CS 580 Client-Server Programming
Fall Semester, 2000
Doc 12 Sample Concurrent Server
Contents

References

Previous Client-Server course notes

Sample Concurrent Server

Goals

- Build a reusable framework
- Use thread pool that grows & shrinks
- Demonstrate testing

Framework Problems

- Thread pool growth policy
- Does not shrink thread pool
- First iteration, so a bit rough
- Making a framework created some complexity

Framework Components

- ConcurrentServer
- ServerThread
- BoundedQueue
- ServerEngine

BoundedQueue

Manages list of client requests waiting to be processed

ServerEngine

Interface for classes that perform the work of the server

ServerThread

Removes requests from BoundedQueue and hands the request to a ServerEngine

ConcurrentServer

Configurable

Manages the thread pool

Accepts client requests and adds them to BoundedQueue

Basic Action

ConcurrentServer reads settings from:

Configuration file and
Command line

Settings indicate which concrete ServerEngine to use

ConcurrentServer

Creates prototype ServerEngine

Creates ServerThreads with different copies of ServerEngine

Creates ServerSocket

Basic ConcurrentServer run loop

```
while (true)
   accept new client connection
   if queue is full close connection, repeat loop
   if need more threads create one
   add connection to queue
```

Some Java Magic - Class

Instances of the class **Class** represent classes and interfaces in a running Java application

```
Class sampleClass = Class.forName( "java.util.Vector");
Vector instance = (Vector) sampleClass.newInstance();
instance.addElement( "hi");
System.out.println( instance.elementAt( 0));
```

Class.forName
- Converts string class name to a class object
- String name must be full name of the class

This is an example of Java's Reflection

Reflection

Reflection refers to the ability of a program to inspect and manipulate itself. Java added reflection in JDK 1.1. The class java.lang.Class and the package java.lang.reflect (classes Array, Constructor, Field, Method, Modifier) implement Java's reflection. The methods of java.lang.Class are used to get information about a class. The methods of the classes in java.lang.reflect provide further information about a class and allow methods to be invoked.

java.lang.Class Methods

forName(String)
getClasses()
getClassLoader()
getComponentType()
getConstructor(Class[])
getConstructors()
getDeclaredClasses()
getDeclaredConstructor(Class[])
getDeclaredConstructors()
getDeclaredField(String)
getDeclaredFields()
getDeclaredMethod(String, Class[])
getDeclaredMethods()
getDeclaringClass()
getField(String)
getFields()
getInterfaces()
getMethod(String, Class[])
getMethods()
getModifiers()
getName()
getResource(String)
getResourceAsStream(String)
getSigners()
getSuperclass()
isArray()
isAssignableFrom(Class)
isInstance(Object)
isInterface()
isPrimitive()
newInstance()
toString()

Sample Class

The following class will be used in examples in this document.

```
public class Sample {
   static private int privateInt = 1;
   protected float protectedField = 1.2F;
   int[] data = { 4, 3, 2, 1 };
   public String message = "Hi mom";

   public Sample() {
      System.out.println( "Default Constructor" );
   }

   public Sample( String newMessage ) {
      System.out.println( "One arg Constructor"  );
      message = newMessage;
   }

   public final void setData( int[] newData ) {
      System.out.println( "Set data" );
      data = newData;
   }

   public void setFields( int anInt, float aFloat ) {
      System.out.println( "Set fields" );
      privateInt = anInt;
      protectedField = aFloat;
   }

   public String toString() {
      return "Sample(" + privateInt + ", " + protectedField + ", " + message +")";
   }
}
```

Obtaining Class Objects

There are a number of ways to get a Class object for a class: Class.forName(), class literals (.class - class is a keyword in java), and Object.getClass(). With forName() the full class name **must** be used. Import statements do not affect forName(). With class literals, one can either use the full class name or use import statements. The class literal is faster than using forName(). However, forName() uses a string, which can be supplied dynamically.

```
public class Test  {
   public static void main( String args[] )
         throws ClassNotFoundException {

      Class aVectorClass = java.util.Vector.class;
      Class bVectorClass = Class.forName( "java.util.Vector" );

      Vector aVector = new Vector();
      Class cVectorClass = aVector.getClass();
      System.out.println( cVectorClass.getName() );

      Class sampleClass = Sample.class;
   }
```

