Programmazione Web

Docente: Novella Bartolini

Lezione del 18 Maggio 2020

Sette passi per connettersi ad una base di dati sfruttando il supporto JDBC

1. Caricare il driver
2. Definire l'URL per la connessione con la base dati
3. Instaurare la connessione
4. Creare un oggetto Statement che rappresenta la query da inoltrare
5. Eseguire la query
6. Elaborare il risultato
7. Chiudere la connessione

I sette passi nel dettaglio

1. Caricare il driver

```
try {
  Class.forName("com.mysql.jdbc.Driver");
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
} catch { ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " cnfe);
}
```

I sette passi nel dettaglio

2. Definire il percorso (URL) della connessione

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String accessURL = "jdbc:mySQL:" + host +
                   ":" + port + ":" + dbName;
```

I sette passi nel dettaglio

3. Instaurare una connessione

```
String username = "Paperone";
String password = "numero1";
Connection connection =
    DriverManager.getConnection(accessURL,
                                username,
                                password);
```

I sette passi nel dettaglio

4. Creare un oggetto Statement che rappresenta la query da inoltrare

```
Statement statement =
   connection.createStatement();
```

I sette passi nel dettaglio

5. Eseguire una query

```
Statement statement = connection.createStatement();
String query =
   "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet =
   statement.executeQuery(query);
```

- Per modificare la base dati, usare l'istruzione **executeUpdate**, fornendo comandi SQL come **UPDATE**, **INSERT**, o **DELETE**
- Usare **setQueryTimeout** per specificare il massimo intervallo di tempo che si è disposti ad aspettare prima per ottenere la risposta

I sette passi nel dettaglio

6. Elaborazione della risposta

```
while(resultSet.next()) {
  System.out.println(resultSet.getString(1) +
  " " + resultSet.getString(2) +
  " " + resultSet.getString(3));
}
```

   - La prima colonna ha indice 1, non 0
   - **`ResultSet`** fornisce numerosi metodi **`getXxx`** con argomento un indice di colonna *o il nome della colonna*

I sette passi nel dettaglio

7. Chiudere la connessione

```
connection.close();
```

   - Dal momento che le operazioni di apertura di una connessione sono molto costose, posporre questa operazione se sono necessarie ulteriori interazioni con la base dati

Analisi dei tempi di risposta

- L'apertura di una connessione ad una base di dati, specialmente se quest'ultima risiede su un host separato dal server, è un processo oneroso, con alti tempi di completamento
  - Ri-uso delle connessioni
  - Realizzazione di un pool di connessioni

Una classe per realizzare un pool di connessioni

- Deve essere in grado di:
  1. Preallocare le connessioni
  2. Gestire le connessioni disponibili
  3. Allocare nuove connessioni se necessario
  4. Attendere che una connessione si renda disponibile
  5. Chiudere le connessioni ove necessario

1. Preallocare le connessioni

- La preallocazione delle connessioni produce vantaggi e svantaggi:
  - Richiede un tempo di avvio maggiore
  - Velocizza gli accessi alla base dati
- In una servlet che gestisce un pool, le connessioni devono essere preallocate nel metodo **`init()`**, affinchè siano disponibili prima che si verifichino le richieste.

1. Preallocare le connessioni (cont.)

- Utilizziamo un vettore per memorizzare
  - connessioni disponibili e inattive
  - connessioni non disponibili perchè in uso

```
availableConnections = new Vector(initialConnections);
busyConnections= new Vector();
for (int i=0; i<initialConnections; i++) {
   availableConnections.addElement(makeNewConnections());
```

(makeNewConnection() utilizza l'URL, e username e password forniti precedentemente)

1. Preallocare le connessioni (cont.)

- Il vettore di connessioni occupate serve
  - per definire un limite superiore al numero di connessioni che possono essere aperte contemporaneamente
  - per poter utilizzare un metodo che chiuda esplicitamente tutte le connessioni.

2. Gestire le connessioni disponibili

- Quando si verifica una richiesta al db, viene fornita una connessione, se ve sono di disponibili.
- La connessione assegnata alla richiesta viene rimossa dal vettore delle connessioni disponibili e viene inserita nel vettore delle connessioni occupate.

2. Gestire le connessioni disponibili

- Attenzione: le connessioni possono avere un *tempo di timeout*, perciò prima di assegnare una connessione ad una richiesta verificare che sia ancora aperta
- Se la connessione non è più aperta, questa deve essere scartata, rendendo così disponibile uno slot per eventuali thread in attesa
  - Se un processo ha richiesto una connessione quando era già stato raggiunto il limite massimo sul numero di connessioni, si può utilizzare il metodo **`notifyAll`** per risvegliare tutti i thread in attesa

2. Gestire le connessioni disponibili (cont.)

