# JSP 프로그래밍

(재)대덕인재개발원

# 12 데이터베이스 프로그래밍 기초

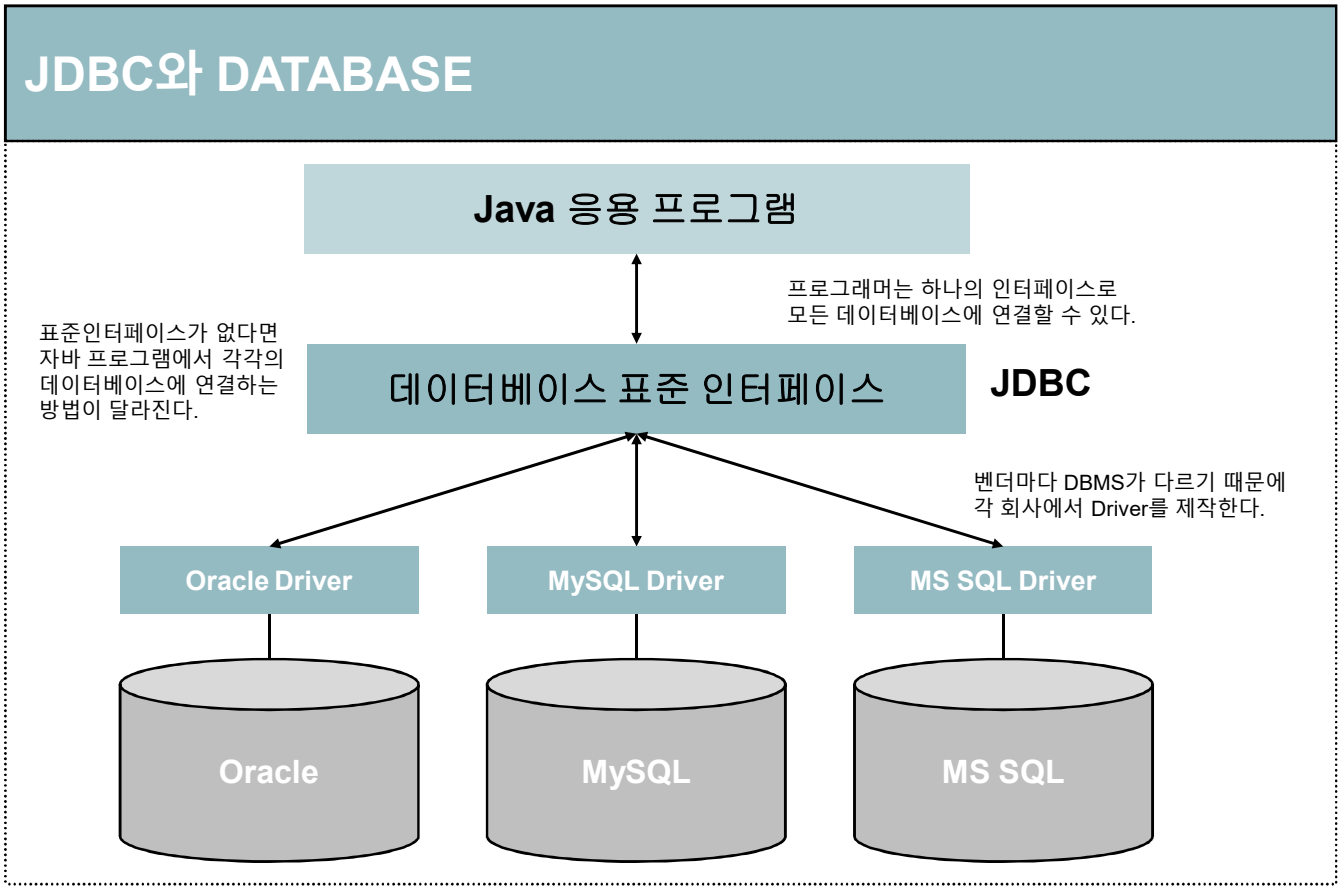
- 1. 데이터베이스 기초
- 2. 예제 실행을 위한 데이터베이스 생성
- 3. SQL 기초
- 4. JSP에서 JDBC 프로그래밍하기
- 5. JDBC에서 트랜잭션 처리
- 6. 커넥션 풀

# 12.4 JSP에서 JDBC 프로그래밍하기

# 12.4.1 JDBC의 구조



- JDBC(Java Database Connectivity)
  - JDBC란 데이터베이스에 연결 및 작업을 하기 위한 자바 표준 인터페이스(API)이다.



