МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Отчет   
по лабораторной работе №4  
на тему: «Реализация взаимодействия между приложениями на основе технологии xml-rpc»  
по дисциплине «Проектирование и архитектура программных систем»

Выполнили: Марочкин М.А. Шифр: 170584   
 Шорин В.Д. Шифр: 171406  
 Щекотихин С.Е. Шифр: 170590  
ИПАИТ  
Направление: 09.03.04 «Программная инженерия»  
Группа: 71-ПГ  
Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Отметка о зачете:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата «\_\_\_\_» \_\_\_\_\_\_ 2019 г.

Орел, 2019 г.

## Задание на лабораторную работу

Разработать две программы – сервер и клиент. Клиент принимает от пользователя целочисленную квадратную матрицу и отсылает ее серверу. Сервер принимает целочисленную квадратную матрицу, после этого находит диагональ с минимальным элементом и заменяет элементы этой диагонали нулями, затем возводит в квадрат элементы стоящие ниже этой диагонали. На экран выводится исходная матрица, минимальный элемент диагоналей и результирующая матрица. Выполнение этого действия оформить как удаленную процедуру и воспользоваться механизмом RPC.

**Описание структуры программы**

Клиент, который хочет взаимодействовать с сервером по протоколу xml-rpc создаёт xml документ с описанием метода сервера который нужно вызвать и параметрами. Сервер получает данный документ и анализирует его. Из документа извлекается информация о методе сервера, который был запрошен и данный метод запускается. Полученный результат выполнения метода записывается в новый xml документ, который отправляется обратно клиенту. Клиент получает документ анализирует его и преобразует в результат выполнения метода. Чтобы сервер и клиент могли взаимодействовать между собой должна быть одинаковая структура xml-документа.

## Контрольные вопросы

*1.Опишите основные принципы архитектуры приложений, основанной на удалённом вызове процедур?*

Удалённый вызов процедур — класс технологий, позволяющих компьютернымпрограммам вызывать функции или процедуры в другом адресном пространстве (как правило, на удалённых компьютерах). Обычно реализация RPC-технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур, для необъектных RPC). Различные реализации RPC имеют очень отличающуюся друг от друга архитектуру и разнятся в своих возможностях: одни реализуют архитектуру SOA, другие — CORBA или DCOM. На транспортном уровне RPC используют в основном протоколы TCP и UDP, однако, некоторые построены на основе HTTP (что нарушает архитектуру ISO/OSI, так как HTTP — изначально не транспортный протокол).

*2.Опишите механизм работы удалённой процедуры?*

Перед непосредственным вызовом на клиентской и серверной стороне должны быть созданы специальные структуры (процедуры, файлы) - это так называемые клиентский стаб (stub) и серверный скелетон (skeleton), которые необходимы для корректной работы RPC. Чаще всего, они генерируются автоматически специальными утилитами по основному коду программы.

При удаленном вызове процедуры в распределенной системе происходят следующие действия:

1. Процедура клиента вызывает стаб как обычную процедуру. Стаб упаковывает параметры (маршализация, marshaling).

2. Стаб обращается к ядру ОС.

3. Ядро посылает сообщение на удаленную машину (ядру удаленного ПК).

4. Передача полученного сообщения скелетону серверного процесса.

5. Распаковка параметров (демаршализация, unmarshaling). Вызов требуемой процедуры.

6. Процедура на сервере выполняется. Возвращает результаты скелетону.

7. Скелетон упаковывает результат.

8. Передача результата ядру.

9. Ядро сервера передает сообщение по сети ядру клиента.

10. Ядро клиента обращается к стабу. Стаб распаковывает полученный результат.

11. Передача от стаба клиентскому процессу.