```
public synchronized Connection getConnection()
                throws SQLException {
   if (!availableConnections.isEmpty()) {
      Connection existingConnection =
           (Connection)availableConnections.lastElement();
      int lastIndex = availableConnections.size() - 1;
      availableConnections.removeElementAt(lastIndex);
      if (existingConnection.isClosed()) {
         notifyAll(); // Segnala l'evento ai thread in attesa
         return(getConnection()); // Ripete il procedimento
      } else {
         busyConnections.addElement(existingConnection);
         return(existingConnection);
      }
   }
}
```

3. Allocare nuove connessioni

- Se
  - viene richiesta una connessione, e
  - non ci sono connessioni disponibili, e
  - non è stato raggiunto il limite massimo sul numero di connessioni
- avviare un thread in background per allocare una nuova connessione e attendere che una connessione si renda disponibile (quella appena creata o eventualmente un’altra)

3. Allocare nuove connessioni (cont.)

```
if ((totalConnections() < maxConnections) &&
      !connectionPending) {
      // Pending = che si sta connettendo in background
      makeBackgroundConnection();
}
try {
   wait(); // Si auto-sospende
} catch(InterruptedException ie) {}
return(getConnection()); // Riprova
```

La creazione di una nuova connessione viene condotta attraverso un thread in background (questo per non perdere i vantaggi dell'utilizzo del pool, ricordiamo che instaurare una connessione richiede tempo...)

4. Attendere che una nuova connessione si renda disponibile

- Se
  - viene richiesta una connessione, e
  - è già stato raggiunto il limite massimo sul numero di connessioni

  attendere che una connessione si renda disponibile oppure lanciare un’eccezione.

4. Attendere che una nuova connessione si renda disponibile (cont.)

- L'approccio più semplice è quello di utilizzare il metodo **`wait()`** che sospende il thread fino al momento in cui non viene chiamato il metodo **`notify`** o **`notifyAll`**
- Dal momento che il segnale di notifica può essere generato per molti motivi, i thread risvegliati devono verificare se possono procedere o no (lo fanno ripetendo il tentativo di ottenere la connessione).

4. Attendere che una nuova connessione si renda disponibile (cont.)

```
try {
    wait();
} catch(InterruptedException ie) {}
return(getConnection());
```

oppure se non si vuole far attendere i client e si preferisce generare un'eccezione:

```
throw new SQLException("Raggiunto il limite sul numero di
                                                connessioni");
```

5. Chiudere le connessioni ove necessario

- Le connessioni vengono chiuse durante le operazioni di garbage collection (non è sempre necessario chiuderle esplicitamente)

```
public synchronized void closeAllConnections() {
   //Il metodo closeConnections esegue un ciclo con cui
   //scandisce tutto il vettore e
   //chiude tutte le connessioni ignorando le eccezioni
   closeConnections(availableConnections);
   availableConnections = new Vector();
   closeConnections(busyConnections);
   busyConnections = new Vector();
}
```

ConnectionPool.java (1/7)

```
import java.sql.*;
import java.util.*;
public class ConnectionPool implements Runnable {
  private String driver, url, username, password;
  private int maxConnections;
  private boolean waitIfBusy;
  private Vector availableConnections, busyConnections;
  private boolean connectionPending = false;

  public ConnectionPool(String driver, String url, String username, String password,
                        int initialConnections, int maxConnections, boolean waitIfBusy)
                        throws SQLException {
    this.driver = driver;
    this.url = url;
    this.username = username;
    this.password = password;
    this.maxConnections = maxConnections;
    this.waitIfBusy = waitIfBusy;
    if (initialConnections > maxConnections) {
      initialConnections = maxConnections;
    }
    availableConnections = new Vector(initialConnections);
    busyConnections = new Vector();
    for(int i=0; i<initialConnections; i++) {
      availableConnections.addElement(makeNewConnection());
    }
  }
```

ConnectionPool.java (2/7)