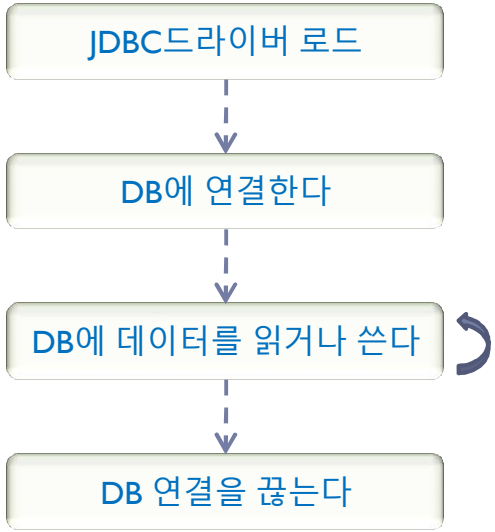
# 12.4.2 JDBC 드라이버 준비하기

- MySQL JDBC 드라이버 Connector/J
  - http://dev.mysql.com/downloads
  - Connector/J 5.x라는 이름으로 등록되어 있으며 zip 또는 tar 파일 형태로 묶여 있다.
  - 압축해제디렉터리₩**mysql-connector-java-5.x.x-bin.jar**
- 오라클 JDBC 드라이버
  - 오라클 설치 폴더에서 가져오기
    - C:₩oraclexe₩app₩oracle₩product₩10.2.0₩server₩jdbc₩lib₩ ojdbc14.jar
  - 오라클 다운로드 사이트에서 가져오기
    - http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html
- MS SQL 서버 JDBC 드라이버
  - http://www.microsoft.com/downloads/

# 12.4.3 JDBC 프로그래밍의 코딩 스타일



- 1. 데이터 베이스 연결을 위한 드라이버 생성
  - Class.forName("com.mysql.jdbc.Driver")으로 드라이버 로딩
  - Class.forName("oracle.jdbc.driver.OracleDriver")으로 드라이버 로딩
- 2.연결을 관리하는 Connection 객체 생성
  - DriverManager의 getConnection()을 이용해서 Connection 객체 생성
- 3. SQL 문 수행.
  - 작업을 처리할 Statement, PrepareStatement, CallableStatement 객체 생성
  - 쿼리 실행 및 결과 사용
- 4. 연결 종료
  - Connection close()




# 12.4.4 JDBC 주요 클래스

# 12.4.4.1 Connection 인터페이스

- DBMS와 연결을 관리하는 클래스

| 반환형 | 메소드 | 설명 |
|---|---|---|
| void | close() | Connection 객체를 해제한다. |
| | commit() | 트랜잭션으로 설정된 모든 자원을 커밋한다. |
| Statement | createStatement() | SQL문을 전송할 수 있는 Statement 객체를 생성한다. |
| | createStatement (int resultSetType, int resultSetConcurrency) | 매개변수로 SQL문을 전송할 수 있는 Statement 객체를 생성한다. 매개변수값을 어떻게 설정하는냐 따라 Statement 객체의 기능이 달라진다. |
| boolean | getAutoCommit() | Connection 객체의 현재 auto-commit 상태를 반환한다. |
| CallableStatement | prepareCall(String sql) | SQL문 전송과 Store Procedure를 호출할 수 있는 CallableStatement 객체를 생성한다. |
| | prepareCall(String sql, int resultSetType, int resultSetConcurrency) | 매개변수로 CallableStatement 객체를 생성한다. 매개변수값을 어떻게 설정하느냐 따라 CallableStatement 객체의 기능이 달라진다. |

# 12.4.4.1 Connection 인터페이스(계속)

- DBMS와 연결을 관리하는 클래스

| 반환형 | 메소드 | 설명 |
|---|---|---|
| PreparedStatement | prepareStatement(String sql) | SQL문을 전송할 수 있는 PreparedStatement 객체를 생성한다. |
| | prepareStatement(String sql, int resultSetType, int resultSetConcurrency) | 매개변수로 PreparedStatement 객체를 생성한다.<br>매개변수값을 어떻게 설정하느냐 따라 PreparedStatement 객체의 기능이 달라진다. |
| void | rollback() | 현재 트랜잭션에 설정된 모든 변화를 되돌린다. |
| | rollback(Savepoint savepoint) | Savepoint로 설정된 이후의 모든 변화를 되돌린다. |
| Savepoint | setSavepoint(String name) | 현재 트랜잭션에서 name으로 Savepoint를 설정한다. |

# 12.4.4.2 Connection 방법

- 1. 드라이버 로딩
  - Class.forName("oracle.jdbc.driver.OracleDriver");
- 2. java.sql.DriverManager 클래스를 이용해서 Connection 객체 생성
  - String url = "jdbc:oracle:thin:@localhost:1521:xe";
  - String url = "jdbc:mysql://localhost:3306/dbname";
  - String user = "user";
  - String password = "password";
  - Connection conn = DriverManager.getConnection(url, user, password);
- 3. 작업
  - 쿼리와 명령 처리
- 4. Connection 닫기
  - conn.close();

