Java Applet与Servlet通讯机制的探讨

上海同济大学计算中心 胡 泳 张志浩 陈福民

本文对基于JAVA技术的WEB应用服务器的重要组成部分：Java Applet和Java Servlet之间的通信机制做了比较详细的论述，并且给出了实例说明。

引 言

Internet 网的火爆滋养了WEB应用服务器，JAVA语言固有的特点使其在开发网络环境下的分布式多层结构应用中有着不可比拟的优势，成为构建WEB应用服务器的首选技术。特别是近来Java Servlet的出现与发展，使JAVA在服务器端的功能有了很大的提高，而且目前各大公司如ORACLE，SUN，IBM等都在其WEB应用服务器中加强了对Java Servlet的支持，使Java Servlet显示出强劲的发展势头。在浏览器端使用Java Applet可以使用户界面更加友好，实时控制功能更加强大，Applet与Servlet的结合会使整个WEB应用服务体系功能、效率更加强大。但是Applet 与Servlet分别运行在客户端和服务器端，两者之间的通信就变的至关重要了，本文的主要目的就是对两者的通信机制进行论述、探讨。

Java Applet与Java Servlet简介

Java Applet是随着JAVA语言一起出现的，它的出现增强了Internet世界客户端的功能，不仅提供了类似与传统C/S结构友好的用户界面，而且可以执行一些标准HTML语言无法完成的工作，如拖拉操作等，使浏览器的功能有了很大的增强;另外，几乎所有的浏览器（包括微软的IE和网景的Netscape）都内置JAVA虚拟机，提供了对JavaApplet的支持，这样就保持了客户机配置简单的特点，有利于JavaApplet在Internet上的发展。

Servlets是一组运行在服务器端的软件。它与Applet同是SUN公司提出的概念，不同的是Apple是运行在浏览器端的JAVA程序，由浏览器中的JAVA虚拟机来执行，有图形界面接口，受大多数浏览器的支持；而Servlets是运行在WEB服务器端的JAVA程序，由WEB服务器中内嵌或外带的JAVA虚拟机来解释执行，没有图形界面接口，正逐渐被越来越多的WEB服务器支持。

Servlets与传统的服务端程序如CGI、API等虽然都运行在WEB服务器端，但从性能、通用性和安全性等许多方面多有了很大的提高。与CGI每接到一个客户请求便启动一个新的外部进程不同，Servlets采用线程的机制，即对于一个新的请求，不必启动新的进程，只是启动一个与当前进程位于同一内存空间的线程，这样既节省了服务器的资源又提高了它的响应速度，从整体上提高了系统的效率。另外，由于Servlets采用与平台无关的JAVA语言编写，而且目前绝大多数的WEB服务器都支持Servlets接口，因此Servlets是平台无关的，与只能在某个固定WEB服务器平台运行的API方式相比，它的通用性和移植性大大的提高了。

由于篇幅的限制，而且本文的重点也不是两者内部具体实现的介绍，所以这里就不对两者的实现机制做过多的介绍，如需要可以参阅相关的资料，下面着重对两者间的通信机制做些探讨。

Java Applet与Servlet通信机制的探讨

浏览器通过HTTP协议将Java Applet下载到本地后，Applet除了实现与用户的交互功能外，要与运行在WEB应用服务器上的Servlet通信，从而达到访问远端资源（如数据库等）的目的。一般两者之间的通信方式有三种：使用URL/URLConnection类、使用Socket和使用RMI。下面便结合一个获取服务器时间的例子分别对三种方式加以说明。

1. 使用 URL / URLConnection 类

URL是一种URL连接类，它提供了访问网络资源的方法，因为每一个Servlet的标识是唯一的（主机名：服务端口号：虚拟路径：名称），所以利用URL类可以与远端的Java Servlet通信。

下面给出利用URL/URLConnection类实现Applet与Servlet通信的示意图：

使用URL /URLConnection类时，只需要在Applet加入如下代码：

```
URL url = new URL(http://Servlet_host:port/servlet/TimeServlet + argString);
URLConnection connection=url.openConnection();
DataInputStream inStream=new DataInputStream(connection.getInputStream());
String date=inStream.readLine();
```

第一行代码首先利用URL类获取一个远端Servlet的统一资源定位符，argString是传递给Servlet的参数，Servlet可以利用它获得Applet的信息，这是Applet把本身信息传递给Servlet最常用的方法，本例中该参数为空。然后利用URL类的方法openConnection()建立一个与远端资源的连接，这样就可以利用DataInputStream类打开一个与该连接相对应的输入流，Applet就可以从输入流中获取Servlet传来的信息。

在Servlet端使用SevletRequest和ServletResponse两个类分别接收Applet请求和向Applet返回响应信息。相应的Servlet的代码如下：

