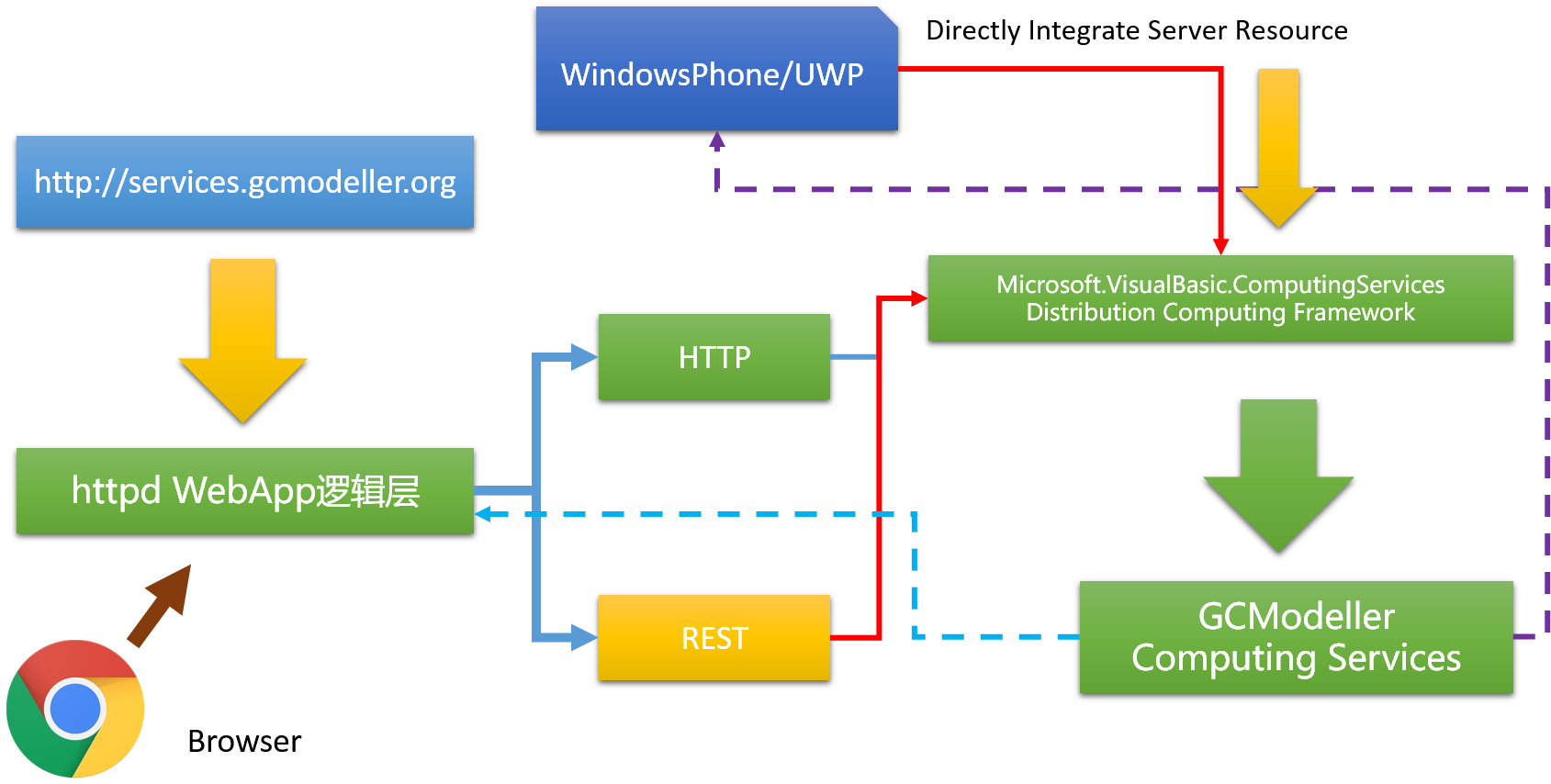
The new reconstruct of the online trading system on the <http://mipaimai.com/> and building the systems biology online annotation system for <http://services.gcmodeller.org/> GCModeller Virtual Cell System required of a parallel library for the distribution computing framework for processing the large amount of the data.



