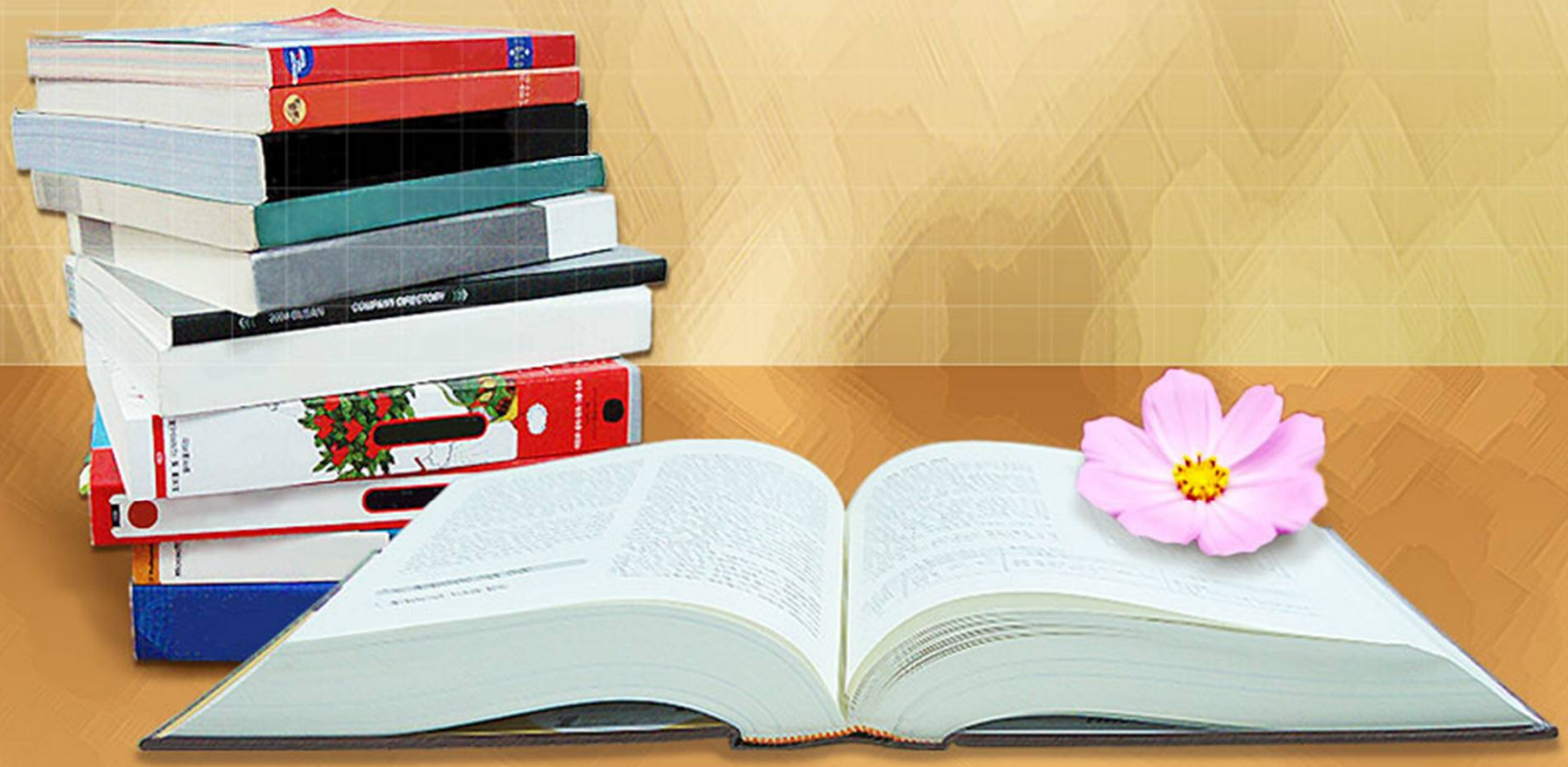
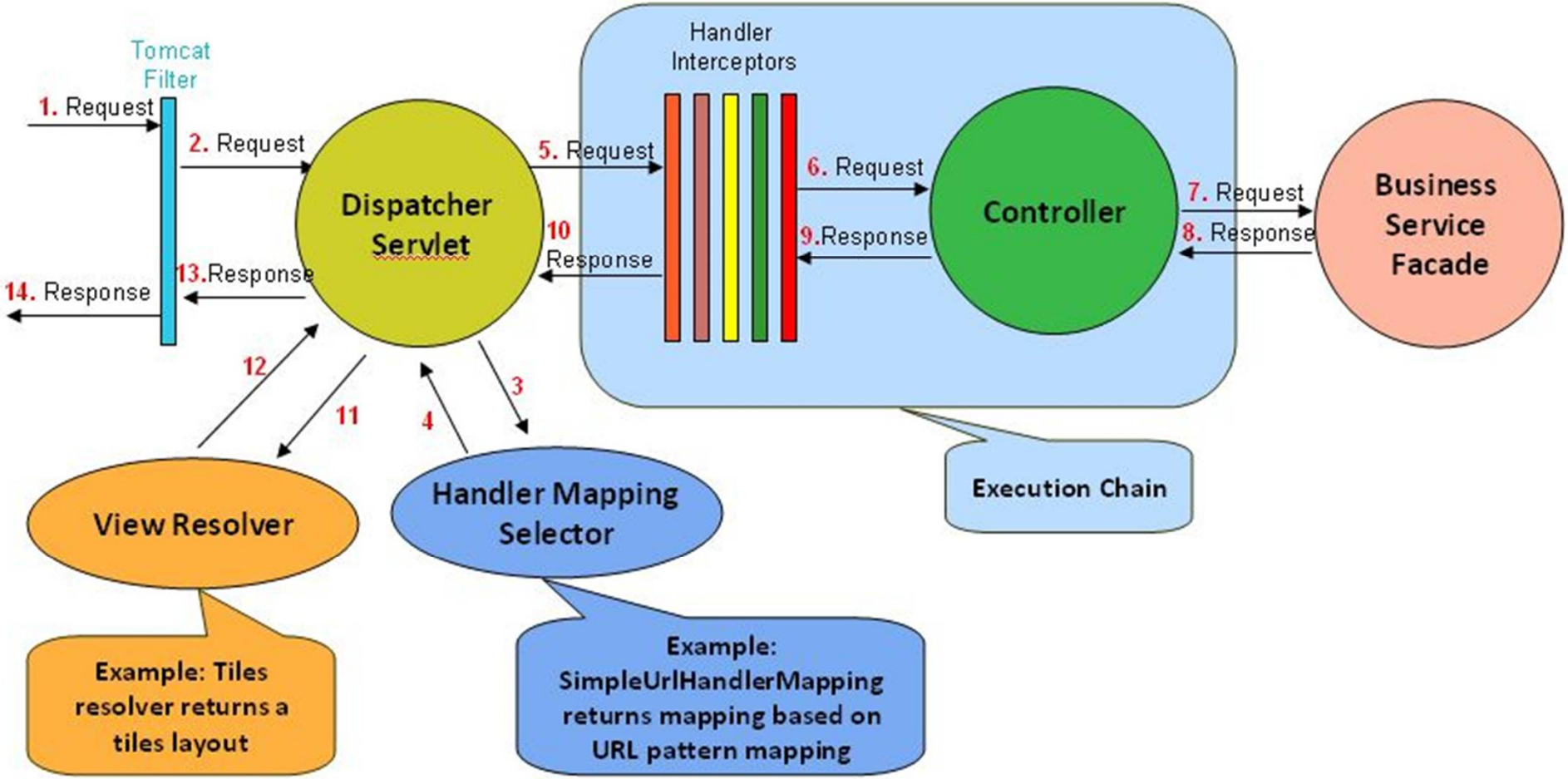
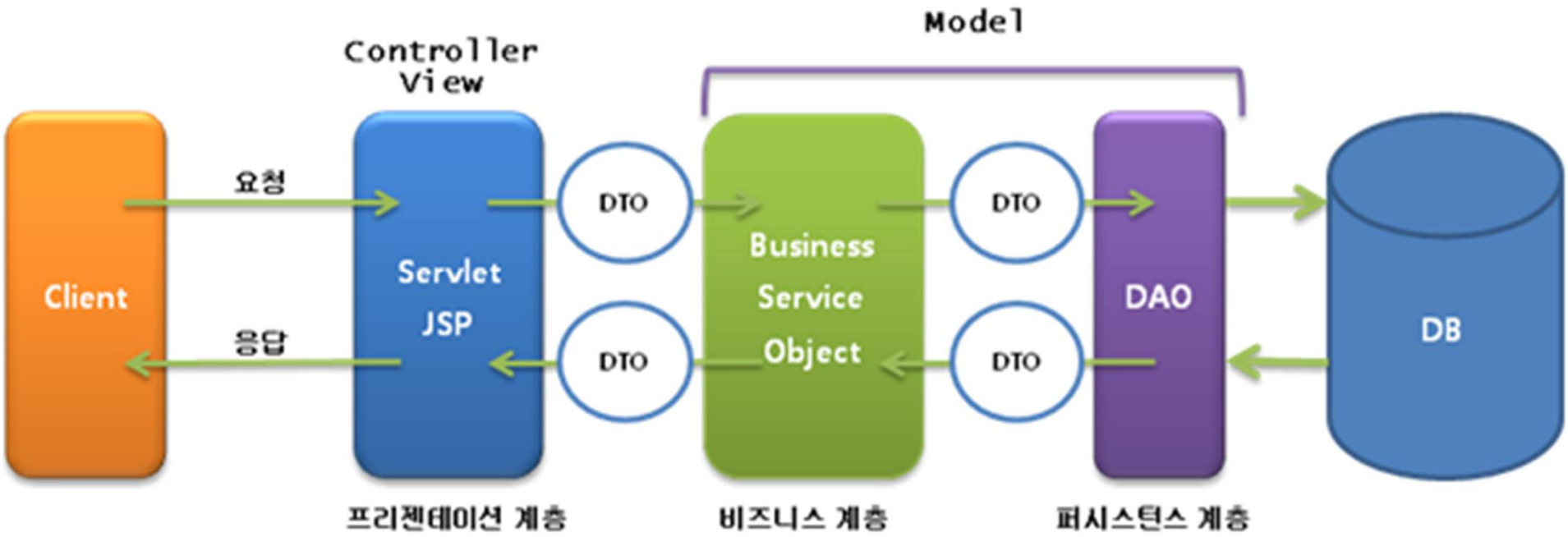
스프링 요약



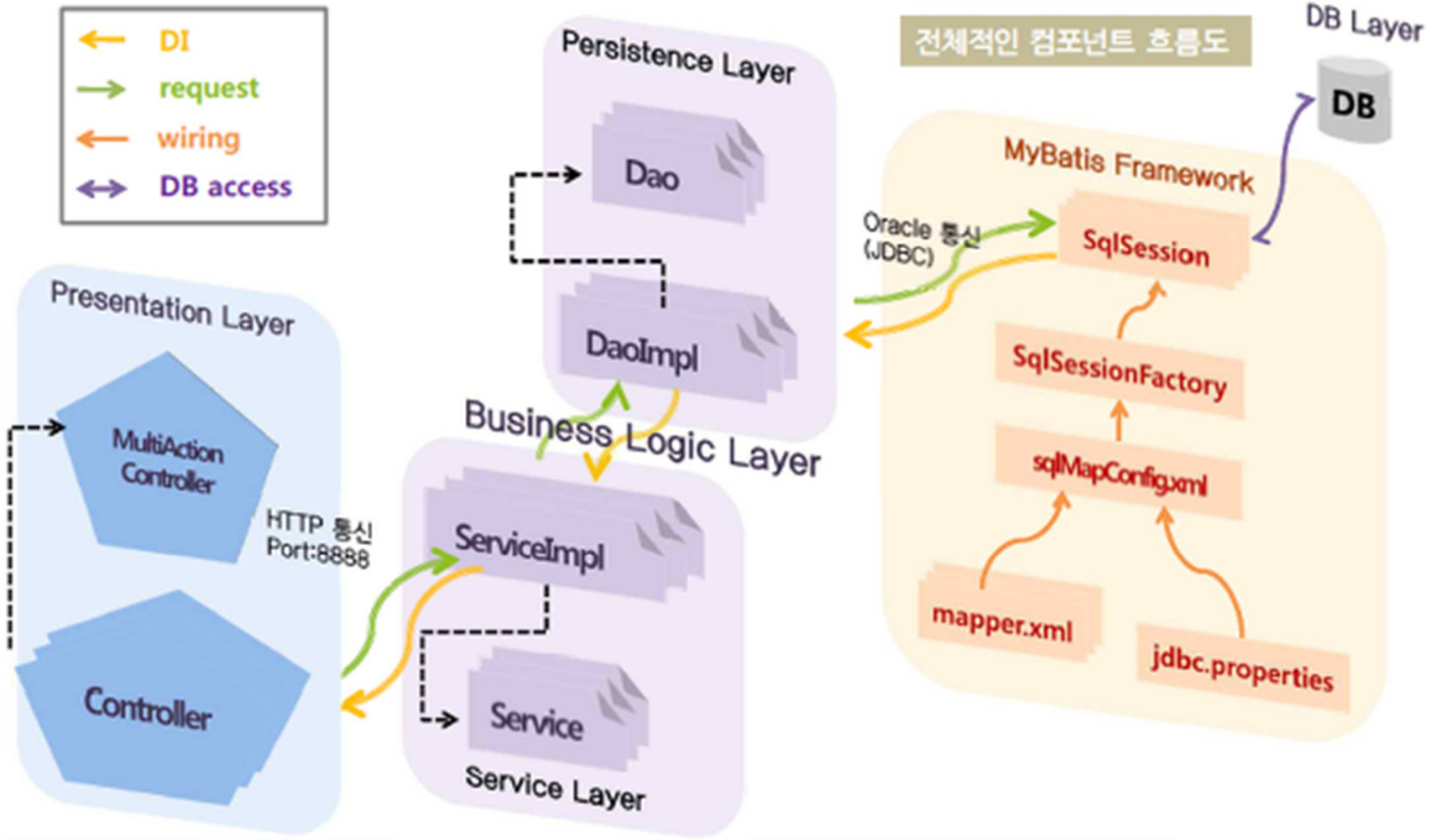
Spring MVC 요청 처리 흐름도



웹 어플리케이션 아키텍처



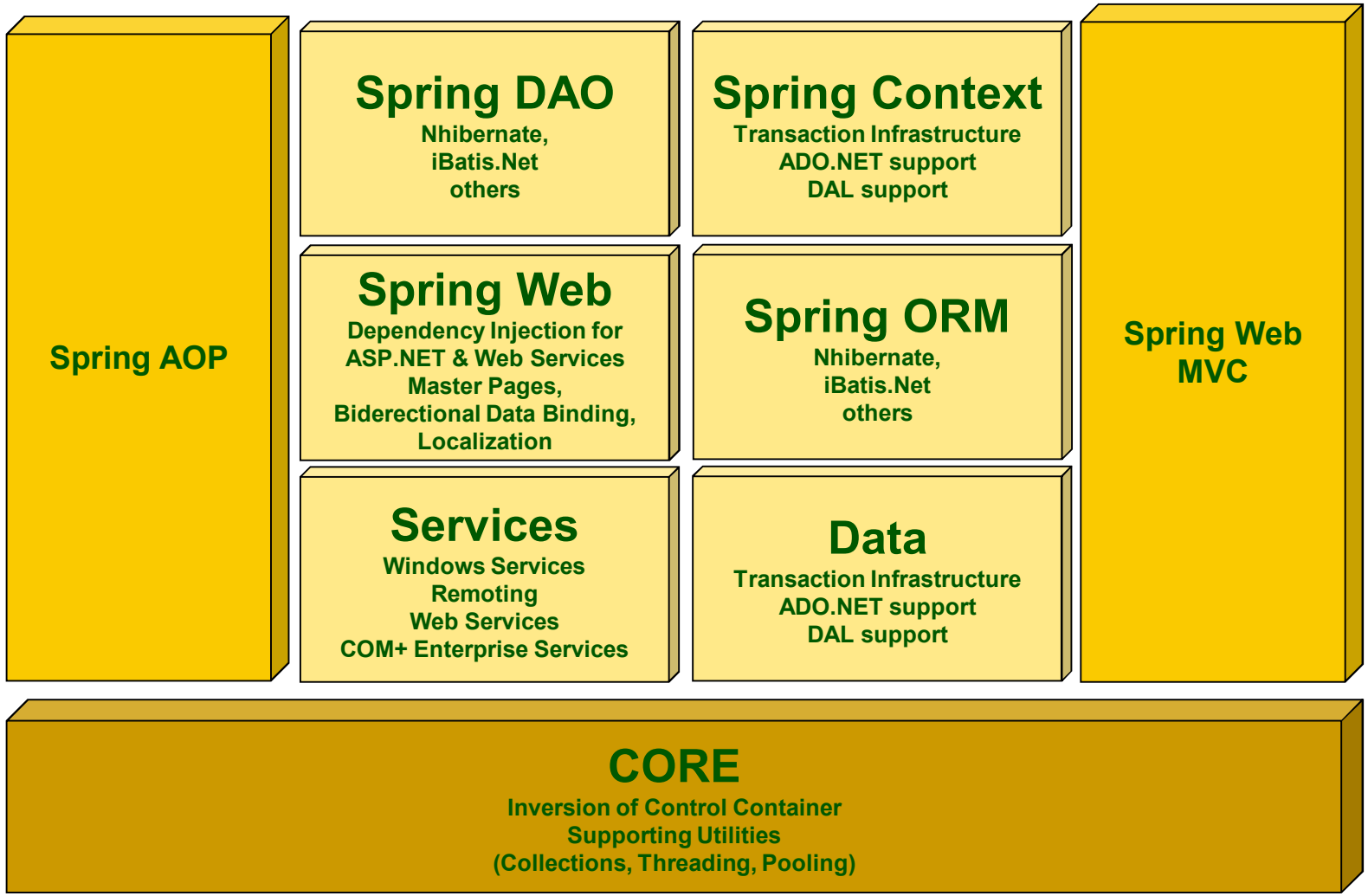
Component Flow



스프링이란?

- Servlet 이란?
- POJO 이란?
- DI 이란?
- AOP 이란?
- MVC 이란?