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
{
    PrintWriter out = res.getWriter();
    out.println(getDate().toString());
    }
```

由于Applet利用URL类向Servlet打开一个连接时，向Servlet发出一个GET型HTTP请求，所以Servlet中的Service()方法把该请求分配给doGet()方法处理。doGet()方法有两个参数：HttpServletRequest 类型的req和HttpServletResponse类型的 res，前者用于接受从客户机发来的请求信息，后者用于向客户机发送回应信息。本例中由于无须从客户端获取任何附加的信息，故没有利用req参数。为了向客户端发送信息，首先获取一个输出流，由于本例向客户传递的是表示时间的字符串，故利用res提供的方法getWriter()打开一个输出流标示out，最后利用out向输出流输出系统当前时间，Applet中利用输入流的readLine()方法即可获得。关于URL 和URLConnection类的详细信息可以查阅JAVA API，以上只对相关的信息作了介绍。

URL/URLConnection类实现Applet与Servlet通信中所采用的通信协议都是应用层的HTTP协议，这样可以避免烦琐的底层编程，是一种最简便的方式。

2. 使用 Socket

Socket最早是UNIX操作系统中的概念，或来被广泛的应用在各种平台及系统中。在Internet/Intranet领域中Socket也是一个相当重要的概念。Socket由（主机名：服务端口号）唯一标志，例如同济大学WWW主机WWW服务的Socket就是www.tongji.edu.cn:80，80是WWW服务默认的端口号。利用Socket可以与相应的服务进行通信，在JAVA语言中，也引入了Socket概念，提供java.net.Socket 和java.net.ServerSocket类库，前者用于客户端，后者用于服务端。某种服务通过监听一个端口号，可以与向该端口发出请求的客户程序进行通信。Java Servlet由于是标准JAVA语言的扩展，所以与Applet可以利用Socket通信。

Applet中相关的代码如下：

```
Socket socket = new Socket(getCodeBase().getHost(), SEVRLET_PORT);
InputStream in = socket.getInputStream();
DataInputStream result = new DataInputStream(new BufferedInputStream(in));
String date = result.readLine();
```

第一行代码中，首先创建一个Socket对象，该Socket的端口号为服务器端Servlet所监听的端口号，主机为该Applet的宿主机（即此Applet是由该主机下载来的）。因为Java Applet的安全性限制，Applet只允许向其宿主机打开Socket连接，如果想与任何主机建立Socket连接，就需要对Applet进行数字签名。有关数字签名问题可以查看相关的资料，这里就不多谈，只考虑向宿主机打开

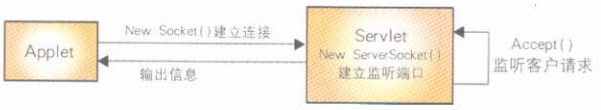
Socket的情况。创建好Socket连接后，在Applet 和Servlet之间就存在了一条双向通道，这样就可以利用Socket类的getInputStream()方法获得该通道的一个输入流，由于该方法返回的是一个InputStream类型抽象类，还需要将其转化为可以供程序调用的实体类DataInputStream。最后可以利用DataInputStream类的实例来接收Servlet传来的信息。

Servlet端的代码比较复杂，需要引入线程来支持相应的操作，其程序代码如下：

```
public class DaytimeServlet extends HttpServlet{
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
   ServerSocket serverSocket = new ServerSocket(SERVLET_PORT);
    while (true)
    // 在端口SERVLET-PORT上监听客户请求
    this.handleClient(serverSocket.accept());
    }
  public void handleClient(Socket client) {
    new DaytimeConnection(this, client).start();
  }
}
class DaytimeConnection extends Thread {
  DaytimeServlet servlet;
  Socket client;
  DaytimeConnection(DaytimeServlet servlet, Socket client) { // 构造函数
    this.servlet = servlet;
    this.client = client;
    setPriority(NORM_PRIORITY - 1);
  }
  public void run() {
    PrintStream out = new PrintStream(client.getOutputStream()); // 生成输出流
    out.println(new Date().toString()); // 向客户端输出信息
    out.close();
    client.close();
```

程序的主类DaytimeServlet在其初始函数init()中在端口SERVLET_PORT上创建一个Server Socket类实例：Server Socket serverSocket=new ServerSocket(SERVLET_PORT)，然后使用accept( )方法在SERVLET_PORT端口上监听客户请求，accept( )在接到客户请求之前一直处于封锁状态。一旦接收到客户请求，就击活DaytimeConnection类型的线程，获取客户程序Socket端口号，利用该端口号就可以与客户程序（这里是Applet）通信，具体代码如函数run()中所示。值得注意的是，在初始函数init()中使用了一个无限循环的while( )语句，这是因为server Socket在接收到一个客户请求并击活相应的线程后，还需要监听下一个客户请求，所以重新调用accept()方法，进入封锁状态。另外，由于篇幅所限，代码中省略了对异常的处理（即省略了try/catch块），但这并不影响整个程序的整体结构。

下面是整个通信流程的示意图：



3. 使用RMI实现Applet与Servlet的通信

RMI（Remote Method Invocation）是一种Java虚拟机之间对象（Object）互相调用对方函数，启动对方进程的一种机制，用这种机制，某一台Java虚拟机上的对象在调用另外一台Java虚拟机上的函数时，使用的程序语法规则和在本台Java虚拟机上对象间的函数调用的语法规则一样。正是这种机制给分布计算的系统设计，编程都带来了极大的方便。只要按照RMI规程设计程序，你可以不必再过问在RMI之下的网络细节了，如TCP/IP，Socket等等，更不必担心其下面的软硬件环境。任意两台Java虚拟机之间的通信完全由Java虚拟机自己的RMI来负这两台Java虚拟机责。对程序员来讲，之间完全是透明的，远在天边的Java虚拟机上的对象，使用起来就象近在眼前一样。

既然Applet 和Servlet都是运行在JAVA虚拟机上的对象，因此可以利用RMI的强大功能进行远程通信。

首先要定义一个远程接口，这里取名为DaytimeServer，其定义如下：