Latest source code on github:

<https://github.com/xieguigang/Microsoft.VisualBasic.Parallel.git>

## How to invoke a function in VisualBasic?

Reflection parts of the invokes can be reviewed at Shoal language article: <http://www.codeproject.com/Articles/820854/Powerful-ShellScript-for-bioinformatics-researcher>

The steps of the reflection way to invoke a method in VisualBasic can be summary as:

1. Get function pointer by using AddressOf operator or lambda expression
2. Gets the MethodInfo from the delegate
3. Using MethodInfo.Invoke to invoke the target function.

Needs 3 basically requirements:

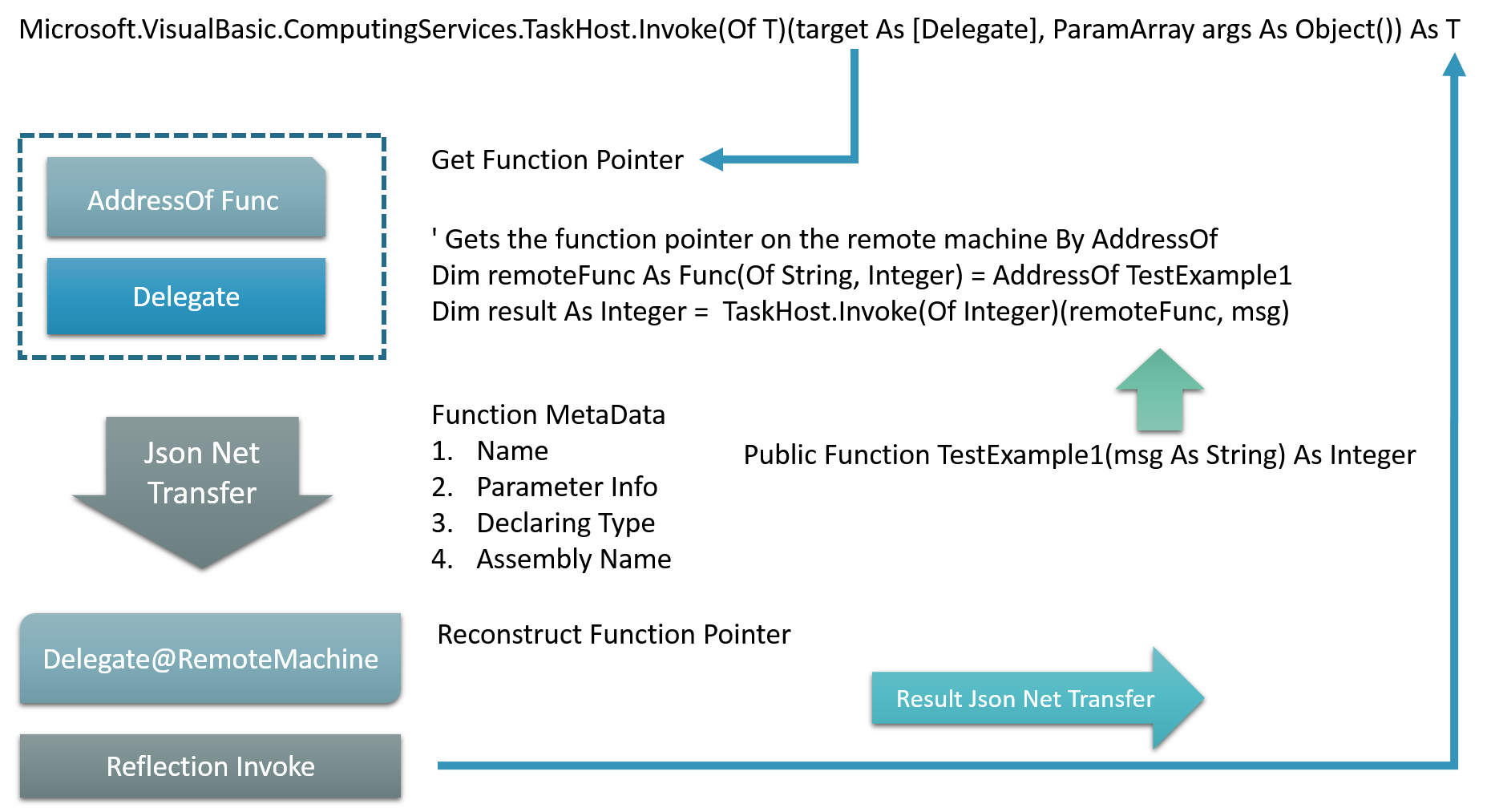
1. TypeInfo

Assembly and type

1. Method Name
2. Method Parameters

And if the target method have several overloads edition, then the parameter is required for get the corrects edition of your method.

## Remote Function



A basically schema diagram of the Microsoft.VisualBasic.ComputingServices Parallel library.

### Invoke on the remote machine

''' <summary>

''' A common function of invoke the method on the remote machine

''' </summary>

''' <param name="params">远程主机上面的函数指针</param>

