01장. 오리엔테이션 &스프링 소개_ JAVA 소개

☞ **Java의 역사**

- 썬 마이크로 시스템즈 제임스 고슬링 팀(1991~1995)
- Oracle이 썬 마이크로시스템즈 인수(2009)
- Oracle과 구글 java API 사용 관련 분쟁(2012)
- Oracle JDK9Java 브라우저 플러그인(Applet)을 중단(2016)
- Oracle이 Java EE를 Eclipse Foundation에 제출
- 상표 문제 발생 (2017)
  Java EE는 javax 패키지를 사용했는데,
  Oracle이 javax 및 java 상표를 Eclipse가 사용하는 것을 허락하지 않음
- Jakarta EE로 명칭 및 패키지명 변경
- Jakarta EE는 버전 8부터 지원(스프링6.0)

☞ **Java의 특징**

- 자바는 컴파일 언어
- JVM(OS)은 인터프리터와 다르다(OS의 특징을 가짐)

- 자바 가상 머신 (JVM)
  JVM은 플랫폼 독립적
  Java로 개발한 프로그램을 컴파일하여 만들어지는 바이트코드를 실행시키기 위한 가상머신

- Write Once, Run Anywhere
  JVM을 통해 가능한 것
  JVM이 실행 가능한 환경이라면 어디서든 Java 프로그램이 실행될 수 있도록 한다.

01장. 오리엔테이션 &스프링 소개_ 스프링 소개

☞ **Spring Framework**

- 자바 애플리케이션 개발을 편하게 해주는 오픈소스 **프레임워크**
  (개발을 할 때 사용하는 공통적 작업을 위한 구조화 된 틀 또는 체계)
- **뜻** : ejb의 복잡성을 극복, 탄력적으로 사용할 수 있음
- **장점**: 개발이 편해 단순 명료, 결과물이 기업형급

☞ **Spring Framework의 특징**

- POJO 지원 (클래스만으로 구성해도 ㄱㅊ)
- 의존성주입(DI) 지원
- 공통 모듈을 재사용하는 AOP 지원
- 일관성 있는 모듈의 트랜잭션 지원
- 전자정부 표준프레임워크의 기반 기술

☞ **Spring Modules(Spring Project)**

모듈	설명
Framework (Core)	•핵심 기능 제공, •컨테이너 및 의존성주입 프레임워크 (Spring Beans, MVC, Web 등)
WebFlux	•Web Framework의Reactive Stack •완전한 비차단 지원 (데이터를 읽고 쓰는 I/O 처리 시 스레드가 대기하지 않고 다른 작업 이어서 실행, 비동기적 처리 방식으로 구현, 이벤트루프와 같은 구현체)
Boot	• 독립 실행형 Spring 앱 지원 • 내장 서블릿 컨테이너 사용 (Tomcat, Jetty, Undertow) • 빌드구성을 단순화하기위해 스타터종속성을 사용 • Spring 타사라이브러리를 자동 구성 • XML 구성이필요하지않음
Data	•데이터저장소(DB)와 상호작용하는 코드를 Java 인터페이스로 정의 •명명 규칙에 따라 데이터 검색/저장 기능이 자동 생성 •여러 종류의 데이터베이스와 연동 관계형(SQL, MySQL) 문서형(MongoDB) 그래프(Neo4j)
Security	•강력한 보안 프레임워크 •인증 및 권한 부여 기능을 제공 •API 및 앱의 보안 관련 작업 처리 (로그인, 암호화, 역할 관리 등)
Integration	• 메시징을 통해 내/외부 애플리케이션 간 통합을 제공
Batch	•대량 데이터 처리를 위한 배치(Batch) 프레임워크 • 반복 작업 자동화, 효율적 실행 로깅/추적 & 트랜잭션 관리 작업 처리 통계 & 작업 재시작 건너뛰기 및 리소스 관리
Cloud	•클라우드 기반 애플리케이션 개발을 지원 •분산 시스템을 쉽게 만들고 관리할 수 있는 도구를 제공 (구성관리, 서비스검색, 회로차단기, 지능형라우팅 등)
Native	Spring Boot 프로젝트를 네이티브 실행파일로 컴파일 할 수 있게 지원