# 12.4.4.3 Statement

- Statement 의 종류
  - Statement
  - PreparedStatement
  - CallableStatement
- Statement
  - 정적쿼리 처리에 유리하다.
- PreparedStatement
  - 객체를 생성하면서 기준 Query를 생성한다.
  - setter()메소드를 가지고 조건만 바꾸면서 동적 쿼리를 수행한다.
- Statement 의 의미
  - 자바에서 사용되는 3가지 종류의 스테이트먼트들은 데이터베이스로 쿼리를 담아서 보내는 그릇 정도로 생각하면 된다. 즉 스테이트먼트에 쿼리를 실어 데이터베이스로 보내버리면 데이터베이스에서 처리되는 것이다. 이 때 한번 사용하고 버리는 그릇은 Statement이며, 재사용 가능한 그릇은 PreparedStatement이다.

- CallableStatement
  - 데이터베이스 내의 스토어드 프로시저(Stored Procedure)를 호출할 수 있는 스테이트먼트

# 12.4.4.4 Statement 인터페이스

- SQL문을 전송을 담당하는 인터페이스이다.
- Statement 객체는 Connection 인터페이스의 createStatement() 메서드를 사용하여 얻어 올 수 있다.
- executeQuery(String sql) : select SQL 구문일 경우
- executeUpdate(String sql) : insert, update, delete SQL 구문일 경우
- execute(String sql) : SQL문을 알지 못하는 경우

| 반환형 | 메소드 | 설명 |
|---|---|---|
| void | addBatch(String sql) | Statement 객체에 SQL문을 추가한다. 이 메서드를 이용해서 SQL의 일괄처리를 할 수 있다. |
| | clearBatch() | Statement 객체에 모든 SQL문을 비운다. |
| | close() | Statement 객체를 해제한다. |
| boolean | execute(String sql) | 매개변수인 SQL문을 수행한다. 만약, 수행한 결과가 ResultSet 객체를 반환하면 true, 어떠한 결과도 없거나, 갱신된 숫자를 반환하면 false를 반환한다. |
| int[ ] | executeBatch() | Statement 객체에 추가된 모든 SQL문을 일괄처리한다. 일괄처리된 각각의 SQL문에 대한 결과값을 int[ ]로 반환한다. |
| ResultSet | executeQuery (String sql) | 매개변수인 SQL문을 수행하고 ResultSet 객체를 반환한다. |
| int | executeUpdate (String sql) | 매개변수인 SQL문을 수행한다. SQL문은 INSERT문, UPDATE문, CREATE문, DROP문 등을 사용한다. |
| ResultSet | getResultSet() | ResultSet 객체를 반환한다. |

# 12.4.4.5 ResultSet 인터페이스

- ResultSet은 SQL문에 대한 결과를 처리할 수 있는 객체이다.
- Statement 인터페이스의 executeQuery() 메서드를 실행한 결과로 ResultSet 객체를 리턴 받는다.
- 모든 데이터를 한번에 가져올 수 없기 때문에 cursor의 개념을 가지고 있다.
- cursor란 ResultSet 객체가 가져올 수 있는 행을 지정해 준다.
- 처음 커서의 위치는 결과물(필드)에 위치하지 않기 때문에 cusror를 이동해야 한다.
- 커서를 이동하는 메서드가 ResultSet 의 next() 메서드이다.
- next() 메서드의 리턴 타입은 boolean 인데 이는 다음 행의 결과물(필드)이 있으면 true, 없으면 false를 리턴 한다.
- ResultSet 객체가 결과물(필드)을 가져올 수 있는 행으로 이동이 되었다면 이제는 실제 결과물(필드)을 가져와야 한다.
- ResultSet 인터페이스에는 결과물(필드)을 가져오는 수많은 메서드(getXXX())를 제공한다.
- getXXX() 메서드는 oracle의 자료형 타입에 따라 달라지게 된다.

# 12.4.4.5 ResultSet 인터페이스

| 반환형 | 메소드 | 설명 |
|---|---|---|
| boolean | absolute(int row) | ResultSet 객체에서 매개변수 row로 커서를 이동한다. 만약, 매개변수 row로 커서를 이동할 수 있으면 true, 그렇지 않으면 false를 반환한다. |
| void | afterLast() | ResultSet 객체에서 커서를 마지막 row 다음으로 이동한다. |
| | beforeFirst() | ResultSet 객체에서 커서를 처음 row 이전으로 이동한다. |
| boolean | last() | ResultSet 객체에서 커서를 마지막 row 로 이동한다. 만약, ResultSet에 row가 있다면 true, 그렇지 않으면 false를 반환한다. |
| | next() | ResultSet 객체에서 현재 커서에서 다음 row로 커서를 이동한다. 만약, ResultSet에 다음 row가 있다면 true, 그렇지 않으면 false를 반환한다. |
| | previous() | ResultSet 객체에서 현재 커서에서 이전 row로 커서를 이동한다. 만약, ResultSet에 다음 row가 있다면 true, 그렇지 않으면 false를 반환한다. |
| void | close() | ResultSet 객체를 해체한다. |