Spring Framework - Introduction

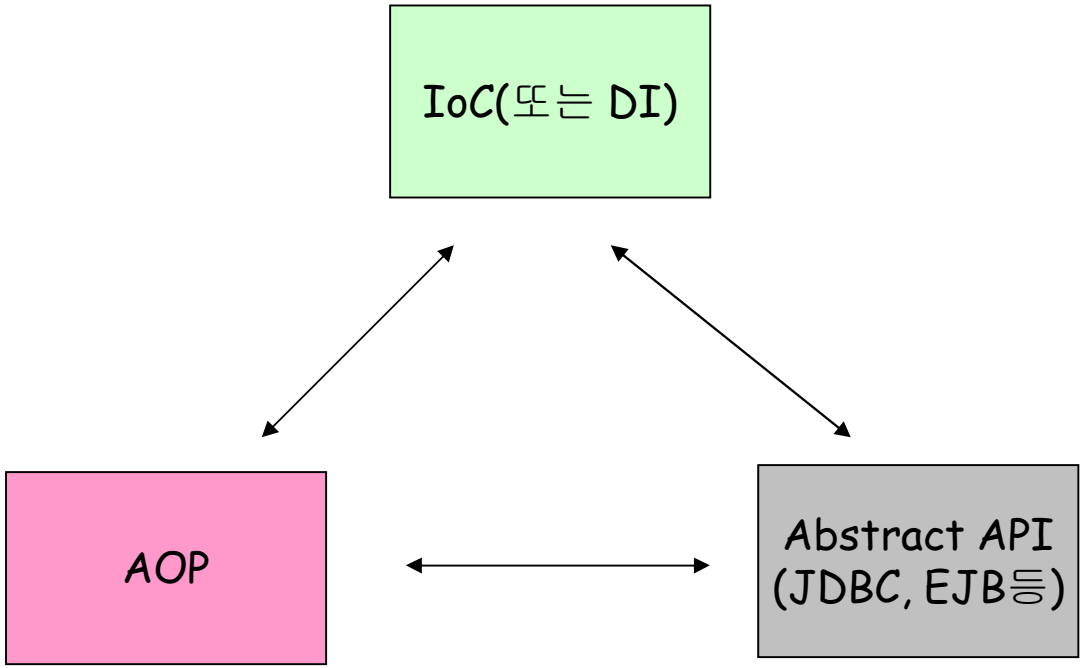


Spring Framework - Introduction

- **Spring Core** : Spring 프레임워크의 근간이 되는 IoC(또는 DI) 기능을 지원하는 영역을 담당하고 있다. BeanFactory를 기반으로 Bean 클래스들을 제어할 수 있는 기능을 지원한다.
- **Spring Context** : Spring Core 바로 위에 있으면서 Spring Core에서 지원하는 기능외에 추가적인 기능들과 좀 더 쉬운 개발이 가능하도록 지원하고 있다. 또한 JNDI, EJB등을 위한 Adaptor들을 포함하고 있다.
- **Spring DAO** : 지금까지 우리들이 일반적으로 많이 사용해왔던 JDBC 기반하의 DAO개발을 좀 더 쉽고, 일관된 방법으로 개발하는 것이 가능하도록 지원하고 있다. Spring DAO를 이용할 경우 지금까지 개발하던 DAO보다 적은 코드와 쉬운 방법으로 DAO를 개발하는 것이 가능하다.
- **Spring ORM** : Object Relation Mapping 프레임워크인 Hibernate, IBatis, JDO와의 결합을 지원하기 위한 기능이다. Spring ORM을 이용할 경우 Hibernate, IBatis, JDO 프레임워크와 쉽게 통합하는 것이 가능하다.
- **Spring AOP** : Spring 프레임워크에 Aspect Oriented Programming을 지원하는 기능이다. 이 기능은 AOP Alliance 기반하에서 개발되었다.
- **Spring Web** : Web Application 개발에 필요한 Web Application Context와 Multipart Request등의 기능을 지원한다. 또한 Struts, Webwork와 같은 프레임워크의 통합을 지원하는 부분을 담당한다.
- **Spring Web MVC** : Spring 프레임워크에서 독립적으로 Web UI Layer에 Model-View-Controller를 지원하기 위한 기능이다. 지금까지 Struts, Webwork가 담당했던 기능들을 Spring Web MVC를 이용하여 대체하는 것이 가능하다. 또한 Velocity, Excel, PDF와 같은 다양한 UI 기술들을 사용하기 위한 API를 제공하고 있다.

Spring Framework - Introduction

IoC : Inversion of Control
AOP : Aspect Oriented Programming



Spring Framework - IoC

- POJO(Plain old Java Object)란 무엇인가?
  - Servlet과 EJB와 같이 특정 API에 종속적이지 않은 모든 자바 클래스.
  - 일반적으로 우리들이 흔히 이야기하는 자바빈은 모두 POJO라고 이야기할 수 있다.

DI(Dependency Inject)

- DI란?
  - 인스턴스를 주입 하는 방법
- DI를 왜 쓰는가?
  - 결합도를 낮추기 위해서
- 결합도를 낮추면
  - 단위테스트 가능

- DI를 위해 사용되는 기술
  - 자바의 인터페이스
  - 자바의 리플렉션(Reflection, Class.forName())
- DI 설정 방식에는?
  - mxl 설정
  - 애노테이션 설정
  - 혼합 설정
- DI 주입 방식에는?
  - setter 주입
  - 생성자 주입

클래스의 관계

- 상속 관계

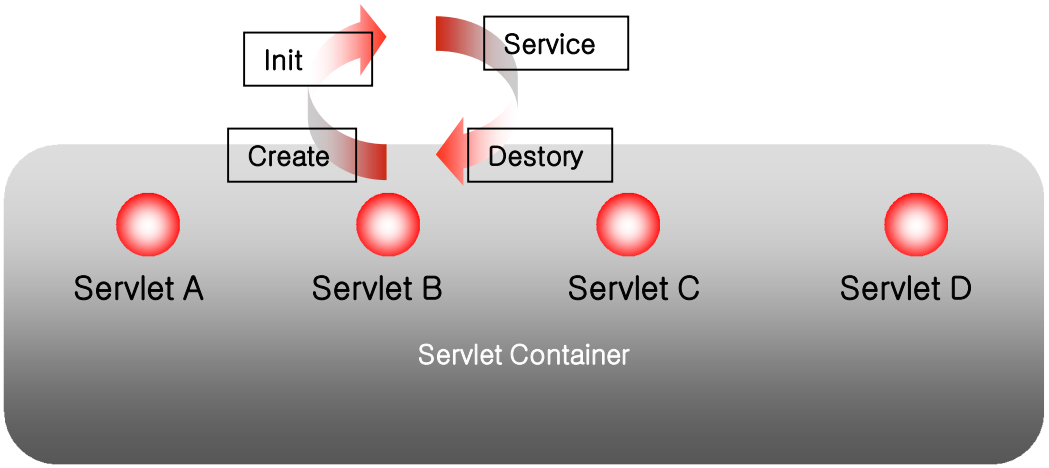
- 연관(의존) 관계

Spring Framework - IoC

- Spring은 하나의 프레임워크이다. 그런데 왜 Spring 컨테이너, IoC 컨테이너라는 말을 사용할까?
- 컨테이너란 무엇일까?
- 컨테이너 종류에는?
  - Servlet Container
  - EJB Container
  - IoC(또는 DI) Container

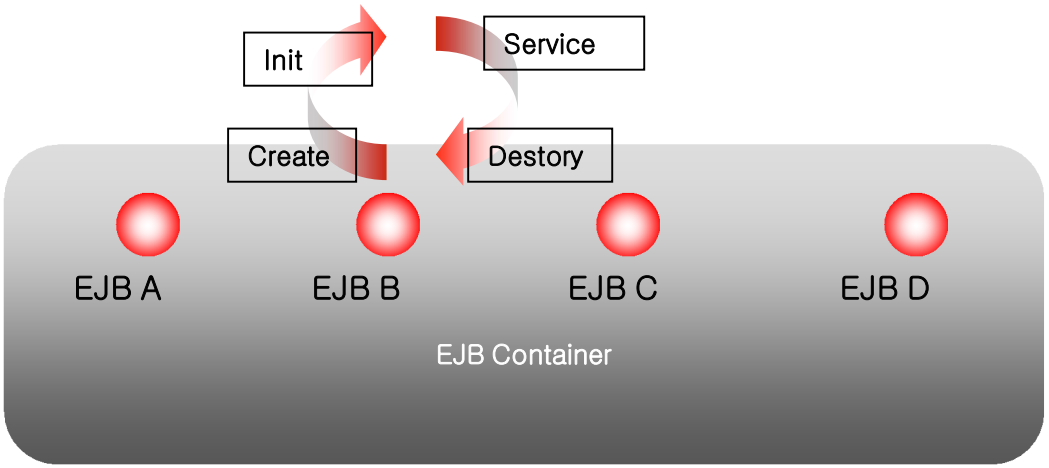
Spring Framework - IoC

- Servlet Container
  - Servlet의 생성, 생성 후 초기화, 서비스 실행, 소멸에 관한 모든 권한을 가지면서 Servlet의 생명주기를 관리한다.
  - 개발자들이 직접 Servlet을 생성하고 서비스하지는 않는다.
  - JSP/Servlet 접근 권한, 에러 처리에 대한 추가적인 기능도 지원한다.



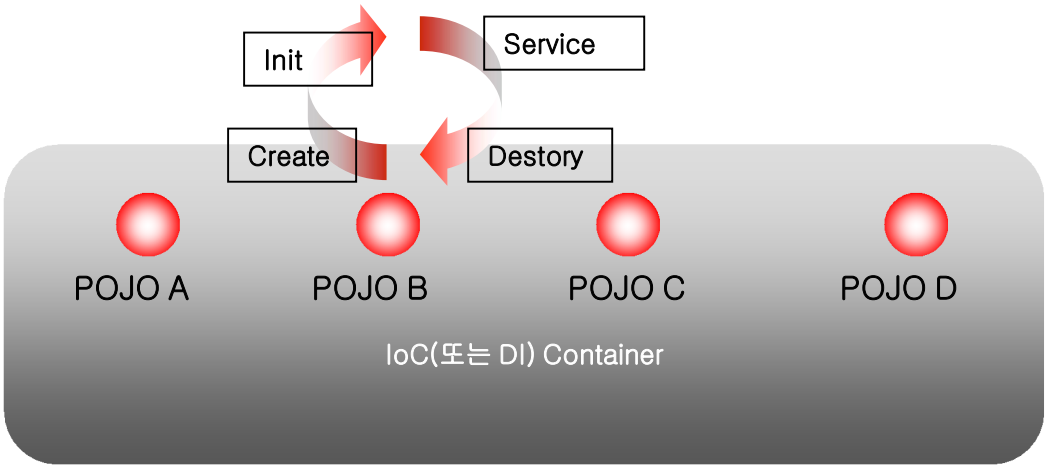
Spring Framework - IoC

- EJB Container
  - EJB(세션빈, 엔티티빈, MDB)의 생성, 생성 후 초기화, 서비스 실행, 소멸에 관한 모든 권한을 가지면서 EJB의 생명주기를 관리한다.
  - 개발자들이 직접 EJB 생성하고 서비스할 수 없다.
  - Transaction, Security, EJB Pooling등의 추가적인 기능을 제공하고 있다.



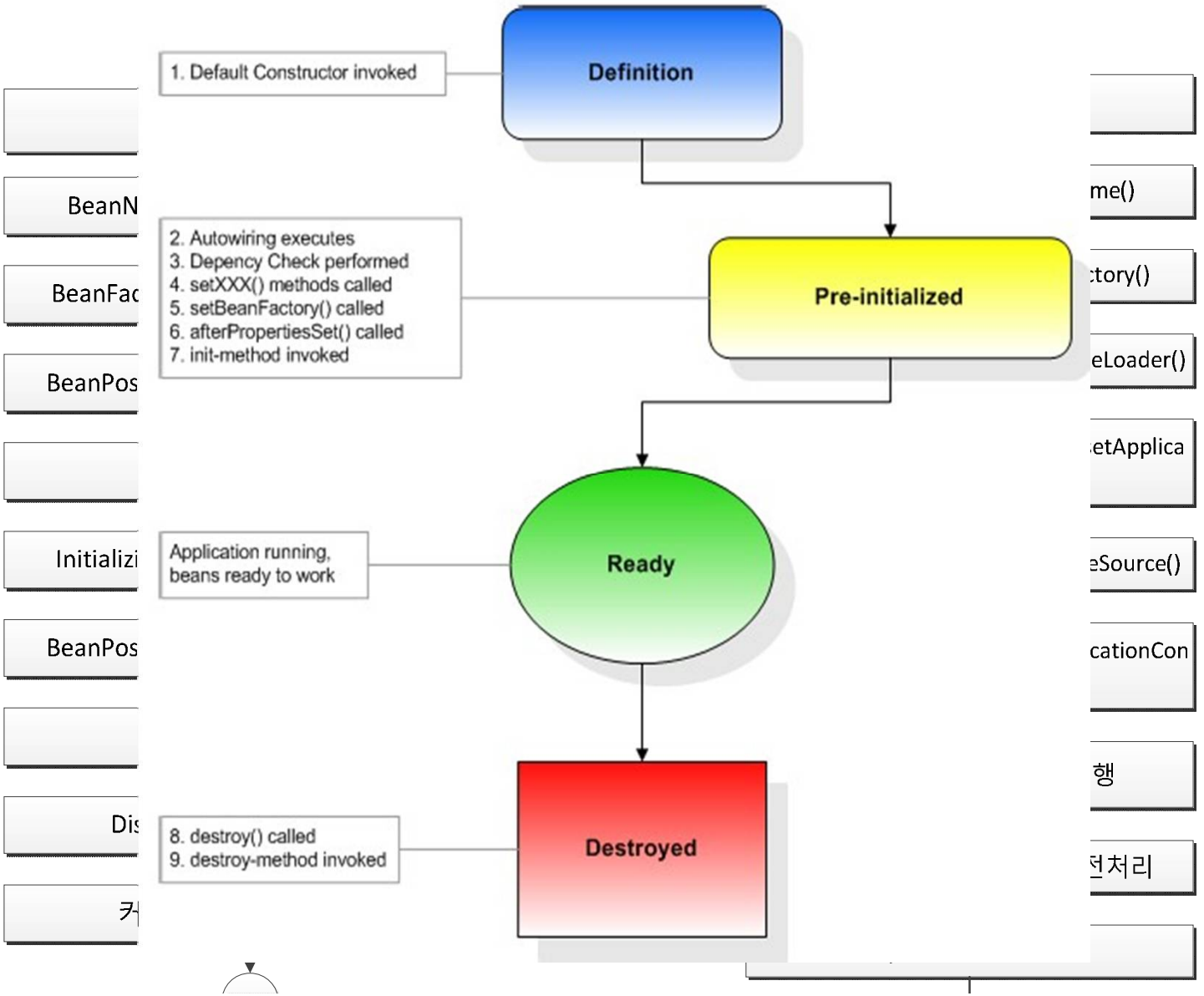
Spring Framework - IoC

- IoC(또는 DI) Container
  - POJO의 생성, 초기화, 서비스 소멸에 관한 모든 권한을 가지면서 POJO의 생명주기를 관리한다.
  - 개발자들이 직접 POJO를 생성할 수도 있지만, 모든 권한을 Container에게 맡긴다.
  - Transaction, Security 추가적인 기능을 제공한다. AOP 기능을 이용하여 새로운 Container 기능을 추가하는 것이 가능하다.

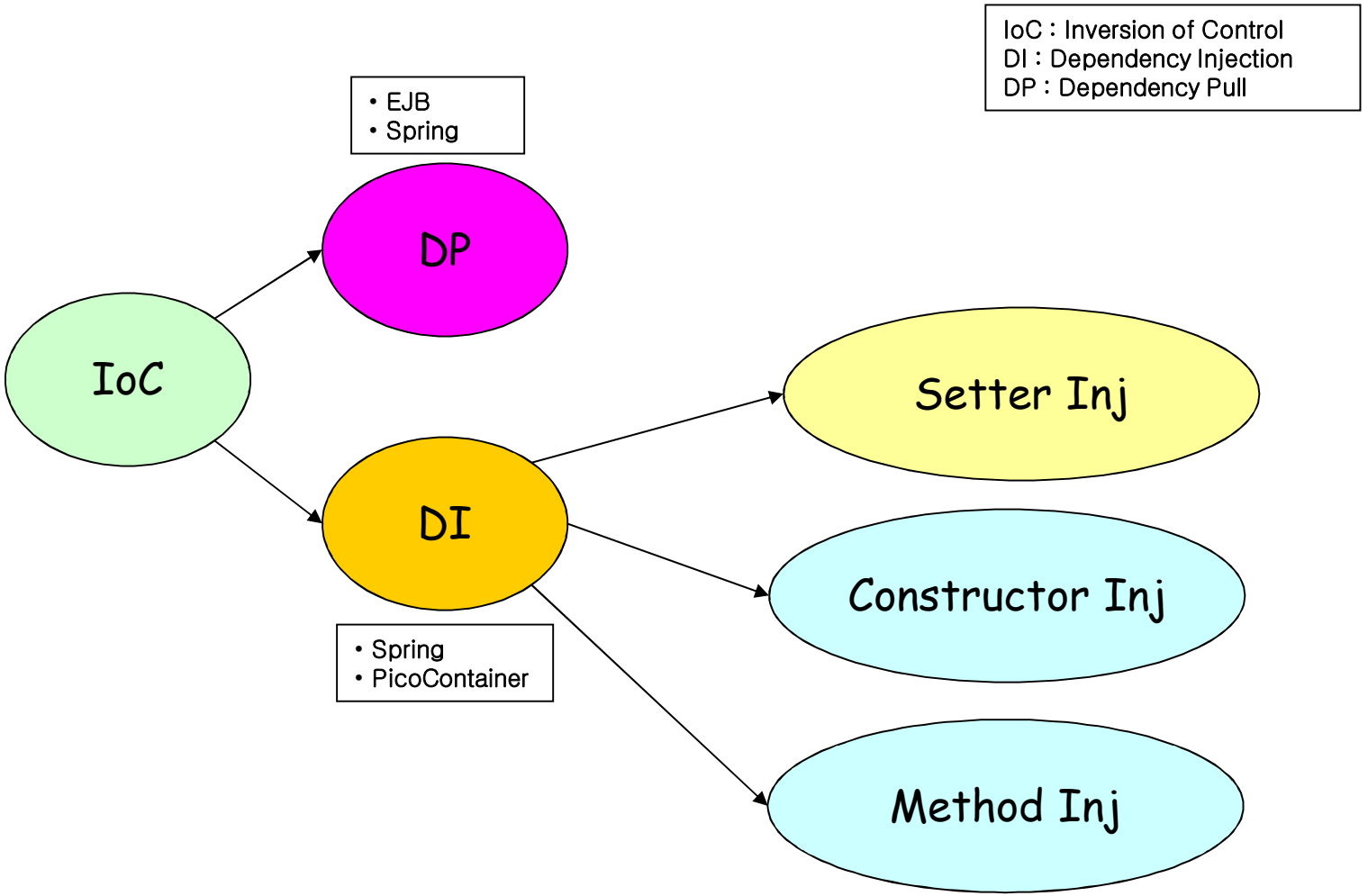


DI 라이프 사이클

- BeanFactory 초기화 과정
- ApplicationContext 초기화 과정



Spring Framework - IoC



DI 주입 방식에는?

