ss", null);
Console.WriteLine(date);
```

2015112411:01:30

とのがったにされたことにしてください。

*1*のはの1つでなければなりません

```
var date = DateTime.ParseExact("11/24/2015", "d", new CultureInfo("en-US"));
var date = DateTime.ParseExact("2015-11-24T10:15:45", "s", null);
var date = DateTime.ParseExact("2015-11-24 10:15:45Z", "u", null);
```

ArgumentNullException

```
var date = DateTime.ParseExact(null, "yyyy-MM-dd", null);
var date = DateTime.ParseExact("2015-11-24", null, null);
```

FormatException

```
var date = DateTime.ParseExact("", "yyyy-MM-dd", null);
var date = DateTime.ParseExact("2015-11-24", "", null);
var date = DateTime.ParseExact("2015-0C-24", "yyyy-MM-dd", null);
var date = DateTime.ParseExact("2015-11-24", "yyyy-QQ-dd", null);

// Single-character format strings must be one of the standard formats
var date = DateTime.ParseExact("2015-11-24", "q", null);

// Format strings must match the input exactly* (see next section)
var date = DateTime.ParseExact("2015-11-24", "d", null); // Expects 11/24/2015 or 24/11/2015
for most cultures
```

なのフォ―マットの

```
var date = DateTime.ParseExact("2015-11-24T10:15:45",
  new [] { "s", "t", "u", "yyyy-MM-dd" }, // Will succeed as long as input matches one of
these
  CultureInfo.CurrentCulture, DateTimeStyles.None);
```

のいをう

```
var dateString = "10/11/2015";
var date = DateTime.ParseExact(dateString, "d", new CultureInfo("en-US"));
Console.WriteLine("Day: {0}; Month: {1}", date.Day, date.Month);
```

11;10

```
date = DateTime.ParseExact(dateString, "d", new CultureInfo("en-GB"));
Console.WriteLine("Day: {0}; Month: {1}", date.Day, date.Month);
```

10;11

TryParse

このメソッドはをとしてけり、それを`DateTime`にし、またはをすブ―ルをします。びしがすると、 `out`パラメ―タ―としてされたに、されたがりま`out`ます。

にすると、 `out`パラメ―タとしてされたはデフォルトの`DateTime.MinValue`されます。

TryParsestring、out DateTime

```
DateTime parsedValue;

if (DateTime.TryParse("monkey", out parsedValue))
```

```
{
   Console.WriteLine("Apparently, 'monkey' is a date/time value. Who knew?");
}
```

このメソッドは、システムのとISO 8601やそののななどののにづいてをしようとします。

```
DateTime.TryParse("11/24/2015 14:28:42", out parsedValue); // true
DateTime.TryParse("2015-11-24 14:28:42", out parsedValue); // true
DateTime.TryParse("2015-11-24T14:28:42", out parsedValue); // true
DateTime.TryParse("Sat, 24 Nov 2015 14:28:42", out parsedValue); // true
```

このメソッドはカルチャをけれないため、システムロケ―ルをします。これによりしないにつながるがあります。

```
// System set to en-US culture
bool result = DateTime.TryParse("24/11/2015", out parsedValue);
Console.WriteLine(result);
```

```
// System set to en-GB culture
bool result = DateTime.TryParse("11/24/2015", out parsedValue);
Console.WriteLine(result);
```

```
// System set to en-GB culture
bool result = DateTime.TryParse("10/11/2015", out parsedValue);
Console.WriteLine(result);
```

あなたがにいる、は1011ではなく1110であることにくかもしれません。

TryParse、IFormatProvider、DateTimeStyles、out DateTime

```
if (DateTime.TryParse(" monkey ", new CultureInfo("en-GB"),
    DateTimeStyles.AllowLeadingWhite | DateTimeStyles.AllowTrailingWhite, out parsedValue)
{
    Console.WriteLine("Apparently, ' monkey ' is a date/time value. Who knew?");
}
```

メソッドとはなり、このオ―バ―ロ―ドにより、のカルチャとスタイルをすることができます。`IFormatProvider`パラメ―タ―の`null`をすと、システムカルチャがされます。

このメソッドでは、のでをスロ―することがであることにしてください。これらは、`IFormatProvider`と`DateTimeStyles`このオ―バ―ロ―ドにされたパラメ―タにしています。

- `NotSupportedException IFormatProvider`はニュ―トラルカルチャをします。
- `ArgumentException DateTimeStyles`がなオプションではないか、`AssumeLocal`や`AssumeUniversal`などののないフラグがまれています。

TryParseExact

このメソッドは、`TryParse`と`ParseExact`みわせとしてします。カスタムをすることができ、がした

にをスロ―するのではなく、またはをすブ―ルをします。

TryParseExact、、IFormatProvider、DateTimeStyles、out DateTime

このオ―バ―ロ―ドは、のにしてをしようとします。は、するためにそのとしなければなりません。

```
DateTime.TryParseExact("11242015", "MMddyyyy", null, DateTimeStyles.None, out parsedValue); //
true
```

TryParseExactstring、string []、IFormatProvider、DateTimeStyles、out DateTime

このオ―バ―ロ―ドは、をのフォ―マットにしてしようとします。は、するためになくとも1つのとするがあります。

```
DateTime.TryParseExact("11242015", new [] { "yyyy-MM-dd", "MMddyyyy" }, null,
DateTimeStyles.None, out parsedValue); // true
```

オンラインでDateTimeのをむ https://riptutorial.com/ja/dot-net/topic/58/datetimeの

10: ForEach

それはまったくしますか

.NETフレームワークのは、クエリにがなく、 `ForEach`メソッドがをきこすということです。わりにプレーンな`foreach`をすれば、コードをよりしやすくテストしやすくなるかもしれません。

Examples

リストのオブジェクトのメソッドをびす

```
public class Customer {
   public void SendEmail()
   {
       // Sending email code here
   }
}

List<Customer> customers = new List<Customer>();

customers.Add(new Customer());
customers.Add(new Customer());

customers.ForEach(c => c.SendEmail());
```

IEnumerableのメソッド

`ForEach()`は`List<T>`クラスでされていますが、 `IQueryable<T>`または`IEnumerable<T>`ではされていません。これらのケースでは2つのがあります。

にリストする

またはクエリがされ、がしいリストにコピーされるか、データベースがびされます。に、このメソッドがにしてびされます。

```
IEnumerable<Customer> customers = new List<Customer>();

customers.ToList().ForEach(c => c.SendEmail());
```

リストがされるので、このメソッドはらかなメモリオーバーヘッドをちます。

メソッド

メソッドをく

```
public static void ForEach<T>(this IEnumerable<T> enumeration, Action<T> action)
{
    foreach(T item in enumeration)
```

```
    {
        action(item);
    }
}
```

つかいます

```
IEnumerable<Customer> customers = new List<Customer>();

customers.ForEach(c => c.SendEmail());
```

フレームワークのLINQメソッドは、 であることをしてされているため、がじません。 `ForEach`メソッドののは、をすることであり、このではのメソッドからしています。わりにな`foreach`ループをすることもできます。

オンラインでForEachをむ https://riptutorial.com/ja/dot-net/topic/2225/foreach

11: HTTPクライアント

、するHTTP / 1.1 RFCはのとおりです。

- 7230メッセージのとルーティング
- 7231セマンティクスと
- 7232きリクエスト
- 7233リクエスト
- 7234キャッシング
- 7235
- 7239されたHTTP
- 7240HTTPのヘッダーをする

のRFCもあります。

- 7236スキームの
- 7237メソッド

そしてなRFC

- 7238Hypertext Transfer Protocolステータスコード308Permanent Redirect

するプロトコル

- 4918WebオーサリングとバージョンWebDAVのHTTP
- 4791WebDAVCalDAVへのの

Examples

System.Net.HttpWebRequestをしてGETレスポンスをとしてみる

```
string requestUri = "http://www.example.com";
string responseData;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(parameters.Uri);
WebResponse response = request.GetResponse();

using (StreamReader responseReader = new StreamReader(response.GetResponseStream()))
{
    responseData = responseReader.ReadToEnd();
}
```

System.Net.WebClientをしてGETレスポンスをとしてみる

```
string requestUri = "http://www.example.com";
string responseData;
```

```
using (var client = new WebClient())
{
    responseData = client.DownloadString(requestUri);
}
```

System.Net.HttpClientをしてGETをとしてみる

`HttpClient`は、 NuGetMicrosoft HTTP Client Librariesからできます。

```
string requestUri = "http://www.example.com";
string responseData;

using (var client = new HttpClient())
{
    using(var response = client.GetAsync(requestUri).Result)
    {
       response.EnsureSuccessStatusCode();
       responseData = response.Content.ReadAsStringAsync().Result;
    }
}
```

System.Net.HttpWebRequestをしてペイロードでPOSTリクエストをする

```
string requestUri = "http://www.example.com";
string requestBodyString = "Request body string.";
string contentType = "text/plain";
string requestMethod = "POST";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUri)
{
  Method = requestMethod,
  ContentType = contentType,
};

byte[] bytes = Encoding.UTF8.GetBytes(requestBodyString);
Stream stream = request.GetRequestStream();
stream.Write(bytes, 0, bytes.Length);
stream.Close();

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
```

System.Net.WebClientをしてペイロードでPOSTリクエストをする

```
string requestUri = "http://www.example.com";
string requestBodyString = "Request body string.";
string contentType = "text/plain";
string requestMethod = "POST";

byte[] responseBody;
byte[] requestBodyBytes = Encoding.UTF8.GetBytes(requestBodyString);

using (var client = new WebClient())
{
    client.Headers[HttpRequestHeader.ContentType] = contentType;
    responseBody = client.UploadData(requestUri, requestMethod, requestBodyBytes);
```

```
}
```

System.Net.HttpClientをしてペイローでPOSTリクエストをする

`HttpClient`は、 NuGetMicrosoft HTTP Client Librariesからできます。

```
string requestUri = "http://www.example.com";
string requestBodyString = "Request body string.";
string contentType = "text/plain";
string requestMethod = "POST";

var request = new HttpRequestMessage
{
    RequestUri = requestUri,
    Method = requestMethod,
};

byte[] requestBodyBytes = Encoding.UTF8.GetBytes(requestBodyString);
request.Content = new ByteArrayContent(requestBodyBytes);

request.Content.Headers.ContentType = new MediaTypeHeaderValue(contentType);

HttpResponseMessage result = client.SendAsync(request).Result;
result.EnsureSuccessStatusCode();
```

System.Net.Http.HttpClientをしたなHTTPダウンローダ

```
using System;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

class HttpGet
{
    private static async Task DownloadAsync(string fromUrl, string toFile)
    {
        using (var fileStream = File.OpenWrite(toFile))
        {
            using (var httpClient = new HttpClient())
            {
                Console.WriteLine("Connecting...");
                using (var networkStream = await httpClient.GetStreamAsync(fromUrl))
                {
                    Console.WriteLine("Downloading...");
                    await networkStream.CopyToAsync(fileStream);
                    await fileStream.FlushAsync();
                }
            }
        }
    }

    static void Main(string[] args)
    {
        try
        {
            Run(args).Wait();
```

```
        }
        catch (Exception ex)
        {
            if (ex is AggregateException)
                ex = ((AggregateException)ex).Flatten().InnerExceptions.First();

            Console.WriteLine("--- Error: " +
                (ex.InnerException?.Message ?? ex.Message));
        }
    }
    static async Task Run(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine("Basic HTTP downloader");
            Console.WriteLine();
            Console.WriteLine("Usage: httpget <url>[<:port>] <file>");
            return;
        }

        await DownloadAsync(fromUrl: args[0], toFile: args[1]);

        Console.WriteLine("Done!");
    }
}
```

オンラインでHTTPクライアントをむ https://riptutorial.com/ja/dot-net/topic/32/httpクライアント

12: HTTPサーバー

Examples

なみりHTTPファイルサーバーHttpListener

ノート

このは、モードでするがあります。

クライアントは1つだけサポートされます。

にするため、ファイルはすべてASCII *Content-Disposition*ヘッダーのファイルとみなされ、ファイルアクセスエラーはされません。

```
using System;
using System.IO;
using System.Net;

class HttpFileServer
{
    private static HttpListenerResponse response;
    private static HttpListener listener;
    private static string baseFilesystemPath;

    static void Main(string[] args)
    {
        if (!HttpListener.IsSupported)
        {
            Console.WriteLine(
                "*** HttpListener requires at least Windows XP SP2 or Windows Server 2003.");
            return;
        }

        if(args.Length < 2)
        {
            Console.WriteLine("Basic read-only HTTP file server");
            Console.WriteLine();
            Console.WriteLine("Usage: httpfileserver <base filesystem path> <port>");
            Console.WriteLine("Request format: http://url:port/path/to/file.ext");
            return;
        }

        baseFilesystemPath = Path.GetFullPath(args[0]);
        var port = int.Parse(args[1]);

        listener = new HttpListener();
        listener.Prefixes.Add("http://*:" + port + "/");
        listener.Start();

        Console.WriteLine("--- Server stated, base path is: " + baseFilesystemPath);
        Console.WriteLine("--- Listening, exit with Ctrl-C");
        try
        {
```

```
            ServerLoop();
        }
        catch(Exception ex)
        {
            Console.WriteLine(ex);
            if(response != null)
            {
                SendErrorResponse(500, "Internal server error");
            }
        }
    }

    static void ServerLoop()
    {
        while(true)
        {
            var context = listener.GetContext();

            var request = context.Request;
            response = context.Response;
            var fileName = request.RawUrl.Substring(1);
            Console.WriteLine(
                "--- Got {0} request for: {1}",
                request.HttpMethod, fileName);

            if (request.HttpMethod.ToUpper() != "GET")
            {
                SendErrorResponse(405, "Method must be GET");
                continue;
            }

            var fullFilePath = Path.Combine(baseFilesystemPath, fileName);
            if(!File.Exists(fullFilePath))
            {
                SendErrorResponse(404, "File not found");
                continue;
            }

            Console.Write("    Sending file...");
            using (var fileStream = File.OpenRead(fullFilePath))
            {
                response.ContentType = "application/octet-stream";
                response.ContentLength64 = (new FileInfo(fullFilePath)).Length;
                response.AddHeader(
                    "Content-Disposition",
                    "Attachment; filename=\"" + Path.GetFileName(fullFilePath) + "\"");
                fileStream.CopyTo(response.OutputStream);
            }

            response.OutputStream.Close();
            response = null;
            Console.WriteLine(" Ok!");
        }
    }

    static void SendErrorResponse(int statusCode, string statusResponse)
    {
        response.ContentLength64 = 0;
        response.StatusCode = statusCode;
        response.StatusDescription = statusResponse;
        response.OutputStream.Close();
```

```
        Console.WriteLine("*** Sent error: {0} {1}", statusCode, statusResponse);
    }
}
```

なみりHTTPファイルサーバーASP.NETコア

1 - のフォルダをすると、のでされたファイルがされます。

2 - ののproject.jsonというのファイルをしますにじてポートとrootDirectoryをします。

```
{
  "dependencies": {
    "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final",
    "Microsoft.AspNet.StaticFiles": "1.0.0-rc1-final"
  },

  "commands": {
    "web": "Microsoft.AspNet.Server.Kestrel --server.urls http://localhost:60000"
  },

  "frameworks": {
    "dnxcore50": { }
  },

  "fileServer": {
    "rootDirectory": "c:\\users\\username\\Documents"
  }
}
```

3 - のコードをStartup.csしてStartup.csというのファイルをします。

```
using System;
using Microsoft.AspNet.Builder;
using Microsoft.AspNet.FileProviders;
using Microsoft.AspNet.Hosting;
using Microsoft.AspNet.StaticFiles;
using Microsoft.Extensions.Configuration;

public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        var builder = new ConfigurationBuilder();
        builder.AddJsonFile("project.json");
        var config = builder.Build();
        var rootDirectory = config["fileServer:rootDirectory"];
        Console.WriteLine("File server root directory: " + rootDirectory);

        var fileProvider = new PhysicalFileProvider(rootDirectory);

        var options = new StaticFileOptions();
        options.ServeUnknownFileTypes = true;
        options.FileProvider = fileProvider;
        options.OnPrepareResponse = context =>
        {
            context.Context.Response.ContentType = "application/octet-stream";
            context.Context.Response.Headers.Add(
```

```
                "Content-Disposition",
                $"Attachment; filename=\"{context.File.Name}\"");
        };

        app.UseStaticFiles(options);
    }
}
```

4 - コマンドプロンプトをき、フォルダにしてのコマンドをします。

```
dnvm use 1.0.0-rc1-final -r coreclr -p
dnu restore
```

これらのコマンドはだけするがあります。 `dnvm list`をして、のインストールみバージョンのコアCLRののをします。

5 - `dnx web`サーバをします。 ファイルは`http://localhost:60000/path/to/file.ext`でできるようになり`http://localhost:60000/path/to/file.ext`。

にするため、ファイルはすべてASCIIContent-Dispositionヘッダーのファイルとみなされ、ファイルアクセスエラーはされません。

オンラインでHTTPサーバーをむ https://riptutorial.com/ja/dot-net/topic/53/httpサーバー

13: JITコンパイラ

き

JITコンパイル、またはジャストインタイムコンパイルは、コードのやコンパイルののです。 JITコンパイルは.NETフレームワークでされます。 CLRコードC、F、Visual Basicなどは、まずInterpreted LanguageILとばれるものにコンパイルされます。これはマシンコードにいレベルのコードですが、プラットフォームではありません。むしろ、に、このコードはシステムのマシンコードにコンパイルされます。

なぜJITコンパイルをするのですか

- よりいCLRはILにして1つのコンパイラしかとせず、このILはマシンコードにできるのプラットフォームでできます。
- スピードJITコンパイルでは、にコンパイルされたコードのとのコンパイルするになのためにされるコードをできる

JITコンパイルのについては、Wikipediaページをしてください。https : //en.wikipedia.org/wiki/Just-in-time_compilation

Examples

ILコンパイルサンプル

シンプルなHello Worldアプリケーション

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

のILコードJITコンパイルされる

```
//  Microsoft (R) .NET Framework IL Disassembler.  Version 4.6.1055.0
//  Copyright (c) Microsoft Corporation.  All rights reserved.



// Metadata version: v4.0.30319
```

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .z\V.4..
  .ver 4:0:0:0
}
.assembly HelloWorld
{
  .custom instance void
[mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor(int32) = ( 01
00 08 00 00 00 00 00 )
  .custom instance void
[mscorlib]System.Runtime.CompilerServices.RuntimeCompatibilityAttribute::.ctor() = ( 01 00 01
00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....T..WrapNonEx

63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01 )       // ceptionThrows.

  // --- The following custom attribute is added automatically, do not uncomment -------
  //  .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(valuetype
[mscorlib]System.Diagnostics.DebuggableAttribute/DebuggingModes) = ( 01 00 07 01 00 00 00 00 )


  .custom instance void [mscorlib]System.Reflection.AssemblyTitleAttribute::.ctor(string) = (
01 00 0A 48 65 6C 6C 6F 57 6F 72 6C 64 00 00 )    // ...HelloWorld..
  .custom instance void
[mscorlib]System.Reflection.AssemblyDescriptionAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void
[mscorlib]System.Reflection.AssemblyConfigurationAttribute::.ctor(string) = ( 01 00 00 00 00 )

  .custom instance void [mscorlib]System.Reflection.AssemblyCompanyAttribute::.ctor(string) =
( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyProductAttribute::.ctor(string) =
( 01 00 0A 48 65 6C 6C 6F 57 6F 72 6C 64 00 00 )    // ...HelloWorld..
  .custom instance void [mscorlib]System.Reflection.AssemblyCopyrightAttribute::.ctor(string)
= ( 01 00 12 43 6F 70 79 72 69 67 68 74 20 C2 A9 20   // ...Copyright ..

20 32 30 31 37 00 00 )                            //  2017..
  .custom instance void [mscorlib]System.Reflection.AssemblyTrademarkAttribute::.ctor(string)
= ( 01 00 00 00 00 )
  .custom instance void
[mscorlib]System.Runtime.InteropServices.ComVisibleAttribute::.ctor(bool) = ( 01 00 00 00 00 )

  .custom instance void [mscorlib]System.Runtime.InteropServices.GuidAttribute::.ctor(string)
= ( 01 00 24 33 30 38 62 33 64 38 36 2D 34 31 37 32   // ..$308b3d86-4172

2D 34 30 32 32 2D 61 66 63 63 2D 33 66 38 65 33   // -4022-afcc-3f8e3

32 33 33 63 35 62 30 00 00 )                      // 233c5b0..
  .custom instance void
[mscorlib]System.Reflection.AssemblyFileVersionAttribute::.ctor(string) = ( 01 00 07 31 2E 30
2E 30 2E 30 00 00 )             // ...1.0.0.0..
  .custom instance void
[mscorlib]System.Runtime.Versioning.TargetFrameworkAttribute::.ctor(string) = ( 01 00 1C 2E 4E
45 54 46 72 61 6D 65 77 6F 72 6B   // ....NETFramework

2C 56 65 72 73 69 6F 6E 3D 76 34 2E 35 2E 32 01   // ,Version=v4.5.2.

00 54 0E 14 46 72 61 6D 65 77 6F 72 6B 44 69 73   // .T..FrameworkDis

70 6C 61 79 4E 61 6D 65 14 2E 4E 45 54 20 46 72   // playName..NET Fr

61 6D 65 77 6F 72 6B 20 34 2E 35 2E 32 )          // amework 4.5.2
```

```
  .hash algorithm 0x00008004
  .ver 1:0:0:0
}
.module HelloWorld.exe
// MVID: {2A7E1D59-1272-4B47-85F6-D7E1ED057831}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00020003    //  ILONLY 32BITPREFERRED
// Image base: 0x0000021C70230000


// =============== CLASS MEMBERS DECLARATION ===================

.class private auto ansi beforefieldinit HelloWorld.Program
       extends [mscorlib]System.Object
{
  .method private hidebysig static void  Main(string[] args) cil managed
  {
    .entrypoint
    // Code size       13 (0xd)
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr      "Hello World"
    IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b:  nop
    IL_000c:  ret
  } // end of method Program::Main

  .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    // Code size       8 (0x8)
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  nop
    IL_0007:  ret
  } // end of method Program::.ctor

} // end of class HelloWorld.Program
```

MS ILDASMツールILアセンブラで

オンラインでJITコンパイラをむ https://riptutorial.com/ja/dot-net/topic/9222/jitコンパイラ

14: JSON with .NET with Newtonsoft.Json

き

NuGetパッケージ`Newtonsoft.Json`は、.NETでJSONのテキストとオブジェクトをしてするためのとなっています。これは、でいやすいなツールです。

Examples

オブジェクトを**JSON**にシリアルする

```
using Newtonsoft.Json;

var obj = new Person
{
    Name = "Joe Smith",
    Age = 21
};
var serializedJson = JsonConvert.SerializeObject(obj);
```

このJSON `{"Name":"Joe Smith","Age":21}`

JSONテキストからオブジェクトをシリアルする

```
var json = "{\"Name\":\"Joe Smith\",\"Age\":21}";
var person = JsonConvert.DeserializeObject<Person>(json);
```

これにより、が "Joe Smith"およびAge 21の`Person`オブジェクトがされます。

オンラインでJSON with .NET with Newtonsoft.Jsonをむ https://riptutorial.com/ja/dot-net/topic/8746/json-with--net-with-newtonsoft-json

15: JSONシリアル

JavaScriptSerializerとJson.NET

JavaScriptSerializerクラスは.NET 3.5でされ、AJAXアプリケーションの.NETのレイヤーでにされています。これは、マネージコードでJSONをするためにできます。

JavaScriptSerializerクラスがするにもかかわらず、シリアルとシリアルにはオープンソースのJson.NETライブラリをすることをおめします。 Json.NETはよりいパフォーマンスとカスタムクラスへのマッピングJSONカスタムのためののインターフェースをしていますJavaScriptConverterオブジェクトは 、じするためにとされるであろうJavaScriptSerializer。

Examples

System.Web.Script.Serialization.JavaScriptSerializerをしたシリアル

JavaScriptSerializer.Deserialize<T>(input)メソッドは、 JavaScriptSerializerによってネイティブにサポートされているデフォルトのマッピングをして、なJSONのをされた<T>オブジェクトにシリアルしようとしJavaScriptSerializer。

```
using System.Collections;
using System.Web.Script.Serialization;

// ...

string rawJSON = "{\"Name\":\"Fibonacci Sequence\",\"Numbers\":[0, 1, 1, 2, 3, 5, 8, 13]}";

JavaScriptSerializer JSS = new JavaScriptSerializer();
Dictionary<string, object> parsedObj = JSS.Deserialize<Dictionary<string, object>>(rawJSON);

string name = parsedObj["Name"].toString();
ArrayList numbers = (ArrayList)parsedObj["Numbers"]
```

JavaScriptSerializerオブジェクトは.NETバージョン3.5でされました。

Json.NETをしたシリアル

```
internal class Sequence{
    public string Name;
    public List<int> Numbers;
}

// ...

string rawJSON = "{\"Name\":\"Fibonacci Sequence\",\"Numbers\":[0, 1, 1, 2, 3, 5, 8, 13]}";

Sequence sequence = JsonConvert.DeserializeObject<Sequence>(rawJSON);
```

については、 Json.NETサイトをしてください。

Json.NETは.NETバージョン2をサポートしています。

Json.NETをったシリアライゼーション

```
[JsonObject("person")]
public class Person
{
    [JsonProperty("name")]
    public string PersonName { get; set; }
    [JsonProperty("age")]
    public int PersonAge { get; set; }
    [JsonIgnore]
    public string Address { get; set; }
}

Person person = new Person { PersonName = "Andrius", PersonAge = 99, Address = "Some address"
};
string rawJson = JsonConvert.SerializeObject(person);

Console.WriteLine(rawJson); // {"name":"Andrius","age":99}
```

プロパティおよびクラスをでして、のjsonのをしたり、jsonJsonIgnoreからそれらをするにしてください。

Json.NETのシリアルのは、 こちらをしてください 。

Cでは、は*PascalCase*でにされています。 JSONでは、すべてのに*camelCase*をします。リゾルバをして、2つのでることができます。

```
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    [JsonIgnore]
    public string Address { get; set; }
}

public void ToJson() {
    Person person = new Person { Name = "Andrius", Age = 99, Address = "Some address" };
    var resolver = new CamelCasePropertyNamesContractResolver();
    var settings = new JsonSerializerSettings { ContractResolver = resolver };
    string json = JsonConvert.SerializeObject(person, settings);

    Console.WriteLine(json); // {"name":"Andrius","age":99}
}
```

シリアライゼーション - Newtonsoft.Jsonをしたシリアル

のヘルパーとはなり、このクラスはクラスヘルパーをしてシリアライズおよびデシリアライズし

ます。したがって、するヘルパーよりもしです。

```
using Newtonsoft.Json;

var rawJSON      = "{\"Name\":\"Fibonacci Sequence\",\"Numbers\":[0, 1, 1, 2, 3, 5, 8, 13]}";
var fibo         = JsonConvert.DeserializeObject<Dictionary<string, object>>(rawJSON);
var rawJSON2     = JsonConvert.SerializeObject(fibo);
```

バインディング

NewtonsoftのJson.NETでは、をにするなく、にExpandoObject / Dynamicオブジェクトをしてjsonをにバインドできます。

シリアライゼーション

```
dynamic jsonObject = new ExpandoObject();
jsonObject.Title   = "Merchent of Venice";
jsonObject.Author  = "William Shakespeare";
Console.WriteLine(JsonConvert.SerializeObject(jsonObject));
```

デシリアライゼーション

```
var rawJson = "{\"Name\":\"Fibonacci Sequence\",\"Numbers\":[0, 1, 1, 2, 3, 5, 8, 13]}";
dynamic parsedJson = JObject.Parse(rawJson);
Console.WriteLine("Name: " + parsedJson.Name);
Console.WriteLine("Name: " + parsedJson.Numbers.Length);
```

rawJsonオブジェクトのキーがオブジェクトのメンバーにされていることにしてください。

これは、アプリケーションが々なのJSONをけれ/できるにです。ただし、Jsonストリングまたはシリアライゼーション/デシリアライゼーションのとしてされたオブジェクトにして、レベルのをすることをおめします。

JsonSerializerSettingsでのJson.NETをしたシリアル

このシリアライザには、デフォルトの.net jsonシリアライザにはないNullのようなれたがありますJsonSerializerSettingsをするだけです。

```
public static string Serialize(T obj)
{
   string result = JsonConvert.SerializeObject(obj, new JsonSerializerSettings {
NullValueHandling = NullValueHandling.Ignore});
   return result;
}
```

.netののなシリアライザのは、ループです。コースにされているの、インスタンスにはコースプロパティがあり、コースにはループをするList<Student>をするのコレクションがあります。これはJsonSerializerSettingsできます

```
public static string Serialize(T obj)
{
   string result = JsonConvert.SerializeObject(obj, new JsonSerializerSettings {
ReferenceLoopHandling = ReferenceLoopHandling.Ignore});
   return result;
}
```

のようなさまざまなシリアライズオプションをできます

```
public static string Serialize(T obj)
{
   string result = JsonConvert.SerializeObject(obj, new JsonSerializerSettings {
NullValueHandling = NullValueHandling.Ignore, ReferenceLoopHandling =
ReferenceLoopHandling.Ignore});
   return result;
}
```

オンラインでJSONシリアルをむ https://riptutorial.com/ja/dot-net/topic/183/jsonシリアル

16: LINQ

き

LINQLanguage Integrated Queryは、データソースからデータをするです。 LINQは、さまざまなのデータソースおよびにわたるデータをするためのしたモデルをすることで、このをします。 LINQクエリでは、にオブジェクトでしています。じコーディングパターンをして、XML、SQLデータベース、ADO.NETデータセット、.NETコレクション、およびプロバイダーがなそののののデータをおよびします。 LINQはCとVBでできます。

- public static TSource<TSource>このIEnumerable <TSource>ソース、Func <TSource、TSource、TSource> func
- public static TAccumulate Aggregate <TSource、TAccumulate>このIEnumerable <TSource>ソース、TAccumulateシード、Func <TAccumulate、TSource、TAccumulate>
- public static TResult Aggregate <TSource、TAccumulate、TResult>このIEnumerable <TSource>ソース、TAccumulateシード、Func <TAccumulate、TSource、TAccumulate> func、Func <TAccumulate、TResult> resultSelector
- public static Booleanすべての<TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static Booleanの<TSource>このIEnumerable <TSource>ソース
- public static Booleanの<TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static IEnumerable <TSource> AsEnumerable <TSource>このIEnumerable <TSource>ソース
- public static Decimal AverageこのIEnumerable <Decimal>ソース
- public static Double AverageこのIEnumerable <Double>ソース
- public static Double AverageこのIEnumerable <Int32>ソース
- public static Double AverageこのIEnumerable <Int64>ソース
- public static Nullable <Decimal> AverageこのIEnumerable <Nullable <Decimal >>ソース
- public static Nullable <Double> AverageこのIEnumerable <Nullable <Double >>ソース
- public static Nullable <Double> AverageこのIEnumerable <Nullable <Int32 >>ソース
- public static Nullable <Double> AverageこのIEnumerable <Nullable <Int64 >>ソース
- public static Nullable <Single> AverageこのIEnumerable <Nullable <Single >>ソース
- public static Single AverageこのIEnumerable <Single>ソース
- public static Decimal Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Decimal>セレクタ
- public static Double <TSource>このIEnumerable <TSource>ソース、Func <TSource、Double>セレクタ
- public static Double Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32>セレクタ
- public static Double Average <TSource>このIEnumerable <TSource>ソース、Func

<TSource、Int64>セレクタ

- public static Nullable <Decimal> Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Decimal >>セレクタ
- パブリックstatic Nullable <Double> Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Double >>セレクタ
- public static Nullable <Double> Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int32 >>セレクタ
- public static Nullable <Double> Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int64 >>セレクタ
- public static Nullable <Single> Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Single >>セレクタ
- public static Single Average <TSource>このIEnumerable <TSource>ソース、Func <TSource、Single>セレクタ
- public static IEnumerable <TResult>キャスト<TResult>このIEnumerableソース
- public static IEnumerable <TSource> Concat <TSource>このIEnumerable <TSource>が、IEnumerable <TSource>がsecond
- public static Boolean <TSource>このIEnumerable <TSource>ソース、TSource
- public static Boolean <TSource>このIEnumerable <TSource>ソース、TSource、IEqualityComparer <TSource>コンペアラ
- public static Int32 Count <TSource>このIEnumerable <TSource>ソース
- public static Int32 Count <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static IEnumerable <TSource> DefaultIfEmpty <TSource>このIEnumerable <TSource>ソース
- public static IEnumerable <TSource> DefaultIfEmpty <TSource>このIEnumerable <TSource>ソース、TSource defaultValue
- public static IEnumerable <TSource> Distinct <TSource>このIEnumerable <TSource>ソース
- public static IEnumerable <TSource> Distinct <TSource>このIEnumerable <TSource>ソース、IEqualityComparer <TSource> comparer
- public static TSource ElementAt <TSource>このIEnumerable <TSource>ソース、Int32インデックス
- public static TSource ElementAtOrDefault <TSource>このIEnumerable <TSource>ソース、Int32インデックス
- public static IEnumerable <TResult> Empty <TResult>
- public static IEnumerable <TSource> <TSource>をくこのIEnumerable <TSource>が、IEnumerable <TSource>がsecond
- public static IEnumerable <TSource> <TSource>をくこのIEnumerable <TSource>、IEnumerable <TSource> second、IEqualityComparer <TSource> comparer
- public static TSourceの<TSource>このIEnumerable <TSource>ソース
- public static TSourceの<TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static TSource FirstOrDefault <TSource>このIEnumerable <TSource>ソース
- public static TSource FirstOrDefault <TSource>このIEnumerable <TSource>ソース、Func

<TSource、Boolean>

- public IEnumerable <IGrouping <TKey、TSource >> GroupBy <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IEqualityComparer <TKey> comparerは、IEnumerable <IGKeying <TKey、TSource >> GroupBy <TSource、TKey>
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelectorこのメソッドは、IEnumerable <IGKeying <TKey、TElement>、GroupBy <TSource、TKey、TElement>
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector、IEqualityComparer <TKey> comparerパブリックstatic IEnumerable <IGrouping <TKey、TElement >> GroupBy <TSource、TKey、TElement>
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TKey、IEnumerable <TSource>、TResult> resultSelectorをして、IEnumerable <TResult>
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TKey、IEnumerable <TSource>、TResult> resultSelector、IEqualityComparer <TKey>コンストラクタ
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector、Func <TKey、IEnumerable <TElement>、TResult > resultSelector
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector、Func <TKey、IEnumerable <TElement>、TResult > resultSelector、IEqualityComparer <TKey> comparer
- IEnumerable <TOuter>、IEnumerable <TInner>、Func <TOuter、TKey>キーセレクタ、Func <TInner、TKey>キーセレクタ、Func <TOuter、 IEnumerable <TInner>、TResult> resultSelector
- IEnumerable <TOuter>、IEnumerable <TInner>、Func <TOuter、TKey>キーセレクタ、Func <TInner、TKey>キーセレクタ、Func <TOuter、 IEnumerable <TInner>、TResult> resultSelector、IEqualityComparer <TKey> comparer
- public static IEnumerable <TSource> Intersect <TSource>このIEnumerable <TSource>が、IEnumerable <TSource>がsecond
- public static IEnumerable <TSource> Intersect <TSource>このIEnumerable <TSource>が、IEnumerable <TSource>がsecond、IEqualityComparer <TSource>が
- IEnumerable <TOuter>、IEnumerable <TInner>、Func <TOuter、TKey> outerKeySelector、Func <TInner、TKey> innerKeySelector、Func <TOuter、TKey、TResult> TInner、TResult> resultSelector
- IEnumerable <TOuter>、IEnumerable <TInner>、Func <TOuter、TKey> outerKeySelector、Func <TInner、TKey> innerKeySelector、Func <TOuter、TKey、TResult> TInner、TResult> resultSelector、IEqualityComparer <TKey> comparer
- public static TSource Last <TSource>このIEnumerable <TSource>ソース
- public static TSource Last <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>

- public static TSource LastOrDefault <TSource>このIEnumerable <TSource>ソース
- public static TSource LastOrDefault <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static Int64 LongCount <TSource>このIEnumerable <TSource>ソース
- public static Int64 LongCount <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static Decimal MaxこのIEnumerable <Decimal>ソース
- public static Double MaxこのIEnumerable <Double>ソース
- public static Int32 MaxこのIEnumerable <Int32>ソース
- public static Int64 MaxこのIEnumerable <Int64>ソース
- public static Nullable <Decimal> MaxこのIEnumerable <Nullable <Decimal >>ソース
- public static Nullable <Double> MaxこのIEnumerable <Nullable <Double >>ソース
- public static Nullable <Int32> MaxこのIEnumerable <Nullable <Int32 >>ソース
- public static Nullable <Int64> MaxこのIEnumerable <Nullable <Int64 >>ソース
- public static Nullable <Single> MaxこのIEnumerable <Nullable <Single >>ソース
- public static Single MaxこのIEnumerable <Single>ソース
- public static TSource Max <TSource>このIEnumerable <TSource>ソース
- public static Decimal Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Decimal>セレクタ
- public static Double Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Double>セレクタ
- public static Int32 Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32>セレクタ
- public static Int64 Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int64>セレクタ
- パブリックstatic Nullable <Decimal> Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Decimal >>セレクタ
- public static Nullable <Double> Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Double >>セレクタ
- public static Nullable <Int32> Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int32 >>セレクタ
- public static Nullable <Int64> Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int64 >>セレクタ
- public static Nullable <Single> Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Single >>セレクタ
- public static Single Max <TSource>このIEnumerable <TSource>ソース、Func <TSource、Single>セレクタ
- public static TResult Max <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、TResult>セレクタ
- public static Decimal MinこのIEnumerable <Decimal>ソース
- public static Double MinこのIEnumerable <Double>ソース
- public static Int32 MinこのIEnumerable <Int32>ソース

