서블릿과 JDBC 연동

6.1_JDBC(Java DataBase Connectivity)의 개요

6.2_JDBC 고급 기법

6.1 JDBC(Java DataBase Connectivity)의 개요 – 1

-6.1.1 JDBC 실행전 확인 사항

-데이터베이스 정보 확인

- 데이터베이스가 설치된 IP정보 및 설정된 데이터베이스명을 반드시 알아야 됨

```
SQL> show parameter db_name;
```

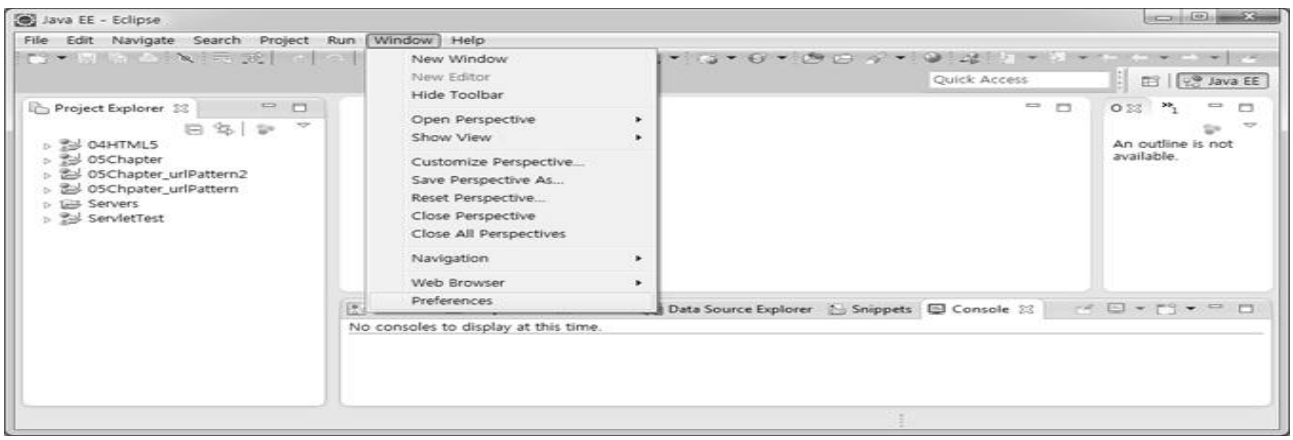
-오라클 드라이버를 클래스 패스에 설정

- 자바 어플리케이션과 오라클 데이터베이스가 연동하기 위해서는 오라클 데이터베이스에서도
- 자바 클래스 파일들이 필요
- 자바 대 자바로 통신이 가능하기 때문
- 오라클 드라이버 : 오라클에서 만든 클래스 파일들의 압축 파일
- 오라클 드라이버는 오라클 데이터베이스를 설치하면 운영체제에 자동 설치
- 현재 사용 중인 Express Edition 11g는 다음 경로에 ojdbc6_g.jar 파일명으로 제공

```
C:₩oraclexe₩app₩oracle₩product₩11.2.0₩server₩jdbc₩lib ₩ojdbc6_g.jar
```

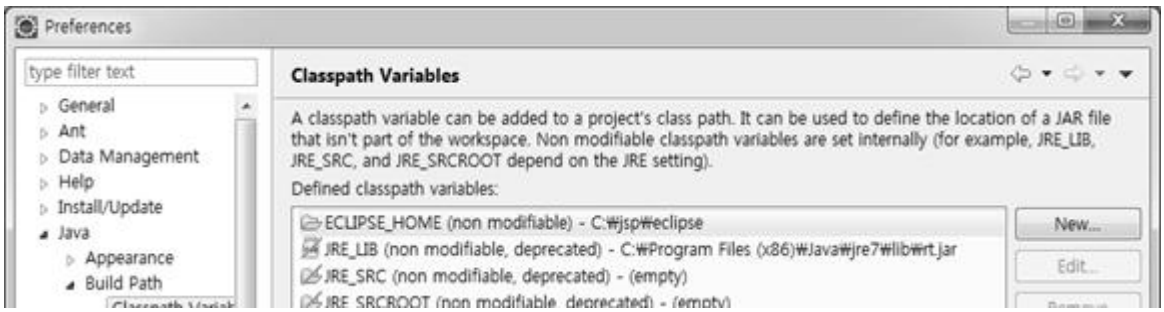
6.1 JDBC(Java DataBase Connectivity)의 개요 – 2

-사용하고자 하는 오라클 드라이버를 이클립스에서 사용하기 위해서는 먼저 [Window]-[Preferences]을 선택

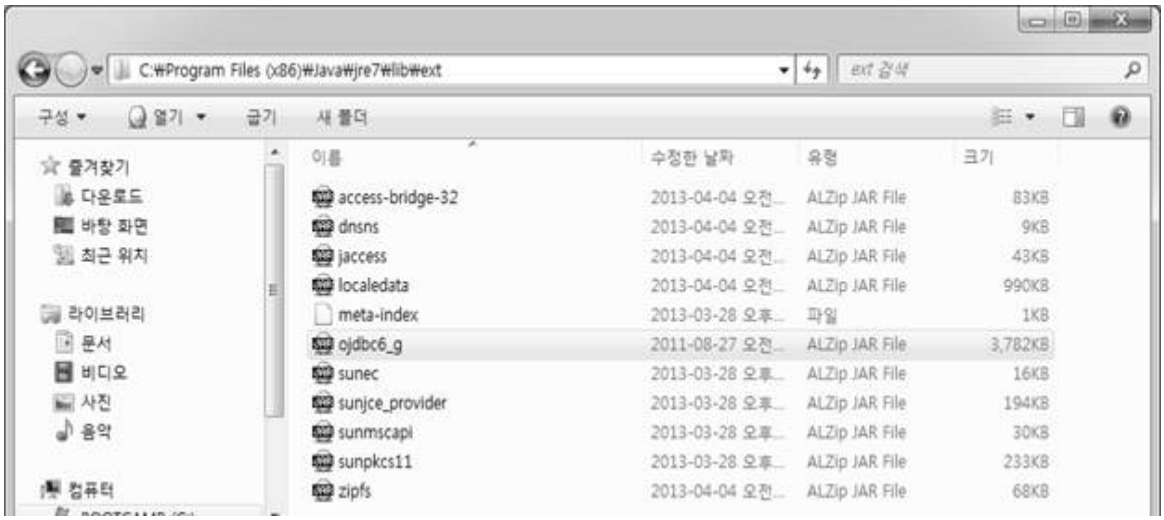


6.1 JDBC(Java DataBase Connectivity)의 개요 – 3

-[Java]-[Build Path]-[ClassPath Variables]에서 JRE_LIB의 경로를 파악



-이 경로에 있는 ext 폴더에 'ojdbc6_g.jar' 파일을 복사하면 자동으로 드라이버가 클래스 패스에 추가



6.1 JDBC(Java DataBase Connectivity)의 개요 – 4

-6.1.2 JDBC 실행 순서

①오라클 데이터베이스 연동을 위한 4가지 정보를 문자열에 저장

```
String driver = "oracle.jdbc.driver.OracleDriver";
String url = "jdbc:oracle:thin:@localhost:1521:xe";
String userid = "scott";
String passwd = "tiger";
```