# 12.4.4.6 ResultSet 인터페이스

| 반환형 | 메소드 | 설명 |
|---|---|---|
| boolean | first() | ResultSet 객체에서 커서를 처음 row로 이동한다. 만약 ResultSet에 row가 있다면 true, 그렇지 않으면 false를 반환한다. |
| InputStream | getBinaryStream (int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 InputStream으로 반환한다. |
| | getBinaryStream (String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 InputStream으로 반환한다. |
| Blob | getBlob (int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 Blob으로 반환한다. |
| | getBlob (String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 Blob으로 반환한다. |
| byte | getByte (int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 byte으로 반환한다. |
| | getByte (String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 byte으로 반환한다. |

# 12.4.4.7 ResultSet 인터페이스

| 반환형 | 메소드 | 설명 |
|---|---|---|
| Clob | getClob<br>(int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 Clob으로 반환한다. |
| | getClob<br>(String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 Clob으로 반환한다. |
| double | getDouble<br>(int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 double으로 반환한다. |
| | getDouble<br>(String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 double으로 반환한다. |
| int | getInt<br>(int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 int으로 반환한다. |
| | getInt<br>(String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 int으로 반환한다. |
| String | getString<br>(int columnIndex) | ResultSet 객체의 현재 row에 있는 columnIndex의 값을 String으로 반환한다. |
| | getString<br>(String columnName) | ResultSet 객체의 현재 row에 있는 columnName의 값을 String으로 반환한다. |

# 12.4.4.8 PreparedStatement 인터페이스

- PreparedStatement는 SQL문을 작성할 때 컬럼 값을 실제로 지정하지 않고, 변수 처리 함으로서 DBMS을 효율적으로 사용한다.
- PreparedStatement의 SQL문은 SQL문의 구조는 같은데 조건이 수시로 변할 때 조건의 변수처리를 "?" 하는데 이를 바인딩 변수라 한다.
- 바인딩 변수는 반드시 컬럼 명이 아닌 컬럼 값이 와야 한다는 것이다.
- 바인딩 변수의 순서는 "?" 의 개수에 의해 결정이 되는데 시작 번호는 1 부터 시작하게 된다.
- 바인딩 변수에 값을 저장하는 메서드는 오라클의 컬럼 타입에 따라 지정해 주면 된다. 전에 ResultSet의 getXXX() 메서드와 유사하게 PreparedStatement 인터페이스에는 바인딩 변수에 값을 저장하는 setXXX() 메서드를 제공하고 있다.

# 12.4.5.1 Connection 예제

```
//1. 드라이버 검색
try{
		Class.forName("oracle.jdbc.driver.OracleDriver");
}catch(ClassNotFoundException e){
		System.out.println("드라이버 검색 실패");
}

// 2. Connection 객체 생성
Connection conn = null;
String url = "jdbc:oracle:thin:@localhost:1521:xe";
String user = "ddit";
String pass = "admin";

try{
		conn = DriverManager.getConnection(url, user, pass);
}catch(SQLException e){
		System.out.println("시스템 연결 실패");
}
```

# 12.4.5.2 Statement 예제

```
// 3. 쿼리 실행 문장 작성
Statement stmt = null;
String query = "";
try{
		query = "SELECT * FROM MEMBER";
		stmt = conn.createStatement();
}catch(SQLException e){
		System.out.println("Statement failed...");
}
// 4.1 결과 얻기
ResultSet rs = null;
try{
		rs = stmt.executeQuery(query);
		while(rs.next()){
			System.out.printf("%d  %s  %s  %s  %s  %s  %n",
					rs.getInt(1), rs.getString("name"), rs.getString(3),
					rs.getString("phone"), rs.getString("Cell_phone"),
rs.getString("address"));
		}
}catch(SQLException e){
		System.out.println("Query failed...");
}
```

# 12.4.5.3 PreparedStatement 예제(입력)

```
PreparedStatement pstmt = null;    // PreparedStatement 객체 생성
String prequery = "";
try{
        prequery = "INSERT INTO MEMBER(NUM, NAME, PHONE, CELL_PHONE,
ADDRESS)                                                VALUES(?, ?, ?, ?, ?) “;

        pstmt = conn.prepareStatement(prequery);
        pstmt.setInt(1, 5);
        pstmt.setString(2, "San");
        pstmt.setString(3, "010-231-4568");
        pstmt.setString(4, "010-2365-4569");
        pstmt.setString(5, "대전시 중구 대흥동 500");

}catch(SQLException e){
        System.out.println("PrepareStatements failed..");
}

// 4.2 PrepareStatement Execute..
try{
        pstmt.executeUpdate();
        System.out.println("실행 성공");
}catch(SQLException e){
        System.out.println("Insert failed...");
}
```