02장. 스프링 웹 MVC 프로젝트의 구조
_ 스프링 웹 MVC 소개

☞ **스프링 웹 MVC**

- 스프링이 제공하는 웹 개발 전용 프레임워크
- 모델(Model)-뷰(View)-컨트롤러(Controller) 패턴

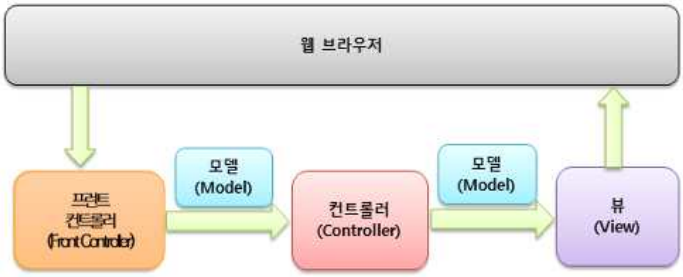
  모델: 데이터가 들어있는 객체

  뷰: 모델의 정보(데이터)를 특정 형식으로 표현(JSP등)

  컨트롤러: 비즈니스 로직을 포함하는 자바 클래스

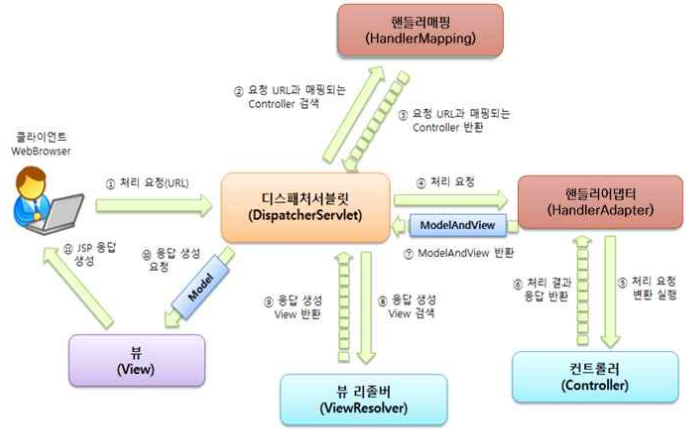
  @Controller

  프론트 컨트롤러: 흐름을 관리, DispatcherServlet



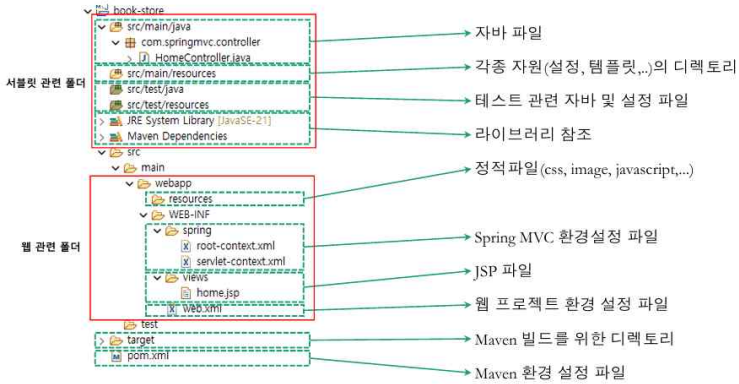
☞ **스프링 웹 MVC의 기본 구성 요소**

구성요소	설명
디스패처 서블릿	웹으로부터 요청을 전달 받음 요청을 컨트롤러에게 전달 컨트롤러의 반환한 값을 뷰에 전달 웹브라우저에 출력
핸들러 매핑	클라이언트의 요청 URL을 어떤 컨트롤러가 처리할지 결정
핸들러 어댑터	핸들러 매핑 클래스에 의해 결정된 컨트롤러 호출
컨트롤러	클라이언트 요청 처리, 결과 반환 데이터를 모델에 담아 뷰로 전달
모델&뷰	컨트롤러가 처리한 결과 정보와 뷰 선택에 필요한 정보를 담는 객체
뷰 리졸버	컨트롤러의 처리 결과를 출력할 뷰 결정
뷰	컨트롤러의 처리 결과 화면 생성 JSP 등 다양한 뷰 템플릿 엔진 클라이언트에 요청 처리 결과를 전송



