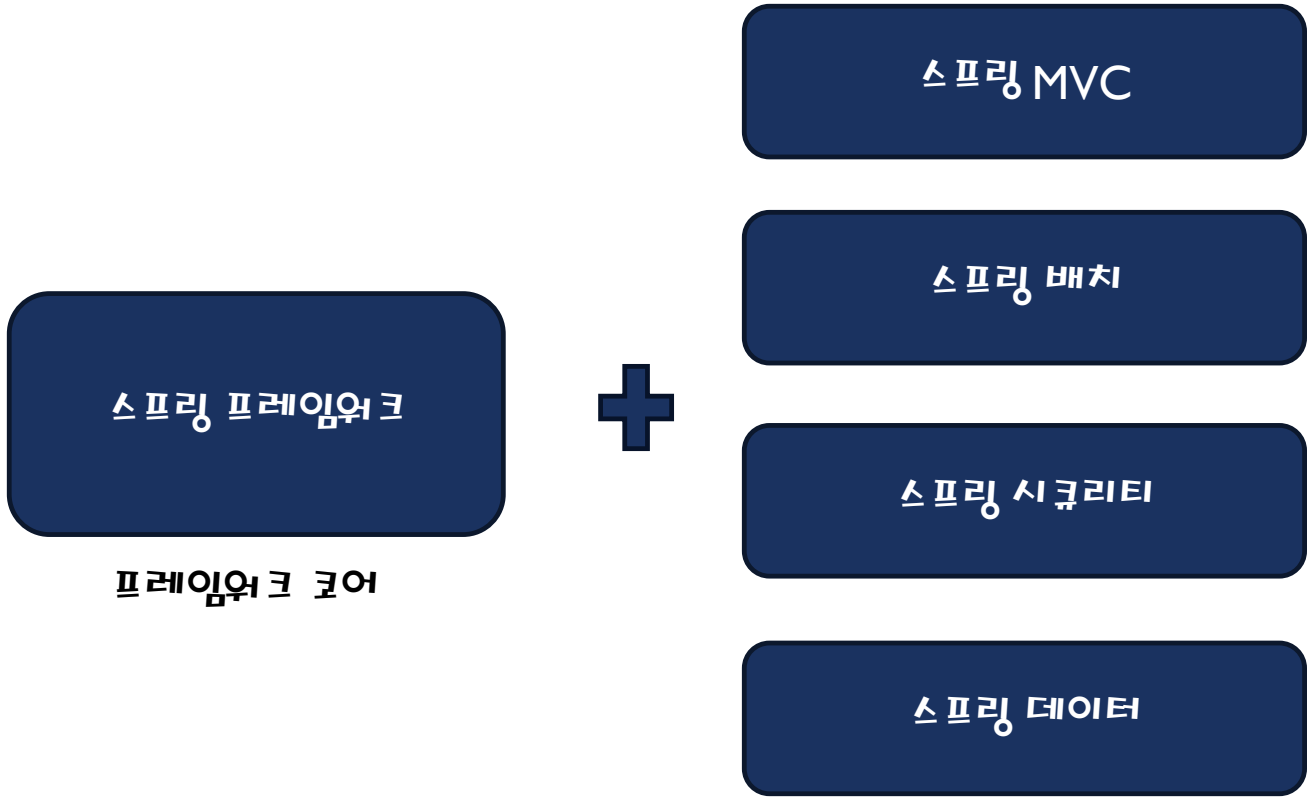
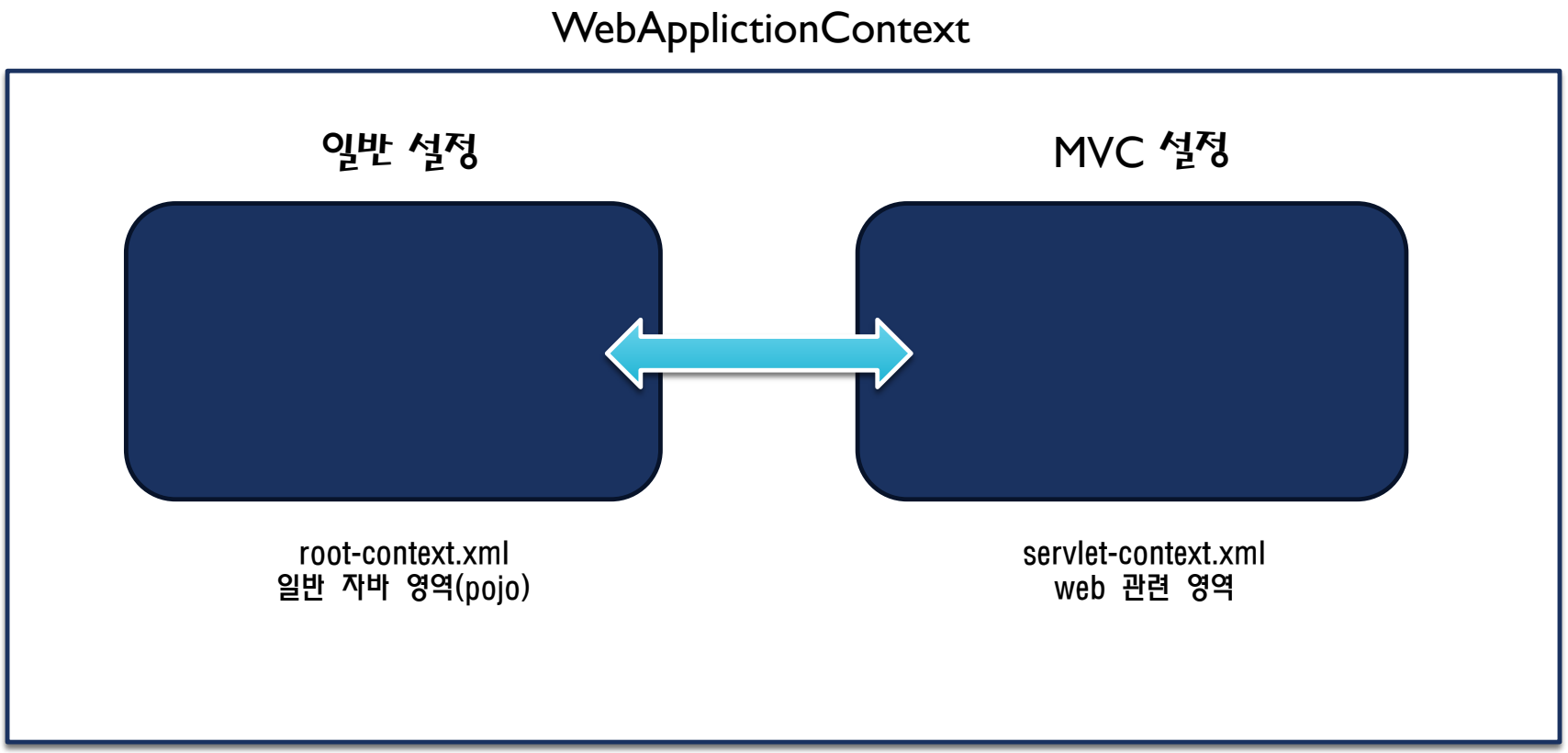
3. 스프링 MVC