*3.Опишите структуру протокола XML-RPC.*

XML-RPC (сокр. от англ. Extensible Markup Language Remote Procedure Call — XML-вызов удалённых процедур) — стандарт/протокол вызова удалённых процедур, использующий XML для кодирования своих сообщений и HTTP в качестве транспортного механизма. Является прародителем SOAP, отличается исключительной простотой в применении. XML-RPC, как и любой другой интерфейс Remote Procedure Call (RPC), определяет набор стандартных типов данных и команд, которые программист может использовать для доступа к функциональности другой программы, находящейся на другом компьютере в сети.

*4.Как реализуется механизм удалённого вызова процедур в платформе .NET?*

В сообщении, отсылаемом клиентом серверу, классы-посредники кодируют название вызываемого метода удаленного объекта и все входящие переменные. На сервере сообщение раскодируется (т. е. определяется название метода и все входящие параметры) и выполняется реальный метод. Чтобы передать какие-либо данные на сервер, достаточно вызвать метод класса-посредника в качестве аргументов метода, взяв эти данные. Например, можно передавать любые двоичные данные или строки текста. Сервер, в свою очередь, аналогичным образом может передать клиенту возвращаемое значение метода - оно будет раскодировано на клиенте через класс-посредник

**Листинг программы**

Server

«Program.cs»

using System;using Nwc.XmlRpc;

namespace XMLRPCServer

{ class Server {

const int PORT = 5050;

static void Main(string[] args) {

XmlRpcServer server = new XmlRpcServer(PORT); server.Add("sample", new Server());

Console.WriteLine("Web Server Running on port {0} ... Press ^C to Stop...", PORT); server.Start();

}

public DateTime Ping() { Console.WriteLine("Ping\n"); return DateTime.Now; }

public String Echo(String arg) { Console.WriteLine("Echo\n"); return arg; }

public void ConvertMatrix(int size, string strMatrix) { int[,] matrix = new int[size, size];

string[] strArray = strMatrix.Split('\n');

for (int i = 0; i < size; i++) { string[] rowArray = strArray[i].Split(' ');

for (int j = 0; j < size; j++) { matrix[i, j] = Convert.ToInt32(rowArray[j]); }

}

int minValue = matrix[0,0]; int minI = 0, minJ = 0;

for (int i = 0; i < size; i++) {

if (minValue > matrix[i, i]) { minValue = matrix[i, i]; minI = minJ = i; }

}

for (int i = size - 1; i >= 0; i--) {

if (minValue > matrix[i, size - i - 1]) { minValue = matrix[i, size - i - 1]; minI = i; minJ = size - i - 1;}

}

Console.WriteLine(); Console.WriteLine("Min value: {0} [{1}][{2}]", minValue, minI, minJ);

if (minI == minJ) {

for (int i = 0; i < size; i++) { matrix[i, i] = 0; }

for (int i = 1; i < size; i++) { for (int j = 0; j < i; j++) {matrix[i, j] \*= matrix[i, j]; } }

} else if (minI + minJ == size - 1) {

for (int i = size - 1; i >= 0; i--) { matrix[i, size - i - 1] = 0; }

for (int i = 1; i < size; i++) { for (int j = size - 1; j > size - i - 1; j--) { matrix[i, j] \*= matrix[i, j]; } }

}

Console.WriteLine("Result");

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) { Console.Write("{0} \t", matrix[i, j]); }

Console.WriteLine();

}

}

}

}

«XmlRpcRequest.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.IO; using System.Xml; using System.Net;

using System.Text; using System.Reflection;

public class XmlRpcRequest {

public String MethodName = null; public ArrayList Params = null;

private Encoding \_encoding = new ASCIIEncoding();

public XmlRpcRequest() { Params = new ArrayList(); }

public String MethodNameObject {

get { int index = MethodName.IndexOf(".");

if (index == -1) return MethodName;

return MethodName.Substring(0,index);

}

}

public String MethodNameMethod {

get {

int index = MethodName.IndexOf(".");

if (index == -1) return MethodName;

return MethodName.Substring(index + 1, MethodName.Length - index - 1);

}

}

public XmlRpcResponse Send(String url) {

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

request.Method = "POST"; request.ContentType = "test/xml";