02장. 스프링 웹 MVC 프로젝트의 구조

p.27 참조



02장. 스프링 웹 MVC 프로젝트의 구조
_ 웹 프로젝트 환경 설정 파일

☞ web.xml

- 웹 프로젝트의 배치 설명자/ 기술서
- 배포되는데 이용되는 XML 형식의 자바 웹 애플리

이션 환경설정 담당

- src/main/webapp/WEB-INF
- Servlet 3.0 이상: Annotation으로 대체가능

02장. 스프링 웹 MVC 프로젝트의 구조
_ Spring MVC 환경 설정 파일

☞ **서블릿컨텍스트(servlet-context.xml)**
HandlerMapping과 ViewResolver를 설정

• 자바클래스의 Bean 객체 설정
1. <component-scan>요소 사용
- 스프링 MVC에서 사용할 Bean 객체를 XML에 등록하지 않고 설정된 패키지 하위 경로의 모든 클래스를 검색하여 자동 등록(필요한 애너테이션을 자동 인식)

```
<context:component-scan
base-package ="com.springmvc.*" />
```

2. <bean>요소 사용
- 애너테이션이 선언된 컨트롤러 및 의존 관계가 있는 자바 클래스를 <bean> 요소로 빈 객체를 일일히 등록

```
<beans:bean class =
"com.springmvc.controller.HomeController"/>
```

• 정적 리소스 설정
- 정적리소스
  브라우저의 요청 리소스가 이미 만들어져 있어 그대로응답 가능한 자원
  별도의 서버처리 없이 그대로 응답 가능

```
<resources mapping ="/resources/**“
location ="/resources/" />
```

- resources 요소의 속성
  서버에서 앞서 처리될 필요가 없는 정적 리소스 파일을 처리하는 역할
  물리적 경로를 설정하고 정적 리소스 파일을 저장하면 소스코드나 웹 주소창에서 해당 리소스의 경로로 직접 접속 가능

속성	설명
mapping	웹 요청 URL 경로 패턴 설정 컨텍스트 경로를 제외한 나머지 부분의 경로와 매핑
location	실제 요청 경로에 해당하는 리소스 위치 설정 위치가 여러 곳이면 쉼표로 구분
cache-period	웹 브라우저에 캐시 시간 관련 응답 헤더를 전송 초 단위로 캐시 시간 지정 값 0: 캐시 X, 항상 새로 가져옴 값없음: 캐시 관련 응답 헤더 전송X

캐시: 웹 브라우저가 같은 데이터를 여러 번 다운로드하지 않고 미리 저장해 두었다가 다시 사용할 수 있도록 하는 메커니즘

• 뷰 매핑 설정
컨트롤러가 어떤 뷰 이름을 반환 할지 결정하는 과정, 뷰 리졸버가 그 이름을 처리
뷰 리졸버: 컨트롤러에서 설정한 뷰 이름으로 실제 사용할 뷰를 선택하는 객체
- 속성

prefix
컨트롤러가 반환한 View 이름 앞에 붙일 접두사
suffix
컨트롤러가 반환한 View 이름 뒤에 붙일 접미사

```
<beans:bean class="org.springframework.web.servlet.view.I
 <beans:property name="prefix" value="/WEB-INF/views/" />
 <beans:property name="suffix" value=".jsp" />
</beans:bean>

@RequestMapping(value="/", method=RequestMethod.GET)
public String home() { ...
    return "home";
}
```

접두사 + View 이름 + 접미사
실제 View의 위치 /WEB-INF/views/home.jsp

02장. 스프링 웹 MVC 프로젝트의 구조 _ Maven 설정

☞ **Maven Scope**
- 종속성범위(Dependency scope)
종속성의 전이성을 제한하고 종속성이 클래스 경로에 포함되는 시점을 결정
pom.xml-<dependencies>안-<dependency>로 정의

