2017

Réseaux et Technologie Internet

***Laurent Reynders & Océane Stasse (2301)***

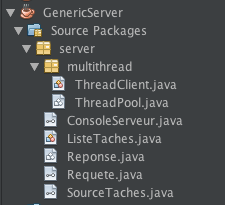
# Définition et implémentation des commandes du protocole LUGAP

|  |  |  |
| --- | --- | --- |
| **commande** | **Sémantique** | **Réponse attendue** |
| Login |  |  |
| Logout |  |  |
| GetFlights |  |  |
| GetLuggages |  |  |
| UpdateFieldReceived |  |  |
| UpdateFieldLoaded |  |  |
| UpdateFieldCheckedByCustoms |  |  |
| UpdateFieldComments |  |  |

## Code

# Code Java du serveur Serveur\_Bagages

Certaines classes n’ont pas été inséré dans cette partie car elles se trouvent dans le Syllabus « Java II – Programmation avancée des applications classiques et cryptographie » et n’ont pas été modifiée. Ces classes sont :



## JFrameServeurBagages.java

1. package serveur\_bagages;
2. import database.utilities.DatabaseAccess;
3. import java.awt.Component;
4. import java.io.FileInputStream;
5. import java.io.FileNotFoundException;
6. import java.io.FileOutputStream;
7. import java.io.IOException;
8. import java.net.ServerSocket;
9. import java.sql.SQLException;
10. import java.util.Properties;
11. import java.util.StringTokenizer;
12. import java.util.Vector;
13. import java.util.logging.Level;
14. import java.util.logging.Logger;
15. import javax.swing.JLabel;
16. import javax.swing.JScrollBar;
17. import javax.swing.JTable;
18. import javax.swing.JTextArea;
19. import javax.swing.table.DefaultTableCellRenderer;
20. import javax.swing.table.DefaultTableModel;
21. import javax.swing.table.TableCellRenderer;
22. import server.ConsoleServeur;
23. import server.ListeTaches;
24. public class JFrameServeurBagages extends javax.swing.JFrame implements ConsoleServeur {
25. private int ligneConsoleServeur = 1;
26. private final String propertyFilePath = "../serveurBagage.conf";
27. private final Properties propertyFile = new Properties();
28. private ThreadPoolLUGAP poolThread;
29. private int NB\_THREAD\_LUGAP;
30. private boolean isRunning = false;
31. private String PORT\_BAGAGES;
32. private String PORT\_CHECKIN;
33. private String DB\_IP\_ADDRESS;
34. private String DB\_PORT;
35. private String DB\_SCHEMA;
36. private String DB\_LOGIN;
37. private String DB\_PASSWORD;
38. private DatabaseAccess databaseAccess;
39. private final DefaultTableModel defaultTableModel = new DefaultTableModel(); /\*\* Creates new form JFrameServeurBagage \*/
40. public JFrameServeurBagages() {
41. initComponents();
42. defaultTableModel.addColumn("Ligne");
43. defaultTableModel.addColumn("Origine");
44. defaultTableModel.addColumn("Message");
45. defaultTableModel.addColumn("Lieu");
46. jTableConsole.setModel(this.defaultTableModel); //On change la taille des colonnes et l'alignement du texte
47. jTableConsole.getColumnModel().getColumn(0).setMaxWidth(40);
48. DefaultTableCellRenderer rightRenderer = new DefaultTableCellRenderer();
49. rightRenderer.setHorizontalAlignment(JLabel.RIGHT);
50. jTableConsole.getColumnModel().getColumn(0).setCellRenderer(rightRenderer);
51. jTableConsole.getColumnModel().getColumn(1).setMaxWidth(250);
52. jTableConsole.getColumnModel().getColumn(1).setPreferredWidth(120);
53. jTableConsole.getColumnModel().getColumn(2).setCellRenderer(new WordWrapCellRenderer());
54. jTableConsole.getColumnModel().getColumn(3).setMaxWidth(250);
55. jTableConsole.getColumnModel().getColumn(3).setPreferredWidth(115);
56. loadPropertyFile();
57. this.jTextFieldPort.setText(this.PORT\_BAGAGES);
58. }
59. /\*\* This method is called from within the constructor to     \* initialize the form.     \* WARNING: Do NOT modify this code. The content of this method is     \* always regenerated by the Form Editor.     \*/
60. @
61. SuppressWarnings("unchecked") // <editor-fold defaultstate="collapsed" desc="Generated Code">
62. private void initComponents() { //[...]
63. } // </editor-fold>
64. private void jButtonStartStopActionPerformed(java.awt.event.ActionEvent evt) {
65. try {
66. if (!isRunning) {
67. TraceEvenements("serveur#Démarrage du serveur...#main");
68. this.jButtonStartStop.setText("Stop");
69. TraceEvenements("serveur#acquisition du port#main");
70. String port = this.jTextFieldPort.getText();
71. if (!this.PORT\_BAGAGES.equals(port)) setPORT\_BAGAGES(port);
72. TraceEvenements("serveur#Lancement des threads#main");
73. int portInt = Integer.valueOf(this.PORT\_BAGAGES);
74. this.poolThread = new ThreadPoolLUGAP(this.NB\_THREAD\_LUGAP, new ListeTaches(), this, portInt, this.DB\_IP\_ADDRESS, this.DB\_PORT, this.DB\_SCHEMA, this.DB\_LOGIN, this.DB\_PASSWORD);
75. new ServerSocket(portInt).close(); //On check si la socket est libre ou non
76. this.poolThread.start();
77. this.isRunning = true;