Stream stream = request.GetRequestStream();

XmlTextWriter xml = new XmlTextWriter(stream, \_encoding);

XmlRpcRequestSerializer.Serialize(xml, this);

xml.Flush(); xml.Close();

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

StreamReader input = new StreamReader(response.GetResponseStream());

XmlRpcResponse resp = XmlRpcResponseDeserializer.Parse(input);

input.Close(); response.Close(); return resp;

}

public Object Invoke(Object target) {

Type type = target.GetType();MethodInfo method = type.GetMethod(MethodNameMethod);

if (method == null)

throw new XmlRpcException(-2,"Method " + MethodNameMethod + " not found.");

if (XmlRpcExposedAttribute.IsExposed(target.GetType()) &&

!XmlRpcExposedAttribute.IsExposed(method))

throw new XmlRpcException(-3, "Method " + MethodNameMethod + " is not exposed.");

Object[] args = new Object[Params.Count];

for (int i = 0; i < Params.Count; i++) args[i] = Params[i];

return method.Invoke(target, args);

}

override public String ToString() {

StringWriter strBuf = new StringWriter(); XmlTextWriter xml = new XmlTextWriter(strBuf);

xml.Formatting = Formatting.Indented; xml.Indentation = 4;

XmlRpcRequestSerializer.Serialize(xml,this); xml.Flush(); xml.Close(); return strBuf.ToString();

}

}

}

«XmlRpcRequestDeserializer.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.IO; using System.Xml;

class XmlRpcRequestDeserializer : XmlRpcDeserializer {

static private XmlRpcRequestDeserializer \_singleton; static public XmlRpcRequestDeserializer Singleton

{ get {

if (\_singleton == null) \_singleton = new XmlRpcRequestDeserializer();

return \_singleton;

}

}

public static XmlRpcRequest Parse(StreamReader xmlData) {

XmlTextReader reader = new XmlTextReader(xmlData);

XmlRpcRequest request = new XmlRpcRequest(); bool done = false;

while (!done && reader.Read()) {

Singleton.ParseNode(reader);

switch (reader.NodeType) {

case XmlNodeType.EndElement:

switch (reader.Name) {

case METHOD\_NAME: request.MethodName = Singleton.\_text; break;

case METHOD\_CALL: done = true; break;

case PARAM: request.Params.Add(Singleton.\_value); Singleton.\_text = null; break;

}

break;

default:

Singleton.ParseNode(reader); break;

}

}

return request;

}

}

}

«XmlRpcRequestSerializer.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.Xml;

class XmlRpcRequestSerializer : XmlRpcSerializer {

static public void Serialize(XmlTextWriter output, XmlRpcRequest req) {

output.WriteStartDocument(); output.WriteStartElement(METHOD\_CALL);

output.WriteElementString(METHOD\_NAME,req.MethodName);

output.WriteStartElement(PARAMS);

foreach (Object param in req.Params)

{

output.WriteStartElement(PARAM); output.WriteStartElement(VALUE);

SerializeObject(output, param); output.WriteEndElement(); output.WriteEndElement();

}

output.WriteEndElement(); output.WriteEndElement();

}

}

}

«XmlRpcResponse.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.IO; using System.Xml;

public class XmlRpcResponse {

protected Object \_value; public bool IsFault;

public XmlRpcResponse() { Value = null; IsFault = false; }

public XmlRpcResponse(int code, String message) : this() { SetFault(code,message); }

public Object Value {

get { return \_value; }

set { IsFault = false; \_value = value; }

}

public int FaultCode {

get {

if (!IsFault) return 0;

else return (int)((Hashtable)\_value)["faultCode"];

}

}

public String FaultString {

get {

if (!IsFault) return "";

else return (String)((Hashtable)\_value)["faultString"];

}

}

public void SetFault(int code, String message)

{ Hashtable fault = new Hashtable(); fault.Add("faultCode", code); fault.Add("faultString", message);

Value = fault; IsFault = true;

