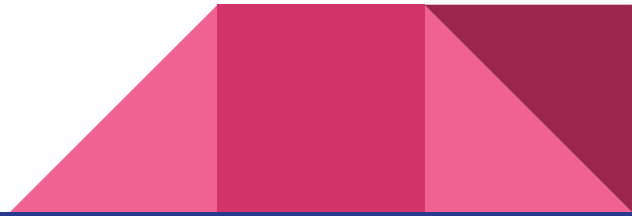
JSP

MVC Model2 + JDBC/DBCP

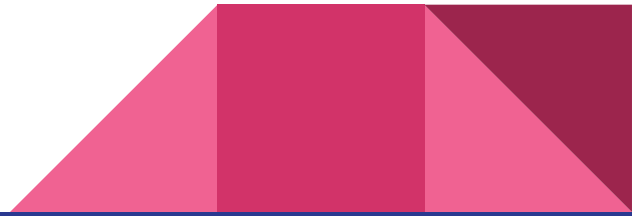
MVC(Model-View-Controller) Pattern

- 아키텍처 패턴의 하나
- 애플리케이션 전체를 모델, 뷰, 컨트롤러의 관점으로 구분하는 패턴
- Page Controller Pattern과 Front Controller Pattern으로 구분
- 주로 웹 애플리케이션 개발에 적용
- MVC 구현 프레임워크
  - 스트럿츠 / 스프링
  - 루비 온 레일즈 / 장고와 파이썬
  - PHP용 젠드 프레임워크
  - 모노레일 (ASP.NET을 MVC 프레임워크) / ASPNET MVC



MVC(Model-View-Controller) Pattern

- 모델
  - 개발자가 처리할 데이터를 표현하는 클래스
  - 데이터의 변경과 조작을 위한 비즈니스 규칙을 표현하는 클래스
- 뷰
  - 애플리케이션의 사용자 인터페이스 영역
- 컨트롤러
  - 사용자와의 커뮤니케이션
  - 애플리케이션의 특정 로직을 처리하는 클래스 영역
  - 애플리케이션의 전체적인 흐름 관리



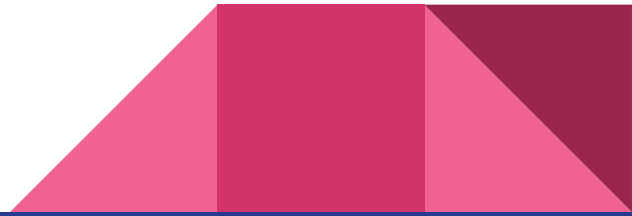
MVC 패턴의 장/단점

장점

- 하나의 모델로 여러가지 뷰를 구현하는 것이 가능
  - 하나의 비즈니스 로직 컴포넌트를 사용하는 웹 페이지와 윈도우 애플리케이션 개발 가능
- 컴포넌트를 쉽게 재사용 할 수 있는 구조
- 한 영역의 컴포넌트에 변경이 발생해도 다른 영역의 컴포넌트에 미치는 영향을 최소화 할 수 있는 구조
  - MySQL DB를 Oracle로 변경해도 뷰에는 영향을 미치지 않는 구조

