

5. 컨트롤러

1. 스프링 MVC 구조
2. 스프링 MVC의 컨트롤러
3. URL 매핑
4. 파라미터 수집과 변환
5. 페이지 이동, 데이터 전달

1.1 스프링 프레임워크

스프링 프레임워크
코어



서브 프로젝트

스프링 MVC

스프링 배치

스프링 시큐리티

스프링 데이터

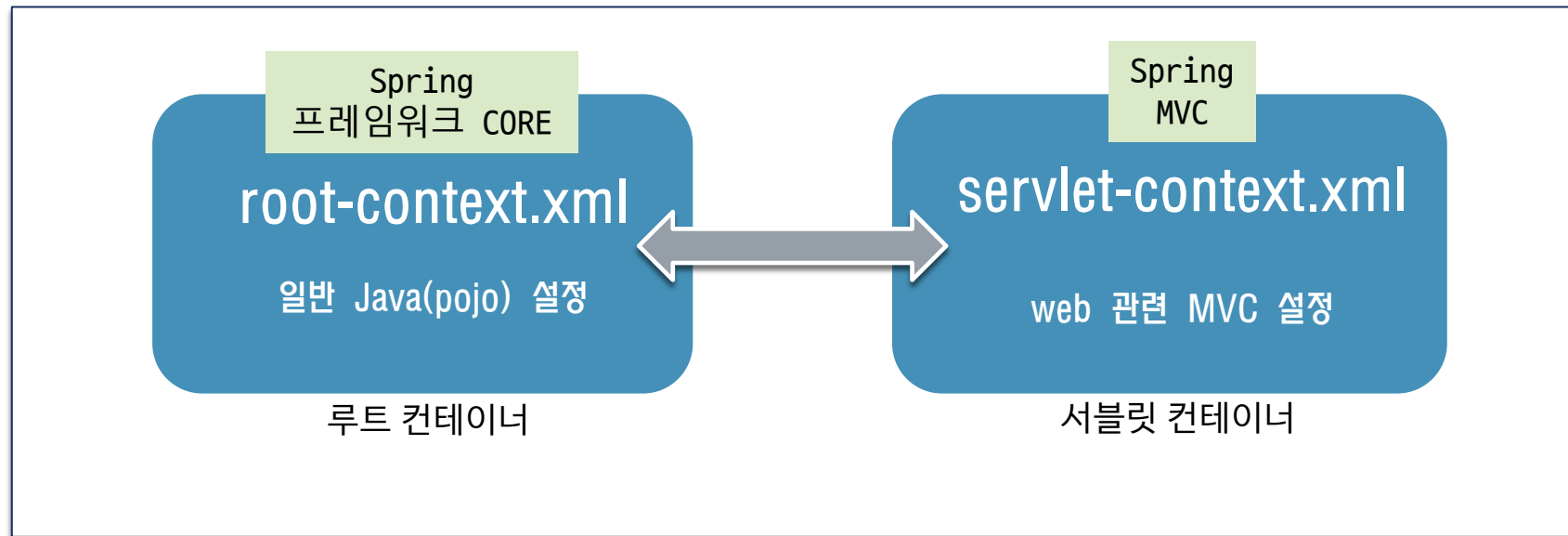
스프링이 웹어플리케이션을 목적으로 나온 프레임워크가
아니기 때문에 완전히 분리하고 연동하는 방식으로 구현됨

1.2 Spring 서브프로젝트(모듈)

프리젠테이션	스프링 MVC	웹프로젝트
	스프링 시큐리티	인증/인가 기능. OAuth
비즈니스 로직	DixAOP 컨테이너	DI: 오브젝트 생성 관리. 소프트웨어의 부품화 및 설계. 인터페이스 기반의 컴포넌트 AOP: 비즈니스 로직 이외의 부가기능(인증, 로깅, 트랜잭션, 예외처리) 등은 소스코드에 명시적으로 기술하지 않고 나중에 추가가능 □
	스프링 캐시	RDB 데이터 캐시해서 RDB의 처리를 줄임으로서 퍼포먼스 향상
데이터 액세스	스프링 JDBC	JDBC 추상화. sql문과 엔티티 클래스의 매핑
	스프링 데이터	JPA
	스프링 ORM 인티그레이션 기능	하이버네이트 등을 간단히 이용
스프링 배치	스프링 BACTCH	대량의 데이터의 일괄처리와 복수 처리, 병행 처리의 실행
스프링 부트	SpringBoot	소프트웨어 개발을 위한 기반 프레임워크. 위의 레이어에 존재하는 스프링 기술이나 그 밖의 라이브러리(Tomcat, H2DB, commoms)를 적절하게 통합한 템플릿을 제공 애자일, 마이크로서비스 아키텍처, 클라우드 등의 키워드와 잘 어울림.