▸JDBC 연동을 위해서는 오라클 드라이버 파일 중에서 핵심이 되는 클래스 파일명이 필요
▸이 클래스 파일은 oracle.jdbc.driver 패키지 내에 있는 OracleDrvier 클래스
▸이 값을 driver 변수에 저장
▸오라클의 위치 및 포트번호, 데이터베이스명의 정보를 url 변수에 저장
▸오라클의 기본 포트번호는 1521번이고 데이터베이스 위치는 현재 로컬 컴퓨터에 설치했기

때문에 localhost라고 지정

- 오라클 데이터베이스가 원격에 있다면 반드시 원격 IP 번호를 지정
- 데이터베이스명은 현재 XE 값
- 마지막으로 접속하고자 하는 계정명과 비밀번호를 userid 변수와 passwd 변수에 저장

6.1 JDBC(Java DataBase Connectivity)의 개요 – 5

②드라이버 로딩

- 앞에서 살펴봤던 OracleDriver 클래스를 메모리에 올려야 함

```
Class.forName( driver );
```

③Connection 맺기

- 자바 코드와 오라클 데이터베이스를 연결하는 것을 의미
- 연결은 java.sql 패키지의 Connection을 사용
- 다음과 같이 DriverManager 클래스를 이용해서 Connection을 얻음

```
Connection con = DriverManager.getConnection( url, userid , passwd );
```

- getConnection 메서드 인자에 저장했던 url 값과 userid, passwd 값을 지정하면, 이 정보를 이용해서 데이터베이스에 연결

6.1 JDBC(Java DataBase Connectivity)의 개요 – 6

④SQL 문 작성

- 자바에서 데이터베이스에 요청할 SQL 문은 문자열로 저장
- SQL 문에서 ;(세미콜론)은 입력하지 않음
- emp 테이블의 데이터를 검색하기 위한 SQL 문

```
String query = "SELECT emp_id, ename, salary FROM emp";
```

- emp 테이블의 데이터 중에서 salary가 4000인 레코드를 삭제하는 SQL 문

```
String sql = "DELETE FROM emp WHERE salary= 4000";
```

⑤Statement 생성

- SQL 문을 전송할 때 사용되는 API
- Connection의 createStatement( ) 메서드를 사용해서 Statement를 생성

```
Statement stmt = con.createStatement( );
```

6.1 JDBC(Java DataBase Connectivity)의 개요 – 7

⑥SQL 문 전송 및 결과 값 얻기

- SQL 문 전송은 Statement 객체를 이용해서 데이터베이스에 전송
- 요청하는 SQL 문에 따라서 두 가지 메서드를 사용 가능
- DML 요청(INSERT, DELETE, UPDATE)

≫DML 요청은 executeUpdate 메서드를 사용

```
int n = stmt.executeUpdate( sql );
```

≫executeUpdate 메서드의 리턴되는 정수 값은 변경된 레코드의 개수
≫DELETE 문을 요청했을 때 리턴되는 정수 값이 10이라면 삭제된 레코드 개수가 10개라는 것
≫반환된 값을 이용해서 적용된 레코드의 개수를 파악할 수 있으며 디버깅용으로도 사용 가능

6.1 JDBC(Java DataBase Connectivity)의 개요 – 8

▸SELECT 요청
≫SELECT 요청은 executeQuery 메서드를 사용

```
ResultSet rs = stmt.executeQuery( query );
```

≫메서드 인자에 SELECT 문을 지정하면 된다. SQL*PLUS 툴에서 SELECT 문을 실행하면 항상 테이블 형태로 결과가 출력
≫테이블 결과를 자바의 객체로 표현한 것이 ResultSet 객체
≫ResultSet 객체는 포인터를 이용해서 원하는 데이터를 얻을 수 있음
≫먼저 포인터를 이용해서 레코드를 선택하고 나중에 포인터가 가리키는 레코드의 컬럼을 지정해서 데이터를 얻음
≫레코드를 선택하는 메서드는 next( ) 메서드를 사용
≫컬럼은 데이터형에 따라서 getInt(컬럼명) , getString(컬럼명) 메서드를 사용

6.1 JDBC(Java DataBase Connectivity)의 개요 – 9

-결과 값인 ResultSet에서 데이터를 얻는 방법

```
while( rs.next( )) {
    String emp_id = rs.getString("emp_id");
    String ename = rs.getString("ename");
    int salary = rs.getInt("salary");
    System.out.println( emp_id +" " + ename + " " + salary );
}
```

≫결과 값에 레코드가 여러 개 있을 수 있기 때문에 WHILE 문을 사용
≫next( ) 메서드는 레코드가 있는 경우에는 true 값을, 없으면 false 값을 리턴
≫emp_id 컬럼은 VARCHAR2 형이기 때문에 getString 메서드를 사용해서 데이터를 얻음
≫salary 컬럼은 NUMBER 형이기 때문에 getInt 메서드를 사용
≫메서드의 인자 값으로 컬럼명을 사용하거나 SELECT 문에서 사용한 컬럼의 인덱스 값을 사용 가능
≫가독성을 위해서 컬럼명을 사용하는 것을 권장

6.1 JDBC(Java DataBase Connectivity)의 개요 – 10

⑦자원 반납

≫파일 및 데이터베이스는 자바 프로그램에서 사용하는 외부 자원이기 때문에 반드시 사용한 후에는 자원을 해제

≫데이터베이스에서 사용한 자원은 ResultSet, Statement, Connection 등

≫이 자원을 해제 시킬 때에는 사용한 역순으로 해제

≫데이터베이스를 사용 시 예외 발생 여부에 관계없이 항상 자원을 해제시켜야 되기 때문에 일반적으로 FINALLY 문에서 자원 반납 코드를 구현

```
rs.close( ); // ResultSet을 사용한 경우
stmt.close( );
con.close)( );
```

6.2 JDBC 고급 기법 – 1

-6.2.1 PreparedStatement 사용

-Statement 객체

- 일반적인 SQL 문을 전송할 때 사용하는 API
- 새로운 레코드를 여러 번 저장하는 경우에 동일한 INSERT 문을 여러 번 작성해야 되며 문자 데이터 및 날짜 데이터인 경우에는 반드시 ' '를 지정
- 중복 코드가 많아져서 성능면에서 떨어지고 ' '을 지정하지 않으면 예외가 발생되기 때문에 코드 작업이 비효율적

-PreparedStatement 객체

- SQL 문을 한 번만 작성하고 데이터만 나중에 추가로 설정하면서 작업을 할 수 있는 API
- 새로운 레코드를 여러 번 저장하는 경우에 한 번만 INSERT 문을 생성하면 되기 때문에 중복 코드가 제거되어 성능이 뛰어나고 ' '을 직접 지정하지 않기 때문에 예외 발생률이 적어짐
- 앞으로의 모든 SQL 문 전송은 PreparedStatement를 사용하여 전송하는 것을 권장

-CallableStatement 객체

- 자바 언어와 오라클의 프로그램 언어인 PL/SQL과의 연동 시 사용되는 API

6.2 JDBC 고급 기법 – 2

-PreparedStatement 객체를 사용하여 데이터베이스를 연동

①오라클 데이터베이스 연동을 위한 4가지 정보를 문자열에 저장

```
String driver = "oracle.jdbc.driver.OracleDriver";
String url = "jdbc:oracle:thin:@localhost:1521:XE";
String userid = "scott";
String passwd = "tiger";
```