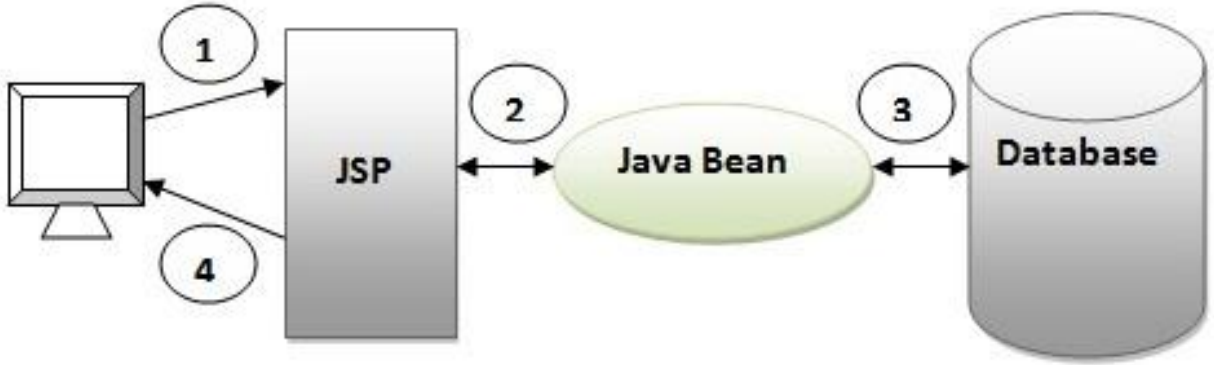
단점

- 복잡성의 증가로 구현의 난이도가 높은 구조
- 모든 컴포넌트간의 상호작용을 고려할 필요가 있는 구조



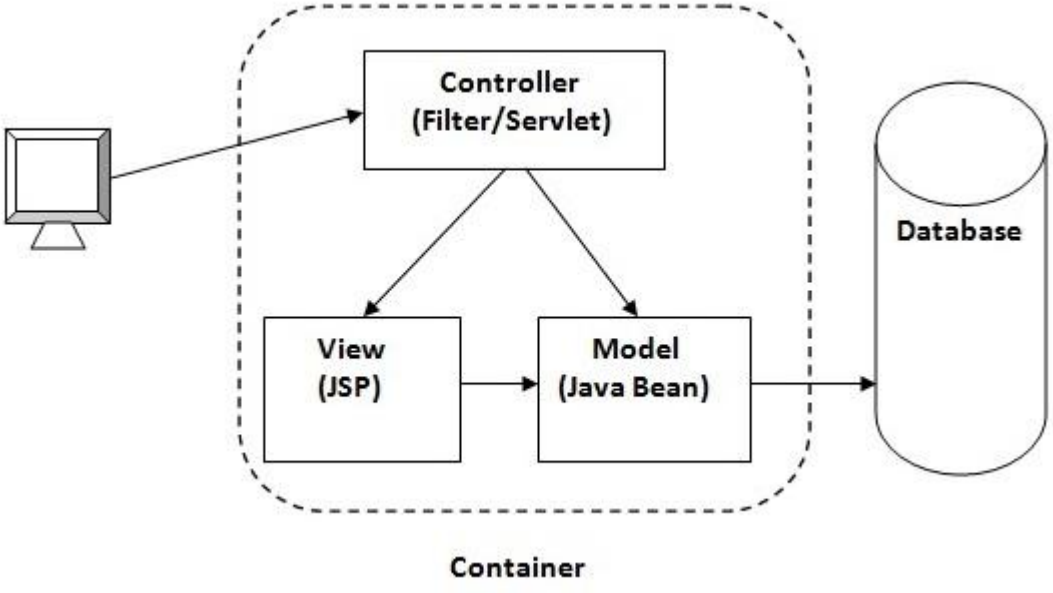
MVC Model 1

- JSP를 이용한 단순한 모델(Page Controller Model)
- JSP에서 요청 처리 및 뷰 생성 처리
  - 구현이 쉬움
  - 요청 처리 및 뷰 생성 코드가 뒤섞여 코드가 복잡함