1.3 설정파일 분리

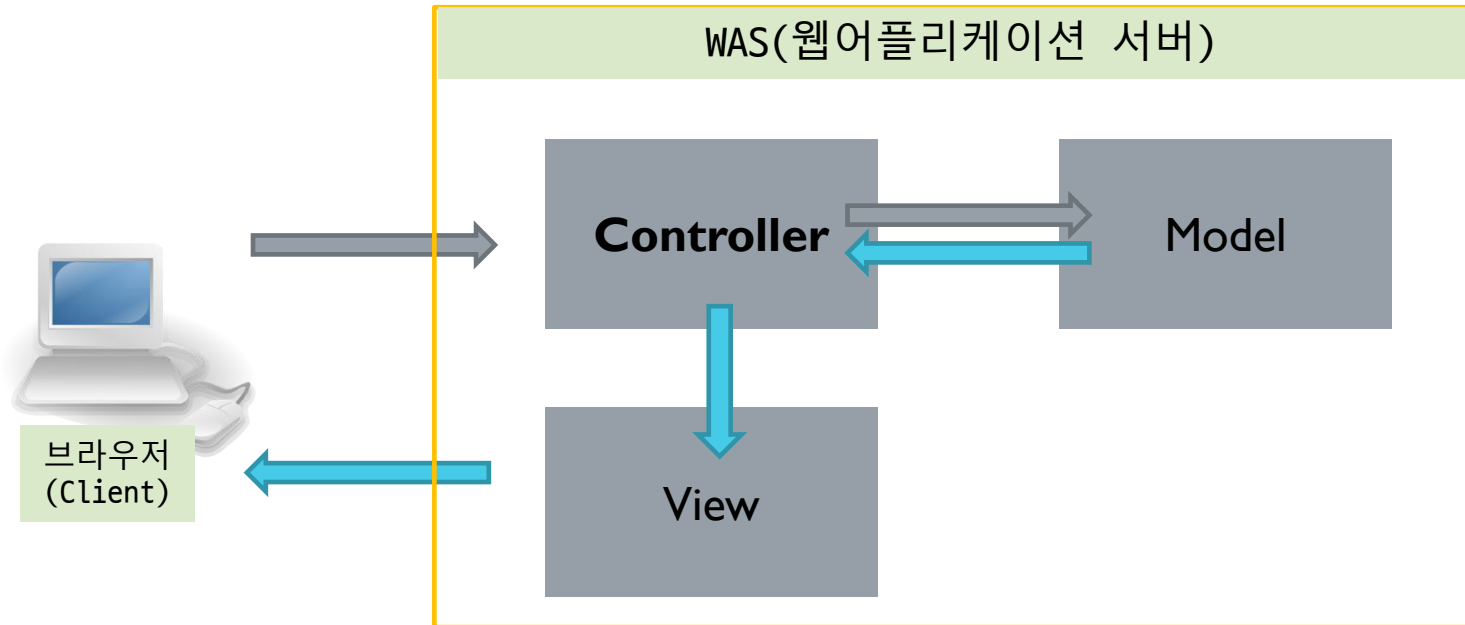
WebApplictionContext



- root-context.xml에 정의된 객체(Bean)들은 컨텍스트 안에 생성되고 객체들 간의 의존성이 처리
 - Spring CORE, MyBatis 설정
- servlet-context.xml에는 스프링 MVC 에서 사용하는 DispatcherServlet 관련 설정이 동작

1.4 모델2와 Spring MVC 구조

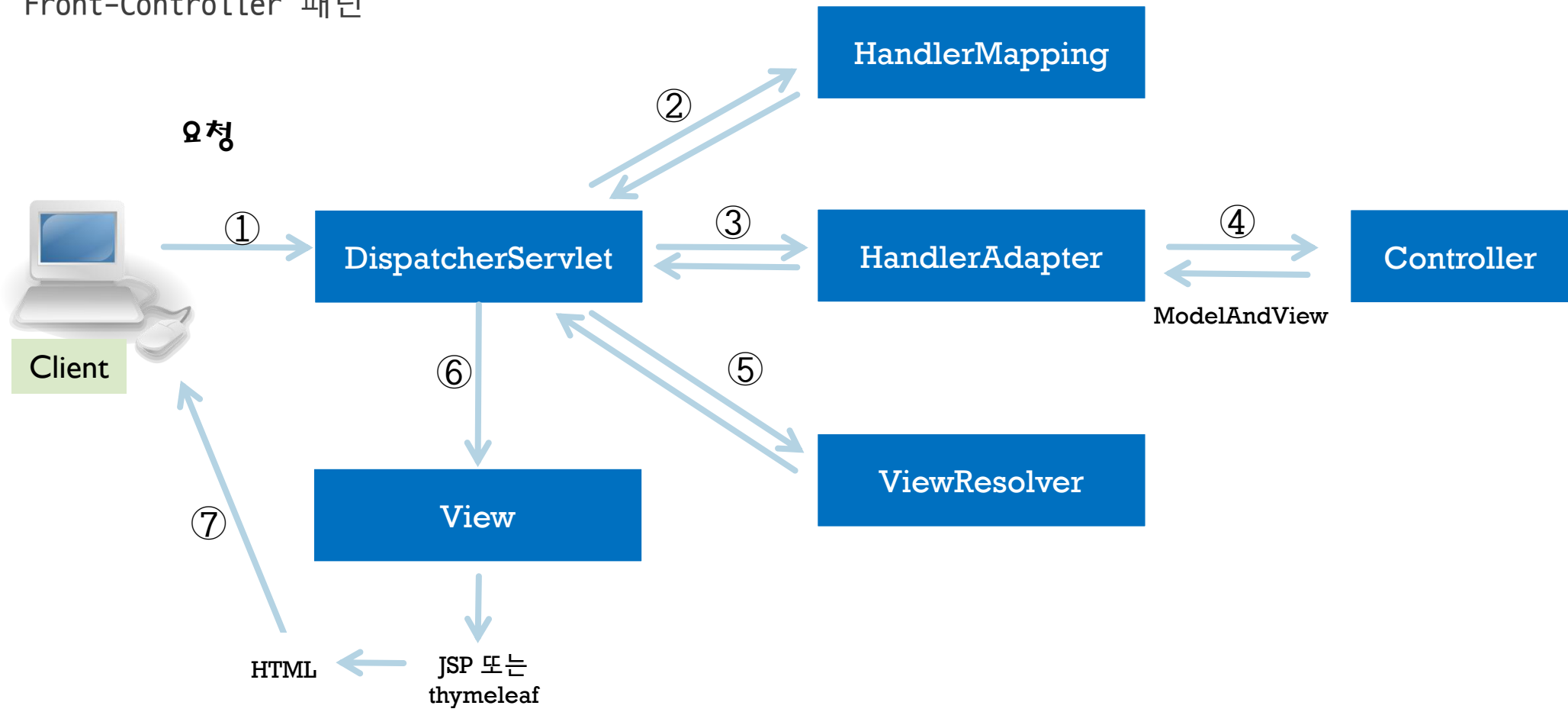
- 모델2 방식
 - 화면과 데이터 처리를 분리해서 재사용이 가능하도록 하는 구조



