ҚАЗ ҰТУ

ҒЫЛЫМИ ЖУРНАЛ
НАУЧНЫЙ ЖУРНАЛ

ISSN 1680-9211

№3 (103)
2014

Қ.И. СӘТБАЕВ АТЫНДАҒЫ
ҚАЗАҚ ҰЛТТЫҚ ТЕХНИКАЛЫҚ
УНИВЕРСИТЕТІНІҢ

ХАБАРШЫСЫ

ВЕСТНИК

КАЗАХСКОГО НАЦИОНАЛЬНОГО
ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА
ИМЕНИ К.И. САТПАЕВА

ҚАЗАҚСТАН РЕСПУБЛИКАСЫ
БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РЕСПУБЛИКИ КАЗАХСТАН

ҚазҰТУ ХАБАРШЫСЫ

ВЕСТНИК КазНТУ

VESTNIK KazNTU

№3 (103)

АЛМАТЫ 2014 МАЙ

Главный редактор
Ж.М. Адилов –
академик, доктор экономических наук, профессор

Зам. главного редактора
Е.И. Кульдеев –
проректор по науке и инновационной деятельности

Отв. секретарь
Н.Ф. Федосенко

Редакционная коллегия:

С.Б. Абдыгаппарова, Б.С. Ахметов, Г.Т. Балакаева, К.К. Бегалинова, В.И. Волчихин (Россия), Д. Харнич (США), К. Дребенштед (Германия), И.Н. Дюсембаев, Г.Ж. Жолтаев, С.Е. Кудайбергенов, С.Е. Кумеков, В.А. Луганов, С.С. Набойченко – член-корр. РАН, И.Г. Милев (Германия), С. Пежовник (Словения), Б.Р. Ракишев – акад. НАН РК, М.Б. Панфилов (Франция), Н.Т. Сайлаубеков, Н.С. Сеитов – член-корр. НАН РК, А.Т. Турдалиев, Г.Т. Турсунова.

Учредитель:

Казахский национальный технический университет
имени К.И. Сатпаева

Регистрация:

Министерство культуры, информации и общественного согласия
Республики Казахстан № 951 – Ж “25” 11. 1999 г.

Основан в августе 1994 г. Выходит 6 раз в год

Адрес редакции:

г. Алматы, ул. Сатпаева, 22,
каб. 904, тел. 292-63-46
n. fedossenko @ ntu. kz

© КазНТУ имени К.И. Сатпаева, 2014

Mirzahmedova G.A.

Application C # programming language for scientific computing

Summary. Here we consider some internal C #, that makes it easy and practical to generate code that is sensitive to the speed of execution. You can see what a big role to play C # in the scientific community, opening the door to the next generation of numerical calculations. I explore the qualities that in C # is a good alternative in the world of numerical calculations and the results of several tests and compare them with the results of an unmanaged C + +, in order to understand at what level is C # in terms of productivity and efficiency.

Key words: Programming language C#, JIT compiler, .NET Framework, Common Language Specification (CLS) and Common Type System (CTS).

ӘОЖ(378.016.02:004.032.6:574)

Г.А. Мирзахмедова, А.Қ. Самбетбаева
(Әл-Фараби атындағы Қазақ ұлттық университеті
Алматы, Қазақстан Республикасы, gulban84@mail.ru, erulan72@mail.ru)

C# ТІЛІНДЕ ПРОКСИ СЕРВЕРДІ ӨҢДЕУ

Аннотация. Прокси – сервер клиенттер мен өзге серверлер және желілік қызметтер арасында мәліметтерді тасымалдауды ұйымдастыруға мүмкіндік береді. Прокси – серверлерді Интернет желісіне қосылу мүмкіндігін алу, желілік қосымшаларды ретке келтіру және т.с.с. қызметтер үшін қолдануға болады. Бұл мақалада C# программалау тілінің көмегімен сокеттерді пайдалану арқылы жеке *HTTP* прокси серверін құруды карастырамыз.

Түйін сөздер: C# программалау тілі, *HTTP* прокси сервері, System.Text.Encoding NET қосымшасы, ItemBase.

