Chapter 13

커넥션 풀 (Connection Pool)

커넥션 풀의 개요

커넥션 풀(Connection Pool)의 개요

■ 데이터베이스 연동의 문제점

- 데이터베이스에 연결하고자하는 모든 문서는 접속 과정을 반드시 거쳐야 함
  - 어플리케이션은 많은 문서들로 구성됨
  - 데이터베이스에 접속해야 하는 모든 문서는 JDBC를 이용해 데이터베이스에 연결하는 소스를 포함하고 있어야 함

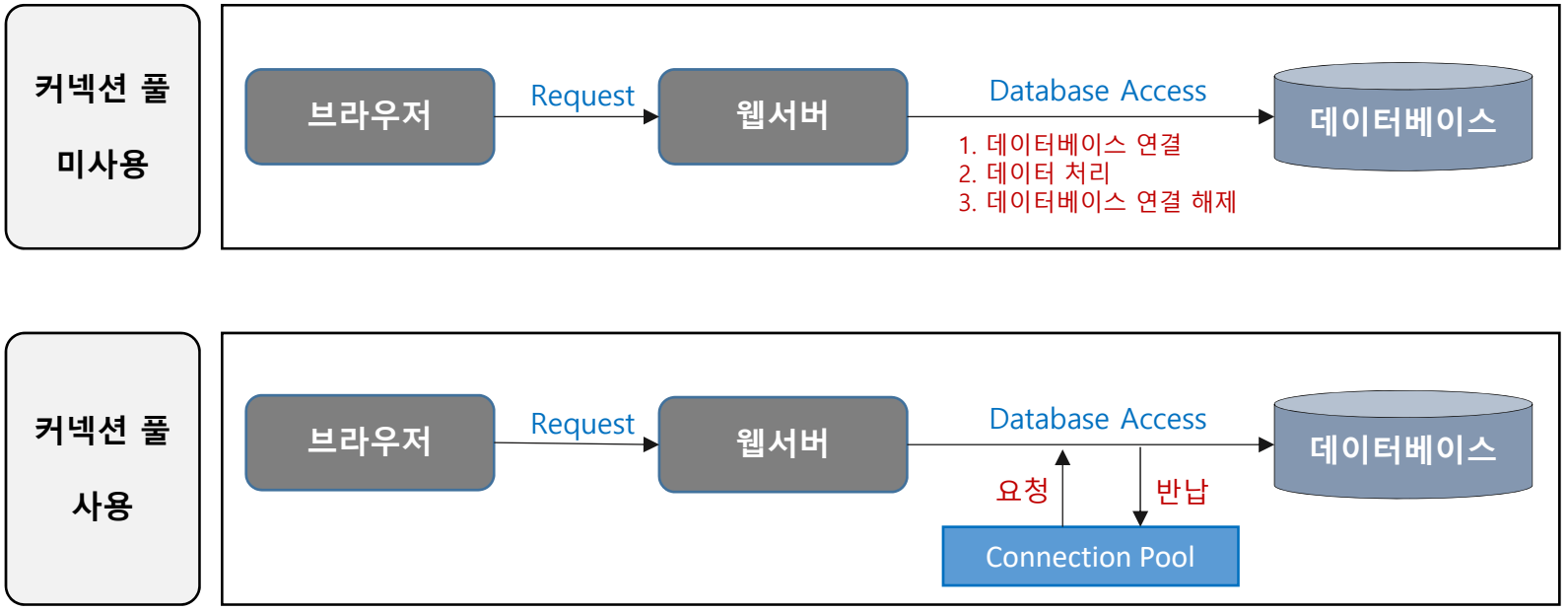
- 동시에 접속 수가 많은 대형 사이트의 경우 많은 오버헤드가 발생됨
  - 데이터베이스에 접속하기 위해서는 많은 시간이 소요됨
  - 접속 수가 많은 대형 사이트의 경우, 접속이 이루어질 때마다 Connection이 생성되어야 함
  - 많은 문서들이 데이터베이스 접속을 위해 동일한 코드를 포함하고 있어야 함

- 이러한 문제를 보완하기 위해 컨넥션풀(ConnectionPool)을 사용함

커넥션 풀(Connection Pool)의 개요

■ 커넥션 풀의 동작

- 데이터베이스 연결을 위한 Connection을 생성해 Pool에 저장
  - 데이터베이스에 대한 접근이 필요한 경우 미리 정의되어 있는 connection을 사용
  - 사용 후 Connection을 Pool에 반납



