Dhruvit Patel (ID:10404032)

Assignment -4 Websockets

Abstract

In this report, I explained how I completed the code to obtain a working condition for Web-sockets in the DHT Environment. I included code snippets of each and every step.

In the second part of this report, I illustrated the testing of this code with the local and remote environment.

I used **'Simple Video Recorder'** in Linux platform to provide a short video for getting better understanding of working of my implementation.

Explanation of Code:

- In **ControllerClient.java,** we use following code to connect the **Server**

```
public void connect(URI uri) throws DeploymentException, IOException {
    System.out.println("ControllerClient: connect(): enter");
    try {
        shell.msg("Requesting control of node at " + uri.toString() + "...");

        // TODO make the connection request. Add by Dhruvit.

        client.asyncConnectToServer(this, cec, uri);

        System.out.println("\n ControllerClient: connect()");

        while (true) {
            try {
                // Synchronize with receipt of an ack from the remote node.
                boolean connected = messageLatch.await(100, TimeUnit.SECONDS);
                // TODO If we are connected, a new top level shell has been pushed, execute its CLI.
                // Be sure to return when done, to exit the loop.

                if(connected){
                    ProxyShell proxyShell = new ProxyShell(shell, session.getBasicRemote());
                    shellManager.addShell( proxyShell ); /* push the proxy shell in stack */
                    shellManager.getCurrentShell().cli();
                    return;
                }

            } catch (InterruptedException e) {
                // Keep on waiting for the specified time interval
                shell.err(e);
            }
        }
    } catch (IOException e) {
        shell.err(e);
    }
    System.out.println("ControllerClient: connect(): exit");
}
```

- Following is the configuration of client end point

```
// TODO configure the client to use proper encoder for messages sent to server
private final ClientEndpointConfig cec = ClientEndpointConfig.Builder.create().
        encoders(Arrays.asList(CommandLineEncoder.class)).
        decoders(Arrays.asList(CommandLineDecoder.class))
        .build();

private final ClientManager client = ClientManager.createClient();
```

- Following is the end point configuration at server side

```
@ServerEndpoint(
        value="/control/{name}",
        encoders = {CommandLineEncoder.class},
        decoders = {CommandLineDecoder.class})
```

- This is onOpen() implementation in ControllerClient to handle initial communication between client and server.

```
@Override
public void onOpen(Session session, EndpointConfig config) {
    // TODO session created, add a message handler for receiving communication from server.
    // We should also cache the session for use by some of the other operations.
    // Add by Dhruvit.
    System.out.println("ControllerClient: onOpen(): enter");

    session.addMessageHandler( new MessageHandler.Whole<String>() {
        @Override
        public void onMessage(String arg0) {
            try {
                session.getBasicRemote().sendText(arg0);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });

    this.session = session;
    System.out.println("ControllerClient: onOpen(): exit");
}
```

- onMessage() implementation in ControllerClient to when connection is established.

```java
@Override
public void onMessage(String message) {
    if (initializing) {
        if (SessionManager.ACK.equals(message)) {
            /*
             * TODO server has accepted our remote control request, push a proxy shell on the shell stack
             * and flag that initialization has finished (allowing the UI thread to continue).
             * Make sure to replace the cached shell in this callback with the new proxy shell!
             *
             * If the server rejects our request, they will just close the channel.
             */
            this.session.addMessageHandler( new MessageHandler.Whole<String>() {
                @Override
                public void onMessage(String message) {
                    // TODO Auto-generated method stub
                    try {
                        session.getBasicRemote().sendText(message);
                    } catch (IOException e) {
                        // TODO: handle exception
                        e.printStackTrace();
                    }
                }
             });
            ProxyShell proxyShell = new ProxyShell(shellManager.getCurrentShell(), session.getBasicRemote());
            shellManager.removeShell();
            shellManager.addShell(proxyShell); // replace has done partially
        } else {
            throw new IllegalStateException("Unexpected response to remote control request: " + message);
        }
    } else {
        // TODO provide the message to the shell. Print the message in stdout. Add by dhruvit.
        try {
            shellManager.getCurrentShell().msg(message);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

- shutdown() implementation in ControllerClient when error or closure initiated in the connection between client and server.

```
    protected void shutdown() throws IOException {
        /*
         * TODO Shutdown initiated by error or closure of the connection.  Three cases:
         * 1. We are still initializing when this happens (need to unblock the client thread).
         * 2. We are running an on-going remote control session (need to remove the proxy shell).
         * 3. The remote control session has terminated (which caused the channel to be closed).
         */
        if(initializing){
            messageLatch.countDown();
        }
        else if(this.session.isOpen()){
            this.shellManager.removeShell();
        }
        else if(!this.session.isOpen()){ //remote session is running or not
            SessionManager sessionManager = SessionManager.getSessionManager();
            sessionManager.closeCurrentSession();
        }
    }
```

```
@OnMessage
public void onMessage(String[] commandLine) {
    if (initializing) {
        throw new IllegalStateException("Communication from client before ack of remote control request: " + commandLine[0]);
    } else if (commandLine.length > 0 && IShell.QUIT.equals(commandLine[0])) {

        /*
         * TODO Stop the current toplevel (local) shell.  It is sufficient to close the session,
         * which will trigger a callback on onClose() on both sides of the connection.
         */

        // Add by Dhruvit.
        shellManager.getCurrentShell().getLocal().stop();
        sessionManager.closeCurrentSession();

    } else {
        /*
         * TODO add the commandLine to the input of the current shell
         */

        //Add by Dhruvit.
        shellManager.getCurrentShell().addCommandLine(commandLine);
    }
}
```

- OnMessage() implementation on ControllerServer.java side.

- In ShellBase.java, we implement accept() and reject() function to accept and reject the current session.