78. TraceEvenements("serveur#Threads Lancés#main");
79. } else {
80. TraceEvenements("serveur#Extinction du serveur...#main");
81. this.jButtonStartStop.setText("Start");
82. this.poolThread.doStop();
83. this.isRunning = false;
84. }
85. } catch (IOException ex) {
86. TraceEvenements("serveur#ERREUR : " + ex.getMessage() + "#main");
87. this.jButtonStartStop.setText("Start");
88. this.poolThread.doStop();
89. this.isRunning = false;
90. }
91. }
92. private void buttonClearActionPerformed(java.awt.event.ActionEvent evt) {
93. ((DefaultTableModel) this.jTableConsole.getModel()).setRowCount(0);
94. }
95. /\*\*     \* @param args the command line arguments     \*/
96. public static void main(String args[]) { /\* Set the Nimbus look and feel \*/ //[...]
97. /\* Create and display the form \*/
98. java.awt.EventQueue.invokeLater(() - > {
99. new JFrameServeurBagages().setVisible(true);
100. });
101. } // Variables declaration - do not modify
102. private javax.swing.JButton buttonClear;
103. private javax.swing.JButton jButtonStartStop;
104. private javax.swing.JLabel jLabelPort;
105. private javax.swing.JScrollPane jScrollPane1;
106. private javax.swing.JSeparator jSeparator1;
107. private javax.swing.JTable jTableConsole;
108. private javax.swing.JTextField jTextFieldPort;
109. private javax.swing.JLabel labelServerStatus; // End of variables declaration
110. @
111. Override public void TraceEvenements(String commentaire) {
112. Vector ligne = new Vector();
113. ligne.add(Integer.toString(ligneConsoleServeur++));
114. StringTokenizer parser = new StringTokenizer(commentaire, "#");
115. while (parser.hasMoreTokens()) ligne.add(parser.nextToken());
116. defaultTableModel.addRow(ligne);
117. jScrollPane1.validate();
118. JScrollBar bar = this.jScrollPane1.getVerticalScrollBar();
119. bar.setValue(bar.getMaximum());
120. }
121. private void setPORT\_BAGAGES(String newPort) {
122. this.PORT\_BAGAGES = newPort;
123. this.propertyFile.setProperty("PORT\_BAGAGES", newPort);
124. try (FileOutputStream fileOutputStream = new FileOutputStream(this.propertyFilePath)) {
125. this.propertyFile.store(fileOutputStream, null);
126. } catch (IOException ex) {
127. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex);
128. System.exit(1);
129. }
130. }
131. private void loadPropertyFile() {
132. try (FileInputStream fileInputStream = new FileInputStream(this.propertyFilePath)) {
133. this.propertyFile.load(fileInputStream);
134. } catch (FileNotFoundException ex) {
135. try {
136. propertyFile.setProperty("THREAD\_NB\_LUGAP", "10");
137. propertyFile.setProperty("PORT\_BAGAGES", "26010");
138. propertyFile.setProperty("PORT\_CHECKIN", "26020");
139. propertyFile.setProperty("DB\_IP\_ADDRESS", "127.0.0.1");
140. propertyFile.setProperty("DB\_PORT", "3306");
141. propertyFile.setProperty("DB\_SCHEMA", "BD\_AIRPORT");
142. propertyFile.setProperty("DB\_LOGIN", "LaboReseaux");
143. propertyFile.setProperty("DB\_PASSWORD", "mysql");
144. try (FileOutputStream fileOutputStream = new FileOutputStream(this.propertyFilePath)) {
145. propertyFile.store(fileOutputStream, null);
146. }
147. } catch (IOException ex1) {
148. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex1);
149. }
150. } catch (IOException ex) {
151. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex);
152. System.exit(1);
153. }
154. this.NB\_THREAD\_LUGAP = Integer.valueOf(propertyFile.getProperty("THREAD\_NB\_LUGAP", "10"));
155. this.PORT\_BAGAGES = propertyFile.getProperty("PORT\_BAGAGES", "26010");
156. this.PORT\_CHECKIN = propertyFile.getProperty("PORT\_CHECKIN", "26020");
157. this.DB\_IP\_ADDRESS = propertyFile.getProperty("DB\_IP\_ADDRESS", "127.0.0.1");
158. this.DB\_PORT = propertyFile.getProperty("DB\_PORT", "3306");
159. this.DB\_SCHEMA = propertyFile.getProperty("DB\_SCHEMA", "BD\_AIRPORT");
160. this.DB\_LOGIN = propertyFile.getProperty("DB\_LOGIN", "LaboReseaux");
161. this.DB\_PASSWORD = propertyFile.getProperty("DB\_PASSWORD", "mysql");
162. }
163. static class WordWrapCellRenderer extends JTextArea implements TableCellRenderer {
164. WordWrapCellRenderer() {
165. setLineWrap(true);
166. setWrapStyleWord(true);
167. setOpaque(true);
168. }@
169. Override public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
170. setText(value.toString());
171. setSize(table.getColumnModel().getColumn(column).getWidth(), getPreferredSize().height);
172. setOpaque(true);
173. setBorder(null);
174. setForeground(table.getForeground());
175. setBackground(table.getBackground());
176. if (table.getRowHeight(row) != getPreferredSize().height) {
177. table.setRowHeight(row, getPreferredSize().height);
178. }
179. return this;
180. }
181. }
182. }

