文章编号:1008-312X(2001)01-0029-04

用 Java 实现一个多线程 HTTP 服务器

宋 波,马 黎,孙连科
(沈阳电力高等专科学校 信息工程系,辽宁 沈阳 110036)

摘要:作为服务器应用的一个必需机能,就是要能同时处理多个客户机的请求。但是,由于客户机的请求而使服务器过载,就有可能造成服务器的崩溃。本文说明了如何设计安全的多线程服务器应用程序,并以一个 HTTP 服务器的开发作为实例。

关 键 词:Java;多线程;HTTP 服务器
中图分类号:TP316.4 文献标识码:A

随着 Internet/ Intranet 的普及以及 Web 技术的迅速发展,开发基于 Browser/ Server 模式的 Web 数据库信息检索系统得到了广泛的应用。Java 所具有的良好兼容性、安全性及跨平台等特性,使得 Java 成为开发基于 B/S 模式的 Web 数据库应用的首选计算机语言。

这种模式的客户端的浏览器可以从普遍的 HTTP 服务器(如 Apache)上下载超文本界面,同时下载并执行内嵌在页面中的客户程序代码 Java Applet。

Web 服务器端主要由普通的 HTTP 服务器、Java 多线程 HTTP 服务器及数据库服务器三部分组成,其主要功能是监听并接受来自客户端的信息,并对其进行分析处理,与 Web 数据库建立连接,完成系统的各项功能。Java 多线程 HTTP 服务器使用 ServerSocket 在服务器专用的 TCP 端口监听接收来自客户方 Applet 程序发出的请求。当有一个连接请求时,多线程服务器生成一个新线程,处理客户的连接直至退出。服务器的主线程仍处于监听状态,等待新的客户请求。

从上述的系统结构体系功能分析中可以看出,在基于 B/S 模式的 Web 数据库应用开发中,Java 多线程 HTTP 服务器程序的设计占有很重要的位置。本文通过作者的工作实践,详细阐述了如何开发一个安全的多线程 HTTP 服务器的方法和步骤。

1 线 程

线程(Thread)这个术语是从操作系统的执行线程派生来的。是指程序中能顺序执行的一个序列。一个线程只有一个入口点,但可能有几个出口点,但在每个时刻的执行点总是只有一个。多线程(Multi-Thread)是 Java 的一个重要特性,指的是通过系统的调度使几个具有不同功能的程序流即线程同时并行地运行。

在多线程程序设计中,每个线程都用编码提供线程的行为,用数据供给编码操作,同一个程序中的所有线程共享一个数据段。而 Java 从语言一级提供对多线程的支持,可由语言和运行系统联合提供对共享数据段的管理功能和同步机制,这样就使得多线程并行程序设计相对比较容易。

在 Java 中,当我们创建一个新线程时,先通过对 Java.lang.Thread 类的继承来派生一个子类。

```
class MyApplication extends Thread{
    …
}
```

然后,通过 run() 方法来实现线程的行为。

```
class MyApplication extends Thread{
    public void run(){
        …
    }
}
```

最后,由子类生成一个对象,并且进行启动操作,这样就得到一个处于可运行状态的线程。生成对象是完成线程的创建,启动则是对已经创建的线程进行操作。

```
MyApplication ex = new MyApplication();
Thread th = new Thread(ex);
th.start();
```

收稿日期:2000-09-18
作者简介:宋 波(1963-),男,山东黄县人,沈阳电力高等专科学校讲师,硕士.

2 HTTP 简介

HTTP(HyperText Transfer Protocol) 是超文本传输协议的简称,是 WWW 上用于发布信息的主要协议。HTTP 协议是基于 TCP/ IP 协议之上的应用层协议,具有简单、无状态、灵活、元信息及无连接等特点。

HTTP 定义了一个 Web 浏览器/ 服务器结构的简单事务处理,这个处理由以下四步组成:

1) 客户与服务器建立连接;

2) 客户向服务器递交请求。HTTP 请求一般是 GET 或 POST 命令;

3) 如果请求被接纳,那么服务器送回一个应答,应答中至少包括状态编码和文件的内容;

4) 客户或服务器断开连接。

由浏览器向 Web 服务器发出请求时,它就向服务器传递一个数据报文,也就是请求元信息。元信息包括:提出请求的浏览器名;浏览器能接受的数据类型等。在每次建立连接时客户机方和服务器方必须对它们的数据表示达成协议。

HTTP 请求格式的例子如下:

```
GET/ path/ file .html HTTP/ 1 .0          请求行
Accept :text/ html
Accept :audio/ x                          请求头部域
User- agent :MacWEB
```

HTTP 应答格式的例子如下:

```
HTTP/ 1 .0 200 OK                         应答头
Server :Apache/ 1 . 1                     应答
Mime vertion :1 .0                        头部域
Content_type : text/ html
Content_length :2000
〈HTML〉                                   实体数据
…                                         (HTML 文件)
〈HTML〉
```

3 多线程 HTTP 服务器的实现

在 Java 的客户/ 服务器应用中,客户与服务器之间的通讯一般是采用基于 TCP/ IP 的 Socket 机制来实现的。Socket 是客户与服务器的程序间进行双向传输的网络通讯端点,在服务器程序中通过 IP 在网络中标识自己,通过一个客户端程序所知的端口号来提供服务。客户端在网络中通过服务器的 IP 地址找到服务器,通过端口号获得服务器的服务。

根据 Java 的 Socket 通讯机制及 HTTP 协议的作用原理,实现 GET 请求的 HTTP(Web) 服务器程序的步骤如下:

1) 创建 ServerSocket 类对象,监听端口 8090 (Web 服务器的缺省端口号为 80);