1. 스프링 MVC 구조
2. 스프링 MVC 컨트롤러
3. 컨트롤러
4. 파라미터 수집과 변환

1.1 스프링 MVC



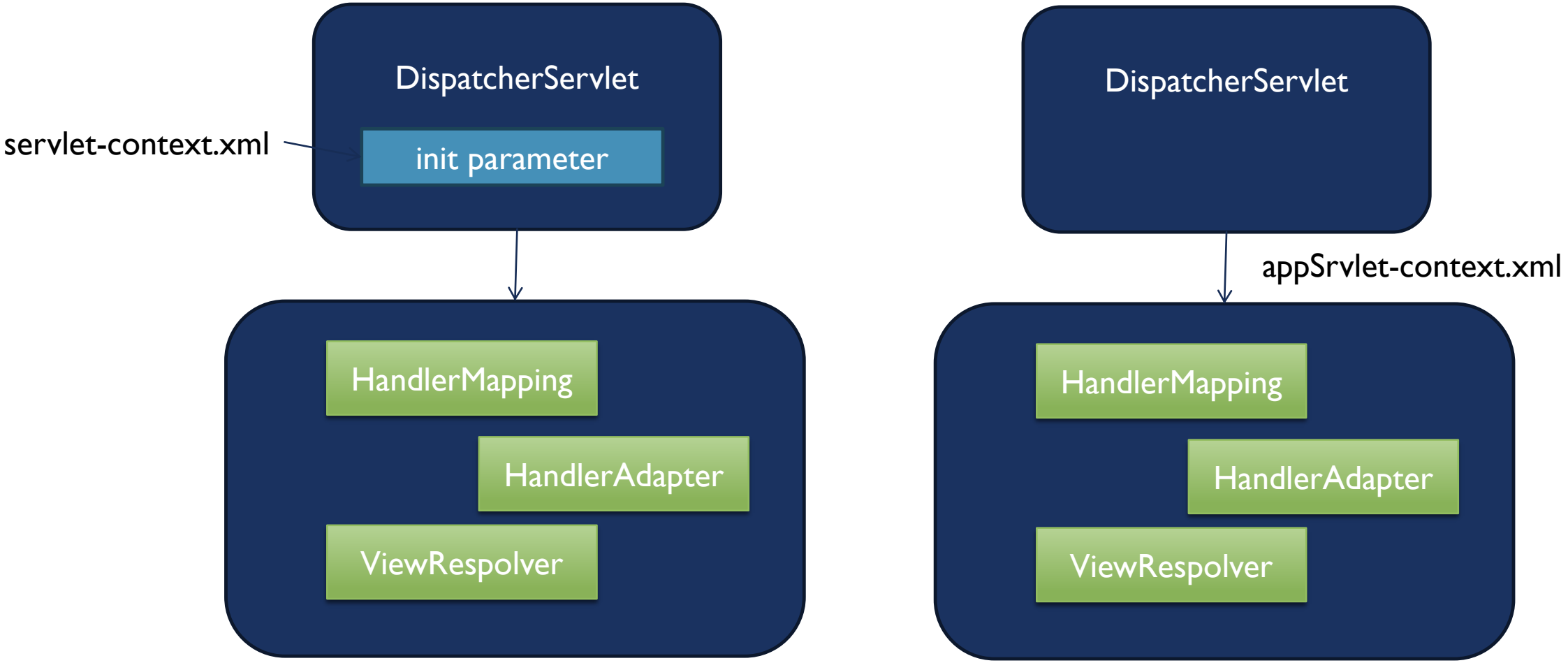
1.2 스프링 MVC 설정


WebApplictionContext
일반 설정
MVC 설정
root-context.xml
일반 자바 영역(pojo)
servlet-context.xml
web 관련 영역

