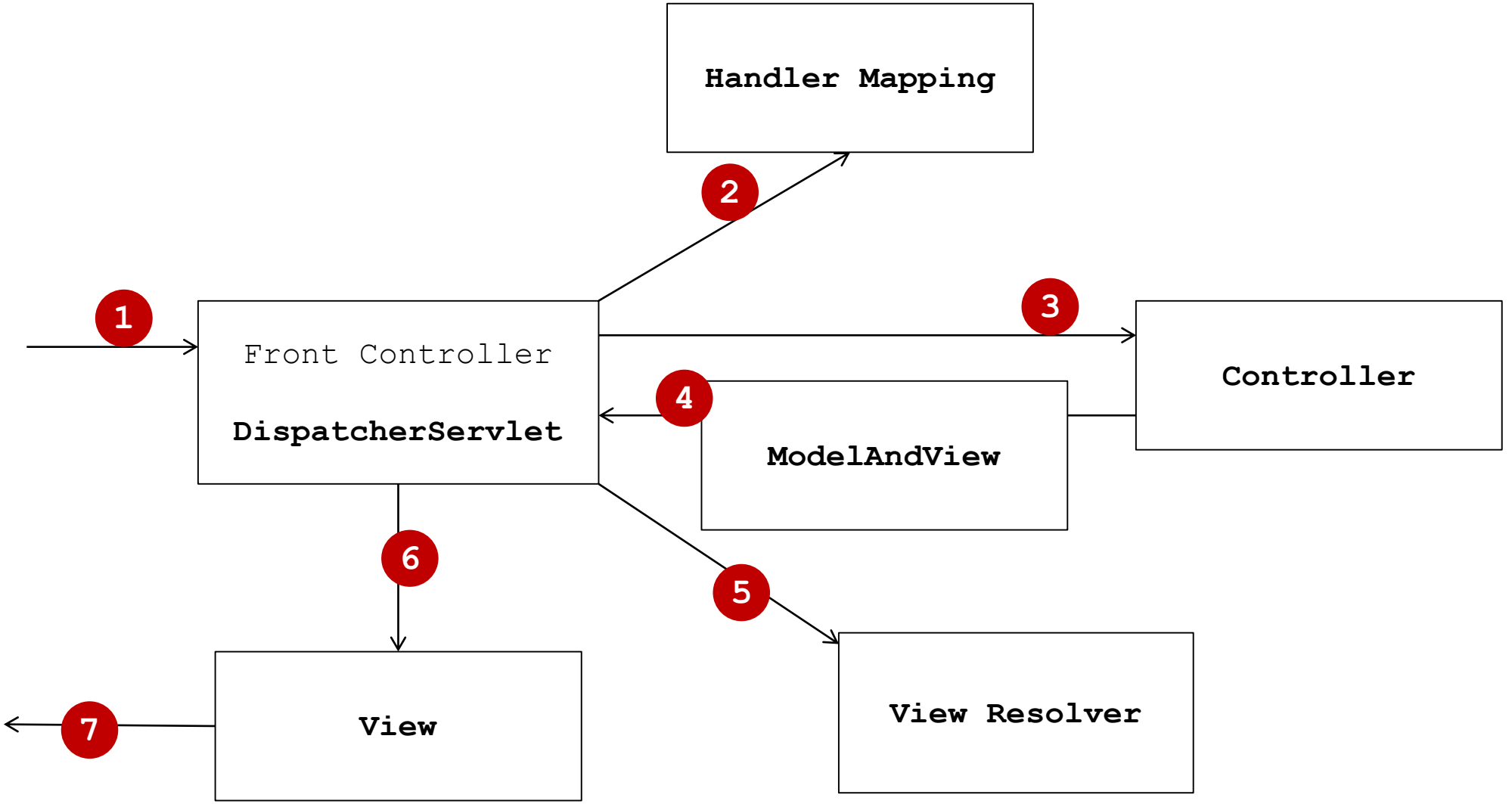
Chapter 02. 자세히 알아보기

01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처


Handler Mapping
2
1
Front Controller
DispatcherServlet
3
Controller
4
ModelAndView
6
5
View
7
View Resolver

1.1 DispatcherServlet 과 MVC

1. 사용자의 요청을 DispatcherServlet이 받는 다.