2) 等待、接受客户机连接到端口 8090,得到与客户机连接的套接字(Socket) 。这是通过调用 ServerSocket 类的 accept( ) 方法来实现的;

3) 创建与 Socket 相关联的输入流 InputStream 和输出流 OutputStream;

4) 从与 Socket 关联的输入流 InputStream 中读取一行客户机提交的请求信息,请求信息的格式为:GET path/ filename HTTP/ 1 .0;

5) 从请求信息中获取请求类型。如果是 GET,则从请求信息中获取所访问的 HTML 文件名。缺省为 index .html;

6) 如果 HTML 文件存在,则打开 HTML 文件,把 HTTP 的头信息和 HTML 文件内容通过 Socket 传回给 Web 浏览器,然后关闭文件。否则发送错误信息给 Web 浏览器;

7) 关闭与相应 Web 浏览器连接的 Socket。

下面举一个实例,具体地阐述一下多线程 HTTP 服务器的实现过程。

这个多线程服务器包含有主类 MultiServer 和连接类 Connect。

MultiServer 类扩展 Thread 类,但是它的对象是单线程的,为实现监听多个客户机的连接请求,其 run( ) 方法的定义如下:

```
//multi- threading-- create a new connection for each re-
quest .
public void run( ) {
  try{
    while( true) {
      Socket client = listen .accept( ) ;
      Connect cc = new Connect( client) ;
    }
  } catch( IOException e) {
    System .out .println( "Exception …"+ e) ;
    }
}
```

客户机每次连接时,ServerSocket 都将创建新的 Socket(套接字) 及生成新的线程。而为了启动线程,就必须调用 start( ) 方法。但调用 start( ) 方法前,应该生成 ServerSocket 对象。因此,Multi-

Server 类的定义如下:

```
public class MultiServer extends Thread{
 //The port number on which the server will be listening on .
   public static final int HTTP_PORT = 8090;
   protected ServerSocket listen;
    //constructor.
   public MultiServer(){
      try{
         listen = new ServerSocket(HTTP_PORT);
      }catch(IOException ex){
           System.out.pritnln("Exception…"+ ex);
         }
      this.start();
   }
      public void run(){
    //the code for run() from above.
   }
    //main program.
   public static void main(string argv[]) throws
IOException{
      new MultiServer();
   }
}
```

类 Connect 也扩展 Thread 类,其功能是初始化通信的数据流,其 run()方法用来处理与客户端的通信,并向客户提供信息服务。其定义如下:

```
public void run(){
   try{
    //get a request and parse it.
      String request = is.readLine();

      System.out.println("Request:"+ request);
      StringTokenizer tt = new
StringTokenizer(request);
      if((tt.countTokens() >= 2) &&
tt.nextToken().equals("GET")){
      if((request = tt.nextToken()).startsWith("/"))
         request = request.substring(1);
      if(request.endsWith("/") ‖request.equals(""))
         request = request + "index.html";
      File f = new File(request);
         showDoc(os,f);
   }else{
         os.writeBytes("400 Bad Request");
      }
         client.close();
   }catch(IOException e){
         System.out.println("I/O error"+ e);
      }catch(Exception ex){
         System.out.println("Exception:"+ ex);
      }
}
```

run()方法在 start()方法被调用之前是一直保持无效状态的,所以我们应该把 start()方法放置在 Connect 类的构造函数中。而调用 start()方法前,我们还应当初始化通信用的数据流。因此,Connect 类的定义如下:

```
class Connect extends Thread{
   Socket client;
   BufferedReader is;
   DataOutputStream os;
      public Connect(Socket s){ //constructor.
      client = s;
      try{
         is = new BufferedReader(new
InputStreamReader(client.getInputStream()));
         os = new
DataOutputStream(client.getOutputStream());
      }catch(IOException e){
      try{
         client.close();
      }cach(IOException ex){
         System.out.println("Error while getting socket
streams.."+ ex);          }
      return;
      this.start(); //Thread starts here…this start()
will call run()
   }
   public void run(){
       //code from above goes here…  }
    //Read the requested file and shows it to the browser if found.
   public static void showDoc(DataOutput
   Stream out,File f) throws Exception{
   try{
         DataInputStream in = new
DataInputStream(new FileInputStream(f));
```

```
        int len = (int) f.length() ;
        byte[ ] buf = new byte[ len] ;
        in.readFully( buf) ;
          out.write( buf, 0, len) ;
          in.close( ) ;
        } catch( FileNotFoundException e) {
          out.writeBytes( "404 Not Found") ;
        } catch( IOException ex) {
        System.out.println( "error writing ···"+ ex) ;}
    }
}
```

4 结束语

Java 作为在 Internet 或局域网上的主流网络编程语言,提供了对网络支持的无平台相关性的完整软件包,可简化服务器程序设计的复杂性。同时,由于 Java 本身是“线程安全的”,因此,在实现服务器程序时采用多线程设计,可以改善整个服务器应用系统的响应,提高程序的运行效率,降低客户的等待时间。

参考文献:

[1] ElliotteRustyHarold. Java Networking Programming[M]. America: O' REILLY, 1997.

[2] [美] Philip Heller 著,邱仲潘等译. Java 2 高级开发指南[M]. 北京:电子工业出版社,1999.

Implementing a multi-threaded HTTP server in Java

SONG Bo, MA Li, SUN Lian-ke

(Department of Information engineering, Shenyang Electric Power Institute, Shenyang 110036, China)

Abstract: For server applications to be useful, they should have the ability to serve multiple clients simultaneously. However, clients' requests may overload the server's machine and the server may crash. Explains how to write safe multi-threaded server applications. The development of an HTTP server is used as an illustration.

Key words: Java; multi-thread; HTTP server

(责任编辑 翟 春)