PeopleSoft and J2EE: Integration Approaches

Al Wold
Arizona State University

Before We Begin...

- Sample code for the presentation:
- http://github.com/alwold/schedule-demo
- `svn co http://svn.github.com/alwold/schedule-demo.git schedule-demo`
- If you have Maven installed, you have everything you need to run it.

Why integrate with Java

- Better looking UI
- More authentication options
- Integration with other applications

Peoplesoft Database

- Standard database structure, except...
- No primary keys
- No foreign keys
- Effective dating

ASU: Historical approaches

- Ogate - app to proxy queries
- Oracle DBLink
- Replication with Integration Broker
- Direct SQL access

Why not Hibernate?

- Effective dating is tricky
- Non-standard relation mapping
  - Primary key is composed of identifying info plus effective date
- Hibernate is probably usable in reality, but difficult to set up

Spring JDBC

- Eliminates the (error prone) try-catch-finally block that is normally needed to do JDBC
- Simplifies the extraction of data from the result set

Standard JDBC

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;
try {
   conn = new InitialContext().lookup(“java:comp/env/jdbc/MyDataSource”).getConnection();
   ps = conn.prepareStatement(“SELECT 1 FROM dual”);
   rs = ps.executeQuery();
   if (rs.next()) {
      int one = rs.getInt(1);
   }
} catch (Exception e) {
   // do stuff
} finally {
   try {
      rs.close();
   } catch (Exception e) {}
   try {
      ps.close();
   } catch (Exception e) {}
   try {
      conn.close();
   } catch (Exception e) {}
}
```

Spring JDBC

```
public class MyDao extends GenericDaoSupport {
   public int getOne () {
      return getJdbcTemplate().queryForInt(“SELECT 1 FROM dual”);
   }
}
```

+

```
<bean id=”myDataSource” class=”org.springframework.jndi.JndiObjectFactoryBean”>
   <property name=”jndiName” value=”java:comp/env/jdbc/MyDataSource”/>
</bean>

<bean id=”myDao” class=”MyDao”>
   <property name=”dataSource” ref=”myDataSource”/>
</bean>
```

Intro to Spring

- Inversion of Control (IoC)
- Bean container
- Configuration file
- How to access in web app
- Integrations: tapestry

Tying it all together

- Devise the necessary query (PS developer?)
- Put file in classpath (e.g. in sql directory)
- Inject SQL file as resource with Spring and load via commons-io (IOUtils.toString)
- Write a RowMapper or ResultSetExtractor
- Call it from your app

Maven: 30 second overview

- Well defined directory structure
- pom.xml defines project parameters and dependencies
- Automatic download of dependencies
- Strict naming/versioning
- Concept of repositories
- Different packaging types (jar, war)
- Plugins: Jetty to run webapps (mvn jetty:run)
- mvn clean install

Schedule Demo: Database Tables

- App designer
  - PS_CLASS_TBL
  - PS_CLASS_INSTR
  - PS_NAMES
  - PS_CLASS_MTG_PAT
- ASU Catalog search view

Writing to PeopleSoft

- Writing to provided tables is risky, but you may be able to do this on custom tables
- Integration Broker
  - Requires development on both PS/ Java sides
  - Can be unreliable

Learn more

- Spring JDBC: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html
- Maven: http://maven.apache.org/
- Derby: http://db.apache.org/derby/
- Using JUnit with Spring: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/testing.html