DESENVOLVIMENTO DE UM SERVIDOR WEB MULTITHREADED

Aldo Ventura da Silva*

RESUMO

Este trabalho busca esclarecer as decisões de projeto no desenvolvimento de um servidor web multithreaded, baseando-se na Norma RFC 1945. Dividimos a explicação do código por suas classes e métodos. O código possui diversos comentários, explicando as classes, métodos e objetos, sendo assim, os conceitos do aplicativo. Dividimos o código pelas classes e traremos uma explicação básica sobre a codificação do método Base64Coder.

Esse trabalho foi desenvolvido na tecnologia Java, abrangendo sua biblioteca.

Palavras-chave: Multithread, Servidor Web, Java

* Bacharel em Sistemas de Informação – Universidade de São Paulo (USP)

1 INTRODUÇÃO

Com a evolução computacional obtivemos um grande aumento de comunicação entre dispositivos computacionais, entre essas comunicações temos a internet juntamente com os servidores web.

Um servidor web é responsável por fornecer aos computadores que requisitam informações, também chamados de “clientes”, os dados solicitados; e também receber informações do mesmo, para que seja efetuada alguma tarefa. Através de um servidor web o cliente pode requerer dados, enviar dados, solicitar a execução de processos e também executá-los. Como tarefas dos servidores web têm o envio de dados para o cliente, acesso a base de dados, e execução de processos. Essas tarefas são executadas através de conexões entre o servidor e o cliente, que é conhecida como sessão HTTP.

Assim, objetivo deste trabalho é esclarecer as decisões de projeto no desenvolvimento de um servidor web multithreaded, baseando-se na Norma RFC 1945. Para uma explicação mais didática, dividiremos a explicação do código, que foi desenvolvido na linguagem de programação Java, por suas classes e métodos. O código possui diversos comentários, explicando diversas classes, métodos e objetos, sendo assim, os conceitos do aplicativo. Por esse motivo não explicaremos novamente, pois está explicito no código.

2 Desenvolvimento do Código

Um servidor WEB deve tratar as solicitações de seus clientes, logo, o formato esperado dessas solicitações segue um padrão definido pelo protocolo HTML. As funcionalidades mínimas que um servidor WEB deve possuir são as mensagens GET e POST.

A mensagem GET faz uma requisição a uma página web. Um exemplo típico:

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/3.0Gold (WinNT; I)
Host: hypothetical.ora.com
Accept: image/gif, image/x-bitmap, image/jpeg, */*
```

O servidor deve interpretar os dados recebidos e enviar uma resposta condizente. A seguir tem-se um exemplo de mensagem resposta enviada ao cliente:

```
HTTP/1.0 200 OK
Server: NSCA/1.4.2
MIME-version: 1.0
Content-type: text/html
Content-lenght: 46
<html>
<Title>
A pagina HTML simples
</Title>
</html>
```

O primeiro passo é a criação de uma classe (WebServer) para o tratamento geral do servidor. Esta classe faz a conexão com o cliente e deixa o tratamento sobre responsabilidade de outra classe.

Para iniciar a conexão, é criado um socket Servidor usando a classe ServerSocket do pacote Java.net. No seu construtor é passada a porta em que se deseja “ouvir” a conexão.

Depois de feita a criação do socket, é usado o método accept() da classe ServerSocket para que ela fique “escutando” o pedido de conexão.

Depois de realizada a conexão, esse socket é passado para outra classe que ira tratar de todo o processamento, chamada HttpRequest, através de seu construtor.

A classe HttpRequest é passada para uma nova Thread, que é iniciada. Sendo assim, temos o código inteiro dentro de um loop. Podendo atentar a diversas solicitações em concorrência. Temos na classe HttpRequest todo o tratamento dos métodos POST e GET.

Seu construtor recebe um socket que irá obter a conexão e assim fazer a troca de arquivos. Para o uso de threads, o código deve possuir a classe Runnable na implementação.

Deve-se acessar as cadeias de dados, sendo assim, utiliza-se os métodos getOutputStream() e o getInputStream() da classe Socket. Eles são armazenados em buffers, das classes correspondentes BufferedReader e a DataOutputStream. Com elas temos o acesso aos dados enviados pelo cliente, pelo método readLine(). Para devolvermos dados, utilizamos o método writeBytes().

Para identificar qual método está sendo utilizado pelo browser, usamos a classe chamada StringTokenizer, que através do método nextToken() retorna a substring com as menssagens desejadas. Assim, o primeiro nextToken seria, de acordo com o protocolo, qual é o tipo de método usado pelo navegador (GET, POST, etc). Com esse valor em mãos, podemos codificar as respostas dos métodos.

Se for um GET, temos que pegar o arquivo desejado. Com o nome do arquivo em mãos (através do método nextToken()), usamos a classe File para fazer a referência ao arquivo desejado. Após, uma chamada ao método write() da classe OutputStream é o suficiente. Caso seja um POST, é feita a captura dos dados no corpo da mensagem do cliente e passadas para uma string.

Caso o arquivo requisitado pelo GET não for encontrado, trataremos essa exceção pelo método catch() com a página HTML de arquivo não encontrado.

Caso o arquivo esteja em um nível protegido, apresenta-se a autenticação de usuários. Caso o diretório seja protegido, deve-se enviar uma menssagem do tipo 401 Unauthorized para o cliente e então pedir a autenticação do tipo Basic. Assim, será enviado de volta ao cabeçalho a Authorization, uma string com o nome de usuário e password. Confira se a senha é a requisitada, caso contrario, mande de volta o erro 401.

A decodificação é feita através do código Base64. Ele é feito por um alfabeto de 64 caracteres. São escolhidos para serem universalmente legíveis e para não possuir significado nos principais protocolos de serviço de mensagens.

O arquivo de log é definido por uma nova classe do tipo DataOutputStream, e todas as mensagens do browser para o servidor são registradas no arquivo. Para obter o IP do cliente, basta usar o método getInetAddress() e getHostAddress() e utilizar o writeBytes() para salvar no log.

2.1 Código Fonte

Abaixo temos o Código fonte do Servidor Web, onde a senha, quando for necessária a Autenticação do diretório “RESTRITO” ou “restrito” é:

usuário: redes

senha: computadores

O código encontra-se com diversos comentários auxiliando o entendimento.

Para a decodificação da senha requisitada pelo servidor web utilizamos o código Base64Coder, que é código aberto e de distribuição livre com licença GNU. Esse código auxilia o desenvolvimento do servidor web, também utilizamos somente o método decode da classe Base64Coder para verificar a autenticação da senha. O código Base64Coder, com os métodos utilizados será mostrado logo abaixo, em tópico próprio.

2.1.1 WebServer.java

```
import java.net.*;

public final class WebServer {
    //método que executa o inicio do programa
    public static void main (String arvg[]) throws Exception{
          //Porta 8181, escolhida sem motivo especial
          int porta = 8181;
          //Estabelece o socket de escuta do Servidor
          ServerSocket socketServ = new ServerSocket(porta);
          //Estabelece o socket do cliente
          Socket socketCli;
          //Processar a requisicao do servico HTTP em um laco infinito
          while (true) {
                //Apenas um mensagem para indicar que o servidor
encontra-se ativo
                System.out.println( "Servidor Ativo" );
                //Escutar requisicao de conexao TCP.
                socketCli = socketServ.accept();
                //Constroi um objeto para processar a mensagem de
requisicao HTTP
                HttpRequest requisicao = new HttpRequest(socketCli);
                //Criado um novo thread para processar as novas
requisicões
                Thread thread = new Thread (requisicao);
                //Inicia o thread.
                thread.start();
          }
    }
}
```

2.1.2 HttpRequest.java

```
import java.io.*;
import java.net.*;
import java.util.*;

final class HttpRequest implements Runnable {
    //Carriage Return + Line Feed
    final static String CRLF = "\r\n";
    //referência do socket da conexão
    Socket socket;
    //senha para autenticacao de pasta protegida
    private String senhaParaAutenticacao = "redes:computadores";