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

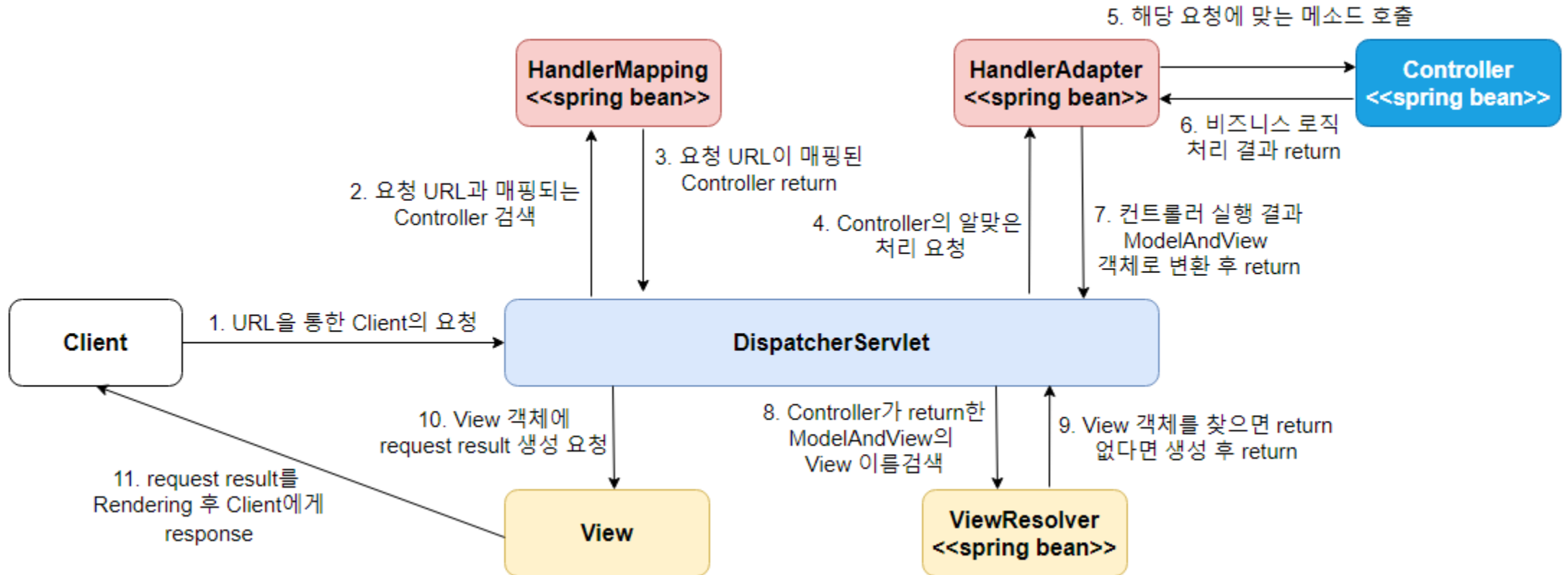
1.4 모델2와 Spring MVC 구조

- 스프링 MVC 구조
 - Front-Controller 패턴



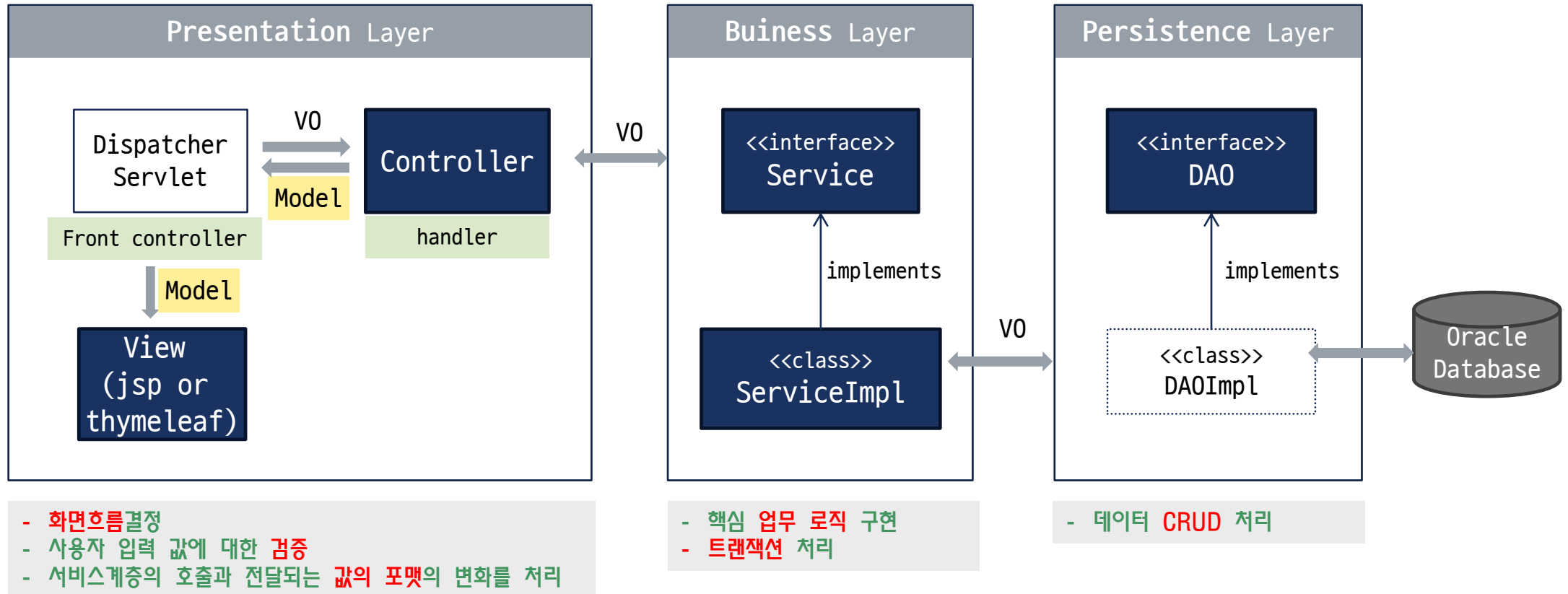
1.4 모델2와 Spring MVC 구조

- 스프링 MVC 구조
 - Front-Controller 패턴














1.5 spring Layer 아키텍처

■ 3 layer 구조



1.5 spring Layer 아키텍처

■ 패키지 구성

- ▼  src/main/java
 - ▼  co.company.mvc.emp.mapper
 - >  EmpMapper.java
 -  EmpMapper.xml
 - ▼  co.company.mvc.emp.service
 - >  EmpService.java
 - >  EmpVO.java
 - ▼  co.company.mvc.emp.service.impl
 - >  EmpServiceImpl.java
 - ▼  co.company.mvc.emp.web
 - >  EmpController.java