Прокси – серверлер тура және кері болып екіге бөлінеді. Тура прокси – сервер қолданушы серверін өзге серверлермен байланыстырушы құрылғы болып табылады, қолданушының сұранысын желілерге тікелей аударады және қайта қолданушыға хабарлама жібереді. Кері прокси серверлер желідегі жүктемелерді тарату, қолданушының сұранысын бір желінің ішінде бірнеше серверлерге қайта аудару үшін қолданылады.

1-сурет. Прокси сервердің қызметінің сұлбасы.

Интернет желісінде көпке танымал ол **HTTP** хаттамасы. Прокси – серверде бір мезгілде бір немесе бірнеше хаттамалармен жұмыс жасауға болады. Прокси-серверді құру үшін «Windows консольді қосымшасы» типіндегі жобаны қолданған жөн. Жобаны «**Прокси_Сервер**» деп атайық. Прокси сервер нақты портта тыңдалуы қажет. Порттарды тыңдау үшін стандартты кластарды қолданамыз. NET **Framework** –та портқа кластың атын **TcpListener** деп береміз. Инициализациялау барысында **TcpListener** класы тыңдалуға қажетті екі парметрді IP-адрес және портты қабылдайды. Жергілікті машинада IP-адрес **127.0.0.1** –ге тең, порт ретінде кез келген бос портты қолдануға болады.

```
TcpListener myTCP = new TcpListener(IPAddress.Parse("127.0.0.1"), 8888);
```

Циклдің денесінде қолданушыдан сұраныстарды қабылдап оларды өңдейтін негізгі код орналастырылуы қажет. Сұраныстардың бар жоғын анықтау үшін **TcpListener** класындағы **Pending** функциясының көмегімен анықтаймыз.

```
}
if (myTCP.Pending())// сұраныс бар
}
using (Socket myClient = myTCP.AcceptSocket())}
if (myClient.Connected) {
```

Жауапты байт массиві түрінде алуға болады. Келіп түсетін сұраныстар бинарлы ақпарат түрінде берілуі мүмкін, осы ақпараттардың мазмұнын жоғалтпас үшін байт массивін міндетті түрде қолдану қажет. Қосымшаларды өңдеу барысында ашық сокеттен ақпараттарды бірнеше рет алу қажет болады, сондықтан сокеттерге арнайы функция қолданамыз. Функция сокеттердің көшірмесін қабылдап, оларды байт массиві түрінде қайта жолдайды.

```
private static byte[] ReadToEnd(Socket mySocket)
{
byte[] b = new byte[mySocket.ReceiveBufferSize];
int len = 0;

using (MemoryStream m = new MemoryStream()){

while (mySocket.Poll(1000000, SelectMode.SelectRead) && (len=mySocket.Receive(b,
mySocket.ReceiveBufferSize, SocketFlags.None)) > 0){

m.Write(b, 0, len);}

return m.ToArray();}

byte[] httpRequest = ReadToEnd(myClient);
```

Содан соң сұранысты көрсетілген серверге бағыттау қажет. Сұранысты қайта бағыттау қажеттігі туралы ақпарат HTTP сұраныстар парақшасында орналасқан. HTTP – тақырыпшасы қатарларды алмастыратын (**\r\n\r\n** немесе **CR LF CR LF** немесе **13 10 13 10**) тесттің негізгі мазмұнынан ерекшеленетін нақты анықталған форматтағы текстен тұрады. Бізге ең маңыздысы регулярлы өрнектердің көмегімен табуға болатын **Host** тақырыпшасы ғана қажет. Қолданушының сұранысы байт массивінде сақталғандықтан, ал регулярлы өрнектер тек тексттермен жұмыс жасайтын болғандықтан мәліметтерді **System.Text.Encoding** класында орналасқан **GetString** функциясының көмегімен түрлендіруге болады.

```
Regex myReg = new Regex(@"Host: (((?<host>.+?):(?<port>\d+?))|(?<host>.+?))\s+",
RegexOptions.Multiline | RegexOptions.IgnoreCase);
Match m = myReg.Match(System.Text.Encoding.ASCII.GetString(httpRequest))
string host = m.Groups["host"].Value;
int port = 0;
if (!int.TryParse(m.Groups["port"].Value, out port)) { port = 80; }
```