''' <param name="value">value's <see cref="system.type"/></param>

''' <returns></returns>

Private Shared Function \_\_invoke(params As InvokeInfo, ByRef value As Type) As Object

Dim func As MethodInfo = params.GetMethod

Dim paramsValue As Object() = InvokeInfo.GetParameters(func, params.Parameters)

Dim x As Object = func.Invoke(Nothing, paramsValue)

value = func.ReturnType

Return x

End Function

### Remote Invoke Protocol

<Protocol(TaskProtocols.Invoke)>

Private Function Invoke(CA As Long, args As RequestStream, remote As System.Net.IPEndPoint) As RequestStream

Dim params As InvokeInfo = Serialization.LoadObject(Of InvokeInfo)(args.GetUTF8String)

Dim value As Rtvl = Invoke(params)

Return New RequestStream(value.GetJson)

End Function

''' <summary>

''' Invoke the function on the remote server.(远程服务器上面通过这个方法执行函数调用)

''' </summary>

''' <param name="params"></param>

''' <returns></returns>

Public Shared Function Invoke(params As InvokeInfo) As Rtvl

Dim rtvl As Rtvl

Try

Dim rtvlType As Type = Nothing

Dim value As Object = \_\_invoke(params, rtvlType)

rtvl = New Rtvl(value, rtvlType)

Catch ex As Exception

ex = New Exception(params.GetJson, ex)

