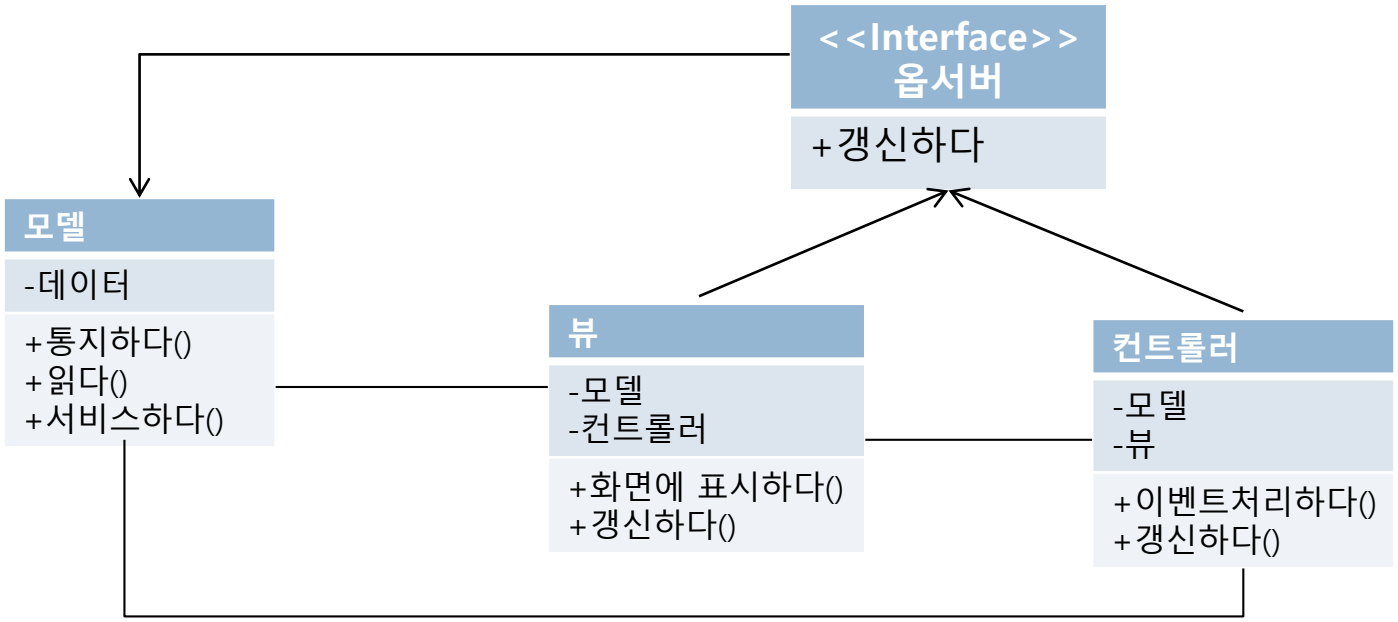
# 8. Spring mvc

1. Spring MVC 개요
2. 스프링 MVC의 구조 살펴보기
3. Resources 폴더
4. 컨트롤러 구현

# 1. Spring MVC 개요

- **MVC 패턴**
  - 애플리케이션의 역할을 모델(Model), 뷰(View), 컨트롤러(controller)로 나누어 작업
  - Smalltalk-80 처음 소개
  - 업무 서비스와 도메인 객체를 사용자 인터페이스로부터 분리, 하나 이상의 컨트롤러를 통해서 이들 사이 상호작용을 통제하는 아키텍처 패턴



# MVC 모델의 구성 요소

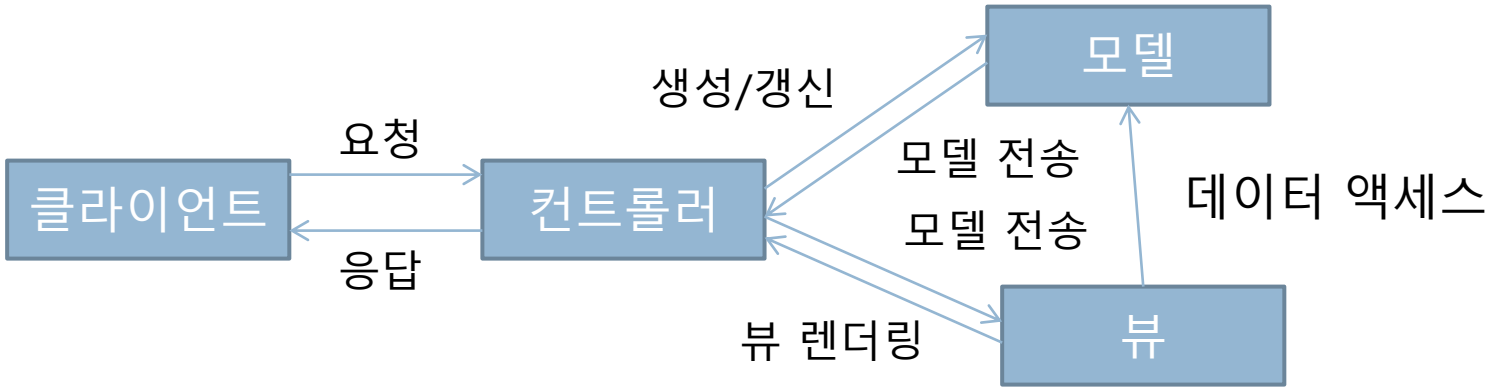
- **모델**
  - 애플리케이션의 핵심적인 기능 제공
  - 뷰와 컨트롤러 등록
  - 데이터 변경 시 뷰와 컨트롤러에 통지
- **뷰**
  - 관련된 컨트롤러 생성, 초기화, 사용자에게 정보 표시
  - 데이터 변경이 통지될 때 모델로부터 데이터를 읽어 처리
- **컨트롤러**
  - 사용자 입력을 이벤트로 받아들여 해석하여 모델에 대한 요청을 서비스하거나 뷰에 대한 요청을 표시
  - 데이터 변경이 통지될 때 모델로부터 데이터 읽어와 처리