    //Construtor
    public HttpRequest (Socket socket) throws Exception{
          this.socket = socket;
    }
    // roda processrequest e trata as exceções
    public void run(){
          try{
                processRequest();
          }
          catch (Exception e) {
                System.out.println (e);
          }
    }
```

```
    private void processRequest () throws Exception{
          //Objeto isr referência para os trechos de entrada
          InputStreamReader isr = new
InputStreamReader(socket.getInputStream());
          //Objeto dos referência para os trechos de saida
          DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());
          //Ajustar os filtros do trecho de entrada
          BufferedReader br = new BufferedReader(isr);
          //Obter a linha de requisição da mensagem de requisição HTTP
          String requestLine = br.readLine();
          //Exibir a linha de requisição no Console
          System.out.println(); // pula uma linha
          System.out.println(requestLine);

          String headerLine = null;
          StringTokenizer senhaAutorizada = null;
          boolean ehAutenticada = false;
          boolean ehRestrito = false;

          //Dados que irao compor o Log
          String log = requestLine +
System.getProperty("line.separator");

          //Percorre todas linhas da mensagem
          while ((headerLine = br.readLine()).length() != 0) {
                //Obtendo linhas do cabecalho para log
                log = log + (headerLine +
System.getProperty("line.separator"));

                //pega a linha que possui a senha
                if(headerLine.contains("Authorization: Basic")){
                      senhaAutorizada = new
StringTokenizer(headerLine);
                      //Pula "Authorization: Basic"
                      senhaAutorizada.nextToken();
                      senhaAutorizada.nextToken();
                      //Pega senha na base64
                      String senha = senhaAutorizada.nextToken();

                      //Decodifica senha na base64
                      if(Base64Coder.decodeString(senha).equals
                                  (this.senhaParaAutenticacao)){
                            ehAutenticada = true;
                      }
                }
                System.out.println (headerLine);
          }
          //Extrair o nome do arquivo a linha de requisição
          StringTokenizer requisicao = new
StringTokenizer(requestLine);
          //pula método mostrado na requisicao (GET, POST)
          String metodo = requisicao.nextToken();
          String arquivo = requisicao.nextToken();
          //Acrescente um "." de modo que a requisição do arquivo
          //esteja dentro do dirtório atual
          arquivo = "." + arquivo;
          //Abre o arquivo requisitado
          FileInputStream fis = null;
          //Verifica existencia do arquivo
```

```
            boolean existeArq = true;

            //Construir a mensagem de resposta
            String linhaStatus = null;
            String linhaContentType = null;
            String msgHtml = null;


            //tratamento quando a requisicao for GET
            if (metodo.equals("GET")){
                  try{
                        fis = new FileInputStream(arquivo);
                  }
                  catch(FileNotFoundException e){
                        existeArq = false;
                  }
                  //Verifica se o arquivo é RESTRITO, forma colocadas
apenas
                  //duas verificações de permissão (RESTRITO, restrito)
                  if (arquivo.contains("RESTRITO")||
                        arquivo.contains("restrito")){
                        ehRestrito = true;
                  }
                  //Verifica se é restrito o local que está sendo
requisitado
                  //informações e se não foi autenticado
                  if((ehRestrito) && (ehAutenticada) == false){
                        linhaStatus = "HTTP/1.0 401 Unauthorized" + CRLF;
                        linhaContentType = "WWW-Authenticate: Basic
realm=\"RESTRITO\""+
                        CRLF;
                        msgHtml = "<HTML><HEAD><TITLE> Acesso Nao
Autorizado " +
                        "</TITLE></HEAD>" +
                        "<BODY> Acesso Nao Autorizado </BODY></HTML>";
                        existeArq = false;
                  }
                  else{
                        if(existeArq){
                              linhaStatus = "HTTP/1.0 200 OK" + CRLF;
                              linhaContentType = "Content-type: " +
contentType(arquivo)+
                              CRLF;
                        }
                        else{
                              linhaStatus = "HTTP/1.0 404 Not found" +
CRLF;
                              linhaContentType = "Content-type: " +
contentType(arquivo)+
                              CRLF;
                              msgHtml = "<HTML><HEAD><TITLE> Arquivo Nao
Encontrado" +
                              "</TITLE></HEAD>" +
                              "<BODY> Arquivo Nao Encontrado
</BODY></HTML>";
                        }
                  }
                  //Enviar a linha de status
                  dos.writeBytes(linhaStatus);
                  //Enviar linha de tipo de conteúdo
                  dos.writeBytes(linhaContentType);
```

