

Ejecución distribuida de una aplicación

PRÁCTICA 5

Andres Rodarte López

ESCUELA SUPERIOR DE CÓMPUTO | INSTITUTO POLITÉCNICO NACIONAL

Introducción

Ejecutar un programa en Java de forma distribuida con la ayuda del código proporcionado en drun y servidor_drun

Código

Código para ejecutar un comando en segundo plano

```
# Execute a Java instance in background and save aoutput add a file difine
function Add-BGJava2File($class, $fileName, $extraArg) {
    Start-Process javaw -ArgumentList $class, $extraArg -
RedirectStandardOutput ('.\' + $fileName) -
RedirectStandardError '.\console.err'
}
```

Código servidor_drun

```
/*
    servidor_drun.java
    Ejecucion distribuida
    Carlos Pineda G. 2020
*/

import java.net.Socket;
import java.net.ServerSocket;
import java.lang.Thread;
import java.nio.ByteBuffer;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;

class servidor_drun
{
    static class Worker extends Thread
    {
        Socket conexion;

        Worker(Socket conexion)
        {
```

```

        this.conexion = conexion;
    }

    static void escribe_archivo(String nombre_archivo,byte[] buffer) throws
Exception
    {
        FileOutputStream f = new FileOutputStream(nombre_archivo);
        try
        {
            f.write(buffer);
        }
        finally
        {
            f.close();
        }
    }

    static void ejecuta_jar(String nombre_archivo,int nodo) throws Exception
    {
        String[] cmd = new String[4];
        cmd[0] = "java";
        cmd[1] = "-jar";
        cmd[2] = nombre_archivo;
        cmd[3] = Integer.toString(nodo);

        // inicia la ejecucion del subprocesso

        Process p = Runtime.getRuntime().exec(cmd);

        // stdInput y stdError permiten obtener la salida estandar y la salida
de errores del subprocesso

        BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.g
etInputStream()));
        BufferedReader stdError = new BufferedReader(new InputStreamReader(p.g
etErrorStream()));
        String s = null;

        // readLine bloquea mientras el subprocesso esta ejecutando
// cuando termina el subprocesso readLine regresa null

        try
        {
            while ((s = stdInput.readLine()) != null)
                System.out.println(s);
        }
    }

```

```

        while ((s = stdError.readLine()) != null)
            System.err.println(s);
    }
    catch (Exception e)
    {
        // si el thread es interrumpido entonces se destruye el subprocesso
        p.destroy();
    }
}

// lee del DataInputStream todos los bytes requeridos

static void read(DataInputStream f,byte[] b,int posicion,int longitud) throws Exception
{
    while (longitud > 0)
    {
        int n = f.read(b,posicion,longitud);
        posicion += n;
        longitud -= n;
    }
}

public void run()
{
    try
    {
        // abre los streams de entrada y salida

        DataInputStream entrada = new DataInputStream(conexion.getInputStream());
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());

        // recibe el numero de nodo
        int nodo = entrada.readInt();

        // recibe la longitud del nombre del archivo
        int longitud_nombre = entrada.readInt();

        // recibe el nombre del archivo
        byte[] buffer_1 = new byte[longitud_nombre];
        read(entrada,buffer_1,0,longitud_nombre);
        String nombre_archivo = new String(buffer_1,"UTF-8");
    }
}

```

```

        // recibe la longitud del archivo
        int longitud_archivo = entrada.readInt();

        // recibe el archivo
        byte[] buffer_2 = new byte[longitud_archivo];
        read(entrada,buffer_2,0,longitud_archivo);

        // cierra los streams de entrada y salida y la conexion

        entrada.close();
        salida.close();
        conexion.close();

        escribe_archivo(nombre_archivo,buffer_2);
        ejecuta_jar(nombre_archivo,nodo);
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
}
}
public static void main(String[] args) throws Exception
{
    if (args.length != 1)
    {
        System.out.println("Se debe pasar el numero de nodo como parametro");
        System.exit(1);
    }

    int nodo = Integer.valueOf(args[0]);
    ServerSocket servidor = new ServerSocket(20000 + nodo);

    for (;;)
    {
        Socket conexion = servidor.accept();
        Worker w = new Worker(conexion);
        w.start();
    }
}
}

```

Código de drun

```
/*
    drun.java
    Ejecucion distribuida
    Carlos Pineda G. 2020
*/

import java.net.Socket;
import java.lang.Thread;
import java.nio.ByteBuffer;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.BufferedReader;
import java.io.FileReader;

class drun
{
    static class Worker extends Thread
    {
        int nodo;
        String host;
        String nombre_programa;
        byte[] buffer;

        Worker(int nodo,String host,String nombre_programa,byte[] buffer)
        {
            this.nodo = nodo;
            this.host = host;
            this.nombre_programa = nombre_programa;
            this.buffer = buffer;
        }

        public void run()
        {
            try
            {
                // conecta con el servidor
                Socket conexion = new Socket(host,20000 + nodo);

                // abre los streams de entrada y salida

                DataInputStream entrada = new DataInputStream(conexion.getInputStrea
m());
```

```

        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());

        // envia el numero de nodo
        salida.writeInt(nodo);

        // envia la longitud del nombre del programa
        salida.writeInt(nombre_programa.length());

        // envia el nombre del programa
        salida.write(nombre_programa.getBytes());

        // envia la longitud del programa
        salida.writeInt(buffer.length);

        // envia el programa
        salida.write(buffer);
        salida.flush();

        // cierra los streams de entrada y salida y la conexion

        entrada.close();
        salida.close();
        conexion.close();
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
}

static byte[] lee_archivo(String archivo) throws Exception
{
    FileInputStream f = new FileInputStream(archivo);
    byte[] buffer;
    try
    {
        buffer = new byte[f.available()];
        f.read(buffer);
    }
    finally
    {
        f.close();
    }
}