1.3 web.xml

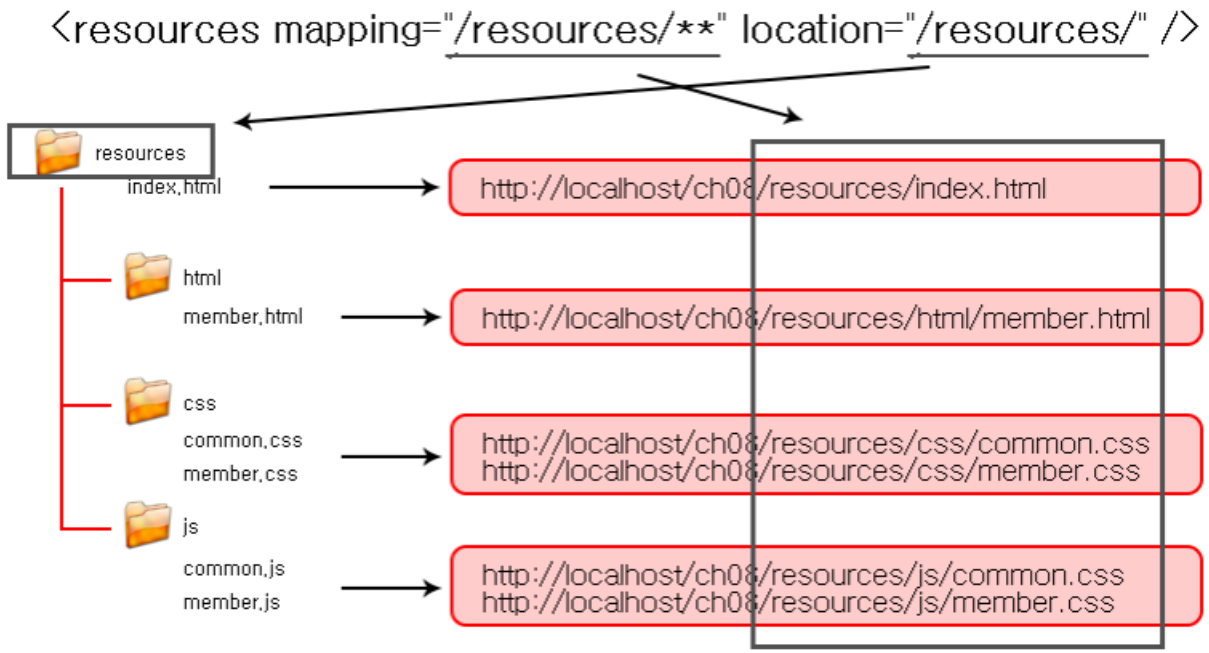
```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- Processes application requests -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

1.4 dispatcherservlet

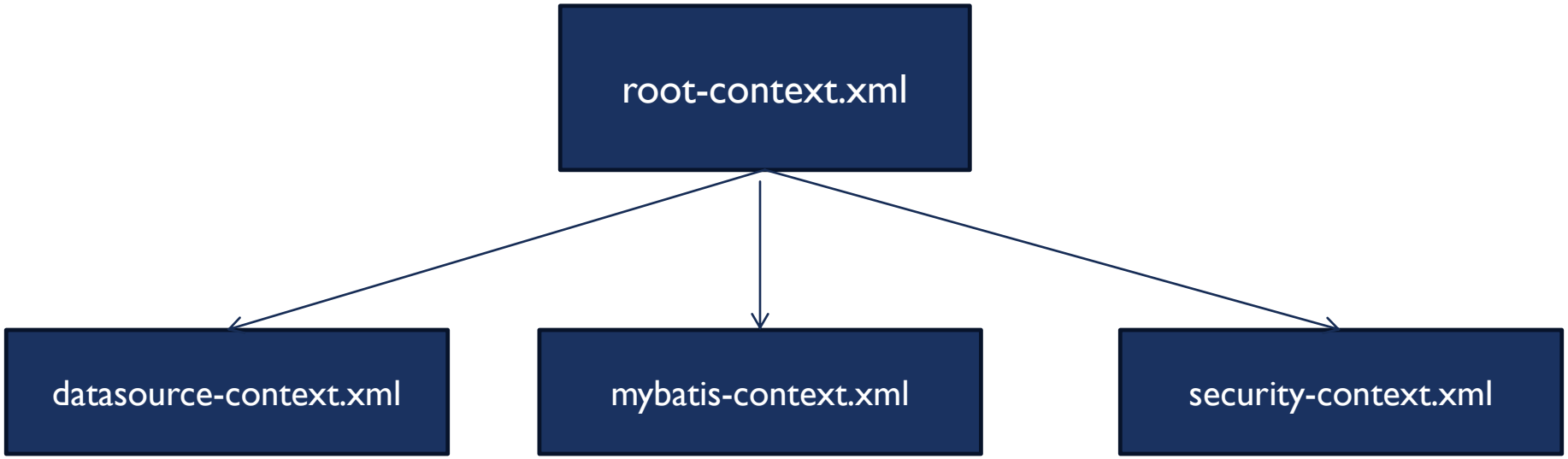


1.5 servlet-context.xml

<resources mapping="" location=""/>

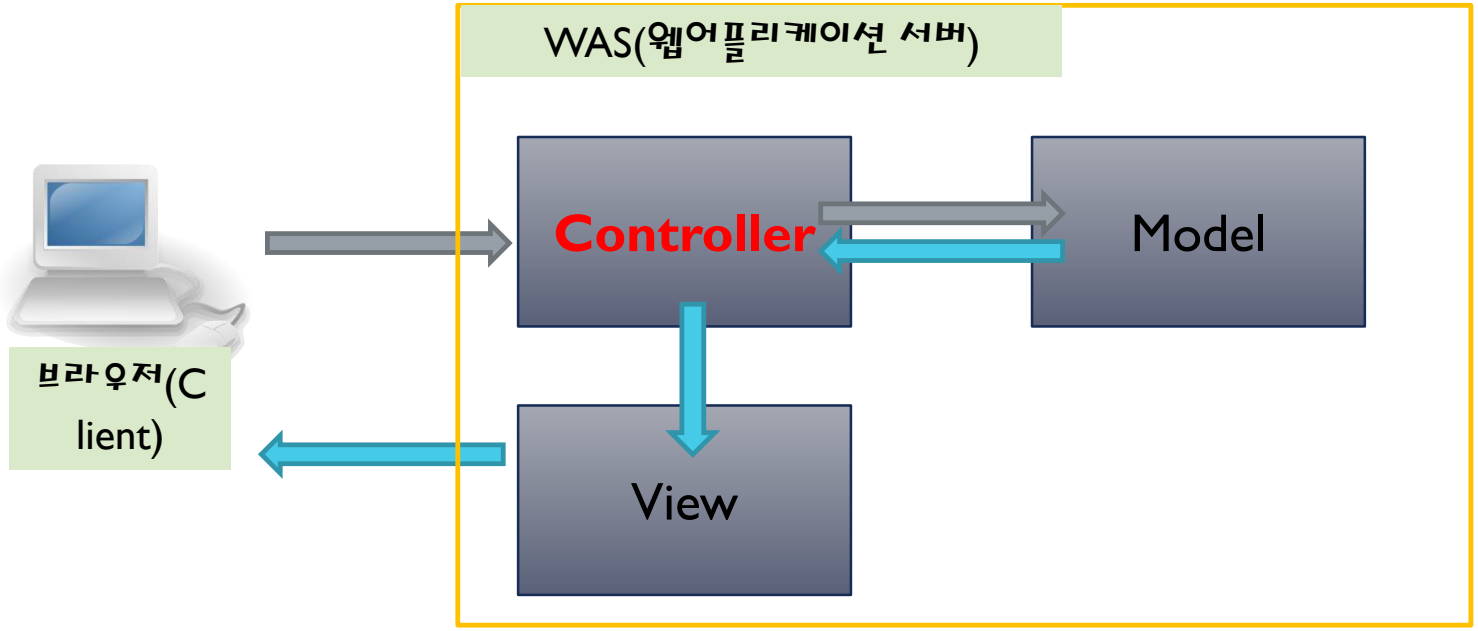


1.6 설정파일 분리


root-context.xml
datasource-context.xml
mybatis-context.xml
security-context.xml

2.1 모델2 방식

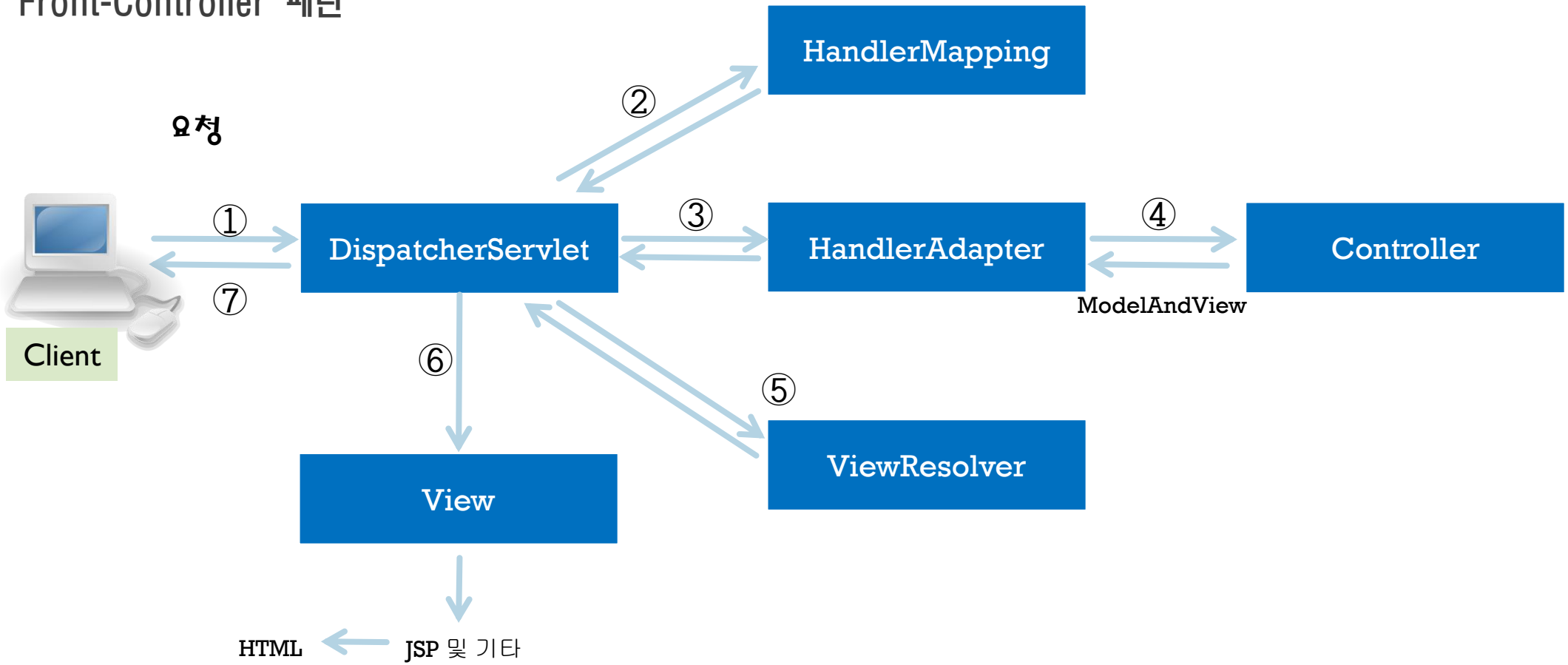
- 화면과 데이터 처리를 분리해서 재사용이 가능하도록 하는 구조



- Model: 데이터 혹은 데이터를 처리하는 영역
- View: 결과화면을 만들어 내는데 사용하는 자원