- public static Int64 MinこのIEnumerable <Int64>ソース
- public static Nullable <Decimal> MinこのIEnumerable <Nullable <Decimal >>ソース
- public static Nullable <Double> MinこのIEnumerable <Nullable <Double >>ソース
- public static Nullable <Int32> MinこのIEnumerable <Nullable <Int32 >>ソース
- public static Nullable <Int64> MinこのIEnumerable <Nullable <Int64 >>ソース
- public static Nullable <Single> MinこのIEnumerable <Nullable <Single >>ソース
- public static Single MinこのIEnumerable <Single>ソース
- public static TSource Min <TSource>このIEnumerable <TSource>ソース
- public static Decimal Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Decimal>セレクタ
- public static Double Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Double>セレクタ
- public static Int32 Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32>セレクタ
- public static Int64 Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int64>セレクタ
- public static Nullable <Decimal> Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Decimal >>セレクタ
- public static Nullable <Double> Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Double >>セレクタ
- public static Nullable <Int32> Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int32 >>セレクタ
- public static Nullable <Int64> Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int64 >>セレクタ
- public static Nullable <Single> Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Single >>セレクタ
- public static Single Min <TSource>このIEnumerable <TSource>ソース、Func <TSource、Single>セレクタ
- public static TResult Min <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、TResult>セレクタ
- public static IEnumerable <TResult> OfType <TResult>このIEnumerableソース
- パブリックIOrderedEnumerable <TSource> OrderBy <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- パブリックIOrderedEnumerable <TSource> OrderBy <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IComparer <TKey> comparer
- パブリックIOrderedEnumerable <TSource> OrderByDescending <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- パブリックIOrderedEnumerable <TSource> OrderByDescending <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IComparer <TKey> comparer
- パブリックIEnumerable <Int32>Int32、Int32カウント
- public static IEnumerable <TResult>りし<TResult>TResult、Int32カウント

- public static IEnumerable <TSource> Reverse <TSource>このIEnumerable <TSource>ソース
- public static IEnumerable <TResult> <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、TResult>セレクタをします。
- public static IEnumerable <TResult> <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、Int32、TResult>セレクタをします。
- public static IEnumerable <TResult> SelectMany <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、IEnumerable <TResult >>セレクタ
- public IEnumerable <TResult> SelectMany <TSource、TResult>このIEnumerable <TSource>ソース、Func <TSource、Int32、IEnumerable <TResult >>セレクタ
- public IEnumerable <TResult> SelectMany <TSource、TCollection、TResult>このIEnumerable <TSource>ソース、Func <TSource、IEnumerable <TCollection >> collectionSelector、Func <TSource、TCollection、TResult> resultSelector
- このIEnumerable <TSource>ソース、Func <TSource、Int32、IEnumerable <TCollection >> collectionSelector、Func <TSource、TCollection、TResult> resultSelectorパブリックstatic IEnumerable <TResult> SelectMany <TSource、TCollection、TResult>
- public static Boolean SequenceEqual <TSource>このIEnumerable <TSource>が、IEnumerable <TSource>がsecond
- public static Boolean SequenceEqual <TSource>このIEnumerable <TSource> first、IEnumerable <TSource> second、IEqualityComparer <TSource> comparer
- public static TSource Single <TSource>このIEnumerable <TSource>ソース
- public static TSource Single <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static TSource SingleOrDefault <TSource>このIEnumerable <TSource>ソース
- public static TSource SingleOrDefault <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static IEnumerable <TSource> Skip <TSource>このIEnumerable <TSource>ソース、Int32カウント
- public static IEnumerable <TSource> SkipWhile <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static IEnumerable <TSource> SkipWhile <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32、Boolean>
- public static Decimal SumこのIEnumerable <Decimal>ソース
- public static Double SumこのIEnumerable <Double>ソース
- public static Int32 SumこのIEnumerable <Int32>ソース
- public static Int64 SumこのIEnumerable <Int64>ソース
- public static Nullable <Decimal> SumこのIEnumerable <Nullable <Decimal >>ソース
- public static Nullable <Double> SumこのIEnumerable <Nullable <Double >>ソース
- public static Nullable <Int32> SumこのIEnumerable <Nullable <Int32 >>ソース
- public static Nullable <Int64> SumこのIEnumerable <Nullable <Int64 >>ソース
- public static Nullable <Single> SumこのIEnumerable <Nullable <Single >>ソース
- public static Single SumこのIEnumerable <Single>ソース

- public static Decimal Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Decimal>セレクタ
- public static Double Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Double>セレクタ
- public static Int32 Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32>セレクタ
- public static Int64 Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int64>セレクタ
- パブリックstatic Nullable <Decimal> Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Decimal >>セレクタ
- パブリックstatic Nullable <Double> Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Double >>セレクタ
- public static Nullable <Int32> Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int32 >>セレクタ
- public static Nullable <Int64> Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Int64 >>セレクタ
- パブリックstatic Nullable <Single> Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Nullable <Single >>セレクタ
- public static Single Sum <TSource>このIEnumerable <TSource>ソース、Func <TSource、Single>セレクタ
- public static IEnumerable <TSource> Take <TSource>このIEnumerable <TSource>ソース、Int32カウント
- public static IEnumerable <TSource> TakeWhile <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- パブリックIEnumerable <TSource> TakeWhile <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32、Boolean>
- public static IOrderedEnumerable <TSource> ThenBy <TSource、TKey>このIOrderedEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- public static IOrderedEnumerable <TSource> ThenBy <TSource、TKey>このIOrderedEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IComparer <TKey> comparer
- public static IOrderedEnumerable <TSource> ThenByDescending <TSource、TKey>このIOrderedEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- public static IOrderedEnumerable <TSource> ThenByDescending <TSource、TKey>このIOrderedEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IComparer <TKey> comparer
- public static TSource [] ToArray <TSource>このIEnumerable <TSource>ソース
- public static Dictionary <TKey、TSource> ToDictionary <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IEqualityComparer <TKey>コンストラクタ
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelectorパブリックstatic Dictionary <TKey、TElement>

ToDictionary <TSource、TKey、TElement>
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector、IEqualityComparer <TKey> comparer
- public static List <TSource> ToList <TSource>このIEnumerable <TSource>ソース
- public static ILookup <TKey、TSource> ToLookup <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector
- public static ILookup <TKey、TSource> ToLookup <TSource、TKey>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、IEqualityComparer <TKey> comparer
- パブリックstatic ILookup <TKey、TElement> ToLookup <TSource、TKey、TElement>このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector
- このIEnumerable <TSource>ソース、Func <TSource、TKey> keySelector、Func <TSource、TElement> elementSelector、IEqualityComparer <TKey> comparerパブリックstatic ILookup <TKey、TElement> ToLookup <TSource、TKey、TElement>
- public static IEnumerable <TSource> Union <TSource>このIEnumerable <TSource>が、IEnumerable <TSource>がsecond
- public static IEnumerable <TSource> Union <TSource>このIEnumerable <TSource> first、IEnumerable <TSource> second、IEqualityComparer <TSource> comparer
- public static IEnumerable <TSource> <TSource>このIEnumerable <TSource>ソース、Func <TSource、Boolean>
- public static IEnumerable <TSource> <TSource>このIEnumerable <TSource>ソース、Func <TSource、Int32、Boolean>
- public static IEnumerable <TResult> Zip <TFirst、TSecond、TResult>このIEnumerable <TFirst> first、IEnumerable <TSecond> second、Func <TFirst、TSecond、TResult> resultSelector

- LINQもしてください。

LINQビルトインメソッドは、 `System.Core`アセンブリの`System.Linq.Enumerable`クラスにする`IEnumerable<T>`インターフェイスのメソッドです。これらは.NET Framework 3.5でできます。

LINQでは、クエリのようなやをして、さまざまな`IEnumerable`をに、、およびみわせることができます。

のLINQメソッドは、なや`List<T>`をむの`IEnumerable<T>`でることができますが、LINQのセットをSQLにできるデータベースオブジェクトでもできます。データオブジェクトはそれをサポートします。 LINQ to SQLをしてください。

オブジェクト `Contains`や`Except`などをするメソッドでは、コレクションのTがそのインタフェースをする、 `IEquatable<T>.Equals`がされます。それのは、ののEqualsおよび`GetHashCode` デフォルトの`Object`からオーバーライドされるがありますがされます。カスタム`IEqualityComparer<T>`をできるこれらのメソッドのオーバーロードもあります。

`...OrDefault`

メソッドの、 `default(T)`がデフォルトをするためにされます。

のリファレンス Enumerableクラス

レ イジ―

`IEnumerable<T>`をすすべてのクエリは、ちにされません。わりに、クエリがされるまでロジックがします。 1つのは、かがこれらのクエリ―の1つ、たとえば`.Where()`からした`IEnumerable<T>`するたびに、なクエリ―ロジックがりされることです。がされている、これはパフォ―マンスののになります。

1つのなのシ―ケンスのおおよそのサイズをっているか、またはできるは、 `.ToArray()`または`.ToList()`をしてをにバッファリングすることです。 `.ToDictionary()`または`.ToLookup()`はじをたすことができます。もちろん、シ―ケンスをして、のカスタムロジックにってをバッファすることもできます。

`ToArray()`または`ToList()`

`.ToArray()`と`.ToList()`は、 `IEnumerable<T>`シ―ケンスのすべてのをルプし、そのをメモリにされたコレクションにします。どちらをするかは、のガイドラインをにしてください。

- いくつかのAPIは`T[]`または`List<T>`とするかもしれません。
- `.ToList()`は、よりにし、 `.ToArray()`よりもないゴミをします。は、すべてのをしいサイズのコレクションにコピ―するがあります。
- がまたはからすることができる`List<T>`によってさ`.ToList()`、 `T[]`からさ`.ToArray()`そのをじてサイズのままです。つまり、 `List<T>`はであり、 `T[]`はです。
- `T[]`からさ`.ToArray()`よりないメモリをする`List<T>`からさ`.ToList()`はされようとしているので、もし、む`.ToArray()`。 びし`List<T>.TrimExcess()`は、メモリのいをにアカデミックにしますが、 `.ToList()`なのをします。

Examples

```
var persons = new[]
{
    new {Id = 1, Name = "Foo"},
    new {Id = 2, Name = "Bar"},
    new {Id = 3, Name = "Fizz"},
    new {Id = 4, Name = "Buzz"}
};

var names = persons.Select(p => p.Name);
Console.WriteLine(string.Join(",", names.ToArray()));

//Foo,Bar,Fizz,Buzz
```

このタイプのは、、プログラミングでは`map`とばれ`map`。

どこフィルタ

このメソッドは、ラムダをたすすべてのをむIEnumerableをします。

```
var personNames = new[]
{
    "Foo", "Bar", "Fizz", "Buzz"
};

var namesStartingWithF = personNames.Where(p => p.StartsWith("F"));
Console.WriteLine(string.Join(",", namesStartingWithF));
```

フー、フィッツ

デモをる

OrderBy

```
var persons = new[]
{
    new {Id = 1, Name = "Foo"},
    new {Id = 2, Name = "Bar"},
    new {Id = 3, Name = "Fizz"},
    new {Id = 4, Name = "Buzz"}
};

var personsSortedByName = persons.OrderBy(p => p.Name);

Console.WriteLine(string.Join(",", personsSortedByName.Select(p => p.Id).ToArray()));

//2,4,3,1
```

OrderByDescending

```
var persons = new[]
{
    new {Id = 1, Name = "Foo"},
    new {Id = 2, Name = "Bar"},
    new {Id = 3, Name = "Fizz"},
    new {Id = 4, Name = "Buzz"}
};

var personsSortedByNameDescending = persons.OrderByDescending(p => p.Name);

Console.WriteLine(string.Join(",", personsSortedByNameDescending.Select(p =>
p.Id).ToArray()));

//1,3,4,2
```

```
var numbers = new[] {1,2,3,4,5};
Console.WriteLine(numbers.Contains(3)); //True
Console.WriteLine(numbers.Contains(34)); //False
```

```
var numbers = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var evenNumbersBetweenSixAndFourteen = new[] { 6, 8, 10, 12 };

var result = numbers.Except(evenNumbersBetweenSixAndFourteen);

Console.WriteLine(string.Join(",", result));

//1, 2, 3, 4, 5, 7, 9
```

```
var numbers1to10 = new[] {1,2,3,4,5,6,7,8,9,10};
var numbers5to15 = new[] {5,6,7,8,9,10,11,12,13,14,15};

var numbers5to10 = numbers1to10.Intersect(numbers5to15);

Console.WriteLine(string.Join(",", numbers5to10));

//5,6,7,8,9,10
```

```
var numbers1to5 = new[] {1, 2, 3, 4, 5};
var numbers4to8 = new[] {4, 5, 6, 7, 8};

var numbers1to8 = numbers1to5.Concat(numbers4to8);

Console.WriteLine(string.Join(",", numbers1to8));

//1,2,3,4,5,4,5,6,7,8
```

にがっていることにしてください。これがましくないは、わりに`Union`してください。

つける

```
var numbers = new[] {1,2,3,4,5};

var firstNumber = numbers.First();
Console.WriteLine(firstNumber); //1

var firstEvenNumber = numbers.First(n => (n & 1) == 0);
Console.WriteLine(firstEvenNumber); //2
```

`InvalidOperationException`が "シ―ケンスにするがまれていません"というメッセ―ジがスロ―されます。

```
var firstNegativeNumber = numbers.First(n => n < 0);
```

シ ン グル

```
var oneNumber = new[] {5};
var theOnlyNumber = oneNumber.Single();
Console.WriteLine(theOnlyNumber);  //5

var numbers = new[] {1,2,3,4,5};

var theOnlyNumberSmallerThanTwo = numbers.Single(n => n < 2);
```

```
Console.WriteLine(theOnlyNumberSmallerThanTwo);  //1
```

シーケンスにのがあるため、の`InvalidOperationException`は`InvalidOperationException`スローします。

```
var theOnlyNumberInNumbers = numbers.Single();
var theOnlyNegativeNumber = numbers.Single(n => n < 0);
```

```
var numbers = new[] {1,2,3,4,5};

var lastNumber = numbers.Last();
Console.WriteLine(lastNumber); //5

var lastEvenNumber = numbers.Last(n => (n & 1) == 0);
Console.WriteLine(lastEvenNumber); //4
```

`InvalidOperationException`スローされる

```
var lastNegativeNumber = numbers.Last(n => n < 0);
```

LastOrDefault

```
var numbers = new[] {1,2,3,4,5};

var lastNumber = numbers.LastOrDefault();
Console.WriteLine(lastNumber); //5

var lastEvenNumber = numbers.LastOrDefault(n => (n & 1) == 0);
Console.WriteLine(lastEvenNumber); //4

var lastNegativeNumber = numbers.LastOrDefault(n => n < 0);
Console.WriteLine(lastNegativeNumber); //0

var words = new[] { "one", "two", "three", "four", "five" };

var lastWord = words.LastOrDefault();
Console.WriteLine(lastWord); // five

var lastLongWord = words.LastOrDefault(w => w.Length > 4);
Console.WriteLine(lastLongWord); // three

var lastMissingWord = words.LastOrDefault(w => w.Length > 5);
Console.WriteLine(lastMissingWord); // null
```

SingleOrDefault

```
var oneNumber = new[] {5};
var theOnlyNumber = oneNumber.SingleOrDefault();
Console.WriteLine(theOnlyNumber);  //5

var numbers = new[] {1,2,3,4,5};

var theOnlyNumberSmallerThanTwo = numbers.SingleOrDefault(n => n < 2);
```

```
Console.WriteLine(theOnlyNumberSmallerThanTwo);  //1

var theOnlyNegativeNumber = numbers.SingleOrDefault(n => n < 0);
Console.WriteLine(theOnlyNegativeNumber);  //0
```

`InvalidOperationException`スローされる

```
var theOnlyNumberInNumbers = numbers.SingleOrDefault();
```

FirstOrDefault

```
var numbers = new[] {1,2,3,4,5};

var firstNumber = numbers.FirstOrDefault();
Console.WriteLine(firstNumber); //1

var firstEvenNumber = numbers.FirstOrDefault(n => (n & 1) == 0);
Console.WriteLine(firstEvenNumber); //2

var firstNegativeNumber = numbers.FirstOrDefault(n => n < 0);
Console.WriteLine(firstNegativeNumber); //0

var words = new[] { "one", "two", "three", "four", "five" };

var firstWord = words.FirstOrDefault();
Console.WriteLine(firstWord); // one

var firstLongWord = words.FirstOrDefault(w => w.Length > 3);
Console.WriteLine(firstLongWord); // three

var firstMissingWord = words.FirstOrDefault(w => w.Length > 5);
Console.WriteLine(firstMissingWord); // null
```

どれか

コレクションにラムダのをたすが`true`は`true`します。

```
var numbers = new[] {1,2,3,4,5};

var isNotEmpty = numbers.Any();
Console.WriteLine(isNotEmpty); //True

var anyNumberIsOne = numbers.Any(n => n == 1);
Console.WriteLine(anyNumberIsOne); //True

var anyNumberIsSix = numbers.Any(n => n == 6);
Console.WriteLine(anyNumberIsSix); //False

var anyNumberIsOdd = numbers.Any(n => (n & 1) == 1);
Console.WriteLine(anyNumberIsOdd); //True

var anyNumberIsNegative = numbers.Any(n => n < 0);
Console.WriteLine(anyNumberIsNegative); //False
```

すべて

```
var numbers = new[] {1,2,3,4,5};

var allNumbersAreOdd = numbers.All(n => (n & 1) == 1);
Console.WriteLine(allNumbersAreOdd); //False

var allNumbersArePositive = numbers.All(n => n > 0);
Console.WriteLine(allNumbersArePositive); //True
```

`All`メソッドは、にって`false`とされるのをチェックすることによってすることにしてください。したがって、このメソッドは、セットがのにはどのでも `true`をし`true`。

```
var numbers = new int[0];
var allNumbersArePositive = numbers.All(n => n > 0);
Console.WriteLine(allNumbersArePositive); //True
```

SelectManyフラットマップ

`Enumerable.Select`はすべてののをし`Enumerable.Select`。 `Enumerable.SelectMany`はごとにのをします。これは、シーケンスにシーケンスよりもくのがまれるがあることをします。

`Enumerable.Select`される`Lambda expressions`は、のをすがあります。 `Enumerable.SelectMany`されるラムダは、シーケンスをするがあり`Enumerable.SelectMany`。 このシーケンスには、シーケンスのにしてさまざまなのをめることができます。

```
class Invoice
{
    public int Id { get; set; }
}

class Customer
{
    public Invoice[] Invoices {get;set;}
}

var customers = new[] {
    new Customer {
        Invoices = new[] {
            new Invoice {Id=1},
            new Invoice {Id=2},
        }
    },
    new Customer {
        Invoices = new[] {
            new Invoice {Id=3},
            new Invoice {Id=4},
        }
    },
    new Customer {
        Invoices = new[] {
            new Invoice {Id=5},
            new Invoice {Id=6},
        }
```

```
    }
};

var allInvoicesFromAllCustomers = customers.SelectMany(c => c.Invoices);

Console.WriteLine(
    string.Join(",", allInvoicesFromAllCustomers.Select(i => i.Id).ToArray()));
```

1,2,3,4,5,6

デモをる

`Enumerable.SelectMany`は、2つのした`from`をするベースのクエリでもでき`Enumerable.SelectMany`。

```
var allInvoicesFromAllCustomers
    = from customer in customers
      from invoice in customer.Invoices
      select invoice;
```

```
var numbers = new[] {1,2,3,4};

var sumOfAllNumbers = numbers.Sum();
Console.WriteLine(sumOfAllNumbers); //10

var cities = new[] {
    new {Population = 1000},
    new {Population = 2500},
    new {Population = 4000}
};

var totalPopulation = cities.Sum(c => c.Population);
Console.WriteLine(totalPopulation); //7500
```

スキップ

SkipはのNのアイテムをさずにします。アイテムN + 1にすると、Skipはすべてのアイテムをすようになります。

```
var numbers = new[] {1,2,3,4,5};

var allNumbersExceptFirstTwo = numbers.Skip(2);
Console.WriteLine(string.Join(",", allNumbersExceptFirstTwo.ToArray()));

//3,4,5
```

る

このメソッドは、の$_n$をからります。

```
var numbers = new[] {1,2,3,4,5};

var threeFirstNumbers = numbers.Take(3);
Console.WriteLine(string.Join(",", threeFirstNumbers.ToArray()));
```

```
//1,2,3
```

SequenceEqual

```
var numbers = new[] {1,2,3,4,5};
var sameNumbers = new[] {1,2,3,4,5};
var sameNumbersInDifferentOrder = new[] {5,1,4,2,3};

var equalIfSameOrder = numbers.SequenceEqual(sameNumbers);
Console.WriteLine(equalIfSameOrder); //True

var equalIfDifferentOrder = numbers.SequenceEqual(sameNumbersInDifferentOrder);
Console.WriteLine(equalIfDifferentOrder); //False
```

```
var numbers = new[] {1,2,3,4,5};
var reversed = numbers.Reverse();

Console.WriteLine(string.Join(",", reversed.ToArray()));

//5,4,3,2,1
```

OfType

```
var mixed = new object[] {1,"Foo",2,"Bar",3,"Fizz",4,"Buzz"};
var numbers = mixed.OfType<int>();

Console.WriteLine(string.Join(",", numbers.ToArray()));

//1,2,3,4
```

```
var numbers = new[] {1,2,3,4};

var maxNumber = numbers.Max();
Console.WriteLine(maxNumber); //4

var cities = new[] {
    new {Population = 1000},
    new {Population = 2500},
    new {Population = 4000}
};

var maxPopulation = cities.Max(c => c.Population);
Console.WriteLine(maxPopulation); //4000
```

```
var numbers = new[] {1,2,3,4};

var minNumber = numbers.Min();
Console.WriteLine(minNumber); //1

var cities = new[] {
    new {Population = 1000},
    new {Population = 2500},
    new {Population = 4000}
```

```
};

var minPopulation = cities.Min(c => c.Population);
Console.WriteLine(minPopulation); //1000
```

```
var numbers = new[] {1,2,3,4};

var averageNumber = numbers.Average();
Console.WriteLine(averageNumber);
// 2,5
```

このメソッドは、なのをします。

```
var cities = new[] {
    new {Population = 1000},
    new {Population = 2000},
    new {Population = 4000}
};

var averagePopulation = cities.Average(c => c.Population);
Console.WriteLine(averagePopulation);
// 2333,33
```

このメソッドは、されたをしてなのをします。

ジップ

.NET 4.0

```
var tens = new[] {10,20,30,40,50};
var units = new[] {1,2,3,4,5};

var sums = tens.Zip(units, (first, second) => first + second);

Console.WriteLine(string.Join(",", sums));

//11,22,33,44,55
```

な

```
var numbers = new[] {1, 1, 2, 2, 3, 3, 4, 4, 5, 5};
var distinctNumbers = numbers.Distinct();

Console.WriteLine(string.Join(",", distinctNumbers));

//1,2,3,4,5
```

GroupBy

```
var persons = new[] {
    new { Name="Fizz", Job="Developer"},
    new { Name="Buzz", Job="Developer"},
```

```
    new { Name="Foo", Job="Astronaut"},
    new { Name="Bar", Job="Astronaut"},
};

var groupedByJob = persons.GroupBy(p => p.Job);

foreach(var theGroup in groupedByJob)
{
    Console.WriteLine(
        "{0} are {1}s",
        string.Join(",", theGroup.Select(g => g.Name).ToArray()),
        theGroup.Key);
}

//Fizz,Buzz are Developers
//Foo,Bar are Astronauts
```

のをグルーープし、レコーード、、でしいオブジェクトをする

```
var a = db.Invoices.GroupBy(i => i.Country)
          .Select(g => new { Country = g.Key,
                             Count = g.Count(),
                             Total = g.Sum(i => i.Paid),
                             Average = g.Average(i => i.Paid) });
```

だけをむならば、グルーープはありません

```
var a = db.Invoices.GroupBy(i => 1)
          .Select(g => new { Count = g.Count(),
                             Total = g.Sum(i => i.Paid),
                             Average = g.Average(i => i.Paid) });
```

かカウントがな

```
var a = db.Invoices.GroupBy(g => 1)
          .Select(g => new { High = g.Count(i => i.Paid >= 1000),
                             Low = g.Count(i => i.Paid < 1000),
                             Sum = g.Sum(i => i.Paid) });
```

ToDictionary

されたkeySelectorをしてキーーをする`IEnumerable`ソーースからしいをします。 keySelectorがインジェクションでないソーースコレクションのメンバにのをします、 `ArgumentException`スローーします。するとキーーをできるオーーバーーローードがあります。

```
var persons = new[] {
    new { Name="Fizz", Id=1},
    new { Name="Buzz", Id=2},
    new { Name="Foo", Id=3},
    new { Name="Bar", Id=4},
};
```

ちょうどキーーセレクターーをすると、され`Dictionary<TKey,TVal>`と`TKey`セレクタ、のり`TVal`のオブジ

ェクト·タイプ、およびされたとのオブジェクト。

```
var personsById = persons.ToDictionary(p => p.Id);
// personsById is a Dictionary<int,object>

Console.WriteLine(personsById[1].Name); //Fizz
Console.WriteLine(personsById[2].Name); //Buzz
```

セレクタをすると、 `Dictionary<TKey,TVal>`もされますが、 `TKey`はキ―セレクタのりのですが、`TVal`はセレクタのりのとりをとしてします。

```
var namesById = persons.ToDictionary(p => p.Id, p => p.Name);
//namesById is a Dictionary<int,string>

Console.WriteLine(namesById[3]); //Foo
Console.WriteLine(namesById[4]); //Bar
```

のように、キ―セレクタによってされるキ―はでなければなりません。はをスロ―します。

```
var persons = new[] {
    new { Name="Fizz", Id=1},
    new { Name="Buzz", Id=2},
    new { Name="Foo", Id=3},
    new { Name="Bar", Id=4},
    new { Name="Oops", Id=4}
};

var willThrowException = persons.ToDictionary(p => p.Id)
```

ソ―スコレクションにしてのキ―をできないは、わりにToLookupをすることをしてください。、ToLookupはToDictionaryとにしますが、のLookupではキ―はするキ―をつのコレクションとペアになります。

```
var numbers1to5 = new[] {1,2,3,4,5};
var numbers4to8 = new[] {4,5,6,7,8};

var numbers1to8 = numbers1to5.Union(numbers4to8);

Console.WriteLine(string.Join(",", numbers1to8));

//1,2,3,4,5,6,7,8
```

がからされることにしてください。これがましくないは、わりに`Concat`してください。

ToArray

```
var numbers = new[] {1,2,3,4,5,6,7,8,9,10};
var someNumbers = numbers.Where(n => n < 6);

Console.WriteLine(someNumbers.GetType().Name);
//WhereArrayIterator`1
```

```
var someNumbersArray = someNumbers.ToArray();

Console.WriteLine(someNumbersArray.GetType().Name);
//Int32[]
```

ToList

```
var numbers = new[] {1,2,3,4,5,6,7,8,9,10};
var someNumbers = numbers.Where(n => n < 6);

Console.WriteLine(someNumbers.GetType().Name);
//WhereArrayIterator`1

var someNumbersList = someNumbers.ToList();

Console.WriteLine(
    someNumbersList.GetType().Name + " - " +
    someNumbersList.GetType().GetGenericArguments()[0].Name);
//List`1 - Int32
```

カウント

```
IEnumerable<int> numbers = new[] {1,2,3,4,5,6,7,8,9,10};

var numbersCount = numbers.Count();
Console.WriteLine(numbersCount); //10

var evenNumbersCount = numbers.Count(n => (n & 1) == 0);
Console.WriteLine(evenNumbersCount); //5
```

ElementAt

```
var names = new[] {"Foo","Bar","Fizz","Buzz"};

var thirdName = names.ElementAt(2);
Console.WriteLine(thirdName); //Fizz

//The following throws ArgumentOutOfRangeException

var minusOnethName = names.ElementAt(-1);
var fifthName = names.ElementAt(4);
```

ElementAtOrDefault

```
var names = new[] {"Foo","Bar","Fizz","Buzz"};

var thirdName = names.ElementAtOrDefault(2);
Console.WriteLine(thirdName); //Fizz

var minusOnethName = names.ElementAtOrDefault(-1);
Console.WriteLine(minusOnethName); //null

var fifthName = names.ElementAtOrDefault(4);
Console.WriteLine(fifthName); //null
```

SkipWhile

```
var numbers = new[] {2,4,6,8,1,3,5,7};

var oddNumbers = numbers.SkipWhile(n => (n & 1) == 0);

Console.WriteLine(string.Join(",", oddNumbers.ToArray()));

//1,3,5,7
```

TakeWhile

```
var numbers = new[] {2,4,6,1,3,5,7,8};

var evenNumbers = numbers.TakeWhile(n => (n & 1) == 0);

Console.WriteLine(string.Join(",", evenNumbers.ToArray()));

//2,4,6
```

DefaultIfEmpty

```
var numbers = new[] {2,4,6,8,1,3,5,7};

var numbersOrDefault = numbers.DefaultIfEmpty();
Console.WriteLine(numbers.SequenceEqual(numbersOrDefault)); //True

var noNumbers = new int[0];

var noNumbersOrDefault = noNumbers.DefaultIfEmpty();
Console.WriteLine(noNumbersOrDefault.Count()); //1
Console.WriteLine(noNumbersOrDefault.Single()); //0

var noNumbersOrExplicitDefault = noNumbers.DefaultIfEmpty(34);
Console.WriteLine(noNumbersOrExplicitDefault.Count()); //1
Console.WriteLine(noNumbersOrExplicitDefault.Single()); //34
```

りたたみ

ステップでしいオブジェクトをする

```
var elements = new[] {1,2,3,4,5};

var commaSeparatedElements = elements.Aggregate(
    seed: "",
    func: (aggregate, element) => $"{aggregate}{element},");

Console.WriteLine(commaSeparatedElements);  //1,2,3,4,5,
```

すべてのステップでじオブジェクトをする

```
var commaSeparatedElements2 = elements.Aggregate(
```

```
    seed: new StringBuilder(),
    func: (seed, element) => seed.Append($"{element},"));

Console.WriteLine(commaSeparatedElements2.ToString());  //1,2,3,4,5,
```

セレクタの

```
var commaSeparatedElements3 = elements.Aggregate(
    seed: new StringBuilder(),
    func: (seed, element) => seed.Append($"{element},"),
    resultSelector: (seed) => seed.ToString());
Console.WriteLine(commaSeparatedElements3);  //1,2,3,4,5,
```

シードがされた、のはシードになります。

```
var seedAndElements = elements.Select(n=>n.ToString());
var commaSeparatedElements4 = seedAndElements.Aggregate(
    func: (aggregate, element) => $"{aggregate}{element},");

Console.WriteLine(commaSeparatedElements4);  //12,3,4,5,
```

げる

```
var persons = new[] {
    new { Name="Fizz", Job="Developer"},
    new { Name="Buzz", Job="Developer"},
    new { Name="Foo", Job="Astronaut"},
    new { Name="Bar", Job="Astronaut"},
};

var groupedByJob = persons.ToLookup(p => p.Job);

foreach(var theGroup in groupedByJob)
{
    Console.WriteLine(
        "{0} are {1}s",
        string.Join(",", theGroup.Select(g => g.Name).ToArray()),
        theGroup.Key);
}

//Fizz,Buzz are Developers
//Foo,Bar are Astronauts
```

する

```
class Developer
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class Project
{
    public int DeveloperId { get; set; }
```

```
    public string Name { get; set; }
}

var developers = new[] {
    new Developer {
        Id = 1,
        Name = "Foobuzz"
    },
    new Developer {
        Id = 2,
        Name = "Barfizz"
    }
};

var projects = new[] {
    new Project {
        DeveloperId = 1,
        Name = "Hello World 3D"
    },
    new Project {
        DeveloperId = 1,
        Name = "Super Fizzbuzz Maker"
    },
    new Project {
        DeveloperId = 2,
        Name = "Citizen Kane - The action game"
    },
    new Project {
        DeveloperId = 2,
        Name = "Pro Pong 2016"
    }
};

var denormalized = developers.Join(
    inner: projects,
    outerKeySelector: dev => dev.Id,
    innerKeySelector: proj => proj.DeveloperId,
    resultSelector:
        (dev, proj) => new {
            ProjectName = proj.Name,
            DeveloperName = dev.Name});

foreach(var item in denormalized)
{
    Console.WriteLine("{0} by {1}", item.ProjectName, item.DeveloperName);
}

//Hello World 3D by Foobuzz
//Super Fizzbuzz Maker by Foobuzz
//Citizen Kane - The action game by Barfizz
//Pro Pong 2016 by Barfizz
```

GroupJoin

```
class Developer
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

```
class Project
{
    public int DeveloperId { get; set; }
    public string Name { get; set; }
}

var developers = new[] {
    new Developer {
        Id = 1,
        Name = "Foobuzz"
    },
    new Developer {
        Id = 2,
        Name = "Barfizz"
    }
};

var projects = new[] {
    new Project {
        DeveloperId = 1,
        Name = "Hello World 3D"
    },
    new Project {
        DeveloperId = 1,
        Name = "Super Fizzbuzz Maker"
    },
    new Project {
        DeveloperId = 2,
        Name = "Citizen Kane - The action game"
    },
    new Project {
        DeveloperId = 2,
        Name = "Pro Pong 2016"
    }
};

var grouped = developers.GroupJoin(
    inner: projects,
    outerKeySelector: dev => dev.Id,
    innerKeySelector: proj => proj.DeveloperId,
    resultSelector:
        (dev, projs) => new {
            DeveloperName = dev.Name,
            ProjectNames = projs.Select(p => p.Name).ToArray()});

foreach(var item in grouped)
{
    Console.WriteLine(
        "{0}'s projects: {1}",
        item.DeveloperName,
        string.Join(", ", item.ProjectNames));
}

//Foobuzz's projects: Hello World 3D, Super Fizzbuzz Maker
//Barfizz's projects: Citizen Kane - The action game, Pro Pong 2016
```

キャスト

`Cast`は、 `IEnumerable`メソッドであり、 `IEnumerable<T>`ではなく、 `Enumerable`ののメソッドとはなり

Enumerable。 したがって、のインスタンスをのインスタンスにするためにできます。

これは、 ArrayListはIEnumerable<T>していないため、コンパイルされません。

```
var numbers = new ArrayList() {1,2,3,4,5};
Console.WriteLine(numbers.First());
```

どおりにします。

```
var numbers = new ArrayList() {1,2,3,4,5};
Console.WriteLine(numbers.Cast<int>().First()); //1
```

Castはキャストをしません 。のコンパイルはにInvalidCastExceptionをスローします。

```
var numbers = new int[] {1,2,3,4,5};
decimal[] numbersAsDecimal = numbers.Cast<decimal>().ToArray();
```

コレクションへのキャストをするなは、のとおりです。

```
var numbers= new int[] {1,2,3,4,5};
decimal[] numbersAsDecimal = numbers.Select(n => (decimal)n).ToArray();
```

の

intののIEnumerableをするには

```
IEnumerable<int> emptyList = Enumerable.Empty<int>();
```

こののIEnumerableは、タイプTにしてキャッシュされるため、のようになります。

```
Enumerable.Empty<decimal>() == Enumerable.Empty<decimal>(); // This is True
Enumerable.Empty<int>() == Enumerable.Empty<decimal>();     // This is False
```

に、

ThenByはOrderByののにのみでき、のをしてすることができます

```
var persons = new[]
{
    new {Id = 1, Name = "Foo", Order = 1},
    new {Id = 1, Name = "FooTwo", Order = 2},
    new {Id = 2, Name = "Bar", Order = 2},
    new {Id = 2, Name = "BarTwo", Order = 1},
    new {Id = 3, Name = "Fizz", Order = 2},
    new {Id = 3, Name = "FizzTwo", Order = 1},
};

var personsSortedByName = persons.OrderBy(p => p.Id).ThenBy(p => p.Order);
```

```
Console.WriteLine(string.Join(",", personsSortedByName.Select(p => p.Name)));
//This will display :
//Foo,FooTwo,BarTwo,Bar,FizzTwo,Fizz
```

Rangeの2つのパラメータは、 のとするのですのではありません。

```
// prints 1,2,3,4,5,6,7,8,9,10
Console.WriteLine(string.Join(",", Enumerable.Range(1, 10)));

// prints 10,11,12,13,14
Console.WriteLine(string.Join(",", Enumerable.Range(10, 5)));
```

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public Person Owner { get; set; }
}

public static void Main(string[] args)
{
    var magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };
    var terry = new Person { FirstName = "Terry", LastName = "Adams" };

    var barley = new Pet { Name = "Barley", Owner = terry };

    var people = new[] { magnus, terry };
    var pets = new[] { barley };

    var query =
        from person in people
        join pet in pets on person equals pet.Owner into gj
        from subpet in gj.DefaultIfEmpty()
        select new
        {
            person.FirstName,
            PetName = subpet?.Name ?? "-" // Use - if he has no pet
        };

    foreach (var p in query)
        Console.WriteLine($"{p.FirstName}: {p.PetName}");
}
```