Tomcat version 9.0 only supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, and 8 Web modules

1.6 web.xml

■ src/main/webapp/WEB-INF/web.xml

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

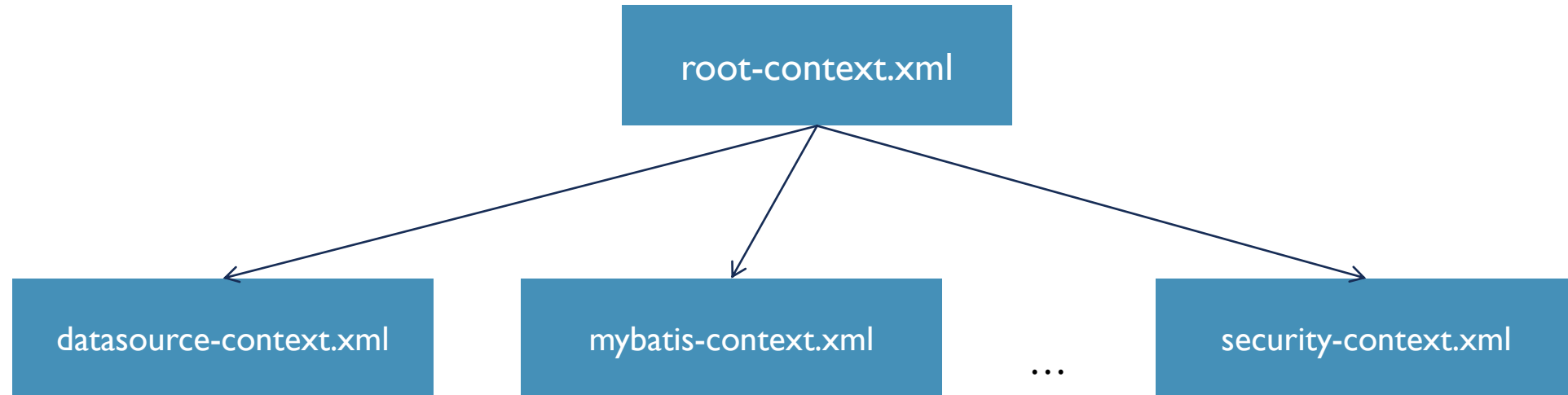
<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- 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>
```

1.6 web.xml

- root-context 설정파일 분리

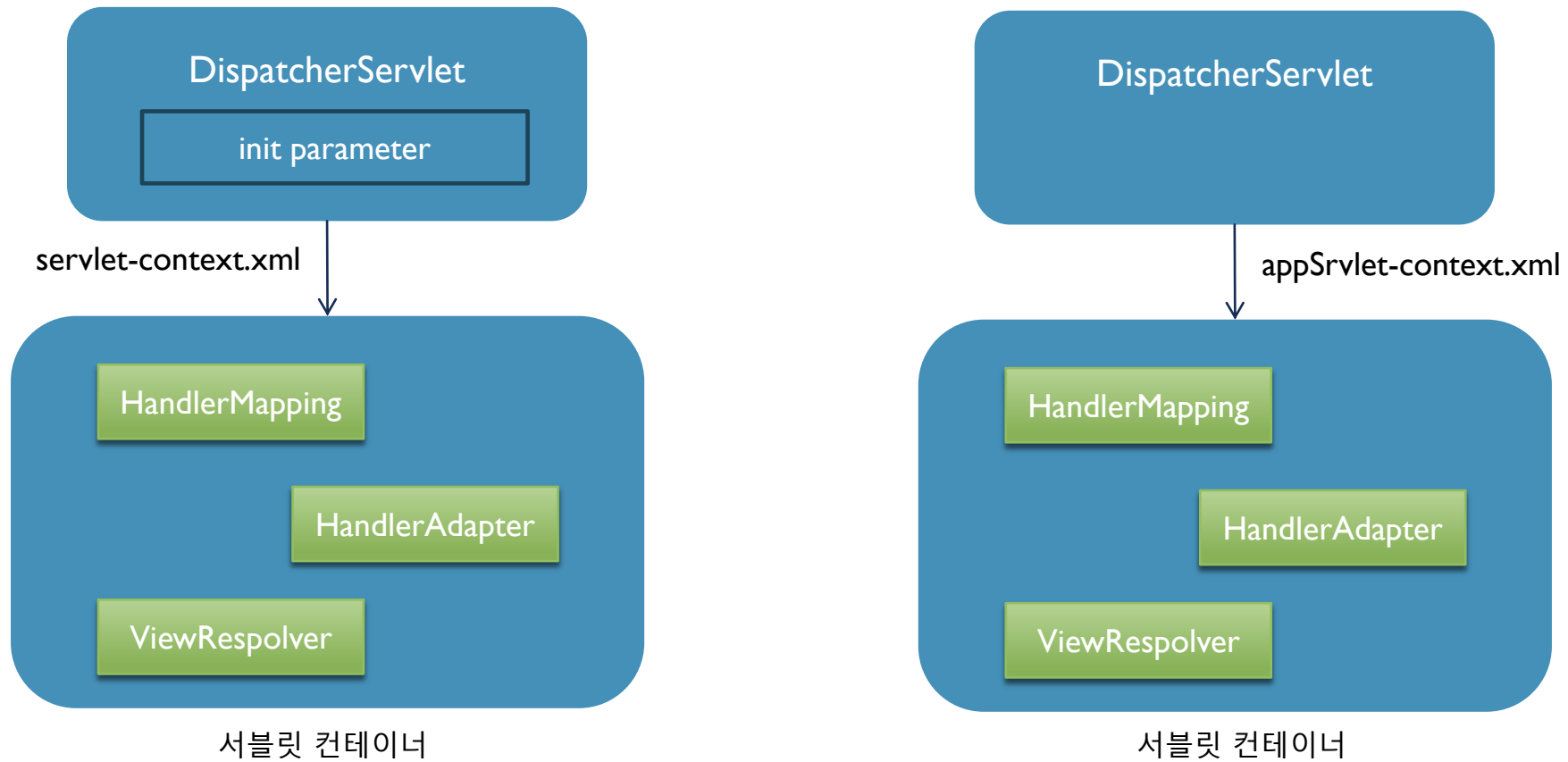