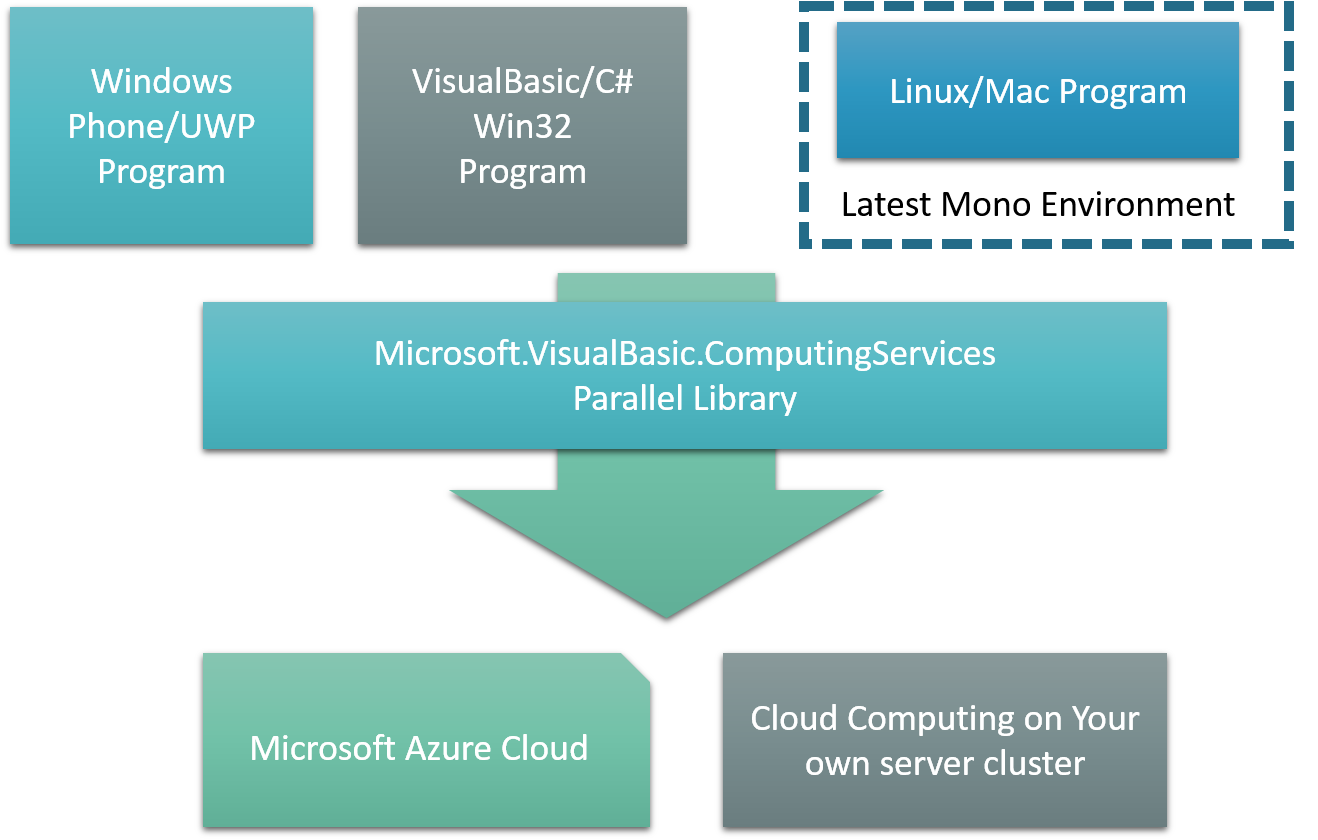
rtvl = New Rtvl(ex)