りす

Enumerable.Repeatは、りしのシーケンスをします。このでは、「Hello」を4します。

```
var repeats = Enumerable.Repeat("Hello", 4);

foreach (var item in repeats)
{
```

```
    Console.WriteLine(item);
}

/* output:
    Hello
    Hello
    Hello
    Hello
*/
```

オンラインでLINQをむ https://riptutorial.com/ja/dot-net/topic/34/linq

17: NuGetパッケージングシステム

NuGet.org

> NuGetは、.NETをむMicrosoftプラットフォームのパッケージマネージャです。 NuGetクライアントツールは、パッケージをしてするをします。 NuGetギャラリーは、すべてのパッケージとコンシューマがするパッケージリポジトリです。

はNuGet.orgのによるものです 。

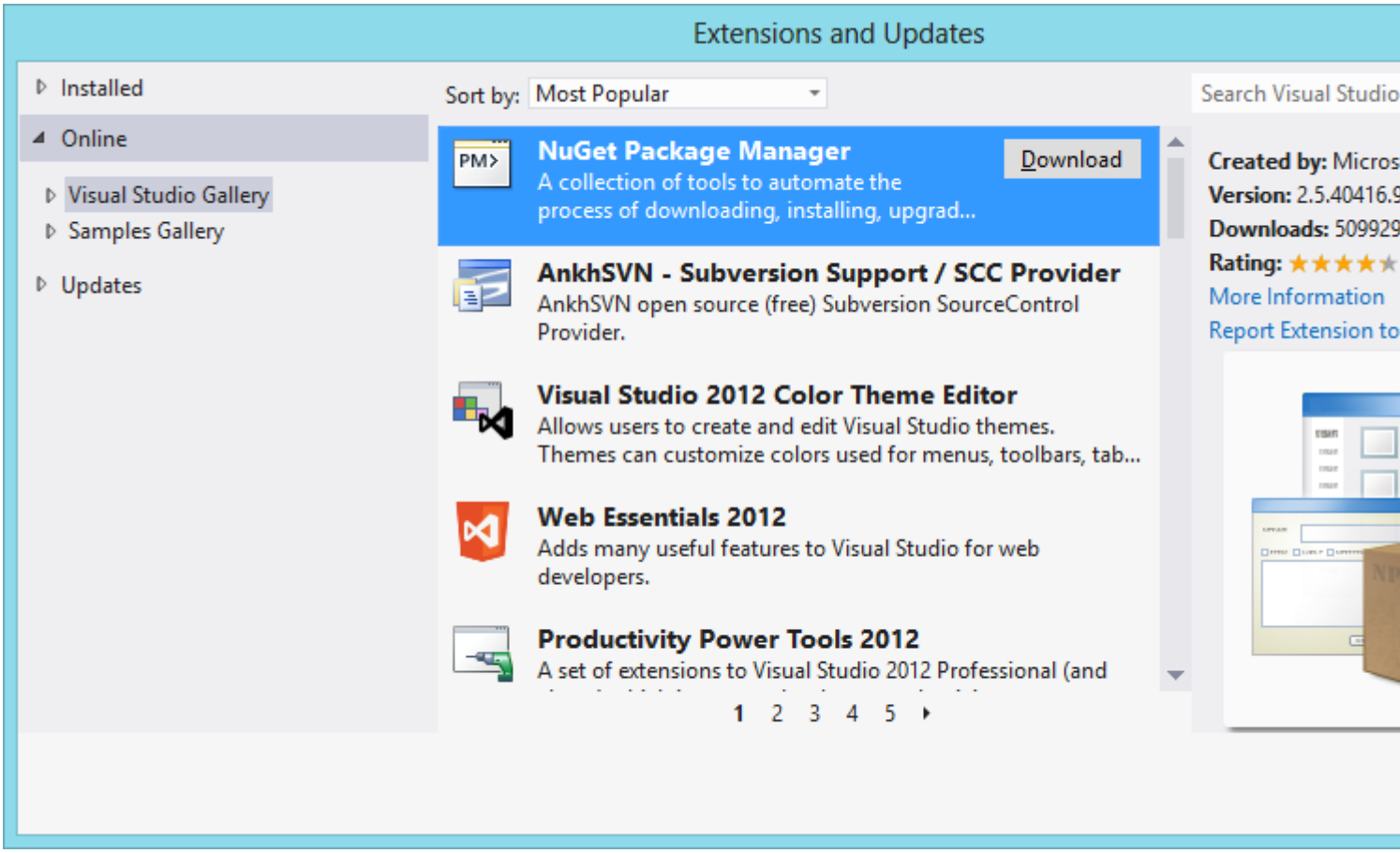
Examples

NuGet Package Managerのインストール

プロジェクトのパッケージをできるようにするには、NuGet Package Managerがです。これはVisual Studio Extensionです。のドキュメント「 NuGet Clientのインストールと 」でしています。

Visual Studio 2012、NuGetはすべてのエディションにまれており、ツール→NuGetパッケージマネージャ→パッケージマネージャコンソールからできます。

Visual Studioの[ツール]メニューから、[と]をクリックします。



これは、のGUIをインストールします。

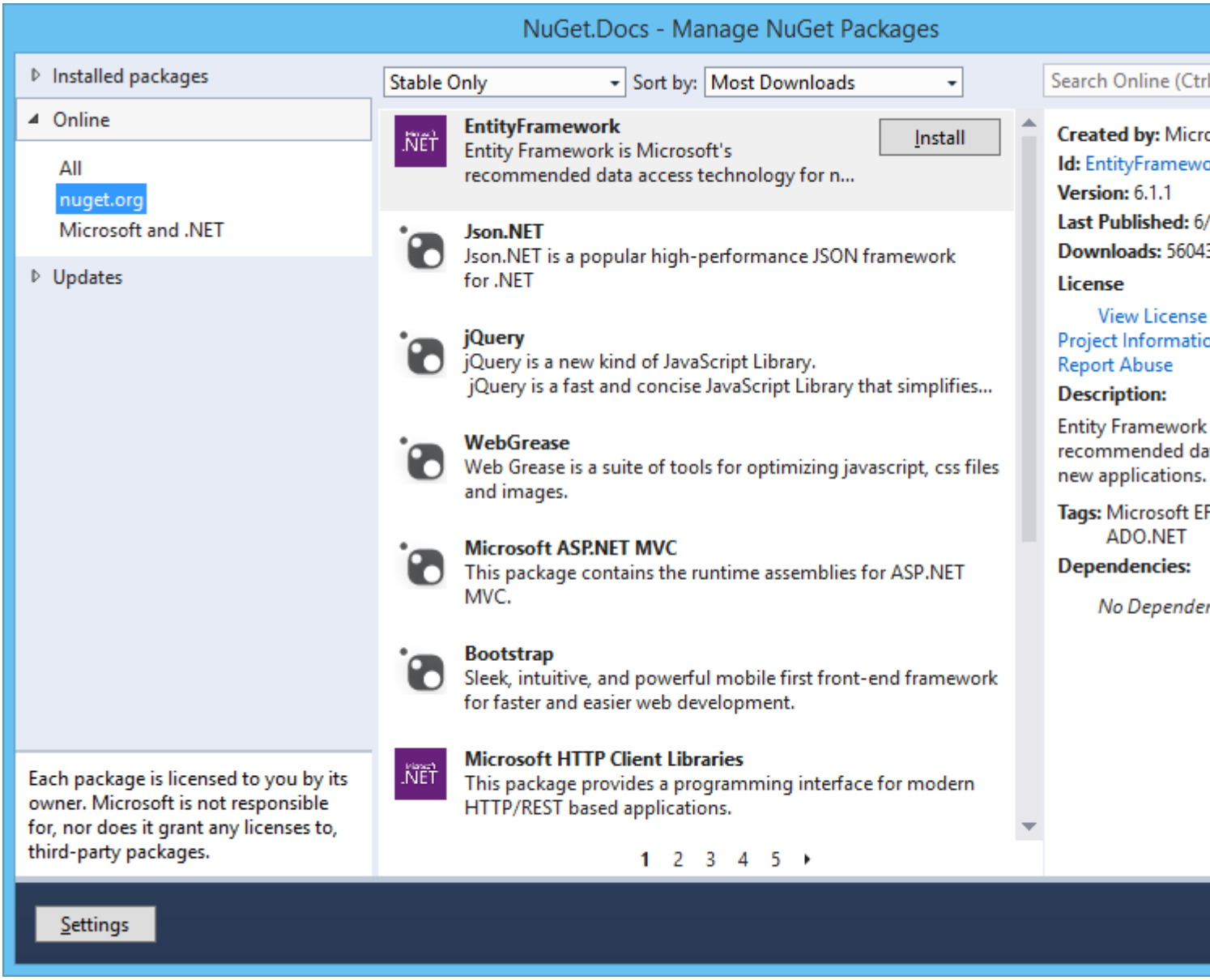
- プロジェクトまたはReferencesフォルダの "Manage NuGet Packages ..."をクリックするとできます

また、パッケージマネージャコンソール

- ツール - > NuGetパッケージマネージャー - >パッケージマネージャーコンソール。

UIをしたパッケージの

プロジェクトまたはReferencesフォルダをクリックすると、[Manage NuGet Packages ...]オプションをクリックできます。 パッケージマネージャダイアログがされます。



コンソールからパッケージをする

[Tools] - > [NuGet Package Manager] - > [Package Manager Console]のメニューをクリックして、IDEにコンソールをします。 のドキュメントはこちら 。

ここでは、パッケージをされている「デフォルトプロジェクト」に`install-package`コマンドをすることができます。

```
Install-Package Elmah
```

パッケージをインストールするプロジェクトをして、[デフォルトプロジェクト]ドロップダウンでしたプロジェクトをきすることもできます。

```
Install-Package Elmah -ProjectName MyFirstWebsite
```

パッケージの

パッケージをするには、のコマンドをします。

```
PM> Update-Package EntityFramework
```

EntityFrameworkはするパッケージのです。はすべてのプロジェクトでされるため、「デフォルトプロジェクト」にのみインストールする`Install-Package EntityFramework`とはなり`Install-Package EntityFramework`。

1つのプロジェクトをにすることもできます。

```
PM> Update-Package EntityFramework -ProjectName MyFirstWebsite
```

パッケージのアン インストール

```
PM> Uninstall-Package EntityFramework
```

ソリューションの1つのプロジェクトからパッケージをアン インストールする

```
PM> Uninstall-Package -ProjectName MyProjectB EntityFramework
```

のバージョンのパッケージをインストールする

```
PM> Install-Package EntityFramework -Version 6.1.2
```

パッケージソースフィードMyGet、Klondikeなどをすると、

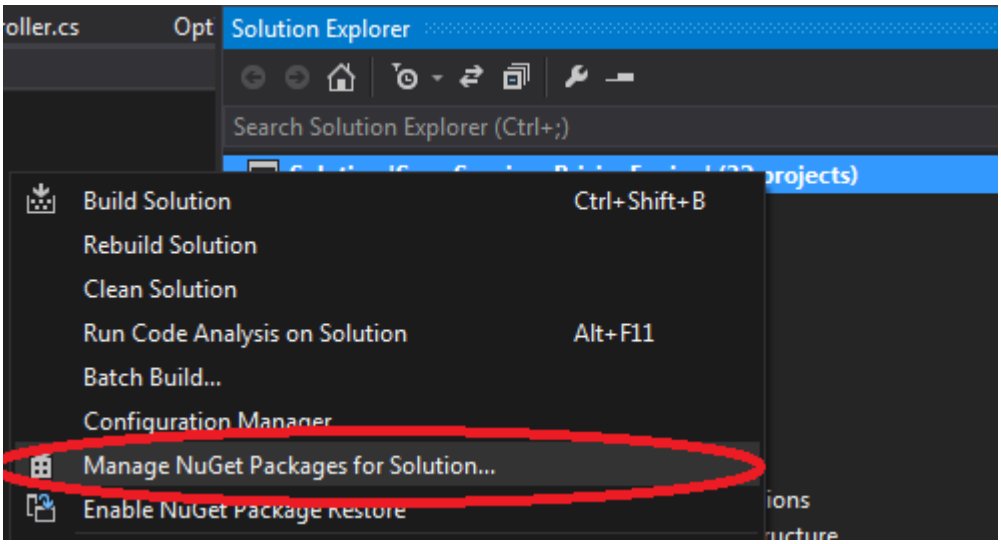
```
nuget sources add -name feedname -source http://sourcefeedurl
```

UIをしてなるローカルNugetパッケージソースをする

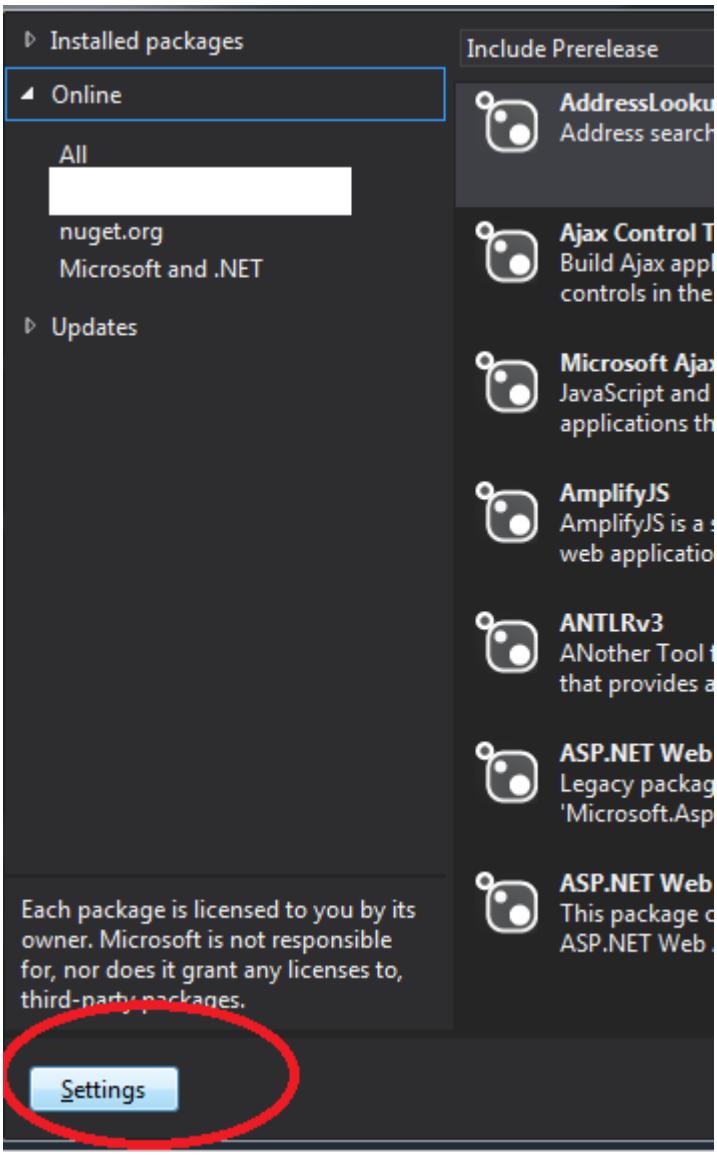
がなるチームでパッケージをするために、それのナゲットサーバーをセットアップするのはです

。

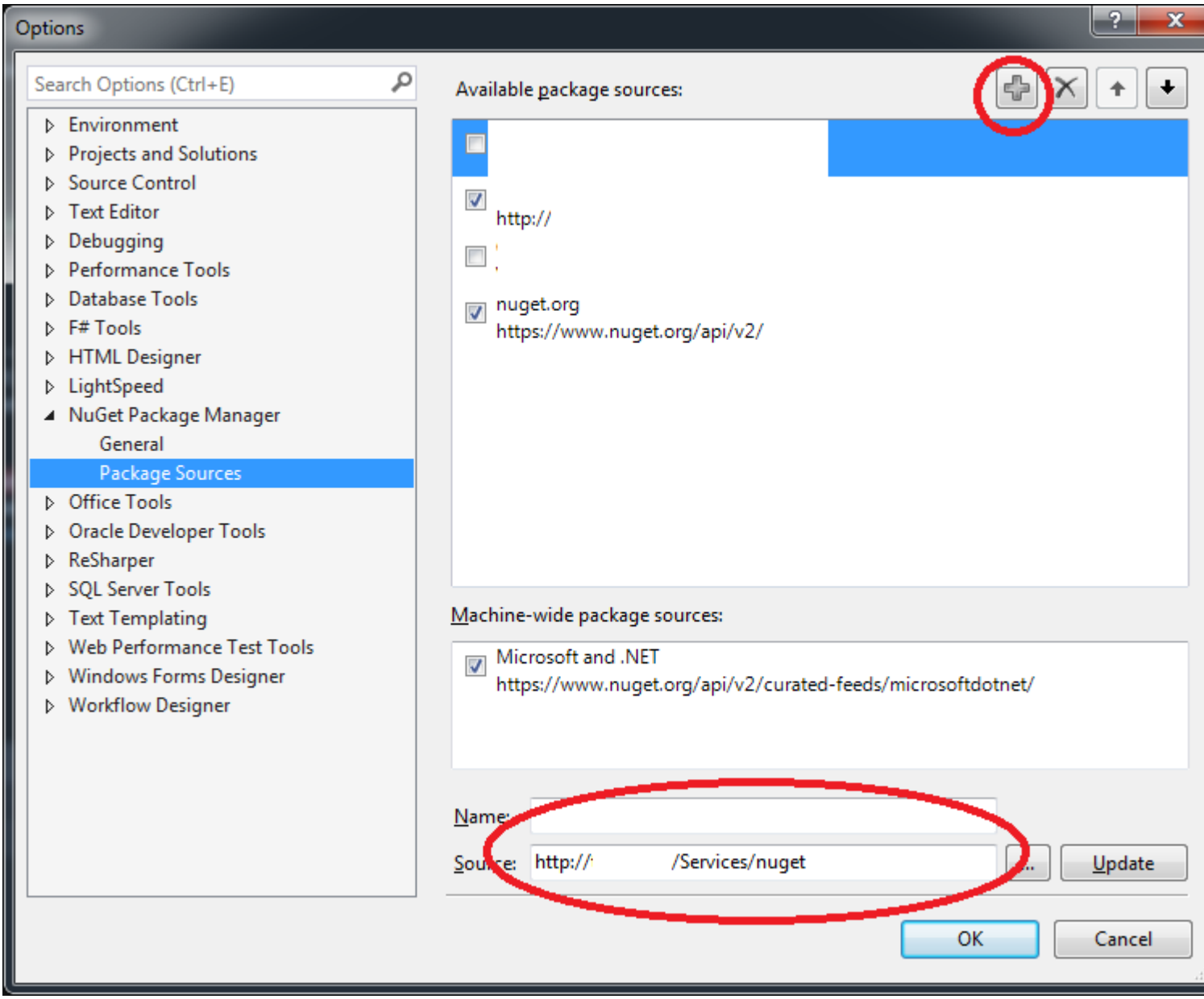
1. ソリューションエクスプローラにし、 マウスボタンをクリックし`Manage NuGet Packages for Solution`をします。



2. いているウィンドウで[`Settings`クリックします



3. にある+をクリックし、ローカルナゲットサーバーにポイントするURLとをします。



のバージョンのパッケージをアンインストールする

```
PM> uninstall-Package EntityFramework -Version 6.1.2
```

オンラインでNuGetパッケージングシステムをむ https://riptutorial.com/ja/dot-net/topic/43/nugetパッケージングシステム

18: ReadOnlyCollections

`ReadOnlyCollection`は、のコレクション'ソースコレクション'にしてみりビューをします。

アイテムは、 `ReadOnlyCollection`またはされません。わりに、それらはされてソースコレクションからされ、 `ReadOnlyCollection`はこれらのをソースにします。

`ReadOnlyCollection`ののとはできませんが、にあるとして、のプロパティをしてメソッドをびすことができます。

コードでコレクションをせずにできるようにするには`ReadOnlyCollection`しますが、コレクションはですることができます。

- `ObservableCollection<T>`
- `ReadOnlyObservableCollection<T>`

ReadOnlyCollectionsImmutableCollection

`ReadOnlyCollection`は、あなたがした`ImmutableCollection`できないというで、 `ImmutableCollection`となります`ImmutableCollection`は、に`n`をみ、きえたりべえたりすることはできません。、`ReadOnlyCollection`はすることはできませんが、ソースコレクションをしてを//べえすることはできます。

Examples

ReadOnlyCollectionの

コンストラクタの

`ReadOnlyCollection`は、の`IList`オブジェクトをコンストラクタにすことによってされ`IList`。

```
var groceryList = new List<string> { "Apple", "Banana" };
var readOnlyGroceryList = new ReadOnlyCollection<string>(groceryList);
```

LINQの

さらに、LINQは`IList`オブジェクトの`AsReadOnly()`メソッドをします。

```
var readOnlyVersion = groceryList.AsReadOnly();
```

、ソースコレクションをにして、 `ReadOnlyCollection`へのパブリックアクセスをします。インラインリストから`ReadOnlyCollection`をすることはできますが、コレクションをしたでコレクションをすることはできません。

```
var readOnlyGroceryList = new List<string> {"Apple", "Banana"}.AsReadOnly();
// Great, but you will not be able to update the grocery list because
//  you do not have a reference to the source list anymore!
```

これをうことがわかったは、 `ImmutableCollection`などののデータのをすることをおめします。

ReadOnlyCollectionの

`ReadOnlyCollection`はできません。わりに、ソースコレクションがされ、 `ReadOnlyCollection`にこれらのがされます。これは、 `ReadOnlyCollection`なです。

```
var groceryList = new List<string> { "Apple", "Banana" };

var readOnlyGroceryList = new ReadOnlyCollection<string>(groceryList);

var itemCount = readOnlyGroceryList.Count;  // There are currently 2 items

//readOnlyGroceryList.Add("Candy");         // Compiler Error - Items cannot be added to a
ReadOnlyCollection object
groceryList.Add("Vitamins");                // ..but they can be added to the original
collection

itemCount = readOnlyGroceryList.Count;      // Now there are 3 items
var lastItem = readOnlyGroceryList.Last();  // The last item on the read only list is now
"Vitamins"
```

デモをる

ReadOnlyCollectionのは、にはみりではありません

ソースコレクションのタイプがでない、 `ReadOnlyCollection`アクセスされるエレメントはできます。

```
public class Item
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}

public static void FillOrder()
{
    // An order is generated
    var order = new List<Item>
    {
        new Item { Name = "Apple", Price = 0.50m },
        new Item { Name = "Banana", Price = 0.75m },
        new Item { Name = "Vitamins", Price = 5.50m }
    };

    // The current sub total is $6.75
    var subTotal = order.Sum(item => item.Price);

    // Let the customer preview their order
    var customerPreview = new ReadOnlyCollection<Item>(order);
```

```
    // The customer can't add or remove items, but they can change
    //   the price of an item, even though it is a ReadOnlyCollection
    customerPreview.Last().Price = 0.25m;

    // The sub total is now only $1.50!
    subTotal = order.Sum(item => item.Price);
}
```

デモをる

オンラインでReadOnlyCollectionsをむ https://riptutorial.com/ja/dot-net/topic/6906/readonlycollections

19: StdErrストリームへのきみとみし

Examples

コンソールをしてエラーにきむ

```
var sourceFileName = "NonExistingFile";
try
{
    System.IO.File.Copy(sourceFileName, "DestinationFile");
}
catch (Exception e)
{
    var stdErr = Console.Error;
    stdErr.WriteLine($"Failed to copy '{sourceFileName}': {e.Message}");
}
```

プロセスのエラーからのみり

```
var errors = new System.Text.StringBuilder();
var process = new Process
{
    StartInfo = new ProcessStartInfo
    {
        RedirectStandardError = true,
        FileName = "xcopy.exe",
        Arguments = "\"NonExistingFile\" \"DestinationFile\"",
        UseShellExecute = false
    },

};
process.ErrorDataReceived += (s, e) => errors.AppendLine(e.Data);
process.Start();
process.BeginErrorReadLine();
process.WaitForExit();

if (errors.Length > 0) // something went wrong
    System.Console.Error.WriteLine($"Child process error: \r\n {errors}");
```

オンラインでStdErrストリームへのきみとみしをむ https://riptutorial.com/ja/dot-net/topic/10779/stderrストリームへのきみとみし

20: System.Diagnostics

Examples

ストップウォッチ

このでは、 `Stopwatch`をしてコードブロックをベンチマークするをします。

```
using System;
using System.Diagnostics;

public class Benchmark : IDisposable
{
    private Stopwatch sw;

    public Benchmark()
    {
        sw = Stopwatch.StartNew();
    }

    public void Dispose()
    {
        sw.Stop();
        Console.WriteLine(sw.Elapsed);
    }
}

public class Program
{
    public static void Main()
    {
        using (var bench = new Benchmark())
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

シェルコマンドをする

```
string strCmdText = "/C copy /b Image1.jpg + Archive.rar Image2.jpg";
System.Diagnostics.Process.Start("CMD.exe",strCmdText);
```

これは、cmdウィンドウをにします。

```
System.Diagnostics.Process process = new System.Diagnostics.Process();
System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
startInfo.FileName = "cmd.exe";
startInfo.Arguments = "/C copy /b Image1.jpg + Archive.rar Image2.jpg";
process.StartInfo = startInfo;
process.Start();
```

コマンドを**CMD**にり、をける

このメソッドをすると、 `Cmd.exe command`をし、エラ―もむをとしてします。

```
private static string SendCommand(string command)
{
    var cmdOut = string.Empty;

    var startInfo = new ProcessStartInfo("cmd", command)
    {
        WorkingDirectory = @"C:\Windows\System32", // Directory to make the call from
        WindowStyle = ProcessWindowStyle.Hidden,   // Hide the window
        UseShellExecute = false,                   // Do not use the OS shell to start the
process
        CreateNoWindow = true,                     // Start the process in a new window
        RedirectStandardOutput = true,             // This is required to get STDOUT
        RedirectStandardError = true               // This is required to get STDERR
    };

    var p = new Process {StartInfo = startInfo};

    p.Start();

    p.OutputDataReceived += (x, y) => cmdOut += y.Data;
    p.ErrorDataReceived += (x, y) => cmdOut += y.Data;
    p.BeginOutputReadLine();
    p.BeginErrorReadLine();
    p.WaitForExit();
    return cmdOut;
}
```

```
var servername = "SVR-01.domain.co.za";
var currentUsers = SendCommand($"/C QUERY USER /SERVER:{servername}")
```

> currentUsers = "USERNAMEセッションID IDIDLEログオンJoe.Bloggs ica-cgp0 2アクティブ24692 + 1329 25/07/2016 07:50 Jim.McFlannegan ica-cgp1 3アクティブ25/07 / 2016 08:33 Andy.McAnderson ica-cgp2 4アクティブ25/07/2016 08:54 John.Smith ica-cgp4 5アクティブ14 25/07/2016 08:57 Bob.Bobbington ica-cgp5 6 Active 24692 + 13 29 25/07/2016 09:05 Tim.Tom ica-cgp6 7アクティブ25/07/2016 09:08 Bob.Joges ica-cgp7 8 Active 24692 + 1329 25 / 07/2016 09:13 "

によっては、のサ―バ―へのアクセスがのユ―ザ―にされることがあります。このユ―ザ―のログインをっているは、このでクエリをすることができます。

```
private static string SendCommand(string command)
{
    var cmdOut = string.Empty;

    var startInfo = new ProcessStartInfo("cmd", command)
    {
        WorkingDirectory = @"C:\Windows\System32",
        WindowStyle = ProcessWindowStyle.Hidden,    // This does not actually work in
conjunction with "runas" - the console window will still appear!
```

```
        UseShellExecute = false,
        CreateNoWindow = true,
        RedirectStandardOutput = true,
        RedirectStandardError = true,

        Verb = "runas",
        Domain = "doman1.co.za",
        UserName = "administrator",
        Password = GetPassword()
    };

    var p = new Process {StartInfo = startInfo};

    p.Start();

    p.OutputDataReceived += (x, y) => cmdOut += y.Data;
    p.ErrorDataReceived += (x, y) => cmdOut += y.Data;
    p.BeginOutputReadLine();
    p.BeginErrorReadLine();
    p.WaitForExit();
    return cmdOut;
}
```

パスワードの

```
static SecureString GetPassword()
{
    var plainText = "password123";
    var ss = new SecureString();
    foreach (char c in plainText)
    {
        ss.AppendChar(c);
    }

    return ss;
}
```

ノート

ののメソッドは、 `OutputDataReceived`と`ErrorDataReceived`がじ`cmdOut`されるため、STDOUTとSTDERRのをします。

オンラインでSystem.Diagnosticsをむ https://riptutorial.com/ja/dot-net/topic/3143/system-diagnostics

21: System.IO

Examples

StreamReaderをしてテキストファイルをみむ

```
string fullOrRelativePath = "testfile.txt";

string fileData;

using (var reader = new StreamReader(fullOrRelativePath))
{
    fileData = reader.ReadToEnd();
}
```

この`StreamReader`コンストラクタのオーバーロードは、ファイルでされているのエンコーディングにするとしないがあるエンコーディングをいます。

`System.IO.File`クラスでできるファイル、つまり`File.ReadAllText(path)`と`File.ReadAllLines(path)`からすべてのテキストをみるなメソッドがいくつかあることにしてください。

System.IO.Fileをしたデータのみり/きみ

まず、ファイルからデータをする3つのなるをてみましょう。

```
string fileText = File.ReadAllText(file);
string[] fileLines = File.ReadAllLines(file);
byte[] fileBytes = File.ReadAllBytes(file);
```

- のでは、ファイルのすべてのデータをとしてみみます。
- 2では、ファイルのデータをにみみます。ファイルのはのになります。
- 3に、ファイルからバイトをみみます。

に、データをファイルにする 3つのなるをてみましょう。したファイルがしない、メソッドはデータをするにファイルをにします。

```
 File.AppendAllText(file, "Here is some data that is\nappended to the file.");
 File.AppendAllLines(file, new string[2] { "Here is some data that is", "appended to the
file." });
 using (StreamWriter stream = File.AppendText(file))
 {
     stream.WriteLine("Here is some data that is");
     stream.Write("appended to the file.");
 }
```

- のでは、されたファイルのにをするだけです。
- 2では、のをファイルのしいにします。
-

に3で`File.AppendText`をして`File.AppendText`をき、きまれたデ―タをします。

に、ファイルにデ―タをきむ3つのなるをてみましょう。 ときみのいは、ファイルのデ―タにをしながら、 きみがデ―タをファイルにきすることです。したファイルがしない、メソッドはデ―タをきむにファイルをにします。

```
File.WriteAllText(file, "here is some data\nin this file.");
File.WriteAllLines(file, new string[2] { "here is some data", "in this file" });
File.WriteAllBytes(file, new byte[2] { 0, 255 });
```

- のはをファイルにきみます。
- 2はのをファイルのそれのにきみます。
- そして3では、バイトをファイルにきむことができます。

System.IO.SerialPortsをするシ リアル ポ―ト

されたシ リアル ポ―トをする

```
using System.IO.Ports;
string[] ports = SerialPort.GetPortNames();
for (int i = 0; i < ports.Length; i++)
{
    Console.WriteLine(ports[i]);
}
```

System.IO.SerialPortオブジェクトのインスタンス

```
using System.IO.Ports;
SerialPort port = new SerialPort();
SerialPort port = new SerialPort("COM 1"); ;
SerialPort port = new SerialPort("COM 1", 9600);
```

これらは、SerialPortタイプのコンストラクタの7つのオ―バ―ロ―ドのうちの3つのみです。

SerialPortをしたデ―タのみき

もなは、 `SerialPort.Read`および`SerialPort.Write`メソッドをすることです。ただし、SerialPortでデ―タをストリ―ミングするためにできる`System.IO.Stream`オブジェクトをすることもできます。これをうには、 `SerialPort.BaseStream`します。

```
int length = port.BytesToRead;
//Note that you can swap out a byte-array for a char-array if you prefer.
byte[] buffer = new byte[length];
port.Read(buffer, 0, length);
```

なすべてのデ―タをむこともできます

```
string curData = port.ReadExisting();
```

あるいは、ってくるデ―タでにつかったをむだけです

```
string line = port.ReadLine();
```

きみ

SerialPortでデ―タをきむもなはのとおりです。

```
port.Write("here is some text to be sent over the serial port.");
```

ただし、にじてのようにデ―タをすることもできます。

```
//Note that you can swap out the byte-array with a char-array if you so choose.
byte[] data = new byte[1] { 255 };
port.Write(data, 0, data.Length);
```

オンラインでSystem.IOをむ https://riptutorial.com/ja/dot-net/topic/5259/system-io

22: System.IO.Fileクラス

- ソース。
- の。

パラメーター

パラメータ	
source	のにするファイル。
destination	sourceをするディレクトリこのには、ファイルのおよびファイルもめるがあります。

Examples

ファイルをする

ファイルをするにはながあるのようにです

```
File.Delete(path);
```

しかし、くのことがっているかもしれません

- ながありません UnauthorizedAccessExceptionがスローされます。
- ファイルがのによってされているがあります IOExceptionがスローされます。
- レベルのエラーまたはメディアがみりのためにファイルをできません IOExceptionがスローされます。
- ファイルはもうしません IOExceptionがスローされます。

のポイントファイルはしないは、のようなコードスニペットでされることにしてください。

```
if (File.Exists(path))
    File.Delete(path);
```

しかし、それはへのびしのにのかによってすることができるアトミックとファイルではありませんFile.Exists()とFile.Delete() I / Oをするためのなアプローチには、がですがしたにのでをうことがです。

```
if (File.Exists(path))
{
    try
```

```
    {
        File.Delete(path);
    }
    catch (IOException exception)
    {
        if (!File.Exists(path))
            return; // Someone else deleted this file

        // Something went wrong...
    }
    catch (UnauthorizedAccessException exception)
    {
        // I do not have required permissions
    }
}
```

このI / Oエラーは、なものたとえば、ファイルがであり、ネットワークがしている、たちのからのもなしににするがあることにしてください。に、のにわずかなをいながら、I / Oをすることがです。

```
public static void Delete(string path)
{
    if (!File.Exists(path))
        return;

    for (int i=1; ; ++i)
    {
        try
        {
            File.Delete(path);
            return;
        }
        catch (IOException e)
        {
            if (!File.Exists(path))
                return;

            if (i == NumberOfAttempts)
                throw;

            Thread.Sleep(DelayBetweenEachAttempt);
        }

        // You may handle UnauthorizedAccessException but this issue
        // will probably won't be fixed in few seconds...
    }
}

private const int NumberOfAttempts = 3;
private const int DelayBetweenEachAttempt = 1000; // ms
```

Windowsファイルでは、このをびすときにははされません。のが`FileShare.Delete`をしてファイルをくと、ファイルをできますが、がファイルをじるときにのみです。

テキストファイルからなをりく

テキストファイルをするには、コンテンツをするがあるため、ではありません。 さなファイルの、もなは、そのをメモリにみんだ、されたテキストをきすことです。

このでは、ファイルからすべてのをみみ、すべてのをしてのパスにきします。

```
File.WriteAllLines(path,
    File.ReadAllLines(path).Where(x => !String.IsNullOrWhiteSpace(x)));
```

ファイルがきすぎてメモリにロードできず、パスがパスとなる

```
File.WriteAllLines(outputPath,
    File.ReadLines(inputPath).Where(x => !String.IsNullOrWhiteSpace(x)));
```

テキストファイルのをする

テキストはエンコードされてされます のトピックもしてください。に、エンコーディングをするがあるがあります。このでは、ファイルがきすぎず、

```
public static void ConvertEncoding(string path, Encoding from, Encoding to)
{
    File.WriteAllText(path, File.ReadAllText(path, from), to);
}
```

をする、ファイルにBOMByte Order Markがまれているがあることをれないでください。 Encoding.UTF8.GetStringは、Preamble / BOMをしていません 。

のファイルを「タッチ」するのきみをする

このは、してファイルのなのきみ`System.IO.Directory.EnumerateFiles`わりの `System.IO.Directory.GetFiles()`オプションでパターンをすることができますデフォルトは`"*.*"`にディレクトリツリーしたディレクトリだけでなくをします。

```
public static void Touch(string path,
                         string searchPattern = "*.*",
                         SearchOptions options = SearchOptions.None)
{
    var now = DateTime.Now;

    foreach (var filePath in Directory.EnumerateFiles(path, searchPattern, options))
    {
        File.SetLastWriteTime(filePath, now);
    }
}
```

されたよりいファイルをする

このスニペットは、されたよりいすべてのファイルをするヘルパーです。たとえば、いログファイルやいキャッシュデータをするがあるなどにです。

```
static IEnumerable<string> EnumerateAllFilesOlderThan(
                                TimeSpan maximumAge,
                                string path,
                                string searchPattern = "*.*",
                                SearchOption options = SearchOption.TopDirectoryOnly)
{
    DateTime oldestWriteTime = DateTime.Now - maximumAge;

    return Directory.EnumerateFiles(path, searchPattern, options)
        .Where(x => Directory.GetLastWriteTime(x) < oldestWriteTime);
}
```

このようにされます