②드라이버 로딩

```
Class.forName( driver );
```

③Connection 맺기

```
Connection con = DriverManager.getConnection( url, userid , passwd );
```

6.2 JDBC 고급 기법 – 3

④SQL 문 작성

≫emp_id가 'a333'인 레코드를 검색하는 SELECT 문

≫실제 값 대신에 '?' 심벌을 이용

≫나중에 메서드를 이용해서 실제 값을 설정

```
String sql = "SELECT ename, salary, depart FROM emp WHERE emp_id = ? ";
```

≫emp_id가 'a222'인 레코드의 depart를 '관리'로 변경하고 salary를 3200으로 변경하는 UPDATE 문

≫실제 값 대신에 '?' 심벌을 이용하고 나중에 실제 값은 메서드를 이용해서 설정

```
String sql= "UPDATE emp SET ename = ? , salary = ? WHERE emp_id = ? ";
```

⑤PreparedStatement 생성

≫요청할 SQL 문을 PreparedStatement를 생성할 때 메서드 인자로 지정하여 작업

≫SQL 문을 한번만 작성하고 나중에 필요한 데이터를 설정하는 방법으로 작업

≫PreparedStatement를 생성하기 전에 요청할 SQL 문을 먼저 작성

```
PreparedStatement pstmt = con.prepareStatement( sql );
```

6.2 JDBC 고급 기법 – 4

⑥SQL 문에 데이터 값 지정

≫PreparedStatement 객체의 set 메서드를 사용

≫ResultSet 객체의 get 메서드와 비슷, 저장할 위치는 인덱스를 이용해서 값 지정

≫인덱스는 SQL 문에서 나오는 '?' 심벌의 순서를 의미

≫NUMBER 형 컬럼에는 setInt(인덱스,값) 메서드를 이용해서 값을 저장

≫VARCHAR2 형 컬럼에는 setString(인덱스, 값) 메서드를 이용해서 저장

≫SQL 문에 지정한 '?' 심벌의 개수와 set 메서드 개수가 SQL 일치해야

```
String sql= "UPDATE emp SET ename = ? , salary = ? WHERE emp_id = ? ";
PreparedStatement pstmt = con.prepareStatement( sql );
    pstmt.setString( 1 , "관리");
    pstmt.setInt( 2 , 3200 0 );
    pstmt.setString( 3, "a222" );
```

≫SELECT인 경우에 데이터를 지정하는 방법

```
String sql = "SELECT ename, salary, depart FROM emp WHERE emp_id = ? ";
PreparedStatement pstmt = con.prepareStatement( sql );
pstmt.setString( 1 , "a333" );
```

6.2 JDBC 고급 기법 – 5

⑦SQL 문 전송

≫DML인 경우에는 인자없는 executeUpdate( ) 메서드를 사용

≫SELECT인 경우에는 인자없는 executeQuery( ) 메서드를 사용

```
int n = pstmt.executeUpdate( );
ResultSet rs = pstmt.executeQuery( );
```

⑧자원 반납

```
rs.close( ); // ResultSet을 사용한 경우
pstmt.close( );
con.close)( );
```

6.2 JDBC 고급 기법 – 6

-6.2.2 DAO 패턴 및 DTO 패턴

-DAO(Data Access Object) 패턴

≫일반적으로 어플리케이션 개발은 GUI 화면을 가지며 화면에 보여줄 수 있는 데이터 관리를 위해서 데이터베이스를 사용해서 개발

≫웹 브라우저에서 보여지는 것처럼 GUI 화면을 구성하는 코드 → presentation logic GUI 화면에 데이터를 보여주기 위해서 데이터베이스를 검색하는 코드 및 GUI 화면에서 새로 발생된 데이터(예를 들면 회원가입)를 데이터베이스에 저장하는 코드와 같은 실제적인 작업을 처리하는 코드 → business logic

≫presentation logic과 business logic을 하나의 클래스로 모두 구현 할 수도 있고 여러

클래스로 모듈화 시켜서 구현이 가능하지만 하나의 클래스로 구현하면 유지보수가 어려워짐

≫모듈화시켜 개발하게 되며 모듈화한 클래스들 중에서 데이터베이스 처리하는 코드만을 관리하는 클래스 작성 → DAO(Data Access Object) 클래스

≫DAO클래스를 사용해서 개발하는 것이 보편화되었기 때문에 DAO 패턴이라고 함

≫일반적으로 DAO 클래스는 테이블 당 한 개씩 생성해서 사용

≫DAO 클래스 안에는 특정 테이블에서 수행할 작업을 메서드로 정의해서 구현

≫presentation logic에서는 DAO 클래스의 메서드를 호출하면서 원하는 작업 구현

6.2 JDBC 고급 기법 – 7

2) DTO(Data Transfer Object) 패턴

≫presentation logic과 business logic을 여러 클래스로 분리해서 작업은 하지만 서로 간에 긴밀한 관계가 유지되면서 작업이 이루어짐

≫presentation logic에서 보여줄 데이터를 얻기 위해서 business logic에게 요청을 하면 business logic은 필요한 데이터를 데이터베이스에서 검색해서presentation logic에게 반환하는 작업 등을 수행

≫데이터를 다른 logic에게 전송 및 반환할 때 효율적으로 데이터를 사용할 수 있게 클래스를 작성 가능 → DTO(Data Transfer Object) 클래스

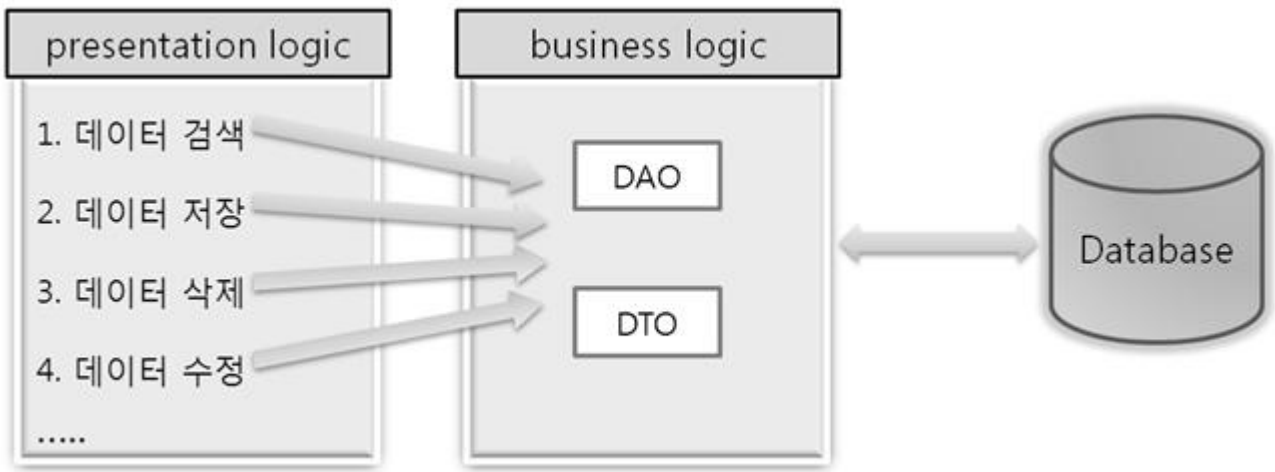
≫DTO 클래스는 이름 그대로 데이터를 전송할 때 사용되는클래스

≫데이터를 전송할 때와 전송된 데이터를 얻어서 사용할 때 효율적으로 사용할 수 있는 장점이 있음

≫일반적으로 도메인 객체(Domain Object), VO(Value Object)라고도 함

6.2 JDBC 고급 기법 – 8

-DAO와 DTO 패턴 사용 구조



6.2 JDBC 고급 기법 – 9

-6.2.3 커넥션 풀(Connection Pool) 기능

≫JDBC를 사용하여 데이터베이스와 연동할 때 가장 많은 자원을 낭비하는 것이 Connection을 연결하는 작업

≫DAO 클래스의 각 메서드 내에서 매번Connection을 맺고 CLOSE 처리 thread-safe 하기 위해서는 반드시 지켜져야 할 사항

≫Connection Pool 기능은 많은 자원을 낭비하는 Connection을 사용자가 요청할 때마다 매번 연결하지 않고, 미리 일정 개수만큼 Connection을 맺어 필요한 DAO 클래스에서는 빌려 사용하고 반환하는 방법