```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/*-context.xml</param-value>
</context-param>
```

web.xml

1.6 web.xml

■ DispatcherServlet 분리



1.6 web.xml

- CharacterEncodingFilter 등록
 - 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>
```

1.7 스프링 MVC 설정 (servlet-context.xml)

`<component-scan/>`

@Component @Controller, @Service, @Repository
스캔해서 컨테이너에 빈으로 등록

`<annotation-driven/>`

@RequestMapping, @GetMapping, @Autowired

`<bean class="ViewResolver">`

view이름으로 사용될 view객체를 매핑.
IntenalResourceViewResolver : jsp페이지

`<resource>`

정적 리소스 경로

1.7 스프링 MVC 설정 (servlet-context.xml)

```
<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />

<!-- Handles HTTP GET requests for /resources/** by efficiently serving
up static resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />

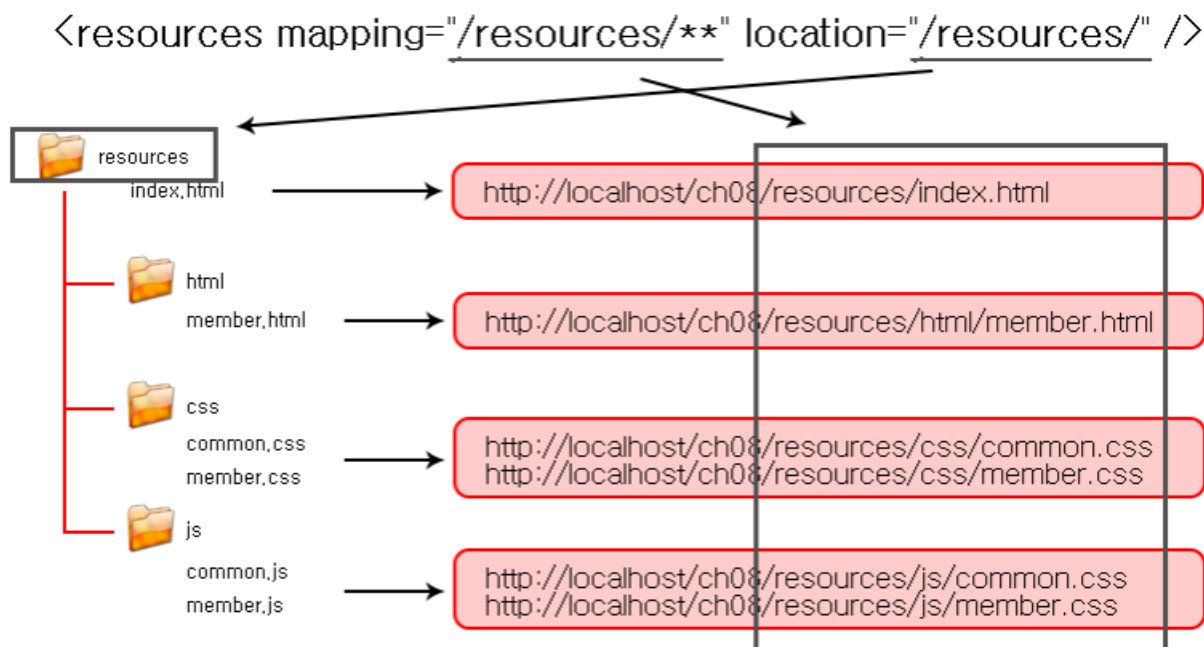
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views
directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="co.company.mvc" >
    <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
</context:component-scan>
```


1.7 스프링 MVC 설정 (servlet-context.xml)

- 정적 리소스 경로 지정

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



2.1 컨트롤러 클래스

■ 컨트롤러 처리과정

1. URI와 핸들러 매핑

2. 파라미터 수집과 변환

3. 서비스 로직

4. 데이터 전달

5. 페이지 이동

```
@Controller
```

```
public class HomeController {
```

```
    @RequestMapping("/")
```

```
    public String home(UserVO vo, Model model){
```

```
        UserVO uservo = service.select();
```

```
        model.addAttribute("data", uservo);
```

```
        return "home";
```

```
    }
```

```
}
```

2.2 컨트롤러 어노테이션

- DispatcherServlet 이 인식하는 Controller 객체로 만들고 컨테이너에 빈 등록
 - @Controller
- URI 요청을 컨트롤러의 특정 메서드와 매핑
 - @RequestMapping
 - @PostMapping, @GetMapping, @DeleteMapping, @PutMapping
- 파라미터 수집과 변환
 - @RequestParam : `"/select?id=park&name=dong"` 질의문자열
 - @PathVariable : `"/select/{park}/{dong}"` URI 형식
 - @RequestPart : 첨부파일 바이너리스트링(multipart)
 - @RequestBody : `'{"id":"park", "name":"dong"}'` JSON 문자열
 - @ModelAttribute

3.1 URL 매핑

- HandlerMapping 객체가 요청 URL에 매핑되는 핸들러(컨트롤러 메소드)를 찾는다

```
public class HomeController {  
    @RequestMapping("/")  
    public String home(){  
        return "home";  
    }  
}
```

@RequestMapping ×

Project: myapp

Resource URL [^]	Request Method	Handler Method
/	GET	● HomeController.home(Locale, Model)
/api/todoDelete	GET	● TodoController.todoDelete(TodoVO)
/api/todoInsert	POST	● TodoController.todoInsert(TodoVO)
/api/todoSelect	GET	● TodoController.todoSelectList(TodoVO)
/api/todoUpdate	GET	● TodoController.todoUpdate(TodoVO)

3.1 @RequestMapping

- @RequestMapping 은 path, method 등을 지정 가능
 - value(=path)는 요청받을 url을 설정.
 - method는 어떤 요청으로 받을지 정의.