}

override public String ToString() {

StringWriter strBuf = new StringWriter(); XmlTextWriter xml = new XmlTextWriter(strBuf);

xml.Formatting = Formatting.Indented; xml.Indentation = 4;

XmlRpcResponseSerializer.Serialize(xml,this); xml.Flush();

xml.Close(); return strBuf.ToString();

}

}

}

«XmlRpcResponseDeserializer.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.IO; using System.Xml;

class XmlRpcResponseDeserializer : XmlRpcDeserializer {

static private XmlRpcResponseDeserializer \_singleton;

static public XmlRpcResponseDeserializer Singleton

{ get {

if (\_singleton == null) \_singleton = new XmlRpcResponseDeserializer();

return \_singleton;

}

}

public static XmlRpcResponse Parse(StreamReader xmlData) {

XmlTextReader reader = new XmlTextReader(xmlData);

XmlRpcResponse response = new XmlRpcResponse(); bool done = false;

while (!done && reader.Read()) {

Singleton.ParseNode(reader); // Parent parse...

switch (reader.NodeType) {

case XmlNodeType.EndElement:

switch (reader.Name) {

case FAULT: response.Value = Singleton.\_value; response.IsFault = true; break;

case PARAM:response.Value = Singleton.\_value;Singleton.\_value = null;Singleton.\_text = null;

break;

}

break;

default: break;

}

}

return response;

}

}

}

«XmlRpcResponseSerializer.cs»

namespace Nwc.XmlRpc{ using System; using System.Collections; using System.Xml;

class XmlRpcResponseSerializer : XmlRpcSerializer {

static public void Serialize(XmlTextWriter output, XmlRpcResponse response) {

output.WriteStartDocument(); output.WriteStartElement(METHOD\_RESPONSE);

if (response.IsFault) output.WriteStartElement(FAULT);

else {

output.WriteStartElement(PARAMS); output.WriteStartElement(PARAM);

}

output.WriteStartElement(VALUE); SerializeObject(output,response.Value);

output.WriteEndElement(); output.WriteEndElement();

if (!response.IsFault) output.WriteEndElement();

output.WriteEndElement();

}

}

}

«XmlRpcSerializer.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.IO; using System.Xml; using System.Diagnostics;

class XmlRpcSerializer : XmlRpcXmlTokens {

static public void SerializeObject(XmlTextWriter output, Object obj) {

if (obj == null) return;

if (obj is byte[]) {

byte[] ba = (byte[])obj; output.WriteStartElement(BASE64);

output.WriteBase64(ba,0,ba.Length); output.WriteEndElement();

}

else if (obj is String) { output.WriteElementString(STRING,obj.ToString()); }

else if (obj is Int32) { output.WriteElementString(INT,obj.ToString()); }

else if (obj is DateTime)

{

output.WriteElementString(DATETIME,((DateTime)obj).ToString(ISO\_DATETIME));

}

else if (obj is Double) { output.WriteElementString(DOUBLE,obj.ToString()); }

else if (obj is Boolean) {

output.WriteElementString(BOOLEAN, ((((Boolean)obj) == true)?"1":"0"));

}

else if (obj is ArrayList) {

output.WriteStartElement(ARRAY); output.WriteStartElement(DATA);

if (((ArrayList)obj).Count > 0) {

foreach (Object member in ((ArrayList)obj)) {

output.WriteStartElement(VALUE); SerializeObject(output,member);

output.WriteEndElement();

}

}

output.WriteEndElement(); output.WriteEndElement();

}

else if (obj is Hashtable) {

Hashtable h = (Hashtable)obj; output.WriteStartElement(STRUCT);

foreach (String key in h.Keys) {

output.WriteStartElement(MEMBER); output.WriteElementString(NAME,key);

output.WriteStartElement(VALUE); SerializeObject(output,h[key]);

output.WriteEndElement(); output.WriteEndElement();

}

output.WriteEndElement();