IP-адрес пен порт нөмірінің көмегімен желілік ену нүктесін алуға болады.

```
IPEndPoint myIPEndPoint = new IPEndPoint(myIPHostEntry.AddressList[0], port);
```

Біздің құрған прокси серверде сұраныстар тізбектеле өңделеді, сұраныстар саны көп болған жағдайда қолданушылар үшін желілік байланыс жылдамдығы өте төмен болып көрінуі мүмкін. Мұндай жағдайларды болдымас үшін көп ағынды **Thread** класын пайдаланған жөн. Ол үшін сұраныстар(**myTCP.Pending**) кезегін тексеру шартының денесін жеке бір әдіске орналастыру жеткілікті, мысалы үшін **myClient** сокетін қабылдайтын **ExecuteRequest** әдісін қолданайық.

```
private static void ExecuteRequest(object arg) {
Socket myClient = (Socket)arg;
if (myClient.Connected) {
 byte[] httpRequest = ReadToEnd(myClient);
Regex myReg = new Regex(@"Host: (((?<host>.+?):(?<port>\d+?))|(?<host>.+?))\s+",
RegexOptions.Multiline | RegexOptions.IgnoreCase);
Match m = myReg.Match(System.Text.Encoding.ASCII.GetString(httpRequest));
string host = m.Groups["host"].Value;
int port = 0;
if (!int.TryParse(m.Groups["port"].Value, out port)) { port = 80; }
IPHostEntry myIPHostEntry = Dns.GetHostEntry(host);
IPEndPoint myIPEndPoint = new IPEndPoint(myIPHostEntry.AddressList[0], port);

using (Socket myRerouting = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp)) {
myRerouting.Connect(myIPEndPoint);

if (myRerouting.Send(httpRequest, httpRequest.Length, SocketFlags.None) != httpRequest.Length)
{
Console.WriteLine("При отправке данных удаленному серверу произошла ошибка..."); }
else{
byte[] httpResponse = ReadToEnd(myRerouting);
 myClient.Send(httpResponse, httpResponse.Length, SocketFlags.None);
}
}}}
```

Және әрбір сұранысты жаңа ағында жібереміз.

```
 while (true)
{
if (myTCP.Pending()){
Thread t = new Thread(ExecuteRequest);
t.IsBackground = true;
t.Start(myTCP.AcceptSocket());
}
}
```

Енді программа қысқа уақыт ішінде мол көлемді сұраныстарды өңдеу алады.

Қолданушылардың сұраныстарын және жойылған сервер жауабын өңдеуді жеңілдету үшін қосымша кластар жиынтығын пайдалануға болады. Әрбір HTTP сұраныс бас тақырыпшасы мен мазмұнынан тұрады. Біздің құратын негізгі класымыз бас тақырыпшаны оның мазмұнынан бөліп тұрады және мәліметтермен жұмыс жасауды жеңілдету үшін көмекші функцияны ұсынады. Бұл класты **Parser** деп атайық. **Parser** класында қасиеттер үш топтан тұрады. Алғашқы топ HTTP хаттамасы үшін ортақ класс болып келеді.

```
public byte[] Source { get; set; }
public string HTTPVersion { get; set; }
```

Екінші топ HTTP - сұраныстарына арналған класс болып келеді.

```
public string Method { get; set;
public string Path { get; set; }
```

үшіншісі HTTP - жауаптарға арналған класс болып келеді.

```
public int StatusCode { get; set; }
public string StatusMessage { get; set; }
```

Бұл кластардың колданушының сұранысын немесе жойылған сервердің жауабынан тұратын вбайтты қабылдайтын бір құрылымдаушысы болады.

```
public Parser(byte[] source)

lis.Source = source;
```