▶ 경로만 지정하면 모든 요청방식(method) 허용

```
@RequestMapping("/boardInsert")  
public string insert(){
```

```
@RequestMapping({"/boardInsert", "/insertBoard"})  
public void insert() {
```

▶ 특정 요청방식만 허용

```
@RequestMapping(path="board", method=RequestMethod.POST)  
public void ex01() {
```

```
@RequestMapping(path="board", method={RequestMethod.GET, RequestMethod.POST})  
public void ex01() {
```

3.2 @PostMapping, @GetMapping

- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping 은 경로만 지정하면 됨

▶ 요청 방식 지정

```
@PostMapping("/boardInsert")  
public string insert(){
```

=

```
@RequestMapping(path="board", method=RequestMethod.POST)  
public void insert() {
```

```
@GetMapping("/boardInsert")  
public string insert(){
```

=

```
@RequestMapping(path="board", method=RequestMethod.GET)  
public void insert() {
```

▶ 매핑 경로 여러 개 지정 가능

```
@PostMapping({"boardInsert","/BoardIns"})  
public void insert() {
```

3.3 URL 매핑 예제

- 클래스위에 지정하면 모든 하위 매핑에 적용됨.

```
@RequestMapping("/board")
public class HomeController {

    @PostMapping("/insert")
    public string insert(){
    }

    @PostMapping("/update")
    public string update(){
    }

    @PostMapping("/delete")
    public string delete(){
    }

    @PostMapping("/select")
    public string select(){
    }
}
```

요청 URL:
http://localhost/board/insert

3.4 컨텍스트 루트

▶ ContextRoot path

Tomcat v9.0 Server at localhost ×

Web Modules

Web Modules
Configure the Web Modules on this server.

Path	Document Base	Module
/myapp	myapp	myapp
/	ex0501	ex0501

Overview Modules

```
@RequestMapping("/board")  
public class HomeController {  
  
    @PostMapping("/insert")  
    public string insert(){  
    }  
}
```

요청 URL: <http://localhost/myApp/board/insert>

4.1 요청정보 받기

- 파라미터가 자동으로 수집되고 파라미터 타입에 따라 자동으로 변환됨

```
<form action="userInsert" method="post">
  <input name="name" value="홍길동">
  <input name="age" value="20">
  <button>등록</button>
</form>
```

```
@Controller
public class SampleController {
    @PostMapping("/userInsert")
    public void userInsert(UserVO vo) {
```

```
public class UserVO {
    private String name;
    private Integer age;

    public String getName() {
        return name;
    }
    public void setName(String name){
        this.name = name;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }
}
```

4.2 커맨드 객체

▶ 커맨드 객체

질의문자열

→

커맨드 객체

```
user?name=choi&age=20
```

```
@RequestMapping("/user")
public String process(UserV0 userV0) {
```

```
public UserV0 {
    private String name;
    private Integer age;
}
```

▶ 커맨드 객체(파라미터 인덱스)

```
mypage.do?userlist[0].name=choi&userlist[0].age=20&userlist[1].name=park&userlist[1].age=30
```

인코딩 후

```
mypage.do?userlist%5B0%5D.name=choi&userlist%5B1%5D.name=park
```

```
@RequestMapping("/user")
public String process(UserListV0 userListV0) {
```

```
public UserListV0 {
    private List<UserV0> userlist;
}
```

4.3 @RequestParam

▶ @RequestParam 질의문자열 → 기본데이터형

mypage.do?name=hong&age=20

@RequestMapping("/mypage.do")

public String rprocess(String name, Integer age) {

생략 가능

@RequestMapping("/mypage.do")

public String rprocess(@RequestParam String name,
 @RequestParam Integer age) {

필수

@RequestMapping("/mypage.do")

public String rprocess(@RequestParam Map<String, Object> map) {

Map

▶ @RequestParam : 필수여부, 초기값 지정

@RequestMapping("/mypage.do")

public String rprocess(@RequestParam(defaultValue = "guest" ,
 value = "name",
 required = true) String username,
 Integer age) {

4.3 @RequestParam



@RequestParam

같은 파라미터이름들

```
mypage.do?name=choi&name=park&name=kim
```

→ List<String> 또는 String[]

```
@RequestMapping("/mypage.do")  
public String login(@RequestParam String[] name) {
```

```
@RequestMapping("/mypage.do")  
public String login(@RequestParam List<String> name) {
```

4.4 @RequestParam

▶ 첨부파일 업로드 → @RequestParam MultipartFile

```
<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 ,
    @RequestParam MultipartFile pic) {
}
```



4.5 요청정보 받기 - @PathVariable

▶ @PathVariable

URI 경로에 값이 포함 → 기본타입변수

```
location.href = "users/hong"
```

```
@RequestMapping("/users/{id}")  
public String process(@PathVariable String id) {
```

4.6 요청정보 받기 - @RequestBody

- Ajax에서 JSON 문자열을 받을 때

▶ @RequestBody

json 문자열

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

→

커맨드객체

```
@RequestMapping("/users")  
public String process(@RequestBody UserVO vo)  
{
```

4.6 파라미터 변환

■ @DateTimeFormat

- 입력값 : 2022/05/10

```
import java.util.date;  
class UserVO {  
    String id;  
    private Date wdate;  
}
```