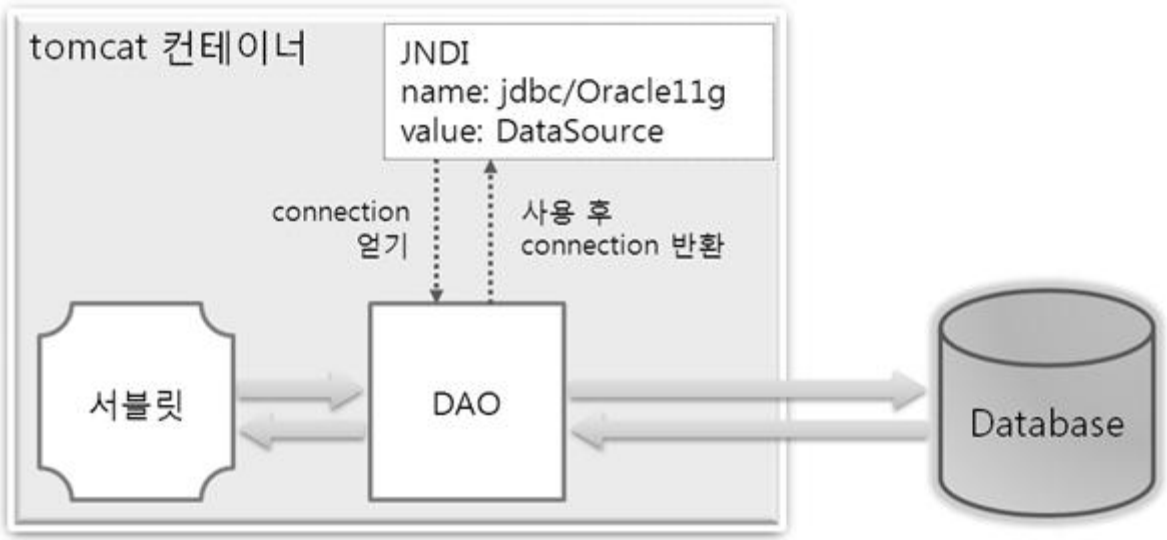
≫과거에는 Connection Pool 기능을 구현한 클래스를 직접 작성하여 사용하였으나, 현재에는 Connection Pool 기능이 포함된 라이브러리를 웹 사이트에서 무료로 다운받아 사용하거나 tomcat 컨테이너에서 제공하는 Pool 기능을 사용

≫Pool 기능을 포함한 API 객체는 javax.sql.DataSource이며 JNDI 기법을 사용하여 tomcat 컨테이너에 등록되어 있음

≫DataSource의 getConnection( ) 메서드를 사용하여 Connection을 얻고 CLOSE 메서드를 사용하여 Pool에 반납

6.2 JDBC 고급 기법 – 10

-전체적인 아키텍처



6.2 JDBC 고급 기법 – 11

-JNDI을 사용한 DataSource 등록 설정

≫이클립스의 [Package Explorer] 탭의 [Servers]에서 'context.xml' 파일을 선택 context.xml 파일에 코드를 추가

≫변경된 context.xml을 tomcat 컨테이너와 동기화하기 위하여 Publish 아이콘 클릭

6.2 JDBC 고급 기법 – 12

-DataSource 객체를 사용하기 위한 두 가지 방법

≫InitialContext 클래스를 사용하는 방법

・이전 버전의 서블릿 스펙에서 사용하던 방법으로 InitialContext 클래스를 사용하여 DataSource 객체를 얻음

≫@Resource 어노테이션을 사용하는 방법

・서블릿에서 @Resource 어노테이션을 사용하여 DataSource 객체를 이용하는 방법

・DAO 클래스와 같은 일반 클래스에서는 사용하지 못하고 서블릿에서만 사용 가능

서블릿 고급

7.1_FrontController 패턴과 Command 패턴
7.2_요청 포워딩(Request Forwarding)
7.3_세션 관리(Session Tracking)
7.4_파일 업로드 및 다운로드 기능

7.1 FrontController 패턴과 Command 패턴 – 1

-7.1.1 FrontController 패턴

-웹 어플리케이션 개발 시 사용자의 요청을 처리하기 위한 최초 진입점(Initial Point)을 정의하고 사용하는 패턴을 의미

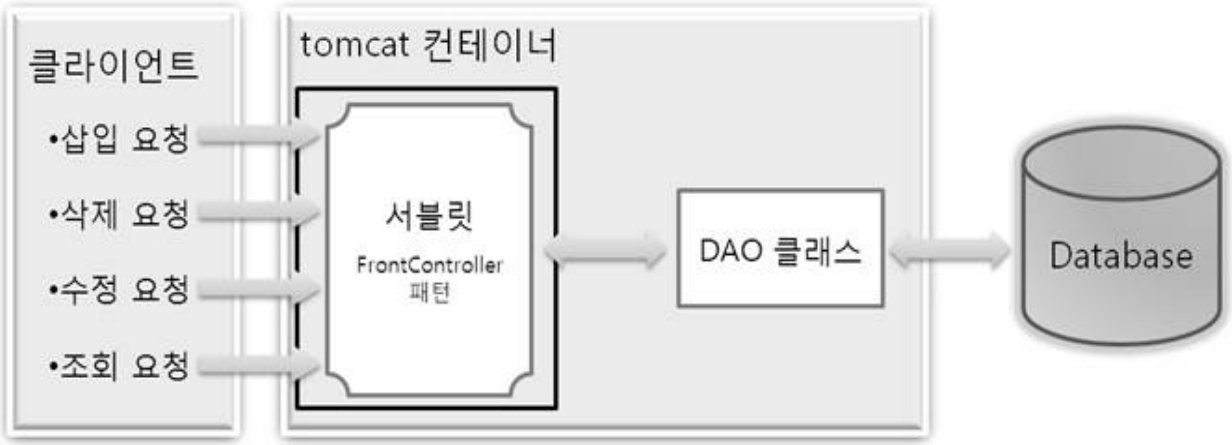
-모든 사용자의 요청을 집중화시키면 요청을 분산시켜 발생되는 중복된 코드를 제거할 수 있고, 사용자의 요청을 일관된 방법으로 관리할 수 있는 장점이 있음

-FrontController 패턴을 적용하지 않은 경우의 아키텍처

-클라이언트의 개별적인 요청을 서로 다른 서블릿이 처리하기 때문에 중복 코드가 발생될 수 있고, 다수의 서블릿으로 인한 유지보수가 어려워질 수 있음