- Controller: 웹의 요청(request)를 처리하는 영역으로 뷰와 모델 사이의 중간통신 역할

2.2 Spring MVC 구조

- Front-Controller 패턴



2.2 Spring MVC 구조

1. 사용자의 모든 요청은 Front-Controller인 DispactherServlet을 통해서 처리

2. HandlerMapping은 Request의 처리를 담당하는 컨트롤러를 찾기 위해서 존재.
   - @RequestMapping 어노테이션 참조

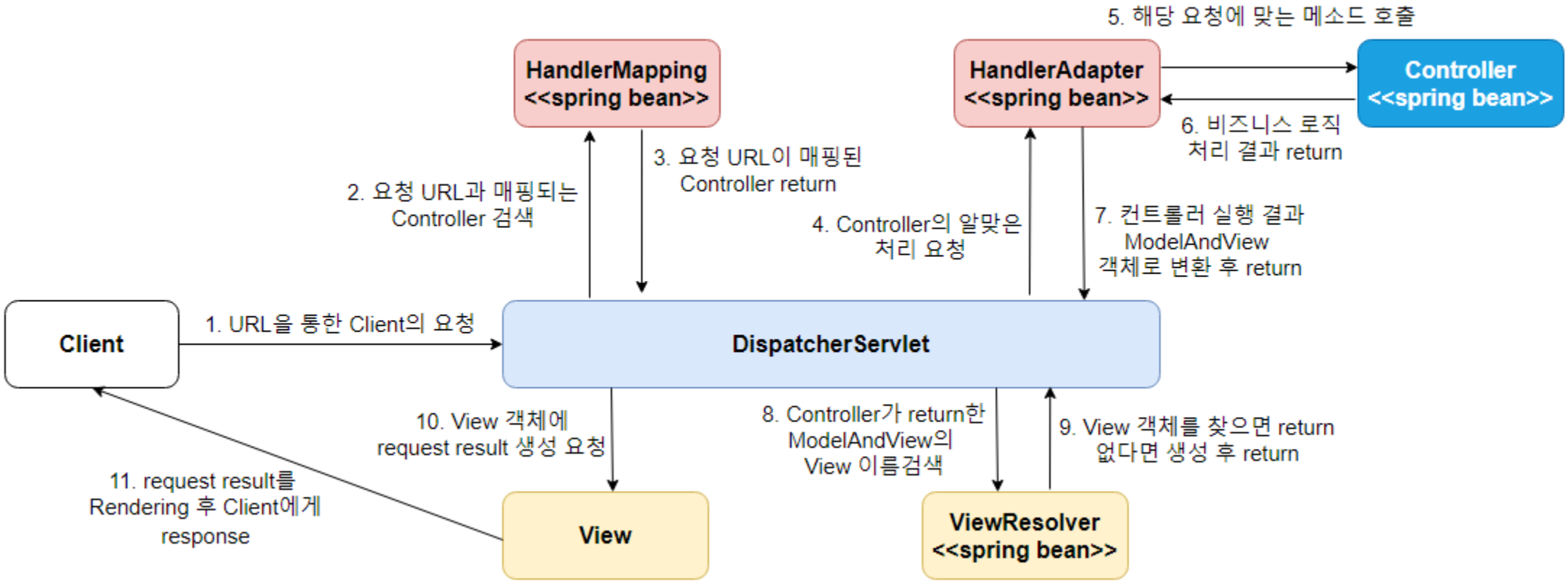
3. 컨트롤러를 찾았다면 HandlerAdapter를 이용해서 컨트롤러를 동작

4. Controller는 실제 요청을 처리하는 로직을 작성.
   - 이때 view에 전달할 데이터는 Model 객체에 담아서 전달

5. ViewResolver는 컨트롤러가 반환한 결과를 어떤 View를 통해서 처리할지 결정

6. View는 실제로 응답 보내야 하는 데이터를 jsp등을 이용해서 생성하는 역할을 함
   - 만들어진 응답은 DispathcerServlet을 통해서 전송

2.2 Spring MVC 구조



3.1 컨트롤러

- @Controller

servlet-context.xml