```
public synchronized Connection getConnection()
      throws SQLException {
    if (!availableConnections.isEmpty()) {
      Connection existingConnection =
        (Connection)availableConnections.lastElement();
      int lastIndex = availableConnections.size() - 1;
      availableConnections.removeElementAt(lastIndex);
      // If connection on available list is closed (e.g.,
      // it timed out), then remove it from available list
      // and repeat the process of obtaining a connection.
      // Also wake up threads that were waiting for a
      // connection because maxConnection limit was reached.
      if (existingConnection.isClosed()) {
        notifyAll(); // Freed up a spot for anybody waiting
        return(getConnection());
      } else {
        busyConnections.addElement(existingConnection);
        return(existingConnection);
      }
    } else { //il vettore delle connessioni disponibili è vuoto
```

ConnectionPool.java (3/7)

```
// Tre casi possibili
      // 1) Non si è raggiunto il limite max sul num. di connessioni.
      //    Ne viene aperta una in bg se non c'è una richiesta pending
      // 2) Si è raggiunto il limite max sul num. di connessioni e waitIfBusy
      //    vale false. Si lancia una SQLException.
      // 3) E' stato raggiunto il max num di connessioni e il flag waitIfBusy
      //    vale true. Si attende la disponibilità di una nuova connessione.
      if ((totalConnections() < maxConnections) &&
          !connectionPending) {
        makeBackgroundConnection();
      } else if (!waitIfBusy) {
        throw new SQLException("Connection limit reached");
      }
      // Wait for either a new connection to be established
      // (if you called makeBackgroundConnection) or for
      // an existing connection to be freed up.
      try {
        wait();
      } catch(InterruptedException ie) {}
      // Someone freed up a connection, so try again.
      return(getConnection());
    }
  }
```

ConnectionPool.java (4/7)

```
private void makeBackgroundConnection() {
    connectionPending = true;
    try {
      Thread connectThread = new Thread(this);
      connectThread.start();
    } catch(OutOfMemoryError oome) {
      // Rinuncia alla nuova connessione
    }
  }

  public void run() {
    try {
      Connection connection = makeNewConnection();
      synchronized(this) {
        availableConnections.addElement(connection);
        connectionPending = false;
        notifyAll();
      }
    } catch(Exception e) { // SQLException or OutOfMemory
      // Rinuncia alla nuova connessione e attende che una di quelle
      //esistenti si liberi    }
  }
```

ConnectionPool.java (5/7)

```
  //Questo metodo crea esplicitamente una nuova connessione. E' invocato in
 // fg all'inizializzazione e in bg in esecuzione (run).

  private Connection makeNewConnection()
      throws SQLException {
    try {
      Class.forName(driver);
      Connection connection =
        DriverManager.getConnection(url, username, password);
      return(connection);
    } catch(ClassNotFoundException cnfe) {
          throw new SQLException("Can't find class for driver: " + driver);
    }
  }

public synchronized void free(Connection connection) {
    busyConnections.removeElement(connection);
    availableConnections.addElement(connection);
    notifyAll();
  }
```

ConnectionPool.java (6/7)

```
public synchronized void closeAllConnections() {
    closeConnections(availableConnections);
    availableConnections = new Vector();
    closeConnections(busyConnections);
    busyConnections = new Vector();
  }

  private void closeConnections(Vector connections) {
    try {
      for(int i=0; i<connections.size(); i++) {
        Connection connection =
          (Connection)connections.elementAt(i);
        if (!connection.isClosed()) {
          connection.close();
        }
      }
    } catch(SQLException sqle) {
      // Ignore errors; garbage collect anyhow
    }
  }
```

ConnectionPool.java (7/7)