7.1 FrontController 패턴과 Command 패턴 – 2

-FrontController 패턴을 적용한 전체적인 아키텍처



-사용자의 모든 요청을 단 하나의 서블릿이 처리하는 집중화 형태이기 때문에 중복코드가 제거되고 유지보수가 쉬워짐

-FrontController 패턴을 적용한 서블릿에서 고려해야 되는 사항은 사용자가 어떤 동작을 요청했는지를 식별할 수 있어야 됨

-사용자가 서블릿에 요청할 때, 다음과 같은 메커니즘으로서블릿이 사용자의 요청을 식별할 수 있도록 지원

http://서버IP번호:포트번호/context명/식별값

7.1 FrontController 패턴과 Command 패턴 – 3

-사용자는 명시적으로 URL 값에 '식별값'을 추가하여 요청하고, 서블릿에서는 '식별값'을 비교하여 어떤 요청인지를 구별할 수 있음

-'식별값'은 임의의 문자열 값으로서 일반적으로 웹 프레임워크(Spring , Struts2)에서 사용하는

방식으로 지정(예: xxx.do, xxx.nhn).

-다음과 같이 데이터를 저장하는 요청인 경우에는 insert.do로 지정하고 조회를 하는 요청인 경우에는 select.do 형식으로 지정할 수 있음

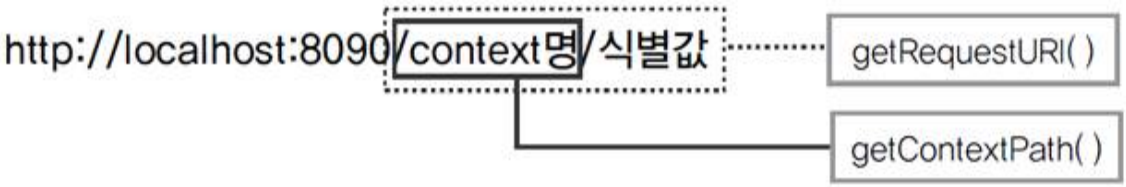
7.1 FrontController 패턴과 Command 패턴 – 4

-요청 받은 서블릿에서는 다음과 같이 두 가지 작업을 통하여 원하는 '식별값'을 얻음

①서블릿의 맵핑명을 다음과 같이 *.do 값의 확장자 패턴 형식으로 지정 따라서 반드시 사용자는 'xxx.do' 형식으로 요청

```
@WebServlet("*.do")
public class 서블릿명 extends HttpServlet { .... }
```

②서블릿에서 다음 코드를 사용하여 '식별값'을 비교 처리할 수 있음




```
String requestURI = request.getRequestURI( );
String contextPath = request.getContextPath( );
String command = requestURI.substring(contextPath.length( ));
if( command.equals( "/insert.do" )) {
    //저장
}else if( command.equals( "/delete.do" )) {
    //삭제
}
```

7.1 FrontController 패턴과 Command 패턴 – 5

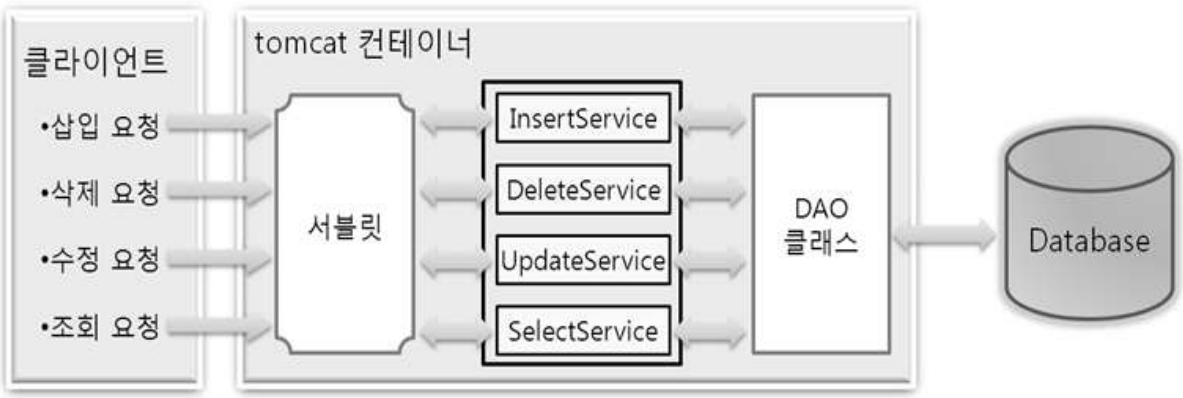
-7.1.2 Command 패턴

-사용자의 요청을 객체인 클래스로 처리하는 것을 의미

-객체 형태로 사용하면 서로 다른 사용자의 요청 값을 필요에 의해서 저장하거나 또는 취소가 가능

-요청을 처리할 작업을 일반화시켜 요청의 종류와 무관하게 프로그램 작성이 가능하게 구현

-구현 방법은 Command 패턴을 적용한 Service 이름의 클래스를 추가



7.1 FrontController 패턴과 Command 패턴 – 6

-Service 클래스는 작업 수행을 요청하는 객체인 서블릿과 실제 작업을 수행하는 객체로 분리시켜 주기 때문에 시스템간의 결합도를 낮출 수 있음

-일반적으로 의존성을 낮추기 위해서 인터페이스를 사용하여 구현

-서블릿과 DAO간의 의존성도 감소시키는 역할을 담당

7.2 요청 포워딩(Request Forwarding) - 1

-사용자의 요청을 받은 서블릿 또는 JSP에서 다른 컴포넌트(서블릿, JSP, html)로 요청을 위임할 수 있는 방법

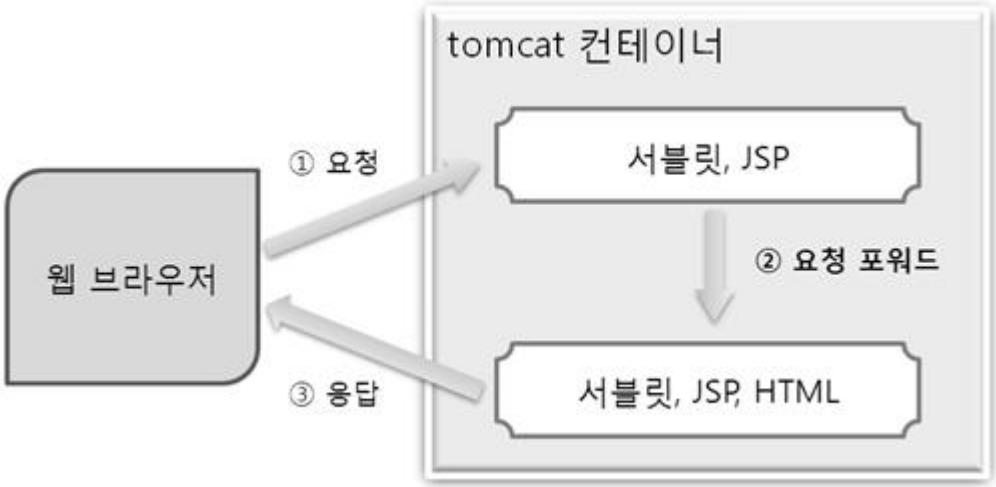
-포워드(forward)하는 이유는 처리 작업을 모듈화하기 위함

-직접 요청받은 서블릿 또는 JSP에서 모든 작업을 처리하지 않고 모듈화 시킨 다른 컴포넌트로 요청을 위임하여 처리할 수 있음

-재사용성도 높아지고 유지보수가 쉬워짐

-일반적으로 MVC 패턴에서 서블릿이 JSP를 포워딩할 때 주로 사용

-사용자의 요청을 받는 웹 컴포넌트와 사용자에게 응답을 처리하는 웹 컴포넌트를 모듈화하여 구현



7.2 요청 포워딩(Request Forwarding) - 2

-일반적으로 요청을 처리하는 웹 컴포넌트는 FrontController 패턴을 적용한 서블릿으로 구현
-응답을 처리하기 위한 웹 컴포넌트는 JSP로 구현할 수 있음
-MVC 패턴 또는 Model 2 Architecture라고 함
-요청 포워딩을 구현하는 방법은 두 가지 방법이 제공
≫RequestDispatcher 클래스를 이용한 forward 방법
≫HttpServletResponse 클래스를 이용한 redirect 방법