End Try

Return rtvl

End Function

### Build Your Own Cloud Computing platform



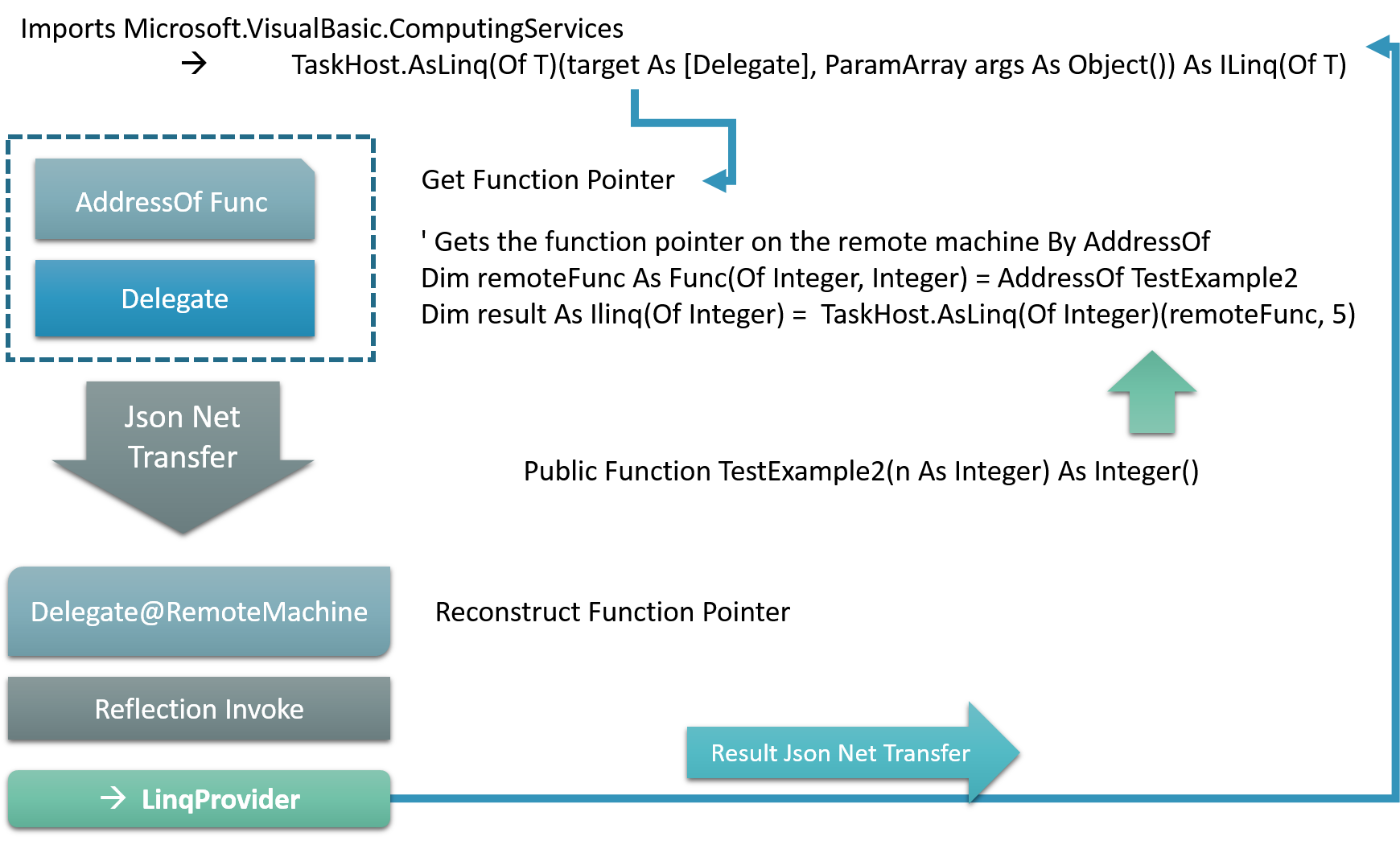
Base on the Parallel library in this article, then you can try build your own cloud computing platform like Microsoft Azure, you can programming to fully access the remote server resource just like running on the local machine.

One example of the mobile cloud computing is the Microsoft bioinformatics scientific project .NET Bio, a mobile library was developed that the developer can integrated their WindowsPhone/Android/IOS App with the NCBI cloud computing BLAST services to analysis the biological sequence data size up to 10GB or 100GB.

By using the cloud computing, this can integrated your server machine resource into the Windows Phone or UWP program directly, then this makes using your Windows Phone processing very large data set possible.

## Remote Linq

The linq expression is much useful in the VisualBasic programming.



### Remote source

#### Generic iterator

For implements a LINQ source an iterator is required:

Imports System.Threading

Imports System.Threading.Thread

Namespace ComponentModel.DataSourceModel

''' <summary>

''' Implements for the IEnumerable(Of T), Supports a simple iteration over a non-generic collection.

''' </summary>

Public Class Iterator : Implements IEnumerator

Implements IDisposable

ReadOnly \_source As IEnumerable

Sub New(source As IEnumerable)

\_source = source

Reset()

End Sub

''' <summary>

''' Gets the current element in the collection.

