Databass Access

Vittayasak Rujivorakul

Lecture 4

Databass Access

Vittayasak Rujivorakul

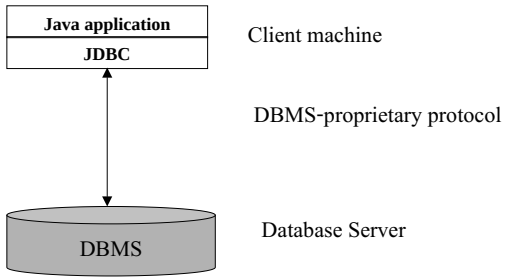
Lecture 4

Distributed Models

- คือการ running หรือทำงาน แยกกัน และมีการติดต่อข้ามเครื่อง
- Two - tier (client / server) เป็นการติดต่อระหว่าง client หลายตัว กับ server หนึ่งหรือหลายตัว
- Multi - tier เป็นการติดต่อระหว่าง client หลายตัว กับ server หนึ่งหรือหลายตัว โดยมีการติดต่อกับ server อื่น ๆ ด้วย

Two - Tier (Client / Server)

- เป็นการเชื่อมต่อแบบ Client / Server โดยที่ Server Share Application และ Database เพื่อให้ client หรือ applet ติดต่อเข้ามาที่ server โดยตรง

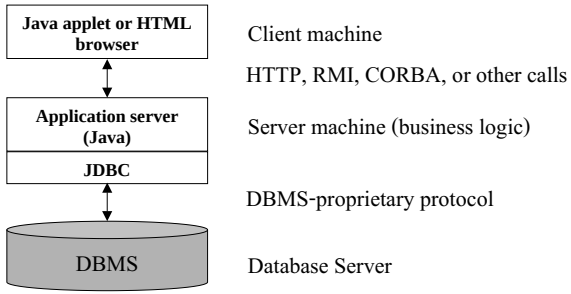


Two - Tier (ต่อ)

- Client ที่ติดต่อเข้ามา จำเป็นต้องมีทรัพยากร ที่มีประสิทธิภาพ (thick client) เพื่อใช้ในการประมวลผลฝั่ง client
- การเชื่อมต่อจะเป็นแบบง่าย ๆ ประสิทธิภาพขึ้นอยู่กับเครื่อง client
- การตรวจสอบจุดผิดพลาดมีแค่ 3 จุด คือ Java Application, JDBC และ Data Base
- ง่ายในการดูแลและควบคุมระบบ

Multi - Tier

- การเชื่อมต่อแบบนี้จะมี client หลาย ๆ เครื่องเชื่อมต่อผ่าน midleware เช่น servlets ซึ่งใช้ในการติดต่อกับ database server, application server หรืออื่น ๆ โดยทำงานเหล่านี้จะผ่านทาง web browser



Multi - tier (ต่อ)

- Client ไม่จำเป็นต้องมีประสิทธิภาพสูง อาจจะมีแค่ browser เท่านั้น (thin client)
- รูปแบบการเชื่อมต่อค่อนข้างซับซ้อน แต่ว่าทางประสิทธิภาพ
- ง่ายในการดูและรักษา
- ง่านในการปรับเปลี่ยนตัวเชื่อมหรือ device driver
- การตรวจสอบจุดผิดพลาดทำได้ยาก เนื่องจากมีหลายจุดที่ต้องตรวจสอบ

Java Database Connectivity (JDBC)

- JDBC API ถูกสร้างด้วย Java technology ใช้เป็นตัวกลางในการติดต่อกับฐานข้อมูล
- Generic interface class เป็นมาตรฐานข้อมูลของแต่ละค่าย ให้ implement driver เฉพาะขึ้นมา
- รองรับการทำงาน หลาย ๆ protocol เช่น Structured Query Language (SQL)
- Package ที่ต้อง import คือ java.sql.*