рылымдаушыда тақырыпшаларды өңдеу және алынған мәліметтердщ непзше сүиене [ағымдағы кластың көшірмесінің касиеттерін толықтыру жұмыстары атқарылады.

```
string sourceString = this.GetSourceAsStringQ;
string httplnfo = soiirceString.SubstrmgCO.sourceString.IndexOfC'^W'));
```

HTTP - еұраныстарда алғашкы сөз - ол көп жағдаіда GET немесе POST шамаларынан [сұраныстың типі (әдісі) болып табылады. Содан соң сұраныс жүргізілетін парақтың адресі ды. Ең соңында HTTP хаттамасының версиясы жазылады. Мысалы: GET http://kbyte.ru/ 10

Регулярлы өрнектердің көмегімен HTTP - сұныстарды тексеруге және қажет ақпараттарды цел сп алуга болады.

```
Regex myReg = new Regex(@f·(?<method> +)\s+(?<path>.+)\s HTTP/(?<version>[\d\.]+)f·,
xOptions.Multiline);
```

регулярлы өрнектердің үлгісі дұрыс жұмыс істесе, онда ағымдағы класс көшіршесіне ан сұраныс жайлы ақпаратты орналастыруға болады.

```
if (myReg.IsMatch(httpInfo))

Match m = myReg.Match(httpInfo);
This.Method = m.Groups["methodff].Value.ToUpper();
this.Path = m.Groups["path"]. Value;
this.HTTPVersion = m.Groups["version"].Value;
```

Ал егер де регулярлы өрнектердің үлгісі дұрыс жұмыс істемесе, онда жуықтап осыны HTTP - деп қабылдаймыз. Жауапта алғашқы сөз HTTP хаттамасына бағытталады, содан соң аның нүсқасы, сервердің қалып күйінің сандық коды және қалып күй кодының тесттік ніктемесі жазылады. Мысалы: HTTP/1.1 200 OK

HTTP - жауаптарды өңдеу үшін регулярлы өрнектердің өзге үлгісі қажет болады.

```
else
myReg = new Regex(@MHTTP/(?<version> ^ *)");
Match m = myReg.Match(httpInfo);
int.TryParse(m.Groups[fstatus"f].Value, out this.StatusCode);
this.StatusMessage = m.Groups["msg"]. Value;
```

```
this.HTTPVersion = m.Groups["version"].Value;}
```

Содан соң тақырыпшаны және оның мәндік шамасын алуға болады.

```
 private int  HeadersTail = -1;
 HeadersTail = sourceString.IndexOf("\x\n\x\n");
if(_HeadersTail !=-1)
{
sourceString = sourceString.Substring(sourceString.IndexOf("\r\n")+ 2, _HeadersTail -
sourceStrmg.IndexOf("\r\n")- 2);
```

Сайт парақшасының мазмұны мен тақырыпшасын ерекшелеп, регулярлы өрнектерЦ көмегімен оларды бірнеше кілттік шамаларға бөлуге болады. ItemBase - базалық класы үнсіз *m* заңы бойынша сөзтіркестерінің параметрлерін сақтау үшін қолданылады.

```
 public class ItemBase {
public string Source { get; set; }
 public ItemBase(string source) {
this.Source = source; }

public class ItemHost: ItemBase
public string Host { get; set; }
public int Port { get; set;}

public ItemHost(string source): base(source)
{
Regex myReg = new Regex(@"^(((?<host>+?):(?<port>\d+?))| (?<host>.+?))$");
Match m = myReg.Match(source);
this.Host = m.Groups["host"].Value;
if(!int.TryParse(m.Groups["port"]Value, out tliis.Port)) tliis.Port== 80;
}

public class ItemContentType : ItemBase
public string Value { get; set; }
public string Charset { get; set; }
public ItemContentType(string source): base(source)
{
if(String.IsNulIOrEmpty(source)) return;

int typeTail = source.IndexOf(";");
if(typeTail==-l)
{
this.Value = source.Trim().ToLow'er();
return;
}

this.Value = source.Substring(0, typeTail).Trim().ToLower();

string p = source. Substring(typeTail + 1. source. Length - typeTail -1);
Regex myReg = new Regex(@"(?<key> +?)=((""(?<value> +?)"")|((?<value>[^\;l+)))[\;]{0,l}^M,
RegexOptions. Singleline);
MatchCollection mc = myReg.Matches(p);
foreach (Match m in mc)
{
if(m.Groups["key"1.Value.Trim().ToLower() == "charset") {
this.Charset = m.Groups["value"].Value:
}}}
```

