8253jang@daum.net
장희정

DBCP(DataBase Connection Pool)란?

데이터베이스 연결(Connection)을 미리 만들어서 풀(pool)에 보관해두고, 필요할 때 빌려쓰는 방식이다. 연결이 필요할 때마다 새로 만들고 닫는 게 아니라 **재사용**해서 성능을 높여주는 기술이다.

왜 DBCP가 필요할까?

▷ 기존방식

```
Connection con = DriverManager.getConnection(...); // 매번 DB 연결
```

▷문제점

일반적으로 백엔드 서버와 DB 서버는 서로 다른 머신에 분리해 운영한다. 두 서버 간 통신은 TCP 기반이라 송수신 신뢰성은 높지만, 매 요청마다 DB 연결을 새로 열고 닫으면 그 자체가 반복적인 오버헤드가 되어 성능을 떨어뜨린다. 사용자가 많아질수록 연결/해제 비용이 누적되고, DB 자원이 불필요하게 소모되며 응답 속도도 느려진다. 이러한 문제를 해결하기 위한 대표적인 대안이 DBCP(데이터베이스 커넥션 풀)이다..

▷ **해결 방법** → DBCP

데이터베이스 연결을 미리 여러 개 만들어 풀에 보관해 두고, 요청이 들어올 때마다 하나씩 빌려 쓰고 작업이 끝나면 다시 반납한다. 이렇게 연결을 재사용하면 불필요한 생성·해제 비용이 줄어들어 전반적으로 더 **효율적이고 빠른** 처리가 가능하다.

DBCP 구성요소

구성 요소	설명
Connection Pool	DB 연결을 미리 만들어 저장해두는 공간
DataSource	Connection을 꺼내주는 객체 (JNDI로 사용)
Resource Ref	web.xml에서 리소스 이름 지정

JNDI(Java Naming and Directory Interface)는 **이름으로 객체를 찾는 표준 API**

https://tomcat.apache.org/tomcat-10.1-doc/jndi-datasource-examples-howto.html

8253jang@daum.net
장희정

DBCP 연동 방법 (Tomcat 기준)

Step 1.context.xml에 리소스 등록

(src/main/webapp/META-INF/context.xml 또는 Tomcat/conf/context.xml)

```
<Resource
 name="jdbc/TestDB"
 auth="Container"
 type="javax.sql.DataSource"
 maxTotal="100"
 maxIdle="30"
 maxWaitMillis="10000"
 username="javauser"
 password="javadude"
 driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/javatest"     />
```

Resource의 속성 정보

name="jdbc/TestDB"
이 리소스의 **JNDI 이름**. 애플리케이션에선 보통 java:comp/env/jdbc/TestDB 로 찾는다.

auth="Container"
누가 인증 정보를 관리하느냐를 뜻. Container 는 톰캣이 리소스(커넥션풀) 생명주기와 접속을 관리한다는 의미이다. 데이터소스는 일반적으로 Container 를 사용한다..

type="javax.sql.DataSource"
이 JNDI 리소스가 반환하는 **객체 타입**. JDBC 의 DataSource 여야 커넥션 풀을 통해 Connection 을 빌려 쓸 수 있다.

maxTotal="100"
풀에서 동시에 만들 수 있는 **총 커넥션 최대 개수**. 이 숫자에 도달하면, 추가 요청은 대기(혹은 실패)한다.

maxIdle="30"
풀에 **유지할 유휴(Idle) 커넥션의 최대 개수**. 반납됐는데 이 수를 넘으면 초과분은 닫힌다. 메모리/리소스 과잉 유지를 막는다..

maxWaitMillis="10000"

풀이 꽉 찼을 때 커넥션을 빌리기 위해 기다릴 최대 시간(밀리초). 여기선 10 초.

- -1 이면 무한 대기, 0 이면 즉시 예외를 던진다.

username / password

DB 접속 계정과 비밀번호. 톰캣이 풀을 만들 때 이 자격증명으로 접속한다.

driverClassName="com.mysql.jdbc.Driver"

JDBC 드라이버 클래스 이름이다.

url="jdbc:mysql://localhost:3306/javatest"

JDBC 접속 URL. 호스트/포트/스키마를 지정한다.

- MySQL 8.x 라면 환경에 따라 다음과 같은 파라미터를 덧붙이는 게 흔하다:

?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Seoul&useSSL=false&allowPublicKeyRetrieval=true

▷ **함께 쓸 수 있는 옵션들**

- initialSize : 기동 시 **미리 만들어 둘 커넥션 수**
- minIdle : 항상 유지할 **최소 유휴 커넥션 수**
- validationQuery="SELECT 1" : 커넥션 **유효성 검사 쿼리**
- testOnBorrow="true" : 커넥션을 **빌릴 때 검사**
- testWhileIdle="true", timeBetweenEvictionRunsMillis : **백그라운드 유효성 검사/청소 주기**
- maxConnLifetimeMillis : 커넥션의 **최대 수명**(DB/네트워크 문제 예방)
- removeAbandonedOnBorrow="true", removeAbandonedTimeout="60" : **대여 후 반납되지 않은 커넥션 회수**

Step 2.web.xml에 ResourceRef 설정– 생략가능

```
<resource-ref>
<description>MySQL Connection</description>
<res-ref-name>jdbc/myDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

context.xml : "진짜 자원(풀)"을 서버에 **실제로 만들고** 어떤 이름으로 노출할지 정함.

web.xml 의 <resource-ref>: 애플리케이션이 사용할 **자원 '참조 선언'(계약서)**. 타입/이름/인증 주체 등을 **컨테이너가 검증**하고, 주입/룩업 매핑 근거가 됨.

<resource-ref>는 없어도 동작한다. Tomcat 이 이 리소스를 **웹앱의 ENC**(Environment Naming Context)인 java:comp/env/jdbc/TestDB 에 자동으로 바인딩한다.

Tomcat 은 관대해서 <resource-ref>가 없어도 종종 **그냥 동작**하지만, **타입 검증·이식성·명세 준수**를 위해 넣는 게 바람직하다.

Tomcat 은 관대하지만, 다른 WAS(예: Payara, WebSphere 등)에서는 <resource-ref>가 없으면 **주입/매핑이 실패**하거나 경고가 난다.

팀/기관 표준, 보안 심사, 교육 자료 등에서는 **명세대로 선언**하는 것이 일반적임.

Step 3. Servlet 또는 DAO에서 DBCP 연결 사용

```
Context initContext = new InitialContext();
Context envContext = (Context)initContext.lookup("java:comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/myDB");

Connection conn = ds.getConnection(); // 풀에서 하나 꺼내기

// 이후 사용
PreparedStatement ps = conn.prepareStatement("SELECT * FROM member");
ResultSet rs = ps.executeQuery();
...
rs.close();
ps.close();
conn.close(); // 실제로는 풀에 반납됨!
```

정리

장점	설명
성능 향상	매번 DB 연결/해제하지 않아 빠름
자원 절약	연결 재사용으로 DB 부하 감소
코드 단순화	JNDI로 쉽게 관리 가능
확장성 ↑	사용자 증가에도 안정적인 대응 가능

8253jang@daum.net
장희정

DBCP는 **데이터베이스 연결을 효율적으로 관리하기 위한 커넥션 풀 기술**이며,
Tomcat의 context.xml + JNDI(DataSource) 방식으로 연동해서 사용한다.