JDBC vs Servlets

```
public class DBServlet extends HttpServlet {
    static final String url = "jdbc:mysql://logic.co.th:1119/mydatabase";
    static final String jdbcclass = "org.gjt.mm.mysql.Driver";
    static final String user = "";
    static final String pass = "";
    static final int initialCons = 5;
    static final int maxCons = 20;
    static final boolean block = true;
    static final long timeout  = 10000;

    public void init(ServletConfig config) throws ServletException{
        super.init(config);
        log("DBServlet.initialize: enter");
        try{
            Class.forName(jdbcclass).newInstance();
        } catch(Exception e){
            e.printStackTrace();
            String errmsg = e.getMessage();
            Log("Failed:" + errmsg);
        }
    }
```

JDBC Servlets (ต่อ)

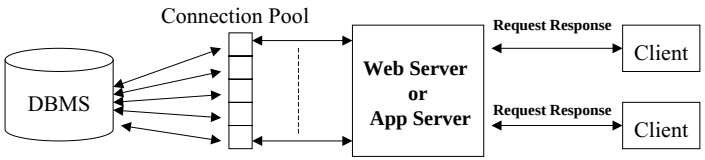
```
public void service(HttpServletRequest req,HttpServletResponse res) throws  ServletException, IOException{
        String lname;
        Connection con = null;
        PrintStream out = null;

        try {
                // get the connection from the database
                con = DriverManager.getConnection(url);

        // create and excute the query
        Statement stmt = con.createStatement();
        String query = "SELECT first_name, last_name, comment" +
                                        "FROM emp_details" + "WHERE last_name='"
                                        + lname + "'";
        ResulSet rs = stmt.excuteQuery(query);
        // print out the result
        dispResultSet(rs,out);
        rs.close();
        stmt.close();
        con.close();
        }
    }
```


JDBC Servlets (ต่อ)

```
    private void dispResultSet(ResultSet rs,PrintStream out) throws SQLException{
    int I;
    // metadata can supply information about the schema
    ResultSetMetaData rsmd = rs.getMetaData();
    // User metadata to print out result set
    ...
    }
}
```


Connection Pool

- Connection pool ถูกสร้างขึ้นมาเพื่อแก้ปัญหา เรื่องของเวลา และการเชื่อมต่อกับฐานข้อมูล โดยการสร้าง pool (กลุ่มของการเชื่อมต่อ) ที่คงอยู่ จนกว่า server จะปิด และสามารถนำ connection นั้น ๆ กลับมาใช้ใหม่ได้โดยไม่ต้องร้องขอการเชื่อมต่อไปยังฐานข้อมูลทุก ๆ ครั้ง



Items to Consider

- Connection pool ต้องสามารถ defect และคลอบคลุมการ failures ของ connection ได้
- จะต้องสามารถกำหนดจำนวน connection ได้ว่าจะเริ่มที่กี่ connection และเมื่อมีการใช้งานเกินจำนวนที่ตั้งไว้ ก็สามารถที่จะ create เพิ่มเองได้โดยอัตโนมัติ
- สามารถตัดสินใจได้ว่าเมื่อเกิดการ connect จนเต็ม client ที่เข้ามาทีหลังจะต้องรอจนกว่า connection จะว่างจึงจะให้ใช้งานได้ (ทำตัวเหมือน queue)
- สามารถเก็บ log events และ errors ต่าง ๆ ได้

Implementing a Connection Pool

- ขั้นตอนการสร้าง connection pool จะแบ่งการ implement ออกเป็น 2 ส่วน คือ
  - ส่วนของ Servlet หรือ connection pool client
  - ส่วนของ connection pool API

Servlet (Connection pool client)

- Connection pool servlet จะอยู่ในส่วนของ middle tier ของ multi-tier model ซึ่งจะทำงานดังต่อไปนี้

1. Load driver ของ database
2. สร้าง instance ของ connection pool
3. Request connection จาก connection pool ซึ่งตัว connection pool จะมีการทำงานอยู่ 2 แบบ คือ
   – แบบ blocking : หากมีการ connect เข้ามาเกินจำนวน connection ที่ตั้งไว้ให้ client ที่เข้ามาทำการรอจนกว่าตัวอื่น จะใช้งาน connection นั้นเสร็จ