- setter 주입
  - <property name="name" value="Text" />
  - <property name="type" ref="webType" />

- 생성자 주입
  - <constructor-arg type="java.lang.String" value="WEB" />

context-my.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="engine" class="di02.basic.service.MySearchEngine" />

    <bean id="documentType" class="di02.basic.model.Type">

    </bean>

</beans>
```

bean은 인스턴스 선언하는 역할

property는 setter 역할

TestMakeInstanceWithSpring(1/2)

```
public class TestMyDocumentsWithSpring {

    private ClassPathXmlApplicationContext context;
    private ISearchEngine engine;
    private Type docType;

    @Before
    public void setup() {
        try {
            context = new ClassPathXmlApplicationContext("di02/basic/context-my.xml");
            engine = context.getBean(ISearchEngine.class);
            docType = context.getBean(Type.class);
        } catch (BeansException e) {
            e.printStackTrace();
        }
    }

    @Test
    public void testListAllWithSpring() {
        List<Document> documents = engine.listAll();
        assertNotNull(documents);
        assertTrue(documents.size() == 4);
    }
```

xml 파일 경로 지정

getBean()은 reflection을 이용하여 인스턴스 생성

TestMakeInstanceWithSpring(2/2)

```
    // 이어서...

    @Test
    public void testWithSpringFindByType() {
        List<Document> documents = engine.findByType(docType);
        assertNotNull(documents);
        assertTrue(documents.size() == 1);
        assertEquals(documentType.getName(), documents.get(0).getType().getName());
        assertEquals(documentType.getDesc(), documents.get(0).getType().getDesc());
        assertEquals(documentType.getExtension(), documents.get(0).getType().getExtension());
    }
}
```

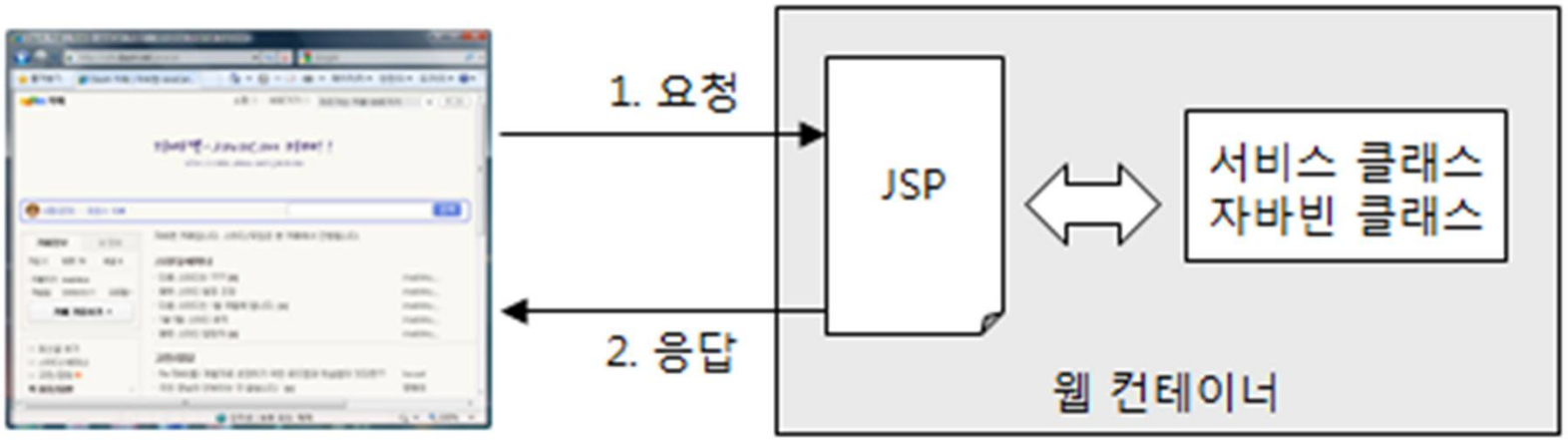
AOP

JSP 모델 구조

- 모델 1 구조
- 모델 2 구조
- MVC 패턴
- MVC 패턴 구현 방식

모델 1 구조

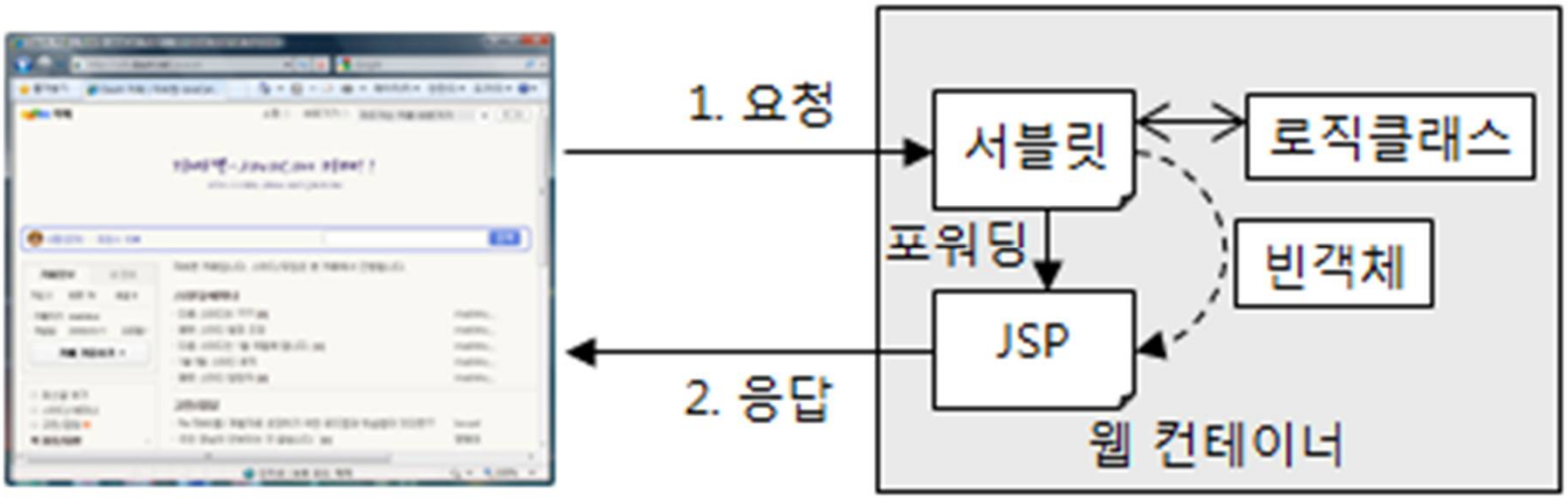
- JSP를 이용한 단순한 모델



- JSP에서 요청 처리 및 뷰 생성 처리
  - 구현이 쉬움
  - 요청 처리 및 뷰 생성 코드가 뒤섞여 코드가 복잡함

모델 2 구조

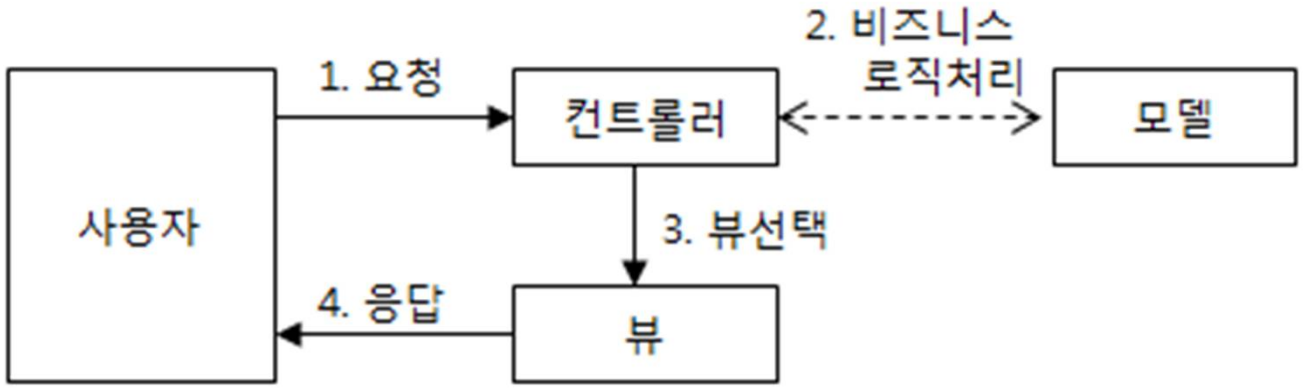
- 서블릿이 요청을 처리하고 JSP가 뷰를 생성



- 모든 요청을 단일 서블릿에서 처리
  - 요청 처리 후 결과를 보여줄 JSP로 이동

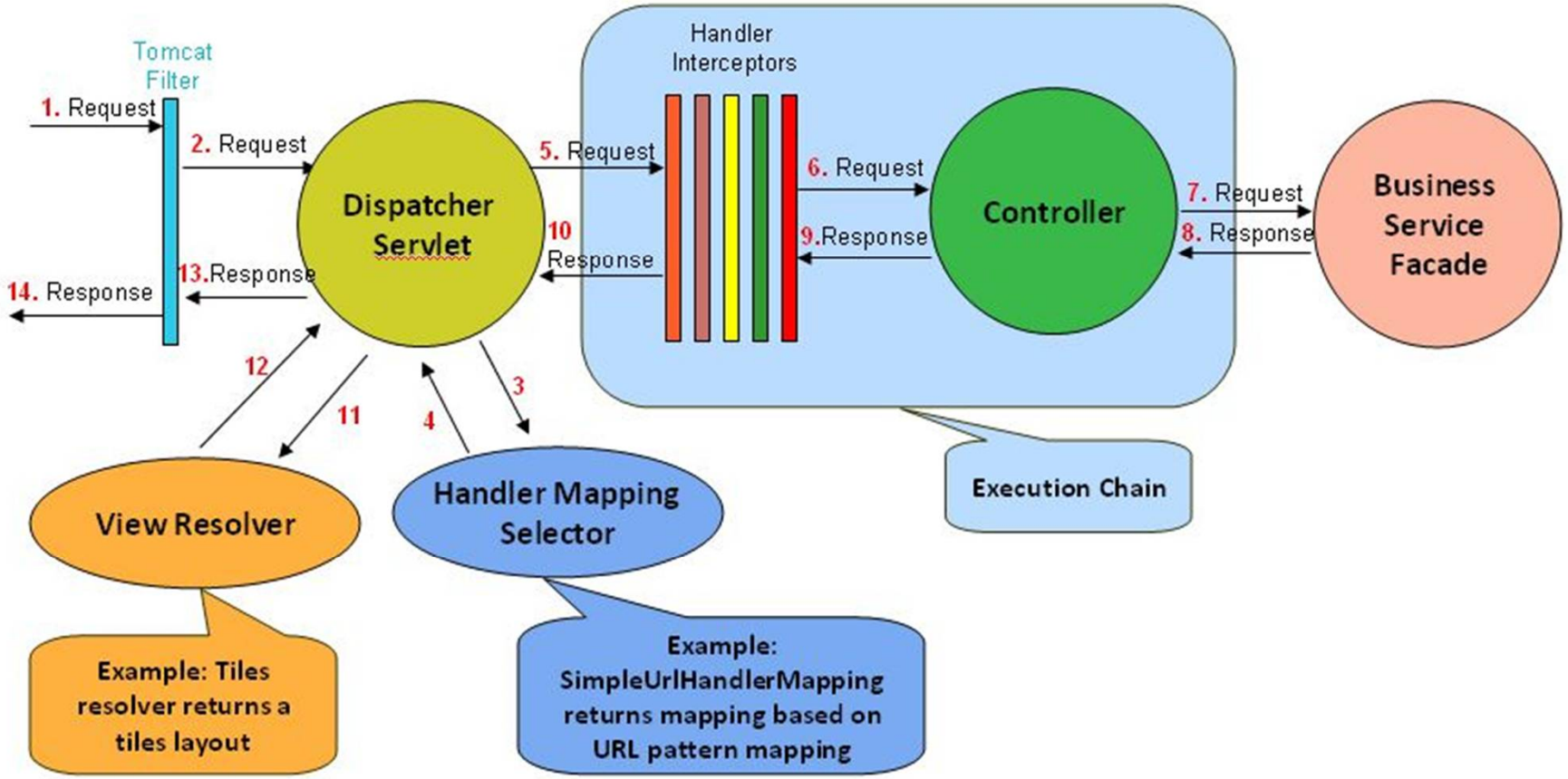
MVC 패턴

- Model-View-Controller
  - 모델 - 비즈니스 영역의 상태 정보를 처리한다.
  - 뷰 - 비즈니스 영역에 대한 프레젠테이션 뷰(즉, 사용자가 보게 될 결과 화면)를 담당한다.
  - 컨트롤러 - 사용자의 입력 및 흐름 제어를 담당한다.

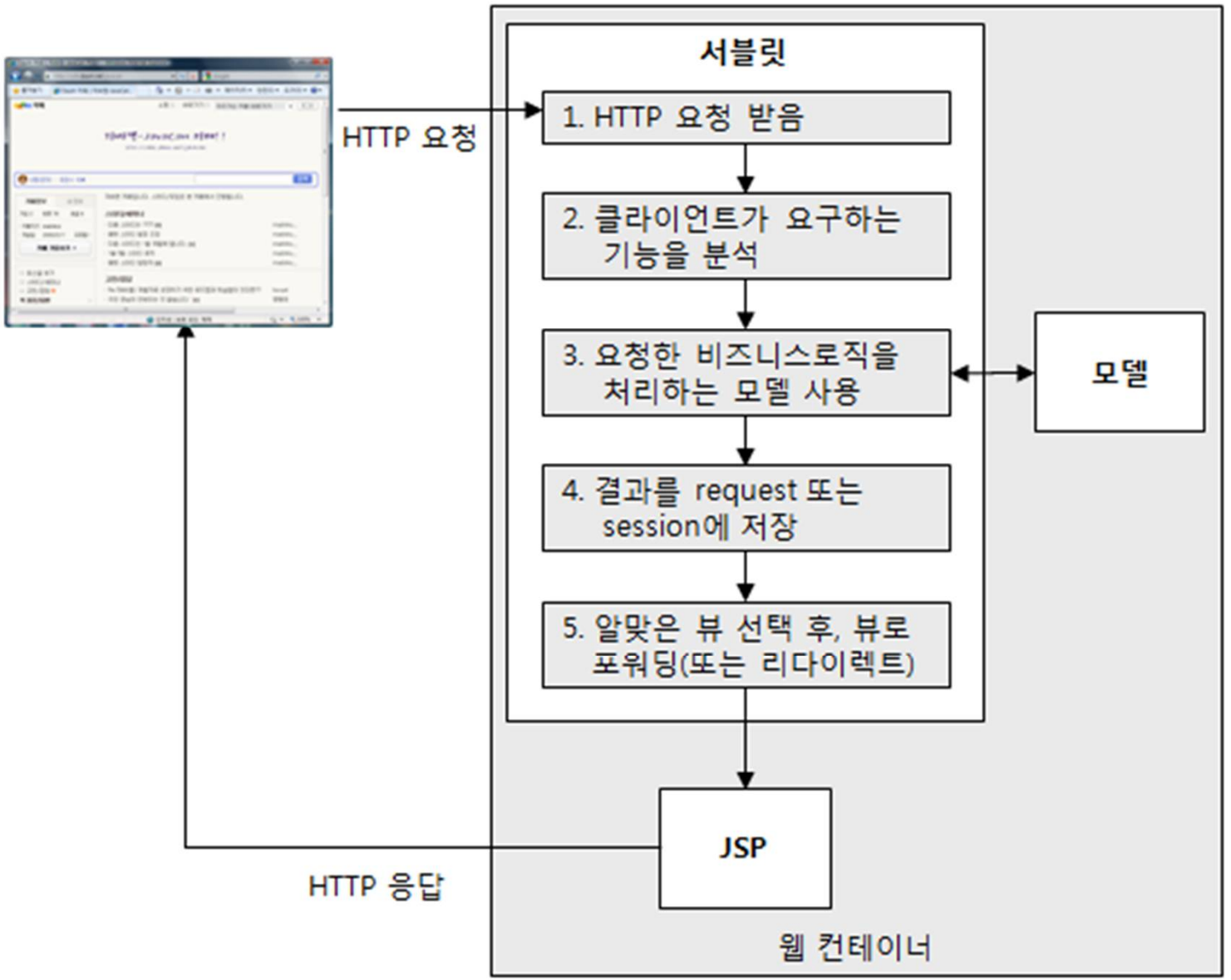


- 특징
  - 로직을 처리하는 모델과 결과 화면을 보여주는 뷰가 분리되