```
                //Enviar uma linha em branco para indicar o fim das
linhas de
                //cabeçalho
                dos.writeBytes(CRLF);
                //Enviar corpo do Html
          }
          //tratamento quando a requisicao for POST
          else if (metodo.equals("POST")){
                //Obtem o corpo do pacote POST
                char[] buffer = new char[2048];
                String corpo = "";
                String corpoPost = "";
                while(br.ready())
                {
                      int i;
                      if((i = br.read(buffer)) > 0)
                            corpo = corpo + (new String(buffer,0,i) +
"\n");
                      corpoPost = corpoPost + (new String(buffer,0,i) +
"<BR>");
                      System.out.println(corpo);
                }

                corpo = "CORPO ENVIADO PELO POST: " + corpo;
                log = log + corpo;

                msgHtml = "<HTML><HEAD><TITLE> MENSAGEM POST
</TITLE></HEAD>" +
                "<BODY> MENSAGEM ENVIADA PELO POST: </BR>" + corpoPost
+
                "</BODY></HTML>";

                existeArq = false;
          }
          if(existeArq){
                sendBytes(fis, dos);
                fis.close();
          }
          else{
                dos.writeBytes(msgHtml);
          }
          Log(dos, log, socket);
          dos.close();
          br.close();
          socket.close();
    }

    private void sendBytes(FileInputStream fis, DataOutputStream os)
    throws Exception {
    //Construir um buffer de 1k para comportar os bytes no caminho para
o socket
          byte[] buffer = new byte[1024];
          int bytes = 0;
          //Copiar o arquivo requisitado dentro da cadeia de saída do
socket
          //enquanto o arquivo não estiver no fim, ou seja, -1..copie
          while((bytes = fis.read(buffer)) != -1){
                os.write(buffer, 0, bytes);
          }
    }
```

```
    private static String contentType(String arquivo){
          if(arquivo.endsWith(".htm")||
                      arquivo.endsWith(".html")||
                      arquivo.endsWith(".txt")) return "text/html";
          if(arquivo.endsWith(".gif")) return "image/gif";
          if(arquivo.endsWith(".jpeg")) return "image/jpeg";
          //caso a extensão do arquivo seja desconhecida
          return "application/octet-stream";
    }

    private void Log(DataOutputStream dos, String log, Socket socket) {
          try{
                //Data de requisicao
                Date date = new Date(System.currentTimeMillis());
                String dataRequisicao = date.toString();
                String pulaLinha =
System.getProperty("line.separator");

                FileWriter fw = new FileWriter("arquivo_de_log.txt",
true);
                fw.write("-----------------------------------------------
---------" +
                            pulaLinha);
                fw.write("Data de Requisicao: " + dataRequisicao + "
GMT " +
                            pulaLinha);
                fw.write("ENDEREÇO DE ORIGEM:PORTA: " +
                            socket.getLocalSocketAddress().toString() +
pulaLinha);
                fw.write("Conteúdo Requisitado: "+ log + pulaLinha);
                fw.write("Quantidade de bytes transmitidos: " +
dos.size() +
                            pulaLinha);
                fw.write("-----------------------------------------------
---------" +
                            pulaLinha);
                fw.write(pulaLinha);
                fw.close();
          }
          catch(IOException io){
                System.out.println(io.getMessage());
          }
    }
}
```

2.1.3 Base64Coder.java

Como citado anteriormente pela complexidade do código e por não ser obrigatório seu desenvolvimento escolhemos utilizar o código abaixo para auxiliar no desenvolvimento do servidor web, essa classe possui apenas a decodificação e também comentários originais. No Base64, os sessenta e quatro símbolos foram escolhidos para serem universalmente legíveis e para não possuírem significado nos principais protocolos de serviço de

mensagens. Os caracteres são de A-Z, a-z,0-9, “/”, “+”, e o caractere “=” é utilizado como um sufixo especial.