커넥션 풀(Connection Pool)의 개요

■ 커넥션 풀의 사용

- **JNDI를 사용**
  - ▸ JNDI를 사용해 JNDI Naming 서비스에 Connection을 등록
- **InitialContext 객체를 생성**
  - ▸ JNDI를 사용해 ConnectionPool 객체에 접근하기 위해서는 InitialContext 객체를 생성해야 함
    - InitialContext 객체는 JNDI를 이용해 객체를 bind하거나 lookup하기 위해 사용하는 객체임
- **DataSource 객체 생성**
  - ▸ DataSource 객체는 JNDI를 사용해 커넥션 풀의 Connection을 관리하기 위한 객체
    - DataSource 객체를 통해서 필요한 Connection을 획득, 반납 등의 작업을 수행
  - ▸ DataSource 객체는 InitialContext 객체에 lookup() 메서드를 사용해 생성

JNDI (Java Naming and Directory Interface)
- 디렉터리 서비스에서 제공하는 데이터 및 객체를 발견하고 참고하기 위한 자바 API
- 필요한 자원을 키-값(key-value) 쌍으로 저장한 후 필요할 때 키를 이용해 값을 얻는 방법

Naming & Directory 서비스
- Naming & Directory 서버에 서비스하고자 하는 자원을 특정 이름으로 연결하여 등록
- 해당 자원을 활용할 경우 Naming & Directory 서버에 접근해 이름으로 검색하여 사용할 수 있음
- DNS 서버는 Naming & Directory 서비스의 대표적인 예

커넥션 풀(Connection Pool)의 개요

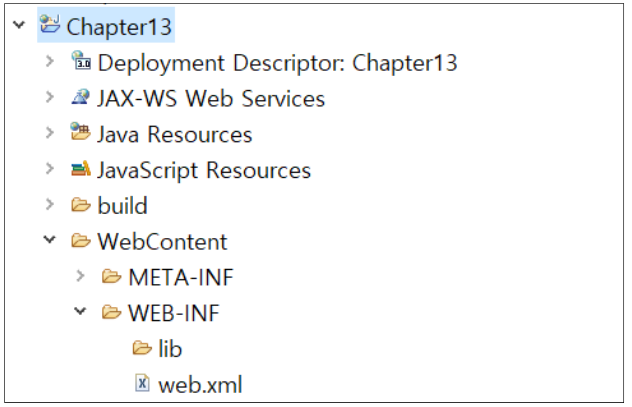
- DataSource를 이용하는 과정
  - ① InitialContext 객체를 생성
  - ② InitialContext 객체의 lookup( ) 메소드를 통해 DataSource 객체를 생성
  - ③ DataSource 객체의 getConnection( ) 메소드를 사용해 Connection 객체를 획득
  - ④ Connection 객체를 통한 DBMS 작업을 수행
  - ⑤ 모든 작업이 끝나면 DataSource 객체를 통해서 Connection Pool에 Connection을 반납