  - 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중
- 모델 2 구조와 매핑: 컨트롤러-서블릿, 뷰-JSP

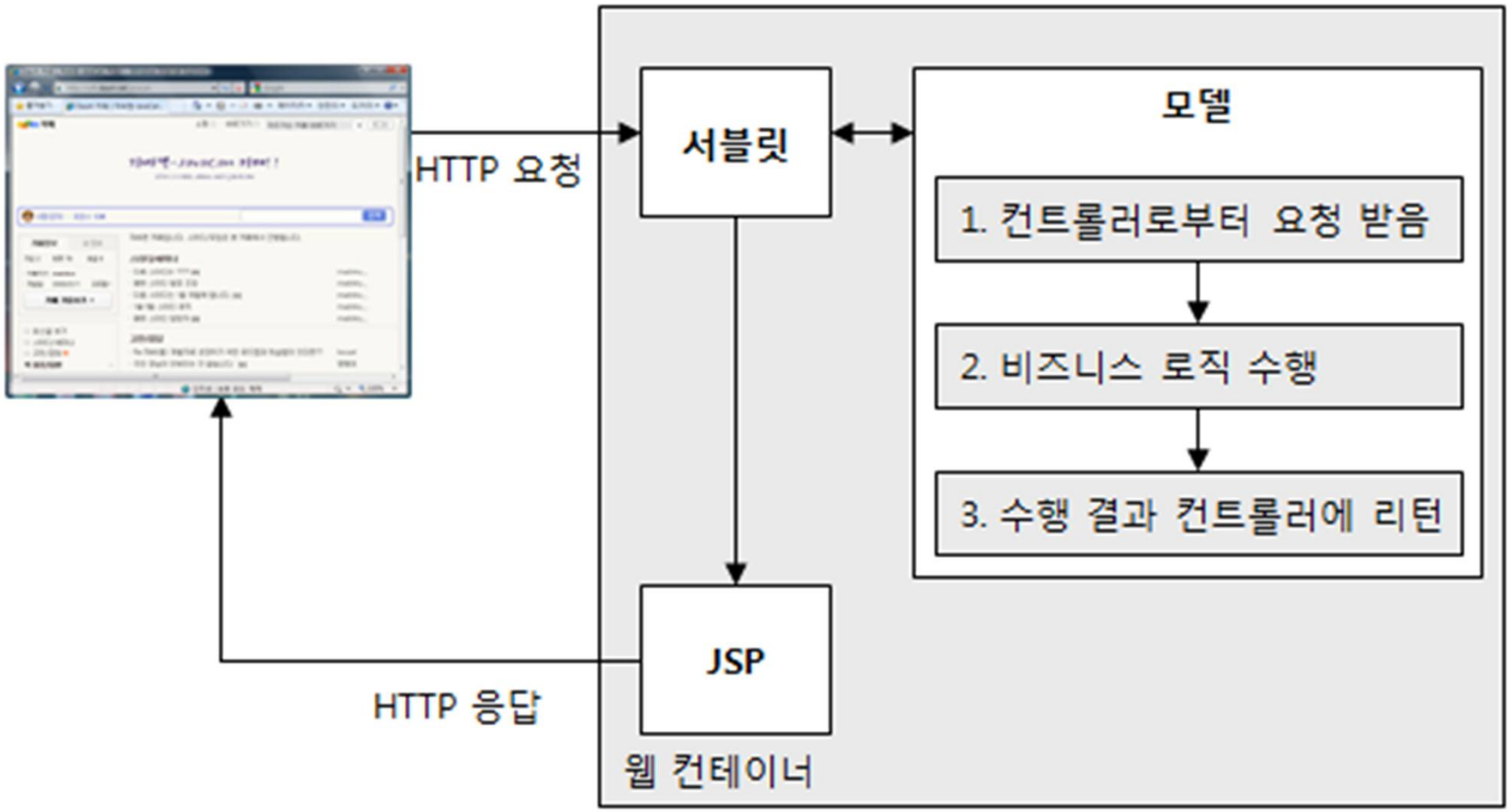
MVC 패턴


Tomcat
Filter
1. Request
2. Request
Dispatcher
Servlet
5. Request
Handler
Interceptors
6. Request
Controller
7. Request
Business
Service
Facade
8. Response
9.Response
10
Response
13.Response
14. Response
12
11
4
3
View Resolver
Handler Mapping
Selector
Execution Chain
Example: Tiles
resolver returns a
tiles layout
Example:
SimpleUrlHandlerMapping
returns mapping based on
URL pattern mapping

MVC의 컨트롤러 : 서블릿



MVC의 모델 : 로직 수행 클래스



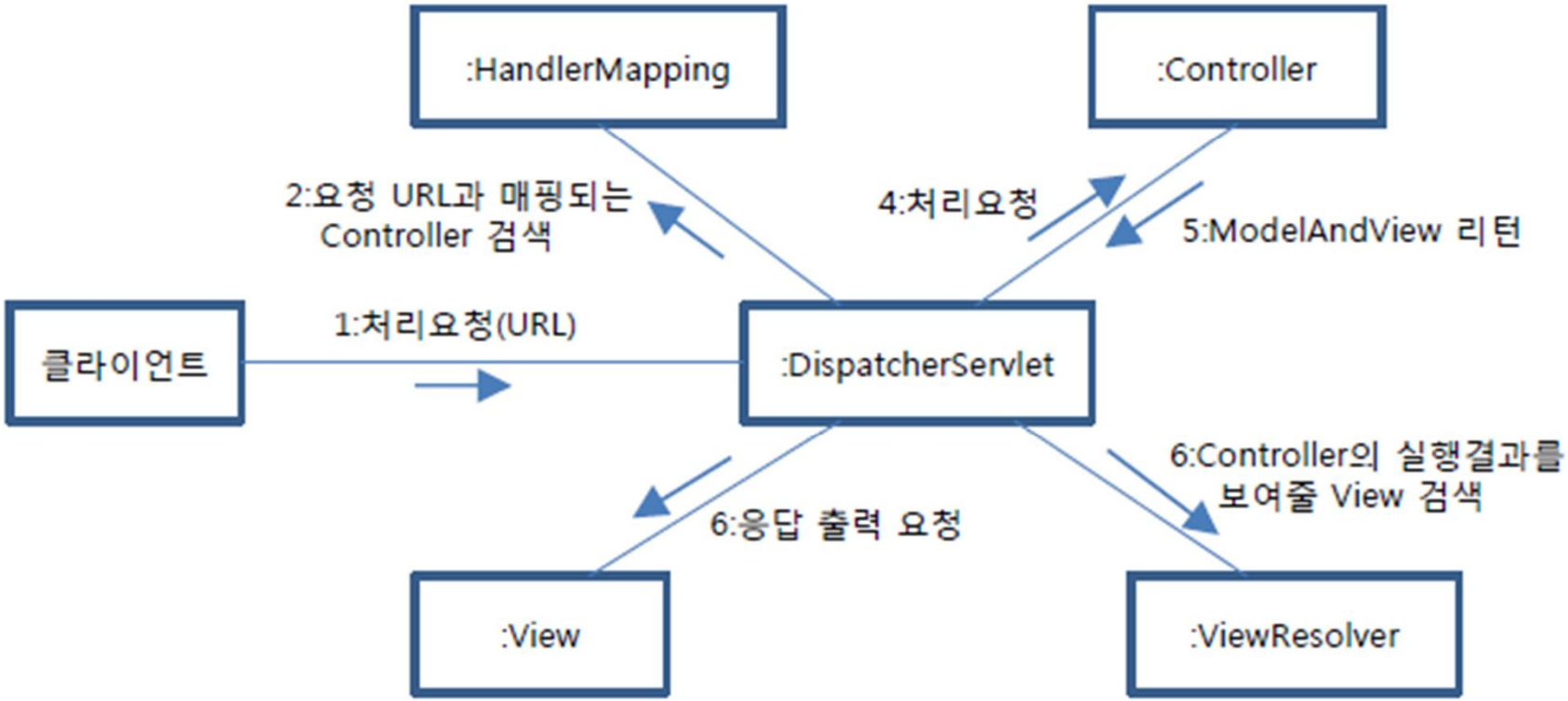
스프링 MVC의 구성 요소

- 스프링 MVC의 주요 구성 요소

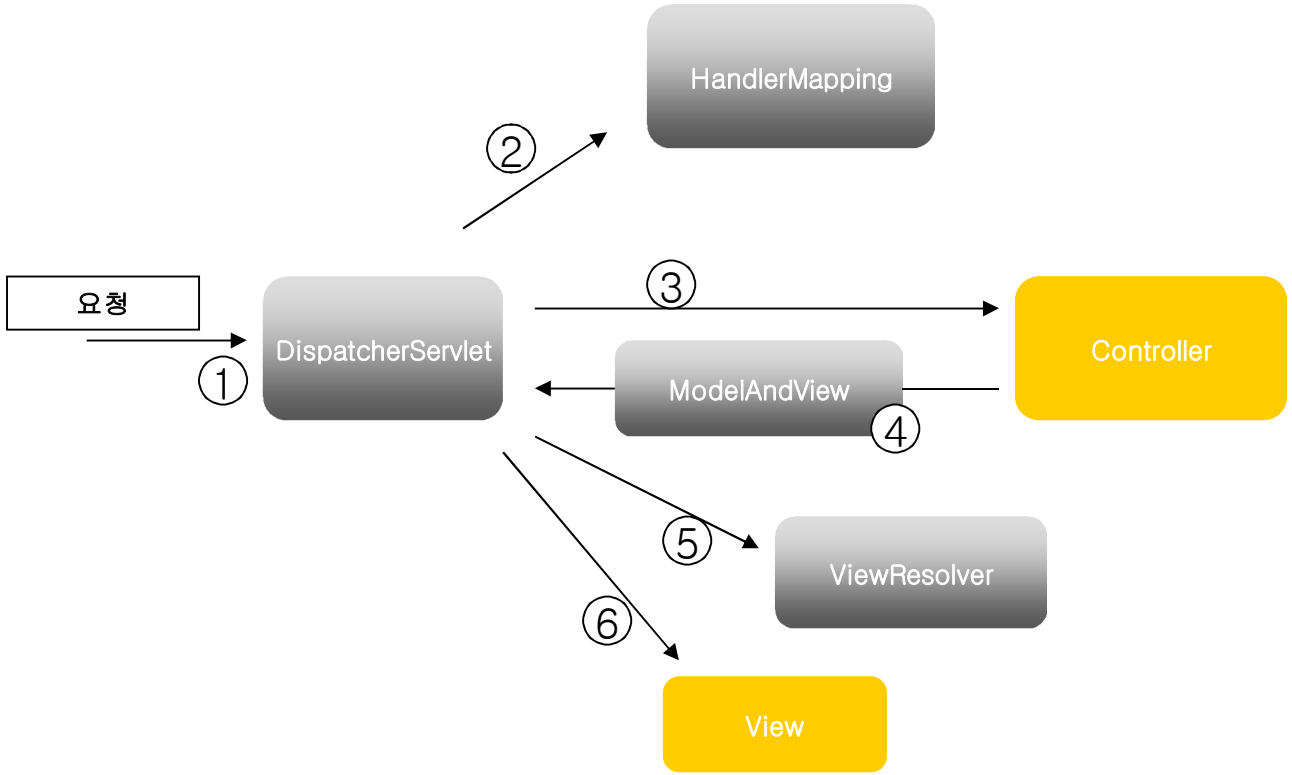
구 성 요 소	설 명
DispatcherServlet	클라이언트의 요청을 컨트롤러에게 전달하고 컨트롤러가 리턴한 결과값을 View에 전달한다.
HandlerMapping	클라이언트의 요청 URL을 어떤 컨트롤러가 처리할지 결정한다.
Controller	클라이언트의 요청을 처리한 뒤 그 결과를 DispatcherServlet에 알려준다. 스트럿츠의 Action과 동일한 역할을 수행
ModelAndView	컨트롤러가 처리한 결과 정보와 뷰 선택에 필요한 정보를 담는다.
ViewResolver	컨트롤러의 처리 결과를 생성할 뷰를 결정한다.
View	컨트롤러의 처리 결과 화면을 생성한다. JSP나 Velocity 템플릿 파일 등을 뷰로 사용한다.

스프링 MVC의 처리 흐름도(1)

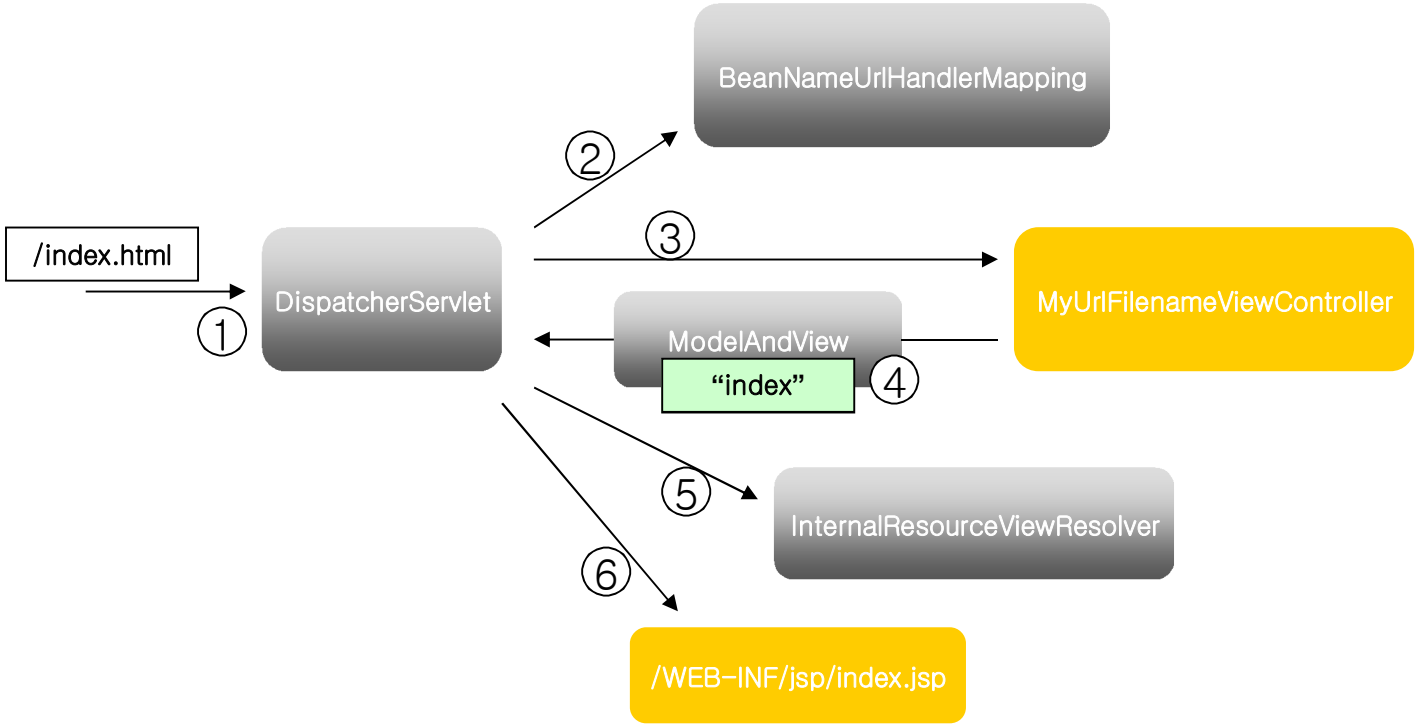
· 이들 각 요소의 메시지 흐름은 다음과 같다.



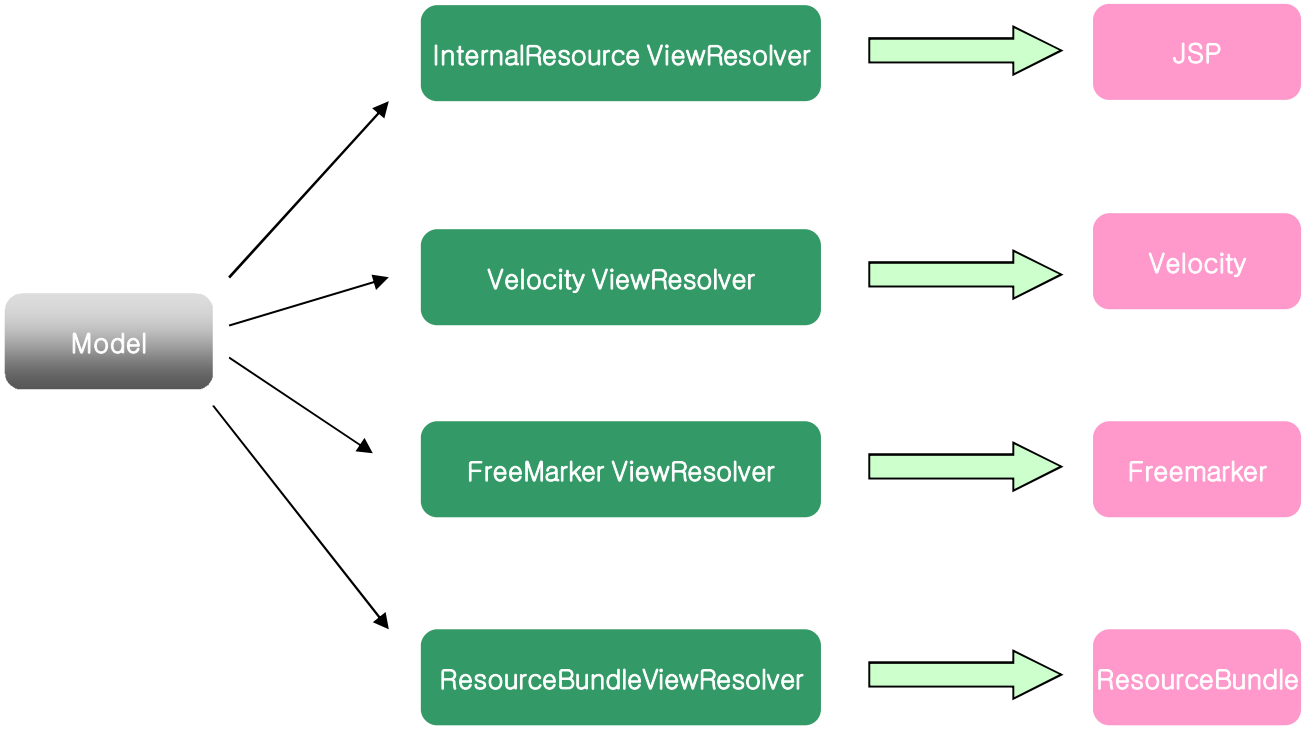
스프링 MVC의 처리 흐름도(2)



스프링 MVC의 처리 흐름도(3)



Spring MVC - ViewResolver



Model

- x.y.User => user
- java.util.HashMap => hashMap
- x.y.User[] => userList
- 하나 이상의 x.y.User 인스턴스를 가지는 java.util.ArrayList => userList

View - ViewResolver

- ViewResolver Chain

```
<beans:bean id="xmlViewResolver"
          class="org.springframework.web.servlet.view.XmlViewResolver">
    <beans:property name="order" value="1"/>
    <beans:property name="location" value="/WEB-INF/ajasee-views.xml"/>
</beans:bean>

<!--  InternalResourceViewResolver는 항상 Chain의 마지막에 위치 해야 함  -->
<beans:bean id="viewResolver"
   class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
    </beans:property>
    <beans:property name="cache" value="false" />
    <beans:property name="prefix" value="/WEB-INF/jsp/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

- **http://localhost:8080/helloworld**
  => WEB-INF/views/helloworld.jsp
- **http://localhost:8080/helloworld/hello**
  => WEB-INF/views/helloworld/hello.jsp

Controller

```
<bean
    class="org.springframework.web.servlet.mvc.support.ControllerClassNameHan
    dlerMapping" />
```

- HelloWorldController => helloworld
- HelloWorldController.hello => helloworld/hello

