Dhruvit Patel (ID: 10404032)

Assignment -3 DHT-Server Side Events

Abstract

In this report, I explained how I completed the code to obtain a working condition for REST API and the DHT SSE. 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.

Code

- In **Application.java**, we use this code to register the **Server Side Events**

```
public class Application extends ResourceConfig {
    public Application() {

        super(NodeResource.class, SseFeature.class);
        packages("edu.stevens.cs549.dhts.resource");
        // TODO register SseFeature.  Add by Dhruvit

    }
}
```

- Here in **WebClient.java** we specify the **putRequest** and **deleteRequest** methods.

```
///////
private Response putRequest(URI uri, Entity<?> entity) {
    // TODO Complete. Add by Dhruvit
    try {
        Response cr = client.target(uri)
                .request(MediaType.APPLICATION_XML_TYPE)
                .header(Time.TIME_STAMP, Time.advanceTime())
                .put(entity);
        processResponseTimestamp(cr);
        return cr;
    } catch (Exception e) {
        error("Exception during PUT request: " + e);
        return null;
    }
}
```

```
/////////
private Response delRequest(URI deletePath) {
    // TODO Auto-generated method stub
    try {
        Response cr = client.target(deletePath)
                .request(MediaType.APPLICATION_XML_TYPE)
                .header(Time.TIME_STAMP, Time.advanceTime())
                .delete();
        processResponseTimestamp(cr);
        return cr;
    } catch (Exception e) {
        error("Exception during DELETE request: " + e);
        return null;
    }
}
```

- In **WebClient.java** we implemented the function to listen for any **bindings**

```
//////
public EventSource listenForBindings(NodeInfo node, int id, String skey) throws DHTBase.Failed {
    // TODO listen for SSE subscription requests on http://.../dht/listen?key=<key>
    // On the service side, don't expect LT request or response headers for this request.
    // Note: "id" is client's id, to enable us to stop event generation at the server.
    String uri = String.format(node.addr + UriApi.LISTEN, id, skey);
    WebTarget target = listenClient.target(uri);
    EventSource eventSource = new EventSource(target, false);
    return eventSource;

}
```

- In **WebClient.java** we define the **ListenOff** condition. The listeners will be released off their ListenOn condition. The syntax would be **ListenOff <key>**.

```
//////
public void listenOff(NodeInfo node, int id, String skey) throws DHTBase.Failed {
    // TODO listen for SSE subscription requests on http://.../dht/listen?key=<key>
    // On the service side, don't expect LT request or response headers for this request.

    String uri = String.format(node.addr + UriApi.LISTEN , id, skey);
    try {
        delRequest(new URI(uri));
    } catch (Exception e) {
        throw new DHTBase.Failed("listen Off error. message: " + e);
    }
}
```

- In **state.java** we add the method for **add and remove listeners.**

```
    /////
    public void addListener(int id, String key, EventOutput eventOutput) {
        if(outputs.containsKey(id)) {
            outputs.get(id).put(key, eventOutput);
        } else {
            Map<String,EventOutput> _map = new HashMap<String,EventOutput>();
            _map.put(key, eventOutput);
            outputs.put(id, _map);
        }

        if(listeners.containsKey(key)){
            listeners.get(key).add(eventOutput);
        } else {
            SseBroadcaster broadcaster = new SseBroadcaster();
            broadcaster.add(eventOutput);
            listeners.put(key, broadcaster);
        }
    }


    /////
    public void removeListener(int id, String key) {
        // TODO Close the event output stream.

        if(listeners.containsKey(key)){
            EventOutput eventOutput = outputs.get(id).get(key);
            try {
                eventOutput.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            listeners.get(key).remove(eventOutput);
        }
    }


/////
private void broadcastAddition(String key, String value) {
    // TODO broadcast an added binding (use IDHTNode.NEW_BINDING_EVENT for event name).

    if(listeners.containsKey(key)) {
        OutboundEvent.Builder eventBuilder = new OutboundEvent.Builder();
        OutboundEvent event = eventBuilder.name(IDHTNode.NEW_BINDING_EVENT).
                data(String.class, value).build();

        listeners.get(key).broadcast(event);
    }
}
```

- In **state.java** we define the remove call back method. If there are any binding on the key, this function removes any existing callbacks

```
/////
public void removeCallback(String key) {
    // TODO remove an existing callback (if any) for bindings on key.
    // Be sure to close the event stream from the broadcaster.
    EventSource eventSource = callbacks.get(key);
    if (eventSource != null) {
        eventSource.close();
        callbacks.remove(key);
    }
}
```

In **NodeService.java** we fill the missing operations which are:

```
@GET
@Path("listen")
@Produces(SseFeature.SERVER_SENT_EVENTS)
public EventOutput listenOn(@QueryParam("id") int id, @QueryParam("key") String key) {
        return new NodeService(headers, uriInfo).listen(id, key);
}
```

Defining the **listenOn** and **listenOff** functions in **DHT.java**. The **listenOn** registers a listener for new bindings for particular key, at the node.

```
/*
 * Client-side callbacks
 */
public void listenOn(String key, EventListener listener) throws DHTBase.Failed {
    /*
     * TODO: Register a listener for new bindings under key, at the node
     * where those bindings are stored.  The event source should
     * be registered in the state object, so the client can shut down the
     * event stream at this point.  The client will also need to contact the server
     * to request that event generation be stopped at the server (for this client
     * and key).  The client should send its own node id to identify itself
     * (for both the listen on and listen off requests).
     */
    // Add by Dhruvit

    int id = NodeKey(key);

    NodeInfo succ = this.findSuccessor(id);
    EventSource eventSource = client.listenForBindings(succ, info.id, key);

    state.addCallback(key, eventSource);
    eventSource.register(listener);
    eventSource.open();

}
```

The **listenOff** stop listening on that particular key.

```
public void listenOff(String key) throws DHTBase.Failed {
    /*
     * TODO: Stop listening for new binding events for this key.  Need to
     * do a Web service call to the server node, to stop event generation,
     * as well as close the event source here at the client.
     */

    //Add by Dhruvit
    int id = NodeKey(key);

    NodeInfo succ = this.findSuccessor(id);
    client.listenOff(succ, info.id, key);
    state.removeCallback(key);

}
```

Commands and Utilization

- LsitenOn: - We use this command to make client listen on that particular key. When anybody add any avlue to that key, client is notified with that value.

  → listenOn song

  → listenOn book

  → listenOn movie

- listenOff : - Make listen off on particular key

  → listenOff book

- Listeners : - Display all keys for those client is currently listening.

  → listeners

  {song, book}

Testing

- Local

We use following commands to set up peer to peer network between three nodes in the system.

java -jar dht.jar --http 8080 --id 23 --host localhost

java -jar dht.jar --http 8081 --id 45 --host localhost

java -jar dht.jar --http 8082 --id 37 --host localhost

I created three nodes 23, 37 and 45.

- Remote

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

1. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-68-9-52.us-west-2.compute.amazonaws.com

2. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com

3. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-69-96-171.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@ec2-54-68-9-52.us-west-2.compute.amazonaws.com:/home/ubuntu

2. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com:/home/ubuntu

3. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-54-69-96-171.us-west-2.compute.amazonaws.com:/home/ubuntu

Following command are used for set up peer-to-peer network between three Amazon EC2 Instances.

1. java -jar dht.jar --http 8080 --id 23 --host 172.31.28.76

2. java -jar dht.jar --http 8080 --id 37 --host 172.31.24.149

3. java -jar dht.jar --http 8080 --id 45 --host 172.31.19.147