Building Enterprise Database Applications

using Rational Rose, Java and Mr Architecture

A short course by Kade Hansson

Course Contents

- **UML, object-orientation and design patterns**
- **Java language and essential APIs**
- **Java GUI components and event model**
- **Java I/O and TCP/IP sockets**
- **JDBC, Servlets and JSPs**
- **Mr Architecture**

What is JDBC?

- Java Database Connectivity– allows Java to interact with Databases through ODBC
  - Provides an API for:
    - connecting to any type of database for which a JDBC driver is provided– **Factory Method** pattern
      e.g. `Connection c = DriverManager.getConnection("`*name*`")`
    - opening a transaction
      e.g. `c.setAutoCommit(false)`
    - executing SQL queries/updates/inserts– **Factory Method** pattern
      e.g. `ResultSet rs =(c.createStatement()).executeQuery("`*SQLStatement*`")`
    - committing a transaction, rolling back a transaction
      e.g. `c.commit()` or `c.rollback()`
    - Closing a connection
      e.g. `c.close()`

Result Sets

- Results from SQL querys are returned as *ResultSets*
  - *ResultSet*s have a built in cursor incremented through *next()*
  - If *next()* returns **`true`**, there is a row to read
    - Read it using *get*<Type>*(*columnID*)* or *get*<Type>*(*columnName*)*
      (where *Type* is the name of the type you want to read)
      - **Types are *Object* (which can return a database type implementation), *Byte, Short, Integer, Long, Float, Double, BigInteger, BigDecimal, String, Date (java.sql.Date), Time, Timestamp* (which all convert types where possible, except for *String* to *Clob* for some drivers)*, Clob, Blob***
      - **If you are not sure of the types, you can get a reference to the *ResultSetMetaData*–
        however, note that such metadata may not correspond directly to Java types for certain drivers**
      - **A *ResultSet* may be more efficient if you read the columns in sequential order and if you reference the columns by *columnID* (*columnID*s start at 1)**
  - Close the *ResultSet* to release resources allocated to it
    i.e. `rs.close()`

What is a Servlet?

- Java based web component, subclass of *Servlet*
- Managed by a *ServletContainer*
  - Web server extensions (to Apache or WebSphere, for example)
  - Common containers include Apache Tomcat, IBM JServ
- Generate dynamic content
  - Can be HTML
  - Can be binary
- Protocol inspecific– a level of abstraction above *ServerSocket*s
  - HTTP must be supported by all *ServletContainer*s
  - HTTPS is also commonoly supported

A Typical Servlet Scenario

1 A client accesses a web server (e.g. makes a HTTP GET request)

2 The request is received by the web server

3 The request is delegated to the server container

4 The servlet container chooses a servlet to delegate the request to

5 The servlet is loaded and initialized, if it isn't already
(*Servlet* life cycle mirrors *Applet*)

6 The servlet processes the data in the request (e.g. URL, URL-encoded parameters, POST or PUT content) and produces a suitable response

7 The servlet container relays the response to the web server

8 The web server relays the response to the client

9 The client receives a response from the web server

Comparing Servlets to Other Dynamic Content Techniques

- Generally faster than CGI scripts due to a lightweight process model
- Standard API which is supported by many web servers
- Leverage Java advantages
  - Easy to develop
  - Write once, run anywhere
  - Rich API set
- Support content filtering using *Filter* interface
  - Convert content types on the fly
  - Manipulate content

Servlets and HTTP Servlets

- A servlet has an *init()* and *destroy()* method (just like an applet)
- A servlet can implement *SingleThreadModel* if it wants the container to instantiate it multiple times during periods of high demand
- A *HTTPServlet* also provides one or more of
  - ***doGet()*** **for servicing GET requests**
  - ***doPut()*** **for servicing HTTP/1.1 PUT requests**
  - ***doPost()*** **for servicing POST requests**
  - ***doHead()*** **for servicing HEAD requests**
  - ***doDelete(), doOptions(), doTrace()*** **for servicing HTTP/1.1 DELETE, OPTIONS, TRACE**
- A non-HTTP(S) servlet may define *service()*

Web Applications

- A **web application** is a collection of Servlets, JSPs and static resources (GIF, JPEG, HTML, applet classes etc.) with high cohesion
  - Packaged as a **web archive** (extension **.war**)
  - Contains a `WEB-INF` directory (not served statically) containing:
    - A `web.xml` **deployment descriptor** containing initialization parameters
    - A `classes` directory containing class files contributing to library functionality
    - A `lib` directory containing required libraries in the form **Java archives** or **jars** (extension **.jar**)
    - Servlets may reside in `classes` or in jars under `lib`
- It is rooted at a specific path on a web server called the **context path**

An Example of a Deployment Descriptor

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
  <display-name>Simple Web Application</display-name>
  <context-param>
    <param-name>contextInitParam1</param-name>
    <param-value>Read this with ServletContext.getInitParameter()</param-value>
  </context-param>
  <servlet>
     <servlet-name>simple</servlet-name>
     <servlet-class>au.gov.tas.dpiwe.simple.SimpleServlet</servlet-class>
       <init-param>
         <param-name>servletInitParam1</param-name>
         <param-value>Read this with ServletConfig.getInitParameter()</param-value>
       </init-param>
  </servlet>
  <servlet-mapping>
     <servlet-name>simple</servlet-name>
     <url-pattern>*</url-pattern>
  </servlet-mapping>
</web-app>
```

Java Server Pages

- A mechanism or convention for constructing servlets used when
  - there is more I/O than processing
    (particularly the case when using middleware to generate mark-up)
  - there needs to be a separation of content layout from code
    - **Providing a custom tag library can allow a domain expert to lay out the content while the tag library does the hard processing work**
- A JSP contains content, but may also include
  - **JSP predefined tags (`jsp:forward`, `jsp:include`, `jsp:useBean` etc.)**
  - **Custom tags (e.g. `ms:format-field`, `ms:compare-fields`, `ms:choose-field`)**
  - **Servlet declarations delimited by `<%! %>` or `<jsp:declaration> </jsp:declaration>`**
  - **Java expressions which are converted into content, delimited by `<%= %>` or `<jsp:expression> </jsp:expression>`**
  - **Scriptlets (*service()* code fragments written in Java), delimited by `<% %>` or `<jsp:scriptlet> </jsp:scriptlet>`**

Tag Libraries

- A tag library consists of:
    - An XML **Tag Library Descriptor** (extension .tld)
        - **e.g.**

```
<taglib>
  <jsp-version>1.2</jsp-version>
  <short-name>ms</short-name>
  <tag>
    <name>format-field</name>
    <body-content>tagdependent</body-content>
    <tag-class>au.gov.tas.dpiwe.ms.tag</tag-class>
  </tag>
  ...
</taglib>
```

- A set of *Tag* classes (usually packaged as a jar) available to the web application
    - Each *Tag* class implements *doStartTag()*, *doEndTag()* and any further methods defined by the implemented *Tag* subinterface