JSTL

JSTL

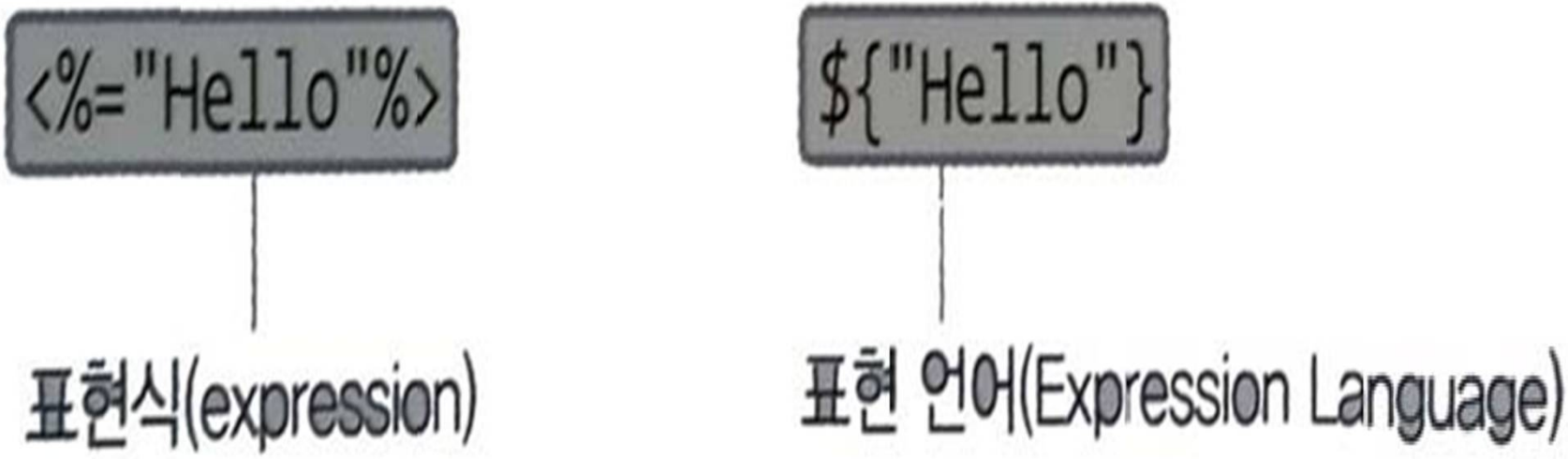
1. JSTL 변수 선언
   - http://localhost/jstl/jstl01
2. JSTL 선택문
   - http://localhost/jstl/jstl03
3. JSTL 반복문
   - http://localhost/jstl/jstl04
   - http://localhost/jstl/jstl05
4. JSTL import
   - http://localhost/jstl/jstl11
5. JSTL redirect
   - http://localhost/jstl/jstl12
6. JSTL forward
   - http://localhost/jstl/jstl13

EL

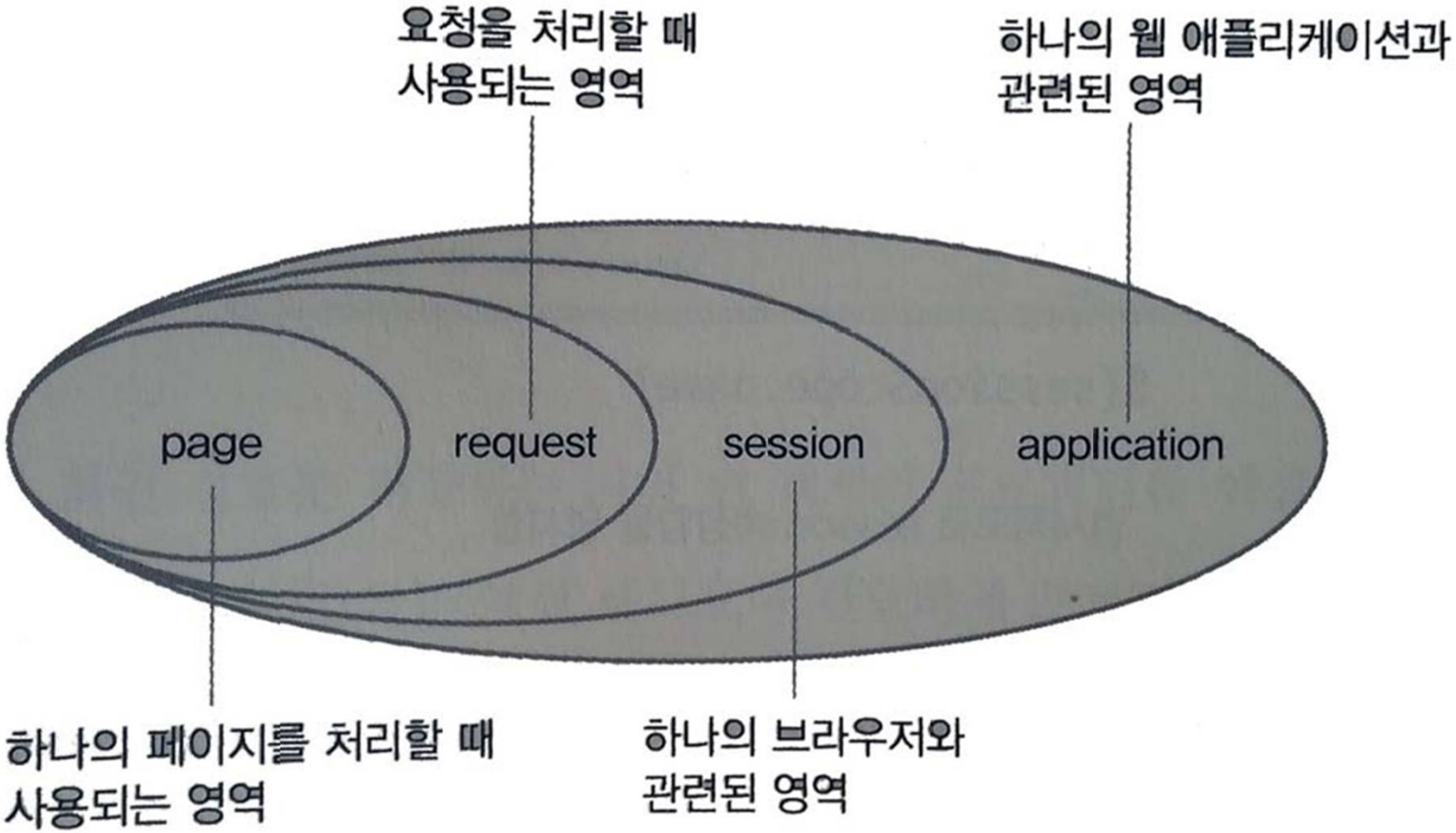
- EL(Exression Language)의 표기
- EL 의 내장 객체와 범

EL(Exression Language)의 표기

표현식	표현 언어
<%=expr%>	${expr}
<%="hello" %>	${ "hello" }



EL 의 내장 객체와 범


요청을 처리할 때
사용되는 영역
하나의 웹 애플리케이션과
관련된 영역
page
request
session
application
하나의 페이지를 처리할 때
사용되는 영역
하나의 브라우저와
관련된 영역

HomeController 만들기

1. 스프링 프로젝트 생성
   - gradle 프로젝트로 변경:
     Configure >> Convert to Gradle ...
   - build.gradle 수정
   - gradle >> refresh dependency 실행
   - gradle >> refresh all 실행
2. src/main/resources 폴더에 아래 파일들을 추가하시오
   - log4j.properties 생성
3. web.xml 수정
   - 인코딩 필터 설정
   - JSP 파일 UTF-8 인코딩 설정

HomeController 만들기

- 절대 경로와 상대 경로 이해하기
- 컨트롤러에서 화면으로 데이터 넘기는 방법
  - model.addAttribute()
- redirect, forward 이해
  - redirect : url이 바뀐다
  - forward: url이 안 바뀐다.
- url에서 값을 넘겨 받는 방법
  - @RequestParam
  - request.getParameter();
  - @PathVariable
- form 태그 사용법
- login GET/POST 처리

절대경로와 상대 경로

- 절대경로와 상대 경로 비교
  - 절대 경로

```
<form action="/spring/login" post="post" enctype="..." />
</form>
```

  - 상대 경로

```
<form action="spring/login" post="post" enctype="..." />
</form>
```

`http://localhost/spring/a/b` + `../c.jpg` = **`http://localhost/spring/c.jpg`**

`http://localhost/spring` + `./aa/c.jpg` = **`http://localhost/aa/c.jpg`**

`http://localhost/spring` + `aa/c.jpg` = **`http://localhost/aa/c.jpg`**

`http://localhost/spring` + `/aa/c.jpg` = **`http://localhost/aa/c.jpg`**

`http://localhost/spring/a/b` + `/c.jpg` = **`http://localhost/c.jpg`**

컨트롤러에서 JSP(화면)으로 데이터 넘기는 방법

1. 값으로 넘기는 방법

```
model.addAttribute("id", "Hello, World!!!" );
```

```
<p> ${id} </p>
```

2. 모델로 넘기는 방법

```
model.addAttribute("info", new ModelLogin() );
```

```
<p> ${info.id} </p>
<p> ${info.pw} </p>
```

3. List로 넘기는 방법

```
List<ModelLogin> list = new ArrayList<ModelLogin>();
model.addAttribute("info", list );
```

```
<c:forEach var="i" items="${info }" varStatus="status">
    <p> ${status.index} </p>
    <p> ${i.id} </p>
    <p> ${i.pw} </p>
</c:forEach>
```

화면에 현재 시간을 출력하시오.

- 접속 url: http://localhost/

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>
    <h2> ${serverTime }</h2>
</body>
</html>
```

```
@RequestMapping(value = "/spring/helloworld", method = RequestMethod.GET)
public String helloworld(Model model) {
    model.addAttribute("msg", "Hello, World!!!" );
    return "home/helloworld";
}
```

화면에 "Hello, Wordl!!!" 를 출력하시오.

- jsp 경로: src/main/webapp/WEB-INF/views/home/helloword.jsp

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>
    <h2> ${msg }</h2>
</body>
</html>
```

- 접속 url: http://localhost/spring/helloworld

```
@RequestMapping(value = "/spring/helloworld", method = RequestMethod.GET)
public String helloworld(Model model) {
    model.addAttribute("msg", "Hello, World!!!" );
    return "home/helloworld";
}
```

ModelAndView를 이용하여 화면에 "Hello, Word!!!" 를 출력하시오.

- jsp 경로: src/main/webapp/WEB-INF/views/home/sayhello.jsp

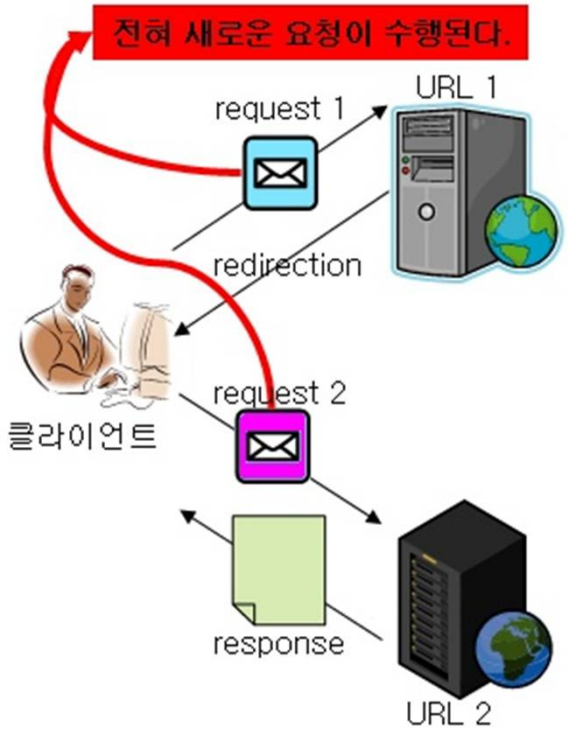
```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>
    <h2> ${msg }</h2>
</body>
</html>
```

- 접속 url: http://localhost/spring/sayHello

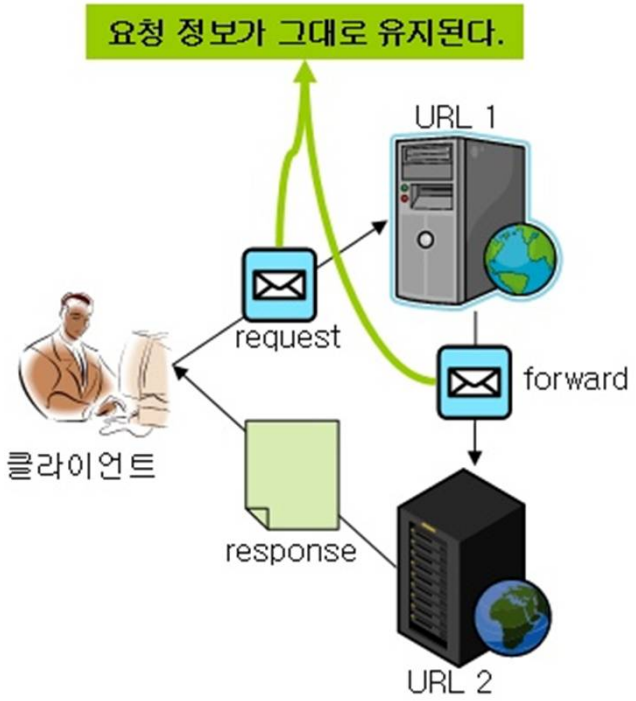
```
@RequestMapping(value = "/spring/sayHello", method = RequestMethod.GET)
public ModelAndView sayHello(Model model) {

    Map<String, String> map = new HashMap<String, String>();
    map.put("msg", "say Hello");

    return new ModelAndView("home/sayHello", map) ;
}
```

redirect, forward 이해

- redirect 테스트
- redirect 된 페이지에서는 request 와 response 객체가 새롭게 생성
- URL을 지시된 주소로 바꾸고 그 주소로 이동



- forward 테스트
- 현재 페이지와 forwad 될 페이지는 request와 response 객체를 공유
- 최초에 호출한 URL이 표시. 이동한 페이지의 URL 정보는 볼 수 없다



redirect와 forward 테스트

- redirect 테스트 : http://localhost/spring/redirect

```
@RequestMapping(value = "/spring/redirect", method = RequestMethod.GET)
public String redirect(Model model) {
    return "redirect:/spring/helloworld";
}
```

- forward 테스트 : http://localhost/spring/forward

```
@RequestMapping(value = "/spring/forward", method = RequestMethod.GET)
public String forward(Model model) {
    return "forward:/spring/helloworld";
}
```

url에서 값을 넘겨 받는 방법

- url 의 QueryString에서 값을 받는 방법

http://localhost/spring/querystring**?name=free**

1. request.getParameter() 를 이용하여 값 넘겨 받기
2. @RequestParam 어노테이션을 이용하여 값 넘겨 받기

- url 의 path에서 값을 받는 방법

http://localhost/spring/querypath/**free**

1. @PathVariable로 값 넘겨 받기

url 의 QueryString에서 변수 값을 넘겨 받는 방법

- Controller에서 url 의 QueryString으로부터 변수 값을 받으려면

http://localhost/spring/writeone1**?name=free&price=100**

```
@RequestMapping(value="/spring/writeone1", method = RequestMethod.GET)
public String writeone1(Model model,  HttpServletRequest request) {
     String name          = request.getParameter("name");
     String price         = request.getParameter("price");
     // ....
}
```

- src/main/webapp/WEB-INF/views/home/writeone.jsp 를 만드시오

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>




</body>
</html>
```

url 의 QueryString에서 변수 값을 넘겨 받는 방법

- Controller에서 url 의 QueryString으로부터 변수 값을 받으려면

http://localhost/spring/writeone2**?name=free&price=100**

```
RequestMapping(value="/spring/writeone2", method = RequestMethod.GET)
public String writeone2(Model model
    , @RequestParam(value="name", defaultValue="" ) String name
    , @RequestParam(value="price", defaultValue="0" ) int price ) {
    // ....
}
```

- jsp 경로: src/main/webapp/WEB-INF/views/home/writeone.jsp 를 만드시오

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>

</body>
</html>
```

- @RequestParam 방식이 getParameter() 메서드보다 편리

url 의 path에서 값을 받는 방법

- Controller에서 url 의 path로부터 변수 값을 받으려면
  - --> @PathVariable 어노테이션을 사용하여 한다.

접속 url: http://localhost/spring/querypath/**free**

- jsp 경로: src/main/webapp/WEB-INF/views/home/querypath.jsp 를 만드시.