Енді негізгі P
болады.
public Dictiona

Parser класын
толыктырамыз.
this.ltcms = nev
myReg = new
MatchCollecti
foreach (Matct
J string key
[f (!this.Item|
зависимости от тип
j iГ(ксу.Triт(
{
this.ltcms.Adc
}
else if (key.T
{
this.ltems.Ac
" ГоCharArrayO)));
)
else
{
this.Itcms.A

Осымен бі
көмекші функци
берілетін Host ж
public strm
gct{ if(!thi
return
((ItemHost

public int
get{if(!th
return
((ItemHos

Сығылға^
jSET Framew^
public byj
if (_HeJ
bytefl res
BufferJ
if (this.lt
, Encoding"].So
{
GZipStrj
Compression^
using (J

Енді негізгі **Parser** класына есімдік **ItemBase** коллекциядан тұратын қасиеттерді енгізуге болады.

```
public Dictionary<string, ItemBase> Items { get; set; }
```

Parser класын регулярлы өрнектердің көмегімен коллекцияларды HTTP – тақырыпшаларымен толықтырамыз.

```
this.Items = new Dictionary<string, ItemBase>(StringComparer.CurrentCultureIgnoreCase);
myReg = new Regex(@"^(?<key>[^\x3A]+)\:\s{1}(?<value>.+)$", RegexOptions.Multiline);
MatchCollection mc = myReg.Matches(sourceString);
foreach (Match mm in mc)
{ string key = mm.Groups["key"].Value;
if (!this.Items.ContainsKey(key)) // если указанного заголовка нет в коллекции, добавляем его в
зависимости от типа
{ if (key.Trim().ToLower() == "host")
{
this.Items.Add (key, new ItemHost(mm.Groups["value"].Value.Trim("\r\n ".ToCharArray())));
}
else if (key.Trim().ToLower() == "content-type")
{
this.Items.Add (key, new ItemContentType(mm.Groups["value"].Value.Trim("\r\n
".ToCharArray())));
}
else
{
this.Items.AddItem(key, new ItemBase(mm.Groups["value"].Value.Trim("\r\n ".ToCharArray())));
}
}
```

Осымен біздің негізгі класымыз дайын. Бұның қызметін бұдан да жеңілдету үшін қосымша көмекші функцияларды енгізуге болады. Сайт парақшаларының тақырыпшасында **Host** атымен берілетін **Host** және **Port** қасиеттері хостқа және порт нөміріне жеңіл енуге мүмкіндік береді.

```
public string Host
get{ if (!this.Items.ContainsKey("Host")) return String.Empty;
return
((ItemHost)this.Items["Host"]).Host; }

public int Port
get{ if (!this.Items.ContainsKey("Host")) return 80;
return
((ItemHost)this.Items["Host"]).Port; }
```

Сығылған ақпараттар туралы мәліметті Content-Encoding тақырыпшасынан аламыз. Стандартты **.NET Framework** әдісінің көмегімен декомпрессия жүргізуге болады.

```
public byte[] GetBody()
if (_HeadersTail == -1) return null;
byte[] result = new byte[this.Source.Length -_HeadersTail - 4];
Buffer.BlockCopy(this.Source, _HeadersTail + 4, result, 0, result.Length);
if (this.Items != null && this.Items.ContainsKey("Content-Encoding") && this.Items["Content-
Encoding"].Source.ToLower() == "gzip")
{
GZipStream myGzip = new GZipStream(new MemoryStream(result),
CompressionMode.Decompress);
using (MemoryStream m = new MemoryStream()){
```

```
byte[] buffer = new byte[512];
int len = 0;
while ((len = myGzip.Read(buffer, 0, buffer.Length)) > 0)
{
m.Write(buffer, 0, len);
}
result = m.ToArray();
}
}
return result;
```