# 12.4.5.4 PreparedStatement 예제(검색)

```
// 4.3 Select data using PreapareStatement
prequery = "SELECT * FROM MEMBER WHERE ADDRESS LIKE ?";

try{
		pstmt = conn.prepareStatement(prequery);
		pstmt.setString(1, "서%");
		rs = pstmt.executeQuery();
		System.out.println();

		while(rs.next()){		// 검색 결과 출력
			System.out.printf("%d  %s  %s  %s  %s  %s  %n",
				rs.getInt(1), rs.getString("name"), rs.getString(3),
				rs.getString("phone"), rs.getString("Cell_phone"),
rs.getString("address"));
		}

}catch(SQLException e){}

// 5. Connection Close
try{
		if(rs != null) rs.close();
		if(stmt != null) stmt.close();
		if(pstmt != null) pstmt.close();
		if(conn != null) conn.close();
}catch(SQLException e){
		System.out.println("Connection Close failed...");
}
```

# 12.4.6 웹어플리케이션 구동시 JDBC 드라이버 로딩하기

- 드라이버 로딩 클래스 파일 작성

```
package com.san.jdbc.DBLoader;

import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import java.util.StringTokenizer;

public class DBLoader extends HttpServlet{

  public void init(ServletConfig config) throws ServletException{
    try{
      String drivers = config.getInitParameter("jdbcdriver");
      StringTokenizer st = new StringTokenizer(drivers, ",");
      while(st.hasMoreTokens()){
        String jdbcDriver = st.nextToken();
        Class.forName(jdbcDriver);
      }

    }catch(Exception e){
        throw new ServletException(e);
    }
  }
}
```

# 12.4.6 웹어플리케이션 구동시 JDBC 드라이버 로딩하기

- 서블릿 클래스 실행을 위한 web.xml 설정 정보 추가
  - 웹 어플리케이션이 시작할 때 DBLoader서블릿 클래스가 자동으로 시작되고 init() 메서드가 호출된다.

```
<servlet>
  <servlet-name>DBLoader</servlet-name>
  <servlet-class>com.san.jdbc.loader.DBLoader</servlet-class>
  <init-param>
    <param-name>jdbcdriver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

# 12.5 JDBC에서 트랜잭션 처리

- JDBC API를 이용한 트랜잭션 처리하기

```
try{
        conn = DriverManager.getConnection(....);
        // 트랜잭션 시작
        conn.setAutoCommit(false);

        ...// 쿼리 실행
        ...// 쿼리 실행

        // 트랜잭션 커밋
        conn.commit();

}catch(SQLException e){
        if(conn != null){

            // 트랜잭션 롤백
            conn.rollback();

        }
}finally{
        if(conn != null){
                try{
                        conn.close();
                }catch(SQLException e){}