범위	설명
compile	기본 범위, 전체 클래스 경로에서 사용됨. 종속 프로젝트로 전파됨.
provided	컴파일/테스트에만 사용, 런타임에는 제외됨. JDK나 컨테이너가 제공.
runtime	실행 시 필요, 컴파일 시에는 불필요.
test	테스트 시에만 사용됨. 일반 컴파일 시에는 불포함
system	provided와 유사 하지만 명시적으로 포함된 JAR를 제공 항상 사용 가능 Maven Repository 사용 안 함
import	<dependencyManagement>POM파일에서 종속성 버전을 관리하는 섹션 모든 모듈에서 동일한 종속성 적용 부모 POM의 종속성 목록을 가져와, <dependencyManagement>섹션의 종속성 목록을 대체

03장. 스프링 웹 MVC 계층적 구조_ 계층적 구조

☞ **계층적 구조**

논리적으로 구분된 여러 레이어(계층)로 설계
각 계층은 자신의 역할만 수행하며, 다른 계층과 느슨하게 결합(Low Coupling)됨.
각 계층이 독립적, 기능 변경 시 다른 계층에 영향을 주지 않음

도매인 객체	객체 정보를 저장하는 데이터모델 (RDBMS, NoSQL 등)
퍼시스턴스 계층 (=데이터액세스)	DB/파일에 접근해 데이터 처리 (Repository, DAO)
서비스 계층 (=비즈니스로직)	핵심 비즈니스 로직을 수행 퍼시스턴스 계층을 호출 프레젠테이션-퍼시스턴스 연결 (Service)
프레젠테이션 계층 (=표현계층)	애플리케이션과 사용자의 최종 접점 데이터를 입력받거나 결과를 웹서버에 전달 사용자의 요청을 받아 처리하고 결과를 전달해 사용자에게 보여줌 (Controller)

- 장점

✔ 유지보수 용이 → 특정 기능을 수정할 때 다른 부분에 영향을 주지 않음.
✔ 재사용성 증가 → 동일한 기능을 여러 곳에서 사용할 수 있음.
✔ 확장성 확보 → 새로운 기능 추가가 쉬움.
✔ 코드 가독성 향상 → 역할별로 코드가 분리되어 이해하기 쉬움.
✔ 보안 강화 → 데이터를 다루는 부분과 사용자 인터페이스가 분리됨.

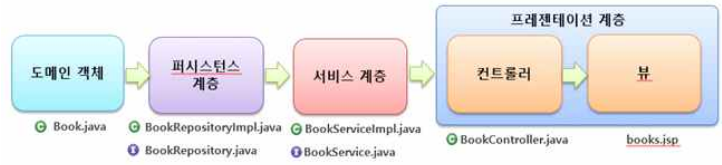
☞ **Layer** vs **Tier**

항목	Layer(레이어)	Tier(티어)
의미	논리적 계층	물리적 계층
분리 방식	개념적 분리 설계와 코드구조 기준	물리적 서버/네트워크 기준
종속성	같은 서버 내 논리적으로 상호작용	네트워크를 통해 통신
예시	프레젠테이션 비즈니스로직 데이터액세스 레이어	프레젠테이션티어 어플리케이션티어 데이터베이스티어

Presentation Tier: 사용자 요청을 처리하는 클라이언트.
  - 예: 웹 브라우저, 모바일 앱.
Application Tier: 비즈니스 로직을 처리하는 서버.
- 예: Spring 애플리케이션이 실행되는 서버.
Database Tier: 데이터 저장/관리 데이터베이스 서버
- 예: MySQL, PostgreSQL 등

☞ **계층적 구조를 적용한 구현 과정**

- 먼저 계층간 데이터를 전달하기 위한 데이터를 정의
- 데이터베이스가 이미 완성되어있다는 전제
도메인객체→ 퍼시스턴스→ 서비스→ 프레젠테이션 순



☞ **MVC를 담당하는 프레젠테이션 계층**

컨트롤러:
  웹에서 들어오는 요청을 처리하는 자바 클래스
  @Controller를 선언
뷰:
  웹 요청의 처리 결과를 보여주는 JSP웹페이지
  JSTL(자바 서버 표준 태그 라이브러리) 적용
모델: JSP 웹페이지에 출력할 데이터
<코드 보기>
BookController.java
books.jsp