Output

```
java.util.Vector
```

Information about a Class

The following example shows how to get information about the fields of a class.

```
import java.lang.reflect.*;

public class Test  {
   public static void main( String args[] )
         throws ClassNotFoundException {
      Class sampleClass = Sample.class;

      Field[] sampleFields = sampleClass.getDeclaredFields();

      System.out.println( "Field name, type, modifiers" );
      for ( int k = 0; k < sampleFields.length; k++ ) {
         String name = sampleFields[k].getName();
         Class type = sampleFields[k].getType();
         int modifiers = sampleFields[k].getModifiers();
         if ( type.isArray() )
            System.out.println( name + ", Array of: " +
               type.getComponentType().getName() + ", " +
               Modifier.toString( modifiers) );
         else
            System.out.println( name + ", " + type.getName() + ", " +
               Modifier.toString( modifiers) );
         }
   }
}
```

Output

```
Field name, type, modifiers
privateInt, int, private static
protectedField, float, protected
data, Array of: int,
message, java.lang.String, public
```

Creating Objects

The following example shows how to use reflection to create new objects. The first example, using newInstance(), calls the default constructor, which must be public. The second example, uses a Constructor object.

```
import java.lang.reflect.*;

public class Test  {
   public static void main( String args[] )
         throws ClassNotFoundException, IllegalAccessException,
                InstantiationException,NoSuchMethodException,
                InvocationTargetException {
      Class sampleClass = Sample.class;

      Sample aSample = (Sample) sampleClass.newInstance();
      System.out.println( aSample );

      Class aClass = Class.forName( "Sample" );

      Class[] argumentTypes = { java.lang.String.class };
      Constructor oneArgument =
         aClass.getConstructor( argumentTypes );

      Object[] arguments = { "Test" };
      Object newObject = oneArgument.newInstance( arguments );
      System.out.println( newObject.getClass() );
      System.out.println( newObject );
   }
}
```

Output

```
Default Constructor
Sample(1, 1.2, Hi mom)
One arg Constructor
class Sample
Sample(1, 1.2, Test)
```

Calling Methods

This example shows how to call methods using reflection. If there are no arguments, set argumentTypes to new Class[0].

```
import java.lang.reflect.*;

public class Test  {
   public static void main( String args[] )
         throws ClassNotFoundException, IllegalAccessException,
                InstantiationException,NoSuchMethodException,
                InvocationTargetException  {

      Class sampleClass = Sample.class;

      Class[] argumentTypes = { Integer.TYPE, String.class };
      Method aMethod =
         sampleClass.getMethod( "setFields", argumentTypes );

      Object aSample =  sampleClass.newInstance();
      System.out.println( aSample );

      Object[] arguments = { new Integer(23), "Bye" };
      aMethod.invoke( aSample, arguments );
      System.out.println( aSample );
   }
}
```

Output

```
Default Constructor
Sample(1, 1.2, Hi mom)
Set fields
Sample(23, 1.2, Bye)
```

Array Parameters

This example shows how to handle array parameters.

```
import java.lang.reflect.*;

public class Test  {
   public static void main( String args[] )
         throws ClassNotFoundException,
                IllegalAccessException,
                InstantiationException,
                NoSuchMethodException,
                InvocationTargetException
                {
      Class sampleClass = Sample.class;

      Class[] argumentTypes = { int[].class };
      Method aMethod =
         sampleClass.getMethod( "setData", argumentTypes );

      Object aSample =  sampleClass.newInstance();
      System.out.println( aSample );

      int[] someData = { 1, 2, 3, 4 };
      Object[] arguments = { someData };
      aMethod.invoke( aSample, arguments );
      System.out.println( aSample );
   }
}
```

ConcurrentServer Source Code
ServerEngine

```
package sdsu.net.server;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.net.InetAddress;
import sdsu.util.ProgramProperties;

interface ServerEngine
   {
   public ServerEngine newInstance(ProgramProperties settings);
   public ServerEngine cleanInstance( );
   public void processRequest(InputStream in, OutputStream out,
      InetAddress clientAddress) throws IOException;
   }
```

newInstance(…)

return a new instance
settings are used to pass data to the serverEngine

cleanInstance( )

return new ServerEngine or return current object with original state

processRequest(…)

perform the client request

BoundedQueue

```
package sdsu.net.server;
import java.util.*;

public class BoundedQueue
   {
   List elements;
   int maxSize;

   public BoundedQueue(int maxQueueSize)
      {
      elements = new ArrayList(maxQueueSize);
      maxSize = maxQueueSize;
      }

   public synchronized Object remove() throws InterruptedException
      {
      while ( isEmpty() )
         wait();
      return elements.remove( 0 );
      }

   public synchronized void add( Object item)
      {
      elements.add( item );
      notifyAll();
      }
```