```java
/**
 * TODO Accept the pending session (see SessionManager) and
 * start running the CLI for the new shell that will have been
 * pushed on the shell stack.
 */
protected void accept(String[] inputs) throws IOException {
    if (inputs.length != 1) {
        msgln("Usage: accept");
    } else {
        // TODO. Add by Dhruvit.
        sessionManager.acceptSession(); // Accept the session
        shellManager.getCurrentShell().getLocal().cli(); // Execute the CLI for the new shell
    }
}

/**
 * TODO Reject and remove the pending session (see SessionManager).
 */
protected void reject(String[] inputs) throws IOException {
    if (inputs.length != 1) {
        msgln("Usage: reject");
    } else {
        // TODO. Add by Dhruvit.
        this.sessionManager.rejectSession(); // reject
        this.sessionManager.closeCurrentSession(); // remove current session
    }
}
```

- In SessionManager.java, we implement acceptSeesionn() function to accept the current session. Here SessionManager worked as singleton pattern.

```java
public void acceptSession() throws IOException {
    lock.lock();
    try {
        /*
         *  TODO We are accepting a remote control request.  Push a local shell with a proxy context
         *  on the shell stack and flag that initialization has completed.  Confirm acceptance of the
         *  remote control request by sending an ACK to the client.  The CLI of the newly installed shell
         *  will be executed by the underlying CLI as part of the "accept" command.
         */

        ShellManager.getShellManager().addShell( LocalShell.createRemotelyControlled
                (SHELL_MANAGER.getCurrentShell().getLocal(),
                ProxyContext.createProxyContext( this.getCurrentSession().getBasicRemote()));
        currentServer.endInitialization();
        currentServer.getSession().getBasicRemote().sendText(ACK);

    } finally {
        lock.unlock();
    }
}
```

- rejectSession() and closeCurrentSession() implementetion in SessionManager.java

```
public void rejectSession() {
    lock.lock();
    try {
        // TODO reject remote control request by closing the session (provide a reason!). Add by Dhruvit.
        SESSION_MANAGER.getCurrentSession().close( new CloseReason(CloseCodes.CANNOT_ACCEPT , "There is already a (pending?) session.") );
    }catch(IOException e){
        e.printStackTrace();
    }
    finally {
        lock.unlock();
    }
}

public void closeCurrentSession() {
    lock.lock();
    try {
        // TODO normal shutdown of remote control session (provide a reason!). Add by Dhruvit.
        SESSION_MANAGER.getCurrentSession().close( new CloseReason(CloseCodes.NORMAL_CLOSURE, "Normal closure of client session") );
    }catch(IOException e){
        e.printStackTrace();
    }
    finally {
        lock.unlock();
    }
}
```

Commands and Utilization

- connect localhost port: - We use this command to make client connection to server by specifying localhost and port number
- accept : - Server will be notified with client connection

By typing accept commands, server agree to accept the client connection

- reject : - If server already have client connection then, it rejects the client connection by typing “reject”
- quit : - If client want to quit from running session then he quit by typing quit commands
- add key value: We can bindings value to key on server side.
- Get key : - Get the all values bind with particular key.
- Bindings :- we can check server bindings.

Testing

- Local

  We use following commands to start the node with specific web server port

java -jar dht.jar --host 127.0.0.1 --http 8080 --ws 8081 --name dhruvit --id 17

java -jar dht.jar --host 127.0.0.1 --http 8082 --ws 8083 --name chirag --id 35

I created two nodes 17 and 35.

- Remote

  Following command are used for login into Amazon EC2 Console of two different instances.

1. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-52-27-30-64.us-west-2.compute.amazonaws.com

2. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-52-34-239-132.us-west-2.compute.amazonaws.com

Following command are used for uploading dht.jar file into Amazon EC2 Instances.

1. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-52-27-30-64.us-west-2.compute.amazonaws.com:/home/ubuntu

2. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-52-34-239-132.us-west-2.compute.amazonaws.com:/home/ubuntu

Following command are used for starting websocket on Amazon EC2 Instances.

1. java -jar dht.jar --host 172.31.28.76 --http 8080 --ws 8081 --name dhruvit --id 17

2. java -jar dht.jar --host 172.31.24.149 --http 8082 --ws 8083 --name chirag --id 35

Error:

My code work successfully in local machine but I don't get response in remote machine.