}

}

}

}

«XmlRpcServer.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.Diagnostics; using System.IO; using System.Net;

using System.Net.Sockets; using System.Text; using System.Threading; using System.Xml;

public class XmlRpcServer : IEnumerable {

private TcpListener \_myListener ; private int \_port; private Hashtable \_handlers;

private XmlRpcSystemObject \_system;

public XmlRpcServer(int port) {

\_port = port; \_handlers = new Hashtable(); \_system = new XmlRpcSystemObject(this);

}

public void Start() {

try {

\_myListener = new TcpListener(\_port) ; \_myListener.Start();

Thread th = new Thread(new ThreadStart(StartListen)); th.Start();

}

catch(Exception e) {

Logger.WriteEntry("AnExceptionOccurred while Listening :" +e.ToString(), EventLogEntryType.Error);

}

}

public IEnumerator GetEnumerator() { return \_handlers.GetEnumerator(); }

public Object this [String name] { get { return \_handlers[name]; } }

public void SendHeader(string sHttpVersion, string sMIMEHeader, long iTotBytes, string sStatusCode, TextWriter output) {

String sBuffer = "";

if (sMIMEHeader.Length == 0 ){ sMIMEHeader = "text/html"; // Default Mime Type is text/html }

sBuffer = sBuffer + sHttpVersion + sStatusCode + "\r\n";

sBuffer = sBuffer + "Connection: close\r\n";

if (iTotBytes > 0) sBuffer = sBuffer + "Content-Length: " + iTotBytes + "\r\n";

sBuffer = sBuffer + "Server: XmlRpcServer \r\n";

sBuffer = sBuffer + "Content-Type: " + sMIMEHeader + "\r\n"; sBuffer = sBuffer += "\r\n";

output.Write(sBuffer);

}

public void StartListen() {

while(true) {

TcpClient client = \_myListener.AcceptTcpClient();

SimpleHttpRequest httpReq = new SimpleHttpRequest(client);

if (httpReq.HttpMethod == "POST") {

try { HttpPost(httpReq); }

catch (Exception e) { Logger.WriteEntry("Failed on post: " + e, EventLogEntryType.Error); }

}

else {

Logger.WriteEntry("Only POST methods are supported: " + httpReq.HttpMethod + " ignored", EventLogEntryType.FailureAudit);

}

httpReq.Close();

}

}

public void HttpPost(SimpleHttpRequest req) {

XmlRpcRequest rpc = XmlRpcRequestDeserializer.Parse(req.Input);

XmlRpcResponse resp = new XmlRpcResponse();Object target = \_handlers[rpc.MethodNameObject];

if (target == null) { resp.SetFault(-1, "Object " + rpc.MethodNameObject + " not registered."); }

else {

try { resp.Value = rpc.Invoke(target); }

catch (XmlRpcException e) { resp.SetFault(e.Code, e.Message); }

catch (Exception e2) { resp.SetFault(-1, e2.Message); }

}

Logger.WriteEntry(resp.ToString(), EventLogEntryType.Information);

SendHeader(req.Protocol, "text/xml", 0, " 200 OK", req.Output); req.Output.Flush();

XmlTextWriter xml = new XmlTextWriter(req.Output);XmlRpcResponseSerializer.Serialize(xml, resp);

xml.Flush(); req.Output.Flush();

}

public void Add(String name, Object obj) { \_handlers.Add(name,obj); }

}

}

«XmlRpcSystemObject.cs»

namespace Nwc.XmlRpc{

using System; using System.Collections; using System.Reflection; using System.Diagnostics;

[XmlRpcExposed]

public class XmlRpcSystemObject {

private XmlRpcServer \_server;

public XmlRpcSystemObject(XmlRpcServer server) {

\_server = server; server.Add("system",this);