2. 요청을 처리해야 하는 컨트롤을 찾기 위해 HandlerMapping에게 질의를 하고 HandlerMapping은 컨트롤 객체에 매핑되어 있는 URL를 찾아낸다.

3. DispatcherServlet은 찾은 컨트롤에게 요청을 전달하고 Controller는 서비스 계층의 인터페이스를 호출하여 적절한 비지니스를 수행한다.

4. 컨트롤러는 비지니스 로직의 수행결과로 받아낸 도메인 모델 객체와 함께 뷰이름을 ModelAndView 객체에 저장하여 반환한다.

5. DispatcherServlet은 응답할 View를 찾기 위해 ViewResolver에게 질의를 한다.

6. DispatcherServlet은 찾아낸 View 객체에게 요청을 전달한다.

Chapter 02. 자세히 알아보기

01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처

2.1.1 메소드 단독 매핑

```
public class UserController {

  @RequestMapping( "/hello" )
  public String hello( .... ) { }

  @RequestMapping( "/main" )
  public String main( .. ) { }
}
```

2.1.2 타입 + 메소드 매핑

```
 @RequestMapping( "/user" )
 public class UserController {

    @RequestMapping( "/add" )
    public String add( .... ) { }

    @RequestMapping( "/delete" )
    public String edelete( ... ) { }
}


@RequestMapping( "/user/add" )
    public class UserController {

    @RequestMapping( method = RequestMethod.GET )
    public String form( .... ) { }

    @RequestMapping( method = RequestMethod.POST )
    public String submit( ... ) { }
}
```

2.1.3 타입 단독 매핑

```
@RequestMapping( "/user/*" )
public class UserController {
   @RequestMapping
   public String add( .... ) {

   }

   @RequestMapping
   public String edit( ... ) {

   }
}
```

/user/add, /user/edit 으로 접근

2.2 @RequestParam – 파라미터 매핑

2.2.1 기본 사용법

- http 요청 파라미터를 메소드 파라미터에 넣어주는 어노테이션

```
public String view( @RequestParam("id") int id, @RequestParam("name") String name ) {
   . . .
}
```

- RequestParam를 사용했다면 해당 파라미터가 반드시 있어야 한다. 없으면 HTTP 400 – Bad Request 를 받는다.

- 보통 다음과 같이 정보를 더 추가해서 파라미터를 매핑한다.

```
public String view( @RequestParam( value="id", required=false, defaultValue="-1") int id ) {
   . . .
}
```

2.3 @PathVariable – URL 패스 기반 파라미터 매핑

2.3.1 사용법

- URL에 쿼리 스트링 대신 URL 패스로 풀어 쓰는 방식

예) /board/view?no=10 -> /board/view/10

```
@RequestMapping( "/board/view/{no}" )
public String view( @PathVariable("no") int no ) {

. . .

}
```

2.4.1 사용법

- 요청 파라미터를 객체에 담을 때 사용

```
public class UserVo{
   long no;
   String name;
   String password;

   . . .

   . . .
}


@RequestMappling( value="/user/join", method=RequestMethod.POST )
public String join( @ModelAttribute UserVo userVo ) {

    userService.join( userVo );

    ....

}
```

2.5.1 다양한 파라미터

- HTTPServletRequest, HttpServletResponse
- HttpSession
- Writer

2.5.2 Model 타입 파라미터

- 모델정보를 담을 수 있는 오브젝트가 전달.

```
public String hello( ModelMap model ) {

  User user = new User( 1, "Spring" );
  model.addAttribute( "user", user );
  . . .
  . . .
}
```

Chapter 02. 자세히 알아보기

01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처

3.1 웹 애플리케이션 컨텍스트 (Web Application Context)

1. web.xml 서블릿 매핑 설정의 <servlet-name>에 '-servlet.xml' 를 붙힌 이름의 파일을 WEB-INF에서 찾아 컨테이너에 Bean을 생성하고 초기화 한다.