        }
}
```

정상처리 시 커밋수행

에러발생 시 롤백수행

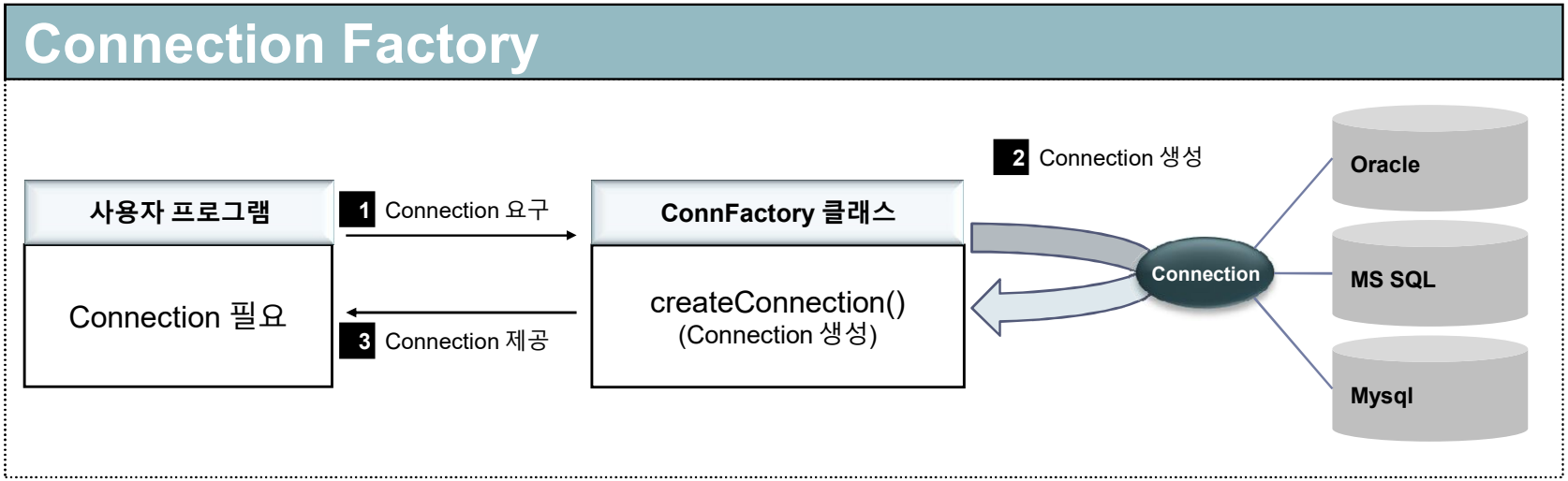
# 12.6 커넥션 풀

# 12.6.1 커넥션 풀이란



- 커넥션 풀 기법이란 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때에 커넥션을 풀에서 가져다 쓰고 다시 풀에 반환하는 기법을 말한다.
- 커넥션 풀의 특징
  - 풀 속에 미리 커넥션이 생성되어 있기 때문에 커넥션을 생성하는데 드는 연결 시간이 소비되지 않는다.
  - 커넥션을 계속해서 재사용하기 때문에 생성되는 커넥션 수가 많지 않다.

# 12.6.1 Connection Factory 기법



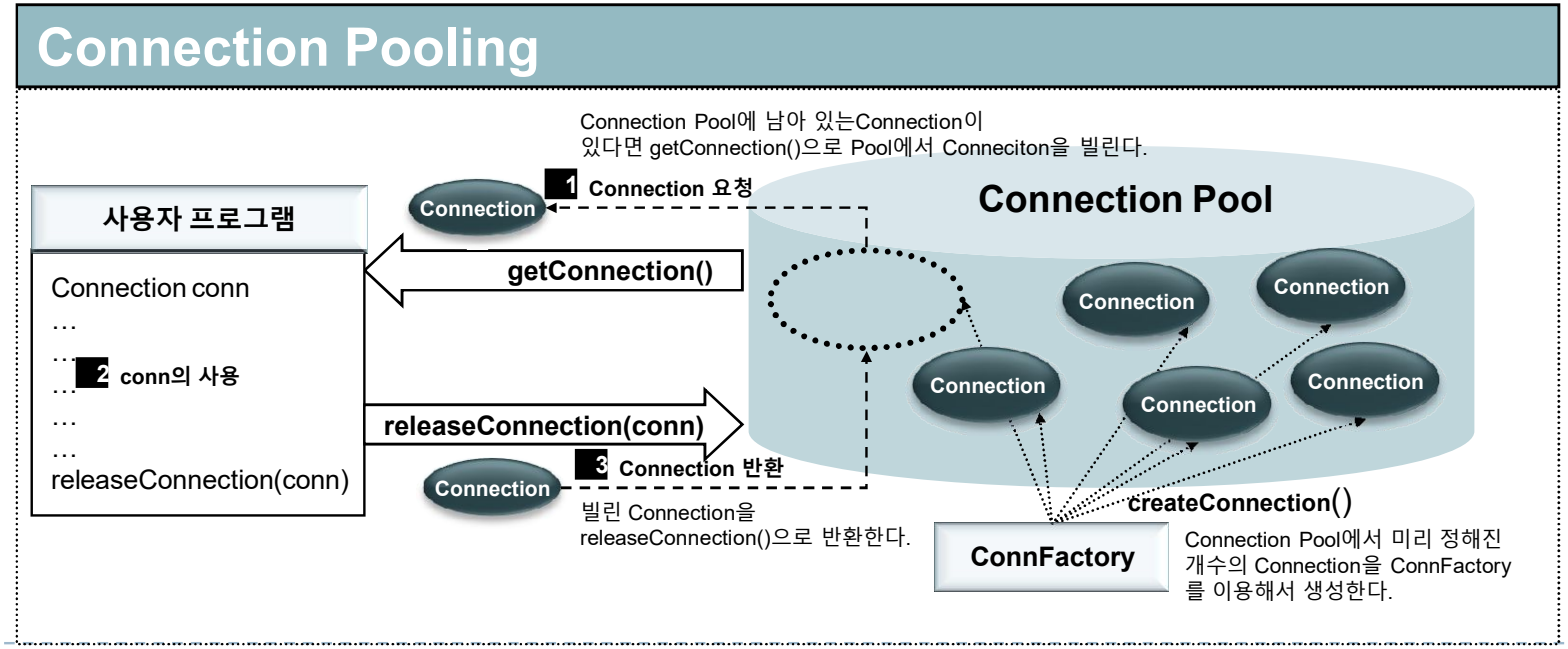

- ConnFactory의 생성자와 멤버
  - private ConnFactory(){ }
  - public static ConnFactory getDefaultFactory(){ };
  - public Connection createConnection(){ };
- 싱글톤(Singleton) 기법으로 ConnFactory 얻어내기
  - ConnFactory connFac = ConnFactory.getDefaultFactory();
- ConnFactory로부터 Connection 얻어내기
  - Connection conn = connFac.createConnection();

- 컨넥션 공장(Connection Factory)을 구현하기 위한 ConnFactory 클래스