```

```

        return buffer;
    }

    public static void main(String[] args) throws Exception
    {
        // verifica que se haya pasado como parametro el programa a ejecutar, en
        // otro caso despliega error y termina

        if (args.length != 1)
        {
            System.err.println("Se debe pasar como parametro el programa a ejecuta
r");
            System.exit(1);
        }

        // lee el programa del disco, si no se pudo leer despliega error y termi
na

        byte[] buffer = null;

        try
        {
            buffer = lee_archivo(args[0]);
        }
        catch (Exception e)
        {
            System.err.println("No se pudo leer el programa");
            System.exit(2);
        }

        // lee el archivo de nodos
        // el archivo "hosts" contiene las direcciones IP o nombres de dominio d
e los nodos desde el nodo 0 en adelante

        try
        {
            BufferedReader f = new BufferedReader(new FileReader("hosts"));

            try
            {
                int nodo = 0;
                String host;

                // para cada host crea un thread pasando como parametros: el numero
de nodo, la direccion del host, el nombre del programa y el programa

```



```

        while ((host = f.readLine()) != null)
        {
            Worker w = new Worker(nodo,host,args[0],buffer);
            w.start();
            nodo++;
        }
    }
    finally
    {
        f.close();
    }
}
catch (Exception e)
{
    System.err.println("No se pudo leer el archivo de hosts");
    System.exit(3);
}
}
}

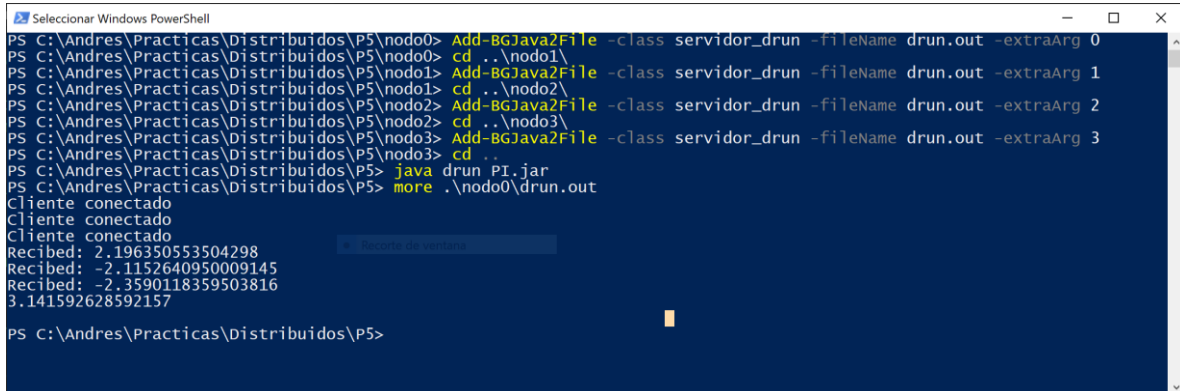
```

Resumen

La práctica consta, fundamentalmente, en dos programas de java. El primero es servidor_drun, el cual recibe un numero entero, el cual indica que nodo es y dará el valor del puerto por el que será accedido. La conexión es manejada por un hilo. El hilo se ejecuta esperando recibir un programa de java empaquetado como en formato jar. Una vez que lo recibe, lo guarda y lo ejecuta.

Del lado del cliente, drun. Drun recibe como parámetro el nombre del programa en /jar a ejecutar de forma distribuida. Después lee de un archivo las direcciones de donde se encuentran los servidores, y ejecuta un thread para cada una de las conexiones. Cada conexión manda el archivo a los servidores.

Capturas



```
PS C:\Andres\Practicas\Distribuidos\P5\nodo0> Add-BGJava2File -class servidor_drun -fileName drun.out -extraArg 0
PS C:\Andres\Practicas\Distribuidos\P5\nodo0> cd ../nodo1\
PS C:\Andres\Practicas\Distribuidos\P5\nodo1> Add-BGJava2File -class servidor_drun -fileName drun.out -extraArg 1
PS C:\Andres\Practicas\Distribuidos\P5\nodo1> cd ../nodo2\
PS C:\Andres\Practicas\Distribuidos\P5\nodo2> Add-BGJava2File -class servidor_drun -fileName drun.out -extraArg 2
PS C:\Andres\Practicas\Distribuidos\P5\nodo2> cd ../nodo3\
PS C:\Andres\Practicas\Distribuidos\P5\nodo3> Add-BGJava2File -class servidor_drun -fileName drun.out -extraArg 3
PS C:\Andres\Practicas\Distribuidos\P5\nodo3> cd ..
PS C:\Andres\Practicas\Distribuidos\P5> java drun PI.jar
PS C:\Andres\Practicas\Distribuidos\P5> more ../nodo0\drun.out
Cliente conectado
Cliente conectado
Cliente conectado
Recibed: 2.196350553504298
Recibed: -2.1152640950009145
Recibed: -2.3590118359503816
3.141592628592157
PS C:\Andres\Practicas\Distribuidos\P5>
```

La imagen muestra los comandos para correr los programas en segundo plano y guardar la información de salida en el archivo drun.out. Después salimos al directorio superior y ejecutamos drun con el programa PI.jar. Después leemos la salida del nodo principal, el cual es el nodo 0.