```
public synchronized int totalConnections() {
    return(availableConnections.size() +
           busyConnections.size());
  }



public synchronized String toString() {
    String info =
      "ConnectionPool(" + url + "," + username + ")" +
      ", available=" + availableConnections.size() +
      ", busy=" + busyConnections.size() +
      ", max=" + maxConnections;
    return(info);
  }
}
```

Studio delle prestazioni del connection pool

- Una pagina html invoca 25 volte una stessa servlet che esegue una query ad un DB
- Confrontiamo tre situazioni:
  1. la servlet alloca un ConnectionPool nel suo metodo init
  2. la servlet utilizza e ricicla una singola connessione
  3. la servlet non utilizza un pool e instaura una nuova connessione ad ogni richiesta

Caso di connessione lenta (DB remoto)

Tempo medio di risposta

- (Caso 1: ConnectionPoolServlet) connessione lenta al DB
  - 10 connessioni iniziali, max_connections 50
  - 11 seconds
- (Caso 2: ConnectionPoolServlet2) connessione lenta al DB
  - Riciclo di un'unica connessione
  - 22 seconds
- (Caso 3: ConnectionPoolServlet3) connessione lenta al DB
  - non viene utilizzato nessun pooling
  - 82 seconds

Confronto nel caso di LAN

Tempo medio di risposta

- (Caso 1: ConnectionPoolServlet) connessione LAN
  - 10 connessioni iniziali, max_connections 50
  - 1.8 seconds
- (Caso 2: ConnectionPoolServlet2) connessione LAN
  - Riciclo di un'unica connessione
  - 2.0 seconds
- (Caso 3: ConnectionPoolServlet3) connessione LAN
  - non viene utilizzato nessun pooling
  - 2.8 seconds

ConnectionPoolServlet.java (1/4)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class ConnectionPoolServlet extends HttpServlet {
  private ConnectionPool connectionPool;

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String table;
    try {
      String query =
        "SELECT Nome, Cognome firstname, lastname " +
        " FROM impiegati";
      Connection connection = connectionPool.getConnection();
      Statement statement = connection.createStatement();
      ResultSet results = statement.executeQuery(query);
      connectionPool.free(connection);
      table = .... //formatta tabella results come visto in NorthwindServlet;
    } catch(Exception e) {
      table = "Error: " + e;
    }
```

ConnectionPoolServlet.java (2/4)

```
response.setContentType("text/html");
   String title="pagina di output";
   PrintWriter out = response.getWriter();
      out.println("<html><head><title>"+title+"</title>"+
                  "<META HTTP-EQUIV="Pragma" CONTENT="no-cache">+
                  "<META HTTP-EQUIV="Expires=-1" CONTENT="-1">+
                  "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                  "<CENTER>\n" + table + "\n" +
                  "</CENTER>\n</BODY></HTML>");
 }
```

ConnectionPoolServlet.java (3/4)

```
public void init() {

    String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    String url = "jdbc:odbc:Northwind";
    String username = "";
    String password = "";
    try {
      connectionPool =
        new ConnectionPool(driver, url, username, password,
                           initialConnections(),
                           maxConnections(), true);
    } catch(SQLException sqle) {
      System.err.println("Error making pool: " + sqle);
      getServletContext().log("Error making pool: " + sqle);
      connectionPool = null;
    }
  }
```

ConnectionPoolServlet.java (4/4)

```
public void destroy() {
    connectionPool.closeAllConnections();
  }

protected int initialConnections() {
    return(10);
  }

protected int maxConnections() {
    return(50);
  }
}
```

ConnectionPool.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD><TITLE>Servlet Connection Pooling: A Test</TITLE></HEAD>

25 richieste simultaneee della stessa servlet -->

<FRAMESET ROWS="*,*,*,*,*" BORDER=0 FRAMEBORDER=0 FRAMESPACING=0>
  <FRAMESET COLS="*,*,*,*,*">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
  </FRAMESET>
  ...
<FRAMESET COLS="*,*,*,*,*">
 <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
    <FRAME SRC="/servlet/ConnectionPoolServlet">
  </FRAMESET>
</FRAMESET>

</HTML>
```

ConnectionServlet2.java

```
/** Una variante di ConnectionPoolServlet **/

public class ConnectionPoolServlet2
      extends ConnectionPoolServlet {

  protected int initialConnections() {
    return(1);
  }

  protected int maxConnections() {
    return(1);
  }
}
```

La classe **DbConnectionBroker**

- E’ uno tra i numerosi esempi esistenti di classi che realizzano un pool di connessioni
- Si crea un’istanza della classe:

```
...
dbPool= new DbConnectionBroker(driverName,connectionURL, userID, password
              minConnections, maxConnections, logFile, daysBetweenResets);

Connection connection=dbPool.getConnection();
Statement stm=connection.createStatement();
ResultSet rset=stm.executeQuery(“SELECT * FROM ...”);
...
dbPool.freeConnection(connection);
```

The End