```
var oldFiles = EnumerateAllFilesOlderThan(TimeSpan.FromDays(7), @"c:\log", "*.log");
```

すべきことはほとんどありません。

- は、 `Directory.EnumerateFiles()`わりに`Directory.GetFiles()`をしてされます。エニュメレーションはきているので、すべてのファイルシステムエントリがフェッチされるまでつはありません。
- のきみをしていますが、やアクセスをすることがありますたとえば、 のキャッシュファイルをするは、アクセスがになるがあることにしてください。
- はすべてのプロパティきみ、アクセス、でではなく、MSDNでこれについてのをしてください。

あるからのにファイルをする

File.Move

あるからのにファイルをするには、なコードでこれをできます。

```
File.Move(@"C:\TemporaryFile.txt", @"C:\TemporaryFiles\TemporaryFile.txt");
```

しかし、このなではっていることがくあります。たとえば、プログラムをしているユーザーに 'C' というラベルのドライブがないはどうなりますからがした - らはそれを 'B'または 'M'にをすることにめましたか

ソースファイルしたいファイルがあなたのらないうちにされた、またはにしないはどうなりますか

これは、まずソースファイルがするかどうかをすることによってできます。

```
string source = @"C:\TemporaryFile.txt", destination = @"C:\TemporaryFiles\TemporaryFile.txt";
if(File.Exists("C:\TemporaryFile.txt"))
{
    File.Move(source, destination);
}
```

これは、そのでファイルがし、のにできることをします。 `File.Exists`へのなびしでは`File.Exists`ながあります。そうでないは、し、がしたことをユーザーにえます。またはをします。

`FileNotFoundException`はするのあるのではありません。

えられるについてはをしてください。

タイプ	
`IOException`	ファイルがすでにするか、ソースファイルがつかりませんでした。
`ArgumentNullException`	SourceおよびまたはDestinationパラメータのはnullです。
`ArgumentException`	Sourceパラメーターまたは Destinationパラメーターのがであるか、ながまれています。
`UnauthorizedAccessException`	このをするためにながありません。
`PathTooLongException`	ソース、、またはされたパスがをえています。 Windowsでは、パスのさは248でなければならず、ファイルは260でなければなりません。
`DirectoryNotFoundException`	されたディレクトリがつかりませんでした。
`NotSupportedException`	SourceまたはDestinationパスまたはファイルのがです。

オンラインでSystem.IO.Fileクラスをむ https://riptutorial.com/ja/dot-net/topic/5395/system-io-file-クラス

23: System.Net.Mail

System.Net.MailMessageをすることはです。すべてのファイルにStreamがまれているため、これらのStreamsをできるだけくするがあるためです。 usingステートメントは、のでもDisposableオブジェクトがDisposedになることをします

Examples

MailMessage

ファイルきのメールメッセージをするをにします。、 `SmtpClient`クラスのけをりてこのメッセージをします。デフォルトの25ポートがここでされます。

```
public class clsMail
{
    private static bool SendMail(string mailfrom, List<string>replytos, List<string> mailtos,
List<string> mailccs, List<string> mailbccs, string body, string subject, List<string>
Attachment)
    {
        try
        {
            using(MailMessage MyMail = new MailMessage())
            {
                MyMail.From = new MailAddress(mailfrom);
                foreach (string mailto in mailtos)
                    MyMail.To.Add(mailto);

                if (replytos != null && replytos.Any())
                {
                    foreach (string replyto in replytos)
                        MyMail.ReplyToList.Add(replyto);
                }

                if (mailccs != null && mailccs.Any())
                {
                    foreach (string mailcc in mailccs)
                        MyMail.CC.Add(mailcc);
                }

                if (mailbccs != null && mailbccs.Any())
                {
                    foreach (string mailbcc in mailbccs)
                        MyMail.Bcc.Add(mailbcc);
                }

                MyMail.Subject = subject;
                MyMail.IsBodyHtml = true;
                MyMail.Body = body;
                MyMail.Priority = MailPriority.Normal;

                if (Attachment != null && Attachment.Any())
                {
                    System.Net.Mail.Attachment attachment;
                    foreach (var item in Attachment)
```

```
                    {
                        attachment = new System.Net.Mail.Attachment(item);
                        MyMail.Attachments.Add(attachment);
                    }
                }

                SmtpClient smtpMailObj = new SmtpClient();
                smtpMailObj.Host = "your host";
                smtpMailObj.Port = 25;
                smtpMailObj.Credentials = new System.Net.NetworkCredential("uid", "pwd");

                smtpMailObj.Send(MyMail);
                return true;
            }
        }
        catch
        {
            return false;
        }
    }
}
```

ファイルきメール

`MailMessage`は、 `SmtpClient`クラスをしてさらにできるメールメッセージをします。いくつかのファイルファイルをメールメッセージにすることができます。

```
using System.Net.Mail;

using(MailMessage myMail = new MailMessage())
{
     Attachment attachment = new Attachment(path);
     myMail.Attachments.Add(attachment);

     // further processing to send the mail message

}
```

オンラインでSystem.Net.Mailをむ https://riptutorial.com/ja/dot-net/topic/7440/system-net-mail

24: System.Reflection.Emit

Examples

にアセンブリをする

```
using System;
using System.Reflection;
using System.Reflection.Emit;

class DemoAssemblyBuilder
{
    public static void Main()
    {
        // An assembly consists of one or more modules, each of which
        // contains zero or more types. This code creates a single-module
        // assembly, the most common case. The module contains one type,
        // named "MyDynamicType", that has a private field, a property
        // that gets and sets the private field, constructors that
        // initialize the private field, and a method that multiplies
        // a user-supplied number by the private field value and returns
        // the result. In C# the type might look like this:
        /*
        public class MyDynamicType
        {
            private int m_number;

            public MyDynamicType() : this(42) {}
            public MyDynamicType(int initNumber)
            {
                m_number = initNumber;
            }

            public int Number
            {
                get { return m_number; }
                set { m_number = value; }
            }

            public int MyMethod(int multiplier)
            {
                return m_number * multiplier;
            }
        }
        */

        AssemblyName aName = new AssemblyName("DynamicAssemblyExample");
        AssemblyBuilder ab =
            AppDomain.CurrentDomain.DefineDynamicAssembly(
                aName,
                AssemblyBuilderAccess.RunAndSave);

        // For a single-module assembly, the module name is usually
        // the assembly name plus an extension.
        ModuleBuilder mb =
            ab.DefineDynamicModule(aName.Name, aName.Name + ".dll");
```

```
        TypeBuilder tb = mb.DefineType(
            "MyDynamicType",
             TypeAttributes.Public);

        // Add a private field of type int (Int32).
        FieldBuilder fbNumber = tb.DefineField(
            "m_number",
            typeof(int),
            FieldAttributes.Private);

        // Next, we make a simple sealed method.
        MethodBuilder mbMyMethod = tb.DefineMethod(
            "MyMethod",
            MethodAttributes.Public,
            typeof(int),
            new[] { typeof(int) });

        ILGenerator il = mbMyMethod.GetILGenerator();
        il.Emit(OpCodes.Ldarg_0); // Load this - always the first argument of any instance
method
        il.Emit(OpCodes.Ldfld, fbNumber);
        il.Emit(OpCodes.Ldarg_1); // Load the integer argument
        il.Emit(OpCodes.Mul); // Multiply the two numbers with no overflow checking
        il.Emit(OpCodes.Ret); // Return

        // Next, we build the property. This involves building the property itself, as well as
the
        // getter and setter methods.
        PropertyBuilder pbNumber = tb.DefineProperty(
            "Number", // Name
            PropertyAttributes.None,
            typeof(int), // Type of the property
            new Type[0]); // Types of indices, if any

        MethodBuilder mbSetNumber = tb.DefineMethod(
            "set_Number", // Name - setters are set_Property by convention
            // Setter is a special method and we don't want it to appear to callers from C#
            MethodAttributes.PrivateScope | MethodAttributes.HideBySig |
MethodAttributes.Public | MethodAttributes.SpecialName,
            typeof(void), // Setters don't return a value
            new[] { typeof(int) }); // We have a single argument of type System.Int32

        // To generate the body of the method, we'll need an IL generator
        il = mbSetNumber.GetILGenerator();
        il.Emit(OpCodes.Ldarg_0); // Load this
        il.Emit(OpCodes.Ldarg_1); // Load the new value
        il.Emit(OpCodes.Stfld, fbNumber); // Save the new value to this.m_number
        il.Emit(OpCodes.Ret); // Return

        // Finally, link the method to the setter of our property
        pbNumber.SetSetMethod(mbSetNumber);

        MethodBuilder mbGetNumber = tb.DefineMethod(
            "get_Number",
            MethodAttributes.PrivateScope | MethodAttributes.HideBySig |
MethodAttributes.Public | MethodAttributes.SpecialName,
            typeof(int),
            new Type[0]);

        il = mbGetNumber.GetILGenerator();
```

```
        il.Emit(OpCodes.Ldarg_0); // Load this
        il.Emit(OpCodes.Ldfld, fbNumber); // Load the value of this.m_number
        il.Emit(OpCodes.Ret); // Return the value

        pbNumber.SetGetMethod(mbGetNumber);

        // Finally, we add the two constructors.
        // Constructor needs to call the constructor of the parent class, or another
constructor in the same class
        ConstructorBuilder intConstructor = tb.DefineConstructor(
            MethodAttributes.Public, CallingConventions.Standard | CallingConventions.HasThis,
new[] { typeof(int) });
        il = intConstructor.GetILGenerator();
        il.Emit(OpCodes.Ldarg_0); // this
        il.Emit(OpCodes.Call, typeof(object).GetConstructor(new Type[0])); // call parent's
constructor
        il.Emit(OpCodes.Ldarg_0); // this
        il.Emit(OpCodes.Ldarg_1); // our int argument
        il.Emit(OpCodes.Stfld, fbNumber); // store argument in this.m_number
        il.Emit(OpCodes.Ret);

        var parameterlessConstructor = tb.DefineConstructor(
            MethodAttributes.Public, CallingConventions.Standard | CallingConventions.HasThis,
new Type[0]);
        il = parameterlessConstructor.GetILGenerator();
        il.Emit(OpCodes.Ldarg_0); // this
        il.Emit(OpCodes.Ldc_I4_S, (byte)42); // load 42 as an integer constant
        il.Emit(OpCodes.Call, intConstructor); // call this(42)
        il.Emit(OpCodes.Ret);

        // And make sure the type is created
        Type ourType = tb.CreateType();

        // The types from the assembly can be used directly using reflection, or we can save
the assembly to use as a reference
        object ourInstance = Activator.CreateInstance(ourType);
        Console.WriteLine(ourType.GetProperty("Number").GetValue(ourInstance)); // 42

        // Save the assembly for use elsewhere. This is very useful for debugging - you can
use e.g. ILSpy to look at the equivalent IL/C# code.
        ab.Save(@"DynamicAssemblyExample.dll");
        // Using newly created type
        var myDynamicType = tb.CreateType();
        var myDynamicTypeInstance = Activator.CreateInstance(myDynamicType);

        Console.WriteLine(myDynamicTypeInstance.GetType()); // MyDynamicType

        var numberField = myDynamicType.GetField("m_number", BindingFlags.NonPublic |
BindingFlags.Instance);
        numberField.SetValue (myDynamicTypeInstance, 10);

        Console.WriteLine(numberField.GetValue(myDynamicTypeInstance)); // 10
    }
}
```

オンラインでSystem.Reflection.Emitをむ https://riptutorial.com/ja/dot-net/topic/74/system-reflection-emit

25: System.Runtime.Caching.MemoryCache ObjectCache

Examples

キャッシュへのの

Setは、CacheItemインスタンスをしてキャッシュエントリのキーとをすることにより、キャッシュエントリをキャッシュにします。

このは`ObjectCache.Set(CacheItem, CacheItemPolicy)`オーバーライドします。

```
private static bool SetToCache()
{
    string key = "Cache_Key";
    string value = "Cache_Value";

    //Get a reference to the default MemoryCache instance.
    var cacheContainer = MemoryCache.Default;

    var policy = new CacheItemPolicy()
    {
        AbsoluteExpiration = DateTimeOffset.Now.AddMinutes(DEFAULT_CACHE_EXPIRATION_MINUTES)
     };
     var itemToCache = new CacheItem(key, value); //Value is of type object.
     cacheContainer.Set(itemToCache, policy);
}
```

System.Runtime.Caching.MemoryCacheObjectCache

このはのアイテムフォームキャッシュをし、アイテムがキャッシュにしないは、valueFetchFactoryについてアイテムをします。

```
    public static TValue GetExistingOrAdd<TValue>(string key, double minutesForExpiration,
Func<TValue> valueFetchFactory)
    {
        try
        {
            //The Lazy class provides Lazy initialization which will evaluate
            //the valueFetchFactory only if item is not in the cache.
            var newValue = new Lazy<TValue>(valueFetchFactory);

            //Setup the cache policy if item will be saved back to cache.
            CacheItemPolicy policy = new CacheItemPolicy()
            {
                AbsoluteExpiration = DateTimeOffset.Now.AddMinutes(minutesForExpiration)
            };

            //returns existing item form cache or add the new value if it does not exist.
            var cachedItem = _cacheContainer.AddOrGetExisting(key, newValue, policy) as
Lazy<TValue>;
```

```
            return (cachedItem ?? newValue).Value;
        }
        catch (Exception excep)
        {
            return default(TValue);
        }
    }
```

オンラインでSystem.Runtime.Caching.MemoryCacheObjectCacheをむ
https://riptutorial.com/ja/dot-net/topic/76/system-runtime-caching-memorycache-objectcache-

26: TPLデータフロー

できるライブラリ

`System.Threading.Tasks.Dataflow`

`System.Threading.Tasks`

`System.Net.Http`

`System.Net`

PostとSendAsyncのい

ブロックにアイテムをするには、 `Post`または`SendAsync`します。

`Post`はしてアイテムをしようとし、したかどうかをす`bool`をします。たとえば、ブロックが`BoundedCapcity`したときに、しいのがまだないは、しないがあります。、 `SendAsync`はあなたが`await`ことができるの`Task<bool>`をします。そのタスクは、ブロックがキューをクリアし、にえば、キャンセルのとしてしている、よりくのアイテムをけれるか、または`false`をけれることができる`true`をって、します。

Examples

ActionBlockへのとをつ

```
// Create a block with an asynchronous action
var block = new ActionBlock<string>(async hostName =>
{
    IPAddress[] ipAddresses = await Dns.GetHostAddressesAsync(hostName);
    Console.WriteLine(ipAddresses[0]);
});

block.Post("google.com"); // Post items to the block's InputQueue for processing
block.Post("reddit.com");
block.Post("stackoverflow.com");

block.Complete(); // Tell the block to complete and stop accepting new items
await block.Completion; // Asynchronously wait until all items completed processingu
```

ブロックをリンクしてパイプラインをする

```
var httpClient = new HttpClient();

// Create a block the accepts a uri and returns its contents as a string
var downloaderBlock = new TransformBlock<string, string>(
    async uri => await httpClient.GetStringAsync(uri));
```

```
// Create a block that accepts the content and prints it to the console
var printerBlock = new ActionBlock<string>(
    contents => Console.WriteLine(contents));

// Make the downloaderBlock complete the printerBlock when its completed.
var dataflowLinkOptions = new DataflowLinkOptions {PropagateCompletion = true};

// Link the block to create a pipeline
downloaderBlock.LinkTo(printerBlock, dataflowLinkOptions);

// Post urls to the first block which will pass their contents to the second one.
downloaderBlock.Post("http://youtube.com");
downloaderBlock.Post("http://github.com");
downloaderBlock.Post("http://twitter.com");

downloaderBlock.Complete(); // Completion will propagate to printerBlock
await printerBlock.Completion; // Only need to wait for the last block in the pipeline
```

BufferBlockをしたプロデューサ/コンシューマ

```
public class Producer
{
    private static Random random = new Random((int)DateTime.UtcNow.Ticks);
    //produce the value that will be posted to buffer block
    public double Produce ( )
    {
        var value = random.NextDouble();
        Console.WriteLine($"Producing value: {value}");
        return value;
    }
}

public class Consumer
{
    //consume the value that will be received from buffer block
    public void Consume (double value) => Console.WriteLine($"Consuming value: {value}");
}

class Program
{
    private static BufferBlock<double> buffer = new BufferBlock<double>();
    static void Main (string[] args)
    {
        //start a task that will every 1 second post a value from the producer to buffer block
        var producerTask = Task.Run(async () =>
        {
            var producer = new Producer();
            while(true)
            {
                buffer.Post(producer.Produce());
                await Task.Delay(1000);
            }
        });
        //start a task that will recieve values from bufferblock and consume it
        var consumerTask = Task.Run(() =>
        {
            var consumer = new Consumer();
            while(true)
```

```
            {
                consumer.Consume(buffer.Receive());
            }
        });

        Task.WaitAll(new[] { producerTask, consumerTask });
    }
}
```

バインドされた**BufferBlock**をつプロデューサコンシューマ

```
var bufferBlock = new BufferBlock<int>(new DataflowBlockOptions
{
    BoundedCapacity = 1000
});

var cancellationToken = new CancellationTokenSource(TimeSpan.FromSeconds(10)).Token;

var producerTask = Task.Run(async () =>
{
    var random = new Random();

    while (!cancellationToken.IsCancellationRequested)
    {
        var value = random.Next();
        await bufferBlock.SendAsync(value, cancellationToken);
    }
});

var consumerTask = Task.Run(async () =>
{
    while (await bufferBlock.OutputAvailableAsync())
    {
        var value = bufferBlock.Receive();
        Console.WriteLine(value);
    }
});

await Task.WhenAll(producerTask, consumerTask);
```

オンラインでTPLデータフローをむ https://riptutorial.com/ja/dot-net/topic/784/tplデータフロー

27: VBフォーム

Examples

VB.NETフォームのHello World

フォームがされているときにメッセージボックスをするには

```
Public Class Form1
    Private Sub Form1_Shown(sender As Object, e As EventArgs) Handles MyBase.Shown
        MessageBox.Show("Hello, World!")
    End Sub
End Class
To show a message box before the form has been shown:

Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        MessageBox.Show("Hello, World!")
    End Sub
End Class
```

Loadはにびされ、フォームがにロードされるとだけびされます。 Showは、ユーザーがフォームをするたびにびされます。 Activateは、ユーザーがフォームをアクティブにするたびにびされます。

Showがびされるにloadがされますが、showでmsgBoxをびすと、Loadがするにmsgboxがされるがあります。 **Load**、Showなどのイベントのけにすることは、にはいえです。

け

すべてのがっておくべきこと/ VBでのいスタートをつのについくつかのこと

のオプションをします。

```
'can be permanently set
' Tools / Options / Projects and Soluntions / VB Defaults
Option Strict On
Option Explicit On
Option Infer Off

Public Class Form1

End Class
```

のには+、+をしないでください。 ストリングは、くわれているので、かくべるがあります。

とのをしてください。

Application.DoEventsをしないでください。「」にしてください。あなたがうべきもののように

えるところにきたら、ねなさい。

ドキュメントはあなたのです。

フォームタイマー

Windows.Forms.Timerコンポーネントは、タイムクリティカルではないユーザーをするためにできます。 1つのボタン、1つのラベル、およびタイマーコンポーネントでフォームをします。

たとえば、ユーザーににをすためにできます。

```
'can be permanently set
' Tools / Options / Projects and Soluntions / VB Defaults
Option Strict On
Option Explicit On
Option Infer Off

Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Button1.Enabled = False
        Timer1.Interval = 60 * 1000 'one minute intervals
        'start timer
        Timer1.Start()
        Label1.Text = DateTime.Now.ToLongTimeString
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
        Label1.Text = DateTime.Now.ToLongTimeString
    End Sub
End Class
```

しかし、このタイマーはタイミングにはしていません。は、カウントダウンのためにそれをしています。このでは、カウントダウンを3にシミュレートします。これはここでもなの1つになるかもしれません。

```
'can be permanently set
' Tools / Options / Projects and Soluntions / VB Defaults
Option Strict On
Option Explicit On
Option Infer Off

Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Button1.Enabled = False
        ctSecs = 0 'clear count
        Timer1.Interval = 1000 'one second in ms.
        'start timers
        stpw.Reset()
        stpw.Start()
        Timer1.Start()
    End Sub

    Dim stpw As New Stopwatch
```

```
    Dim ctSecs As Integer

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
        ctSecs += 1
        If ctSecs = 180 Then 'about 2.5 seconds off on my PC!
            'stop timing
            stpw.Stop()
            Timer1.Stop()
            'show actual elapsed time
            'Is it near 180?
            Label1.Text = stpw.Elapsed.TotalSeconds.ToString("n1")
        End If
    End Sub
End Class
```

button1をクリックすると3がし、label1がをします。 label1には180がされますかおそらくそうではありません。のマシンでは182.5をしました

のは、「Windowsフォームタイマコンポーネントはシングルスレッドで、は55ミリにされています」というドキュメントにあります。これがタイミングのためにわれるべきでないです。

タイマーとストップウォッチをしずついければ、よりいがられます。

```
'can be permanently set
' Tools / Options / Projects and Soluntions / VB Defaults
Option Strict On
Option Explicit On
Option Infer Off

Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Button1.Enabled = False
        Timer1.Interval = 100 'one tenth of a second in ms.
        'start timers
        stpw.Reset()
        stpw.Start()
        Timer1.Start()
    End Sub

    Dim stpw As New Stopwatch
    Dim threeMinutes As TimeSpan = TimeSpan.FromMinutes(3)

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
        If stpw.Elapsed >= threeMinutes Then '0.1 off on my PC!
            'stop timing
            stpw.Stop()
            Timer1.Stop()
            'show actual elapsed time
            'how close?
            Label1.Text = stpw.Elapsed.TotalSeconds.ToString("n1")
        End If
    End Sub
End Class
```

にじてできるのタイマーがあります。このは、そのでつはずです。

オンラインでVBフォームをむ https://riptutorial.com/ja/dot-net/topic/2197/vbフォーム

28: XmlSerializer

`HTML`をするために`XmlSerializer`をしないでください。このために、なライブラリはHTML Agility Pack

Examples

オブジェクトのシリアル

```
public void SerializeFoo(string fileName, Foo foo)
{
    var serializer = new XmlSerializer(typeof(Foo));
    using (var stream = File.Open(fileName, FileMode.Create))
    {
        serializer.Serialize(stream, foo);
    }
}
```

オブジェクトをする

```
public Foo DeserializeFoo(string fileName)
{
    var serializer = new XmlSerializer(typeof(Foo));
    using (var stream = File.OpenRead(fileName))
    {
        return (Foo)serializer.Deserialize(stream);
    }
}
```

をプロパティにマップする

```
<Foo>
    <Dog/>
</Foo>
```

-

```
public class Foo
{
    // Using XmlElement
    [XmlElement(Name="Dog")]
    public Animal Cat { get; set; }
}
```

をプロパティにマップするXmlArray

```
<Store>
    <Articles>
```

```
        <Product/>
        <Product/>
    </Articles>
</Store>
```

-

```
public class Store
{
    [XmlArray("Articles")]
    public List<Product> Products {get; set; }
}
```

フォーマットカスタム**DateTime**フォーマット

```
public class Dog
{
    private const string _birthStringFormat = "yyyy-MM-dd";

    [XmlIgnore]
    public DateTime Birth {get; set;}

    [XmlElement(ElementName="Birth")]
    public string BirthString
    {
        get { return Birth.ToString(_birthStringFormat); }
        set { Birth = DateTime.ParseExact(value, _birthStringFormat,
CultureInfo.InvariantCulture); }
    }
}
```

にされたをつのシリアライザをにする

たちがたところ

によっては、XmlSerializerフレームワークになすべてのメタデータをでることはできません。シリアライズされたオブジェクトのクラスがあり、クラスのがクラスでであるとします。のにられていないすべてのクラスにしてをすることはできません。クラスのいくつかをするのチームができました。

たちはができる

たちは`new XmlSerializer(type, knownTypes)`をうことができますが、なくともNのシリアライザのO N ^ 2であり、なくともargumentsでされたすべてをするがあります。

```
// Beware of the N^2 in terms of the number of types.
var allSerializers = allTypes.Select(t => new XmlSerializer(t, allTypes));
var serializerDictionary = Enumerable.Range(0, allTypes.Length)
    .ToDictionary (i => allTypes[i], i => allSerializers[i])
```

このでは、はOOPでのをしません。

にう

いにも、こののにするがあります。これは、のシリアライザののをにします。

System.Xml.Serialization.XmlSerializer.FromTypesメソッド[]

> FromTypesメソッドをすると、Typeオブジェクトのをするためのオブジェクトのをにできます。XmlSerializer

```
var allSerializers = XmlSerializer.FromTypes(allTypes);
var serializerDictionary = Enumerable.Range(0, allTypes.Length)
    .ToDictionary(i => allTypes[i], i => allSerializers[i]);
```

ここになコードサンプルがあります

```
using System;
using System.Collections.Generic;
using System.Xml.Serialization;
using System.Linq;
using System.Linq;

public class Program
{
    public class Container
    {
        public Base Base { get; set; }
    }

    public class Base
    {
        public int JustSomePropInBase { get; set; }
    }

    public class Derived : Base
    {
        public int JustSomePropInDerived { get; set; }
    }

    public void Main()
    {
        var sampleObject = new Container { Base = new Derived() };
        var allTypes = new[] { typeof(Container), typeof(Base), typeof(Derived) };

        Console.WriteLine("Trying to serialize without a derived class metadata:");
        SetupSerializers(allTypes.Except(new[] { typeof(Derived) }).ToArray());
        try
        {
            Serialize(sampleObject);
        }
        catch (InvalidOperationException e)
        {
            Console.WriteLine();
            Console.WriteLine("This error was anticipated,");
            Console.WriteLine("we have not supplied a derived class.");
```

```
            Console.WriteLine(e);
        }
        Console.WriteLine("Now trying to serialize with all of the type information:");
        SetupSerializers(allTypes);
        Serialize(sampleObject);
        Console.WriteLine();
        Console.WriteLine("Slides down well this time!");
    }

    static void Serialize<T>(T o)
    {
        serializerDictionary[typeof(T)].Serialize(Console.Out, o);
    }

    private static Dictionary<Type, XmlSerializer> serializerDictionary;

    static void SetupSerializers(Type[] allTypes)
    {
        var allSerializers = XmlSerializer.FromTypes(allTypes);
        serializerDictionary = Enumerable.Range(0, allTypes.Length)
            .ToDictionary(i => allTypes[i], i => allSerializers[i]);
    }
}
```

```
Trying to serialize without a derived class metadata:
<?xml version="1.0" encoding="utf-16"?>
<Container xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
This error was anticipated,
we have not supplied a derived class.
System.InvalidOperationException: There was an error generating the XML document. --->
System.InvalidOperationException: The type Program+Derived was not expected. Use the
XmlInclude or SoapInclude attribute to specify types that are not known statically.
   at Microsoft.Xml.Serialization.GeneratedAssembly.XmlSerializationWriter1.Write2_Base(String
n, String ns, Base o, Boolean isNullable, Boolean needType)
   at
Microsoft.Xml.Serialization.GeneratedAssembly.XmlSerializationWriter1.Write3_Container(String
n, String ns, Container o, Boolean isNullable, Boolean needType)
   at
Microsoft.Xml.Serialization.GeneratedAssembly.XmlSerializationWriter1.Write4_Container(Object
o)
   at System.Xml.Serialization.XmlSerializer.Serialize(XmlWriter xmlWriter, Object o,
XmlSerializerNamespaces namespaces, String encodingStyle, String id)
   --- End of inner exception stack trace ---
   at System.Xml.Serialization.XmlSerializer.Serialize(XmlWriter xmlWriter, Object o,
XmlSerializerNamespaces namespaces, String encodingStyle, String id)
   at System.Xml.Serialization.XmlSerializer.Serialize(XmlWriter xmlWriter, Object o,
XmlSerializerNamespaces namespaces, String encodingStyle)
   at System.Xml.Serialization.XmlSerializer.Serialize(XmlWriter xmlWriter, Object o,
XmlSerializerNamespaces namespaces)
   at Program.Serialize[T](T o)
   at Program.Main()
Now trying to serialize with all of the type information:
<?xml version="1.0" encoding="utf-16"?>
<Container xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Base xsi:type="Derived">
    <JustSomePropInBase>0</JustSomePropInBase>
    <JustSomePropInDerived>0</JustSomePropInDerived>
  </Base>
```

```
</Container>
Slides down well this time!
```

にまれるもの

このエラーメッセージは、たちがけようとしたことまたはいくつかのシナリオではできないものをおめします。ベースクラスからしたをする

```
Use the XmlInclude or SoapInclude attribute to specify types that are not known statically.
```

これはXMLでクラスをするです。

```
<Base xsi:type="Derived">
```

`Base`は`Container`タイプでされたプロパティタイプにし、 `Derived`はにされたインスタンスのタイプである。

ここではなフィドルです

オンラインでXmlSerializerをむ https://riptutorial.com/ja/dot-net/topic/31/xmlserializer

29: Zipファイルのみき

き

ZipFileクラスは、 **System.IO.Compression**にします。これは、Zipファイルのみきにできます。

- FileStreamのわりにMemoryStreamをすることもできます。
-

ArgumentException	ストリームがすでにじられているか、またはストリームのがモードにしていませんたとえば、みりストリームにきもうとしています
ArgumentNullException	ストリームがnull
ArgumentOutOfRangeException	モードのがです
InvalidDataException	のリストを

InvalidDataExceptionがスローされると、3つのがあります。

- ストリームのをzipアーカイブとしてできませんでした
- モードがされ、エントリがアーカイブにないか、してみめません
- モードがUpdateであり、エントリがきすぎてメモリにまらない

すべてのはこのMSDNページからされています

Examples

ZIPコンテンツのリスト

このスニペットには、zipアーカイブのすべてのファイルがリストされます。ファイルはジップルートをにしています。

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))
{
    for (int i = 0; i < archive.Entries.Count; i++)
    {
        Console.WriteLine($"{i}: {archive.Entries[i]}");
    }
}
```

ZIPファイルからのファイルの

すべてのファイルをディレクトリにするのはとてもです

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))
{
    archive.ExtractToDirectory(AppDomain.CurrentDomain.BaseDirectory);
}
```

ファイルがすでにするは、 **System.IO.IOException**がスロ―されます。

のファイルをする

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))
{
    // Get a root entry file
    archive.GetEntry("test.txt").ExtractToFile("test_extracted_getentries.txt", true);

    // Enter a path if you want to extract files from a subdirectory
    archive.GetEntry("sub/subtest.txt").ExtractToFile("test_sub.txt", true);

    // You can also use the Entries property to find files
    archive.Entries.FirstOrDefault(f => f.Name ==
"test.txt")?.ExtractToFile("test_extracted_linq.txt", true);

    // This will throw a System.ArgumentNullException because the file cannot be found
    archive.GetEntry("nonexistingfile.txt").ExtractToFile("fail.txt", true);
}
```

これらのメソッドのいずれもじをします。

ZIPファイルの

ZIPファイルをするには、そのファイルをZipArchiveMode.Updateでくがあります。

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Update))
{
    // Add file to root
    archive.CreateEntryFromFile("test.txt", "test.txt");

    // Add file to subfolder
    archive.CreateEntryFromFile("test.txt", "symbols/test.txt");
}
```

また、アーカイブのファイルにきむこともできます。

```
var entry = archive.CreateEntry("createentry.txt");
using(var writer = new StreamWriter(entry.Open()))
{
    writer.WriteLine("Test line");
```

```
}
```

オンラインでZipファイルのみきをむ https://riptutorial.com/ja/dot-net/topic/9943/zipファイルのみき

30: カスタムタイプ

、 `struct`は、パフォーマンスがになにのみされます。はスタックにするため、クラスよりもはるかににアクセスできます。ただし、スタックは、ヒープよりもはるかにないをっているので、はさくされなければならないMicrosoftがしています`struct` sがこれ16のバイトをります。

`class`はCでもっともされているこれらの3つの`class` で、にはにうべきことです。

`enum`は、コンパイルにだけするがある、にされたのリストをつことができるときはいつでもされます。は、いくつかのにとしてプログラマにちます。わりのリストの`constant`とするを、あなたはをして、あなたはったをしていないことをするインテリセンスサポートをけることができます。

Examples

StructsはSystem.ValueTypeからし、のであり、スタックにします。がパラメータとしてされると、しされます。

```
Struct MyStruct
{
    public int x;
    public int y;
}
```

しとは、メソッドのパラメータのがメソッドのためにコピーされ、メソッドのパラメータにえられたがメソッドのにされないことをします。たとえば、 `AddNumbers`というのメソッドをびし、Valueの`int`の`a`と`b`をすのコードをえます。

```
int a = 5;
int b = 6;

AddNumbers(a,b);

public AddNumbers(int x, int y)
{
    int z = x + y; // z becomes 11
    x = x + 5; // now we changed x to be 10
    z = x + y; // now z becomes 16
}
```

々は5をしたにもかかわらず`x`メソッドのそれはなので、されないまま、それはし`a` `x`のコピーだったのではなく、に。 `a` `a`

Valueはスタックにし、しであることをえておいてください。

クラス

クラスは**System.Object**からし、であり、ヒープにします。をパラメータとしてすと、がされます。

```
public Class MyClass
{
    public int a;
    public int b;
}
```

されたは、パラメータへのがメソッドにされ、がメモリのまったくじオブジェクトにしてわれるため、パラメータのがメソッドのにされることをします。とじをってみましょう。しかし、にクラスの`int`"ラップ"します。

```
MyClass instanceOfMyClass = new MyClass();
instanceOfMyClass.a = 5;
instanceOfMyClass.b = 6;

AddNumbers(instanceOfMyClass);

public AddNumbers(MyClass sample)
{
    int z = sample.a + sample.b; // z becomes 11
    sample.a = sample.a + 5; // now we changed a to be 10
    z = sample.a + sample.b; // now z becomes 16
}
```

は、 `sample.a`を`10`にしたときに`instanceOfMyClass.a`のもされました 。しとは、オブジェクトのコピーではなく、オブジェクトへの ポインタともばれるがメソッドにされたことをします。

はヒープにあり、しであることをえておいてください。

の

はなのクラスです。 **`enum`**キーワードは、このクラスが**System.Enum**クラスからすることをコンパイラにします。Enumはののリストにされます。

```
public enum MyEnum
{
    Monday = 1,
    Tuesday,
    Wednesday,
    //...
}
```

ををあるにマッピングするなとえることができます。でされたは、ごとにをし、でまる`1`。`Tuesday`はに`2`、 `Wednesday` `3`などにマップされます。

デフォルトでは、enumはになるとして`int`をし、0からしますが`byte, sbyte, short, ushort, int, uint, long, or ulong` をできます。。のがにされていても、にされていないがある、にされたのろのは1だけインクリメントされます。

たちはこのように、 *MyEnum*にのをキャストしてします

```
MyEnum instance = (MyEnum)3; // the variable named 'instance' gets a
                             //value of MyEnum.Wednesday, which maps to 3.

int x = 2;
instance = (MyEnum)x; // now 'instance' has a value of MyEnum.Tuesday
```

もうつのな、よりなのは、 `Flags`とばれ`Flags`。ることで`Flags`を、に、にのをりてることができます。これをうときは、2のでにをするがあることにしてください。

```
[Flags]
public enum MyEnum
{
    Monday = 1,
    Tuesday = 2,
    Wednesday = 4,
    Thursday = 8,
    Friday = 16,
    Saturday = 32,
    Sunday = 64
}
```

ビットごとのをするか、.NET 4.0をしているは、みみの`Enum.HasFlag`メソッドをして、にのをできます。

```
MyEnum instance = MyEnum.Monday | MyEnum.Thursday; // instance now has a value of
                                                   // *both* Monday and Thursday,
                                                   // represented by (in binary) 0100.

if (instance.HasFlag(MyEnum.Wednesday))
{
    // it doesn't, so this block is skipped
}
else if (instance.HasFlag(MyEnum.Thursday))
{
    // it does, so this block is executed
}
```

Enumクラスは`System.ValueType`からサブクラスされているため、としてわれ、しではなくしされます。ヒープにベースオブジェクトがされますが、enumをびしにすと、EnumのはSystem.Int32をするのコピーがスタックにプッシュされます。コンパイラは、このとスタックにされたオブジェクトとのけをします。については、 ValueTypeクラスシステムMSDNをしてください。

オンラインでカスタムタイプをむ https://riptutorial.com/ja/dot-net/topic/57/カスタムタイプ

31: ガベ―ジコレクション

き

.Netではnewでされたオブジェクトはマネ―ジヒ―プにりてられます。これらのオブジェクトは、それらをするプログラムによってにファイナライズされることはありません。わりに、このプロセスは.Netガベ―ジコレクタによってされます。

のは、ガ―ベッジ･コレクタ―がしていることをす「ラボ･ケ―ス」とそののなをしていますが、ガ―ベッジ･コレクタ―によるなのためのクラスのにをてています。

ガベ―ジコレクタは、りてられたメモリのからプログラムのコストをげることをとしていますが、そのためにはのでコストがかかります。なをするために、ガベ―ジコレクタをしてプログラミングするにするがあるいくつかのがあります。

- Collectメソッドをにびすは、メモリがになにのみデッドオブジェクトをさせる「」モ―ドのをしてください
- Collectメソッドをびすわりに、になにのみメモリコレクションをするAddMemoryPressureメソッドとRemoveMemoryPressureメソッドのをしてください
- メモリ―･コレクションは、すべてのデッド･オブジェクトのファイナライズをするものではありません。わりに、ガベ―ジコレクタは3つの「」をします。オブジェクトはからのに「きる」ことがあります
- ガベ―ジコレクタスレッドとのアプリケ―ションスレッドとののののいがなるとなる、セットアップのをむ々なにじて、いくつかのスレッドモデルがされるがあります。

Examples

ゴミの

のクラスをえます