커넥션 풀 설정과 실습

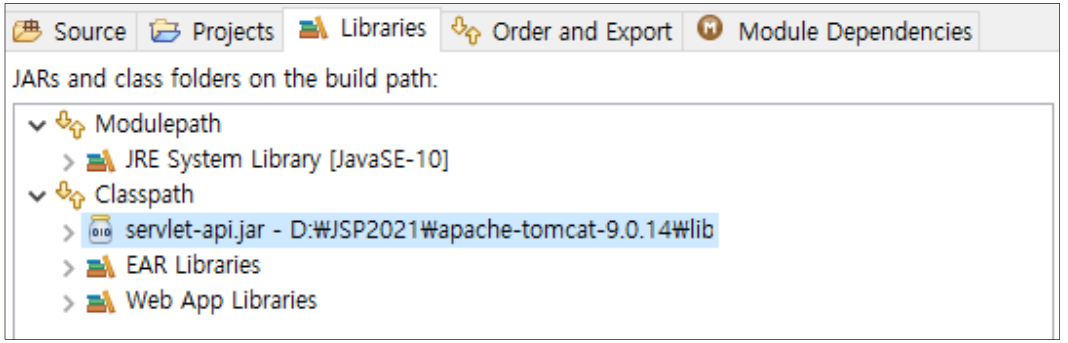
프로젝트 생성

■ 프로젝트 생성

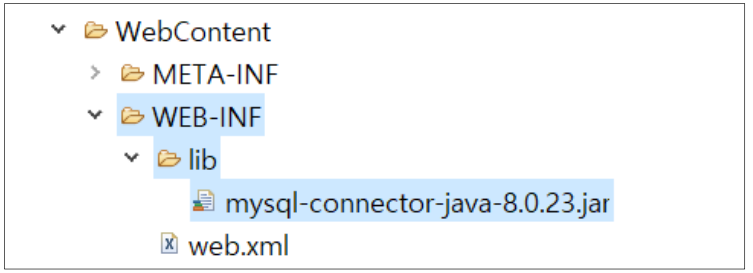
- 프로젝트 이름 : chapter13
  - web.xml 파일은 반드시 생성해야 함



- servlet-api 라이브러리 추가



- Connector 추가
  - 이전 프로젝트에서 복사 가능



DBCP

■ DBCP(DataBase Connection Pool)

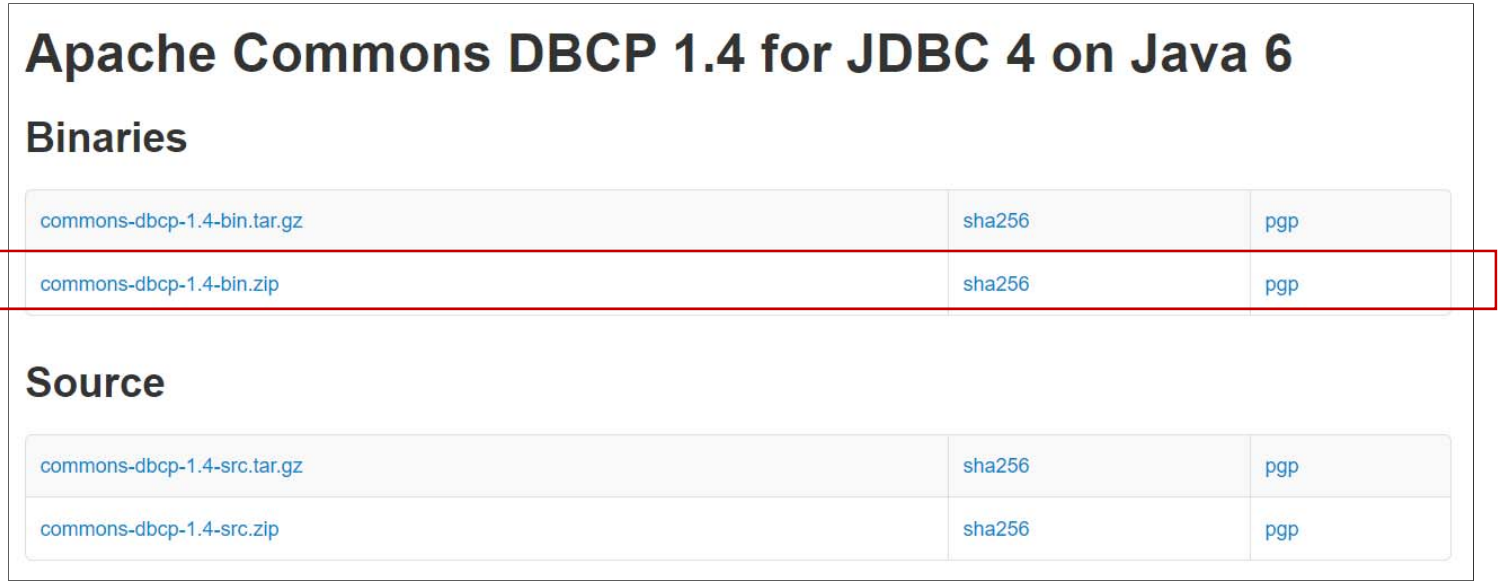
- 데이터베이스와 애플리케이션을 효율적으로 연결하는 커넥션 풀 라이브러리
  - WAS가 실행되면서 미리 일정량의 DB Connection 객체를 생성하고 Pool이라는 공간에 저장
  - 저장된 DB Connection 객체는 요청에 따라 필요할 때마다 Pool에서 가져다 쓰고 반환
  - 요청할 때마다 물리적인 Connection 객체를 생성하는 비용을 줄일 수 있음

- DBCP 라이브러리 설치
  - Tomcat 7부터는 자체적으로 DBCP를 내장하고 있으므로 별도의 라이브러리를 설치하지 않아도 됨
  - Tomcat 7버전 이하일 경우 버전에 해당하는 라이브러리를 설치해야 함