BoundedQueue - Continued

```
public boolean isEmpty()
   {
   return elements.isEmpty();
   }
public boolean isFull()
   {
   return elements.size() == maxSize;
   }

public int capacity()
   {
   return maxSize;
   }

public int size()
   {
   return elements.size();
   }
}
```

Sample BoundedQueue Test

```
package sdsu.net.server;
import junit.framework.TestCase;

public class TestBoundedQueue extends TestCase
   {
   public TestBoundedQueue(String name)
      {
      super(name);
      }

   public void testAddRemove() throws InterruptedException
      {
      BoundedQueue buffer = new BoundedQueue( 4);
      assert( "empty", buffer.isEmpty() );
      assert( "not full", !buffer.isFull() );
      String[] input = { "a", "b", "c", "d" };
      for (int k = 0; k < input.length; k++ )
         buffer.add( input[k]);

      assert( "full", buffer.isFull() );

      int k = 0;
      while (! buffer.isEmpty() )
         {
         assert( "remove " + k, buffer.remove().equals( input[k]));
         k++;
         }
      assert( "empty again", buffer.isEmpty() );
      }
   }
```

ServerThread

Note: Used inner class for tests

```
package sdsu.net.server;
import java.util.*;
import java.net.Socket;
import sdsu.logging.Logger;

//For tests
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.net.InetAddress;
import sdsu.io.SimpleFile;
import sdsu.logging.*;
import sdsu.util.ProgramProperties;

public class ServerThread extends Thread
  {
  private Set inactiveThreads;
  private Set activeThreads;
  private BoundedQueue requests;
  private ServerEngine factory;

  public ServerThread(Set inactiveThreadList, Set activeThreadList,
    BoundedQueue input,  ServerEngine clientProcessor)
    {
    inactiveThreads = inactiveThreadList;
    activeThreads = activeThreadList;
    requests = input;
    factory = clientProcessor;
    inactivate();
    }
```

ServerThread - Continued

```
public void run()
  {
  while (!isInterrupted())
    {
    try
      {
      processRequest();
      }
    catch (InterruptedException stopThread)
      {
      Logger.log( "InterruptedException, thread killed " +
        stopThread);
      break; //end thread
      }
    catch (Exception problem)
      {
      Logger.error( "Error in processing client request" );
      Logger.error( problem );
      }
    }
  unlist();
  }
```

ServerThread - Continued

```
public void processRequest() throws Exception
  {
  Socket client = null;
  try
    {
    client = (Socket) requests.remove();
    activate();
    ServerEngine processor = factory.cleanInstance();
    processor.processRequest( client.getInputStream(),
      client.getOutputStream(), client.getInetAddress());
    }
  finally
    {
    if (client != null)
      client.close();
    inactivate();
    }
  }
```

ServerThread - Continued

```
private void activate()
   {
   inactiveThreads.remove(this);
   activeThreads.add( this);
   }

private void inactivate()
   {
   activeThreads.remove(this);
   inactiveThreads.add( this);
   }

private void unlist()
   {
   inactiveThreads.remove( this);
   activeThreads.remove(this);
   }
```

ServerThread - Tests
Normal Case

```
public static class TestServerThread extends junit.framework.TestCase
   {
   public TestServerThread(String name)
      {
      super( name);
      }

   public void testThreadNormalCase() throws Exception
      {
      int testSize = 2;
      Set active = new HashSet();
      Set inactive = new HashSet();
      CountAction  recorder = new CountAction();
      BoundedQueue buffer = new BoundedQueue( testSize);

      ServerThread a = new ServerThread(inactive, active, buffer, recorder);
      for (int k = 0; k < testSize; k++)
         buffer.add( new Socket("www.sdsu.edu", 80) );
      a.setPriority( 8);
      a.start();
      assert( "inactive 1", inactive.contains( a));
      assert( "count" , recorder.actionsDone == testSize);
      }
```