```
public class FinalizableObject
{
    public FinalizableObject()
    {
        Console.WriteLine("Instance initialized");
    }

    ~FinalizableObject()
    {
        Console.WriteLine("Instance finalized");
    }
}
```

それをわなくても、インスタンスをするプログラム

```
new FinalizableObject(); // Object instantiated, ready to be used
```

のをします。

```
<namespace>.FinalizableObject initialized
```

もこらなければ、プログラムがするまでオブジェクトはファイナライズされませんされたヒープのすべてのオブジェクトをし、プロセスでこれらをさせます。

のように、ガベージコレクタをにのポイントですることができます。

```
new FinalizableObject(); // Object instantiated, ready to be used
GC.Collect();
```

のがられます。

```
<namespace>.FinalizableObject initialized
<namespace>.FinalizableObject finalized
```

は、ガベージコレクタがびされるとすぐに、「んでいる」オブジェクトがされ、されたヒープからされました。

ライブオブジェクトとデッドオブジェクト

まかなルールガベージコレクションがした、「ライブオブジェクト」はまだされているオブジェクトであり、「デッドオブジェクト」はされなくなったものですコレクションがするにスコープをしているまたはフィールドがあれば、。

の、FinalizableObject1とFinalizableObject2はののFinalizableObjectのサブクラスであるため、メッセージのをしています。

```
var obj1 = new FinalizableObject1(); // Finalizable1 instance allocated here
var obj2 = new FinalizableObject2(); // Finalizable2 instance allocated here
obj1 = null; // No more references to the Finalizable1 instance
GC.Collect();
```

はのようになります。

```
<namespace>.FinalizableObject1 initialized
<namespace>.FinalizableObject2 initialized
<namespace>.FinalizableObject1 finalized
```

ガベージコレクタがびされたで、FinalizableObject1はんでいるオブジェクトであり、FinalizableObject2はライブオブジェクトであり、マネージヒープにされます。

のんだオブジェクト

2つまたはのんだオブジェクトがいにしているはどうなりますかこれは、OtherObjectがFinalizableObjectのパブリックプロパティであるとして、のにされています。

```
var obj1 = new FinalizableObject1();
var obj2 = new FinalizableObject2();
obj1.OtherObject = obj2;
obj2.OtherObject = obj1;
obj1 = null; // Program no longer references Finalizable1 instance
obj2 = null; // Program no longer references Finalizable2 instance
// But the two objects still reference each other
GC.Collect();
```

これにより、のがされます。

```
<namespace>.FinalizedObject1 initialized
<namespace>.FinalizedObject2 initialized
<namespace>.FinalizedObject1 finalized
<namespace>.FinalizedObject2 finalized
```

2つのオブジェクトは、されているにもかかわらず、されたヒープからファイナライズされ、されますにするオブジェクトのいずれにものがしないため。

い

とは、のオブジェクト「ターゲット」ともばれますへのですが、それらのオブジェクトがガベージコレクションされないようにするための「い」ものです。いえれば、ガベージコレクタがオブジェクトを「きている」または「んだ」とした、いはカウントされません。

のコード

```
var weak = new WeakReference<FinalizableObject>(new FinalizableObject());
GC.Collect();
```

をします。

```
<namespace>.FinalizableObject initialized
<namespace>.FinalizableObject finalized
```

オブジェクトは、WeakReferenceによってされているにもかかわらず、されたヒープからされますガベージコレクタがびされたではまだスコープにあります。

1WeakReferenceターゲットがまだされているヒープにりてられているかどうかをすることは、いつでもではありません。

2プログラムがWeakreferenceのターゲットにアクセスするがあるは、ターゲットがまだりてられているかどうかののにコードをするがあります。ターゲットにアクセスするはTryGetTargetです

```
var target = new object(); // Any object will do as target
var weak = new WeakReference<object>(target); // Create weak reference
```

```
target = null; // Drop strong reference to the target

// ... Many things may happen in-between

// Check whether the target is still available
if(weak.TryGetTarget(out target))
{
    // Use re-initialized target variable
    // To do whatever the target is needed for
}
else
{
    // Do something when there is no more target object
    // The target variable value should not be used here
}
```

WeakReferenceのバージョンは.Net 4.5でです。すべてのフレームワークのバージョンは、じでビルドされ、のようにチェックされるジェネリックのなしのバージョンをします。

```
var target = new object(); // Any object will do as target
var weak = new WeakReference(target); // Create weak reference
target = null; // Drop strong reference to the target

// ... Many things may happen in-between

// Check whether the target is still available
if (weak.IsAlive)
{
    target = weak.Target;

    // Use re-initialized target variable
    // To do whatever the target is needed for
}
else
{
    // Do something when there is no more target object
    // The target variable value should not be used here
}
```

Disposeとファイナライザ

オブジェクトがもはやされなくなるとすぐにされるメモリをするとして、Disposeメソッドをするクラスを「IDisposableとしてする」とは、Disposeメソッドがびされるといういがないということですオブジェクトのわりににびされるファイナライザとはなります。

1つのシナリオは、にしたオブジェクトにしてDisposeをびすプログラムです。

```
private void SomeFunction()
{
    // Initialize an object that uses heavy external resources
    var disposableObject = new ClassThatImplementsIDisposable();

    // ... Use that object

    // Dispose as soon as no longer used
    disposableObject.Dispose();
```

```
    // ... Do other stuff

    // The disposableObject variable gets out of scope here
    // The object will be finalized later on (no guarantee when)
    // But it no longer holds to the heavy external resource after it was disposed
}
```

のシナリオでは、フレームワークによってクラスがインスタンスされるとしています。この、しいクラスは、クラスをします。たとえば、MVCでは、System.Web.Mvc.ControllerBaseのサブクラスとしてコントローラクラスをします。クラスがIDisposableインターフェイスをする、これはフレームワークによってDisposeがにびされることをすヒントですが、ここでもいはありません。

したがって、Disposeはファイナライザのわりではありません。わりに、2つはなるでするがあります。

- ファイナライザはにリソースをし、そうでなければするメモリリークをします
- Disposeは、リソースがになったらすぐにリソースをし、メモリりてのをします。

オブジェクトのなとファイナライズ

Disposeとファイナライザはなるをとしているため、のメモリがいリソースをするクラスでをするがあります。その、の2つのシナリオがにされるようにクラスをします。

- ファイナライザのみがびされたとき
- にDisposeがびされ、でファイナライザがびされると

1つのは、クリーンアップコードを1または2すると、1だけするのとじがられるようなでクリーンアップコードをきむことです。は、クリーンアップのによってなります。たとえば、のようになります。

- すでにじられているデータベースをじるとがないためします
- いくつかの ""をするのはで、1ではなく2びすとったになります。

よりなは、によって、クリーンアップコードがだけびされ、コンテキストがであれだけびされることです。これは、のフラグをして "な"をすることができます

```
public class DisposableFinalizable1: IDisposable
{
    private bool disposed = false;

    ~DisposableFinalizable1() { Cleanup(); }

    public void Dispose() { Cleanup(); }

    private void Cleanup()
    {
        if(!disposed)
        {
```

```
            // Actual code to release resources gets here, then
            disposed = true;
        }
    }
}
```

わりに、ガベ―ジコレクタは、Disposeがびされたにファイナライザをスキップできるようにする、のメソッドSuppressFinalizeをします。

```
public class DisposableFinalizable2 : IDisposable
{
    ~DisposableFinalizable2() { Cleanup(); }

    public void Dispose()
    {
        Cleanup();
        GC.SuppressFinalize(this);
    }

    private void Cleanup()
    {
        // Actual code to release resources gets here
    }
}
```

オンラインでガベ―ジコレクションをむ https://riptutorial.com/ja/dot-net/topic/9636/ガベ―ジコレクション

32: コード

コードコントラクトは、メソッドの/とオブジェクトののコンパイルまたはをにします。これらのは、びしとりがアプリケーションのなにするようにするためにされます。コードのそののには、がまれます。

Examples

は、メソッドがパラメータになをできるようにします

...

```
void DoWork(string input)
{
    Contract.Requires(!string.IsNullOrEmpty(input));

    //do work
}
```

の...

```
string s = null;
p.DoWork(s);
```

CodeContracts: requires is false: !string.IsNullOrEmpty(input)

は、メソッドからされたが、されたとすることをします。これは、びしにされるのをします。ポストコンディションでは、ツールによっていくつかのながされるため、なインプリメントがです。

...

```
string GetValue()
{
    Contract.Ensures(Contract.Result<string>() != null);

    return null;
}
```

...

```
string GetValue()
{
    Contract.Ensures(Contract.Result<string>() != null);

    return null;
}
```

CodeContracts: Invoking method 'GetValue' will always lead to an error. If this is wanted, consider adding Contract.Requires(false) to

インタフェースの

コードコントラクトをすると、インタフェースにコントラクトをすることができます。これは、インタフェースをするクラスをすることによってわれます。インタフェースは `ContractClassAttribute`でタグけされ、 `ContractClassAttribute`クラスは`ContractClassForAttribute`でタグけされるべきです

Cの...

```
[ContractClass(typeof(MyInterfaceContract))]
public interface IMyInterface
{
    string DoWork(string input);
}
//Never inherit from this contract defintion class
[ContractClassFor(typeof(IMyInterface))]
internal abstract class MyInterfaceContract : IMyInterface
{
    private MyInterfaceContract() { }

    public string DoWork(string input)
    {
        Contract.Requires(!string.IsNullOrEmpty(input));
        Contract.Ensures(!string.IsNullOrEmpty(Contract.Result<string>()));
        throw new NotSupportedException();
    }
}
public class MyInterfaceImplmentation : IMyInterface
{
    public string DoWork(string input)
    {
        return input;
    }
}
```

の...

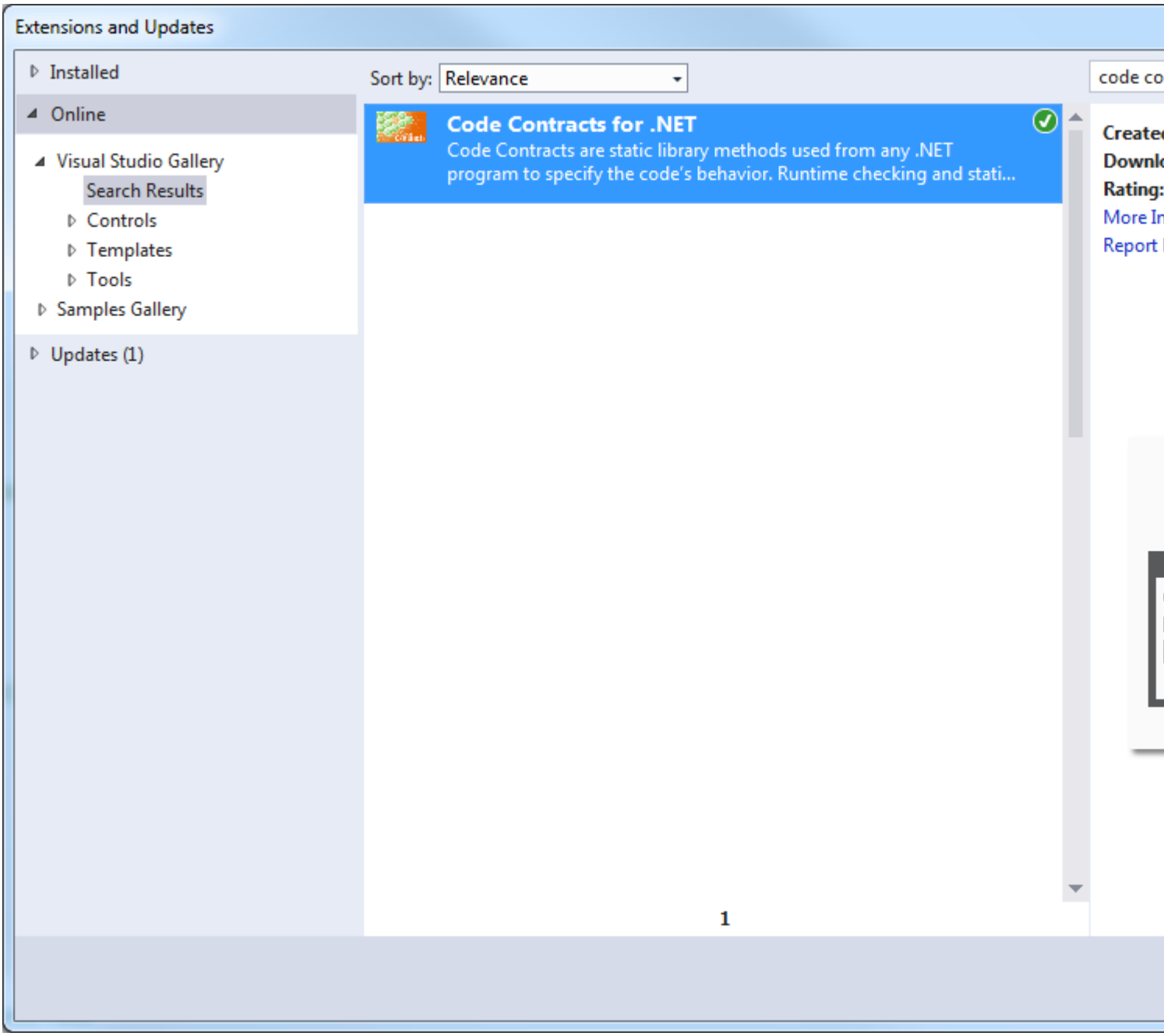
```
var m = new MyInterfaceImplmentation();
var ret = m.DoWork(null);
```

CodeContracts: requires is false: !string.IsNullOrEmpty(input)

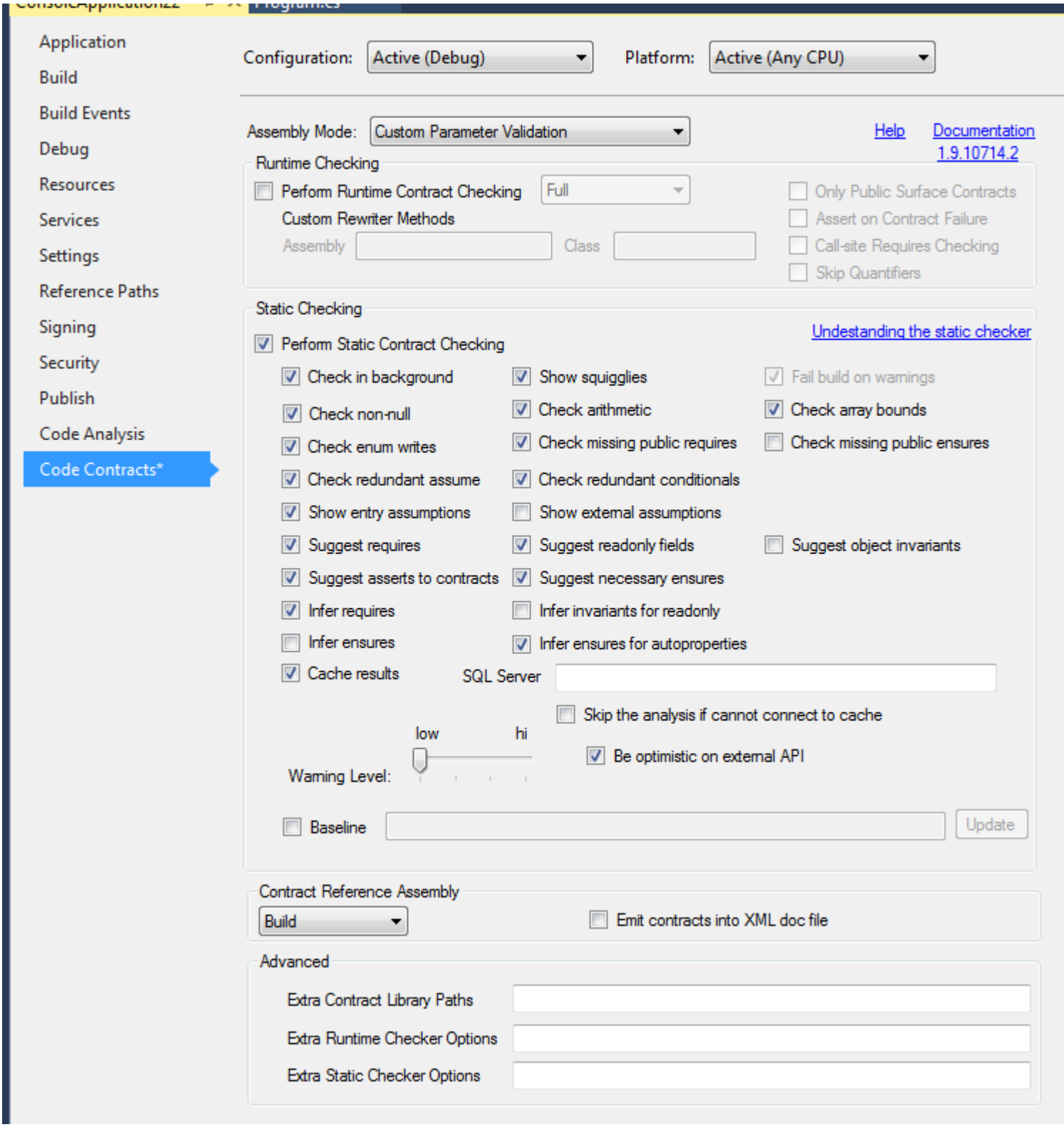
コードのインストールと

`System.Diagnostics.Contracts`は.Net Frameworkにまれています。コードコントラクトをするには、Visual Studioをインストールするがあります。

`Extensions and Updates` `Code Contracts`で、 `Code Contracts Tools`インストールし`Code Contracts Tools`



ツールをインストールしたら、プロジェクトソリューションで`Code Contracts`にするがあります。、 `Static Checking`ビルドのチェックをにしたいとうかもしれません。のソリューションでされるライブラリをするは、 `Runtime Checking`にすることもえてください。



オンラインでコードをむ https://riptutorial.com/ja/dot-net/topic/1937/コード

33: コレクション

コレクションにはいくつかのがあります

- Array
- List
- Queue
- SortedList
- Stack
-

Examples

カスタムをつリストの

```
public class Model
{
    public string Name { get; set; }
    public bool? Selected { get; set; }
}
```

ここでは、つのをつなしのコンストラクタをつクラスをっているNameとNULLブールプロパティSelected。 List<Model>をしたいは、これをするがいくつかあります。

```
var SelectedEmployees = new List<Model>
 {
      new Model() {Name = "Item1", Selected = true},
      new Model() {Name = "Item2", Selected = false},
      new Model() {Name = "Item3", Selected = false},
      new Model() {Name = "Item4"}
 };
```

ここでは、 Modelクラスのいくつかのnewインスタンスをし、それらをデータでします。コンストラクタをしたはどうなりますか

```
public class Model
{

    public Model(string name, bool? selected = false)
    {
        Name = name;
        selected = Selected;
    }
    public string Name { get; set; }
    public bool? Selected { get; set; }
}
```

これにより、リストをしってすることができます。

```
var SelectedEmployees = new List<Model>
```

```
{
    new Model("Mark", true),
    new Model("Alexis"),
    new Model("")
};
```

プロパティの1つがクラスそのものであるクラスはどうですか

```
public class Model
{
    public string Name { get; set; }
    public bool? Selected { get; set; }
}

public class ExtendedModel : Model
{
    public ExtendedModel()
    {
        BaseModel = new Model();
    }

    public Model BaseModel { get; set; }
    public DateTime BirthDate { get; set; }
}
```

をしするために、 `Model`クラスのコンストラクタをにしたことにしてください。

```
var SelectedWithBirthDate = new List<ExtendedModel>
{
    new ExtendedModel()
    {
        BaseModel = new Model { Name = "Mark", Selected = true},
        BirthDate = new DateTime(2015, 11, 23)
    },
                    new ExtendedModel()
    {
        BaseModel = new Model { Name = "Random"},
        BirthDate = new DateTime(2015, 11, 23)
    }
};
```

`List<ExtendedModel>`を`Collection<ExtendedModel>`、 `ExtendedModel[]`、 `object[]`、またはに`[]`ます。

キュー

.Netには、 FIFOれしのをする`Queue`のをするためのコレクションがあります。キューのは、キューにをするための`Enqueue(T item)`メソッドと、のをしてキューからするための`Dequeue()`メソッドです。バージョンは、のキューにしてのコードのようにできます。

まず、をします。

```
using System.Collections.Generic;
```

それをする

```
Queue<string> queue = new Queue<string>();
queue.Enqueue("John");
queue.Enqueue("Paul");
queue.Enqueue("George");
queue.Enqueue("Ringo");

string dequeueValue;
dequeueValue = queue.Dequeue(); // return John
dequeueValue = queue.Dequeue(); // return Paul
dequeueValue = queue.Dequeue(); // return George
dequeueValue = queue.Dequeue(); // return Ringo
```

オブジェクトでするのではないバ―ジョンがあります。

はのとおりです。

```
using System.Collections;
```

キュ―のコ―ドサンプルをする

```
Queue queue = new Queue();
queue.Enqueue("Hello World"); // string
queue.Enqueue(5); // int
queue.Enqueue(1d); // double
queue.Enqueue(true); // bool
queue.Enqueue(new Product()); // Product object

object dequeueValue;
dequeueValue = queue.Dequeue(); // return Hello World (string)
dequeueValue = queue.Dequeue(); // return 5 (int)
dequeueValue = queue.Dequeue(); // return 1d (double)
dequeueValue = queue.Dequeue(); // return true (bool)
dequeueValue = queue.Dequeue(); // return Product (Product type)
```

Peekというメソッドもあります。このメソッドは、をせずにキュ―のにオブジェクトをします。

```
Queue<int> queue = new Queue<int>();
queue.Enqueue(10);
queue.Enqueue(20);
queue.Enqueue(30);
queue.Enqueue(40);
queue.Enqueue(50);

foreach (int element in queue)
{
    Console.WriteLine(i);
}
```

なし

```
10
20
```

```
30
40
50
```

スタック

.Netには、 LIFOラスト·イン·ファーストアウトのをするStackのをするためのコレクションがあります。スタックのは、スタックにをするためのPush(T item)メソッドと、にされたをしてスタックからするPop()メソッドです。バ―ジョンは、のキュ―にしてのコ―ドのようにできます。

まず、をします。

```
using System.Collections.Generic;
```

それをする

```
Stack<string> stack = new Stack<string>();
stack.Push("John");
stack.Push("Paul");
stack.Push("George");
stack.Push("Ringo");

string value;
value = stack.Pop(); // return Ringo
value = stack.Pop(); // return George
value = stack.Pop(); // return Paul
value = stack.Pop(); // return John
```

オブジェクトでするのではないバ―ジョンがあります。

はのとおりです。

```
using System.Collections;
```

スタックのコ―ドサンプル

```
Stack stack = new Stack();
stack.Push("Hello World"); // string
stack.Push(5); // int
stack.Push(1d); // double
stack.Push(true); // bool
stack.Push(new Product()); // Product object

object value;
value = stack.Pop(); // return Product (Product type)
value = stack.Pop(); // return true (bool)
value = stack.Pop(); // return 1d (double)
value = stack.Pop(); // return 5 (int)
value = stack.Pop(); // return Hello World (string)
```

されたのをStackからせずにすPeekというメソッドもあります。

```
Stack<int> stack = new Stack<int>();
stack.Push(10);
stack.Push(20);

var lastValueAdded = stack.Peek(); // 20
```

スタックのをすることができ、スタックのLIFOをします。

```
Stack<int> stack = new Stack<int>();
stack.Push(10);
stack.Push(20);
stack.Push(30);
stack.Push(40);
stack.Push(50);

foreach (int element in stack)
{
   Console.WriteLine(element);
}
```

なし

```
50
40
30
20
10
```

コレクションの

いくつかのコレクションは、にできます。たとえば、のは、 `numbers`をいくつかのでしてします。

```
List<int> numbers = new List<int>(){10, 9, 8, 7, 7, 6, 5, 10, 4, 3, 2, 1};
```

には、Cコンパイラはこのをに`Add`メソッドののびしにします。したがって、このは、に`Add`メソッドをサポートするコレクションにしてのみできます。

`Stack<T>`クラスと`Queue<T>`クラスはそれをサポートしていません。

キーとのペアをる`Dictionary<TKey, TValue>`クラスなどのなコレクションの、リストでキーとのペアをとしてできます。

```
Dictionary<int, string> employee = new Dictionary<int, string>()
     {{44, "John"}, {45, "Bob"}, {47, "James"}, {48, "Franklin"}};
```

ペアののはキーで、2のはです。

オンラインでコレクションをむ https://riptutorial.com/ja/dot-net/topic/30/コレクション

34: シリアルポート

Examples

```
var serialPort = new SerialPort("COM1", 9600, Parity.Even, 8, StopBits.One);
serialPort.Open();
serialPort.WriteLine("Test data");
string response = serialPort.ReadLine();
Console.WriteLine(response);
serialPort.Close();
```

なポートをする

```
string[] portNames = SerialPort.GetPortNames();
```

リード

```
void SetupAsyncRead(SerialPort serialPort)
{
    serialPort.DataReceived += (sender, e) => {
        byte[] buffer = new byte[4096];
        switch (e.EventType)
        {
            case SerialData.Chars:
                var port = (SerialPort)sender;
                int bytesToRead = port.BytesToRead;
                if (bytesToRead > buffer.Length)
                    Array.Resize(ref buffer, bytesToRead);
                int bytesRead = port.Read(buffer, 0, bytesToRead);
                // Process the read buffer here
                // ...
                break;
            case SerialData.Eof:
                // Terminate the service here
                // ...
                break;
        }
    };
```

テキストエコーサービス

```
using System.IO.Ports;

namespace TextEchoService
{
    class Program
    {
        static void Main(string[] args)
        {
            var serialPort = new SerialPort("COM1", 9600, Parity.Even, 8, StopBits.One);
            serialPort.Open();
```

```
            string message = "";
            while (message != "quit")
            {
                message = serialPort.ReadLine();
                serialPort.WriteLine(message);
            }
            serialPort.Close();
        }
    }
}
```

メッセージ

```
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Text;
using System.Threading;

namespace AsyncReceiver
{
    class Program
    {
        const byte STX = 0x02;
        const byte ETX = 0x03;
        const byte ACK = 0x06;
        const byte NAK = 0x15;
        static ManualResetEvent terminateService = new ManualResetEvent(false);
        static readonly object eventLock = new object();
        static List<byte> unprocessedBuffer = null;

        static void Main(string[] args)
        {
            try
            {
                var serialPort = new SerialPort("COM11", 9600, Parity.Even, 8, StopBits.One);
                serialPort.DataReceived += DataReceivedHandler;
                serialPort.ErrorReceived += ErrorReceivedHandler;
                serialPort.Open();
                terminateService.WaitOne();
                serialPort.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occurred: {0}", e.Message);
            }
            Console.ReadKey();
        }

        static void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
        {
            lock (eventLock)
            {
                byte[] buffer = new byte[4096];
                switch (e.EventType)
                {
                    case SerialData.Chars:
                        var port = (SerialPort)sender;
                        int bytesToRead = port.BytesToRead;
```

```
                        if (bytesToRead > buffer.Length)
                            Array.Resize(ref buffer, bytesToRead);
                        int bytesRead = port.Read(buffer, 0, bytesToRead);
                        ProcessBuffer(buffer, bytesRead);
                        break;
                    case SerialData.Eof:
                        terminateService.Set();
                        break;
                }
            }
        }
        static void ErrorReceivedHandler(object sender, SerialErrorReceivedEventArgs e)
        {
            lock (eventLock)
                if (e.EventType == SerialError.TXFull)
                {
                    Console.WriteLine("Error: TXFull. Can't handle this!");
                    terminateService.Set();
                }
                else
                {
                    Console.WriteLine("Error: {0}. Resetting everything", e.EventType);
                    var port = (SerialPort)sender;
                    port.DiscardInBuffer();
                    port.DiscardOutBuffer();
                    unprocessedBuffer = null;
                    port.Write(new byte[] { NAK }, 0, 1);
                }
        }

        static void ProcessBuffer(byte[] buffer, int length)
        {
            List<byte> message = unprocessedBuffer;
            for (int i = 0; i < length; i++)
                if (buffer[i] == ETX)
                {
                    if (message != null)
                    {
                        Console.WriteLine("MessageReceived: {0}",
                            Encoding.ASCII.GetString(message.ToArray()));
                        message = null;
                    }
                }
                else if (buffer[i] == STX)
                    message = null;
                else if (message != null)
                    message.Add(buffer[i]);
            unprocessedBuffer = message;
        }
    }
}
```

このプログラムは、 `STX`および`ETX`バイトでまれたメッセ―ジをち、そのにるテキストをします。のすべてはされます。ライト･バッファ･オ―バ―フロ―では、それはします。のエラ―では、バッファをリセットし、それのメッセ―ジをちます。

コ―ドはのようにします。

- シリアルポ―トのみり `SerialPort.DataReceived`を。

- シリアルポートエラー `SerialPort.ErrorReceived`を。
- テキストメッセージベースのプロトコル。
- なメッセージのみり。
    - `SerialPort.DataReceived`イベントは、メッセージ `ETX` よりくするがあります。メッセージがバッファでできないもありますSerialPort.Read...、...、port.BytesToReadはメッセージののみをみみます。この、した `unprocessedBuffer` をして、それのデータをっています。
- にのメッセージをする。
    - `SerialPort.DataReceived`イベントは、のメッセージがもうのからされたにのみすることがあります。

オンラインでシリアルポートをむ https://riptutorial.com/ja/dot-net/topic/5366/シリアルポート

35: スタックとヒープ

をすると、そのは`null`になり`null`。これは、まだメモリのをしていないため、になです。
ただし、nullなをいて、のはにをたなければなりません。

Examples

のの

のはにをみます。

すべてののはSystem.ValueTypeクラスからし、これにはみみのほとんどがまれます。

しいをするときは、スタックとばれるメモリがされます。
スタックは、されたのサイズにじてそれにじてします。たとえば、intはにスタックに32ビットのメモリをりてられます。のがスコープにしなくなると、スタックののりてがされます。

のコードは、しいにされるのをしています。は、カスタムをするなとしてされています System.ValueTypeクラスはできません。

しておくべきなことは、をりてるときに、がしいにコピーされることです。つまり、オブジェクトに2つのなるインスタンスがあり、いにをぼしないということです。

```
struct PersonAsValueType
{
    public string Name;
}

class Program
{
    static void Main()
    {
        PersonAsValueType personA;

        personA.Name = "Bob";

        var personB = personA;

        personA.Name = "Linda";

        Console.WriteLine(                // Outputs 'False' - because
            object.ReferenceEquals(       // personA and personB are referencing
                personA,                  // different areas of memory
                personB));

        Console.WriteLine(personA.Name);  // Outputs 'Linda'
        Console.WriteLine(personB.Name);  // Outputs 'Bob'
    }
}
```

の

は、メモリへのとそのにされたのでされます。
これはC / C ++のポインタにています。

すべてのは、ヒープとばれるものにされます 。
ヒープは、にオブジェクトがされるメモリのされたです。しいオブジェクトがインスタンスされると、ヒープのがそのオブジェクトによってのためにりてられ、ヒープのそのへのがされます。
ヒープはガベージコレクタによってされされ、でのはできません。

インスタンスになメモリにえて、をするためのと、.NET CLRでとされるのながです。

のコードは、しいにされるをしています。このでは、クラスをしています。すべてのクラスはですであっても。

がのにされるとき、それはそのものではなくコピーされたオブジェクトへのです。これは、タイプとタイプのないです。

これは、じオブジェクトへのが2つあることをします。
そのオブジェクトののは、のにされます。

```
class PersonAsReferenceType
{
    public string Name;
}

class Program
{
    static void Main()
    {
        PersonAsReferenceType personA;

        personA = new PersonAsReferenceType { Name = "Bob" };

        var personB = personA;

        personA.Name = "Linda";

        Console.WriteLine(               // Outputs 'True' - because
            object.ReferenceEquals(      // personA and personB are referencing
                personA,                 // the *same* memory location
                personB));

        Console.WriteLine(personA.Name); // Outputs 'Linda'
        Console.WriteLine(personB.Name); // Outputs 'Linda'
    }
```

オンラインでスタックとヒープをむ https://riptutorial.com/ja/dot-net/topic/9358/スタックとヒープ

36: スピーチをするSpeechRecognitionEngineクラス

- SpeechRecognitionEngine
- SpeechRecognitionEngine.LoadGrammar
- SpeechRecognitionEngine.SetInputToDefaultAudioDevice
- SpeechRecognitionEngine.RecognizeAsyncRecognizeModeモード
- GrammarBuilder
- GrammarBuilder.Appendの
- params[]の
- GrammarBuilder Builder

パラメーター

LoadGrammar パラメータ	
	ロードする。たとえば、 `DictationGrammar`オブジェクトをして、フリーテキストのディクテーションをします。
RecognizeAsync パラメータ	
モード	の`RecognizeMode`ための`RecognizeMode` 1つののための`Single`もの、`Multiple`をにするのもの。
GrammarBuilder.Append パラメータ	
	ビルダーにいくつかのをします。これは、ユーザがをすると、がのなる「」をたどることができることをする。
Choicesコンストラクタパラメータ	
	ビルダのの。 `GrammarBuilder.Append`してください。
Grammarコンストラクタパラメータ	
ビルダー	`GrammarBuilder`は`Grammar`をします。

`SpeechRecognitionEngine`をするには、Windowsでをにするがあります。

スピーチクラスをするには、 `System.Speech.dll`へのをするがあります。

Examples

フリーテキストのをに

```
using System.Speech.Recognition;

// ...

SpeechRecognitionEngine recognitionEngine = new SpeechRecognitionEngine();
recognitionEngine.LoadGrammar(new DictationGrammar());
recognitionEngine.SpeechRecognized += delegate(object sender, SpeechRecognizedEventArgs e)
{
    Console.WriteLine("You said: {0}", e.Result.Text);
};
recognitionEngine.SetInputToDefaultAudioDevice();
recognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
```

されたフレーズセットについてをにする

```
SpeechRecognitionEngine recognitionEngine = new SpeechRecognitionEngine();
GrammarBuilder builder = new GrammarBuilder();
builder.Append(new Choices("I am", "You are", "He is", "She is", "We are", "They are"));
builder.Append(new Choices("friendly", "unfriendly"));
recognitionEngine.LoadGrammar(new Grammar(builder));
recognitionEngine.SpeechRecognized += delegate(object sender, SpeechRecognizedEventArgs e)
{
    Console.WriteLine("You said: {0}", e.Result.Text);
};
recognitionEngine.SetInputToDefaultAudioDevice();
recognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
```

オンラインでスピーチをするSpeechRecognitionEngineクラスをむ https://riptutorial.com/ja/dot-net/topic/69/スピーチをするspeechrecognitionengineクラス

37: スレッディング

Examples

のスレッドからのフォームコントロールへのアクセス

テキストボックスやラベルなどのコントロールのを、コントロールをしたGUIスレッドのスレッドからしたいは、びすがあります。そうでないは、のエラーメッセージがされることがあります。

> "クロススレッドがでないコントロール 'control_name'がされたスレッドのスレッドからアクセスされました。

このサンプルコードをsystem.windows.formsフォームですると、そのメッセージでがキャストされます。

```
private void button4_Click(object sender, EventArgs e)
{
    Thread thread = new Thread(updatetextbox);
    thread.Start();
}

private void updatetextbox()
{
    textBox1.Text = "updated"; // Throws exception
}
```

テキストボックスのテキストをしていないスレッドからするは、Control.InvokeまたはControl.BeginInvokeをします。 Control.InvokeRequiredをして、コントロールをびすがあるかどうかをすることもできます。

```
private void updatetextbox()
{
    if (textBox1.InvokeRequired)
        textBox1.BeginInvoke((Action)(() => textBox1.Text = "updated"));
    else
        textBox1.Text = "updated";
}
```

これをにうがあるは、このチェックをうためになコードのをらすために、びしなオブジェクトのをすることができます。

```
public static class Extensions
{
    public static void BeginInvokeIfRequired(this ISynchronizeInvoke obj, Action action)
    {
        if (obj.InvokeRequired)
            obj.BeginInvoke(action, new object[0]);
        else
```

```
            action();
    }
}
```

そして、どのスレッドからでもテキストボックスをするのはしシンプルになります

```
private void updatetextbox()
{
    textBox1.BeginInvokeIfRequired(() => textBox1.Text = "updated");
}
```

このでされているControl.BeginInvokeはであり、されたデリゲートがまだされているかどうかにかかわらず、Control.BeginInvokeをびしたのコードをimmedeatelyできることにしてください。

するにtextBox1がされていることをするがあるは、わりにControl.Invokeをして、がされるまでびしスレッドをブロックします。くのびしをびし、GUIスレッドがびしスレッドがされたリソースをまたはするのをっている、アプリケーションのデッドロックがすると、このアプローチはコードをにくするがあることにしてください。

オンラインでスレッディングをむ https://riptutorial.com/ja/dot-net/topic/3098/スレッディング

38: タスクライブラリTPL

と

タスクライブラリのは、マルチスレッドおよびコードのおよびプロセスをすることです。

いくつかのユースケース*

- のタスクでバックグラウンドをしてUIをにつ
- ワークロードの
- クライアントアプリケーションがリクエストをにできるようにする、TCP / UDPなど
- にのファイルをみきする

*コードは、マルチスレッドのためにケースバイケースでするがあります。たとえば、ループのりしがわずかであるや、がないは、のオーバーヘッドがメリットをるがあります。

TPL with .Net 3.5

TPLは、NuGetパッケージにまれている.Net 3.5にもで、Task Parallel Libraryとばれています。

Examples

なプロデューサ - コンシューマループBlockingCollection

```
var collection = new BlockingCollection<int>(5);
var random = new Random();

var producerTask = Task.Run(() => {
    for(int item=1; item<=10; item++)
    {
        collection.Add(item);
        Console.WriteLine("Produced: " + item);
        Thread.Sleep(random.Next(10,1000));
    }
    collection.CompleteAdding();
    Console.WriteLine("Producer completed!");
});
```

あなたが`collection.CompleteAdding();`びさなければ、それはにする`collection.CompleteAdding();`コンシューマタスクがであっても、コレクションにしけることができます。に`collection.CompleteAdding();`びし`collection.CompleteAdding();`あなたがしているときは、それのはありません。このをしてのプロデューサをのコンシューマパターンにすることができます。ここでは、のソースからBlockingCollectionにアイテムをし、のがアイテムをりしてかをうことができます。なをびすにBlockingCollectionがの、 `collection.GetConsumingEnumerable()` Enumerableは、しいアイテムがコレクションにされるか、BlockingCollection.CompleteAdding;キューがです。

```
var consumerTask = Task.Run(() => {
    foreach(var item in collection.GetConsumingEnumerable())
    {
        Console.WriteLine("Consumed: " + item);
        Thread.Sleep(random.Next(10,1000));
    }
    Console.WriteLine("Consumer completed!");
});

Task.WaitAll(producerTask, consumerTask);

Console.WriteLine("Everything completed!");
```

タスクなインスタンスと

タスクは、 `Task`クラスをインスタンスすることでできます...