DBCP

■ DBCP 라이브러리 다운로드

- 다운로드 사이트
  - https://commons.apache.org/proper/commons-dbcp/download_dbcp.cgi
  - 예를 들어 JDK 6일 경우
    - Binaries의 zip 버전을 다운로드



Apache Commons DBCP 1.4 for JDBC 4 on Java 6

Binaries

commons-dbcp-1.4-bin.tar.gz	sha256	pgp
commons-dbcp-1.4-bin.zip	sha256	pgp

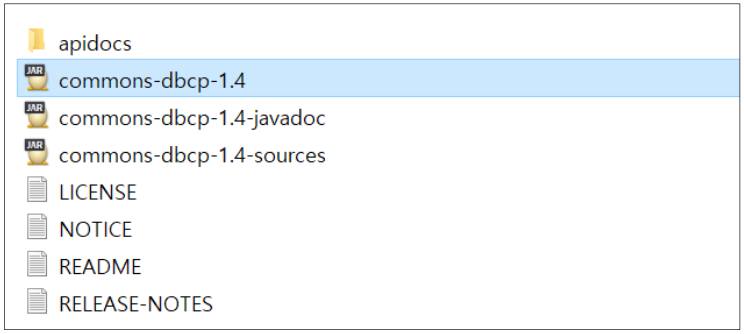
Source

commons-dbcp-1.4-src.tar.gz	sha256	pgp
commons-dbcp-1.4-src.zip	sha256	pgp

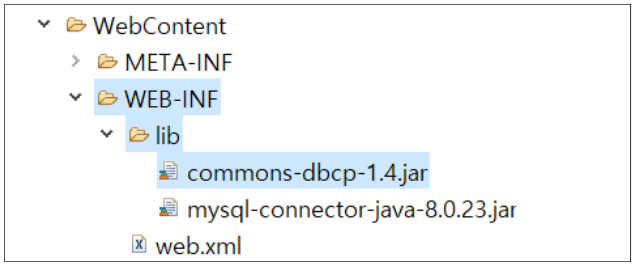
DBCP

■ DBCP 라이브러리 설치

- 다운로드한 압축 파일을 임의의 폴더에 압축 해제
  - commons-dbcp-1.4 폴더의 commons-dbcp-1.4.jar 파일을 복사



- 프로젝트/WebContent/WEB-INF/lib 폴더에 복사



- 현재 JDK 9버전을 사용하고 있으므로 라이브러리를 추가로 설치할 필요는 없음

DataSource 설정

■ DataSource 설정

- **Servers/Tomcat 9.0 Server at localhost-config/server.xml를 이용한 설정**
  - server.xml은 Tomcat Server이 주요 설정 파일임
    - 적용된 내용은 서버에 존재하는 모든 어플리케이션에 영향을 미침
  - JDK 4.0 이전에 DataSource 설정에 사용되었음
    - 현재는 context.xml 문서를 사용해 지정함

- **Servers/Tomcat 9.0 Server at localhost-config/context.xml를 이용한 설정**
  - 서버 내의 어플리케이션에 적용되는 내용을 설정하는 파일
    - 이전에는 server.xml에 포함되어 있었으나 현재는 별도의 파일로 존재한
  - DataSource 설정에 주로 사용됨

- **프로젝트/WebContent/META-INF/context.xml을 이용한 설정**
  - 어플리케이션 차원에서 DataSource 설정에 사용
  - 직접 생성해야 함

DataSource 객체

■ DataSource의 설정

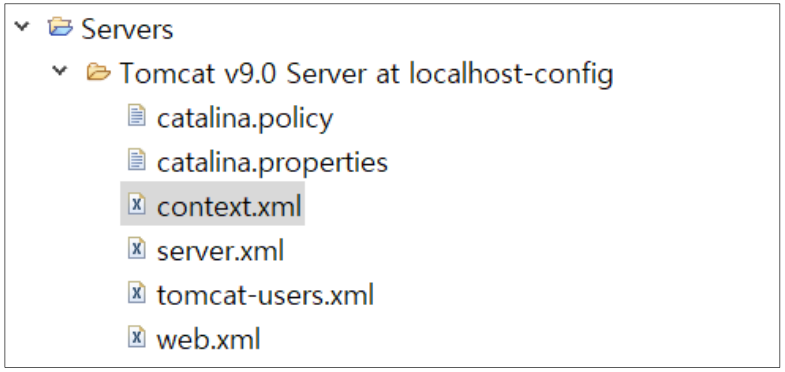
- Servers/Tomcat 9.0 Server at localhost-config/context.xml를 사용
  - <Resource> 태그의 사용해 설정