```
<component-scan />
<annotation-driven/>
```

```
@Controlle
public class HomeController {

  @RequestMapping("/")
  public string home(){

  }
}
```

3.2 모델

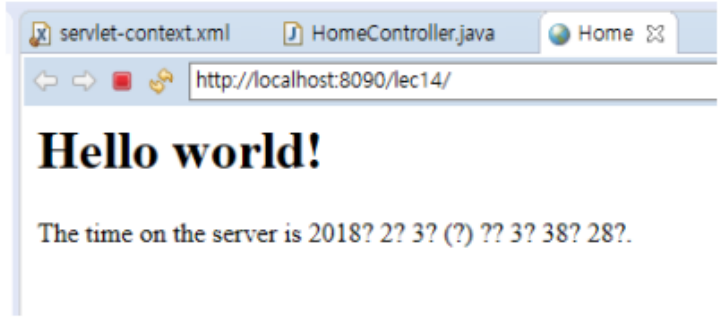
```
@Controlle
public class HomeController {

  @RequestMapping(”/”)
  public string home(Model model){
    model.addAttribute(”today”new Date());
    return “home”;
  }
}
```

3.3 뷰

메서드 반환값 → `return "home";`

JSP 파일 → `/WEB-INF/views/home.jsp`

사용자 응답 브라우저 →



3.3 뷰

```
<beans:bean class="...view.InternalResourceViewResolver">
  <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

```
@Controller
public class LoginController{
    @RequestMapping("/login.do")
     public String login()  {
          return "users/login";
     }
}
```

뷰 : /WEB-INF/views/ users/login .jsp

4.1 컨트롤러 어노테이션

- DispatcherServlet 이 인식하는 Controller 객체로 만들고 컨테이너에 빈 등록
  - @Controller

- URI **요청을 컨트롤러의 특정 메서드와 매핑**
  - @RequestMapping
  - @PostMapping, @GetMapping, @DeleteMapping, @PutMapping