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class>
</servlet>

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

3.1 웹 애플리케이션 컨텍스트 (Web Application Context)

2. <servlet-name> '-servlet.xml' 설정파일

```
<context:annotation-config />

<context:component-scan base-package="com.example.controller"/>
```

- Controller 빈을 등록하고 빈의 이름(URL)로 핸들러가 매핑된다.
- @MVC 기반에서 빈의 생성은 어노테이션 기반의 컴포넌트 스캐닝을 통해 생성되고 메서드가 핸들러 매핑과 어댑터의 대상이 된다.

3.1 웹 애플리케이션 컨텍스트 (Web Application Context)

2. <servlet-name> '-servlet.xml' 설정파일

```
<bean  id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
    <property name="mappings">
        <props>
            <prop key="/member">memberController</prop>
        </props>
    </property>
</bean>

<bean id=" memberController" class= "com.example.controller.MemberController">
```

- 핸들러 어댑터의 대상이 객체이고 객체의 handleRequest(HttpServletRequest req, HttpServletResponse resp) 메소드 하나만이 url대상이 된다.

3.2 루트 애플리케이션 컨텍스트 (Root Application Context)

1. 리스너를 등록해 두면, 루트 컨텍스트가 생성되게 되며, 설정 파일은 디폴트로 /WEB-INF/applicationContext.xml 이다.

```
<listener>
   <listener-class> org.springframework.web.context.ContextLoaderListener </listener-class>
</listener>

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

---

2. 서비스 계층, 데이터 액세스 계층을 포함해서 웹 환경과 직접 관련이 없는 모든 빈은 여기에 등록한다.

3.2 루트 애플리케이션 컨텍스트 (Root Application Context)

3. applicationContext.xml 예

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd
        http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd
        http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd
        http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task.xsd">

            <context:annotation-config />

            <context:component-scan base-package="com.example.springex">
                        <context:include-filter type="annotation"
                                    expression="org.springframework.stereotype.Repository" />
                        <context:include-filter type="annotation"
                                    expression="org.springframework.stereotype.Service" />
                        <context:include-filter type="annotation"
                                    expression="org.springframework.stereotype.Component" />

            </context:component-scan>


</beans>
```

[실습] emaillist3

emaillist 웹 애플리케이션을 Spring @MVC를 적용해 봅니다.

한글처리를 위해서 다음 필터 설정을 사용합니다.

```
<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>

   <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
   </init-param>
</filter>

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

[실습] emaillist3

emaillist 웹 애플리케이션을 Spring @MVC를 적용해 봅니다.

컨트롤에서 redirect 응답을 위해서는 다음 코드를 참고 합니다.

```
@RequestMappling( "/user/join", method=RequestMethod.POST )

public String join( @ModelAttribute UserVo userVo ) {
    userService.join( userVo );

    return "redirect:/user/joinsuccess";
}
```

Maven POM에 다음 mysql jdbc driver dependency를 추가 합니다.

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
</dependency>
```

[실습] emaillist3

Maven POM에 다음 jstl dependency를 추가 합니다.

```
<!-- jstl -->
<dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>jstl</artifactId>
          <version>1.2</version>
</dependency>
```