04장. 데이터베이스 연동_ 데이터베이스설치

☞ **Spring JDBC**

JDBC 추상화를 제공하는 Spring Framework의 모듈
<Spring의 역할>
데이터베이스 커넥션 연결
JDBC Statement 준비 및 실행
SQL 실행 결과 반복 및 처리
예외 처리
트랜잭션 관리
데이터베이스 커넥션, JDBC Statement, ResultSet 종료 및 회수
<개발자의 역할>
데이터베이스 접속 파라미터 정의
SQL 문장 정의
SQL 파라메터 및 파라메터 값 정의
SQL 실행 결과의 각 행 처리

☞ **Spring 설정- 전체 설정 내용**

코드 보고 각 설정의 의미와 역할 이해
src/main/webapp/WEB-INF/spring/root-context.xml

05장. 스프링 웹 MVC의 컨트롤러_ 컨트롤러 정의

☞ **@Controller를 이용한 컨트롤러 정의**

- 컨트롤러

MVC패턴에서 Controller 역할을 담당하는 클래스
@Controller 클래스 수준에서 선언
웹 브라우저에서 들어온 요청을 처리하는 메소드를 포함하고 있는 Java 클래스

☞ **<context:component-scan>요소로 컨트롤러 등록**

- 스프링 MVC 설정 파일(servlet-context.xml)에 빈 클래스로 등록

@Controller가 선언된 컨트롤러를 빈 객체로 자동 등록
자동 의존성 관리

기본 패키지 명 하위에 있는 컨트롤러 및 의존 관계에 있는 자바 클래스를 검색하여 빈 객체로 자동 등록

```
<context:component-scan
base-package ="기본 패키지 명"/>
```

context:component-scan

```
<context:component-scan base-package="com.springmvc.*" />
</beans:beans>
```

수동 설정

```
<beans:bean
class="com.springmvc.controller.HomeController" />

<beans:bean id="bookRepository"
class="com.springmvc.repository.BookRepositoryImpl" />

<beans:bean id="bookService"
class="com.springmvc.service.BookServiceImpl">
  <beans:property name="bookRepository"
ref="bookRepository" />
</beans:bean>

<beans:bean
class="com.springmvc.controller.BookController">
  <beans:property name="bookService" ref="bookService" />
</beans:bean>

</beans:beans>
```

05장. 스프링 웹 MVC의 컨트롤러_ 요청 경로 매핑

☞ **@RequestMapping**

사용자 요청을 처리하는 컨트롤러와 메소드를 매핑
클래스 및 메소드에 지정 할 수 있음

```
@RequestMapping(value="웹요청URL",
method=RequestMethod.HTTP요청방식)
```

☞ **컨트롤러에 @RequestMapping 적용**

1. 클래스 수준의 @RequestMapping

요청한 URL에 매핑되는 컨트롤러에 선언함
기본 매핑 경로를 설정하지 않고 생략 가능

2. 메소드 수준의 @RequestMapping

요청한 URL을 처리 할 컨트롤러의 메소드를 지정
method 속성의 기본값은 GET (생략가능)

```
@Controller
@RequestMapping("/books")
public class BookController {
    ...

    @RequestMapping("/all")
    public String requestAllBooks(Model model) {
        List<Book> list = this.bookService.getAllBookList();
        model.addAttribute("bookList", list);
        return "books";
    }
```

매핑되는 URL= 클래스 수준 value/메소드 수준 value

http://localhost:8080/books/all

05장. 스프링 웹 MVC의 컨트롤러_ 요청 처리 메소드와 모델

☞ **요청처리 메소드와 모델**

@RequestMapping의 요청 경로 매핑 메소드 호출

- 모델과 뷰

모델

사용자의 요청을 처리한 결과 데이터를 관리하고 전달

뷰

결과 데이터를 웹브라우저에 출력하는 웹페이지 역할

클래스	설명
Model	데이터를 저장하고 반환 받는 단순한 메소드를 제공하는 모델
ModelMap	Map 인터페이스를 통해 데이터를 저장하고 반환 받을 수 있는 모델
ModelAndView	모델과 뷰 정보를 모두 관리할 수 있는 모델