- **요청정보 받기**
  - **커맨드 객체** : "select?id=park&name=dong"
  - @RequestParam :
  - @RequestBody : {"id":"park", "name":"dong"}
  - @RequestPart : **첨부파일**(multipart)
  - @PathVariable : "select/park/dong"
  - @ModelAttribute

4.2 요청정보 받기

질의문자열 → 커맨드객체, @RequestParam

```
<form action ="mypage.do"
        method="post">
   <input name="id" value="hong">
</form>
```

```
location.href ="mypage.do?id=hong"
```

```
@RequestMapping("/mypage.do")
public String login(UserVO vo)  {
```

```
@RequestMapping("/mypage.do")
public String login(@RequestParam String id)  {
```

uri 경로에 변수 → @PathVariable

```
location.href ="mypage.do/hong"
```

```
@RequestMapping("/mypage.do/{id}")
public String login(@PathVariable String id)  {
```

json 문자열 → @RequestBody

```
$.ajax({
     contentType : "json",
     data : json.stringify(  {id : id}  )
})
```

```
@RequestMapping("/mypage.do")
public String login(@RequestBody UserVO vo)  {
```

4.2 요청정보 받기

첨부파일 업로드

```
<form action ="mypage.do"
        method="post"
        enctype="multipart/form-data">
    <input name="id" value="hong">
    <input name= "pic" type="file">
</form>
```

```
class UserVO {
  String id;
 MultipartFile pic;
}
```

```
@RequestMapping("/mypage.do")
public String login(UserVO vo)  {
```

```
class UserVO {
  String id;
}
```

```
@RequestMapping("/mypage.do")
public String login(UserVO vo ,
                    @RequestPart MultipartFile pic)  {
}
```

4.3 CharacterEncodingFilter 등록

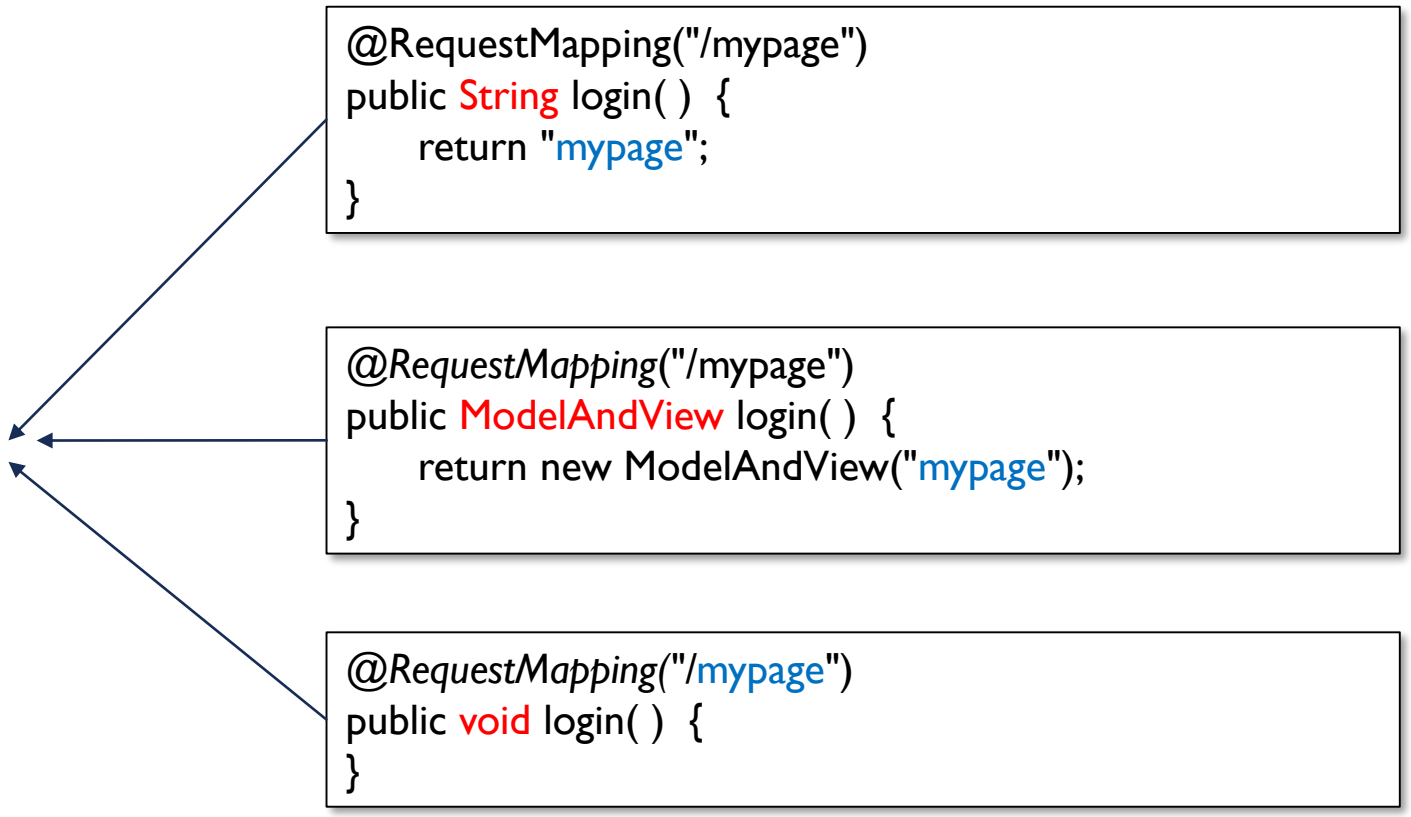
- 파일위치 : /src/main/webapp/WEB-INF/web.xml
- Encoding 파라미터 정보를 읽어 인코딩 방식을 설정
- <url-pattern> 설정의 요청에 대해서 일괄적으로 한글 처리