```
public class ConnFactory {
        private static ConnFactory connFactory = new ConnFactory();
        private ConnFactory(){};
        public static ConnFactory getDefaultFactory(){
            if(connFactory == null){
                connFactory = new ConnFactory();
            }
            return connFactory;
        }
        public Connection createConnection(){...}
}
```

# 12.6.2 Connection Pooling

- Connection Pool
  - 미리 컨넥션을 일정 수만큼 생성시킨 뒤 컨넥션을 빌려주고 다시 반환 받는 형식으로 컨넥션을 관리한다.
  - 컨넥션 풀에서 필요한 만큼 Connection을 미리 생성
  - Connection이 필요할 때 컨넥션 풀에서 빌려서 사용
  - 사용이 끝난 다음에는 Connection을 컨넥션 풀에 반환
- Connection Pooling 기법의 장점
  - Connection Pooling 기법을 이용하면 데이터베이스에 연결하는 시간을 절약할 수 있다.



# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 자카르타 프로젝트의 DBCP API를 사용시 절차
  - 1. DBCP 관련 jar 파일 및 JDBC 드라이버 jar 파일 설치하기
  - 2. 커넥션 풀 관련 설정 파일 초기화하기
  - 3. 커넥션 풀 관련 드라이버 로딩하기
  - 4. 커넥션 풀로부터 커넥션 사용하기

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 1. 필요한 jar 파일 복사하기
  - Commons-DBCP API 관련 jar파일
    - commons-dbcp-1.2.2.jar
  - Commons-DBCP API 가 사용하는 Commons-Pool API의 jar 파일
    - commons-pool-1.4.jar
  - http://commons.apache.org 에서 다운로드.
  - 위의 두 jar 파일을 WEB-INF/lib 디렉토리에 복사한다.

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 2. 설정 파일 작성 및 위치(WEB-INF/classes/pool.jocl)
  - DBCP API는 웹 어플리케이션의 클래스 패스로 부터 JOCL 설정 파일을 검색하므로 WEB-INF/classes 에 위치 시킨다.

```
<object class="org.apache.commons.dbcp.PoolableConnectionFactory"
  xmlns="http://apache.org/xml/xmlns/jakarta/commons/jocl">

  <!-- the first argument is the ConnectionFactory -->
  <object class="org.apache.commons.dbcp.DriverManagerConnectionFactory" >
    <string value="jdbc:oracle:thin:@127.0.0.1:1521:XE" />
    <string value="user">
    <string value="password">
  </object>

  <!-- the next argument is the ObjectPool -->
  <object class="org.apache.commons.pool.impl.GenericObjectPool" >
    <object class="org.apache.commons.pool.PoolableObjectFactory" null="true" />
  </object>

  <!-- the next argument is the KeyedObjectPoolFactory -->
  <object class="org.apache.commons.pool.KeyedObjectPoolFactory" null="true" />

  <string value="SELECT COUNT(*) FROM DUAL" />      <!-- validation query -->
  <boolean value="false" />   <!-- default read only -->
  <boolean value="true" />    <!-- default auto commit -->
</object>
```

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 3. 컨넥션 풀 초기화 하기(클래스 파일 작성)

```
package com.san.jdbc.DBLoader;

import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import java.util.StringTokenizer;

public class DBLoader extends HttpServlet{

  public void init(ServletConfig config) throws ServletException{
    try{
      String drivers = config.getInitParameter("jdbcdriver");
      StringTokenizer st = new StringTokenizer(drivers, ",");
      while(st.hasMoreTokens()){
        String jdbcDriver = st.nextToken();
        Class.forName(jdbcDriver);
      }

      Class.forName("org.apache.commons.dbcp.PoolingDriver");

    }catch(Exception e){
        throw new ServletException(e);
    }
  }
}
```

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 3. 컨넥션 풀 초기화 하기(web.xml 설정 정보 추가)
  - 웹 어플리케이션이 시작할 때 DBLoader서블릿 클래스가 자동으로 시작되고 init() 메서드가 호출된다.