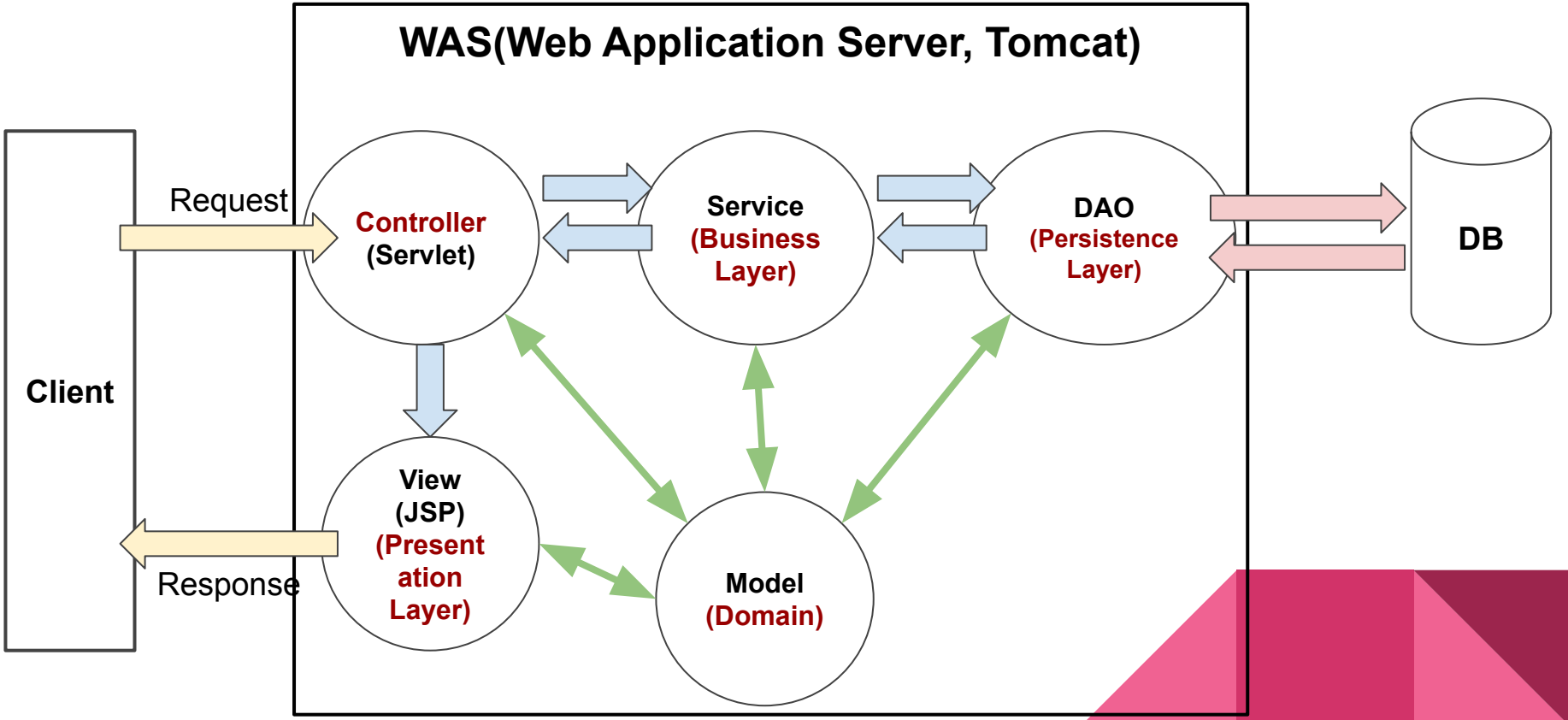


MVC Model 2

- 서블릿이 요청을 처리하고 JSP가 뷰를 생성
- 모든 요청을 단일 서블릿에서 처리
- 요청 처리 후 결과를 보여줄 JSP로 이동

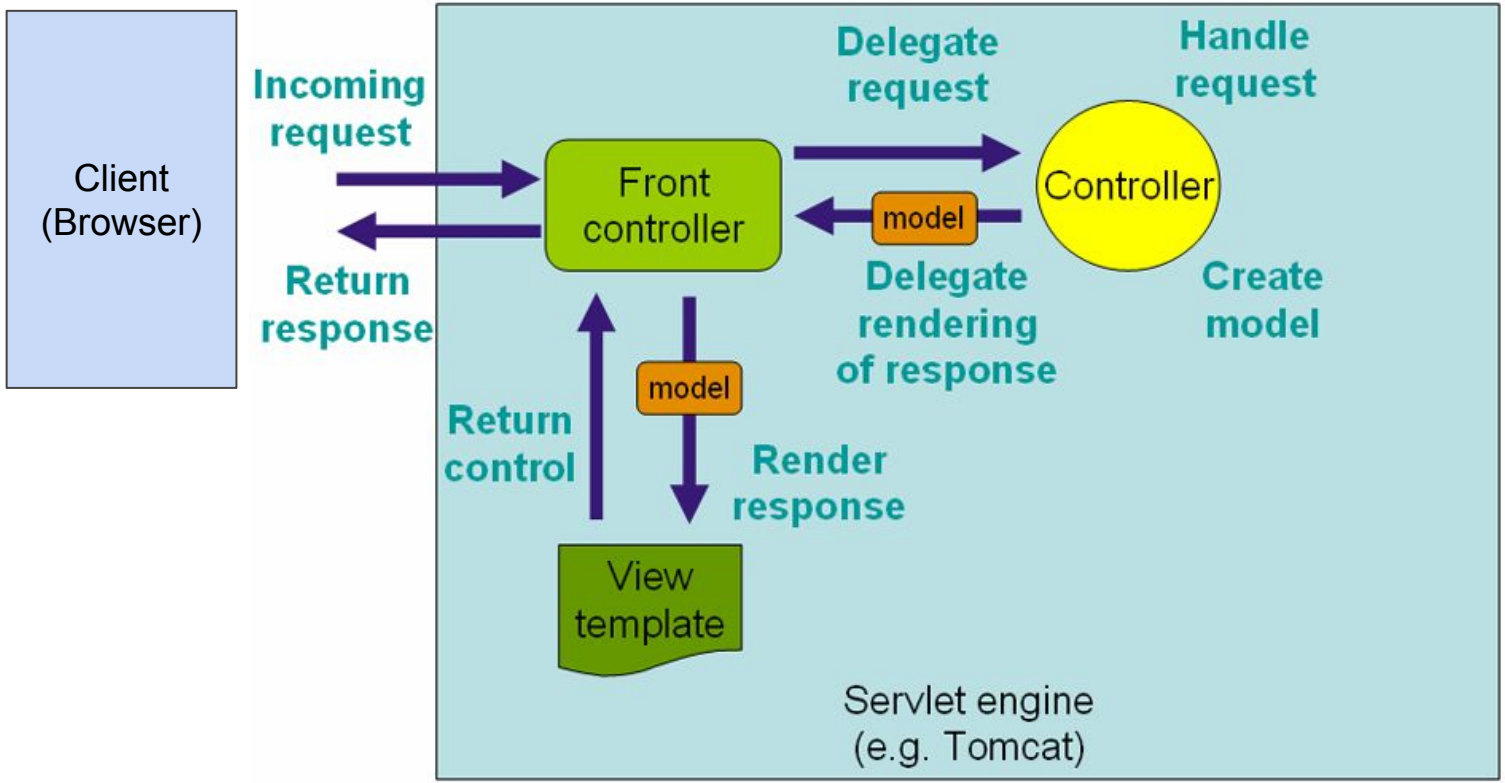


MVC를 활용한 웹 애플리케이션 구조



Model 2 MVC + Front Controller

Delegation pattern(위임 패턴)

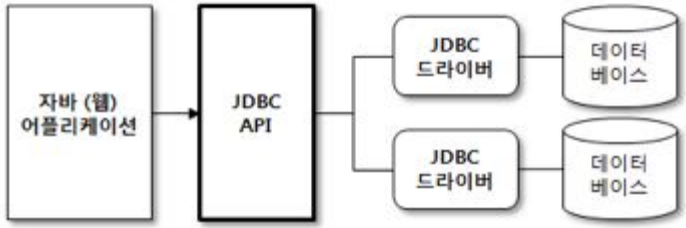


MVC를 활용한 웹 애플리케이션 구조

- VO(Value Object) 클래스*(Domain)*
  - DB와 관련된 ***Model*** 정의
- JSP*(Presentation 계층)*
  - Controller가 처리한 결과를 화면에 출력하는 ***View***를 담당
- Servlet 클래스*(Controller)*
  - 사용자의 요청을 Service에 전달하고 Service의 실행 결과를 JSP와 같은 뷰에 전달하는 ***Controller***를 담당
- Service 클래스*(Business/Service 계층)*
  - 사용자의 요청을 처리하기 위한 비즈니스 로직을 구현.
  - DAO 클래스를 통해서 DB 연동을 처리
  - (예) 가입 신청 처리, 글 목록 제공 등의 기능을 구현.Service
- DAO 클래스*(Persistence/Repository 계층)*
  - DB와 관련된 CRUD(Create/Read/Update/Delete) 작업을 처리(SQL 쿼리를 실행)