- <Resource> 태그의 주요 속성

속성	설명
name	DataSource에 대한 JNDI 이름
auth	인증 주체
type	ConnectionPool을 사용할 수 있도록 하는 DataSource
driverClassName	연결할 데이터베이스 종류에 따른 드라이버 클래스 이름
url	접속할 데이터베이스 주소와 포트 번호 및 SID
username	데이터베이스 접속 ID
password	데이터베이스 접속 비밀번호
maxActive	동시에 최대로 데이터베이스에 연결할 수 있는 Connection 수
maxWait	새로운 연결이 생길 때까지 기다릴 수 있는 최대 시간

DataSource 객체

■ context.xml를 수정




```
25      <!-- Uncomment this to disable session persistence across Tomcat restarts -->
26      <!--
27      <Manager pathname="" />
28      -->
29
30    <Resource
31          name="jdbc/mysql"
32          auth="Container"
33          type="javax.sql.DataSource"
34          driverClassName="com.mysql.cj.jdbc.Driver"
35          url="jdbc:mysql://localhost:3306/jspdb?serverTimezone=UTC"
36          username="jspstudy"
37          password="jsppass"
38          maxActive="5"
39          MaxWait="-1" />
40
41  </Context>
```

커넥션 풀의 활용

■ 커넥션 풀의 활용

- 클래스 추가
  - ▸ InitialContext 객체, DataSource 객체 생성을 위한 클래스 추가 필요

```
<%@ page import="javax.naming.Context"; %>
<%@ page import="javax.naming.InitialContext; %>
<%@ page import="javax.sql.DataSource; %>
```

커넥션 풀의 활용

• DataSource 객체 생성

```
Context   ctx = new InitialContext();
DataSource   ds = (DataSource) ctx.lookup(" java:/comp/env/jdbc/mysql");
```

▸ **InitialContext 객체 생성**

- JNDI를 사용해 객체에 접근하기 위해서는 반드시 InitialContext 객체를 생성해야 함
- InitialContext 객체는 객체를 저장(bind)하고 검색(lookup)하는 방법을 제공함

▸ **lookup( ) 메서드를 사용해 DataSource 객체 생성**

- InitialContext 객체에 lookup( ) 메서드를 사용해 DataSource 객체를 생성
- lookup( ) 메서드의 인자로 JNDI 기본 네임 스페이스(java:/comp/env)와 JNDI의 이름(/jdbc/mysql)을 지정

• 커넥션 풀에서 connection 추출

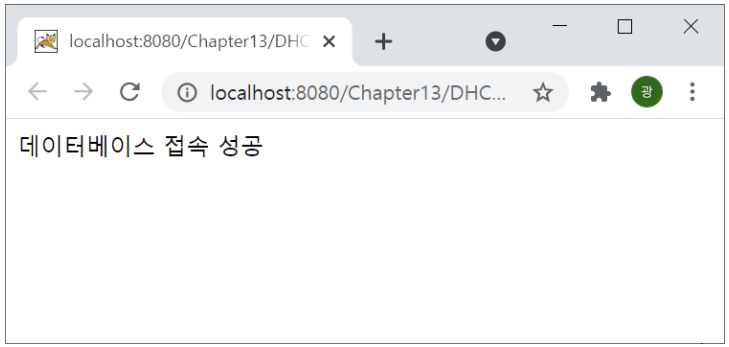
▸ **DataSource 객체의 getConnection() 메서드를 사용해 connection추출**

```
Connection conn = ds.getConnection();
```

커넥션 풀의 활용

DHCPConnection.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<%@ page import="javax.naming.Context" %>
<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.sql.DataSource" %>

<%

    try {

        Context init = new InitialContext();
        DataSource ds = (DataSource)init.lookup("java:comp/env/jdbc/mysql");

        Connection conn = ds.getConnection();

        if( conn != null ) {
            out.println("데이터베이스 접속 성공");
        }

    } catch(SQLException e){
        out.println("데이터베이스 접속 실패");
        e.printStackTrace();
    }

%>
```



수고하셨습니다