GetBody функциясы байт массивінің мәнін қайтарады. Тесттік мәліметтерді енгізу және массив байтты текстті ақпаратқа түрлендіру үшін осы функцияны пайдаланамыз.

```
public string GetBodyAsString()
Encoding e = Encoding.UTF8;
if (this.Items != null && this.Items.ContainsKey("Content-Type") &&
!String.IsNullOrEmpty(((ItemContentType) this.Items["Content-Type"]).Charset))
{
try{
 e = Encoding.GetEncoding(((ItemContentType)this.Items["Content-Type"]).Charset);
}
catch { }
}
return e.GetString(GetBody());
```

Сайт парақшасының мазмұнын өзгерту үшін негізгі **Parser** класын пайдаланамыз. Егер класқа қандайда бір өзгерістер енгізетін болсақ онда таратылатын мәліметтердің көлемі сақталатын **Content-Length** тақырыпшасын өзгерту қажет болады.

```
public void SetStringBody(string newBody){
if (this.StatusCode <= 0)
{ throw new Exception("Жойылған сереверден алынған жауаптың мазмұнын ғана өзгертуге
болады."); }
Encoding e = Encoding.UTF8;
 string result = String.Format("HTTP/{0} {1} {2}", this.HTTPVersion, this.StatusCode,
this.StatusMessage);
foreach (string k in this.Items.Keys) {
ItemBase itm = this.Items[k];
if (!String.IsNullOrEmpty(result)) result += "\r\n";
if (k.ToLower() == "content-length")
{
result += String.Format("{0}: {1}", k, newBody.Length);
}
else if (k.ToLower() == "content-encoding" && itm.Source.ToLower() == "gzip")
{
}
else
{
result += String.Format("{0}: {1}", k, itm.Source);
if (k.ToLower() == "content-type" &&
!String.IsNullOrEmpty(((ItemContentType)this.Items["Content-Type"]).Charset))
{
try
{
e = Encoding.GetEncoding(((ItemContentType)this.Items["Content-Type"]).Charset);
```

```
}
catch { }
}
}
}
result += "\r\n\r\n";
result += newBody;
this.Source = e.GetBytes(result);
```

Енді қолданушының сұраныстарын қайта бағыттау коды қарапайым және түсінікті түрде болады.

```
private static void ExecuteRequest(object arg)
using (Socket myClient = (Socket)arg) {
if (myClient.Connected)
{
byte[] httpRequest = ReadToEnd(myClient);
Parser http = new Parser(httpRequest);
 if (http.Items == null || http.Items.Count <= 0 || !http.Items.ContainsKey("Host"))
{
Console.WriteLine("{0} байт сұраныс қабылданды, Сайт тақырыпшалары табылмады.",
httpRequest.Length);
 }
else
{
Console.WriteLine("{0} байт сұраныс қабылданды , {1} әдіс, хост {2}:{3}", httpRequest.Length,
http.Method, http.Host, http.Port);
IPHostEntry myIPHostEntry = Dns.GetHostEntry(http.Host);
if (myIPHostEntry == null || myIPHostEntry.AddressList == null ||
myIPHostEntry.AddressList.Length <= 0)
{
Console.WriteLine("{0} хосты бойынша IP-адресті анықтау мүмкін болмады.", http.Host);
}
else
{
IPEndPoint myIPEndPoint = new IPEndPoint(myIPHostEntry.AddressList[0], http.Port);
using (Socket myRerouting = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp))
{
myRerouting.Connect(myIPEndPoint);
if (myRerouting.Send(httpRequest, httpRequest.Length, SocketFlags.None) != httpRequest.Length)
{
Console.WriteLine(" Мәліметтерді жойылған серверге жіберу барысында қателіктер пайда
болды...");
}
Else
{
byte[] httpResponse = ReadToEnd(myRerouting); myClient.Send(httpResponse,
httpResponse.Length, SocketFlags.None);
}
}
}
}
myClient.Close();
}
}
```