## ThreadPool.java

1. package server.multithread;
2. import communicator.CommunicatorException;
3. import java.io.IOException;
4. import java.net.ServerSocket;
5. import java.net.Socket;
6. import java.util.logging.Level;
7. import java.util.logging.Logger;
8. import server.ConsoleServeur;
9. import server.SourceTaches;
10. public abstract class ThreadPool extends Thread {
11. protected final int listeningPort;
12. protected final SourceTaches tachesAExecuter;
13. protected final ConsoleServeur guiApplication;
14. protected ServerSocket serverSocket = null;
15. protected final int nbThreads;
16. public ThreadPool(int nbThreads, SourceTaches tachesAExecuter, ConsoleServeur guiApplication, int port) {
17. this.listeningPort = port;
18. this.tachesAExecuter = tachesAExecuter;
19. this.guiApplication = guiApplication;
20. this.nbThreads = nbThreads;
21. }@
22. Override public void run() {
23. try {
24. this.serverSocket = new ServerSocket(this.listeningPort);
25. } catch (IOException ex) {
26. Logger.getLogger(ThreadPool.class.getName()).log(Level.SEVERE, null, ex);
27. System.exit(1);
28. } //Démarrage du pool de threads
29. for (int i = 0; i < this.nbThreads; i++) {
30. new ThreadClient(tachesAExecuter, "Thread du pool n°" + String.valueOf(i)).start();
31. } //Mise en attente du serveur
32. Socket socket;
33. while (!isInterrupted()) {
34. try {
35. socket = serverSocket.accept();
36. this.guiApplication.TraceEvenements("serveur#Accepted an incoming connection#ThreadPoolLUGAP");
37. tachesAExecuter.recordTache(getProtocolRunnable(socket));
38. } catch (IOException ex) {
39. System.err.println("Erreur d'accept ! ? [" + ex.getMessage() + "]");
40. } catch (CommunicatorException ex) {
41. System.err.println("Erreur de communication ! ? [" + ex.getMessage() + "]");
42. }
43. }
44. }
45. public void doStop() {
46. this.guiApplication.TraceEvenements("serveur#Arret du serveur#ThreadPoolLUGAP");
47. try {
48. if (this.serverSocket != null) this.serverSocket.close();
49. } catch (IOException ex) {
50. this.guiApplication.TraceEvenements("serveur#" + ex.getMessage() + "#ThreadPoolLUGAP");
51. }
52. this.interrupt();
53. }
54. protected abstract Runnable getProtocolRunnable(Socket socket) throws CommunicatorException;
55. }