```
// Copyright 2003-2010 Christian d'Heureuse, Inventec Informatik AG,
Zurich, Switzerland
// www.source-code.biz, www.inventec.ch/chdh
//
// This module is multi-licensed and may be used under the terms
// of any of the following licenses:
//
//  EPL, Eclipse Public License, http://www.eclipse.org/legal
//  LGPL, GNU Lesser General Public License,
http://www.gnu.org/licenses/lgpl.html
//  AL, Apache License, http://www.apache.org/licenses
//  BSD, BSD License, http://www.opensource.org/licenses/bsd-
license.php
//
// Please contact the author if you need another license.
// This module is provided "as is", without warranties of any kind.



/**
* A Base64 encoder/decoder.
*
* <p>
* This class is used to encode and decode data in Base64 format as
described
* in RFC 1521.
*
* <p>
* Project home page: <a href="http://www.source-
code.biz/base64coder/java/">
* www.source-code.biz/base64coder/java</a><br>
* Author: Christian d'Heureuse, Inventec Informatik AG, Zurich,
Switzerland<br>
* Multi-licensed: EPL / LGPL / AL / BSD.
*/
public class Base64Coder
{

    // Mapping table from 6-bit nibbles to Base64 characters.
    private static char[] map1 = new char[64];
    static
    {
          int i = 0;
          for (char c = 'A'; c <= 'Z'; c++) map1[i++] = c;
          for (char c = 'a'; c <= 'z'; c++) map1[i++] = c;
          for (char c = '0'; c <= '9'; c++) map1[i++] = c;
          map1[i++] = '+'; map1[i++] = '/';
    }

    // Mapping table from Base64 characters to 6-bit nibbles.
    private static byte[] map2 = new byte[128];
    static
    {
          for (int i = 0; i < map2.length; i++) map2[i] = -1;
          for (int i = 0; i < 64; i++) map2[map1[i]] = (byte)i;
    }
```

```
    /**
* Decodes a string from Base64 format.
* No blanks or line breaks are allowed within the Base64 encoded input
data.
* @param s  A Base64 String to be decoded.
* @return   A String containing the decoded data.
* @throws   IllegalArgumentException If the input is not valid Base64
encoded
* data.
*/
    public static String decodeString(String s)
    {
          return new String(decode(s));
    }

    /**
* Decodes a byte array from Base64 format and ignores line separators,
tabs
* and blanks.
* CR, LF, Tab and Space characters are ignored in the input data.
* This method is compatible with
* <code>sun.misc.BASE64Decoder.decodeBuffer(String)</code>.
* @param s  A Base64 String to be decoded.
* @return   An array containing the decoded data bytes.
* @throws   IllegalArgumentException If the input is not valid Base64
* encoded data.
*/
    public static byte[] decodeLines(String s)
    {
          char[] buf = new char[s.length()];
          int p = 0;
          for (int ip = 0; ip < s.length(); ip++)
          {
                        char c = s.charAt(ip);
                        if (c != ' ' && c != '\r' && c != '\n' &&
c != '\t')
                              buf[p++] = c;
          }
                return decode(buf, 0, p);
    }

          /**
* Decodes a byte array from Base64 format.
* No blanks or line breaks are allowed within the Base64 encoded input
data.
* @param s  A Base64 String to be decoded.
* @return   An array containing the decoded data bytes.
* @throws   IllegalArgumentException If the input is not valid Base64
encoded
* data.
*/
          public static byte[] decode(String s)
    {
                return decode(s.toCharArray());
    }

          /**
    * Decodes a byte array from Base64 format.
    * No blanks or line breaks are allowed within the Base64 encoded
input data.
```

```
    * @param in  A character array containing the Base64 encoded data.
* @return    An array containing the decoded data bytes.
* @throws    IllegalArgumentException If the input is not valid Base64
encoded
* data.
*/
    public static byte[] decode(char[] in)
    {
          return decode(in, 0, in.length);
    }