```
var task = new Task(() =>
{
    Console.WriteLine("Task code starting...");
    Thread.Sleep(2000);
    Console.WriteLine("...task code ending!");
});

Console.WriteLine("Starting task...");
task.Start();
task.Wait();
Console.WriteLine("Task completed!");
```

...またはな`Task.Run`メソッドをして

```
Console.WriteLine("Starting task...");
var task = Task.Run(() =>
{
    Console.WriteLine("Task code starting...");
    Thread.Sleep(2000);
    Console.WriteLine("...task code ending!");
});
task.Wait();
Console.WriteLine("Task completed!");
```

のケースでのみ、に`Start`びすがあることにしてください。

タスク**WaitAll**とキャプチャ

```
var tasks = Enumerable.Range(1, 5).Select(n => new Task<int>(() =>
{
    Console.WriteLine("I'm task " + n);
    return n;
})).ToArray();

foreach(var task in tasks) task.Start();
Task.WaitAll(tasks);

foreach(var task in tasks)
```

```
    Console.WriteLine(task.Result);
```

タスクWaitAny

```
var allTasks = Enumerable.Range(1, 5).Select(n => new Task<int>(() => n)).ToArray();
var pendingTasks = allTasks.ToArray();

foreach(var task in allTasks) task.Start();

while(pendingTasks.Length > 0)
{
    var finishedTask = pendingTasks[Task.WaitAny(pendingTasks)];
    Console.WriteLine("Task {0} finished", finishedTask.Result);
    pendingTasks = pendingTasks.Except(new[] {finishedTask}).ToArray();
}

Task.WaitAll(allTasks);
```

`WaitAll` becasueがある`WaitAny`、がされることはありません。

タスクWaitを

```
var task1 = Task.Run(() =>
{
    Console.WriteLine("Task 1 code starting...");
    throw new Exception("Oh no, exception from task 1!!");
});

var task2 = Task.Run(() =>
{
    Console.WriteLine("Task 2 code starting...");
    throw new Exception("Oh no, exception from task 2!!");
});

Console.WriteLine("Starting tasks...");
try
{
    Task.WaitAll(task1, task2);
}
catch(AggregateException ex)
{
    Console.WriteLine("Task(s) failed!");
    foreach(var inner in ex.InnerExceptions)
        Console.WriteLine(inner.Message);
}

Console.WriteLine("Task 1 status is: " + task1.Status); //Faulted
Console.WriteLine("Task 2 status is: " + task2.Status); //Faulted
```

タスクWaitをしない

```
var task1 = Task.Run(() =>
{
    Console.WriteLine("Task 1 code starting...");
    throw new Exception("Oh no, exception from task 1!!");
```

```
});

var task2 = Task.Run(() =>
{
    Console.WriteLine("Task 2 code starting...");
    throw new Exception("Oh no, exception from task 2!!");
});

var tasks = new[] {task1, task2};

Console.WriteLine("Starting tasks...");
while(tasks.All(task => !task.IsCompleted));

foreach(var task in tasks)
{
    if(task.IsFaulted)
        Console.WriteLine("Task failed: " +
            task.Exception.InnerExceptions.First().Message);
}

Console.WriteLine("Task 1 status is: " + task1.Status); //Faulted
Console.WriteLine("Task 2 status is: " + task2.Status); //Faulted
```

タスク**CancellationToken**をしてキャンセルする

```
var cancellationTokenSource = new CancellationTokenSource();
var cancellationToken = cancellationTokenSource.Token;

var task = new Task((state) =>
    {
        int i = 1;
        var myCancellationToken = (CancellationToken)state;
        while(true)
        {
            Console.Write("{0} ", i++);
            Thread.Sleep(1000);
            myCancellationToken.ThrowIfCancellationRequested();
        }
    },
    cancellationToken: cancellationToken,
    state: cancellationToken);

Console.WriteLine("Counting to infinity. Press any key to cancel!");
task.Start();
Console.ReadKey();

cancellationTokenSource.Cancel();
try
{
    task.Wait();
}
catch(AggregateException ex)
{
    ex.Handle(inner => inner is OperationCanceledException);
}

Console.WriteLine($"{Environment.NewLine}You have cancelled! Task status is: {task.Status}");
//Canceled
```

ThrowIfCancellationRequestedのわりに、キャンセルをIsCancellationRequestedでできる、OperationCanceledExceptionをでスローすることができます。

```
//New task delegate
int i = 1;
var myCancellationToken = (CancellationToken)state;
while(!myCancellationToken.IsCancellationRequested)
{
    Console.Write("{0} ", i++);
    Thread.Sleep(1000);
}
Console.WriteLine($"{Environment.NewLine}Ouch, I have been cancelled!!");
throw new OperationCanceledException(myCancellationToken);
```

キャンセルトークンがcancellationTokenパラメータのタスクコンストラクタにどのようにされるかにしてください。これがとされているようにタスクCanceled、ではないにFaulted、ThrowIfCancellationRequestedびされます。また、じで、キャンセルトークンは、2のケースでOperationCanceledExceptionのコンストラクターでにされます。

Task.WhenAny

```
var random = new Random();
IEnumerable<Task<int>> tasks = Enumerable.Range(1, 5).Select(n => Task.Run(async() =>
{
    Console.WriteLine("I'm task " + n);
    await Task.Delay(random.Next(10,1000));
    return n;
}));

Task<Task<int>> whenAnyTask = Task.WhenAny(tasks);
Task<int> completedTask = await whenAnyTask;
Console.WriteLine("The winner is: task " + await completedTask);

await Task.WhenAll(tasks);
Console.WriteLine("All tasks finished!");
```

Task.WhenAll

```
var random = new Random();
IEnumerable<Task<int>> tasks = Enumerable.Range(1, 5).Select(n => Task.Run(() =>
{
    Console.WriteLine("I'm task " + n);
    return n;
}));

Task<int[]> task = Task.WhenAll(tasks);
int[] results = await task;

Console.WriteLine(string.Join(",", results.Select(n => n.ToString())));
// Output: 1,2,3,4,5
```

Parallel.Invoke

```
var actions = Enumerable.Range(1, 10).Select(n => new Action(() =>
{
    Console.WriteLine("I'm task " + n);
    if((n & 1) == 0)
        throw new Exception("Exception from task " + n);
})).ToArray();

try
{
    Parallel.Invoke(actions);
}
catch(AggregateException ex)
{
    foreach(var inner in ex.InnerExceptions)
        Console.WriteLine("Task failed: " + inner.Message);
}
```

Parallel.ForEach

このでは、 `Parallel.ForEach`をして、のスレッドをして110000ののをします。スレッドセーフティをするために、 `Interlocked.Add`をしてをします。

```
using System.Threading;

int Foo()
{
    int total = 0;
    var numbers = Enumerable.Range(1, 10000).ToList();
    Parallel.ForEach(numbers,
        () => 0, // initial value,
        (num, state, localSum) => num + localSum,
        localSum => Interlocked.Add(ref total, localSum));
    return total; // total = 50005000
}
```

Parallel.For

このでは、 `Parallel.For`をして、のスレッドをして110000ののをします。スレッドセーフティをするために、 `Interlocked.Add`をしてをします。

```
using System.Threading;

int Foo()
{
    int total = 0;
    Parallel.For(1, 10001,
        () => 0, // initial value,
        (num, state, localSum) => num + localSum,
        localSum => Interlocked.Add(ref total, localSum));
    return total; // total = 50005000
}
```

AsyncLocalをしたコンテキストのフロー

タスク`AsyncLocal`タスクにいくつかのデータをすがある、それはにとともにれますので、`AsyncLocal` クラスをして`AsyncLocal`

```
void Main()
{
    AsyncLocal<string> user = new AsyncLocal<string>();
    user.Value = "initial user";

    // this does not affect other tasks - values are local relative to the branches of
execution flow
    Task.Run(() => user.Value = "user from another task");

    var task1 = Task.Run(() =>
    {
        Console.WriteLine(user.Value); // outputs "initial user"
        Task.Run(() =>
        {
            // outputs "initial user" - value has flown from main method to this task without
being changed
            Console.WriteLine(user.Value);
        }).Wait();

        user.Value = "user from task1";

        Task.Run(() =>
        {
            // outputs "user from task1" - value has flown from main method to task1
            // than value was changed and flown to this task.
            Console.WriteLine(user.Value);
        }).Wait();
    });

    task1.Wait();

    // ouputs "initial user" - changes do not propagate back upstream the execution flow
    Console.WriteLine(user.Value);
}
```

のからわかるように、 `AsynLocal.Value`は`copy on read`セマンティックがありますが、をいくつかしてそのプロパティをすると、のタスクにします。したがって、 `AsyncLocal`ベストプラクティスは、またはをすることです。

VB.NETのParallel.ForEach

```
For Each row As DataRow In FooDataTable.Rows
    Me.RowsToProcess.Add(row)
Next

Dim myOptions As ParallelOptions = New ParallelOptions()
myOptions.MaxDegreeOfParallelism = environment.processorcount

Parallel.ForEach(RowsToProcess, myOptions, Sub(currentRow, state)
                                               ProcessRowParallel(currentRow, state)
                                           End Sub)
```

タスクをす

をす`Task< TResult >`りのは`Task< TResult >`ここで、TResultはされるがあるのです。タスクのをResultプロパティですることができます。

```
Task<int> t = Task.Run(() =>
    {
        int sum = 0;

        for(int i = 0; i < 500; i++)
            sum += i;

        return sum;
    });

Console.WriteLine(t.Result); // Outuput 124750
```

タスクがにされると、タスクがタスクをつよりもがされます。

```
public async Task DoSomeWork()
{
    WebClient client = new WebClient();
    // Because the task is awaited, result of the task is assigned to response
    string response = await client.DownloadStringTaskAsync("http://somedomain.com");
}
```

オンラインでタスクライブラリTPLをむ https://riptutorial.com/ja/dot-net/topic/55/タスクライブラリ-tpl-

39: タスクライブラリTPLAPIの

タスクパラレルライブラリには、アプリケーションにとをするプロセスをにするパブリックタイプとAPIがされています。 。ネット。 TPLは.Net 4でされ、マルチスレッドとコードをくためのされるです。

TPLは、スケジューリング、スレッドアフィニティ、キャンセルのサポート、、ロードバランシングをい、プログラマがなレベルのにをやすのではなく、のにできるようにします。

Examples

ボタンのクリックにじてをし、UIをする

このでは、ワーカースレッドでらかのをしてボタンクリックにし、ユーザーインターフェイスをしてをすをします

```
void MyButton_OnClick(object sender, EventArgs args)
{
    Task.Run(() => // Schedule work using the thread pool
        {
            System.Threading.Thread.Sleep(5000); // Sleep for 5 seconds to simulate work.
        })
    .ContinueWith(p => // this continuation contains the 'update' code to run on the UI thread
    {
        this.TextBlock_ResultText.Text = "The work completed at " + DateTime.Now.ToString()
    },
    TaskScheduler.FromCurrentSynchronizationContext()); // make sure the update is run on the
UI thread.

}
```

オンラインでタスクライブラリTPLAPIのをむ https://riptutorial.com/ja/dot-net/topic/5164/タスクライブラリ-tpl-apiの

40: ネットワーキング

HTTPクライアント

Examples

なTCPチャットTcpListener、TcpClient、NetworkStream

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;

class TcpChat
{
    static void Main(string[] args)
    {
        if(args.Length == 0)
        {
            Console.WriteLine("Basic TCP chat");
            Console.WriteLine();
            Console.WriteLine("Usage:");
            Console.WriteLine("tcpchat server <port>");
            Console.WriteLine("tcpchat client <url> <port>");
            return;
        }

        try
        {
            Run(args);
        }
        catch(IOException)
        {
            Console.WriteLine("--- Connection lost");
        }
        catch(SocketException ex)
        {
            Console.WriteLine("--- Can't connect: " + ex.Message);
        }
    }

    static void Run(string[] args)
    {
        TcpClient client;
        NetworkStream stream;
        byte[] buffer = new byte[256];
        var encoding = Encoding.ASCII;

        if(args[0].StartsWith("s", StringComparison.InvariantCultureIgnoreCase))
        {
            var port = int.Parse(args[1]);
            var listener = new TcpListener(IPAddress.Any, port);
            listener.Start();
            Console.WriteLine("--- Waiting for a connection...");
            client = listener.AcceptTcpClient();
```

```
        }
        else
        {
            var hostName = args[1];
            var port = int.Parse(args[2]);
            client = new TcpClient();
            client.Connect(hostName, port);
        }

        stream = client.GetStream();
        Console.WriteLine("--- Connected. Start typing! (exit with Ctrl-C)");

        while(true)
        {
            if(Console.KeyAvailable)
            {
                var lineToSend = Console.ReadLine();
                var bytesToSend = encoding.GetBytes(lineToSend + "\r\n");
                stream.Write(bytesToSend, 0, bytesToSend.Length);
                stream.Flush();
            }

            if (stream.DataAvailable)
            {
                var receivedBytesCount = stream.Read(buffer, 0, buffer.Length);
                var receivedString = encoding.GetString(buffer, 0, receivedBytesCount);
                Console.Write(receivedString);
            }
        }
    }
}
```

SNTPクライアントUdpClient

SNTPプロトコルのについては、 RFC 2030をしてください。

```
using System;
using System.Globalization;
using System.Linq;
using System.Net;
using System.Net.Sockets;

class SntpClient
{
    const int SntpPort = 123;
    static DateTime BaseDate = new DateTime(1900, 1, 1);

    static void Main(string[] args)
    {
        if(args.Length == 0) {
            Console.WriteLine("Simple SNTP client");
            Console.WriteLine();
            Console.WriteLine("Usage: sntpclient <sntp server url> [<local timezone>]");
            Console.WriteLine();
            Console.WriteLine("<local timezone>: a number between -12 and 12 as hours from
UTC");
            Console.WriteLine("(append .5 for an extra half an hour)");
            return;
        }
```

```
        double localTimeZoneInHours = 0;
        if(args.Length > 1)
            localTimeZoneInHours = double.Parse(args[1], CultureInfo.InvariantCulture);

        var udpClient = new UdpClient();
        udpClient.Client.ReceiveTimeout = 5000;

        var sntpRequest = new byte[48];
        sntpRequest[0] = 0x23; //LI=0 (no warning), VN=4, Mode=3 (client)

        udpClient.Send(
            dgram: sntpRequest,
            bytes: sntpRequest.Length,
            hostname: args[0],
            port: SntpPort);

        byte[] sntpResponse;
        try
        {
            IPEndPoint remoteEndpoint = null;
            sntpResponse = udpClient.Receive(ref remoteEndpoint);
        }
        catch(SocketException)
        {
            Console.WriteLine("*** No response received from the server");
            return;
        }

        uint numberOfSeconds;
        if(BitConverter.IsLittleEndian)
            numberOfSeconds = BitConverter.ToUInt32(
                sntpResponse.Skip(40).Take(4).Reverse().ToArray()
                ,0);
        else
            numberOfSeconds = BitConverter.ToUInt32(sntpResponse, 40);

        var date = BaseDate.AddSeconds(numberOfSeconds).AddHours(localTimeZoneInHours);

        Console.WriteLine(
            $"Current date in server: {date:yyyy-MM-dd HH:mm:ss}
UTC{localTimeZoneInHours:+0.#;-0.#;.}");
    }
}
```

オンラインでネットワーキングをむ https://riptutorial.com/ja/dot-net/topic/35/ネットワーキング

41: ファイルとPOSTデータをWebサーバーにアップロードする

Examples

WebRequestでファイルをアップロードする

のでファイルとフォームデータをするには、コンテンツにmultipart / form-dataタイプがです。

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Threading.Tasks;

public async Task<string> UploadFile(string url, string filename,
    Dictionary<string, object> postData)
{
    var request = WebRequest.CreateHttp(url);
    var boundary = $"{Guid.NewGuid():N}"; // boundary will separate each parameter
    request.ContentType = $"multipart/form-data; {nameof(boundary)}={boundary}";
    request.Method = "POST";

    using (var requestStream = request.GetRequestStream())
    using (var writer = new StreamWriter(requestStream))
    {
        foreach (var data in postData)
            await writer.WriteAsync( // put all POST data into request
                $"\r\n--{boundary}\r\nContent-Disposition: " +
                $"form-data; name=\"{data.Key}\"\r\n\r\n{data.Value}");

        await writer.WriteAsync( // file header
            $"\r\n--{boundary}\r\nContent-Disposition: " +
            $"form-data; name=\"File\"; filename=\"{Path.GetFileName(filename)}\"\r\n" +
            "Content-Type: application/octet-stream\r\n\r\n");

        await writer.FlushAsync();
        using (var fileStream = File.OpenRead(filename))
            await fileStream.CopyToAsync(requestStream);

        await writer.WriteAsync($"\r\n--{boundary}--\r\n");
    }

    using (var response = (HttpWebResponse) await request.GetResponseAsync())
    using (var responseStream = response.GetResponseStream())
    {
        if (responseStream == null)
            return string.Empty;
        using (var reader = new StreamReader(responseStream))
            return await reader.ReadToEndAsync();
    }
}
```

```
var response = await uploader.UploadFile("< YOUR URL >", "< PATH TO YOUR FILE >",
    new Dictionary<string, object>
    {
        {"Comment", "test"},
        {"Modified", DateTime.Now }
    });
```

オンラインでファイルとPOSTデータをWebサーバーにアップロードするをむ
https://riptutorial.com/ja/dot-net/topic/10845/ファイルとpostデータをwebサーバーにアップロードする

42: ファイルの

パラメーター

パラメータ	
パス	するファイルのパス。 またはされた

ファイルがするはtrue、そうでないはfalseをします。

Examples

VB WriteAllText

```
Imports System.IO

Dim filename As String = "c:\path\to\file.txt"
File.WriteAllText(filename, "Text to write" & vbCrLf)
```

VB StreamWriter

```
Dim filename As String = "c:\path\to\file.txt"
If System.IO.File.Exists(filename) Then
    Dim writer As New System.IO.StreamWriter(filename)
    writer.Write("Text to write" & vbCrLf) 'Add a newline
    writer.close()
End If
```

CStreamWriter

```
using System.Text;
using System.IO;

string filename = "c:\path\to\file.txt";
//'using' structure allows for proper disposal of stream.
using (StreamWriter writer = new StreamWriter(filename"))
{
    writer.WriteLine("Text to Write\n");
}
```

CWriteAllText

```
using System.IO;
using System.Text;

string filename = "c:\path\to\file.txt";
```

```
File.writeAllText(filename, "Text to write\n");
```

CFile.Exists

```
using System;
using System.IO;

public class Program
{
    public static void Main()
    {
        string filePath = "somePath";

        if(File.Exists(filePath))
        {
            Console.WriteLine("Exists");
        }
        else
        {
            Console.WriteLine("Does not exist");
        }
    }
}
```

でもできます。

```
Console.WriteLine(File.Exists(pathToFile) ? "Exists" : "Does not exist");
```

オンラインでファイルのをむ https://riptutorial.com/ja/dot-net/topic/1376/ファイルの

43: プラットフォームびし

- [DllImport "Example.dll"]なextern void SetTextstring inString;
- [DllImport "Example.dll"]extern void GetTextStringBuilder outString;
- [MarshalAsUnmanagedType.ByValTStr、SizeConst = 32]テキスト。
- [MarshalAsUnmanagedType.ByValArray、SizeConst = 128] byte [] byteArr;
- [StructLayoutLayoutKind.Sequential] public struct PERSON {...}
- [StructLayoutLayoutKind.Explicit] publicMarshaledUnion {[FieldOffset0] ...}

Examples

Win32のdllをびす

```
using System.Runtime.InteropServices;

class PInvokeExample
{
    [DllImport("user32.dll", CharSet = CharSet.Auto)]
    public static extern uint MessageBox(IntPtr hWnd, String text, String caption, int
options);

    public static void test()
    {
        MessageBox(IntPtr.Zero, "Hello!", "Message", 0);
    }
}
```

`static extern` stll `DllImportAttribute`として、 `Value`プロパティを.dll nameにしてをします。`System.Runtime.InteropServices`をすることをれないでください。その、のメソッドとしてびします。

Platform Invocation Servicesは、.dllのみみとのをいます。ほとんどのなケースでは、P / Invokeはパラメータをマーシャリングして.dllとのでりをしますつまり、.NETデータからWin32データへの、もまたです。

Windows APIの

pinvoke.netをしてください。

あなたのコードで`extern` Windows APIをするに、 pinvoke.netでそれをすことをしてください。らはすでに、すべてのサポートタイプといでなをっているがもいでしよう。

のマーシャリング

の

```
[DllImport("Example.dll")]
static extern void SetArray(
    [MarshalAs(UnmanagedType.LPArray, SizeConst = 128)]
    byte[] data);
```

の

```
[DllImport("Example.dll")]
static extern void SetStrArray(string[] textLines);
```

をマーシャリングする

シンプルな

C ++シグネチャ

```
typedef struct _PERSON
{
    int age;
    char name[32];
} PERSON, *LP_PERSON;

void GetSpouse(PERSON person, LP_PERSON spouse);
```

Cの

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct PERSON
{
    public int age;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 32)]
    public string name;
}

[DllImport("family.dll", CharSet = CharSet.Auto)]
public static extern bool GetSpouse(PERSON person, ref PERSON spouse);
```

フィールドがな。

C ++

```
typedef struct
{
    int length;
    int *data;
} VECTOR;

void SetVector(VECTOR &vector);
```

コードからコードにすと、これは

`data`はIntPtrとしてするがあり、メモリは`Marshal.AllocHGlobal()`にりてられなければなりません`Marshal.FreeHGlobal()`

にされます。

```
[StructLayout(LayoutKind.Sequential)]
public struct VECTOR : IDisposable
{
    int length;
    IntPtr dataBuf;

    public int[] data
    {
        set
        {
            FreeDataBuf();
            if (value != null && value.Length > 0)
            {
                dataBuf = Marshal.AllocHGlobal(value.Length * Marshal.SizeOf(value[0]));
                Marshal.Copy(value, 0, dataBuf, value.Length);
                length = value.Length;
            }
        }
    }
    void FreeDataBuf()
    {
        if (dataBuf != IntPtr.Zero)
        {
            Marshal.FreeHGlobal(dataBuf);
            dataBuf = IntPtr.Zero;
        }
    }
    public void Dispose()
    {
        FreeDataBuf();
    }
}

[DllImport("vectors.dll")]
public static extern void SetVector([In]ref VECTOR vector);
```

フィールドがな。

C ++シグネチャ

```
typedef struct
{
    char *name;
} USER;

bool GetCurrentUser(USER *user);
```

そのようなデータがアンマネージコードからされ、アンマネージによってメモリがりてられると、マネージドコールはそれを`IntPrt`にけり、そのバッファをマネージドにする`IntPrt`ます。の、な`Marshal.PtrToStringAnsi()`メソッドがあります

```
[StructLayout(LayoutKind.Sequential)]
public struct USER
{
```

```
    IntPtr nameBuffer;
    public string name { get { return Marshal.PtrToStringAnsi(nameBuffer); } }
}

[DllImport("users.dll")]
public static extern bool GetCurrentUser(out USER user);
```

のマ―シャリング

フィ―ルドのみ

C ++

```
typedef union
{
    char c;
    int i;
} CharOrInt;
```

C

```
[StructLayout(LayoutKind.Explicit)]
public struct CharOrInt
{
    [FieldOffset(0)]
    public byte c;
    [FieldOffset(0)]
    public int i;
}
```

フィ―ルドとフィ―ルドをさせる

タイプ1のをさせることはできません。 ~~FieldOffset(0) text; FieldOffset(0) i;~~ のためにコンパイルされません

```
typedef union
{
    char text[128];
    int i;
} TextOrInt;
```

にカスタムマ―シャリングをするがあります。しかし、にこのようなよりなテクニクスのは、のようにできます。

```
[StructLayout(LayoutKind.Sequential)]
public struct TextOrInt
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 128)]
    public byte[] text;
    public int i { get { return BitConverter.ToInt32(text, 0); } }
}
```

オンラインでプラットフォームびしをむ https://riptutorial.com/ja/dot-net/topic/1643/プラットフォームびし

44: プロセスとスレッドの

パラメーター

パラメータ	
	プロセスのがされているプロセッサのセットをす。たとえば、8プロセッサー·システムでは、プロセッサー3と4でのみプロセスをしたいは、のようなアフィニティをするよりも00001100 = 12

スレッドのプロセッサは、それがしているプロセッサのです。いえれば、のものです。

プロセッサアフィニティは、プロセッサをビットとしてします。ビット0はプロセッサ1をし、ビット1はプロセッサ2をし、である。

Examples

プロセスアフィニティマスクをする

```
    public static int GetProcessAffinityMask(string processName = null)
    {
        Process myProcess = GetProcessByName(ref processName);

        int processorAffinity = (int)myProcess.ProcessorAffinity;
        Console.WriteLine("Process {0} Affinity Mask is : {1}", processName,
FormatAffinity(processorAffinity));

        return processorAffinity;
    }

    public static Process GetProcessByName(ref string processName)
    {
        Process myProcess;
        if (string.IsNullOrEmpty(processName))
        {
            myProcess = Process.GetCurrentProcess();
            processName = myProcess.ProcessName;
        }
        else
        {
            Process[] processList = Process.GetProcessesByName(processName);
            myProcess = processList[0];
        }
        return myProcess;
    }

    private static string FormatAffinity(int affinity)
```

```
    {
        return Convert.ToString(affinity, 2).PadLeft(Environment.ProcessorCount, '0');
    }
}
```

```
private static void Main(string[] args)
{
    GetProcessAffinityMask();

    Console.ReadKey();
}
// Output:
// Process Test.vshost Affinity Mask is : 11111111
```

プロセスアフィニティマスクをする

```
    public static void SetProcessAffinityMask(int affinity, string processName = null)
    {
        Process myProcess = GetProcessByName(ref processName);

        Console.WriteLine("Process {0} Old Affinity Mask is : {1}", processName,
FormatAffinity((int)myProcess.ProcessorAffinity));

        myProcess.ProcessorAffinity = new IntPtr(affinity);
        Console.WriteLine("Process {0} New Affinity Mask is : {1}", processName,
FormatAffinity((int)myProcess.ProcessorAffinity));
    }
```

```
private static void Main(string[] args)
{
    int newAffinity = Convert.ToInt32("10101010", 2);
    SetProcessAffinityMask(newAffinity);

    Console.ReadKey();
}
// Output :
// Process Test.vshost Old Affinity Mask is : 11111111
// Process Test.vshost New Affinity Mask is : 10101010
```

オンラインでプロセスとスレッドのをむ https://riptutorial.com/ja/dot-net/topic/4431/プロセスとスレッドの

45: メモリ

.NETアプリケーションのパフォーマンスアプリケーションは、GCによってなをけるがあります。 GCがされると、のすべてのスレッドはするまでされます。このため、GCプロセスをにし、するタイミングをにえるをすることをおめします。

Examples

されていないリソース

GCと「ヒープ」についてすると、にはマネージヒープとばれるものについてしています。 されたヒープのオブジェクトは、ファイルへのきみやファイルからのみみなど、されたヒープにないリソースにアクセスできます。しないは、ファイルがみりのためにかれたにがし、ファイルハンドルがどおりにじるのをぐにするがあります。このため、.NETではされていないリソースで`IDisposable`インターフェイスがされているがあります。このインタフェースには、パラメータなしの`Dispose`というのメソッドがあります。

```
public interface IDisposable
{
    Dispose();
}
```

されていないリソースをするときは、それらがにされていることをするがあります。これは、`finally`ブロックまたは`using`ステートメントで`Dispose()`にびすことによってできます。

```
StreamReader sr;
string textFromFile;
string filename = "SomeFile.txt";
try
{
    sr = new StreamReader(filename);
    textFromFile = sr.ReadToEnd();
}
finally
{
    if (sr != null) sr.Dispose();
}
```

または

```
string textFromFile;
string filename = "SomeFile.txt";

using (StreamReader sr = new Streamreader(filename))
{
    textFromFile = sr.ReadToEnd();
}
```

がましいであり、コンパイルににされます。

されていないリソースをラップするときにSafeHandleをする

されていないリソースのラッパーをするときは、 `IDisposable`とfinalizerをするのではなく、`SafeHandle`サブクラス`SafeHandle`をうがあります。 `SafeHandle`サブクラスは、ハンドルリークのをにえるため、できるだけさく、シンプルにするがあります。これは、SafeHandleのが、なAPIをするためにそれをラップするクラスののであるがいことをします。このクラスは、たとえプログラムが`SafeHandle`インスタンスをリークしても、アンマネージハンドルがされることをします。

```
using System.Runtime.InteropServices;

class MyHandle : SafeHandle
{
    public override bool IsInvalid => handle == IntPtr.Zero;
    public MyHandle() : base(IntPtr.Zero, true)
    { }

    public MyHandle(int length) : this()
    {
        SetHandle(Marshal.AllocHGlobal(length));
    }

    protected override bool ReleaseHandle()
    {
        Marshal.FreeHGlobal(handle);
        return true;
    }
}
```

このは、 `IDisposable`をし、ファイナライザをにする`SafeHandle`でリソースをするをすためのものです。このでメモリのチャンクをりてることはにされており、であるがあります。

オンラインでメモリをむ https://riptutorial.com/ja/dot-net/topic/59/メモリ

46:

- MSDNとCプログラミングガイド
- MSDNのとスロー
- MSDNCA1031なのをキャッチしない
- MSDNtry-catchCリファレンス

Examples

をキャッチする

なでコードでをスローすることができます。これのとしては、

- ストリームのわりをぎてみろうとしています
- ファイルにアクセスするためにながない
- ゼロでるなどのなをしようとしています
- インターネットからファイルをダウンロードするときにするタイムアウト

びしは、これらのを「キャッチ」することでできます。のにのみしてください。

- それはになをしたり、にすることができます。
- をスローするがあるになにのコンテキストをすることができますスローされたはハンドラによってコールスタックのにキャッチされます

をキャッチしないことをすることは、それがよりいレベルでされることがされているにはにであることにされたい。

をキャッチするには、にスローするコードを`try { ... }`ブロックにのようにラップし、`catch (ExceptionType) { ... }`ブロックでできる`catch (ExceptionType) { ... }`ます。

```
Console.Write("Please enter a filename: ");
string filename = Console.ReadLine();

Stream fileStream;

try
{
    fileStream = File.Open(filename);
}
catch (FileNotFoundException)
{
    Console.WriteLine("File '{0}' could not be found.", filename);
}
```

finallyブロックの

がしたかどうかになく、`try-finally`または`try-catch-finally`の`finally { ... }`ブロックはにされ

ますは`StackOverflowException`がスロ―されたか、 `Environment.FailFast()`。

`try { ... }`ブロックでしたリソ―スをにまたはクリ―ンアップするためにできます。

```
Console.Write("Please enter a filename: ");
string filename = Console.ReadLine();

Stream fileStream = null;

try
{
    fileStream = File.Open(filename);
}
catch (FileNotFoundException)
{
    Console.WriteLine("File '{0}' could not be found.", filename);
}
finally
{
    if (fileStream != null)
    {
        fileStream.Dispose();
    }
}
```

キャッチされたのキャッチとスロ―

をキャッチしてかをしたいが、のためにのコ―ドブロックのをできないときは、コ―ルスタックののハンドラにをすことがながあります。これをうにはいといがあります。

```
private static void AskTheUltimateQuestion()
{
    try
    {
        var x = 42;
        var y = x / (x - x); // will throw a DivideByZeroException

        // IMPORTANT NOTE: the error in following string format IS intentional
        // and exists to throw an exception to the FormatException catch, below
        Console.WriteLine("The secret to life, the universe, and everything is {1}", y);
    }
    catch (DivideByZeroException)
    {
        // we do not need a reference to the exception
        Console.WriteLine("Dividing by zero would destroy the universe.");

        // do this to preserve the stack trace:
        throw;
    }
    catch (FormatException ex)
    {
        // only do this if you need to change the type of the Exception to be thrown
        // and wrap the inner Exception

        // remember that the stack trace of the outer Exception will point to the
        // next line
```

```
        // you'll need to examine the InnerException property to get the stack trace
        // to the line that actually started the problem

        throw new InvalidOperationException("Watch your format string indexes.", ex);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Something else horrible happened. The exception: " + ex.Message);

        // do not do this, because the stack trace will be changed to point to
        // this location instead of the location where the exception
        // was originally thrown:
        throw ex;
    }
}

static void Main()
{
    try
    {
        AskTheUltimateQuestion();
    }
    catch
    {
        // choose this kind of catch if you don't need any information about
        // the exception that was caught

        // this block "eats" all exceptions instead of rethrowing them
    }
}
```

あなたはタイプとプロパティC6.0の、VB.NETでもうしですがですでフィルタリングすることもできます

ドキュメンテーション/ C/

フィルタ

C6.0のは、 `when`をしてフィルタリングできるためです。

これはな`if`をするのとていますが、 `when`のがたされていない`when`はスタックをきしません。

```
try
{
  // ...
}
catch (Exception e) when (e.InnerException != null) // Any condition can go in here.
{
  // ...
}
```

じはC6.0のにありますフィルタ

catchブロックでのの

catchブロックでは、をせずにthrowキーワードをですることができ、 catchされたばかりのをすることができます。をすると、のはチェーンをし、そのびしスタックまたはするデータはされます。

```
try {...}
catch (Exception ex) {
  // Note: the ex variable is *not* used
  throw;
}
```

なアンチパターンはわりthrow exをthrow exすることです。スタックトレースののハンドラのビューをするというがあります。

```
try {...}
catch (Exception ex) {
  // Note: the ex variable is thrown
  //  future stack traces of the exception will not see prior calls
  throw ex;
}
```

にしてthrow exスタックトレースをし、のハンドラのみにまでさかのぼるとしてびしをることができますよう、ましくないthrow ex。 exをし、 throwキーワードだけをすると、のは"バブルアップ"します。

をしながらのメソッドからをスローする

をキャッチして、のスタックをしながら、のスレッドまたはメソッドからをスローすることがながあります。 ExceptionDispatchInfoこれをうことができExceptionDispatchInfo

```
using System.Runtime.ExceptionServices;

void Main()
{
    ExceptionDispatchInfo capturedException = null;
    try
    {
        throw new Exception();
    }
    catch (Exception ex)
    {
        capturedException = ExceptionDispatchInfo.Capture(ex);
    }

    Foo(capturedException);
}

void Foo(ExceptionDispatchInfo exceptionDispatchInfo)
{
    // Do stuff

    if (capturedException != null)
    {
        // Exception stack trace will show it was thrown from Main() and not from Foo()
        exceptionDispatchInfo.Throw();
```

```
    }
}
```

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/33/

47:

によってされる

をしなかった、 `Greeter`クラスはのようになります。

```
public class ControlFreakGreeter
{
    public void Greet()
    {
        var greetingProvider = new SqlGreetingProvider(
            ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString);
        var greeting = greetingProvider.GetGreeting();
        Console.WriteLine(greeting);
    }
}
```

それは、をするクラスのをし、SQLがどこからているかをし、をするので、「コントロ―ル･フリ―ク」です。

インジェクションをして、 `Greeter`クラスはをしてのをえ、されたをいています。

Dependency Inversion Principleでは、クラスはのなクラスではなくインター―フェイスなどにするがあることをしています。クラスののは、を々ににするがあります。にじて、そのをさせることができる。