}

[XmlRpcExposed]

public ArrayList listMethods() {

ArrayList methods = new ArrayList(); Boolean considerExposure;

foreach (DictionaryEntry handlerEntry in \_server) {

considerExposure = XmlRpcExposedAttribute.IsExposed(handlerEntry.Value.GetType());

foreach (MemberInfo mi in handlerEntry.Value.GetType().GetMembers()) {

if (mi.MemberType != MemberTypes.Method) continue;

if(!((MethodInfo)mi).IsPublic) continue;

if (considerExposure && !XmlRpcExposedAttribute.IsExposed(mi)) continue;

methods.Add(handlerEntry.Key + "." + mi.Name);

}

}

return methods;

}

[XmlRpcExposed]

public ArrayList methodSignature(String name) {

ArrayList signatures = new ArrayList(); int index = name.IndexOf('.');

if (index < 0) return signatures;

String oName = name.Substring(0,index); Object obj = \_server[oName];

if (obj == null) return signatures;

MemberInfo[] mi = obj.GetType().GetMember(name.Substring(index + 1));

if (mi == null || mi.Length != 1) // for now we want a single signature return signatures;

MethodInfo method;

try { method = (MethodInfo)mi[0]; }

catch (Exception e) {

Logger.WriteEntry("Attempted methodSignature call on " + mi[0] + " caused: " + e,

EventLogEntryType.Information);

return signatures;

}

if (!method.IsPublic) return signatures;

ArrayList signature = new ArrayList(); signature.Add(method.ReturnType.Name);

foreach (ParameterInfo param in method.GetParameters()){ signature.Add(param.ParameterType.Name); }

signatures.Add(signature);

return signatures;

}

[XmlRpcExposed]

public String methodHelp(String name) { return "methodHelp not yet implemented."; }

}

}

«XmlRpcXmlTokens.cs»

namespace Nwc.XmlRpc{ using System;

class XmlRpcXmlTokens {

public const String ISO\_DATETIME = "yyyyMMdd\\THH\\:mm\\:ss";

public const String BASE64 = "base64"; public const String STRING = "string";

public const String INT = "i4"; public const String ALT\_INT = "int";

public const String DATETIME = "dateTime.iso8601"; public const String BOOLEAN = "boolean";

public const String VALUE = "value"; public const String NAME = "name";

public const String ARRAY = "array"; public const String DATA = "data";

public const String MEMBER = "member"; public const String STRUCT = "struct";

public const String DOUBLE = "double"; public const String PARAM = "param";

public const String PARAMS = "params"; public const String METHOD\_CALL = "methodCall";

public const String METHOD\_NAME = "methodName";

public const String METHOD\_RESPONSE = "methodResponse"; public const String FAULT = "fault";

}

}

Client

«Program.cs»

using System;

namespace XMLRPCClient{

class Program {

static ServersideObject obj;

static void Main(string[] args) {

int size; size = int.Parse(Console.ReadLine()); int[,] matrix = new int[size, size];

for (int i = 0; i < size; i++) { for (int j = 0; j < size; j++) { matrix[i, j] = int.Parse(Console.ReadLine()); } }

obj = new ServersideObject("http://127.0.0.1:5050");

obj.SendMatrix(size, matrix);

Console.ReadLine();

}

}

}

«ServersideObject.cs»

using System;using Nwc.XmlRpc;

namespace XMLRPCClient{

class ServersideObject {

XmlRpcRequest client; String host = "http://127.0.0.1:5050";

public ServersideObject(String host) {

this.host = host; client = new XmlRpcRequest(); }

public void SendMatrix(int size, int[,] matrix) {

XmlRpcResponse response; client.MethodName = "sample.ConvertMatrix";

client.Params.Clear(); client.Params.Add(size);

string str = "";

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (j == size - 1) { str += matrix[i, j].ToString(); break;

}

str += matrix[i, j].ToString() + ' ';

}

if (i == size - 1) break;

str += '\n';

}

client.Params.Add(str); Console.WriteLine(client); response = client.Send(host);

if (response.IsFault) {

Console.WriteLine("Fault {0}: {1}", response.FaultCode, response.FaultString);

return;

} else { Console.WriteLine("OK!"); }

}

}

}