- 입력값 : 2022-05-10

```
import java.util.date;  
class UserVO {  
    String id;  
    @DateTimeFormat(pattern = "yyyy-MM-dd")  
    private Date wdate;  
}
```

```
import java.sql.date;  
class UserVO {  
    String id;  
    private Date wdate;  
}
```


5.1 리턴타입

▶ 리턴 타입

1. String
2. ModelAndView
3. void
4. 객체타입
5. ResponseEntity

```
@RequestMapping("/mypage")
public String login( ) {
    return "mypage";
}
```

```
@RequestMapping("/mypage")
public ModelAndView login( ) {
    return new ModelAndView("mypage");
}
```

```
@RequestMapping ("/mypage")
public void login( ) { }
```

```
@RequestMapping ("/mypage")
@ResponseBody
public UserVO login( ) {
    return vo;
}
```

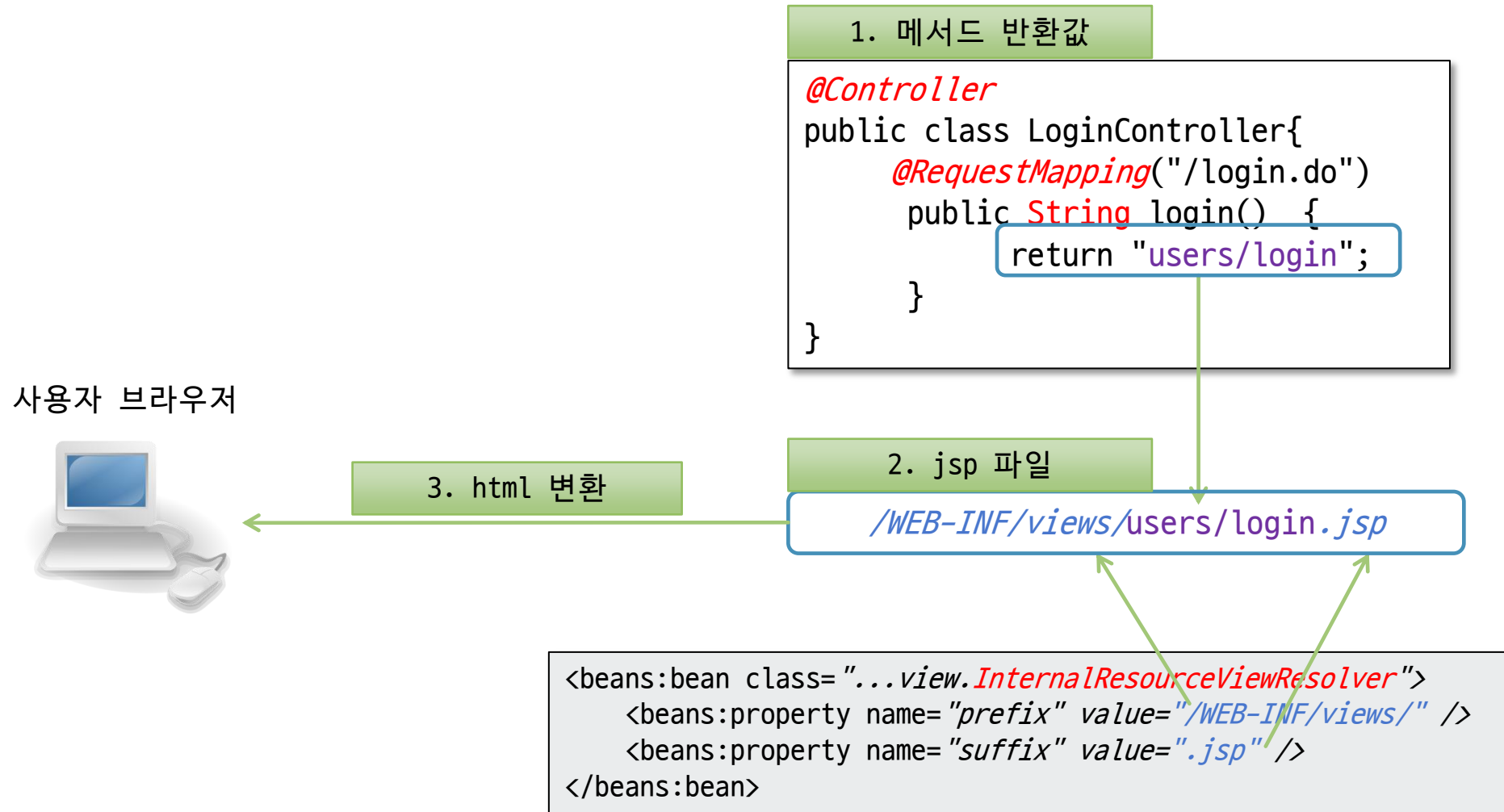
```
@RequestMapping ("/mypage")
public ResponseEntity<> login( ) {
    return new ResponseEntity();
}
```

WEB-INF/views/[mypage.jsp](#)

'{"name":"park", "age":20}'