JDBC(Java Database Connectivity)

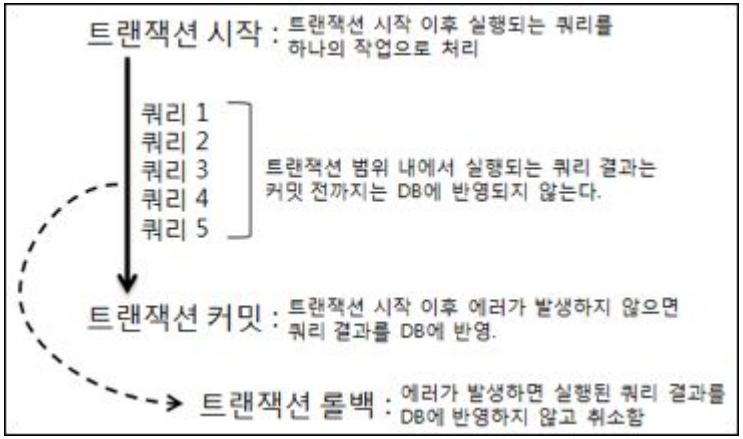
- JDBC: 자바에서 DB 프로그래밍을 하기 위해 사용되는 API
- JDBC 구현 기본 순서
  - JDBC 드라이버 로딩
  - 데이터베이스 Connection 생성
  - Statement(PreparedStatement) 생성
  - 쿼리 실행: executeQuery(), executeUpdate()
  - 쿼리 실행 결과 처리/출력
  - 리소스(ResultSet, Statement, Connection) 해제