Құрылған программаны жергілікті желідегі немесе Инетрнеттегі серверге орналастыруға болады. Сервердің адресі мен порттың номерін біле отырып қолданушылар **HTTP** хаттамасы бойынша прокси сервер арқылы интернетке қосыла алады. Прокси сервердің көмегі арқылы кіріс және шығыс трафиктер бойынша есеп жүргізуге, кейбір сайттарға ену мүмкіндігін шектеуге және әрбір сайт парақшаларының сазмұнын бақылап отыруға болады. HTTP хаттамасының көмегімен өзіміздің құрған прокси серверге кіру осы хаттама арқылы қауіпсіз болады.

ӘДЕБИЕТТЕР

1. Microsoft. Разработка Web- приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET 2003, 704 стр.
2. Microsoft. Разработка Web-сервисов XML и серверных компонентов на MS Visual Basic .NET и Microsoft Visual C#. 2004, 572 стр.
3. Microsoft. Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C#. 2003, 514 стр.
4. Агуров Павел. C#. Разработка компонентов в MS Visual Studio 2005-2008. 2008, 467 стр.
5. Троелсен Э. C# и платформа .NET. Библиотека программиста. 2004. 796 стр.

REFERENCES

1. Microsoft. Razrabotka veb-prilozheniy na Microsoft Visual Basic. NET i Microsoft Visual C #. NET. 2003, 704 str .
2. Microsoft. Razrabotka veb-servisov XML i servernykh komponentov na MS Visual Basic. NET i Microsoft Visual C #. 2004, 572 str.
3. Microsoft. Razrabotka Windows, prilozheniy na Microsoft Visual Basic. NET i Microsoft Visual C #. 2003, 514 str.
4. Agurov Pavel. C #. Razrabotka komponentov v MS Visual Studio 2005-2008. 2008, 467 str.
5. Troyelsen E. S # i platforma. NET. Biblioteka programmista. 2004. 796 str.

Мирзахмедова Г.А., Самбетбаева А.Қ.

C# тілінде прокси серверді өңдеу

Түйіндеме. C# программалау тілінің көмегімен сокеттерді пайдалану арқылы жеке *HTTP* прокси серверін құруды қарастырылған. Қолданушының сұраныстары мен жауаптары прокси сервер арқылы өтетін болса, оның трафиктерді талдау мен өңдеу мүмкіндіктері өте жоғары болады. Трафиктерді үнемді пайдалану үшін мағлұматтарды сығуға немесе сұраныс нәтижелерін бір жадыда сақтауға болады. Прокси сервердің көмегімен желі арқылы таратылатын хабарламаларды реттестіруге және бақылап отыруға болады.

Түйін сөздер: C# программалау тілі, *HTTP* прокси сервері, System.Text.Encoding NET қосымшасы, ItemBase.

Г.А.Мирзахмедова, А.К.Самбетбаева

Разработка прокси сервера в языке программирования C#

Резюме. Прокси-сервера позволяют организовывать передачу данных между клиентами, другими серверами и сетевыми службами. Прокси-сервера можно использовать для распределения доступа в Интернет в локальных сетях, для учета и обработки трафика, для отладки сетевых приложений, да и много еще для чего. В этой статье пойдет речь о создании собственного HTTP прокси–сервера при помощи языка программирования C#, с использованием сокетов.

Ключевые слова: Язык программирования C#, прокси сервер *HTTP*, System.Text.Encoding приложение NET, ItemBase.

Г.А.Мирзахмедова, А.Қ.Самбетбаева

Разработка прокси сервера в языке программирование C#

Summary. Proxy servers allow to organize data transfer between clients, servers, and other network services. Proxy servers can be used to distribute Internet access LANs for recording and processing traffic for debugging network applications and a lot more for that. In this article we will talk about creating your own HTTP proxy server using the programming language C #, using sockets.

Key words: Programming Language C #, proxy HTTP, System.Text.Encoding application. NET, ItemBase.

УДК 537.f

Анн характерис способ ан; значениям металличеі скорости < данных. В его осуще простоте и Кл* озона, бал