☞ **ModelAndView 클래스**
- 모델과 뷰 이름을 합쳐 놓은 것
- 처리 결과를 addObject() 메서드에 담아 전달
- setViewName() 메서드로 뷰 이름 설정

1. 사용자 요청 처리 결과 전달
```
public ModelAndView addObject (
  String attributeName,
  @Nullable Object attributeValue)
```

역할: 지정된 이름으로 제공된 속성을 등록
매개변수:
- attributeName: 속성의 이름(null이 될 수 없음)
- attributeValue: 속성의 값(null이 될 수 있음)

2. 뷰 이름 설정
```
public void setViewName (
  @Nullable String viewName)
```

역할: ModelAndView를 위한 뷰 이름을 설정
매개변수: viewNam - 뷰 이름

06장. 요청처리 메소드의 파라미터 유형
_ 경로 변수와 @PathVariable

☞ **경로변수(path variables)**
- 웹 요청 URL에 포함된 파라미터 값을 전달 받는데 사용하는 변수
- @RequestMapping에 중괄호({변수명})를 사용하여 웹 요청 URL에 포함된 요청 조건 값을 전달

경로 변수는 URL에서 값을 추출하여 요청 처리에 활용
@PathVariable을 사용하여 동적으로 값을 받아
데이터베이스 조회 등 처리 가능
모델에 데이터 추가 후 뷰로 전달하여 화면에 출력

06장. 요청처리 메소드의 파라미터 유형
_ 매트릭스 변수와 @MatrixVariable

☞ **매트릭스변수(matrix variables)**
- 요청 URL에 포함 된 파라미터 값을 전달 받는데 사용
- 세미콜론(;)으로 데이터를 구분. 다중 데이터를 전달 받음
- @RequestMapping의 경로 변수에 '매트릭스_변수=값' 형태로 사용
- 매트릭스 변수가 여러 개 일 경우: 콤마(,)로 구분. 변수 이름을 반복

☞ **@MatrixVariable**
- 매트릭스 변수는 기본적으로 Spring에 의해 URL에서 제거됨
- 매트릭스 변수를 사용하려면
  해당 변수의 위치를 URL 패턴에서 URI 변수로 선언
  즉, 경로 변수(@PathVariable) 안에서 매트릭스 변수를 처리해야 함

- URL에서 매트릭스 변수 제거를 막는 방법
1. servlet-context.xml의 <annotation-driven> 요소에 enable-matrix-variable=true를 설정
2. enable-matrix-variable 요소를 true로 설정하면 매트릭스변수는 URL에서 제거되지않음
3. 요청 매핑 패턴은 매트릭스 변수가 예상 되는 경로 세그먼트에 서 URI 변수로 사용 해야함

```
@GetMapping ("/filter/{bookFilter}")
  public String requestBooksByFilter(
    @MatrixVariable (pathVar="bookFilter")
    Map <String, List <String>> bookFilter
  ){ //bookFilter에서 매트릭스 변수 추출 가능
}
```

☞ **실습 코드 보기**
매트릭스변수값과일치하는도서목록출력
BookController.java에 requestBooksByFilter() 메소드 구현
동작 테스트 하고 안되는 이유 정리

06장. 요청처리 메소드의 파라미터 유형
_ 요청 파라미터와 @RequestParam

☞ **요청 파라미터 (request parameters)**

- 클라이언트가 웹 서버로 데이터 전달 시 URL에 포함된 파라미터

  요청 파라미터는 URL에 포함되어 서버가 처리하고 필요한 데이터를 반환하도록 함

http://localhost:8080/books?category=IT&author=길벗

- 파라미터 구성

파라미터 이름과 값은 =로 구분

  이름: category 값: IT

파라미터가 여러 개일 때 &로 구분

  category=IT&author=길벗

☞ **@RequestParam**

- 요청 매핑 경로에 포함된 요청 파라미터 값을 메서드 매개변수로 전달
- 클라이언트가 URL을 통해 전달한 파라미터를 쉽게 처리할 수 있도록 지원
- URL 쿼리 파라미터를 추출하여 서드에서 사용 가능