は、にするクラスのにつながるため、のをするのにちます。 `Greeter`クラスは、 `IGreetingProvider`と`IGreetingWriter`のについてはもらない。されたがそれらのインタフェ―スをしていることだけがわかります。つまり、 `IGreetingProvider`と`IGreetingWriter`をするなクラスをしても、 `Greeter`はしません。どちらも、それらをまったくなるにきえません。インタフェ―スにするのみがわれます。 `Greeter`はされています。

`ControlFreakGreeter`は`ControlFreakGreeter`をしく`ControlFreakGreeter`ができません。さなコ―ドをテストしたいが、テストにはSQLへのとストアドプロシ―ジャのがまれる。コンソ―ルのテストもまれます。 ControlFreakGreeterはそれほどくのことをするので、のクラスとはしてテストすることはです。

`Greeter`はテストがです。なぜなら、ストアドプロシ―ジャをびすか、コンソ―ルのをみるよりも、とがなのをすることができるからです。 app.configにはありません。

`IGreetingProvider`と`IGreetingWriter`のなはもっとになるかもしれません。らはのをっているかもしれません。 たとえば、SQLを`SqlGreetingProvider`ます。しかし、そのさは、インター―フェイスにのみするのクラスからは「されています」。これにより、のクラスにするをえるがある「」なしに、あるクラスをすることがになります。

Examples

- な

このクラスは`Greeter`とばれます。そのは、をすることです。それには2つのがあります。それにはするをえるかがです。そして、をするがです。これらのは、どちらもインタフェーース、`IGreetingProvider`、 `IGreetingWriter`としてされています。このでは、これら2つのが`Greeter`「」されています。 このにく

```
public class Greeter
{
    private readonly IGreetingProvider _greetingProvider;
    private readonly IGreetingWriter _greetingWriter;

    public Greeter(IGreetingProvider greetingProvider, IGreetingWriter greetingWriter)
    {
        _greetingProvider = greetingProvider;
        _greetingWriter = greetingWriter;
    }

    public void Greet()
    {
        var greeting = _greetingProvider.GetGreeting();
        _greetingWriter.WriteGreeting(greeting);
    }
}

public interface IGreetingProvider
{
    string GetGreeting();
}

public interface IGreetingWriter
{
    void WriteGreeting(string greeting);
}
```

`Greeting`クラスは`IGreetingProvider`と`IGreetingWriter`にしますが、いずれかのインスタンスをするはありません。わりに、コンストラクタでそれらをとします。 `Greeting`インスタンスをするものは、その2つのをするがあります。を「」することができます。

はコンストラクタのクラスにされるため、これはコンストラクタインジェクションともばれます。

いくつかのな

- コンストラクタは、を`private`フィーールドとしてします。クラスがインスタンスされるとすぐに、それらのはクラスののすべてのメソッドでできます。
- `private`フィーールドは`readonly`です。それらがコンストラクタにされると、することはできません。これは、これらのフィーールドがコンストラクタのでできないようにするできないことをします。これにより、クラスのにこれらのができるようになります。
- はインタフェーースです。これはにはではありませんが、の1つのをのとにきえることができるため、です。また、テストのでインターフェイスのバージョンをすることもできます。

がテストをよりにする

これは、 `IGreetingProvider`と`IGreetingWriter` 2つのをつ`Greeter`クラスののをベースにしています。

`IGreetingProvider`ののでは、APIびしまたはデータベースからをできます。 `IGreetingWriter`のによって、コンソールにがされることがあります。しかし、 `Greeter`はそのコンストラクタにがされているため、これらのインタフェースのバージョンをするテストをくのはです。では、 Moqのようなフレームワークをうかもしれませんが、この、はそれらのをくでしょう。

```
public class TestGreetingProvider : IGreetingProvider
{
    public const string TestGreeting = "Hello!";

    public string GetGreeting()
    {
        return TestGreeting;
    }
}

public class TestGreetingWriter : List<string>, IGreetingWriter
{
    public void WriteGreeting(string greeting)
    {
        Add(greeting);
    }
}

[TestClass]
public class GreeterTests
{
    [TestMethod]
    public void Greeter_WritesGreeting()
    {
        var greetingProvider = new TestGreetingProvider();
        var greetingWriter = new TestGreetingWriter();
        var greeter = new Greeter(greetingProvider, greetingWriter);
        greeter.Greet();
        Assert.AreEqual(greetingWriter[0], TestGreetingProvider.TestGreeting);
    }
}
```

`IGreetingProvider`と`IGreetingWriter`のはこのテストとはありません。たちは`Greeter`がをしてそれをくことをテストしたいとっています。 `Greeter`のをにより、なをせずにをすることができます。たちがテストしていることは、 `Greeter`がそれらのをしているからです。

コンテナIoCコンテナをする

とは、をしないようにクラスをすることです。そのわりに、がされますされます

Castle Windsor、Autofac、SimpleInjector、Ninject、Unityなどのフレームワークしばしば "DIコンテナ"、 "IoCコンテナ"、またはに "コンテナ"とばれるをするのとじことではありません。

コンテナはにをにします。たとえば、にするいくつかのクラスをするとします。あるクラスはいくつかのインターフェースにし、それらのインターフェースをするクラスはのインターフェースにします。いくつかはのにします。そしてしみのために、これらのクラスのいくつかは`IDisposable`をし、するがあります。

それぞれのクラスはよくかれており、にテストできます。しかしはのがありますクラスのインスタンスをすることはずっとになっています。`CustomerService`クラスのインスタンスをしているとします。それにはがあり、にはがあります。インスタンスのはのようになります。

```
public CustomerData GetCustomerData(string customerNumber)
{
    var customerApiEndpoint =
ConfigurationManager.AppSettings["customerApi:customerApiEndpoint"];
    var logFilePath = ConfigurationManager.AppSettings["logwriter:logFilePath"];
    var authConnectionString =
ConfigurationManager.ConnectionStrings["authorization"].ConnectionString;
    using(var logWriter = new LogWriter(logFilePath ))
    {
        using(var customerApiClient = new CustomerApiClient(customerApiEndpoint))
        {
            var customerService = new CustomerService(
                new SqlAuthorizationRepository(authorizationConnectionString, logWriter),
                new CustomerDataRepository(customerApiClient, logWriter),
                logWriter
            );

            // All this just to create an instance of CustomerService!
            return customerService.GetCustomerData(string customerNumber);
        }
    }
}
```

なを`CustomerService`すのにれてみてはどうでしょうか 1つのは、クラスのがそれにされるため、クラスはそのが`IDisposable`かどうかをるがないということです。それはにそれらをします。そのため、にされた`CustomerService`をす`GetCustomerService()`がある、そのクラスにはいてなリソースがくまれており、アクセスまたはするはありません。

そして、`IDisposable`をするのとはに、そのようなれになったコンストラクタをびたいはいますかこれはなです。それははるかにするがあります。りしますが、それはたちがったでクラスをいたことをするものではありません。クラスはにかもしれません。はそれらをにすることです。

コンテナはそれをします。これにより、をたすためにするクラスまたはをすることができます。このわずかにされたは、Castle Windsorをしています。

```
var container = new WindsorContainer()
container.Register(
    Component.For<CustomerService>(),
    Component.For<ILogWriter, LogWriter>()
        .DependsOn(Dependency.OnAppSettingsValue("logFilePath", "logWriter:logFilePath")),
    Component.For<IAuthorizationRepository, SqlAuthorizationRepository>()
        .DependsOn(Dependency.OnValue(connectionString,
ConfigurationManager.ConnectionStrings["authorization"].ConnectionString)),
    Component.For<ICustomerDataProvider, CustomerApiClient>()
```

```
        .DependsOn(Dependency.OnAppSettingsValue("apiEndpoint",
"customerApi:customerApiEndpoint"))
);
```

これを「の」または「コンテナの」とびます。み、これは`WindsorContainer`えます

- クラスに`ILogWriter`がなは、 `LogWriter`インスタンスをします。 `LogWriter`はファイルパスがです。 `AppSettings`からこのをします。
- クラスに`IAuthorizationRepository`がなは、 `SqlAuthorizationRepository`インスタンスをします。がです。 `ConnectionStrings`セクションからこのをします。
- クラスに`ICustomerDataProvider`がなは、 `CustomerApiClient`をし、なを`AppSettings`からします。

コンテナからをするときは、それをを「する」とびます。コンテナをってうのはいですが、それはのです。デモンストレーションので、これをうことができます

```
var customerService = container.Resolve<CustomerService>();
var data = customerService.GetCustomerData(customerNumber);
container.Release(customerService);
```

コンテナは、 `CustomerService`が`IAuthorizationRepository`および`ICustomerDataProvider`していることを`ICustomerDataProvider`ます。これらのをたすためにどのクラスをするがあるかはわかっています。これらのクラスにはがく、コンテナにはそれらをたすがあります。それは、 `CustomerService`インスタンスをすまで、なすべてのクラスをします。

`IDoesSomethingElse`ように`IDoesSomethingElse`にされていないがなは、 `CustomerService`をしようとすると、そのをたすためにもしていないことをするながスローされます。

DIフレームワークのはしなりますが、は、のクラスがどのようにインスタンスされるかをします。たとえば、 `LogWriter`インスタンスを1つし、それを`ILogWriter`にするすべてのクラスにしたいのですか、しいクラスをするがありますかほとんどのコンテナには、それをするがあります。

`IDisposable`をするクラスはどうですかだから々は`container.Release(customerService);`をびし`container.Release(customerService);`に。 ウィンザーむほとんどのコンテナがされたのすべてをとなり`Dispose`なものを。 `CustomerService`が`IDisposable`、それもされます。

のようにをすると、よりくのコードをすることができます。しかし、くのをつクラスがたくさんあるときは、それはにわれます。また、をわずにじクラスをくがある、クラスがいじアプリケーションではやテストがしくなります。

これはなぜコンテナをうのかをにつける。アプリケーションをする そしてしくするはなる1つのトピックではなく、とがコンテナごとになるため、くのトピックです。

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/5085/

48: テスト

Examples

MSTestユニットテストプロジェクトをのソリューションにする

- ソリューションをクリックし、しいプロジェクトをします。
- [テスト]セクションで、[ユニットテストプロジェクト]をします
- アセンブリのをふ - プロジェクト`Foo`をテストしている、そのは`Foo.Tests`
- テストプロジェクトリファレンスで、テストみプロジェクトへのをする

サンプルテストメソッドの

MSTestのテストフレームワークでは、`[TestClass]`でされたテストクラスと`[TestMethod]`でテストメソッドをし、パブリックにするがあります。

```
[TestClass]
public class FizzBuzzFixture
{
    [TestMethod]
    public void Test1()
    {
        //arrange
        var solver = new FizzBuzzSolver();
        //act
        var result = solver.FizzBuzz(1);
        //assert
        Assert.AreEqual("1",result);
    }
}
```

オンラインでテストをむ https://riptutorial.com/ja/dot-net/topic/5365/テスト

49:

Examples

アセンブリとはですか

アセンブリは、 ランタイムCLRアプリケーションのビルディングブロックです。したすべてのは、そのメソッド、プロパティ、およびバイトコードとともに、アセンブリでコンパイルされ、パッケージされます。

```
using System.Reflection;
```

```
Assembly assembly = this.GetType().Assembly;
```

アセンブリは、、メソッド、およびILコードだけでなく、コンパイルおよびので、およびになメタデータもみます。

```
Assembly assembly = Assembly.GetExecutingAssembly();

foreach (var type in assembly.GetTypes())
{
    Console.WriteLine(type.FullName);
}
```

アセンブリには、でのIDをすがけられます。

```
Console.WriteLine(typeof(int).Assembly.FullName);
// Will print: "mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
```

このにPublicKeyTokenまれている、そのはPublicKeyToken なとばれます。アセンブリのなは、アセンブリでされたにするをしてをするプロセスです。このシグネチャは、アセンブリをするすべてのファイルのとハッシュをむアセンブリマニフェストにされ、そのPublicKeyTokenはのになります。じなをつアセンブリはであるがあります。バージョニングにはながされ、アセンブリのをします。

リフレクションをしてTのオブジェクトをする

デフォルトコンストラクタの

```
T variable = Activator.CreateInstance(typeof(T));
```

パラメータされたコンストラクタの

```
T variable = Activator.CreateInstance(typeof(T), arg1, arg2);
```

リフレクションをしたオブジェクトのとプロパティの

プロパティーをつ`Classy`クラスがあるとしましょう

```
public class Classy
{
    public string Propertua {get; set;}
}
```

をって`Propertua`をする

```
var typeOfClassy = typeof (Classy);
var classy = new Classy();
var prop = typeOfClassy.GetProperty("Propertua");
prop.SetValue(classy, "Value");
```

リフレクションきののをするおよびキャッシングする

は、enumのメタデータをすのにです。これのをることはくなるがあるので、をキャッシュすることがです。

```
    private static Dictionary<object, object> attributeCache = new Dictionary<object,
object>();

    public static T GetAttribute<T, V>(this V value)
        where T : Attribute
        where V : struct
    {
        object temp;

        // Try to get the value from the static cache.
        if (attributeCache.TryGetValue(value, out temp))
        {
            return (T) temp;
        }
        else
        {
            // Get the type of the struct passed in.
            Type type = value.GetType();
            FieldInfo fieldInfo = type.GetField(value.ToString());

            // Get the custom attributes of the type desired found on the struct.
            T[] attribs = (T[])fieldInfo.GetCustomAttributes(typeof(T), false);

            // Return the first if there was a match.
            var result = attribs.Length > 0 ? attribs[0] : null;

            // Cache the result so future checks won't need reflection.
            attributeCache.Add(value, result);

            return result;
        }
    }
```

リフレクションで2つのオブジェクトをする

```
public class Equatable
{
    public string field1;

    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;

        var type = obj.GetType();
        if (GetType() != type)
            return false;

        var fields = type.GetFields(BindingFlags.Instance | BindingFlags.NonPublic |
BindingFlags.Public);
        foreach (var field in fields)
            if (field.GetValue(this) != field.GetValue(obj))
                return false;

        return true;
    }

    public override int GetHashCode()
    {
        var accumulator = 0;
        var fields = GetType().GetFields(BindingFlags.Instance | BindingFlags.NonPublic |
BindingFlags.Public);
        foreach (var field in fields)
            accumulator = unchecked ((accumulator * 937) ^
field.GetValue(this).GetHashCode());

        return accumulator;
    }
}
```

このでは、のためにフィールドベースのをっていますフィールドとプロパティをします。

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/44/

50: コンテキスト

コンテキストとは、のスケジュールをすることなく、をスケジューラにすようにコードをするです。

コンテキストはにコードがのスレッドでされるようにするためにされます。 WPFおよびWinformsアプリケーションでは、UIスレッドをす`SynchronizationContext`がプレゼンテーションフレームワークによってされます。このようにして、 `SynchronizationContext`はデリゲートのプロデューサ/コンシューマパターンとえることができます。ワーカースレッドは、なコードデリゲートをし、UIメッセージループによってそれやをキューにれます。

タスクパラレルライブラリには、コンテキストをにキャプチャしてするためのがされています。

Examples

バックグラウンドをった、UIスレッドでコードをする

このは、 `SynchronizationContext`をしてバックグラウンドスレッドからUIコンポーネントをするをしています

```
void Button_Click(object sender, EventArgs args)
{
    SynchronizationContext context = SynchronizationContext.Current;
    Task.Run(() =>
    {
        for(int i = 0; i < 10; i++)
        {
            Thread.Sleep(500); //simulate work being done
            context.Post(ShowProgress, "Work complete on item " + i);
        }
    }
}

void UpdateCallback(object state)
{
    // UI can be safely updated as this method is only called from the UI thread
    this.MyTextBox.Text = state as string;
}
```

このでは、 `for`ループの`MyTextBox.Text`をしようとすると、スレッドエラーがします。 `SynchronizationContext` `UpdateCallback`アクションをポストすることで、りのUIとじスレッドでテキストボックスがされます。

には、のは`System.IProgress<T>`インスタンスをしてするがあります。の`System.Progress<T>`は、されたコンテキストをにします。

オンラインでコンテキストをむ https://riptutorial.com/ja/dot-net/topic/5407/コンテキスト

51:

.NETの`System.String`は`System.Char`で、はUTF-16でコードされたコードです。 と.NETおよびのくのののののしのがなっているので、このはです。

1つのは 、しくとばれるはずですが、 グリフとしてされ、1つのUnicode コードポイントでされます。コードポイントは、コードユニットのシーケンスでコードされます。では、の`System.Char`がずしもをしているわけではないことがらかになるはずです。でどのようになるのかをてみましょう。

- のは、ための 、つのコードポイントをもたらしる。A 2つのコードポイントによってされているU + 0061ラテンAとU + 0300 GRAVEアクセントをみわせます 。これはもないです。なぜなら、 `"à ".Length == 2` 、あなたが1しているからです。
- したように、えばaが GRAVEまたは2つのコードポイントをするのコードポイントU + 00E0ラテンのAとすることができるため、がしています。らかに、 `"\u00e0" == "\u0061\u0300"` たとえ`"\u00e0".Length != "\u0061\u0300".Length` でもじものをしなければなりません。これは 、 `String.Normalize()`メソッドによってされるのためにです。
- えば U + D55C HANは 、 コードポイントUTF-16でのコードとしてまたはそののであってもよいため Unicodeシーケンスは、またはをんでいてもよいですそして 。それらはしいとされなければならない。
- 1つのコードポイントがのコードユニットにエンコードされているがあります。 U + 2008A HAN CHARACTERはなるコードポイントでも2つの`System.Char` `"\ud840\udc8a"` としてエンコードされますUTF-16エンコーディングはサイズではありませんたとえば、アプリケーションがをし、をにりててなをした、これはのバグなセキュリティバグのです。
- いくつかのがつグラフえばチェコのCH ととのリストをするとき、は 、あなたがケミーfyzikaをっていますに、スタンドアロンので、トリグラフを。

テキストにしては、もっとくのがあります。たとえば、でUnicodeをするにはどうすればよいですかをしてください。よりいやするへのリンクがたくさんあります。

に、 なテキストをうときは、のテキストをするためにこのなをうことができますUnicodeサロゲートやエンコーディングをさないようにする。

```
public static class StringExtensions
{
    public static IEnumerable<string> EnumerateCharacters(this string s)
    {
        if (s == null)
            return Enumerable.Empty<string>();

        var enumerator = StringInfo.GetTextElementEnumerator(s.Normalize());
        while (enumerator.MoveNext())
            yield return (string)enumerator.Value;
    }
}
```

Examples

なるをカウントする

あなたははセクションでしたため、その、のをカウントするがあるは、にすることはできません`Length`、それはのさだからプロパティを`System.Char`が、コード･ユニットないUnicodeのコードポイントではありませんグラフェムもありません。たとえば、に`text.Distinct().Count()`すると、ったがられます`text.Distinct().Count()`コードをしてください

```
int distinctCharactersCount = text.EnumerateCharacters().Count();
```

1つのステップは、パフォーマンスがではないは、このようにこのではになくにうことができるは、のをカウントすることです。

```
var frequencies = text.EnumerateCharacters()
    .GroupBy(x => x, StringComparer.CurrentCultureIgnoreCase)
    .Select(x => new { Character = x.Key, Count = x.Count() };
```

カウント

あなたはその、をカウントするがあるは、のために、それはのさだからあなたは、にLengthプロパティをすることはできません、セクションでした`System.Char`が、コードはではないではないのUnicodeコードポイントもグラフェム。しいコードはのとおりです。

```
int length = text.EnumerateCharacters().Count();
```

さなは、こののためにに`EnumerateCharacters()`メソッドをきえるかもしれません

```
public static class StringExtensions
{
    public static int CountCharacters(this string text)
    {
        if (String.IsNullOrEmpty(text))
            return 0;

        int count = 0;
        var enumerator = StringInfo.GetTextElementEnumerator(text);
        while (enumerator.MoveNext())
            ++count;

        return count;
    }
}
```

のをカウントする

セクションでされているにより、のコードユニットのをカウントしないりこれをにうことはできません

```
int count = text.Count(x => x == ch);
```

よりながです。

```
public static int CountOccurrencesOf(this string text, string character)
{
    return text.EnumerateCharacters()
        .Count(x => String.Equals(x, character, StringComparer.CurrentCulture));
}
```

ののであるとはは、ルーールにってのににされなければならないことにしてください。

をブロックに

ので、々は、のにをすことができない`System.Char`でではないがあり、それはのまたはだから、そののコーードは、アカウントにそれをるがあります さのことにしてください、はのをするものではありませコーードユニットの

```
public static IEnumerable<string> Split(this string value, int desiredLength)
{
    var characters = StringInfo.GetTextElementEnumerator(value);
    while (characters.MoveNext())
        yield return String.Concat(Take(characters, desiredLength));
}

private static IEnumerable<string> Take(TextElementEnumerator enumerator, int count)
{
    for (int i = 0; i < count; ++i)
    {
        yield return (string)enumerator.Current;

        if (!enumerator.MoveNext())
            yield break;
    }
}
```

をのエンコーーディングに/からする

.NETには、 `System.Char` UTF-16コードがまれています。のエンコーーディングでテキストをまたはするは、 `System.Byte`でするがあり`System.Byte`。

は、 `System.Text.Encoder`および`System.Text.Decoder`からしたクラスによってされます。これらのクラスは、のエンコーーディングバイトXエンコードされた`byte[]`からUTF-16でエンコードされた`System.String`およびvice -versa。

エンコーーダ/デコーーダは、ににくするため、 `System.Text.Encoding`からしたクラスでグルーープされてい`System.Text.Encoding`。 クラスはなエンコーーディングUTF-8、UTF-16などとのをします。

をUTF-8にする

```
byte[] data = Encoding.UTF8.GetBytes("This is my text");
```

UTF-8データをにする

```
var text = Encoding.UTF8.GetString(data);
```

のテキストファイルのエンコードをする

このコードは、UTF-8でエンコードされたテキストファイルのをみり、UTF-16としてエンコードしてします。このコードは、ファイルがきければすべてのコンテンツをメモリにみむのでではないことにしてください。

```
var content = File.ReadAllText(path, Encoding.UTF8);
File.WriteAllText(content, Encoding.UTF16);
```

Object.ToStringメソッド

.NETのすべてがオブジェクトであるため、すべてのはオーバーライドできる`Object`クラスでされた`ToString()`メソッドをっています。このメソッドのデフォルトのは、のをします。

```
public class Foo
{
}

var foo = new Foo();
Console.WriteLine(foo); // outputs Foo
```

`ToString()`は、valueとをconcatinatingするときににびされます。

```
public class Foo
{
    public override string ToString()
    {
        return "I am Foo";
    }
}

var foo = new Foo();
Console.WriteLine("I am bar and "+foo);// outputs I am bar and I am Foo
```

このメソッドのは、デバッグツールでもくされています。らかのでこのメソッドをオーバーライドするのではなく、 デバッガでのがどのようにされるかをカスタマイズするには、DebuggerDisplay Attribute MSDN をします。

```
// [DebuggerDisplay("Person = FN {FirstName}, LN {LastName}")]
[DebuggerDisplay("Person = FN {"+nameof(Person.FirstName)+"}, LN
```

```
{"+nameof(Person.LastName)+"}")]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set;}
    // ...
}
```

の

はです。のをすることはできません。のによって、しいをつのしいインスタンスがされます。これは、にいのの1をするがある、メモリはしいにりてられることをします。

```
string veryLongString = ...
// memory is allocated
string newString = veryLongString.Remove(0,1); // removes first character of the string.
```

でくのをするがあるは、なにされた`StringBuilder` クラスをします。

```
var sb = new StringBuilder(someInitialString);
foreach(var str in manyManyStrings)
{
    sb.Append(str);
}
var finalString = sb.ToString();
```

`String`にもかかわらず、`==`はではなくをします。

ごのとおり、 `string`はなる`string`です。しかし、のチェックとがでわれるとえると、っています。このはカルチャーですの「」をしてください。いくつかのシーケンスは、 カルチャにじてしくうことができます。

2つのの`Length` プロパティをすることにより、チェックのに2えます。

デフォルトのをするがあるは、の`StringComparison` をけれる`String.Equals` メソッドのオーバーロードをします。

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/2227/

52: /

.NET Frameworkは、くのアルゴリズムのをします。にはアルゴリズム、アルゴリズム、ハッシュなどがあります。

Examples

RijndaelManaged

なネームスペース `System.Security.Cryptography`

```
private class Encryption {

          private const string SecretKey = "topSecretKeyusedforEncryptions";

          private const string SecretIv = "secretVectorHere";

          public string Encrypt(string data) {
            return string.IsNullOrEmpty(data) ? data :
Convert.ToBase64String(this.EncryptStringToBytesAes(data, this.GetCryptographyKey(),
this.GetCryptographyIv()));
          }

          public string Decrypt(string data) {
            return string.IsNullOrEmpty(data) ? data :
this.DecryptStringFromBytesAes(Convert.FromBase64String(data), this.GetCryptographyKey(),
this.GetCryptographyIv());
          }

          private byte[] GetCryptographyKey() {
            return Encoding.ASCII.GetBytes(SecretKey.Replace('e', '!'));
          }

          private byte[] GetCryptographyIv() {
            return Encoding.ASCII.GetBytes(SecretIv.Replace('r', '!'));
          }

          private byte[] EncryptStringToBytesAes(string plainText, byte[] key, byte[] iv) {
            MemoryStream encrypt;
            RijndaelManaged aesAlg = null;
            try {
              aesAlg = new RijndaelManaged {
                Key = key,
                IV = iv
              };
              var encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
              encrypt = new MemoryStream();
              using (var csEncrypt = new CryptoStream(encrypt, encryptor,
CryptoStreamMode.Write)) {
                using (var swEncrypt = new StreamWriter(csEncrypt)) {
                  swEncrypt.Write(plainText);
                }
              }
            } finally {
              aesAlg?.Clear();
```

```
            }
            return encrypt.ToArray();
          }

          private string DecryptStringFromBytesAes(byte[] cipherText, byte[] key, byte[] iv) {
            RijndaelManaged aesAlg = null;
            string plaintext;
            try {
              aesAlg = new RijndaelManaged {
                Key = key,
                IV = iv
              };
              var decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
              using (var msDecrypt = new MemoryStream(cipherText)) {
                using (var csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read)) {
                  using (var srDecrypt = new StreamReader(csDecrypt))
                    plaintext = srDecrypt.ReadToEnd();
                }
              }
            } finally {
              aesAlg?.Clear();
            }
            return plaintext;
          }
        }
```

```
var textToEncrypt = "hello World";

 var encrypted = new Encryption().Encrypt(textToEncrypt); //-> zBmW+FUxOvdbpOGm9Ss/vQ==

 var decrypted = new Encryption().Decrypt(encrypted); //-> hello World
```

- Rijndaelは、なアルゴリズムAESのです。

AESをしてデ―タをおよびするC

```
using System;
using System.IO;
using System.Security.Cryptography;

namespace Aes_Example
{
    class AesExample
    {
        public static void Main()
        {
            try
            {
                string original = "Here is some data to encrypt!";

                // Create a new instance of the Aes class.
                // This generates a new key and initialization vector (IV).
                using (Aes myAes = Aes.Create())
                {
                    // Encrypt the string to an array of bytes.
                    byte[] encrypted = EncryptStringToBytes_Aes(original,
                                                                myAes.Key,
```

```
                                                    myAes.IV);

            // Decrypt the bytes to a string.
            string roundtrip = DecryptStringFromBytes_Aes(encrypted,
                                                          myAes.Key,
                                                          myAes.IV);

            //Display the original data and the decrypted data.
            Console.WriteLine("Original:   {0}", original);
            Console.WriteLine("Round Trip: {0}", roundtrip);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: {0}", e.Message);
    }
}

static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[] IV)
{
    // Check arguments.
    if (plainText == null || plainText.Length <= 0)
        throw new ArgumentNullException("plainText");
    if (Key == null || Key.Length <= 0)
        throw new ArgumentNullException("Key");
    if (IV == null || IV.Length <= 0)
        throw new ArgumentNullException("IV");

    byte[] encrypted;

    // Create an Aes object with the specified key and IV.
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = Key;
        aesAlg.IV = IV;

        // Create a decrytor to perform the stream transform.
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key,
                                                            aesAlg.IV);

        // Create the streams used for encryption.
        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new CryptoStream(msEncrypt,
                                                             encryptor,
                                                             CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                {
                    //Write all data to the stream.
                    swEncrypt.Write(plainText);
                }

                encrypted = msEncrypt.ToArray();
            }
        }
    }

    // Return the encrypted bytes from the memory stream.
    return encrypted;
}
```

```
        static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key, byte[] IV)
        {
            // Check arguments.
            if (cipherText == null || cipherText.Length <= 0)
                throw new ArgumentNullException("cipherText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");

            // Declare the string used to hold the decrypted text.
            string plaintext = null;

            // Create an Aes object with the specified key and IV.
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create a decrytor to perform the stream transform.
                ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key,
                                                                    aesAlg.IV);

                // Create the streams used for decryption.
                using (MemoryStream msDecrypt = new MemoryStream(cipherText))
                {
                    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt,
                                                                     decryptor,
                                                                     CryptoStreamMode.Read))
                    {
                        using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                        {

                            // Read the decrypted bytes from the decrypting stream
                            // and place them in a string.
                            plaintext = srDecrypt.ReadToEnd();
                        }
                    }
                }
            }

            return plaintext;
        }
    }
}
```

このはMSDNのものです。

これはコンソールデモアプリケーションで、の**AES**をしてをすると、でそれをするをしています。

AES = Advanced Encryption Standard 、 2001にNISTによってされたデータのであり、としてのデファクトスタンダードである

ノート

- のシナリオでは、なモードをするがあります `CipherMode`からをすることによって、 `Mode`プロ

パティにりてることができます。 `CipherMode.ECB` コードブックモードをしないでください。これはいサイファーストリームをするためです

- いくない `Key`をするには、をするか、の パスワードからをする をしてください。**KeySize**は256ビットです。サポートされているキーのサイズは、 `LegalKeySizes`プロパティから `LegalKeySizes`ます。
- ベクトル`IV`をするには、の **Random SALT** のようにSALTをし、
- サポートされているブロックサイズは`SupportedBlockSizes`プロパティでき、ブロックサイズは`BlockSize`プロパティでりてることができます

Mainメソッドをしてください。

パスワード/ランダム**SALT**からキーをする**C**

```
using System;
using System.Security.Cryptography;
using System.Text;

public class PasswordDerivedBytesExample
{
    public static void Main(String[] args)
    {
        // Get a password from the user.
        Console.WriteLine("Enter a password to produce a key:");

        byte[] pwd = Encoding.Unicode.GetBytes(Console.ReadLine());

        byte[] salt = CreateRandomSalt(7);

        // Create a TripleDESCryptoServiceProvider object.
        TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider();

        try
        {
            Console.WriteLine("Creating a key with PasswordDeriveBytes...");

            // Create a PasswordDeriveBytes object and then create
            // a TripleDES key from the password and salt.
            PasswordDeriveBytes pdb = new PasswordDeriveBytes(pwd, salt);

            // Create the key and set it to the Key property
            // of the TripleDESCryptoServiceProvider object.
            tdes.Key = pdb.CryptDeriveKey("TripleDES", "SHA1", 192, tdes.IV);

            Console.WriteLine("Operation complete.");
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            // Clear the buffers
            ClearBytes(pwd);
```

```
            ClearBytes(salt);

            // Clear the key.
            tdes.Clear();
        }

        Console.ReadLine();
    }

    #region Helper methods

    /// <summary>
    /// Generates a random salt value of the specified length.
    /// </summary>
    public static byte[] CreateRandomSalt(int length)
    {
        // Create a buffer
        byte[] randBytes;

        if (length >= 1)
        {
            randBytes = new byte[length];
        }
        else
        {
            randBytes = new byte[1];
        }

        // Create a new RNGCryptoServiceProvider.
        RNGCryptoServiceProvider rand = new RNGCryptoServiceProvider();

        // Fill the buffer with random bytes.
        rand.GetBytes(randBytes);

        // return the bytes.
        return randBytes;
    }

    /// <summary>
    /// Clear the bytes in a buffer so they can't later be read from memory.
    /// </summary>
    public static void ClearBytes(byte[] buffer)
    {
        // Check arguments.
        if (buffer == null)
        {
            throw new ArgumentNullException("buffer");
        }

        // Set each byte in the buffer to 0.
        for (int x = 0; x < buffer.Length; x++)
        {
            buffer[x] = 0;
        }
    }

    #endregion
}
```

このはMSDNからったものです。

これはコンソールのデモであり、ユーザーのパスワードについてなをすると、ランダムプログラムについてランダムなSALTをするをしています。

ノート

- みみ`PasswordDeriveBytes`は、のPBKDF1アルゴリズムをしてパスワードからキーをします。デフォルトでは、100のをしてをし、ブルートフォースをさせます。されたSALTは、キーをさらにランダムにします。
- `CryptDeriveKey`は、されたハッシュアルゴリズムここでは "SHA1"をして、`PasswordDeriveBytes`によってされたキーをされたアルゴリズムここでは "TripleDES"とのあるキーにします。このkeysizeは192バイトで、ベクトルIVはトリプルDESプロバイダからされます
- 、このメカニズムは、のデータをするパスワードによってランダムにされたなをするためにされます。また、なるユーザーののパスワードをして、じデータのランダムキーでされているにアクセスすることもできます。
- ながら、 `CryptDeriveKey`はAESをサポートしていません。 ここをしてください。
  として、AESでするデータをするランダムAESキーをし、 `CryptDeriveKey`されたキーをするTripleDESコンテナにAESキーをすることが`CryptDeriveKey`ます。しかし、これはセキュリティをTripleDESにし、AESのよりきいキーサイズをせず、TripleDESへのをします。

Mainメソッドをしてください。

AESによると

コード

```
    public static string Decrypt(string cipherText)
    {
        if (cipherText == null)
            return null;

        byte[] cipherBytes = Convert.FromBase64String(cipherText);
        using (Aes encryptor = Aes.Create())
        {
            Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(CryptKey, new byte[] { 0x49, 0x76,
0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
            encryptor.Key = pdb.GetBytes(32);
            encryptor.IV = pdb.GetBytes(16);

            using (MemoryStream ms = new MemoryStream())
            {
                using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateDecryptor(),
CryptoStreamMode.Write))
                {
                    cs.Write(cipherBytes, 0, cipherBytes.Length);
                    cs.Close();
                }

                cipherText = Encoding.Unicode.GetString(ms.ToArray());
```

```
            }
        }

        return cipherText;
    }
```

コード

```
    public static string Encrypt(string cipherText)
    {
        if (cipherText == null)
            return null;

        byte[] clearBytes = Encoding.Unicode.GetBytes(cipherText);
        using (Aes encryptor = Aes.Create())
        {
            Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(CryptKey, new byte[] { 0x49, 0x76,
0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
            encryptor.Key = pdb.GetBytes(32);
            encryptor.IV = pdb.GetBytes(16);

            using (MemoryStream ms = new MemoryStream())
            {
                using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(),
CryptoStreamMode.Write))
                {
                    cs.Write(clearBytes, 0, clearBytes.Length);
                    cs.Close();
                }

                cipherText = Convert.ToBase64String(ms.ToArray());
            }
        }
        return cipherText;
    }
```

```
var textToEncrypt = "TestEncrypt";

var encrypted = Encrypt(textToEncrypt);

var decrypted = Decrypt(encrypted);
```

オンラインで/をむ https://riptutorial.com/ja/dot-net/topic/7615/-

53: System.Text.RegularExpressions

Examples

パターンがとするかどうかをする

```
public bool Check()
{
    string input = "Hello World!";
    string pattern = @"H.ll. W.rld!";

    // true
    return Regex.IsMatch(input, pattern);
}
```

オプション

```
public bool Check()
{
    string input = "Hello World!";
    string pattern = @"H.ll. W.rld!";

    // true
    return Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase | RegexOptions.Singleline);
}
```

シンプルなマッチときえ

```
public string Check()
{
    string input = "Hello World!";
    string pattern = @"W.rld";

    // Hello Stack Overflow!
    return Regex.Replace(input, pattern, "Stack Overflow");
}
```

グループにマッチ

```
public string Check()
{
    string input = "Hello World!";
    string pattern = @"H.ll. (?<Subject>W.rld)!";

    Match match = Regex.Match(input, pattern);

    // World
    return match.Groups["Subject"].Value;
}
```

のをからする

```
public string Remove()
{
    string input = "Hello./!";

    return Regex.Replace(input, "[^a-zA-Z0-9]", "");
}
```

すべてのをつける

```
using System.Text.RegularExpressions;
```

コード

```
static void Main(string[] args)
{
    string input = "Carrot Banana Apple Cherry Clementine Grape";
    // Find words that start with uppercase 'C'
    string pattern = @"\bC\w*\b";

    MatchCollection matches = Regex.Matches(input, pattern);
    foreach (Match m in matches)
        Console.WriteLine(m.Value);
}
```

```
Carrot
Cherry
Clementine
```

オンラインでSystem.Text.RegularExpressionsをむ https://riptutorial.com/ja/dot-net/topic/6944/-system-text-regularexpressions-

54:

Examples

.Netの

JITやGCのようないくつかのは、くのプログラミングとランタイムにできるほどです。

CLRランタイム
IL
EEエンジン
JITジャストインタイムコンパイラ
GCガベージコレクタ
OOMメモリ
STAシングルスレッドアパートメント
MTAマルチスレッドアパートメント

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/10939/

55: フレームワーク

パターンをサポートするのテクノロジとして、MEFのきなの1つは、のなをとせずに、にはられていないのをサポートすることです。

すべてのでは、System.ComponentModel.Compositionアセンブリへのがです。

また、すべてのでは、これらをサンプルビジネスオブジェクトとしてしています。

```
using System.Collections.ObjectModel;

namespace Demo
{
    public sealed class User
    {
        public User(int id, string name)
        {
            this.Id = id;
            this.Name = name;
        }

        public int Id { get; }
        public string Name { get; }
        public override string ToString() => $"User[Id: {this.Id}, Name={this.Name}]";
    }

    public interface IUserProvider
    {
        ReadOnlyCollection<User> GetAllUsers();
    }
}
```

Examples

タイプのエクスポート

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel.Composition;

namespace Demo
{
    [Export(typeof(IUserProvider))]
    public sealed class UserProvider : IUserProvider
    {
        public ReadOnlyCollection<User> GetAllUsers()
        {
            return new List<User>
            {
                new User(0, "admin"),
                new User(1, "Dennis"),
                new User(2, "Samantha"),
            }.AsReadOnly();
```

```
        }
    }
}
```

これはどこでもできます。なことは、アプリケーションがするComposablePartCatalogをしてどこをすかをアプリケーションがっていることです。

インポート

```
using System;
using System.ComponentModel.Composition;

namespace Demo
{
    public sealed class UserWriter
    {
        [Import(typeof(IUserProvider))]
        private IUserProvider userProvider;

        public void PrintAllUsers()
        {
            foreach (User user in this.userProvider.GetAllUsers())
            {
                Console.WriteLine(user);
            }
        }
    }
}
```

これは`IUserProvider`にするで、どこにでもできます。のとに、アプリケーションは、するComposablePartCatalogをしてするエクスポートをすをアプリケーションがっていることだけです。

する

ののなをしてください。

```
using System.ComponentModel.Composition;
using System.ComponentModel.Composition.Hosting;

namespace Demo
{
    public static class Program
    {
        public static void Main()
        {
            using (var catalog = new ApplicationCatalog())
            using (var exportProvider = new CatalogExportProvider(catalog))
            using (var container = new CompositionContainer(exportProvider))
            {
                exportProvider.SourceProvider = container;

                UserWriter writer = new UserWriter();

                // at this point, writer's userProvider field is null
```

```
                container.ComposeParts(writer);

                // now, it should be non-null (or an exception will be thrown).
                writer.PrintAllUsers();
            }
        }
    }
}
```

アプリケーションのアセンブリパスに`[Export(typeof(IUserProvider))]`がある`UserWriter`、`UserWriter`のするインポートがたされ、ユーザーがされます。

のタイプのカタログえば、`DirectoryCatalog`は、`ApplicationCatalog`わりにまたはそれにえて、のでインポートをたすエクスポートをすためにできます。

オンラインでフレームワークをむ https://riptutorial.com/ja/dot-net/topic/62/フレームワーク

56:

ツリーは、.NET Frameworkのコードをすためにされるデータです。コードでし、プログラムでトラバースしてコードをのにしたり、したりすることができます。 Expression Treesのもなジェネレータはcコンパイラです。ラムダがExpression <Func <... >>のにりてられている、cコンパイラはツリーをできます。、これはLINQのコンテキストでします。もなコンシューマは、Entity FrameworkのLINQプロバイダです。 Entity Frameworkにえられたツリーをし、のSQLコードをし、それをデータベースにしてします。

Examples

cコンパイラでされるなツリー

のcコードをえてみましょう

```
Expression<Func<int, int>> expression = a => a + 1;
```

cコンパイラでは、ラムダがデリゲートではなくExpressionにりてられていることがわかるため、このコードとほぼのツリーがされます

```
ParameterExpression parameterA = Expression.Parameter(typeof(int), "a");
var expression = (Expression<Func<int, int>>)Expression.Lambda(
                                                Expression.Add(
                                                    parameterA,
                                                    Expression.Constant(1)),
                                                parameterA);
```

ツリーのルートは、とパラメーターのリストをむラムダです。ラムダには "a"という1つのパラメータがあります。は、CLRBinaryExpressionとAddのNodeTypeの1つのです。このはをします。との2つのがあります。 leftはパラメータ "a"のParameterExpressionで、Rightは1のConstantExpressionです。

こののもないは、それをすることです。

```
Console.WriteLine(expression); //prints a => (a + 1)
```

のcコードをします。

ツリーはcデリゲートにコンパイルされ、CLRによってされます

```
Func<int, int> lambda = expression.Compile();
Console.WriteLine(lambda(2)); //prints 3
```

、はSQLのようなのにされますが、Reflectionのわりにまたはのプライベート、プロテクト、およびメンバーをびすためにもできます。

フォームフィールド==のをする

に_ => _.Field == "VALUE"ようなをする。

_ => _.Fieldと"VALUE"がされている、がであるかどうかをテストするをします。

はのものにしています。

- IQueryable<T> 、 IEnumerable<T>をしてをテストします。
- エンティティフレームワークまたはLinq to SQLをして、をテストするWhereをしSQL。

このメソッドは、 Fieldが"VALUE"しいかどうかをテストするなEqualをします。

```
public static Expression<Func<T, bool>> BuildEqualPredicate<T>(
    Expression<Func<T, string>> memberAccessor,
    string term)
{
    var toString = Expression.Convert(Expression.Constant(term), typeof(string));
    Expression expression = Expression.Equal(memberAccessor.Body, toString);
    var predicate = Expression.Lambda<Func<T, bool>>(
        expression,
        memberAccessor.Parameters);
    return predicate;
}
```

は、 Whereメソッドにをめることによってできます。

```
var predicate = PredicateExtensions.BuildEqualPredicate<Entity>(
    _ => _.Field,
    "VALUE");
var results = context.Entity.Where(predicate).ToList();
```

フィールドをするための

のようなタイプのがあります

```
public TestClass
{
    public static string StaticPublicField = "StaticPublicFieldValue";
}
```

StaticPublicFieldのをできます。

```
var fieldExpr = Expression.Field(null, typeof(TestClass), "StaticPublicField");
var labmda = Expression.Lambda<Func<string>>(fieldExpr);
```

それは、フィールドをりすためのデリゲートにコンパイルすることができます。

```
Func<string> retriever = lambda.Compile();
var fieldValue = retriever();
```

// fieldValueのはStaticPublicFieldValueです。

InvocationExpression クラス

InvocationExpressionクラスをすると、じExpressionツリーのであるのラムダをびすことができます。

な`Expression.Invoke`メソッドでします。

たちは、そのに「」をつアイテムをにれたいとっています。でをするにnullをチェックするがありますが、がになるがあるので、それをにびたくはありません。

```
using System;
using System.Linq;
using System.Linq.Expressions;

public class Program
{
    public static void Main()
    {
        var elements = new[] {
            new Element { Description = "car" },
            new Element { Description = "cargo" },
            new Element { Description = "wheel" },
            new Element { Description = null },
            new Element { Description = "Madagascar" },
        };

        var elementIsInterestingExpression = CreateSearchPredicate(
            searchTerm: "car",
            whereToSearch: (Element e) => e.Description);

        Console.WriteLine(elementIsInterestingExpression.ToString());

        var elementIsInteresting = elementIsInterestingExpression.Compile();
        var interestingElements = elements.Where(elementIsInteresting);
        foreach (var e in interestingElements)
        {
            Console.WriteLine(e.Description);
        }

        var countExpensiveComputations = 0;
        Action incCount = () => countExpensiveComputations++;
        elements
            .Where(
                CreateSearchPredicate(
                    "car",
                    (Element e) => ExpensivelyComputed(
                        e, incCount
                    )
                ).Compile()
            )
            .Count();

        Console.WriteLine("Property extractor is called {0} times.",
countExpensiveComputations);
    }
```

```
    private class Element
    {
        public string Description { get; set; }
    }

    private static string ExpensivelyComputed(Element source, Action count)
    {
        count();
        return source.Description;
    }

    private static Expression<Func<T, bool>> CreateSearchPredicate<T>(
            string searchTerm,
            Expression<Func<T, string>> whereToSearch)
    {
        var extracted = Expression.Parameter(typeof(string), "extracted");

        Expression<Func<string, bool>> coalesceNullCheckWithSearch =
            Expression.Lambda<Func<string, bool>>(
                Expression.AndAlso(
                    Expression.Not(
                        Expression.Call(typeof(string), "IsNullOrEmpty", null, extracted)
                    ),
                    Expression.Call(extracted, "Contains", null,
Expression.Constant(searchTerm))
                ),
                extracted);

        var elementParameter = Expression.Parameter(typeof(T), "element");

        return Expression.Lambda<Func<T, bool>>(
            Expression.Invoke(
                coalesceNullCheckWithSearch,
                Expression.Invoke(whereToSearch, elementParameter)
            ),
            elementParameter
        );
    }
}
```

```
element => Invoke(extracted => (Not(IsNullOrEmpty(extracted)) AndAlso
extracted.Contains("car")), Invoke(e => e.Description, element))
car
cargo
Madagascar
Predicate is called 5 times.
```

にすべきは、Invokeでラップされたのアクセスのアクセスです。

```
Invoke(e => e.Description, element)
```

これは`e.Description`にれるのであり、そのわりに`string`の`extracted`パラメ―タがのものにされ`string`

```
(Not(IsNullOrEmpty(extracted)) AndAlso extracted.Contains("car"))
```

ここですべきもうつのなことは、 `AndAlso`です。のが 'false'をしたは、のみをします。ビット 'And'をするのはよくあるいです。このはにのをし、このではNullReferenceExceptionでします。

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/2657/

57:

Examples

.NET 1.xのConfigurationSettingsからのAppSettings

された

ConfigurationSettingsクラスは、.NET 1.0および1.1のアセンブリのをするのでした。これは、ConfigurationManagerクラスとWebConfigurationManagerクラスによってきえられました。

ファイルの`appSettings`セクションにじのキーが2つあるは、のものがされます。

app.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="keyName" value="anything, as a string"/>
    <add key="keyNames" value="123"/>
    <add key="keyNames" value="234"/>
  </appSettings>
</configuration>
```

Program.cs

```
using System;
using System.Configuration;
using System.Diagnostics;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            string keyValue = ConfigurationSettings.AppSettings["keyName"];
            Debug.Assert("anything, as a string".Equals(keyValue));

            string twoKeys = ConfigurationSettings.AppSettings["keyNames"];
            Debug.Assert("234".Equals(twoKeys));

            Console.ReadKey();
        }
    }
}
```

.NET 2.0でConfigurationManagerからAppSettingsをみむ

ConfigurationManagerクラスは`AppSettings`プロパティをサポートして`AppSettings`ます。これによ

り、.NET 1.xとじでファイルの`appSettings`セクションからをみけることができます。

app.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="keyName" value="anything, as a string"/>
    <add key="keyNames" value="123"/>
    <add key="keyNames" value="234"/>
  </appSettings>
</configuration>
```

Program.cs

```
using System;
using System.Configuration;
using System.Diagnostics;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            string keyValue = ConfigurationManager.AppSettings["keyName"];
            Debug.Assert("anything, as a string".Equals(keyValue));

            var twoKeys = ConfigurationManager.AppSettings["keyNames"];
            Debug.Assert("234".Equals(twoKeys));

            Console.ReadKey();
        }
    }
}
```

Visual Studioのなけされたアプリケーションとユーザーのサポートの

Visual Studioでは、ユーザーとアプリケーションのをできます。このをすると、ファイルの`appSettings`セクションをすることにべて、これらのがられます。

1. をくすることができます。シリアルできるタイプは、としてできます。
2. アプリケーションのは、ユーザーからにりすことができます。 アプリケーションは、のファイルにされている`web.config`のWebサイトやWebアプリケーションのための、およびapp.configを、 .exe.configと、 アセンブリがファイルのです。ユーザーWebプロジェクトではされませんは、ユーザーのApplication Dataフォルダオペレーティングシステムのバージョンによってなりますの`user.config`ファイルにされます。
3. クラスライブラリにはのカスタムセクションがあるので、クラスライブラリのアプリケーションをののなしにのファイルにまとめることができます。

ほとんどのプロジェクトタイプでは、 プロジェクトプロパティデザイナには、カスタムアプリケ

—ションとユーザーをするためのである[ ]タブがあります。では、[]タブはになり、デフォルトのファイルをするためのリンクが1つされます。リンクをクリックすると、のがわれます。

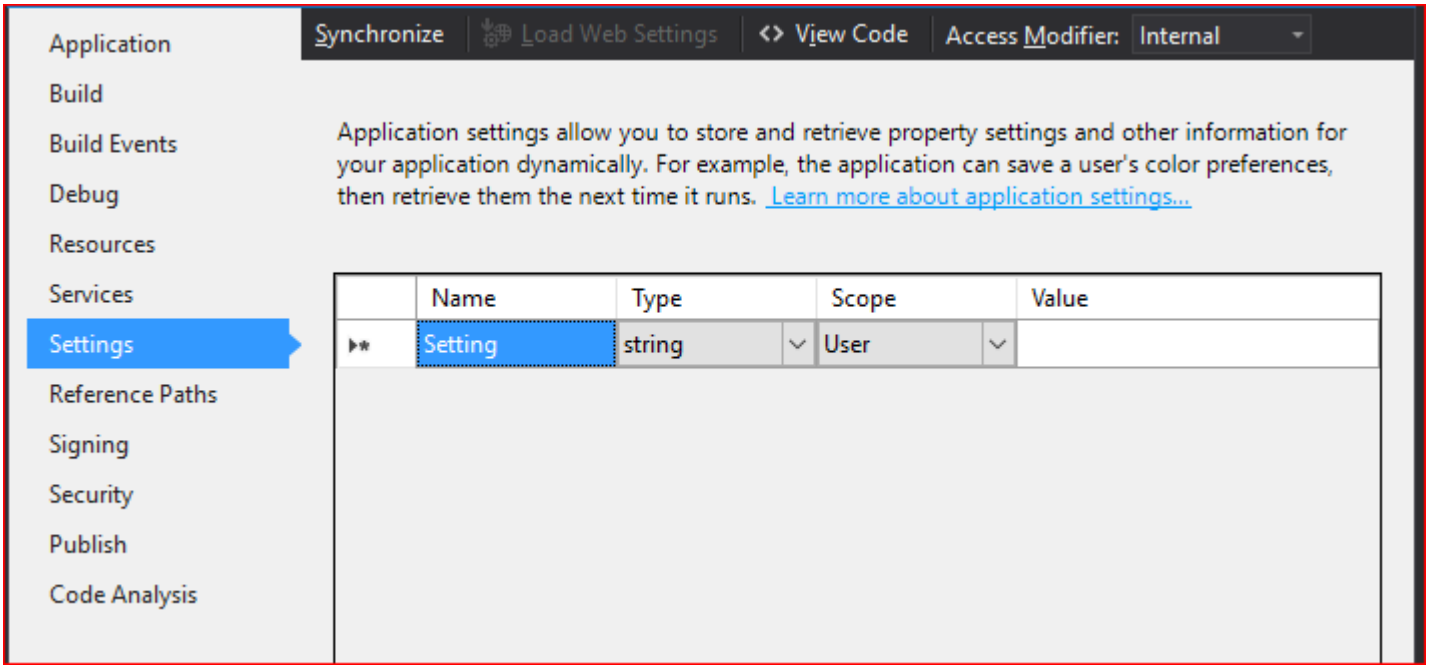
1. プロジェクトのファイル `app.config`または`web.config` がしないは、そのファイルがされます。
2. []タブは、々のエントリの、、およびをにするグリッドコントロールにきえられます。
3. ソリューションエクスプローラで、プロパティのなフォルダのに`Settings.settings`アイテムがされます。このをくと、[]タブがきます。
4. しいクラスをむしいファイルがプロジェクトフォルダの`Properties`フォルダにされます。このしいファイルのは`Settings.Designer.__`.cs、.vbなどで、クラスのは`Settings`です。クラスはコードされているのでするはありませんが、クラスはクラスであるため、のメンバーをのファイルにれてクラスをできます。さらに、このクラスは、プロパティのでシングルトンインスタンスをさせ、Singletonパターンをしてされる`Default`。

しいエントリを[]タブにすると、Visual Studioはの2つのことをいます。

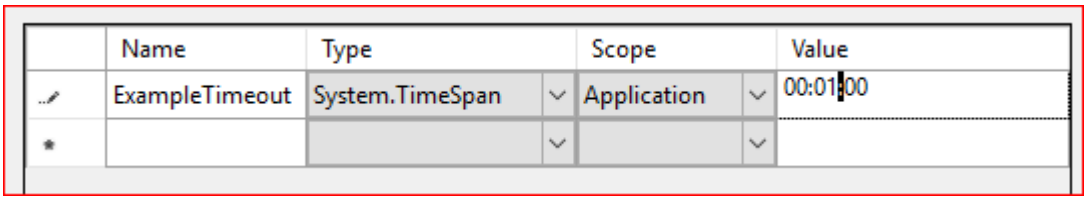
1. Settingsクラスによってされるようにされたカスタムセクションで、ファイルにをします。
2. Settingsクラスでしいメンバーをし、[]タブでしたのタイプのをみきしてします。

にけされたをファイルのカスタムセクションからみむ

しいSettingsクラスとカスタムセクションから



ExampleTimeoutというのアプリケーションをtime System.Timespanをしてし、を1にします。



タブのエントリをし、カスタムクラスをし、プロジェクトファイルをするプロジェクトプロパティをします。

コードCのをします。

Program.cs

```
using System;
using System.Diagnostics;
using ConsoleApplication1.Properties;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            TimeSpan exampleTimeout = Settings.Default.ExampleTimeout;
            Debug.Assert(TimeSpan.FromMinutes(1).Equals(exampleTimeout));

            Console.ReadKey();
        }
    }
}
```

カバーのに

プロジェクトファイルをて、アプリケーションエントリがどのようにされているかをします。

app.config Visual Studioによりにされます

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" >
      <section name="ConsoleApplication1.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
    </sectionGroup>
  </configSections>
  <appSettings />
  <applicationSettings>
    <ConsoleApplication1.Properties.Settings>
      <setting name="ExampleTimeout" serializeAs="String">
        <value>00:01:00</value>
      </setting>
    </ConsoleApplication1.Properties.Settings>
```

```
  </applicationSettings>
</configuration>
```

appSettingsセクションはされないことにしてください。 applicationSettingsセクションには、エントリのsettingをつカスタムセクションがまれています。のタイプはファイルにはされません。これはSettingsクラスのみがSettingsます。

Settingsクラスをて、このカスタムセクションをむためにConfigurationManagerクラスをどのようにするかをしてください。

Settings.designer.cs Cプロジェクトの

```
...
    [global::System.Configuration.ApplicationScopedSettingAttribute()]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Configuration.DefaultSettingValueAttribute("00:01:00")]
    public global::System.TimeSpan ExampleTimeout {
        get {
            return ((global::System.TimeSpan)(this["ExampleTimeout"]));
        }
    }
...
```

DefaultSettingValueAttributeは、プロジェクトプロパティデザイナの[]タブにされたをするためにされたことにしてください。エントリがファイルにない、わりにこのデフォルトがされます。

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/54/

58:

Examples

の

ディクショナリを3つののいずれかでできます。

KeyValueペアの

```
Dictionary<int, string> dict = new Dictionary<int, string>();
foreach(KeyValuePair<int, string> kvp in dict)
{
   Console.WriteLine("Key : " + kvp.Key.ToString() + ", Value : " + kvp.Value);
}
```

キーの

```
Dictionary<int, string> dict = new Dictionary<int, string>();
foreach(int key in dict.Keys)
{
    Console.WriteLine("Key : " + key.ToString() + ", Value : " + dict[key]);
}
```

の

```
Dictionary<int, string> dict = new Dictionary<int, string>();
foreach(string s in dict.Values)
{
    Console.WriteLine("Value : " + s);
}
```

コレクションをしたディクショナリの

```
// Translates to `dict.Add(1, "First")` etc.
var dict = new Dictionary<int, string>()
{
    { 1, "First" },
    { 2, "Second" },
    { 3, "Third" }
};

// Translates to `dict[1] = "First"` etc.
// Works in C# 6.0.
var dict = new Dictionary<int, string>()
{
    [1] = "First",
    [2] = "Second",
    [3] = "Third"
};
```

にする

```
Dictionary<int, string> dict = new Dictionary<int, string>();
dict.Add(1, "First");
dict.Add(2, "Second");

// To safely add items (check to ensure item does not already exist - would throw)
if(!dict.ContainsKey(3))
{
   dict.Add(3, "Third");
}
```

あるいは、インデクサをして/することもできます。 インデクサはにプロパティのようにえ、getとsetをちますが、でまれたののパラメ―タをとります

```
Dictionary<int, string> dict = new Dictionary<int, string>();
dict[1] = "First";
dict[2] = "Second";
dict[3] = "Third";
```

をスロ―するAddメソッドとはなり、キ―がににまれている、インデクサはのをきえます。

スレッドセ―フなConcurrentDictionary<TKey, TValue>

```
var dict = new ConcurrentDictionary<int, string>();
dict.AddOrUpdate(1, "First", (oldKey, oldValue) => "First");
```

からをる

このセットアップコ―ドをえてみましょう

```
var dict = new Dictionary<int, string>()
{
    { 1, "First" },
    { 2, "Second" },
    { 3, "Third" }
};
```

キ―1があるエントリのをみることができます。キ―がしないは、 KeyNotFoundExceptionがスロ―さContainsKey。 そのため、まずContainsKeyしてそのをチェックしContainsKey。

```
if (dict.ContainsKey(1))
    Console.WriteLine(dict[1]);
```

これにはが1つあります。を2しますをするために1、をみるために1。きなの、これはパフォ―マンスにするがあります。いにも、のをにすることができます

```
string value;
if (dict.TryGetValue(1, out value))
    Console.WriteLine(value);
```

をる **Case-Insensivitve**キーをします。

```
var MyDict = new Dictionary<string,T>(StringComparison.InvariantCultureIgnoreCase)
```

ConcurrentDictionary .NET 4.0より

のスレッドがにアクセスできるキーとのペアのスレッドセーフなコレクションをします。

インスタンスの

インスタンスのは`Dictionary<TKey, TValue>`とほぼじです。

```
var dict = new ConcurrentDictionary<int, string>();
```

または

`Add`メソッドがないの`Add`くかもしれませんが、わりに2つのオーバーロードをつ`AddOrUpdate`があり`AddOrUpdate`

1 `AddOrUpdate(TKey key, TValue, Func<TKey, TValue, TValue> addValue)` - キーがしないはキー/のペアをし、キーがするはされたをしてキー/もうしている。

2 `AddOrUpdate(TKey key, Func<TKey, TValue> addValue, Func<TKey, TValue, TValue> updateValueFactory)` - されたをして、キーがしない、またはキーがすでにするは、キーとのペアをします。

されたkey1にすでにしていた、がであってもをまたはする

```
string addedValue = dict.AddOrUpdate(1, "First", (updateKey, valueOld) => "First");
```

をまたはするが、の1につぃてのをする

```
string addedValue2 = dict.AddOrUpdate(1, "First", (updateKey, valueOld) => $"{valueOld}
Updated");
```

overload2をして、ファクトリをしてしいをすることもできます。

```
string addedValue3 = dict.AddOrUpdate(1, (key) => key == 1 ? "First" : "Not First",
(updateKey, valueOld) => $"{valueOld} Updated");
```

をる

をすることは、 `Dictionary<TKey,TValue>`とじです。

```
string value = null;
bool success = dict.TryGetValue(1, out value);
```

のまたは

スレッドセ―フなでをまたはする2つのメソッドオ―バ―ロ―ドがあります。

キ―2でをするか、キ―がしないに「」をします。

```
string theValue = dict.GetOrAdd(2, "Second");
```

がしない、ファクトリをしてをする

```
string theValue2 = dict.GetOrAdd(2, (key) => key == 2 ? "Second" : "Not Second." );
```

IEnumerableからDictionary≧.NET 3.5

IEnumerable <T>からDictionary <TKey、TValue>をします。

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
public class Fruits
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

```
var fruits = new[]
{
    new Fruits { Id = 8 , Name = "Apple" },
    new Fruits { Id = 3 , Name = "Banana" },
    new Fruits { Id = 7 , Name = "Mango" },
};


// Dictionary<int, string>                  key      value
var dictionary = fruits.ToDictionary(x => x.Id, x => x.Name);
```

からの

このセットアップコ―ドをえてみましょう

```
var dict = new Dictionary<int, string>()
{
    { 1, "First" },
    { 2, "Second" },
    { 3, "Third" }
```

```
};
```

キーとそのをするには、 `Remove`メソッドをします。

```
bool wasRemoved = dict.Remove(2);
```

このコードをすると、キー`2`とそのがからされます。 `Remove`は、されたキーがされてからされたかどうかをすブールをします。キーがディクショナリにしない、からもされず、falseがされますはスローされません。

キーのを`null`してキーをしようとするのはりです。

```
dict[2] = null; // WRONG WAY TO REMOVE!
```

これでキーはされません。のを`null`にきえ`null`。

からすべてのキーとをするには、 `Clear`メソッドをし`Clear`。

```
dict.Clear();
```

`Clear`すると、の`Count`は0になりますが、はされません。

ContainsKeyTKey

`Dictionary`にキーがあるかどうかをするには、 `ContainsKey(TKey)`メソッドをびして、 `TKey`タイプのキーをします。このメソッドは、キーがにするときに`bool`をします。サンプル

```
var dictionary = new Dictionary<string, Customer>()
{
   {"F1", new Customer() { FirstName = "Felipe", ... } },
   {"C2", new Customer() { FirstName = "Carl", ... } },
   {"J7", new Customer() { FirstName = "John", ... } },
   {"M5", new Customer() { FirstName = "Mary", ... } },
};
```

また、に`C2`がするかどうかをします。

```
if (dictionary.ContainsKey("C2"))
{
   // exists
}
```

ContainsKeyメソッドは、`Dictionary<TKey, TValue>`できます。

リストをに

KeyValuePairのリストをする

```
Dictionary<int, int> dictionary = new Dictionary<int, int>();
List<KeyValuePair<int, int>> list = new List<KeyValuePair<int, int>>();
list.AddRange(dictionary);
```

キ―のリストをする

```
Dictionary<int, int> dictionary = new Dictionary<int, int>();
List<int> list = new List<int>();
list.AddRange(dictionary.Keys);
```

のリストをする

```
Dictionary<int, int> dictionary = new Dictionary<int, int>();
List<int> list = new List<int>();
list.AddRange(dictionary.Values);
```

ConcurrentDictionaryをLazy'1ですると、したがする

ConcurrentDictionaryは、キャッシュからのキ―をにすことになります。ほとんどの、ロックがなく、かいレベルでします。しかし、オブジェクトのがにで、コンテキストりえのコストをり、キャッシュミスがするはどうでしょうか

のスレッドからじキ―がされた、にするオブジェクトの1つがコレクションにされ、りのオブジェクトがされ、オブジェクトをするCPUリソ―スとオブジェクトをにするメモリリソ―スがになります。のリソ―スもになるがあります。これはにいです。

`ConcurrentDictionary<TKey, TValue>`と`Lazy<TValue>`をみわせることができます。アイデアは、ConcurrentDictionaryのGetOrAddメソッドは、にコレクションにされただけをすことができるということです。 Lazyオブジェクトは、このもになるがありますが、Lazyオブジェクトがであるため、あまりにはなりません。にコレクションにされたValueプロパティGetOrAddメソッドからされたもののみをするのはなので、うLazyのValueプロパティはされません。

```
public static class ConcurrentDictionaryExtensions
{
    public static TValue GetOrCreateLazy<TKey, TValue>(
        this ConcurrentDictionary<TKey, Lazy<TValue>> d,
        TKey key,
        Func<TKey, TValue> factory)
    {
        return
            d.GetOrAdd(
                key,
                key1 =>
                    new Lazy<TValue>(() => factory(key1),
                    LazyThreadSafetyMode.ExecutionAndPublication)).Value;
    }
}
```

XmlSerializerオブジェクトのキャッシュは、にになることがあります。また、アプリケ―ションのにくのがします。そして、これのことがありますそれらがカスタムシリアライザである、りの

プロセスライフサイクルでもメモリリークがします。このConcurrentDictionaryののは、プロセスライフサイクルのりのではロックはありませんが、アプリケーションのとメモリのはできないということです。これは、ConcurrentDictionaryのであり、Lazy

```
private ConcurrentDictionary<Type, Lazy<XmlSerializer>> _serializers =
    new ConcurrentDictionary<Type, Lazy<XmlSerializer>>();

public XmlSerializer GetSerialier(Type t)
{
    return _serializers.GetOrCreateLazy(t, BuildSerializer);
}

private XmlSerializer BuildSerializer(Type t)
{
    throw new NotImplementedException("and this is a homework");
}
```

オンラインでをむ https://riptutorial.com/ja/dot-net/topic/45/

59: のとIProgress

Examples

シンプルプログレスレポート

`IProgress<T>`は、あるプロシージャのをのプロシージャにするためにできます。このは、をするなメソッドをするをしています。

```
void Main()
{
    IProgress<int> p = new Progress<int>(progress =>
    {
        Console.WriteLine("Running Step: {0}", progress);
    });
    LongJob(p);
}

public void LongJob(IProgress<int> progress)
{
    var max = 10;
    for (int i = 0; i < max; i++)
    {
        progress.Report(i);
    }
}
```

```
Running Step: 0
Running Step: 3
Running Step: 4
Running Step: 5
Running Step: 6
Running Step: 7
Running Step: 8
Running Step: 9
Running Step: 2
Running Step: 1
```

このコードをすると、ができされることがあります。これは、 `IProgress<T>.Report()`メソッドがにされるため、をにするがあるにしていないためです。

IProgressの

`System.Progress<T>`クラスでは、 `Report()`メソッドをできないことにしてください。このメソッドは`IProgress<T>`インターフェイスからにされているため、 `IProgress<T>`キャストされるときに`Progress<T>`でびされるがあります。

```
var p1 = new Progress<int>();
p1.Report(1); //compiler error, Progress does not contain method 'Report'
```

```
IProgress<int> p2 = new Progress<int>();
p2.Report(2); //works

var p3 = new Progress<int>();
((IProgress<int>)p3).Report(3); //works
```

オンラインでのとIProgress をむ https://riptutorial.com/ja/dot-net/topic/5628/の-t-とiprogress--t-

クレジット

S. No		Contributors
1	.NET Frameworkをいめる	Adriano Repetti, Alan McBee, ale10ander, Andrew Jens, Andrew Morton, Andrey Shchekin, Community, Daniel A. White, Ehsan Sajjad, harriyott, hillary.fraley, Ian, James Thorpe, Jamie Rees, Joel Martinez, Kevin Montrose, Lirrik, MarcinJuraszek, matteeyah, naveen, Nicholas Sizer, Paweł Izdebski, Peter, Peter Gordon, Peter Hommel, PSN, Richard Lander, Rion Williams, Robert Columbia, RubberDuck, SeeuD1, Serg Rogovtsev, Squidward, Stephen Leppik, Steven Doggart, svick, ɬolɐəz əɥɬ qoq
2	.NETコア	Mihail Stancescu
3	.NETフレームワークをした	Yahfoufi
4	ADO.NET	Akshay Anand, Andrew Morton, Daniel A. White, DavidG, Drew, elmer007, Hamid, Harjot, Heinzi, Igor, user2321864
5	ASP.NETのスマートなをしたASP.NET MVCのグローバリゼーション	Scott Hannen
6	CでSHA1をする	mahdi abasi
7	CLR	Gajendra, starbeamrainbowlabs, Theodoros Chatzigiannakis
8	DateTimeの	GalacticCowboy, John
9	ForEach	Dr Rob Lang, just.ru, Lucas Trzesniewski
10	HTTPクライアント	CodeCaster, Konamiman, MuiBienCarlota
11	HTTPサーバー	Devon Burriss, Konamiman
12	JITコンパイラ	Krikor Ailanjian
13	JSON with .NET with Newtonsoft.Json	DLeh

14	JSONシリアル	Akshay Anand, Andrius, Eric, hasan, M22an, PedroSouki, Thriggle, Tolga Evcimen
15	LINQ	A. Raza, Adil Mammadov, Akshay Anand, Alexander V., Benjamin Hodgson, Blachshma, Bradley Grainger, Bruno Garcia, Carlos Muñoz, CodeCaster, dbasnett, DoNot , dotctor, Eduardo Molteni, Ehsan Sajjad, GalacticCowboy, H. Pauwelyn, Haney, J3soon, jbtule, jnovo, Joe Amenta, Kilazur, Konamiman, MarcinJuraszek, Mark Hurd, McKay, Mellow, Mert Gülsoy, Mike Stortz, Mr.Mindor, Nate Barbettini, Pavel Voronin, Ruben Steins, Salvador Rubio Martinez, Sammi, Sergio Domínguez, Sidewinder94
16	NuGetパッケージングシステム	Andrey Shchekin, Anik Saha, Ashtonian, CodeCaster, Daniel A. White, Matas Vaitkevicius, Ozair Kafray
17	ReadOnlyCollections	tehDorf
18	StdErrストリームへのきみとみし	Aleks Andreev
19	System.Diagnostics	Adi Lester, Bassie, Fredou, Ogglas, Ondřej Štorc, RamenChef
20	System.IO	CodeCaster, Daniel A. White, demonplus, Filip Frącz, RoyalPotato
21	System.IO.Fileクラス	Adriano Repetti, delete me
22	System.Net.Mail	demonplus, Steve, vicky
23	System.Reflection.Emit	Luaan, NikolayKondratyev, RamenChef, toddmo
24	System.Runtime.Caching.MemoryCache ObjectCache	Guanxi, RamenChef
25	TPLデータフロー	i3arnon, Jacobr365, Nikola.Lukovic, RamenChef
26	VBフォーム	ale10ander, dbasnett
27	XmlSerializer	Aphelion, George Polevoy, RamenChef, Rowland Shaw, Thomas Levesque, void, Yogi

28	Zipファイルのみき	Arxae
29	カスタムタイプ	Alan McBee, DrewJordan, matteeyah
30	ガベージコレクション	avat
31	コード	JJS, Matthew Whited, RamenChef
32	コレクション	Alan McBee, Aman Sharma, Anik Saha, Daniel A. White, demonplus, Felipe Oriani, harriyott, Ian, Mark C., Ravi A., Virtlink
33	シリアルポート	Dmitry Egorov
34	スタックとヒープ	Hywel Rees
35	スピーチをするSpeechRecognitionEngineクラス	ProgramFOX, RamenChef
36	スレッディング	Behzad, Martijn Pieters, Mellow
37	タスクライブラリTPL	Adi Lester, Aman Sharma, Andrew, i3arnon, Jacobr365, JamyRyals, Konamiman, Mathias Müller, Mert Gülsoy, Mikhail Filimonov, Pavel Mayorov, Pavel Voronin, RamenChef, Thomas Bledsoe, TorbenJ
38	タスクライブラリTPLAPIの	Gusdor, Jacobr365
39	ネットワーキング	Konamiman
40	ファイルとPOSTデータをWebサーバーにアップロードする	Aleks Andreev
41	ファイルの	ale10ander, Alexander Mandt, Ingenioushax, Nitram
42	プラットフォームびし	Dmitry Egorov, Imran Ali Khan
43	プロセスとスレッドの	MSE, RamenChef
44	メモリ	Big Fan, binki, DrewJordan
45		Adi Lester, Akshay Anand, Alan McBee, Alfred Myers, Arvin Baccay, BananaSft, CodeCaster, Dave R., Kritner, Mafii, Matt, Rob, Sean, starbeamrainbowlabs, STW, Yousef Al-Mulla

46		Phil Thomas, Scott Hannen
47	テスト	Axarydax
48		Aleks Andreev, Bjørn-Roger Kringsjå, demonplus, Jean-Baptiste Noblot, Jigar, JJP , Kirk Broadhurst, Lorenzo Dematté, Matas Vaitkevicius, NetSquirrel, Pavel Mayorov, Peter, smdrager, Terry, user1304444, void
49	コンテキスト	DLeh, Gusdor
50		Adriano Repetti, Alexander Mandt, Matt, Pavel Voronin, RamenChef
51	/	Alexander Mandt, Daniel A. White, demonplus, Jagadisha B S, lokusking, Matt
52	System.Text.RegularExpressions	BrunoLM, Denuath, Matt dc, tehDorf
53		Tanveer Badar
54	フレームワーク	Joe Amenta, Kirk Broadhurst, RamenChef
55		Akshay Anand, George Polevoy, Jim, n.podbielski, Pavel Mayorov, RamenChef, Stephen Leppik, Stilgar, wangengzheng
56		Alan McBee
57		Adriano Repetti, Bjørn-Roger Kringsjå, Daniel Plaisted, Darrel Lee, Felipe Oriani, George Duckett, George Polevoy, hatchet, Hogan, Ian, LegionMammal978, Luke Bearl, Olivier Jacot-Descombes, RamenChef, Ringil, Robert Columbia, Stephen Byrne, the berserker, Tomáš Hübelbauer
58	のとIProgress	DLeh