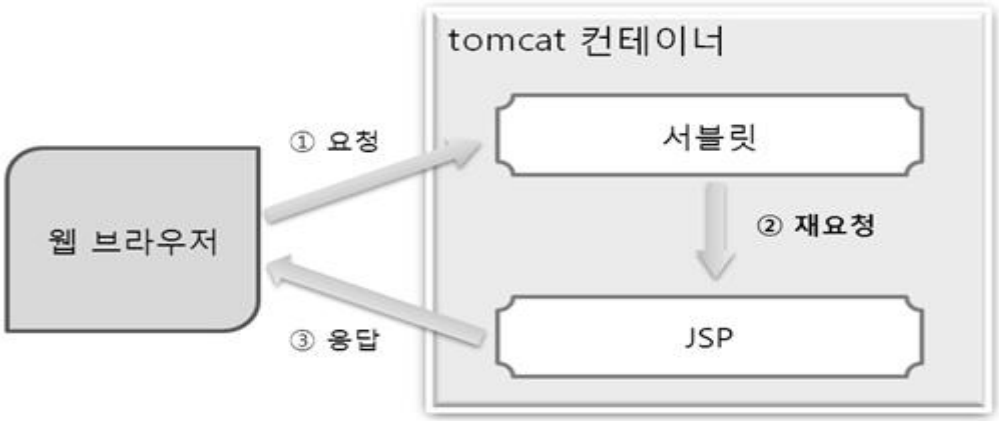
7.2 요청 포워딩(Request Forwarding) - 3

-7.2.1 RequestDispatcher 클래스를 이용한 forward 방법
-웹 브라우저를 통하여 사용자가 서블릿에 요청을 하면 HttpServlet Request 객체가 자동으로 생성
-생성된 request 객체를 사용하여 응답 처리하는 웹 컴포넌트(서블릿, JSP, html)로 재요청하는 방식을 의미
-request 객체의 주된 용도는 사용자의 입력 파라미터 값을 얻어오거나 또는 한글 인코딩 처리 및 request scope에 해당되는 속성(Attribute) 설정에 사용 가능
-요청받은 서블릿에서 다른 컴포넌트로 포워드하는 방법

```
RequestDistatcher dis = request.getRequestDispatcher(target);
dis.forward( request, response );
```

-지정된 target에는 서블릿 맵핑명 또는 JSP 파일 및 html 파일 지정 가능

7.2 요청 포워딩(Request Forwarding) - 4



-1번의 요청과 2번의 요청이 같은 HttpServletRequest를 사용
-JSP로 응답이 되었기 때문에 웹 브라우저의 URL 값이 서블릿에서 JSP로 변경되어야 하지만 포워드 방법은 동일한 HttpServletRequest이기 때문에 URL 값이 서블릿으로 고정
-1번의 요청을 받은 서블릿에서 reqeust scope에 속성(Attribute) 설정을 하면, 2번의 요청을 받은 JSP에서 속성 값을 가져올 수 있음
-일반적으로 MVC 기반의 웹 어플리케이션에서 주로 사용

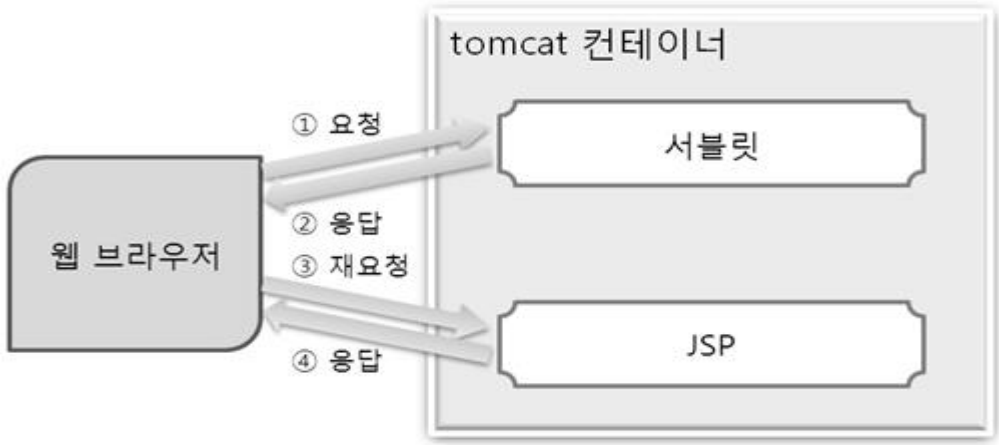
7.2 요청 포워딩(Request Forwarding) - 5

-7.2.2 HttpServletResponse 클래스를 이용한 redirect 방법
-forward 방식과 마찬가지로 다른 웹 컴포넌트에게 재요청을 하는 방법
-차이점은 응답을 먼저 하고 재요청이 되기 때문에, 동일한 HttpServlet Request가 아닌 새로운 request 객체가 생성
-웹 브라우저의 URL 값이 변경되고 속성(Attribute)에 설정된 값을 가져오지 못함
-요청받은 서블릿에서 다른 컴포넌트로 redirect 하는 방법

```
response.sendRedirect(target);
```

-지정된 target에는 서블릿 맵핑명 또는 JSP 파일 및 html 파일 지정 가능

7.2 요청 포워딩(Request Forwarding) - 6



-1번 요청의 결과로 2번의 응답이 발생되고, 자동으로 response.send Redirect(target) 메서드에 의해서 target으로 재요청이 발생
-1번의 요청과 3번의 요청은 서로 다른 요청
-서블릿에서 reqeust scope에 속성(Attribute) 설정을 하면, 3번의 요청을 받은 JSP에서 속성 값을 가져올 수 없음
-일반적으로 JSP를 이용한 Model 1 Architecture 웹 어플리케이션 개발 방법에 주로 사용

7.3 세션 관리(Session Tracking) - 1

-일반적으로 사용되는 세션의 정의는 '서버와 클라이언트간의 지속적인 연결'을 의미
-연결을 통하여 클라이언트는 지속적으로 서버에 특정 동작을 요청할 수 있으며 서버는 실행 결과를 클라이언트에 응답할 수 있음
-데이터베이스를 사용하는 경우에도 클라이언트와 DB서버 간에 지속적인 연결을 의미하는 세션이 필요
-HTTP 프로토콜을 기반으로 하는 웹 서비스는 동작 메커니즘 다름
-불특정 다수인 클라이언트와 지속적인 연결방식으로는 웹 서버의 부하가 매우 크기 때문
-동시 접속자가 100만 건이라고 가정했을 때, 100만 클라이언트와 지속적으로 연결된 서버가 동작하는 것은 불가능
-클라이언트가 웹 서버에 요청하고 응답 받으면 즉시 연결을 끊는connectionless 방식으로 동작