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>
    <h2> ${name }</h2>
</body>
</html>
```

```
@RequestMapping(value="/spring/querypath/{name}", method=RequestMethod.GET)
public String querypath(Model model
        , @PathVariable(value="name") String name ) {
    model.addAttribute("name" , name  );
    return "home/querypath" ;
}
```

form 태그 사용법

- 11강 (다양한 input 태그)
  - http://youtu.be/SSV63gkvBSo
- 12강 (input 태그의 속성)
  - http://youtu.be/hapY8EVIUn8
- 13강 (상품 주문서 만들기)
  - http://youtu.be/DMO03XhJNKQ

폼(<form />) 태그 사용법. 1

```
<form action="" method="" enctype="">

    <!-- 사용자가 입력하는 입력 양식 -->
    <input type="text"     name="text"     value="text"     /> <br />
    <input type="password" name="password" value="password" /><br />
    <input type="file"     name="file"     value="file"     /><br />
    <input type="checkbox" name="checkbox" value="checkbox" /><br />
    <input type="radio"    name="radio"    value="radio"    /><br />

    <!-- 보이지 않는 입력 양식 -->
    <input type="hidden" name="hidden" value="hidden" /><br />

    <!-- 버튼 -->
    <input type="button" value="button" /><br />
    <input type="reset"  value="reset"  /><br />
    <input type="submit" value="submit" /><br />
    <input type="image"  src="http://placehold.it/100x100" />

</form>
```

폼(<form />) 태그 사용법. 2

```
<form method="get" action="https://search.naver.com/search.naver">
    <input type="text"    name="query" />
    <input type="submit"  value="전송" />
</form>
```

네이버 get 요청

text 전송

- form 태그 사용시 method, action, enctype 속성(attribute)은 필수다.
  - method : **get** or **post**
  - enctype : **application/x-www-form-urlencoded(기본값), multipart/form-data**
- input 태그 사용시 type, name 속성(attribute)은 필수다.
- submit 버튼은 반드시 <form> ...</form> 사이에 있어야만 form이 작동된다.

폼(<form />) 태그 사용법. 3

```
<form method="post" action="https://search.naver.com/search.naver">
    <input type="text"    name="query" />
    <input type="submit"  value="전송" />
</form>
```

네이버 post 요청

text 전송

Forbidden

You don't have permission to access /search.naver on this server.

- form 태그 사용시 method, action, enctype 속성(attribute)은 필수다.
  - method : **get** or **post**
  - enctype : **application/x-www-form-urlencoded(기본값), multipart/form-data**
- input 태그 사용시 type, name 속성(attribute)은 필수다.
- submit 버튼은 반드시 <form> ...</form> 사이에 있어야만 form이 작동된다.

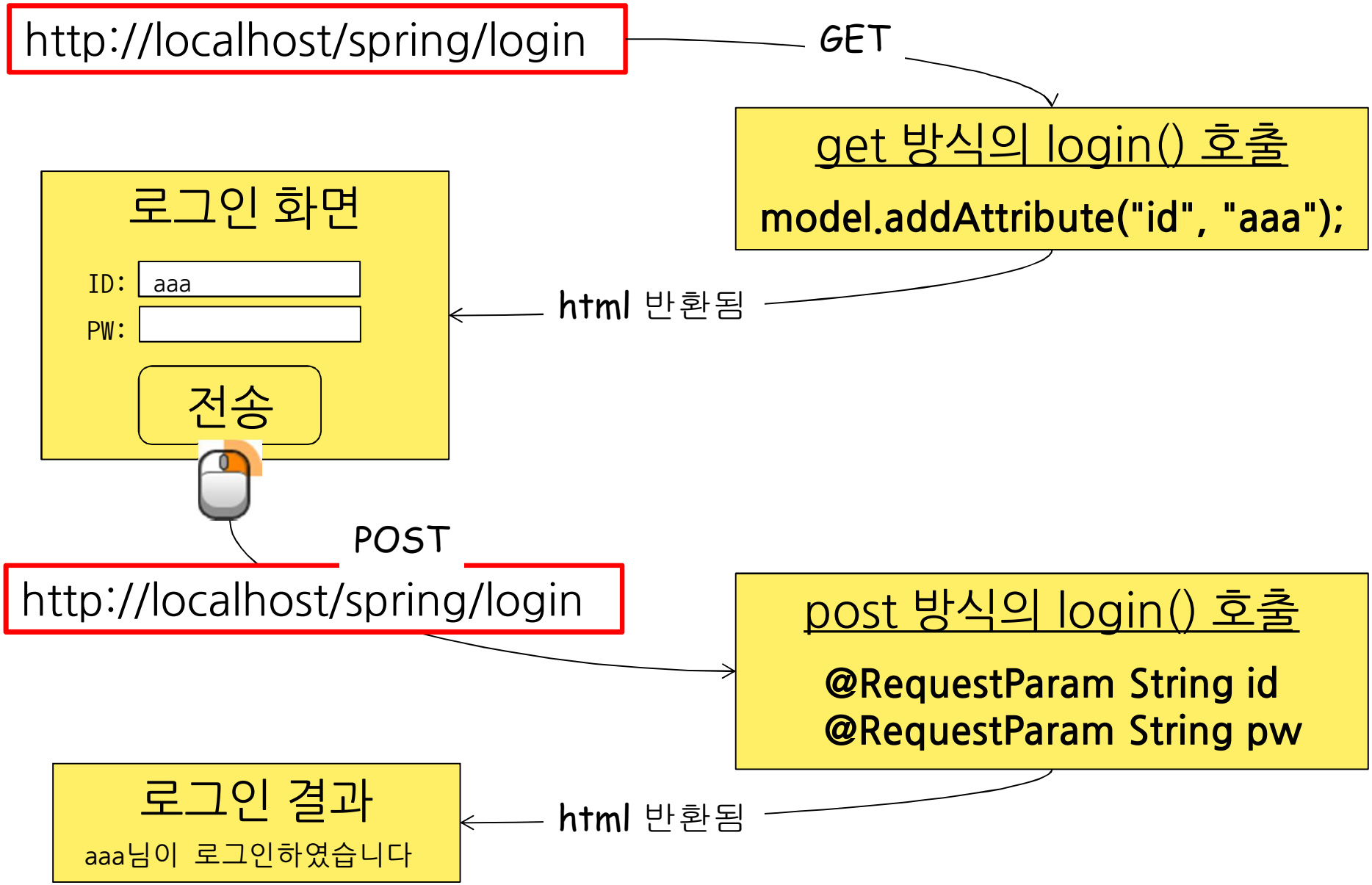
jquery로 폼(<form />) 태그 호출하기. 4

```
<script src="https://code.jquery.com/jquery-3.1.0.js"></script>
<script>
    $(document).ready( function(e){
        // c1 버튼에 클릭 이벤트 핸들러 설정
        $('#c1').click( function(e){
            var form = $('form');
            $(form).attr('action' , 'https://search.naver.com/search.naver');
            $(form).attr('method' , 'get');
            $(form).attr('enctype', 'multipart/form-data');
            $(form).sumbmit();
        });
    });
</script>
<form action="" method="" enctype="">
    <!-- 사용자가 입력하는 입력 양식 -->
    <input type="text"   name="query"   value="text"   /><br />

    <!-- 버튼 -->
    <input type="submit" value="submit" /><br />
    <input type="image"  src="http://placehold.it/100x100" id="c1" />
</form>
```

자바스크립로 폼(<form />) 태그 만들기. 5

```
<script>
    var f = document.createElement('form');
    f.setAttribute('method' , 'get');
    f.setAttribute('action' , 'https://search.naver.com/search.naver'');
    f.setAttribute('enctype', 'application/x-www-form-urlencoded');

    var i = document.createElement('input'); //input element, hidden
    i.setAttribute('type' ,'text');
    i.setAttribute('name' ,'query');
    i.setAttribute('value','text');
    f.appendChild(i);

    f.submit();
</script>
<body>
</body>
```

@RequestParam 을 이용한 login GET/POST 처리



폼(<form />)에서 POST로 @RequestParam을 이용해 값을 넘겨 받는 방법

- src/main/webapp/WEB-INF/views/home/loginmodel.jsp 를 만드시오

```
<form action="/spring/login" method="post" enctype="application/x-www-form-urlencoded">
    <label for="name">이름<input type="text" name="name" /></label>
    <label for="phone">폰 <input type="text" name="phone" /></label>
    <input type="submit" value="전송" />
</form>
```

- 실행 url: http://localhost/spring/loginmodel

```
@RequestMapping(value="/spring/login", method=RequestMethod.POST)
public String login( Model model , @ModelAttribute ModelLogin login) {

    model.addAttribute("logininfo", login);

    return "phone/writeOneResult";
}
```

PhoneController 만들기

1. 스프링 프로젝트 생성
   - gradle 프로젝트로 변경:
     Configure >> Convert to Gradle ...
   - build.gradle 수정
   - gradle >> refresh dependency 실행
   - gradle >> refresh all 실행
2. resources 폴더에 아래 파일 추가
   - log4j.properties 생성
   - jdbc.properties 생성
3. web.xml 수정
   - 인코딩 필터 설정
   - JSP 파일 UTF-8 인코딩 설정
4. servlet-context.xml 수정
   - annotation driven 설정
   - component 스캔 설정
   - 데이터베이스 설정
   - SessionFactory 설정
   - 트랜잭션 설정
5. 프로젝트의 lib 폴더에 ojdbc.jar 파일 추가

PhoneController 만들기

6. TB_BBS_Phone 테이블 생성
7. ModelPhone 클래스 생성
   - Configuration.xml 에 모델 추가
8. mapperPhone.xml 생성
   - Configuration.xml 에 mapperPhone 추가
9. Dao 생성
   - IDaoPhone 만들기
   - DaoPhone 만들기
     - 클래스에 @Repository 설정
     - 필드에 SqlSession 추가 하고 @Autorwired 설정
10. Service 생성
    - IServicePhone 만들기
    - ServicePhone 만들기
      - 클래스에 @Service 설정
      - 필드에 IDaoPhone 추가하고 @Autorwired 설정
11. JUnit Test 클래스 생성 & 테스트
12. PhoneController 생성 & 테스트

JSP(화면)에서 컨트롤러로 데이터 넘기는 방법

1. request.getParameter()로 값 넘겨 받기

```
@RequestMapping(value = "writeone1", method = RequestMethod.POST)
 public String writeOnePost1(Model model,  HttpServletRequest request) {
     String name         = request.getParameter("name");
     String manufacturer = request.getParameter("manufacturer");
     String price        = request.getParameter("price");
}
```

2. @RequestParam 로 값 넘겨 받기

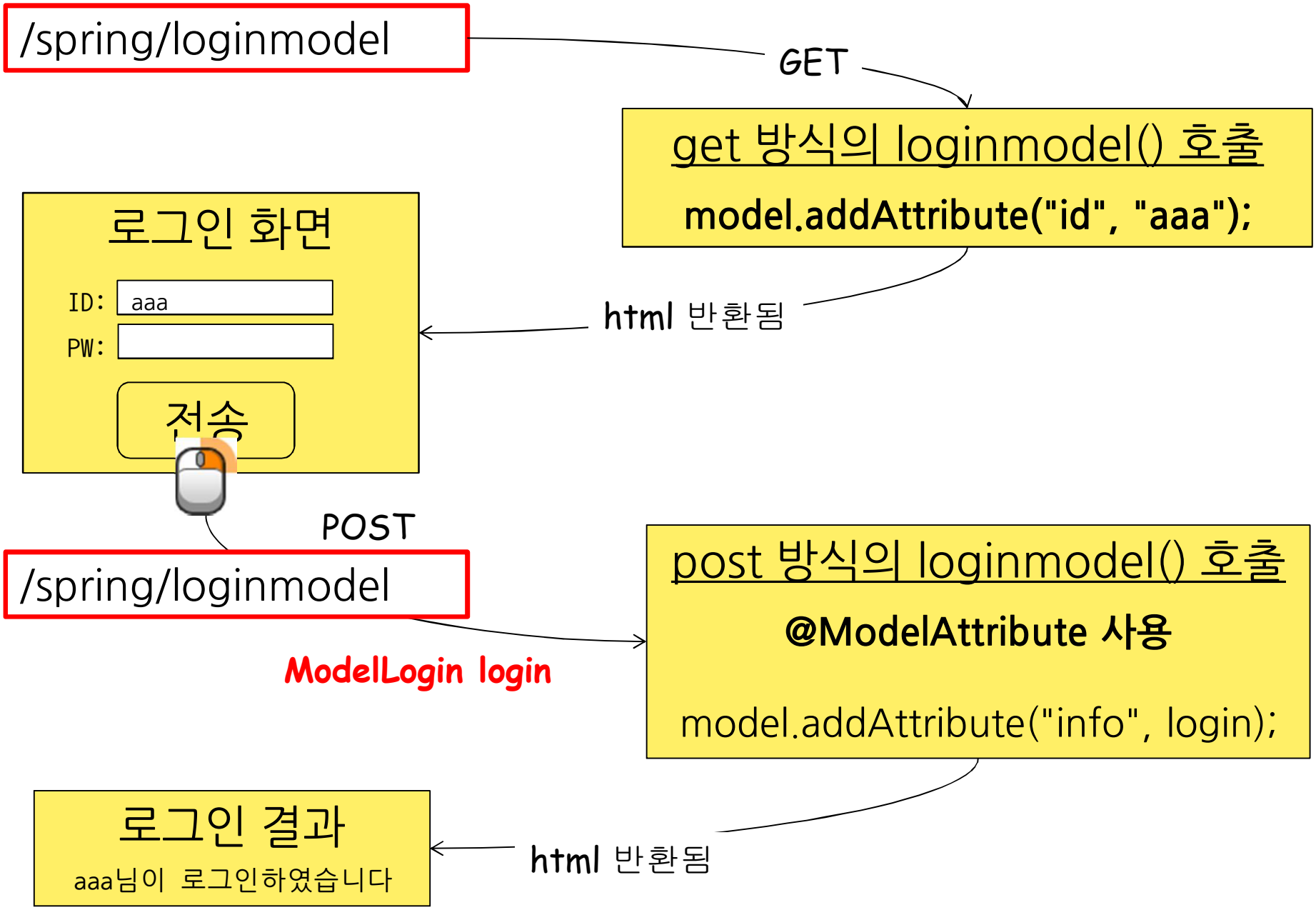
```
@RequestMapping(value = "writeone2", method = RequestMethod.POST)
public String writeOnePost2(Model model
        , @RequestParam(value="name"        , defaultValue="") String name
        , @RequestParam(value="manufacturer", defaultValue="") String manufacturer
        , @RequestParam(value="price"       , defaultValue="") int  price) {
}
```

3. @ModelAttribute 로 값 넘겨 받기

```
@RequestMapping(value = "writeone3", method = RequestMethod.POST)
public String writeOnePost3(
        @ModelAttribute ModelPhone phone,
        Model model) {
}
```

4. Repository로 배열 넘겨 받기

@ModelAttribute 을 이용한 login GET/POST 처리



폼(<form />)에서 POST로 @ModelAttribute 을 이용해 값을 넘겨 받는 방법

- src/main/webapp/WEB-INF/views/home/loginmodel.jsp 를 만드시오

```
<form action="/spring/loginmodel" method="post"
enctype="application/x-www-form-urlencoded">
    <label for="name">이름<input type="text" name="name" /></label>
    <label for="phone">폰 <input type="text" name="phone" /></label>
    <input type="submit" value="전송" />
</form>
```

- 실행 url: http://localhost/spring/loginmodel

```
@RequestMapping(value="/spring/loginmodel", method = RequestMethod.POST)
public String loginmodel(Model model, @ModelAttribute ModelLogin login) {

    model.addAttribute("info", login);

    return "home/loginmodelpost";
}
```

폼(<form />)에서 POST로 @ModelAttribute 을 이용해 값을 넘겨 받는 방법

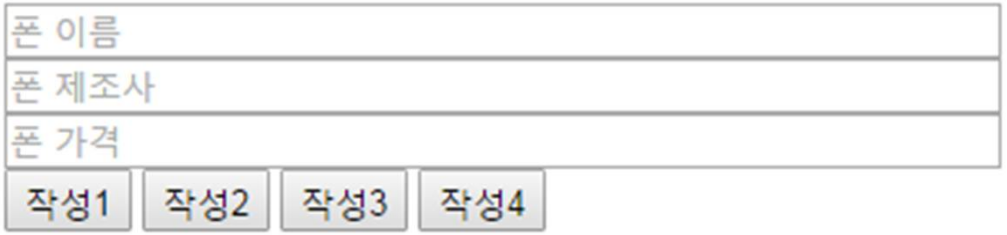
- src/main/webapp/WEB-INF/views/home/loginmodelpost.jsp 를 만드시오

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <meta charset="utf-8" />
</head>
<body>
    <h2> ID : ${info.id }</h2>
    <h2> PW : ${info.pw }</h2>
</body>
</html>
```

PhoneController DB 입력과 결과 출력

입력 값을 DB에 insert 하고 select 된 결과를 화면에 출력하시오

- http://localhost/phone/writeone



name	manufacturer	price
1	1	1

```
<script src="/resources/jquery-3.1.1.js"></script>
<script>
    $(document).ready( function(e){
        // 작성1 버튼에 클릭 이벤트 핸들러 설정
        $('#c1').click( function(e){
            var form = $('form');
            $(form).attr('action' , '/phone/writeone1');
            $(form).attr('method' , 'post');
            $(form).attr('enctype', 'multipart/form-data');
            $(form).sumbmit();
        });
    });
</script>
```

writeOneForm.jsp

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <meta charset="utf-8" />
    <style>
        table, td, th { border: 1px solid green; }
        th { background-color: green; color: white; }
    </style>
    <script src="/resources/js/jquery-3.1.1.js"></script>
    <script>
    </script>
</head>
<body>
    <form action="" method="post" enctype="multipart/form-data">
        <input type="text" name="name"  size="50" placeholder="폰 이름" required="required" /> <br>
        <input type="text" name="manufacturer" size="50" placeholder="폰 제조사" required="required"/> <br>
        <input type="text" name="price" size="50" placeholder="폰 가격" required="required" /> <br>

        <input type="button" id="writeone1" value="작성1" />
        <input type="button" id="writeone2" value="작성2" />
        <input type="button" id="writeone3" value="작성3" />
        <input type="button" id="writeone4" value="작성4" />
    </form>
</body>
</html>
```

writeOneResult.jsp

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <meta charset="utf-8" />
    <style>
        table, td, th { border: 1px solid green; }
        th { background-color: green; color: white; }
    </style>
</head>
<body>
    <table>
        <tr>
            <th>name</th>
            <th>manufacturer</th>
            <th>price</th>
        </tr>
        <tr>
            <td>${phone.name}</td>
            <td>${phone.manufacturer}</td>
            <td>${phone.price}</td>
        </tr>
    </table>
</body>
</html>
```

폼(<form />)에서 POST로 값을 넘겨 받는 방법

1. request.getParameter()로 값 넘겨 받기

```
@RequestMapping(value = "writeone1", method = RequestMethod.POST)
 public String writeOnePost1(Model model,  HttpServletRequest request) {
     String name         = request.getParameter("name");
     String manufacturer = request.getParameter("manufacturer");
     String price        = request.getParameter("price");
}
```

2. @RequestParam 로 값 넘겨 받기

```
@RequestMapping(value = "writeone2", method = RequestMethod.POST)
public String writeOnePost2(Model model
        , @RequestParam(value="name"        , defaultValue="") String name
        , @RequestParam(value="manufacturer", defaultValue="") String manufacturer
        , @RequestParam(value="price"       , defaultValue="") int  price) {
}
```

3. @ModelAttribute 로 모델 넘겨 받기

```
@RequestMapping(value = "writeone3", method = RequestMethod.POST)
public String writeOnePost3(
        @ModelAttribute ModelPhone phone,
        Model model) {
}
```

4. Repository로 배열 넘겨 받기

PhoneController DB 입력과 결과 출력

입력 값을 DB에 insert 하고 select 된 결과를 화면에 출력하시오

- http://localhost/phone/writelist

1
1
1

2
2
2

작성 취소

no	name	manufacturer	price
0	1	1	1
1	2	2	2

writeListForm.jsp

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <meta charset="utf-8" />
    <style>
        table, td, th { border: 1px solid green; }
        th { background-color: green; color: white; }
    </style>
</head>
<body>
    <form action="writelist" method="post" enctype="application/x-www-form-urlencoded">
        <input type="text" name="phoneItems[0].name" placeholder="폰 이름1" size="50"><br>
        <input type="text" name="phoneItems[0].manufacturer" placeholder="폰 제조사1" size="50"><br>
        <input type="text" name="phoneItems[0].price" placeholder="폰 가격1" size="50"><br>
        <hr>

        <input type="text" name="phoneItems[1].name" placeholder="폰 이름2" size="50"><br>
        <input type="text" name="phoneItems[1].manufacturer" placeholder="폰 제조사2" size="50"><br>
        <input type="text" name="phoneItems[1].price" placeholder="폰 가격2" size="50"><br>
        <hr>

        <input type="submit" value="작성"><input type="reset" value="취소">
    </form>
</body>
</html>
```

writeListResult.jsp

```
<%@ page session="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <style>
        table, td, th { border: 1px solid green; }
        th { background-color: green; color: white; }
    </style>
</head>
<body>
    <table>
        <tr>
            <th>no</th> <th>name</th> <th>manufacturer</th> <th>price</th>
        </tr>
        <!--  반복 구간 시작 -->
        <c:forEach var="phone" items="${list }" varStatus="status">
        <tr>
            <td>${status.index}</td>
            <td>${phone.name}</td>
            <td>${phone.manufacturer}</td>
            <td>${phone.price}</td>
        </tr>
        </c:forEach>
        <!--  반복 구간 끝 -->
    </table>
</body>
</html>
```

폼(form)에서 배열을 POST로 넘겨 받는 방법

- Repository 패턴을 사용하면 된다.