''' </summary>

''' <returns></returns>

Public ReadOnly Property Current As Object Implements IEnumerator.Current

Public ReadOnly Property ReadDone As Boolean

Dim \_read As Boolean = False

Private Sub \_\_moveNext()

\_ReadDone = False

' Single thread safely

For Each x As Object In \_source ' 单线程安全

Do While \_read

Call Sleep(1)

Loop

\_Current = x

\_read = True

Next

\_read = False

\_ReadDone = True

End Sub

Dim \_forEach As Thread

''' <summary>

''' Sets the enumerator to its initial position, which is before the first element in the collection.

''' </summary>

Public Sub Reset() Implements IEnumerator.Reset

If Not \_forEach Is Nothing Then ' 终止这条线程然后再新建

Call \_forEach.Abort()

End If

\_forEach = New Thread(AddressOf \_\_moveNext)

\_forEach.Start()

End Sub

''' <summary>

''' Advances the enumerator to the next element of the collection.

''' </summary>

''' <returns>

''' true if the enumerator was successfully advanced to the next element; false if the enumerator has passed the end of the collection.

''' </returns>

Public Function MoveNext() As Boolean Implements IEnumerator.MoveNext

\_read = False

Return Not ReadDone

End Function

#Region "IDisposable Support"

Private disposedValue As Boolean ' To detect redundant calls

' IDisposable

Protected Overridable Sub Dispose(disposing As Boolean)

If Not Me.disposedValue Then

If disposing Then

' TODO: dispose managed state (managed objects).

Call \_forEach.Abort()

Call \_forEach.Free

End If

' TODO: free unmanaged resources (unmanaged objects) and override Finalize() below.

' TODO: set large fields to null.

End If

Me.disposedValue = True

End Sub

' TODO: override Finalize() only if Dispose(disposing As Boolean) above has code to free unmanaged resources.

'Protected Overrides Sub Finalize()

' ' Do not change this code. Put cleanup code in Dispose(disposing As Boolean) above.

' Dispose(False)

' MyBase.Finalize()

'End Sub

' This code added by Visual Basic to correctly implement the disposable pattern.

Public Sub Dispose() Implements IDisposable.Dispose

' Do not change this code. Put cleanup code in Dispose(disposing As Boolean) above.

Dispose(True)

' TODO: uncomment the following line if Finalize() is overridden above.

' GC.SuppressFinalize(Me)

End Sub

#End Region

End Class

End Namespace

Due to the reason of the remote linq is usually using for processing the large size data set

#### Linq source provider

### Linq local reader

''' <summary>

''' Remote LINQ source reader

''' </summary>

''' <typeparam name="T"></typeparam>

Public Class ILinq(Of T) : Implements IEnumerable(Of T)

Implements IDisposable

''' <summary>

''' Element type in the source collection.

''' </summary>

''' <returns></returns>

Public ReadOnly Property Type As Type = GetType(T)

''' <summary>

''' Remote entry point

''' </summary>

''' <returns></returns>

Public ReadOnly Property Portal As IPEndPoint

ReadOnly invoke As AsynInvoke

ReadOnly req As New RequestStream(Protocols.ProtocolEntry, TaskProtocols.MoveNext)

''' <summary>

''' Creates a linq source reader from the remote entry point

''' </summary>

''' <param name="portal"></param>

Sub New(portal As IPEndPoint)

Me.Portal = portal

Me.invoke = New AsynInvoke(portal)

End Sub

Public Overrides Function ToString() As String

Return $"{Type.FullName}@{Portal.ToString}"

End Function

#Region "Implements IEnumerable(Of T)"

Public Iterator Function AsQuerable() As IEnumerator(Of T) Implements IEnumerable(Of T).GetEnumerator

Call invoke.SendMessage(Protocols.LinqReset) ' resets the remote linq source read position

Do While True

Dim rep As RequestStream = invoke.SendMessage(req)