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
  </init-param>
 </filter>

 <filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
 </filter-mapping>
```

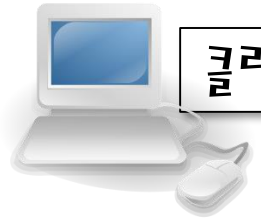
4.5 리턴타입

리턴 타입

1. String
2. ModelAndView
3. void



4.6 응답결과 보내기 – forward, @ModelAttribute



클라이언트 요청: mypage.do?id=hong

컨트롤러

```
@RequestMapping("/mypage.do")
public String login(Model model, UserVO vo) {
        model.addAttribute("profile",  service.getUser(vo.getId()));
        return "mypage";
}
```

뷰 : mypage.jsp

```
<div>
${param.id}
${userVO.id}

${profile.id}
</div>
```

```
@RequestMapping("/mypage.do")
public String login(Model model ,
                    @ModelAttribute("user") UserVO vo) {
```

```
${param.id}
${user.id}
```

4.6 응답결과 보내기 – redirect, RedirectAttributes

- RedirectAttributes : redirect 될 때 데이타가 여러개인 경우에 유용

컨트롤러

```
@PostMapping("/insert.do")
public String insert(BoardVO vo,
                     RedirectAttributes rttr,
                     @RequestParam String page) {
  service.insert(vo);
  rttr.addFlashAttribute("msg","등록완료");
  rttr.addAttribute("page", page);
  return "redirect:list.do";
}
```

redirect

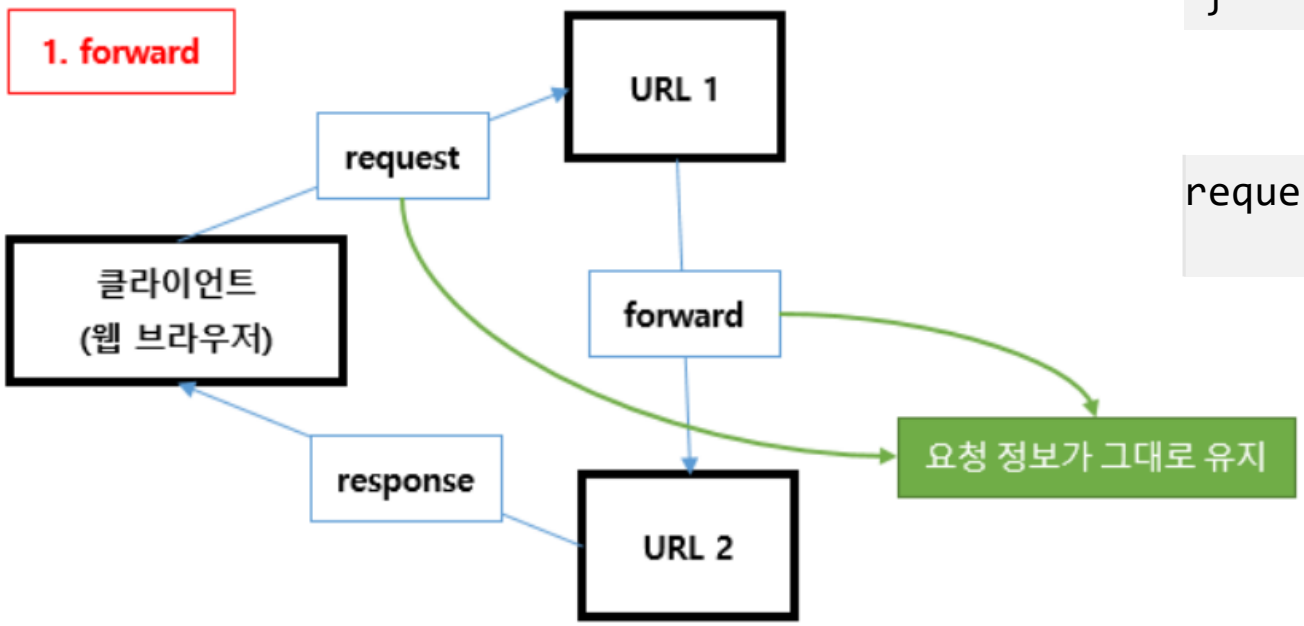
```
@RequestMapping("/list.do")
public String list(Model model,
                   @RequestParam String page) {

   model.addAttribute("boards", service.select(page));

   Map<String, ?> flashMap =
          RequestContextUtils.getInputFlashMap(request);

if(flashMap!=null) {
            System.out.println(flashMap.get("msg"));
   }

   return "list";
}
```

forward

list.jsp

```
<script type="text/javascript">
   var msg = '${msg}';
   if( msg !='') {
        alert("게시물이 등록되었습니다!");
    }
</script>
<body>
  <c:forEach items="${boards}">
```

4.7 페이지이동 - forward

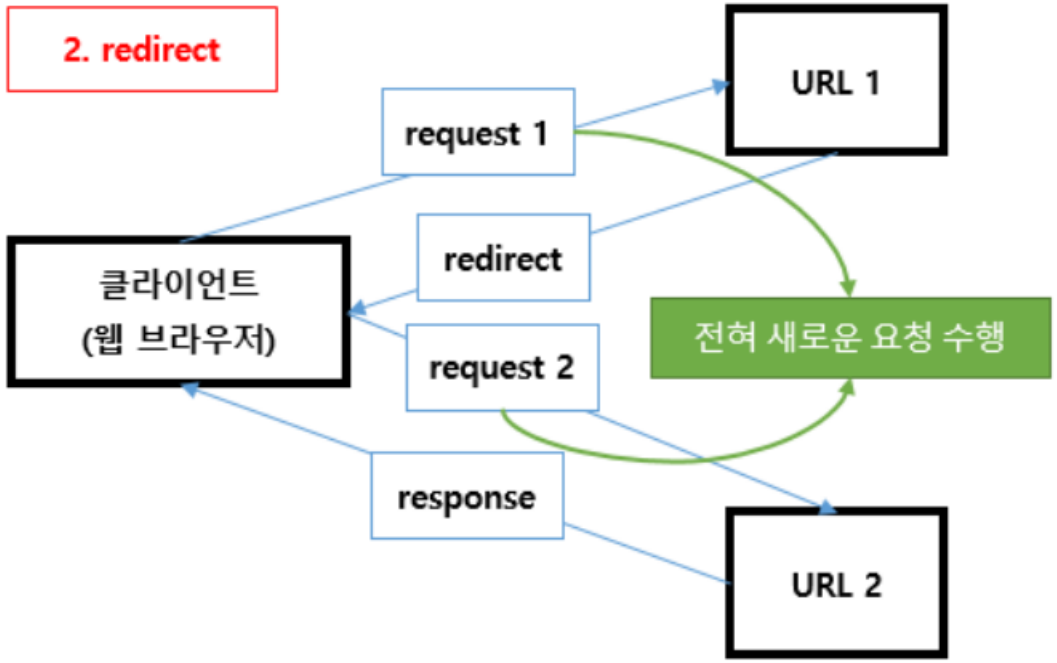