7.3 세션 관리(Session Tracking) - 2

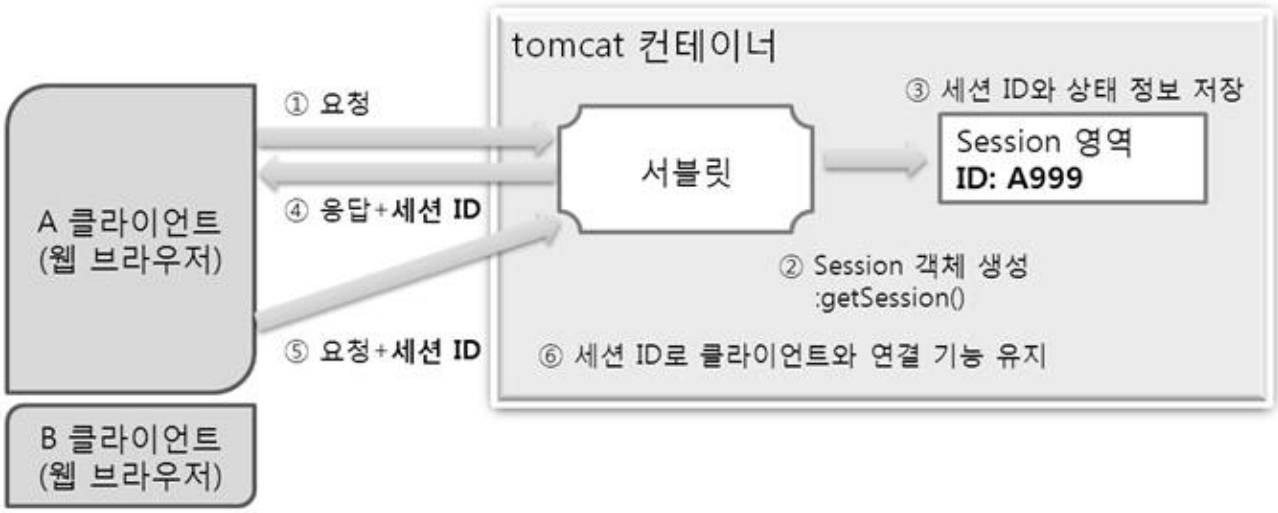
-이것은 웹 브라우저의 각 페이지는 서로 간에 연결고리가 없다는 것을 의미
-첫 화면에서 선택한 물건을 장바구니에 담고 다음 페이지에서 결제할 때 이전 화면의 장바구니에 담긴 정보를 확인 할 수 없다는 것
-서비스되고 있는 많은 웹 사이트를 보면 장바구니 및 로그인 기능 같은 처리가 구현
-HTTP 프로토콜의 문제점을 극복하기 위해서 사용자의 상태 정보를 관리하는 메커니즘이 필요 → '세션 관리'
-세션 관리의 방법
≫HttpSession 클래스를 이용한 세션 처리 방법
≫Cookie 클래스를 이용한 쿠키 처리 방법

7.3 세션 관리(Session Tracking) - 3

-7.3.1 HttpSession 클래스를 이용한 세션 처리 방법
-세션(Session)이란 사용자의 상태 정보를 서버에서 관리하는 메커니즘을 의미
-세션의 정보는 클라이언트가 서버에 접속해서 종료될 때까지(브라우저를 종료할 때까지) 유지 상태 정보가 서버에 저장되기 때문에 서버의 부하가 클 수 있음

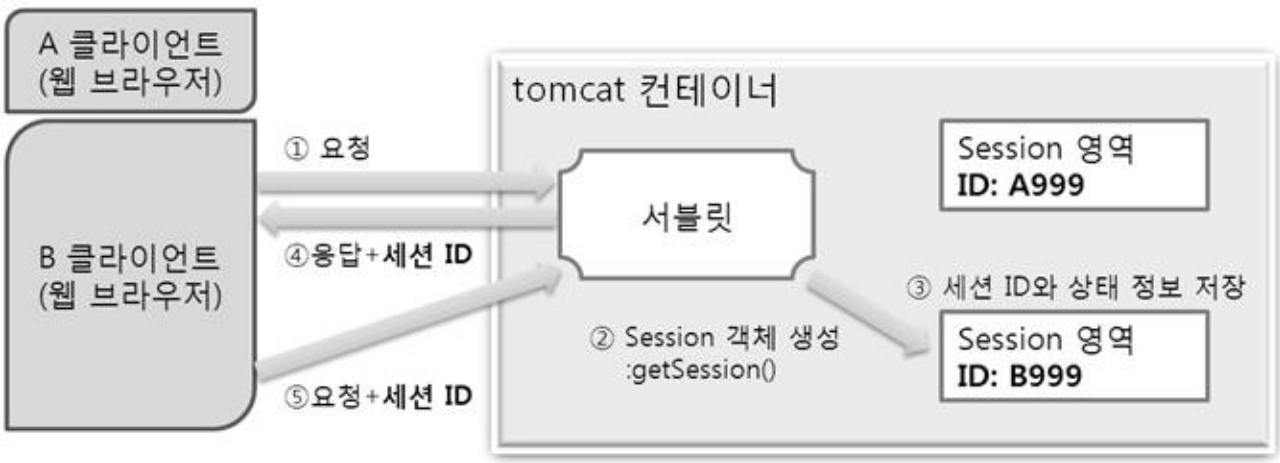
-time-out 제한을 두어 일정시간(기본 30분) 동안 요청이 없으면 서버는 세션 정보를 유지하지 않고 제거

-A 클라이언트가 서버에 요청하는 구조



7.3 세션 관리(Session Tracking) - 4

-B 클라이언트가 서버에 요청하는 구조



7.3 세션 관리(Session Tracking) - 5

-A 클라이언트(B 클라이언트)가 서블릿에 요청을 하면, 서블릿에서는 getSession( ) 메서드를 사용하여 session 영역을 생성

-A 클라이언트의 고유 식별 값인 세션 ID를 생성하여 session 영역에 저장

-일반적으로 session 영역에는 장바구니 정보 및 로그인 정보 등을 저장

-서블릿의 실행 결과가 응답 처리될 때 자동으로 세션 ID 값이 포함되며, 동일한

브라우저에서 재요청이 발생되면 세션 ID 값을 포함하여 요청처리
-서버는 재요청에 포함된 세션 ID 값을 이용하여 클라이언트와 연결 기능을 유지 가능
-A 클라이언트가 일정시간(기본 30분) 동안 요청을 하지 않으면 서버는 클라이언트 정보를 제거할 목적으로 session 영역을 삭제
-로그아웃 같은 기능을 구현하기 위해서 session 영역을 즉시 삭제 가능