```
<servlet>
  <servlet-name>DBLoader</servlet-name>
  <servlet-class>com.san.jdbc.loader.DBLoader</servlet-class>
  <init-param>
    <param-name>jdbcdriver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 4. 커넥션 풀로부터 커넥션 사용하기
  - jdbc:apache:commons:dbcp:/[풀이름]

```
Connection conn = null;
...

try{

  String url = "jdbc:apache:commons:dbcp:/pool";
  conn = DriverManager.getConnection(url);
  ...

}finally{
  ...
  if(conn != null){
    try{
      conn.close();
    }catch(SQLException e){}
  }
}
```

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 5. 커넥션 풀 속성 설명
  - 최대 커넥션 개수, 최소 유휴 커넥션 개수, 최대 유휴 커넥션 개수, 유휴 커넥션 검사 여부 등의 속성을 지정할 수 있다

```
<!-- the next argument is the ObjectPool -->
 <object class="org.apache.commons.pool.impl.GenericObjectPool" >
   <object class="org.apache.commons.pool.PoolableObjectFactory" null="true" />

     <int value="10"/> <!-- max active -->
     <byte value="1"/> <!-- when exhausted action, 0 = fail, 1 = block, 2 = grow -->
     <long value="2000"/> <!-- max wait -->
     <int value="10"/> <!-- max idle -->
     <boolean value="false"/> <!-- test on borrow -->
     <boolean value="false"/> <!-- test on return -->
     <long value="10000"/> <!-- time between eviction runs -->
     <int value="5"/> <!-- number of connections to test per eviction run -->
     <long value="5000"/> <!-- min evictable idle time -->
     <boolean value="true"/> <!-- test while idle -->

 </object>
```

# 12.6.2 DBCP를 이용해서 커넥션 풀 사용하기

- 5. 커넥션 풀 속성 설명

| 속 성 | 설 명 |
|---|---|
| maxActive | 커넥션 풀이 제공할 최대 커넥션 개수 |
| whenExhaustedAction | 커넥션 풀에서 가져올 수 잇는 커넥션이 없을 때 어떻게 동작할지를 지정한다. 1일 경우 maxWait 속성에서 지정한 시간만큼 커넥션을 구할 때까지 기다리며, 0일 경우 에러를 발생시킨다. 2일 경우에는 일시적으로 커넥션을 생성해서 사용한다. |
| maxWait | whenExhaustedAction 속성의 값이 1일 때 사용되는 대기 시간. 단위는 1/1000초이며, 0보다 작을 경우 무한히 대기한다. |
| maxIdle | 사용되지 않고 풀에 저장될 수 있는 최대 커넥션 개수. 음수일 경우 제한이 없다. |
| minIdle | 사용되지 않고 풀에 저장될 수 있는 최소 커넥션 개수 |
| testOnBorrow | true일 경우 커넥션 풀에서 커넥션을 가져올 때 커넥션이 유효한지의 여부를 검사한다. |
| testOnReturn | true일 경우 커넥션 풀에 커넥션을 반환할 때 커넥션이 유효한지의 여부를 검사한다. |
| timeBetweenEvictionRunsMills | 사용되지 않은 커넥션을 추출하는 쓰레드의 실행 주기를 지정한다. 양수가 아닐 경우 실행되지 않는다. 단위는 1/1000초이다. |
| numTestPerEvictionRun | 사용되지 않는 커넥션을 몇 개 검사할지 지정한다. |
| minEvictableIdleTimeMills | 사용되지 않는 커넥션을 추출할 때 이 속성에서 지정한 시간 이상 비활성화 상태인 커넥션만 추출한다. 양수가 아닌 경우 비활성화된 시간으로는 풀에서 제거되지 않는다. 시간 단위는 1/1000최이다. |
| testWhileIdle | true일 경우 비활성화 커넥션을 추출할 때 커넥션이 유효한지의 여부를 검사해서 유효하지 않은 커넥션은 풀에서 제거한다. |

# *. JNDI 방식 커넥션 생성하기

- server.xml 에 리소스 등록
  - <Resource auth="*Container*"
    - *driverClassName="oracle.jdbc.driver.OracleDriver"*
    - *maxActive="100" maxIdle="30"*
    - *maxWait="10000"*
    - *name="jdbc/mainDB"*
    - *password="java"*
    - *type="javax.sql.DataSource“*
    - *url="jdbc:oracle:thin:@127.0.0.1:1521:XE“*
    - *username="pc17"/>*
- context.xml 수정
  - <ResourceLink
    - globalName="*jdbc/mainDB*"
    - *name="jdbc/mainDB"*
    - *type="javax.sql.DataSource*" />
- connection 생성 코드
  - Context ctx = **new InitialContext();**
  - DataSource dataSource = (DataSource) ctx.lookup("java:/comp/env/jdbc/mainDB");
  - conn = dataSource.getConnection();