Dim json As String = rep.GetUTF8String

Dim value As Object = Serialization.LoadObject(json, Type)

Dim x As T = DirectCast(value, T)

Yield x

If rep.ProtocolCategory = TaskProtocols.ReadsDone Then

Exit Do

End If

Loop

End Function

Private Iterator Function IEnumerable\_GetEnumerator() As IEnumerator Implements IEnumerable.GetEnumerator

Yield AsQuerable()

End Function

#End Region

''' <summary>

''' Automatically free the remote resource.(释放远程主机上面的资源)

''' </summary>

Private Sub \_\_free()

Dim uid As String = Portal.ToString

Dim req As New RequestStream(ProtocolEntry, TaskProtocols.Free, uid)

Call invoke.SendMessage(req)

End Sub

#Region "IDisposable Support"

Private disposedValue As Boolean ' To detect redundant calls

' IDisposable

Protected Overridable Sub Dispose(disposing As Boolean)

If Not Me.disposedValue Then

If disposing Then

' TODO: dispose managed state (managed objects).

Call \_\_free()

Call invoke.Free

End If

' TODO: free unmanaged resources (unmanaged objects) and override Finalize() below.

' TODO: set large fields to null.

End If

Me.disposedValue = True

End Sub

' TODO: override Finalize() only if Dispose(disposing As Boolean) above has code to free unmanaged resources.

'Protected Overrides Sub Finalize()

' ' Do not change this code. Put cleanup code in Dispose(disposing As Boolean) above.

' Dispose(False)

' MyBase.Finalize()

'End Sub

' This code added by Visual Basic to correctly implement the disposable pattern.

Public Sub Dispose() Implements IDisposable.Dispose

' Do not change this code. Put cleanup code in Dispose(disposing As Boolean) above.

Dispose(True)

' TODO: uncomment the following line if Finalize() is overridden above.

' GC.SuppressFinalize(Me)

End Sub

#End Region

End Class

## Remote FileSystem

## Additional: Network data transfer in VisualBasic

### Object Json Serialization

#### Object Serialize as Json string

Imports Microsoft.VisualBasic.Serialization

Dim json As String = <Your Object>.GetJson

#### Deserialize the json string as .NET object

Dim params As InvokeInfo = Microsoft.VisualBasic.Serialization.LoadObject(Of InvokeInfo)(args.GetUTF8String)

An alternative solution of the Json serialization in the .NET programming is NewtonSoft Json library.

### SendMessage

‘ Remote machine network portal

Dim remote As IPEndPoint

‘ Protocol request

Dim req As RequestStream = New RequestStream(ProtocolEntry, TaskProtocols.Invoke, ProtocolParameter)

‘ Send Protocol to invoke on the remote machine

Dim rep As RequestStream = New AsynInvoke(remote).SendMessage(req)

‘ Gets the remote returns value

Dim rtvl As Rtvl = Serialization.LoadObject(Of Rtvl)(rep.GetUTF8String)

### Protocol Handler

On the remote server machine

Imports Microsoft.VisualBasic.Net

Imports Microsoft.VisualBasic.Net.Protocol

Imports Microsoft.VisualBasic.Net.Protocol.Reflection

Dim Responsehandler = AddressOf New ProtocolHandler(Me).HandleRequest

The Microsoft.VisualBasic.Net.Protocol.Reflection.Protocol attribute is required of building a protocols flexible and high performance multiple threading server side program.

### Services Socket

Dim Server As New TcpSynchronizationServicesSocket(listenPort)

‘ Binding the protocol to your server socket

Server.Responsehandler = AddressOf New ProtocolHandler(Me).HandleRequest

Call Server.Run

‘ Then the current thread will be bloked at Server.Run statement until the socket is disposed at other place

‘ Or using the thread to run the socket

Call Microsoft.VisualBasic.Parallel.Run(AddressOf Server.Run)