ServerThread Tests - Interrupted Case

```
  public void testThreadInteruptCase() throws Exception
    {
    String logName = "Log";
    FileLogger.register( logName );
    SimpleFile logFile = new SimpleFile( logName + ".log");
    try
      {
      Set active = new HashSet();
      Set inactive = new HashSet();
      CountAction  recorder = new CountAction();
      BoundedQueue buffer = new BoundedQueue( 4);

      ServerThread a = new ServerThread(inactive, active, buffer, recorder);
      a.setPriority( 8);
      a.start();
      a.interrupt();
      Thread.currentThread().yield( ); //let thread react to interrupt
      assert( "inactive 1", !inactive.contains( a));
      assert( "inactive 2", !active.contains( a));
      assert( "file exists" , logFile.exists());
      String contents = logFile.getContents();
      assert( "contents",
        contents.indexOf( "InterruptedException, thread killed " ) > -1 );
      }
    finally
      {
      logFile.delete();
      }
    }
  }
}
```

ConcurrentServer

```
package sdsu.net.server;

import sdsu.util.ProgramProperties;
import java.util.*;
import sdsu.logging.Logger;
import java.net.ServerSocket;
import java.net.Socket;
import sdsu.logging.FileLogger;

public class ConcurrentServer extends Thread
   {
   static final String PORT_KEY = "p";
   static final String MAX_THREADS_KEY = "maxThreads";
   static final String INITIAL_THREADS_KEY = "initialThreads";
   static final String MAX_WAIT_SIZE_KEY = "maxWaitSize";
   static final String SERVER_TYPE_KEY = "s";
   static final String LOG_FILE_KEY = "log";
   static final String CONFIGURATION_FILE = "server.labeledData'";
   static final int CLIENT_THREAD_PRIORITY = 7;
   static final int MAIN_THREAD_PRIORITY = 8;

   ServerSocket clientInput;
   int maxThreads;
   BoundedQueue waitingClients;
   ServerEngine processorClone;
   Set activeThreads = Collections.synchronizedSet( new HashSet());
   Set inactiveThreads = Collections.synchronizedSet( new HashSet());
   ProgramProperties settings;
```

ConcurrentServer Main

```
public static void main( String[] args)
  {
  try
    {
    ProgramProperties serverSettings =
      new ProgramProperties( args, CONFIGURATION_FILE );

    if ( !serverSettings.containsKey( PORT_KEY) &&
      !serverSettings.containsKey( SERVER_TYPE_KEY))
      {
      System.out.println( "comand line format: java " +
        "sdsu.net.server.ConcurrentServer " +
        "-p portNumber -s serverClassFile");
      System.exit( 0);
      }
    ConcurrentServer server =
      new ConcurrentServer( serverSettings);
    server.run();
    }
  catch (Exception error)
    {
    System.err.println( "Exception on start up of server");
    error.printStackTrace( System.err);
    System.exit( 0);
    }
  }
```

ConcurrentServer Constructor & Run

```
   public ConcurrentServer( ProgramProperties serverSettings) throws Exception
      {
      settings = serverSettings;
      maxThreads = serverSettings.getInt(MAX_THREADS_KEY , 4);
      waitingClients =
         new BoundedQueue( serverSettings.getInt( MAX_WAIT_SIZE_KEY, 5));
      String ServerEngineClass = serverSettings.getString(
SERVER_TYPE_KEY);

      processorClone =
         (ServerEngine) Class.forName( ServerEngineClass ).newInstance();

      int port = serverSettings.getInt( PORT_KEY);
      clientInput = new ServerSocket( port );
      setPriority( MAIN_THREAD_PRIORITY );
      String logFileName = serverSettings.getString( SERVER_TYPE_KEY);
      FileLogger.register( logFileName );

      //do last as uses state of object
      addThreads(  serverSettings.getInt( INITIAL_THREADS_KEY, 1));
      }

   public void run()
      {
      while (true)
         {
         processNewConnection();
         }
      }
```

ConcurrentServer Main Loop

```
private void processNewConnection()
  {
  try
    {
    Socket newClient = clientInput.accept();
    if (null == newClient)
      return;

    if (waitingClients.isFull() )
      {
      newClient.close();
      Logger.log( "Wait queue full, client connection refused" );
      return;
      }

    if (!waitingClients.isEmpty() )
      {
      addThreads( 1);
      }
    waitingClients.add( newClient);
    }
  catch (Exception trouble)
    {
    Logger.error("Exception on accept");
    Logger.error(trouble);
    }
  }
```