Chapter 02. 자세히 알아보기

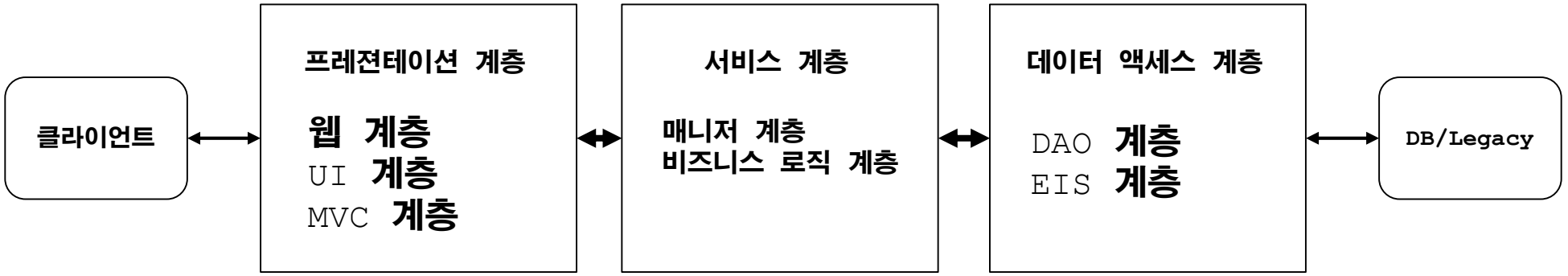
01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처

4.1 애플리케이션 아키텍처



스프링에서는...

1. 3계층은 스프링을 사용하는 엔터프라이즈 애플리케이션에서 가장 많이 사용되는 구조

2. 스프링 주요 모듈과 기술을 보면 3계층 구조에 맞게 설계

3. 논리적 개념이므로 언제든지 상황과 조건에 따라 달라 질 수 있다.

4.2 애플리케이션 아키텍처 예제

[예제] mysite

1. 비즈니스 분석 (사용자 스토리 도출)

(1) 사용자는 회원가입을 한다.
(2) 사용자는 로그인을 한다.
(3) 사용자는 로그아웃을 한다.

(4) 사용자는 방명록에 글을 남긴다.
(5) 사용자는 방명록 자신의 글을 삭제한다.

(6) 로그인한 사용자는 자신의 정보를 수정한다.
(7) 로그인한 사용자는 게시판에 글을 작성한다.
(8) 로그인한 사용자는 자신의 글을 수정한다.
(9) 로그인한 사용자는 자신의 글을 삭제한다.
(10) 로그인한 사용자는 다른 사람의 글에 댓글을 남긴다.
(11) 사용자는 게시판글의 목록을 볼 수 있다.
(12) 사용자는 게시판글을 읽을 수 있다.

4.2 애플리케이션 아키텍처 예제

[예제] mysite

1. 서비스 정의

UserService
- (1) 사용자는 회원가입을 한다. (join)
- (2) 사용자는 로그인을 한다. (login)
- (3) 사용자는 로그아웃을 한다. (logout)
- (6) 로그인한 사용자는 자신의 정보를 수정한다. (modifyInformation)

BoardService
- (7) 로그인한 사용자는 게시판에 글을 작성한다. (write)
- (8) 로그인한 사용자는 자신의 글을 수정한다. (modify)
- (9) 로그인한 사용자는 자신의 글을 삭제한다. (remove)
- (10) 로그인한 사용자는 다른 사람의 글에 댓글을 남긴다. (write)
- (11) 사용자는 게시판글의 목록을 볼 수 있다. (list)
- (12) 사용자는 게시판글을 읽을 수 있다. (view)

GuestbookService
- (4) 사용자는 방명록에 글을 남긴다. (write)
- (5) 사용자는 방명록 자신의 글을 삭제한다. (remove)

[예제] mysite

1. 서비스 인터페이스 정의 (애플리케이션 기능 목록)

```
Interface UserService {
   void join();
   void login();
   void logout();
   void modyfyInfo();
}
```

```
Interface BoardService {
   void write();
   void remove();
   void modify();
   void list();
   void view
}
```

```
Interface GuestbookService {
   void write();
   void remove();
}
```

4.2 애플리케이션 아키텍처 예제