    /**
* Decodes a byte array from Base64 format.
* No blanks or line breaks are allowed within the Base64 encoded input
data.
* @param in    A character array containing the Base64 encoded data.
* @param iOff  Offset of the first character in <code>in</code> to be
processed.
* @param iLen  Number of characters to process in <code>in</code>,
starting at
* <code>iOff</code>.
* @return      An array containing the decoded data bytes.
* @throws      IllegalArgumentException If the input is not valid
Base64
* encoded data.
    */
          public static byte[] decode(char[] in, int iOff, int iLen)
    {
                  if (iLen % 4 != 0) throw new
IllegalArgumentException("Length" +
                  "of Base64 encoded input string is not a multiple of
4.");
                  while (iLen > 0 && in[iOff + iLen - 1] == '=') iLen-
-;
                  int oLen = (iLen * 3) / 4;
                  byte[] out = new byte[oLen];
          int ip = iOff;
          int iEnd = iOff + iLen;
          int op = 0;
          while (ip < iEnd)
          {
                int i0 = in[ip++];
                int i1 = in[ip++];
                            int i2 = ip < iEnd ? in[ip++] : 'A';
                            int i3 = ip < iEnd ? in[ip++] : 'A';
                            if (i0 > 127 || i1 > 127 || i2 > 127 ||
i3 > 127)
                                        throw new
IllegalArgumentException("Illegal" +
                                        "character in Base64 encoded
data.");
                            int b0 = map2[i0];
                            int b1 = map2[i1];
                int b2 = map2[i2];
                int b3 = map2[i3];
                if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
                     throw new IllegalArgumentException("Illegal
character in" +
                     "Base64 encoded data.");
                int o0 = (b0 << 2) | (b1 >>> 4);
                int o1 = ((b1 & 0xf) << 4) | (b2 >>> 2);
```

```
                                int o2 = ((b2 & 3) << 6) | b3;
                                out[op++] = (byte)o0;
                                if (op < oLen) out[op++] = (byte)o1;
                                if (op < oLen) out[op++] = (byte)o2;
            }
                        return out;
        }

    } // end class Base64Coder
```

Conclusão

Neste projeto foi desenvolvido um servidor web multithreaded de acordo com as normas da RFC 1945, que possui as funcionalidades básicas que um servidor necessita, além disso, possui um diretório denominado restrito, que para ser acessado necessita de autenticação, além disso, todas as operações efetuadas pelo servidor são salvas em um registro, que denominamos Arquivo_Log.txt.

Servidores multithreaded são capazes de processar diversas requisições simultaneamente em paralelo, sendo de extrema importância para a era da Web 2.0 que vivenciamos atualmente. Servidores populares como APACHE, WINDOWS NT e LINUX, possuem tratamento multithreaded, e é amplamente utilizado.

REFERÊNCIAS

D'HEUREUSE, Christian. **Base64Coder.** Disponível em: http://www.source-code.biz/base64coder/java. Acesso em 18 de Setembro de 2010.

KUROSE, James F.. **Redes de Computadores e a Internet.** 10. Ed. São Paulo: Pearson Prentice Hall, 2010.

MORIMOTO, Carlos E.. **Servidoers Linux, Guia Prático.** 1. Ed. GHD Press e Sul Editores, 2009.

MORIMOTO, Carlos E.. **Redes, Guia Prático.** 1. Ed. GHD Press e Sul Editores, 2008.

RFC 2068, **Hypertext Transfer Protocol -- HTTP/1.1.** Disponível em: http://www.w3.org/Protocols/rfc2616/rfc2616.html. Acessos em: 18 de Agosto de 2014.

SIERRA, Kathy. **Use a Cabeça! Java.** 2. Ed. São Paulo: Alta Books, 2008.

STENZEL, Sebastian. **Base64 encoder/decoder in Java.** Disponível em: http://www.source-code.biz/snippets/java/2.htm. Acesso em: 18 de Setembro de 2010.