## ThreadPoolLUGAP.java

1. package serveur\_bagages;
2. import communicator.Communicator;
3. import communicator.CommunicatorException;
4. import database.utilities.DatabaseAccess;
5. import entities.Flight;
6. import java.net.Socket;
7. import java.util.HashMap;
8. import server.ConsoleServeur;
9. import server.SourceTaches;
10. import server.multithread.ThreadPool;
11. public class ThreadPoolLUGAP extends ThreadPool {
12. private final String DBip;
13. private final String DBport;
14. private final String DBSID;
15. private final String DBschema;
16. private final String DBpassword;
17. private final HashMap < RunnableLUGAP, Flight > usedFlightList;
18. public ThreadPoolLUGAP(int nbThreads, SourceTaches tachesAExecuter, ConsoleServeur guiApplication, int port, String ip, String DBport, String SID, String schema, String password) {
19. super(nbThreads, tachesAExecuter, guiApplication, port);
20. this.DBip = ip;
21. this.DBport = DBport;
22. this.DBSID = SID;
23. this.DBschema = schema;
24. this.DBpassword = password;
25. this.usedFlightList = new HashMap < > ();
26. }@
27. Override protected Runnable getProtocolRunnable(Socket socket) throws CommunicatorException {
28. return new RunnableLUGAP(this, this.guiApplication, new Communicator(socket), new DatabaseAccess(DatabaseAccess.databaseType.MYSQL, this.DBip, this.DBport, this.DBSID, this.DBschema, this.DBpassword));
29. }
30. public synchronized HashMap < RunnableLUGAP, Flight > getUsedFlightList() {
31. return usedFlightList;
32. }
33. public synchronized void putFlightInList(RunnableLUGAP runnableLUGAP, Flight flight) {
34. this.usedFlightList.put(runnableLUGAP, flight);
35. }
36. public synchronized void removeFlightFromList(RunnableLUGAP runnableLUGAP) {
37. this.usedFlightList.remove(runnableLUGAP);
38. }
39. }