7.3 세션 관리(Session Tracking) - 6

-세션 사용 시 많이 사용하는 메서드 목록

메서드명	설 명
request.getSession( )	•session 영역을 얻을 때 사용하며 session 영역이 없으면 새로 생성하고, 있으면 생성된 영역을 참조 •request.getSession(true) 메서드와 동일한 기능 •일반적으로 세션을 처음 생성하는 서블릿에서 지정
request.getSession(false)	•session 영역을 얻을 때 사용하며 session 영역이 없으면 null 값을 리턴하고, 있으면 생성된 영역을 참조 •일반적으로 세션을 사용하는 서블릿에서 주로 사용
sess.getId( )	생성된 세션 ID 값을 반환
sess.setAttribute(key,value)	•session 영역에 속성(Attribute) 값을 설정 •Session scope를 따르기 때문에 브라우저를 종료하기 전까지 사용 가능 •단, 기본으로 30분 이상 요청이 없을 시 제거
sess.getAttribute(key)	session 영역에 저장된 속성 값을 반환
sess.removeAttribute(key)	session 영역에 저장된 속성을 제거
setMaxInactiveInterval(sec)	지정된 시간만큼 세션을 유지. 단위는 초(second).
getMaxInactiveInterval()	세션의 유지시간을 반환
sess.invalidate( )	즉시 session 영역을 제거. 로그아웃 기능 시 사용

7.3 세션 관리(Session Tracking) - 7

-세션을 제거하는 방법

▸time-out을 지정하여 제거하는 방법

≫setMaxInactiveInterval(second) 메서드 사용

```
session.setMaxInactiveInterval( 60*60*24 ); //24시간 유지
```

≫web.xml 사용

```
<web-app>
    <!-- 단위는 분(minute) -->
    <session-config>
        <session-timeout>60</session-timeout>
    <session-config>
</web-app>
```

7.3 세션 관리(Session Tracking) - 8

-즉시 제거하는 방법

▸invalidate( ) 메서드 사용

```
session.invalidate( ); // 즉시 제거
```

-세션에 저장된 특정 속성 값을 제거하는 방법

▸removeAttribute(name) 메서드 사용

```
session.removeAttribute(name); //name 값에 해당되는 속성 값만 제거
```

7.3 세션 관리(Session Tracking) - 9

-7.3.2 Cookie 클래스를 이용한 쿠키 처리 방법

-쿠키는 사용자의 상태 정보를 클라이언트에서 관리하는 메커니즘을 의미

-클라이언트에 정보가 저장되기 때문에 서버의 부하가 크지 않지만 보안에 매우 취약

-쿠키는 웹 사이트의 도메인당 300개까지 저장이 가능

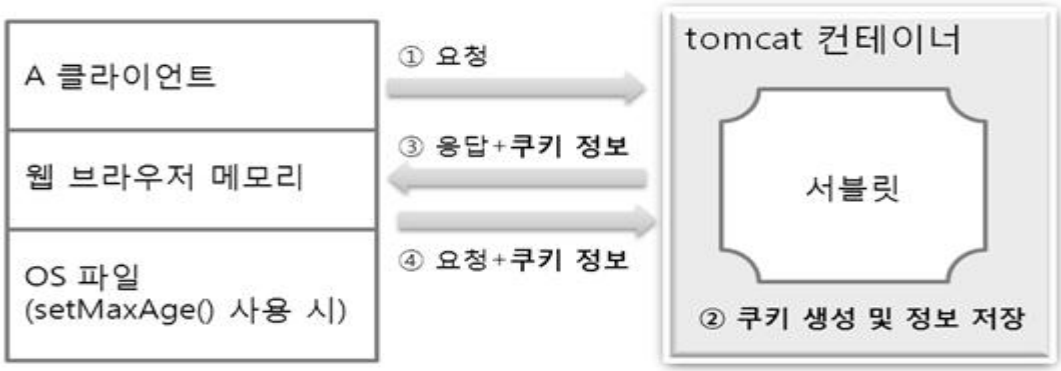
-클라이언트에서 쿠키를 사용하지 못하도록 설정할 수 있기 때문에 제약이 있음

-쿠키는 클라이언트의 브라우저 메모리 및 OS 파일에 저장 가능

-기본 저장은 브라우저 메모리이기 때문에 브라우저를 종료하면 자동으로 쿠키 정보도 제거

-쿠키 만료시간은 setMaxAge(sec) 메서드를 사용하며 지정된 시간까지 OS 파일에 저장

7.3 세션 관리(Session Tracking) - 10



생성자 및 메서드명	설 명
Cookie(name, value)	쿠키 생성시 사용되는 생성자
response.addCookie(cookie)	생성된 쿠키를 응답처리
request.getCookies( )	클라이언트로부터 쿠키 정보를 배열로 리턴
setMaxAge(second)	쿠키 만료시간을 설정
getMaxAge( )	쿠키 만료시간을 얻음
getName( )	쿠키 이름을 얻음
setValue(value)	쿠키 값을 설정
getValue( )	쿠키 값을 얻음
length	쿠키 개수를 리턴

7.4 파일 업로드 및 다운로드 기능 – 1

-7.4.1 @MultipartConfig 어노테이션을 이용한 파일 업로드

-가장 많이 사용되고 알려진 것은 아파치 그룹에서 제공하는 Commons FileUpload 라이브러리

-Spring 프레임워크 및 Struts2 프레임워크 같은 유명한 프레임워크에서도 사용되는 매우 안정적인 라이브러리

-하지만 서블릿 3.0 버전부터는 @MultipartConfig 어노테이션과 javax.servlet.http.Part 인터페이스를 사용하여 보다 쉽게 파일 업로드 기능을 구현 가능

-@MultipartConfig 어노테이션에서 사용 가능한 속성 목록

속 성	설 명
maxFileSize	업로드 파일의 최대 크기 값. 기본 값은 -1(크기 제한 없음)
maxRequestSize	HTTP 요청의 최대 크기 값. 기본 값은 -1(크기 제한 없음)
location	파일 저장 경로. 파일은 Part의 write 메서드가 호출될 때 저장됨

7.4 파일 업로드 및 다운로드 기능 – 2

-사용 가능한 Part 인터페이스의 메서드 목록이며, 폼 태그 요청의 모든 속성은 Part로 변환되어처리

메서드	설 명
String getName( )	•HTML 태그의 폼 태그 이름을 리턴 •태그명이 파트의 이름이 됨
String contentType( )	파일의 contentType을 리턴
Collection getHeaderNames( )	Part의 모든 헤더명을 리턴
getHeader(name)	설정한 헤더의 값을 리턴
write(path)	업로드한 파일을 출력
delete( )	파일과 임시 파일을 삭제
InputStream getInputStream( )	업로드한 파일의 내용을 InputStream객체로 리턴

-다음은 사용자가 파일을 업로드 한 경우의 Part의 헤더 정보 헤더 정보에서 문자열을 파싱하여 업로드한 파일명을 얻을 수 있음

```
content-type: 값
content-disposition:form-data; name="폼태그명"; filename="파일명"
```

7.4 파일 업로드 및 다운로드 기능 - 3

-7.4.2 파일 다운로드

-파일 업로드 기능을 하는 라이브러리는 제공되지만 다운로드 기능을 구현한 라이브러리는

없기 때문에 직접 프로그래밍 작업으로 처리해야 함
-구현 방법은 다운로드에서 사용했던 Content-Disposition 응답 헤더를 추가하고, 값을 'attachment; filename=파일명' 형식으로 지정
-마지막으로 OutputStream 객체를 사용하여 웹 브라우저로 출력