Servlet (ต่อ)

   – แบบ Dynamic คือหากมีการ connect เข้ามาเกิน จำนวน connection ที่ตั้งไว้ให้ทำการเพิ่ม connection โดยอัตโนมัติ
4. ใช้ connection ที่ได้ทำการ initiate query
5. เมื่อใช้งานเสร็จแล้วคืน connection ให้กับ pool
6. ตัดการเชื่อมต่อกับ pool เมื่อ servlet destroyed

Connection Pool API

- Connection pool จะต้องสามารถทำงานพื้นฐานเกี่ยวกับ database ได้เช่นการ create และ close connection

Create ConnectionPool

```
// Load the JDBC driver
// Instantiate a connection pool
ConnectionPool cp = new ConectionPool(url, user, pass, initialCons, maxCons, block, timeout);
// Get a connection from the connection pool
con = cp.getConnection();

// Submit a query using the connection

// Release the connection to the pool
cp.releaseConnection(con);
// Close all of the connections
cp.closeAll();
```


ConnectionPool Constructor

```
public ConnectionPool(String url, String user, String password, int initialCons, int maxCons, boolean
    block, long timeout) throws SQLException{
// Create vectors large enough to store all the connections we make now.
    free = new Vector (initialCons);
    used = new Vector (initialCons);
// Create some connections
while(numCons < initialCons){
    addConnection();
}
}


private void addConnection() throws SQLException{
    free.addElement(getNewConnection());
}
```


getNewConection

```
private Connection getNewConnection() throws SQLException {
    Connection con = null;
    System.out.println("About to connect to" + url);
    try{
        con = DriverManager.getConnection(url);
    } catch(Exception e) {
        e.printStackTrace();
    }
    ++numCons;
    return con;
}
```


getConnection

```
public synchronized Connection getConnection(boolean block, long timeout) throws SQLException {
if(free.isEmpty()){
    // None left, do we create more?
    if(maxCons <= 0 || numCons < maxCons) {
        AddConnection();
        }
        else if (block){
                wait();
                // Did anyone release a connection while we were waiting?
                if(free.isEmpty()){
                        If(maxCons <= 0 || numCons < maxCons) {
                                AddConnection();
                        }else {
                                throw new SQLException("Timeout waiting for a connection to be released");
                        }
                }
        }else {
        // No connections left and we don't wait for more.
throw new SQLException("Maximum number of allowed connections reached");
}
}
// If we get this far at least one connection is available.
Connection con;
synchronized (used){
con = (Connection) free.lastElement();
// Move this connection of the free list
free.removeElement(con);
used.addElement(con);
}
return con;
}
```

releaseConnection

```
public synchronized void releaseConnection(Connection con) throws SQLException{
        boolean reuseThisCon = reuseCons;
        if (used.cotains(con)){
                // Move this connection from the used list to the free list
                used.removeElement(con);
                numCons--;
        }else {
                con.close();
        }
        try {
                if(reuseThisCon) {
                        free.addElement(con);
                        numCons++;
                } else {
                        con.close();
                }
                // Wake up a thread waiting for a connection
        notify();
        } catch(SQLException e) {
                // Just to be sure
                try{
                        con.close();
                } catch(Exception e2) {
                        // We are expecting an SQLException hear
                }
        notify();
        }
}
```


closeAll connection

```
public synchronized void closeAll(){
    // Close unallocated connections
    Enumeration cons = ((Vector)free.clone()).elements();
    while(cons.hasMoreElements()) {
            Connection con = (Connection) cons.nextElement();
            Free.removeElement(con);
            NumCons--;
            try{
                    con.close();
            } catch(SQLException e) {
    // The Connection appears to be broken anyway, so we will ignore it
            }
    }
    cons = ((Vector)used.clone()).elements();
    while (cons.hasMoreElements()){
            Connection con = (Connection)cons.nextElement();
            Used.removeElement(con);
    }
}
```