## RunnableLUGAP.java

1. package serveur\_bagages;
2. import communicator.Communicator;
3. import communicator.CommunicatorException;
4. import database.utilities.DatabaseAccess;
5. import java.sql.SQLException;
6. import java.util.ArrayList;
7. import java.util.logging.Level;
8. import java.util.logging.Logger;
9. import lugap.reponse.ReponseLUGAP\_getFlights;
10. import lugap.reponse.ReponseLUGAP\_getLuggages;
11. import lugap.reponse.ReponseLUGAP\_updateField;
12. import lugap.requete.RequeteLUGAP;
13. import lugap.requete.RequeteLUGAP\_getFlights;
14. import lugap.requete.RequeteLUGAP\_getLuggages;
15. import lugap.requete.RequeteLUGAP\_login;
16. import lugap.requete.RequeteLUGAP\_logout;
17. import lugap.requete.RequeteLUGAP\_updateFieldCheckedByCustoms;
18. import lugap.requete.RequeteLUGAP\_updateFieldComments;
19. import lugap.requete.RequeteLUGAP\_updateFieldLoaded;
20. import lugap.requete.RequeteLUGAP\_updateFieldReceived;
21. import server.ConsoleServeur;
22. public class RunnableLUGAP implements Runnable {
23. private final ThreadPoolLUGAP parent;
24. private final ConsoleServeur guiApplication;
25. private final Communicator communicator;
26. private final DatabaseAccess databaseAccess;
27. private Class previousRequete;
28. private boolean endOfTransaction;
29. private boolean clientConnected;
30. RunnableLUGAP(ThreadPoolLUGAP parent, ConsoleServeur guiApplication, Communicator communicator, DatabaseAccess databaseAccess) throws CommunicatorException {
31. this.parent = parent;
32. this.guiApplication = guiApplication;
33. this.communicator = communicator;
34. this.databaseAccess = databaseAccess;
35. try {
36. this.databaseAccess.connect();
37. } catch (ClassNotFoundException | SQLException ex) {
38. throw new CommunicatorException(ex.getMessage());
39. }
40. }@
41. Override public void run() {
42. this.endOfTransaction = false;
43. try {
44. while (!this.endOfTransaction) {
45. RequeteLUGAP req = this.communicator.receiveSerializable(RequeteLUGAP.class);
46. System.out.println("Received request type : " + req.getClass().toString());
47. Runnable runnable = req.createRunnable(this.parent, this.communicator, this.guiApplication, this.databaseAccess);
48. if (req instanceof RequeteLUGAP\_login) {
49. runnable.run();
50. if (req.requeteSucceeded()) {
51. setClientIsConnected();
52. previousRequete = RequeteLUGAP\_login.class;
53. }
54. } else if (req instanceof RequeteLUGAP\_logout) {
55. runnable.run();
56. endTransaction();
57. previousRequete = null;
58. } else if (req instanceof RequeteLUGAP\_getFlights) {
59. if (!isClientConnected()) this.communicator.SendSerializable(ReponseLUGAP\_getFlights.KO("Please connect first!"));
60. else {
61. ((RequeteLUGAP\_getFlights) req).setUsedFlights(new ArrayList(this.parent.getUsedFlightList().values()));
62. runnable.run();
63. previousRequete = RequeteLUGAP\_getFlights.class;
64. }
65. } else if (req instanceof RequeteLUGAP\_getLuggages) {
66. if (!isClientConnected() || !previousRequete.equals(RequeteLUGAP\_getFlights.class)) this.communicator.SendSerializable(ReponseLUGAP\_getLuggages.KO("Please do a getFlights first!"));
67. else {
68. this.parent.putFlightInList(this, ((RequeteLUGAP\_getLuggages) req).getFlight());
69. runnable.run();
70. previousRequete = RequeteLUGAP\_getLuggages.class;
71. }
72. } else if (req instanceof RequeteLUGAP\_updateFieldReceived || req instanceof RequeteLUGAP\_updateFieldLoaded || req instanceof RequeteLUGAP\_updateFieldCheckedByCustoms || req instanceof RequeteLUGAP\_updateFieldComments) {
73. if (!isClientConnected() || !previousRequete.equals(RequeteLUGAP\_getLuggages.class)) this.communicator.SendSerializable(ReponseLUGAP\_updateField.KO("Please do a getLuggages first!"));
74. else runnable.run();
75. }
76. }
77. this.parent.removeFlightFromList(this);
78. this.databaseAccess.commit();
79. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#Commit successful" + "#LUGAPRunnable");
80. this.communicator.close();
81. } catch (CommunicatorException | SQLException ex) {
82. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#" + ex.getMessage() + "#LUGAPRunnable");
83. try {
84. this.databaseAccess.rollback();
85. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#Rollback successful" + "#LUGAPRunnable");
86. } catch (SQLException ex1) {
87. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#" + ex.getMessage() + "#LUGAPRunnable");
88. }
89. }
90. }
91. private void endTransaction() {
92. this.endOfTransaction = true;
93. }
94. private boolean isClientConnected() {
95. return this.clientConnected;
96. }
97. private void setClientIsConnected() {
98. this.clientConnected = true;
99. }
100. }

# Trames échangées et vues par un sniffer