```java
@RequestMapping("/mypage.do")
public String insert() {
    return "list";
}
```

=

```java
request.getRequestDispatcher("/list.jsp")
       .forward(request, response);
```

3.7 페이지이동 - sendRedirect

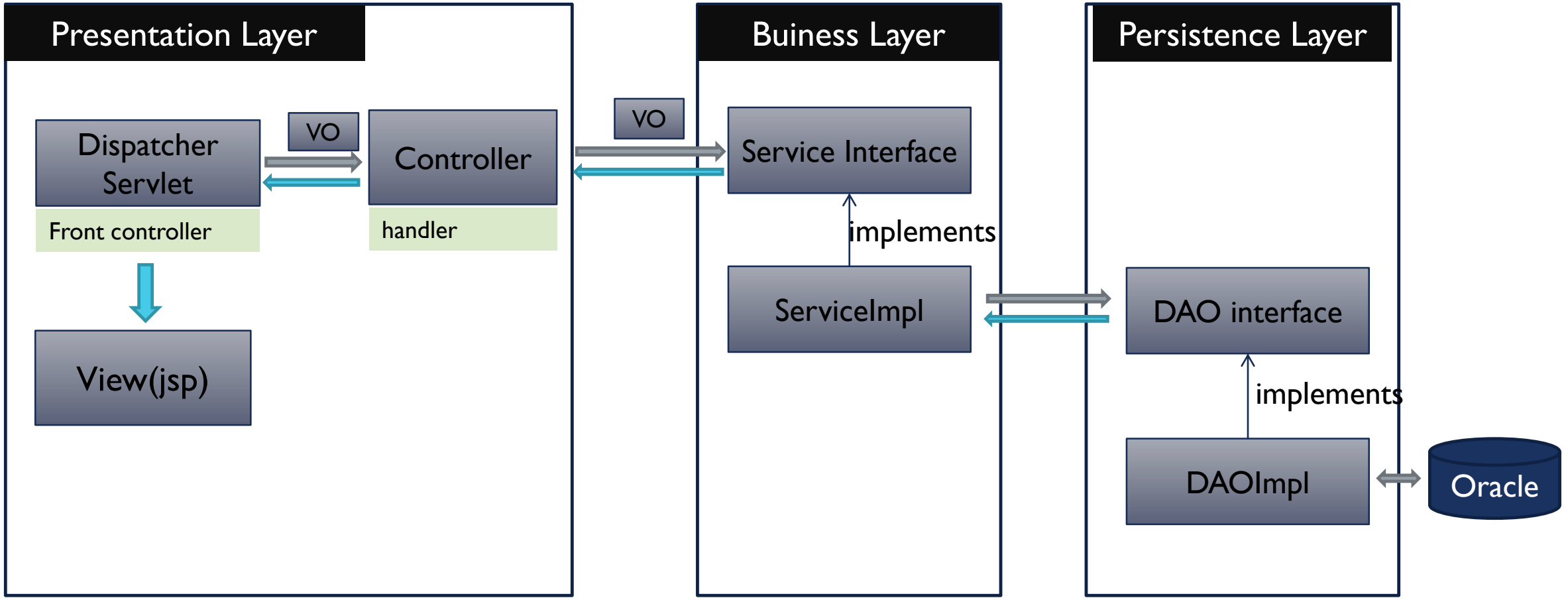



```
@RequestMapping("/mypage.do")
public String insert() {
    return "redirect:list.do";
}
```

=

```
response.sendRedirect("main.jsp");
```

4.8 spring 어플리케이션 아키텍처



4.8 spring 어플리케이션 아키텍처

- 3-layer 구조
  - 프리젠테이션(UI) 계층
    - 웹 계층, UI 계층, MVC 계층
  - 비즈니즈 계층
    - 메니저 계층, 비즈니스 계층
    - Pojo 로 작성되며 DAO 계층을 호출
  - 퍼시스턴스 계층
    - DAO 계층, EIS(Enterprise Information System) 계층
    - 화면흐름결정, 사용자 입력 값에 대한 검증, 서비스계층의 호출과 전달되는 값의 포맷의 변화를 처리

HandlerMapping

- 파일위치 : /WEB-INF/spring/appServlet/servlet-context.xml
- 웹 요청을 실제로 처리하는 객체를 Handler라고 표현
- 특정 요청 경로를 처리해주는 Hanlder를 찾아주는 객체를 HandlerMapping이라고 부릅니다
- HandlerMapping 직접 등록 / 자동으로 스캔해서 등록

```
<!– HandlerMapping 직접 등록 -->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
       <props>prop key="/login.do">login<prop>
        <property>
</bean>

<!– HandlerMapping 스캔하여 자동 등록 →
<annotation-driven />
```