ConcurrentServer Helper Functions

```
int threadPoolSize( )
   {
   return activeThreads.size() + inactiveThreads.size();
   }

private void addThreads( int count)
   {
   int numberToAdd =
      Math.min( count, maxThreads - threadPoolSize());

   for (int k = 0; k < numberToAdd; k++)
      {
      ServerEngine clientProcessor =
         processorClone.newInstance( settings);
      ServerThread addMe =  new ServerThread(inactiveThreads,
            activeThreads, waitingClients, clientProcessor);
      addMe.setPriority(CLIENT_THREAD_PRIORITY);
      addMe.start();
      }
   }
}
```

Sample Server Test

```
package sdsu.net.server;

import sdsu.util.*;
import sdsu.io.*;
import sdsu.logging.*;
import java.io.*;
import java.net.Socket;

public  class TestConcurrentServer extends junit.framework.TestCase
   {
   static int port = 3000;

   SimpleFile configFile;
   public TestConcurrentServer(String name)
      {
      super( name);
      }
   public void setUp() throws IOException
      {
      LabeledData configData = new LabeledData();
      configData.put( ConcurrentServer.PORT_KEY, "33333");
      configData.put( ConcurrentServer.MAX_THREADS_KEY, "2");
      configData.put( ConcurrentServer.INITIAL_THREADS_KEY, "1");
      configData.put( ConcurrentServer.MAX_WAIT_SIZE_KEY, "3");
      configFile = new SimpleFile("server.LabeledData");
      configFile.setContents( configData.toString());

      CountAction.reset();
      port ++;
      }

   public void tearDown() { configFile.delete(); }
```

Sample Server Test - The Test

```
public void testLoadedServerCase() throws Exception
   {
   Logger.log( "Start load test");
   String[] args = { "-p=" + port, "-s=sdsu.net.server.SleepAction"};
   ProgramProperties serverSettings =
         new ProgramProperties( args, "server.LabeledData" );

   ConcurrentServer test = new ConcurrentServer( serverSettings );
   test.setPriority( 7);
   test.start();
   assert( "active ", 0 == test.activeThreads.size()  );
   assert( " inactive", 1 == test.inactiveThreads.size()  );
   assert( "max threads", 2 == test.maxThreads );
   assert( "socket", null != test.clientInput );
   assert( "wait capacity", 3 == test.waitingClients.capacity() );
   assert( "wait size", 0 == test.waitingClients.size() );
   assert( "Server engine", null != test.processorClone  &&
      test.processorClone instanceof SleepAction);
   for (int k = 0; k < 6 ;k++)
      {
      ClientThread a = new ClientThread( port);
      a.setPriority( 6);
      a.start();
      Thread.currentThread().yield();
      }

   assert( "after wait size", 3 == test.waitingClients.size() );
   assert( "after active ", 2 == test.activeThreads.size()  );
   assert( "after inactive", 0 == test.inactiveThreads.size()  );
   Logger.log( "Load test done");
   }
```

SleepAction - Test Helper Class

```
package sdsu.net.server;

import java.io.*;
import java.net.InetAddress;
import sdsu.io.SimpleFile;
import sdsu.logging.*;
import sdsu.util.ProgramProperties;

public class SleepAction implements ServerEngine
   {
   public ServerEngine newInstance(ProgramProperties settings)
      {
      return this;
      }
   public ServerEngine cleanInstance( )
      {
      return this;
      }
   public void processRequest(InputStream in, OutputStream out,
      InetAddress clientAddress) throws IOException
      {
      try
         {
         //Force server to accumulate requests
         Thread.currentThread().sleep( 10000);
         }
      catch (Exception e)
         {
         }
      }
   }
```

ClientThread -Test Helper Class

```
package sdsu.net.server;

import java.io.*;
import java.net.*;
import sdsu.io.SimpleFile;
import sdsu.logging.*;
import sdsu.util.ProgramProperties;

public class ClientThread extends Thread
   {
   int port;
   String result;

   public ClientThread(int portNumber)
      {
      port = portNumber;
      }

   public String answer()
      {
      return result;
      }
```

ClientThread - Continued

```
public void run()
   {
   try
      {
      Socket toServer = new Socket( "127.0.0.1", port);
      if (null == toServer)
         Logger.error( "Null socket");
      else
         {
         InputStream in = toServer.getInputStream();
         BufferedReader easyIn =
            new BufferedReader(new InputStreamReader( in ));
         result = easyIn.readLine().trim();
         }
      }
   catch (Exception p)
      {
      Logger.error( "client error");
      Logger.error( p);
      }
   }
}
```