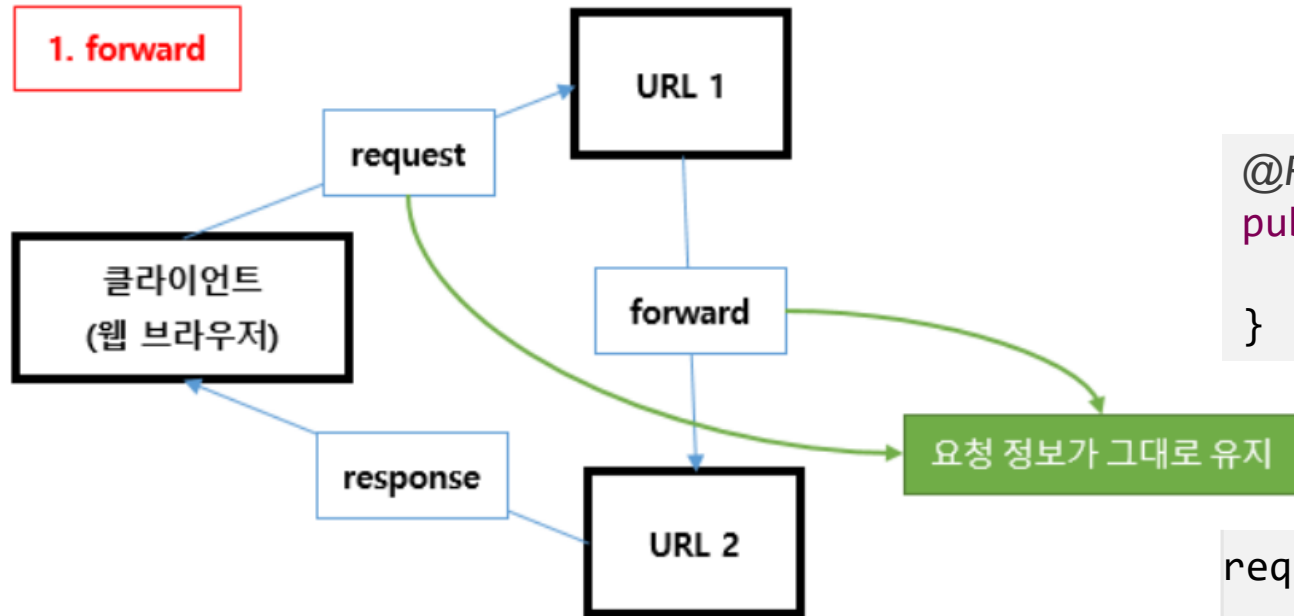
ResponseEntity

5.2 페이지이동



5.3 forward

■ forward

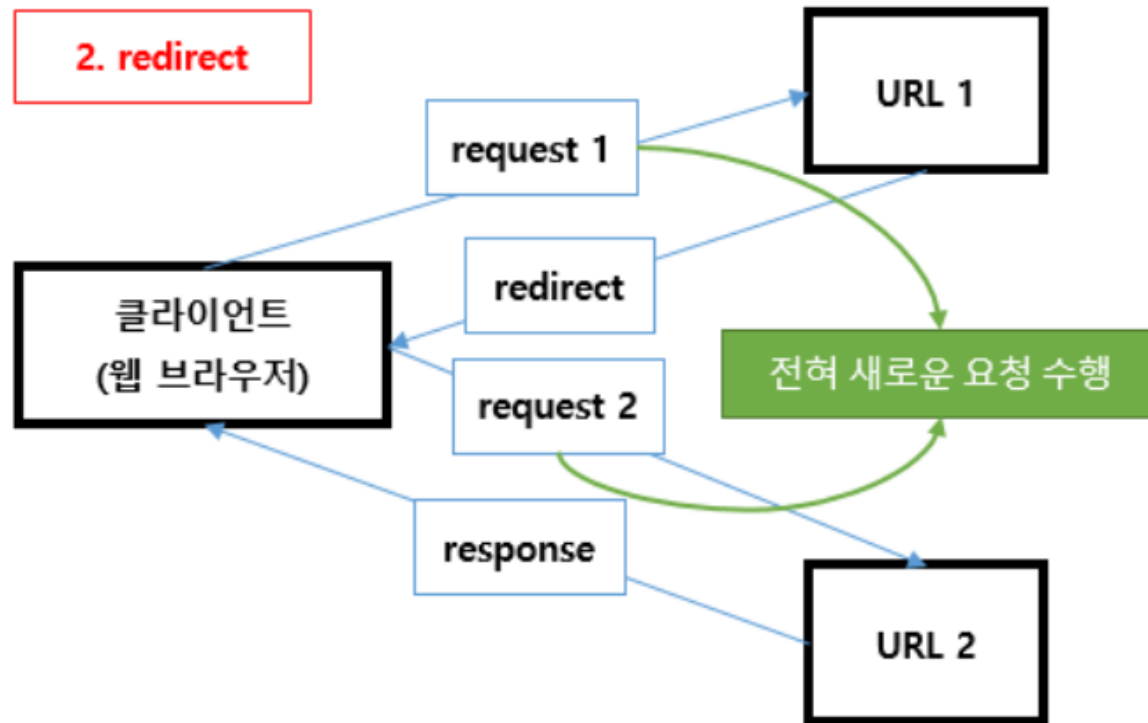


```
@RequestMapping("/mypage.do")
public String insert() {
    return "list";
}
```

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

5.4 sendRedirect

■ sendRedirect



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

=

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

5.5 ResponseEntity

■ response 헤더 지정

```
@PostMapping("/insert")
public ResponseEntity<EmpVO> insert2(EmpVO vo) {
    HttpHeaders header \ new HttpHeaders( );
    Header.add("Content-type", application/json;charset=UTF-8" );
    return new ResponseEntity<>(vo, header, HttpStatus.OK);
}
```

```
new ResponseEntity<>(body, header, status);
```

5.6 데이터 전달 – Model

- forward
- Model에 담으면 view 페이지로 데이터를 전달해 준다.
- request.setAttribute("key", value)와 같다.

컨트롤러

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

뷰 : mypage.jsp

```
<div>
    → ${profile.username}
</div>
```

5.7 데이터 전달 - @ModelAttribute

- 커맨드 객체는 모델에 담겨서 전달이 됨
- 모델의 이름을 변경하고자 할 때 사용



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";
}
```

: mypage.jsp

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

▶ @ModelAttribute

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

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

5.7 데이터 전달 - @ModelAttribute

- 파라미터를 강제로 담고자 할 때 사용

▶ @ModelAttribute



클라이언트 요청: mypage.do?id=hong

컨트롤러

```
@RequestMapping("/mypage.do")
public String login(Model model, @ModelAttribute("id") String id) {
    return "mypage";
}
```

jsp : mypage.jsp

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


5.8 데이터 전달 – RedirectAttributes

▶ RedirectAttributes : redirect 될 때 데이터가 여러 개인 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}">
```

6.1 세션 정보 조회

6.2 컨트롤러 에러 처리