Transaction

- 데이터의 무결성을 위해 하나의 작업을 위한 쿼리들은 트랜잭션으로 처리될 필요
- 트랜잭션 구현 방법: 오토 커밋 해제, JTA 이용 방식

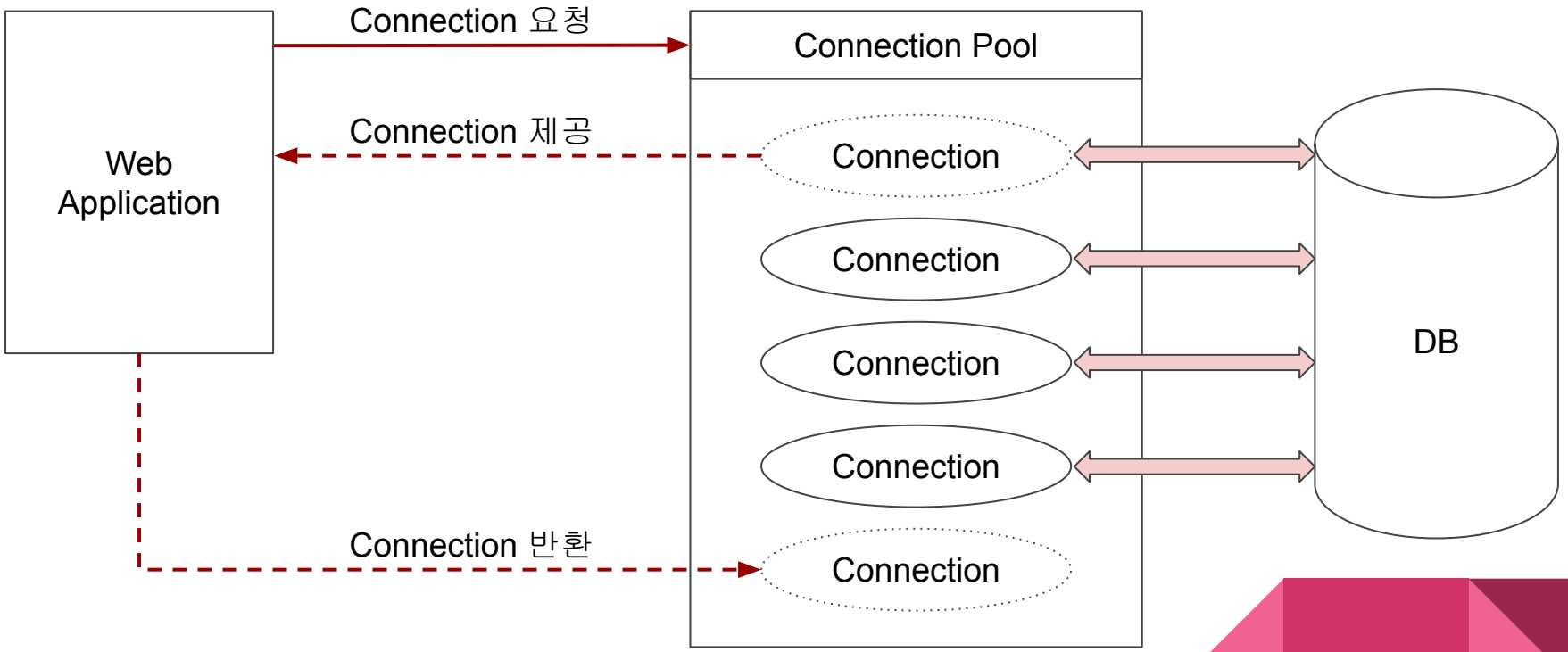
```
try {
      // Connection 생성
      connection.setAutoCommit(false);
      // 필요한 쿼리 실행
      // 필요한 쿼리 실행
      connection.commit(); // 트랜잭션 커밋
} catch (Exception e) {
      connection.rollback(); // 트랜잭션 롤백
      // 필요한 예외 처리
} finally {
      // 리소스 해제
}
```



DBCP(Database Connection Pool)

- JDBC의 단점
  - DB에 연결하기 위해서는 Driver를 로드하고, Connection 객체를 생성해야 함
  - DB 요청이 있을 때마다 Driver를 로드하고, Connection 객체를 생성해서 사용한 후 해제하는 것은 DB에 부하를 많이 주고 비효율적
- DBCP
  - 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때에 커넥션을 풀에서 가져다 쓰고 다시 풀에 반환하는 기법
  - 커넥션을 생성하는 데 드는 연결 시간이 소비되지 않는다.
  - 커넥션을 재사용하기 때문에 생성되는 커넥션 수가 많지 않다.
  - DB의 부하를 줄이고 유동적으로 연결을 관리할 수 있다.
- DBCP 동작 원리
  - 웹 컨테이너가 실행되면서 Connection 객체를 미리 Connection Pool에 생성
  - Connection이 필요할 때마다 Pool에서 객체를 가져다 쓰고 반환

DBCP


Connection 요청
Connection Pool
Connection 제공
Web
Application
Connection
Connection
Connection
Connection
DB
Connection 반환
Connection

Tomcat JNDI DataSource를 이용한 DBCP 설정

- /META-INF/context.xml 파일 설정

```
<Context>
     <Resource name="jdbc/jsptest" auth="Container"
         type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
         url="jdbc:oracle:thin:@127.0.0.1:1521:xe" username="scott" password="tiger"
         maxTotal="100" maxIdle="30" maxWaitMillis="10000" />
</Context>
```

- /WEB-INF/web.xml 파일 설정

```
<web-app>
     <resource-ref>
          <description>MySQL DataSource example</description>
          <res-ref-name>jdbc/jsptest</res-ref-name>
          <res-type>javax.sql.DataSource</res-type>
          <res-auth>Container</res-auth>
     </resource-ref>
</web-app>
```

참조: http://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html