옵션	타입	설명
defaultValue	String	요청 매개변수가 없거나 빈 값일 경우 기본값으로 대체
name	String	전달하는 요청 파라미터의 이름 지정 (value와 동일).
required	bool	요청 파라미터가 필수인지 여부 (true-필수, 없으면 예외발생)
value	String	name()에 대한 별칭

06장. 스프링 폼 태그 라이브러리_ 스프링 폼 태그 개요

☞ **스프링 폼 태그**

- 스프링 웹 MVC와 연동되는 JSP 태그 라이브러리

☞ **스프링 폼 태그의 종류**

태그 유형	설명	출력되는 HTML
<form>	폼의 시작과 끝을 나타낼 때 사용	<form>
<input>	사용자가 일반 텍스트를 입력할 수 있는 입력 필드를 만들 때 사용	<input type="text">
<checkbox>	여러 옵션 중 여러 개를 선택할 수 있는 체크박스를 만들 때 사용	<input type="checkbox" >
<checkboxes>	<form:checkbox>목록을 나타낼 때 사용	
<radiobutton>	여러 옵션 중 하나를 선택할 수 있는 라디오버튼을 만들 때 사용	<input type="radio">
<radiobuttons>	<form:radiobuutton> 목록을 나타낼 때 사용	
<password>	사용자가 텍스트를 입력하면 자동으로 *로 변환되는 비밀번호 입력 필드를만들때사용	<input type="password" >
<select>	콤보 박스나 리스트 박스를 만들 때 사용	<select>
<option>	<select>...</select> 에 포함되어 목록을 구성할 때 사용	<option>
<options>	<form:option> 목록을 나타낼 때 사용	
<textarea>	사용자가 여러 줄의 텍스트를 입력할 수 있는 입력 박스를 제공할 때 사용	<textarea>
<hidden>	웹 브라우저가 출력하지 않는 입력 폼을 만들 때 사용	<input type="hidden">
<errors>	유효성 검사(validation)에서 생긴 오류 메시지를 만들 때 사용	

☞ **스프링 폼 태그 <from> 태그**

```
<form :form속성1="값1" [속성2="값2" ...]>
  <input>,<select> 등 //다양한 입력 양식 태그
</form :form>
```

속성	설명
modelAttribute	폼 객체가 노출되는 모델 속성의 이름 기본값은'command'
action	데이터를 받아 처리하는 웹 페이지의 URL을 설정
method	데이터가 전송되는 HTTP 방식을 설정 (기본 post)
name	폼을 식별하는 이름을 설정
target	폼 처리 결과의 응답을 실행할 프레임의 이름을 설정
enctype	폼을 전송하는 콘텐츠 MIME 유형을 설정
accept-charset	폼 전송에 사용할 문자 인코딩을 설정

- 역할:
데이터를 입력받고 서버로 전송하는 최상위 태그
Spring의 모델 객체와 HTML 폼을 쉽게 연결
  → 데이터 바인딩을 자동으로 수행
입력값 검증 및 오류 메시지 표시 가능
  → BindingResult와 함께 사용
Spring MVC 환경에서 효율적으로 폼 데이터를 처리

- 제약 조건:
단독 사용 못함(<form:form> 내부에 입력 요소 포함)
1. Spring의 spring-form 라이브러리 추가 필수
  → JSP에서 <form:form>을 사용하려면 Spring 폼 태그 라이브러리를 선언해야 함:
2. 모델 객체(modelAttribute) 필요
  → 폼이 바인딩할 모델 객체를 설정해야 올바르게 동작.
3. 기본적인 HTML 폼 태그 포함
  → 내부에는 <form:input>, <form:checkbox>, <form:select> 등의 입력 요소가 필요함

- 조건
Spring MVC 컨트롤러에서 모델 객체를 설정해야 함
Spring 태그 라이브러리를 선언해야 폼 태그를 사용할 수 있음
modelAttribute를 통해 바인딩할 모델을 명확하게 지정해야 함