```
public interface DaytimeServer extends Remote {
  public Date getDate() throws RemoteException;
```

然后要定义实现该远程接口的类:

```
public class RMIServlet extends HttpServlet implements DaytimeServer {protected Registry registry;
public Date getDate(){
  return new Date();
}
   public void init(ServletConfig config) throws Servlet Exception {
      super.init(config);
      UnicastRemoteObject.exportObject(this);  //输出该远程对象以使它可以接收远程调用
    bind(); // 将本servlet绑定到注册表中。
   }
   public void destroy() {   //终止Servlet的RMI操作和监听Socket连接的线程
   super.destroy() ;
   if (registry != null) registry.unbind(this.getClass().getName());
  }
   protected void bind() {
try {
  //查找在端口Registry.REGISTRY〖CD#*2〗PORT上已经运行的注册表
    registry = LocateRegistry.getRegistry(Registry.REGISTRY_PORT);
      registry.list();  // 验证该注册表的有效性
   }
   catch (Exception e) {
      registry = null; //没有得到一个有效的注册，则将registry置为null.
   }
  if (registry == null)// 如果registry为null，则创建它。
      registry = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
   registry.rebind(this.getClass().getName(), this); //注册servlet实例到注册表中。
  }
}
```

程序在init()方法中首先调用UnicastRemoteObject类的方法exportObject()输出该远程对象（因为RMIServlet类实现了Remote接口，所以它是远程对象）以使它可以接收远程调用。然后调用函数bind()将本servlet绑定到注册表中。在函数bind()中，首先利用LocateRegistry类的getRegistry()方法查找在端口Registry.REGISTRY_PORT（缺省的注册表端口号，默认值为1099）上已经运行的注册表，并调用list()方法验证该注册表的有效性，如果没有得到一个有效的注册表，则调用createRegistry()方法创建一个新的，并且利用rebind()方法将servlet的 实例注册到注册表中。至此，RMIServlet便可以在端口Registry. REGISTRY_PORT上监听远程客户的请求。另外，RMIServlet类还实现了其远程接口DaytimeServer中的getDate方法，返回结果是当前日期和时间。

Applet中的代码如下:

```
Registry registry =LocateRegistry.getRegistry(getCodeBase().getHost(),Registry.REGISTRY_PORT);
DaytimeServer daytime =
  (DaytimeServer)registry.lookup("RMIServlet");
String mytime=daytime.getDate().toString();
```

程序首先利用LocateRegistry类的getRegistry()方法获得运行在远端主机相应端口上的注册表，然后利用lookup()返回与注册表中相应名字（本例中为RMIServlet）相联系的远程对象。最后调用远程对象中的相应方法获得服务器的系统时间。

本例中主要使用的是接口java.rmi.registry.*中的类和方法，它们的文档在JAVA API中有具体的说明。

结束语

Java Applet与Java Servlet之间的三种通信方式各有特点，一般的，URL/URLConnection类方法中所采用的通信协议都是应用层的HTTP协议，这样可以避免烦琐的底层编程，是一种最简便的方式，但是完全依赖于HTTP协议，缺乏灵活性。Socket和RMI虽然灵活性较大，但编程相对比较复杂。在实际应用中，应该根据不同的情况有选择的使用。■

参考资料

1 http://www.servlets.com

2 http://www.sun.com/product/servlet

3 <<计算机世界>> 1999第35期