```
public class RepositoryPhone {

    private List<ModelPhone> phoneItems;

    // getter & setter
    public List<ModelPhone> getPhoneItems() {
        return phoneItems;
    }
    public void setPhoneItems(List<ModelPhone> phoneItems) {
        this.phoneItems = phoneItems;
    }

    // 생성자
    public RepositoryPhone() {
        super();
    }

    // toString()
}
```

폼(form)에서 배열로 넘어온 값을 받는 방법

```
@Controller
@RequestMapping("/phone")
public class PhoneController {

    @RequestMapping(value = "writelist", method = RequestMethod.GET)
    public String writeListGet(Model model) {
        return "phone/writeListForm";
    }

    @RequestMapping(value = "writelist", method = RequestMethod.POST)
    public String writeListPost( Model model
            ,  @ModelAttribute RepositoryPhone phone ) {
        List<ModelPhone> phonelist = phone.getPhoneItems();

        // DB insert. 어떻게?

        model.addAttribute("list", phonelist);

        return "phone/writeListResult";
    }
}
```

bbs 게시판 만들기

1. 스프링 프로젝트 생성
   - gradle 프로젝트로 변경:
     Configure >> Convert to Gradle ...
   - build.gradle 수정
   - gradle >> refresh dependency 실행
   - gradle >> refresh all 실행
2. resources 폴더에 아래 파일 추가
   - log4j.properties 생성
   - jdbc.properties 생성
3. web.xml 수정
   - 인코딩 필터 설정
   - JSP 파일 UTF-8 인코딩 설정
4. servlet-context.xml 수정
   - annotation driven 설정
   - component 스캔 설정
   - 데이터베이스 설정
   - SessionFactory 설정
   - 트랜잭션 설정
5. 프로젝트의 lib 폴더에 ojdbc.jar 파일 추가

bbs 게시판 만들기

6. 게시판 테이블 생성 TB_BBS_xxxx
7. 게시판 모델 클래스 생성
    - Configuration.xml 에 모델 추가
8. mapperBoard.xml 추가
    - Configuration.xml 에 mapper 추가
9. Dao 생성
    - IDaoBoard, DaoBoard 만들기
    - IDaoUser, DaoUser 만들기
        - 클래스에 @Repository 설정
        - 필드에 @Autorwired 설정
10. Service 생성
    - IServiceBoard, ServiceBoard 만들기
    - IServiceUser, ServiceUser 만들기
        - 클래스에 @Service 설정
        - 필드에 @Autorwired 설정
11. JUnit Test 클래스 생성

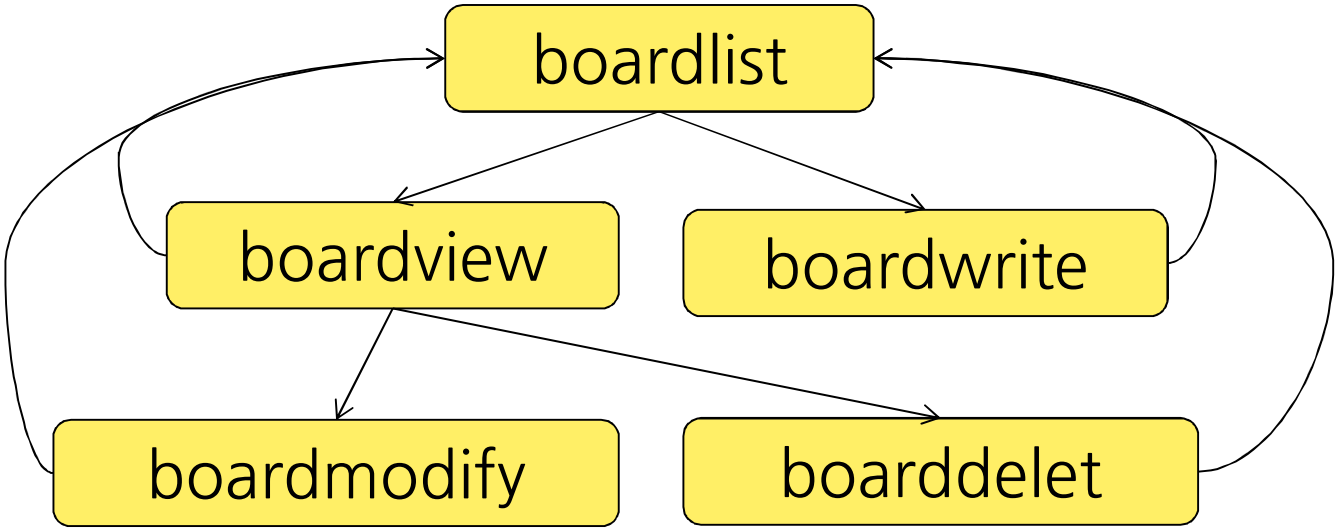
bbs 게시판 만들기

12.Controller 생성

- HomeController 생성
- BoardController 생성
  - /board/boardlist 액션 메서드 작성(GET)
    - boardlist의 tr 에 클릭 이벤트 달기
  - /board/boardview 액션 메서드 작성(GET)
  - /board/boardwrite 액션 메서드 작성(GET,POST)
  - /board/boardmodify 액션 메서드 작성(GET,POST)
  - /board/boarddelete 액션 메서드 작성(POST)

  - /board/articlelist 액션 메서드 작성(GET)
  - /board/articleview 액션 메서드 작성(GET)
  - /board/articlewrite 액션 메서드 작성(GET,POST)
  - /board/articlemodify 액션 메서드 작성(GET,POST)
  - /board/articledelete 액션 메서드 작성(POST)
- UserController 생성

board 페이지 작성 순서

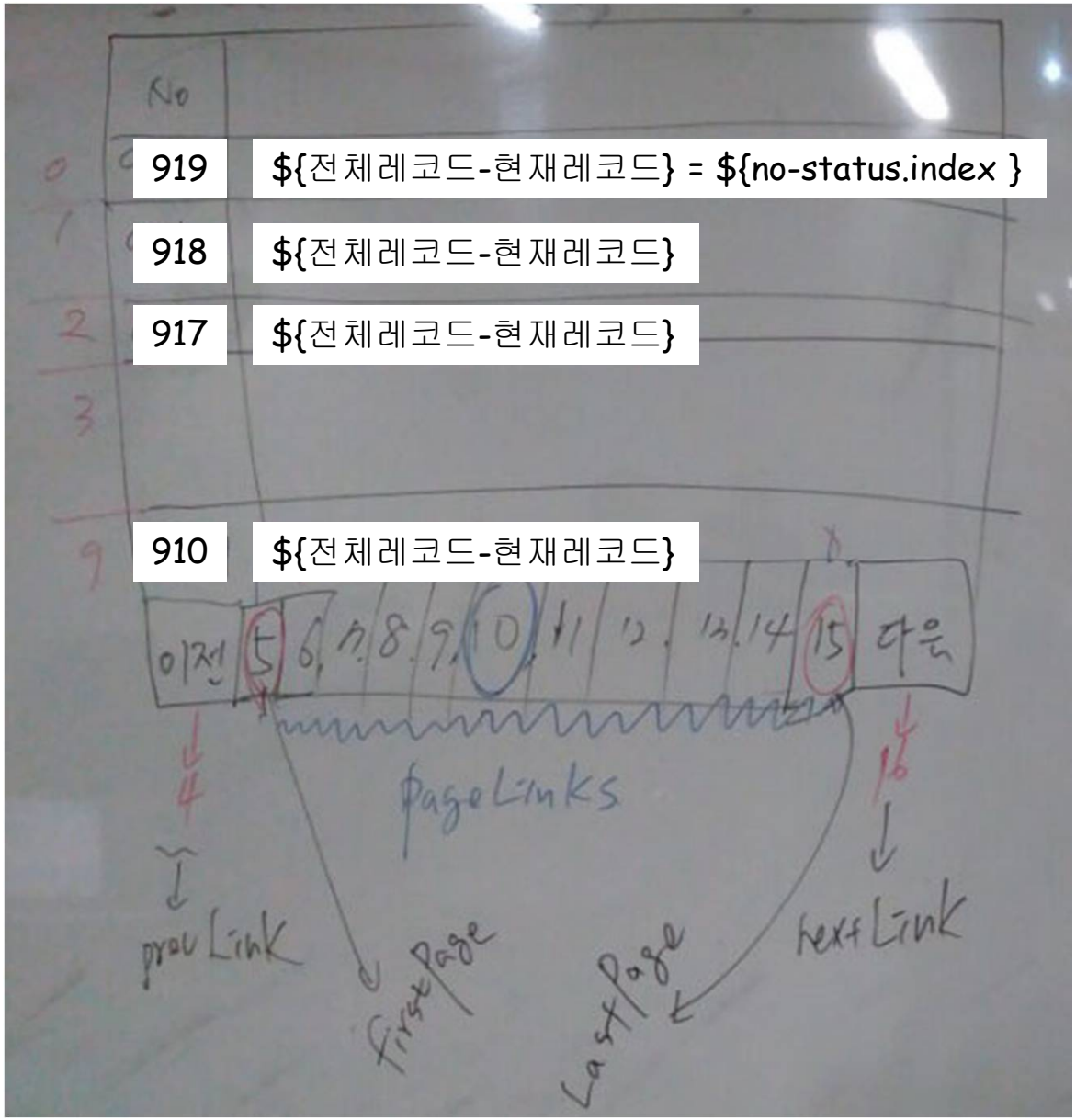


1. /board/boardlist?curPage=1&searchWord= 액션 메서드 작성(GET)
   - boardlist의 tr을 클릭하면 boardview가 열리게
2. /board/boardview/17?curPage=1&searchWord= 액션 메서드 작성(GET)
3. /board/boardmodify/17?curPage=1&searchWord= 액션 메서드 작성(GET,POST)
4. /board/boarddelete/17?curPage=1&searchWord= 액션 메서드 작성(POST)

5. /board/boardwrite?curPage=1&searchWord= 액션 메서드 작성(GET,POST)

6. /board/boardlist?curPage=1&searchWord= 에 페이징 기능 추가
7. /board/boardlist?curPage=1&searchWord= 에 검색 기능 추가

boardlist 페이징 처리



페이징 처리 클래스

페이징 처리 클래스 테스트 코드

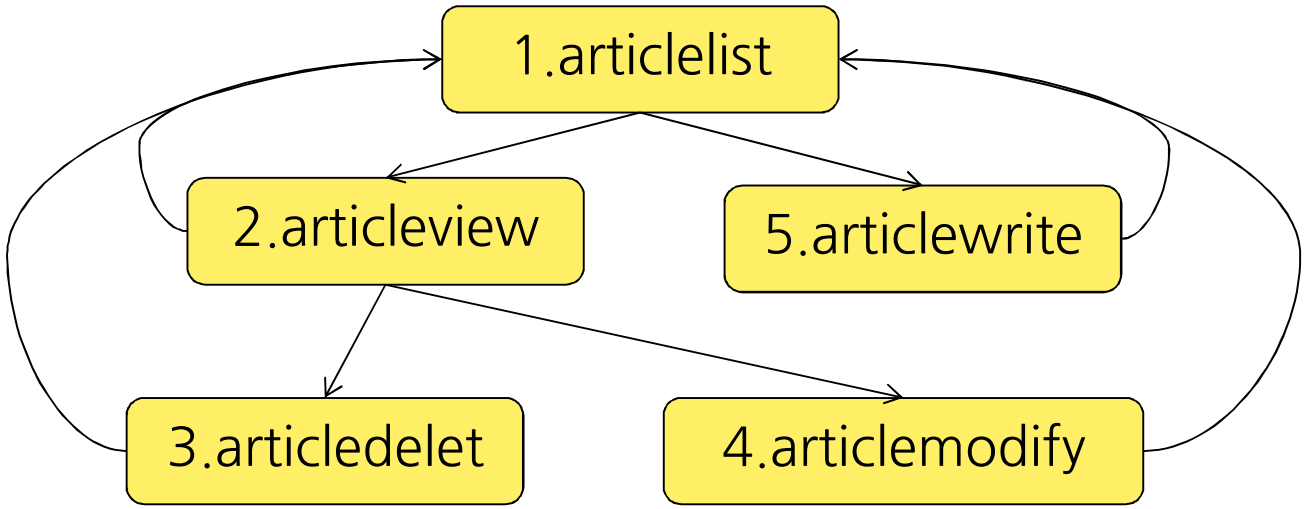
controller code

```
// step1. 전체 게시글 갯수 가져오기
int totalRecord = boardsrv.getBoardTotalRecord(searchWord);

// step2. 페이지 처리
PagingHelper paging = new PagingHelper(totalRecord, curPag);

// step3. 출력할 게시글 가져오기
List<ModelArticle> boardList = boardsrv.getBoardPaging(searchWord, paging.getStartRecord(),
paging.getEndRecord());

// step4. jsp data binding
model.addAttribute("prevLink"  , paging.getPrevLink() );
model.addAttribute("nextLink"  , paging.getNextLink() );
model.addAttribute("pageLinks" , paging.getPageLinks());
model.addAttribute("curPage"   , curPage              );
model.addAttribute("searchWord", searchWord           );
```

jsp code

```
<div id="paging" style="text-align: center;">
  <script type="text/javascript">
    var goList = function(page) {
        window.location.href = './boardlist?curPage=' + page + '&searchWord=${param.searchWord}';
    }
  </script>

  <c:if test="${totalFirst > 0 }"> <a href="javascript:goList('${totalFirst }'); return false;">[
처음]</a> </c:if>
  <c:if test="${prevLink   > 0 }"> <a href="javascript:goList('${prevLink   }'); return false;">[
이전]</a> </c:if>
  <c:forEach var="i" items="${pageLinks }" varStatus="stat">
    <c:choose>
      <c:when test="${curPage == i}"> <span class="bbs-strong">${i }</span> </c:when>
      <c:otherwise> <a href="javascript:goList('${i }'); return false;">${i }</a> </c:otherwise>
    </c:choose>
  </c:forEach>
  <c:if test="${nextLink > 0 }"> <a href="javascript:goList('${nextLink }'); return false;">[다음
]</a> </c:if>
  <c:if test="${totalLast> 0 }"> <a href="javascript:goList('${totalLast}'); return false;">[마지
막]</a> </c:if>
</div>
```

Article 페이지 작성 순서



※ 번호순으로 액션 메서드 작성

- **free는 boardcd 값을, 17은 arcticleno 값을 의미한다.**

1. /board/articlelist/**free**?curPage=1&searchWord= (GET)
2. /board/articleview/**free**/**17**?curPage=1&searchWord= (GET)
3. /board/articlewrite/**free**?curPage=1&searchWord= (GET,POST)
4. /board/articlemodify/**free**/**17**?curPage=1&searchWord= (GET,POST)
5. /board/articledelete/**free**/**17**?curPage=1&searchWord= (POST)

Create a Form Dynamically Via Javascript

```
<script>

    var f = document.createElement('form');
    f.setAttribute('method','post');
    f.setAttribute('action','/board/articledelete/${boardcd}/${articleno}');
    f.setAttribute('enctype','application/x-www-form-urlencoded');

    var i = document.createElement('input'); //input element, hidden
    i.setAttribute('type','hidden');
    i.setAttribute('name','curPage');
    i.setAttribute('value','${curPage}');
    f.appendChild(i);

    var i = document.createElement('input'); //input element, hidden
    i.type = 'hidden';
    i.name = 'searchWord';
    i.value = '${searchWord}';
    f.appendChild(i);

    f.submit();

</script>
```

UploadController 만들기

- servlet-context.xml 에 MultipartResolver 추가
- RepositoryFiles 작성
- ModelAttachfile 모델 작성
- UploadController 컨트롤러 작성
- uploadfile.jsp 작성
- uploadsuccess.jsp 작성

Rest 서비스 만들기

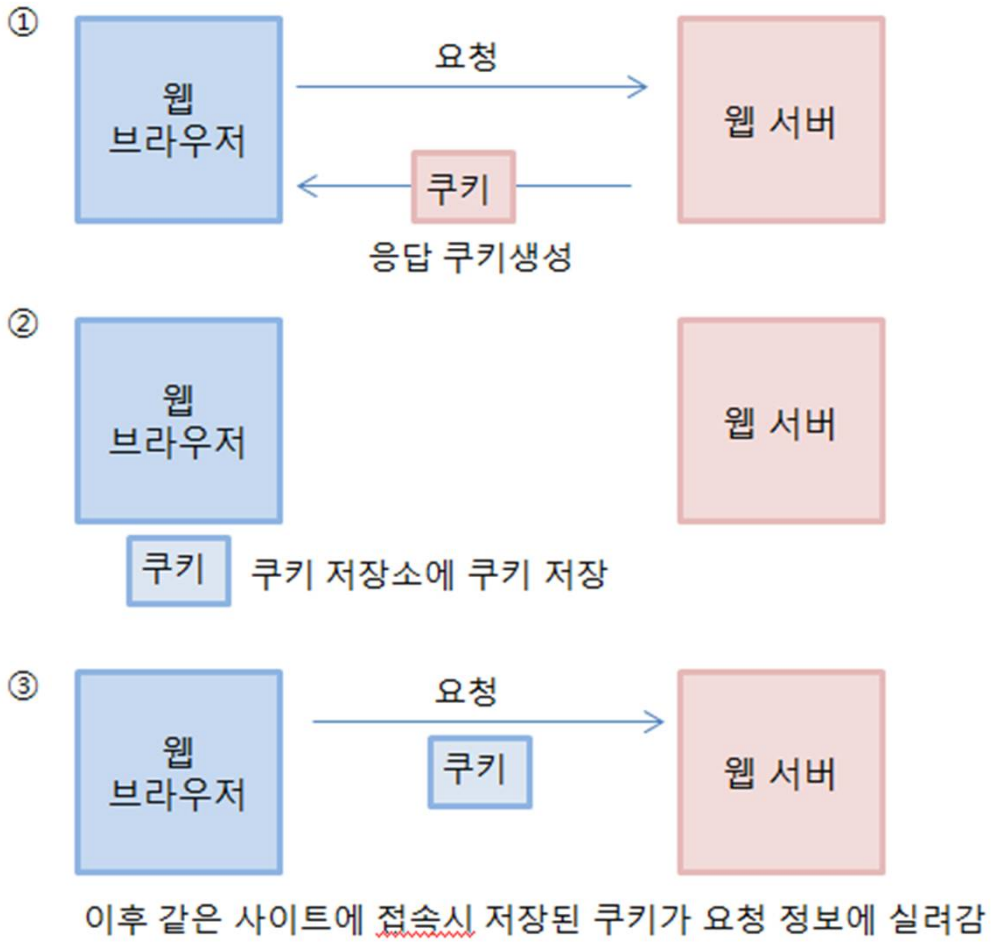
1. servlet-context.xml 에 jsonView 추가

웹 데이터 scope

- 클라이언트측 데이터 저장
  - cookie
- 서버측 데이터 저장
  - page
  - request
  - **session**
  - application
- 영구 데이터 저장
  - 데이터베이스

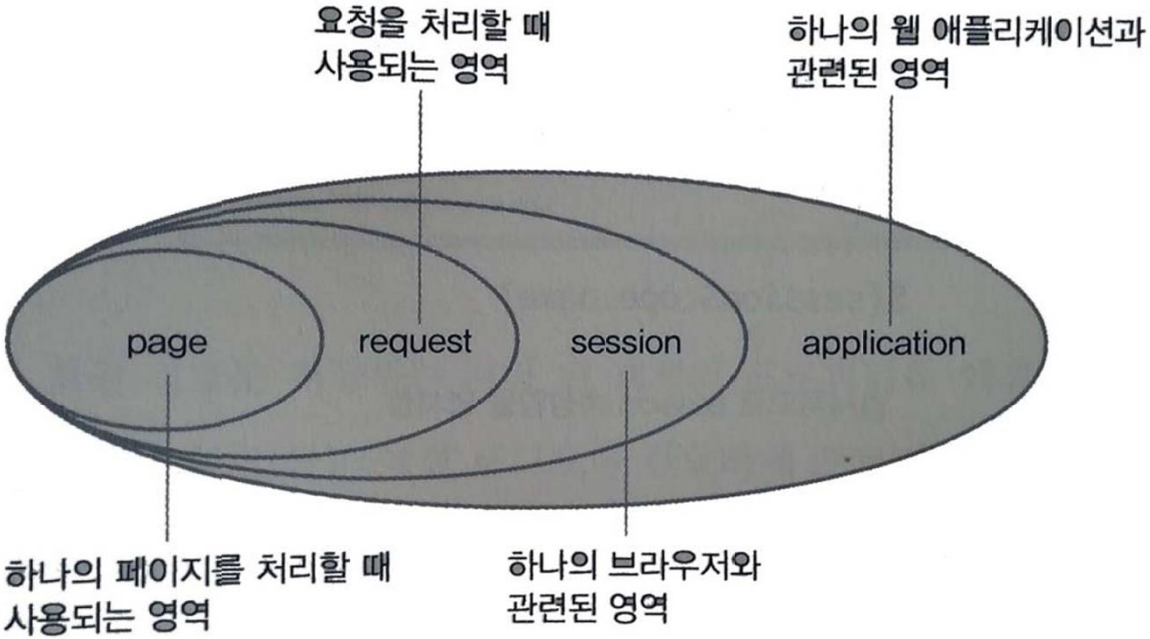
클라이언트측 데이터 저장

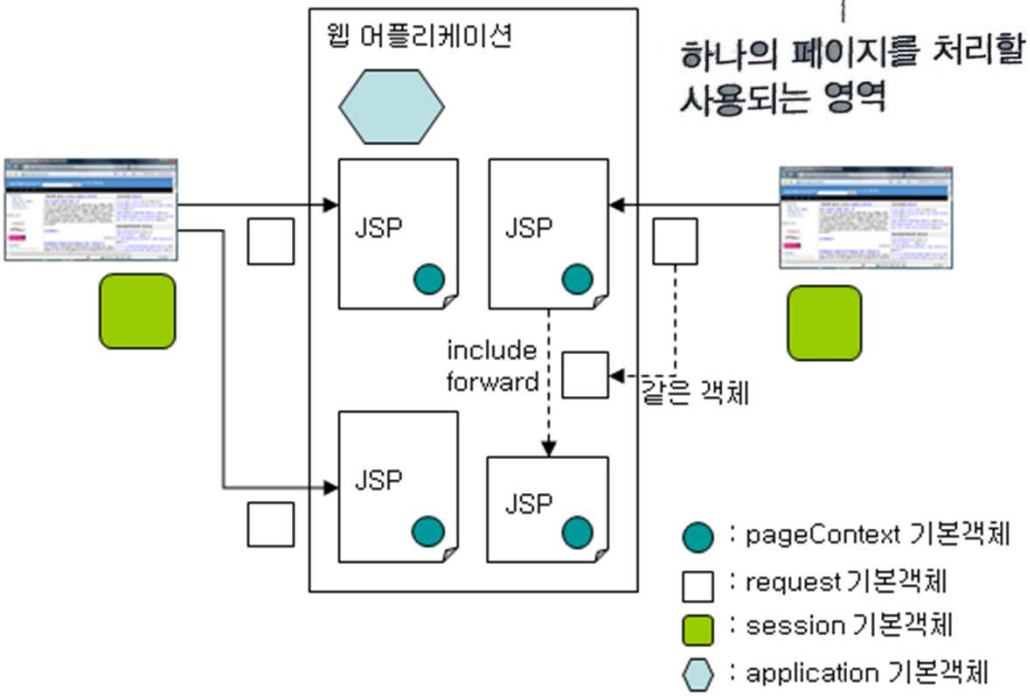
- 쿠키
  - 쿠키는 클라이언트 데이터 저장 방법이다.



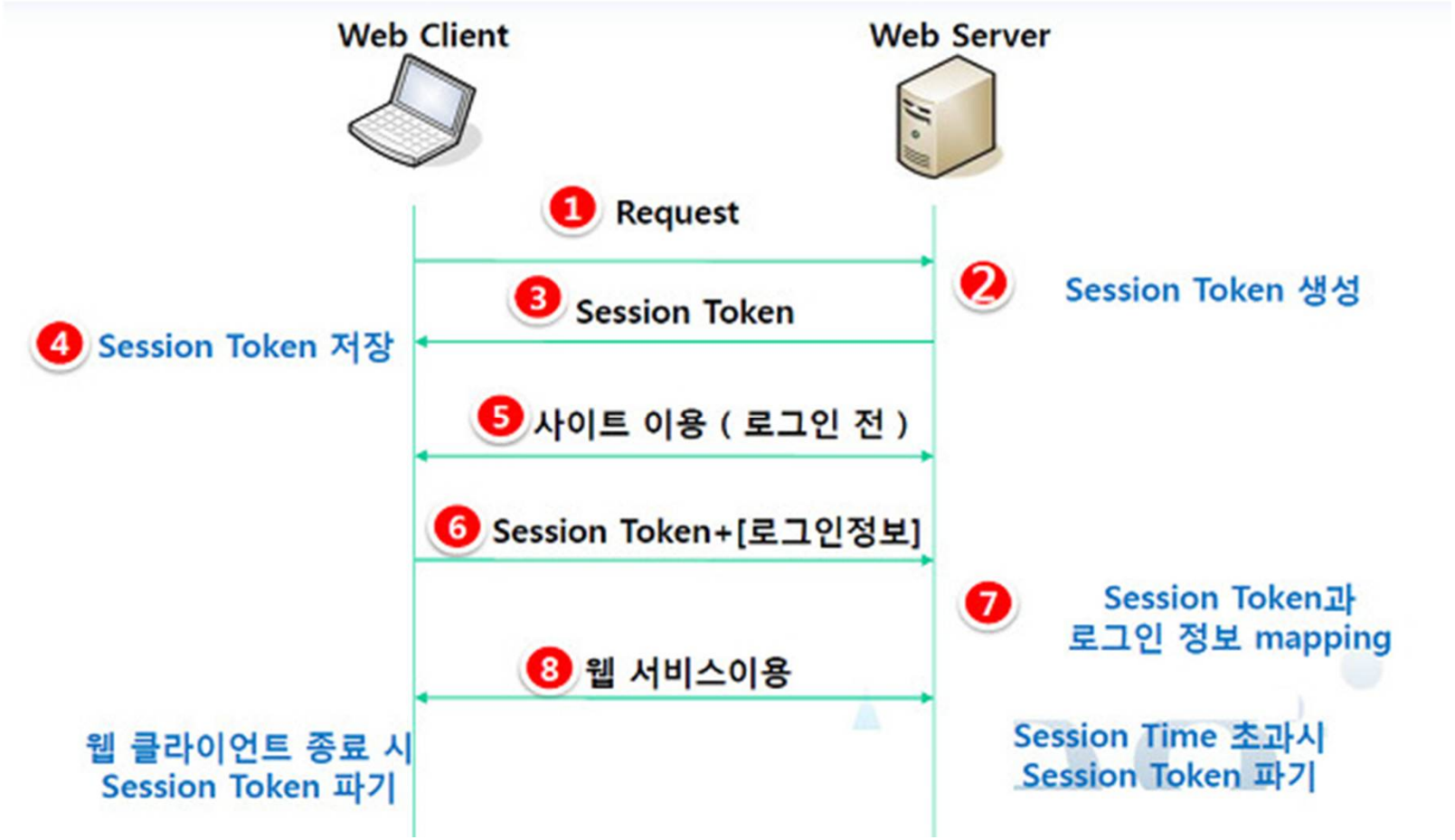
서버측 데이터 저장

- 서버측 데이터 저장
  - page
  - request
  - session
  - application





세션


Web Client
Web Server
1 Request
2 Session Token 생성
3 Session Token
4 Session Token 저장
5 사이트 이용 (로그인 전)
6 Session Token+[로그인정보]
7 Session Token과
로그인 정보 mapping
8 웹 서비스이용
웹 클라이언트 종료 시
Session Token 파기
Session Time 초과시
Session Token 파기

쿠키 vs 세션

- 쿠키란?
  - 쿠키란 서버측에서 클라이언트측에 상태 정보를 저장하고 추출할 수 있는 메커니즘
  - 클라이언트의 매 요청마다 웹 브라우저로부터 서버에게 전송되는 정보패킷의 일종
  - HTTP에서 클라이언트의 상태 정보를 클라이언트의 하드 디스크에 저장하였다가 필요 시 정보를 참조하거나 재사용할 수 있음.
  - 사용 예
    - 방문했던 사이트에 다시 방문 하였을 때 아이디와 비밀번호 자동 입력
    - 팝업에서 "오늘 이 창을 다시 보지 않음" 체크
  - 쿠키의 제약조건
    - 클라이언트에 총 300개까지 쿠키를 저장할 수 있다
    - 하나의 도메인 당 20개의 값만을 가질 수 있다
    - 하나의 쿠키 값은 4096Byte까지 저장 가능하다

하나의 도메인에서 설정한 쿠키값이 20개를 초과하면 가장 적게 사용된 쿠키부터 지워짐. 또한 쿠키는 기존에 설정한 값이 있는 곳에 값을 저장하거나 배열형태의 쿠키에 단일 값을 저장하려고 할 때 아무런 경고 없이 덮어쓰기 때문에 주의를 해야 한다.

- 세션이란?
  - 세션이란 클라이언트와 웹서버 간에 네트워크 연결이 지속적으로 유지되고 있는 상태를 말함
  - 클라이언트가 웹서버에 요청하여 처음 접속하면 JSP(혹은ASP)엔진은 요청한 클라이언트에 대하여 유일한 ID를 부여하게 되는데, 이 ID를 세션이라 부른다
  - 세션 ID를 임시로 저장하여 페이지 이동 시 이용하거나, 클라이언트가 재 접속 했을 때 클라이언트를 구분할 수 있는 유일한 수단이 된다
  - 세션의 장점
    - 각각의 클라이언트마다 고유의 ID 부여
    - 세션 객체마다 저장해 둔 데이터를 이용하여 서로 다른 클라이언트의 요구에 맞게 서비스 제공
    - 클라이언트 자신만의 고유한 페이지를 열어놓아서 생길 수 있는 보안상의 문제 해결 용이

쿠키 vs 세션

- **쿠키** : 클라이언트가 요청한 데이터를 브라우저에서 보관한다.
- **세션** : 클라이언트가 요청한 데이터를 브라우저 마다 서버에 보관한다.

구분	쿠키	세션
저장 위치	클라이언트	서버
저장 형식	Text로 저장	Object 로 저장
종료 시점	expire data가 지났거나 expire data 설정하지 않았으면 브라우저 종료 시나	브라우저를 닫거나, 서버에 의해 지워지는 경우
자 원	클라이언트의 자원을 사용	서버의 자원을 사용
용도	사이트 재 방문시 사용자 정보를 기억하기 위해 사용 (ID, PW, 팝업창 제한등)	서버를 이용하는 동안에 사용자 정보를 유지하기 위해 사용
용량 제한	한 도메인 당 20개, 쿠키 하나 당 4KB, 총 300개	서버가 허용하는 한 용량에 제한이 없음

쿠키 생성 및 사용

(1) 생성

```
Cookie cooke = new Cookie(String name, String value);
```

(2) 쿠키 추가(생성후에 반드시 추가)

```
response.addCookie(cookie);
```

(3) 값을 수정

```
cookie.setValue(newValue);
```

(4) 읽기

- 쿠키를 읽어 올 때

```
Cookie[] cookies = request.getCookies();
```

- 쿠키 이름 읽기

```
String cookies[i].getName();
```

- 쿠키 값읽기

```
String cookies[i].getValue();
```

(5) 쿠키의 수명(지속시간)

```
cookie.setMaxAge(int expiry);
```

세션 사용 및 설정

(1) 세션 설정 - 세션 값은 객체 형태

```
session.setAttribute("memId","test");
session.setAttribute("user", new ModelUser() );
```

(2) 세션 가져오기 - 리턴 타입이 object이므로 String으로 변환

```
String id = (String)seesion.getAttribute("memId");
ModelUser user = (ModelUser)seesion.getAttribute("user");
String userid = user.getUserid();
```

(3) 세션 삭제

```
session.removeAttribute("memId");
session.removeAttribute("user");
```

(4) 세션 속성의 이름을 반환

```
Enumeration names = session.getAttributeNames();
```

(5) 세션의 모든 속성 해제

```
session.invalidate();
```

(6) 세션의 최대 유지시간 설정(초단위 설정)

```
session.setMaxInactiveInterval(int interval);
```

(7) 세션의 최대 유지시간 반환

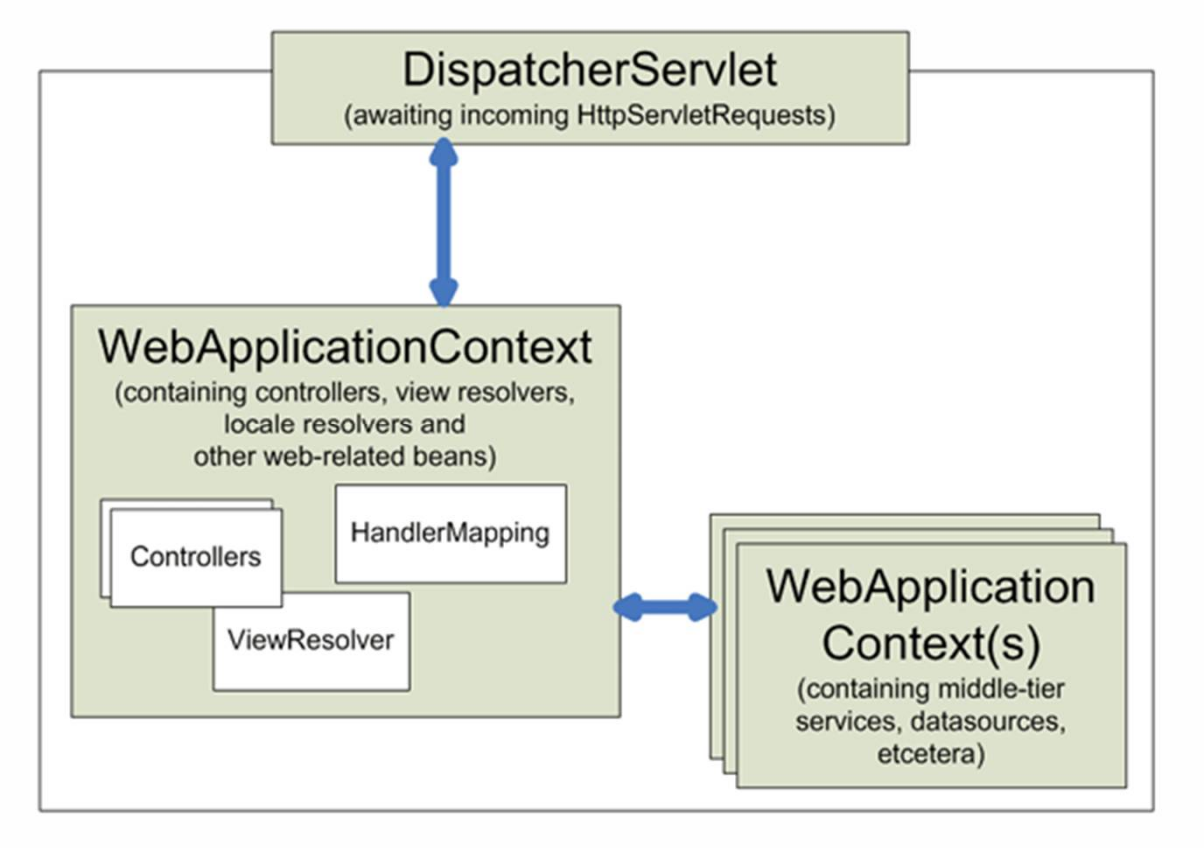
```
int sec = session.getMaxInactiveInterval();
```

세션을 이용한 login / logout 예제 코드

```
@RequestMapping(value = "login", method = RequestMethod.POST)
public String login(
          @RequestParam String userid
        , @RequestParam String passwd
        , HttpSession session) {
    ModelUser user = serviceuser.login(userid, passwd);

    session.setAttribute( "user" , user);
    return "redirect:/";
}

@RequestMapping(value = "logout", method = RequestMethod.GET)
public String login( HttpSession session) {

    session.removeAttribute( "user" );
    return "redirect:/";
}
```

Scope	Annotation	Description
Request	@RequestScoped	Single HTTP request
Session	@SessionScoped	Multiple HTTP requests
Application	@ApplicationScoped	Shared state across all interactions in a web application
Conversation	@ConversationScoped	multiple invocations of the JavaServer Faces lifecycle
Dependent	@Dependent	Its lifecycle depends on the client it serves (Default scope)
Singleton	@Singleton	State shared among all clients

Spring MVC


DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers,
locale resolvers and
other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication
Context(s)
(containing middle-tier
services, datasources,
etcetera)