# 웹 애플리케이션 MVC

- Front Controller 패턴 적용 – 컨트롤러가 중심이 되는 변형된 구조를 가짐

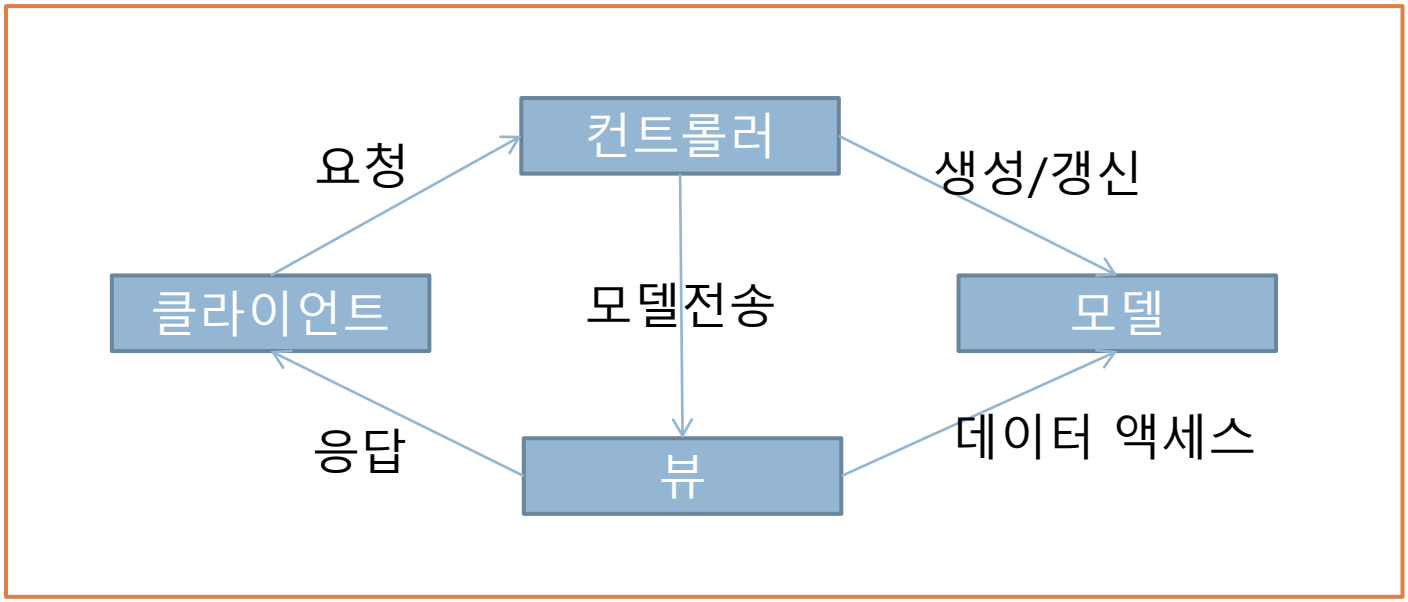
## 동작

- 클라이언트가 컨트롤러에게 데이터 조회, 저장 등의 작업 요청
- 컨트롤러 : 서블릿으로 구현, 업무 로직 처리, 모델생성, 모델의 데이터 갱신, 뷰에 모델 전송, 뷰가 변경된 데이터 사용 가능하게 함
- 모델 : POJO Java 클래스로 구현, 데이터와 데이터처리에 필요한 메서드 포함, 자신이 관리하는 데이터에 집중
- 뷰 : JSP로 구현, 컨트롤러가 제공한 모델을 사용하여 데이터에 액세스하여 웹페이지(뷰)를 렌더링하여 응답을 생성하고 컨트롤러에 전달
- 컨트롤러는 클라이언트에 응답을 전송하고 서비스 종료



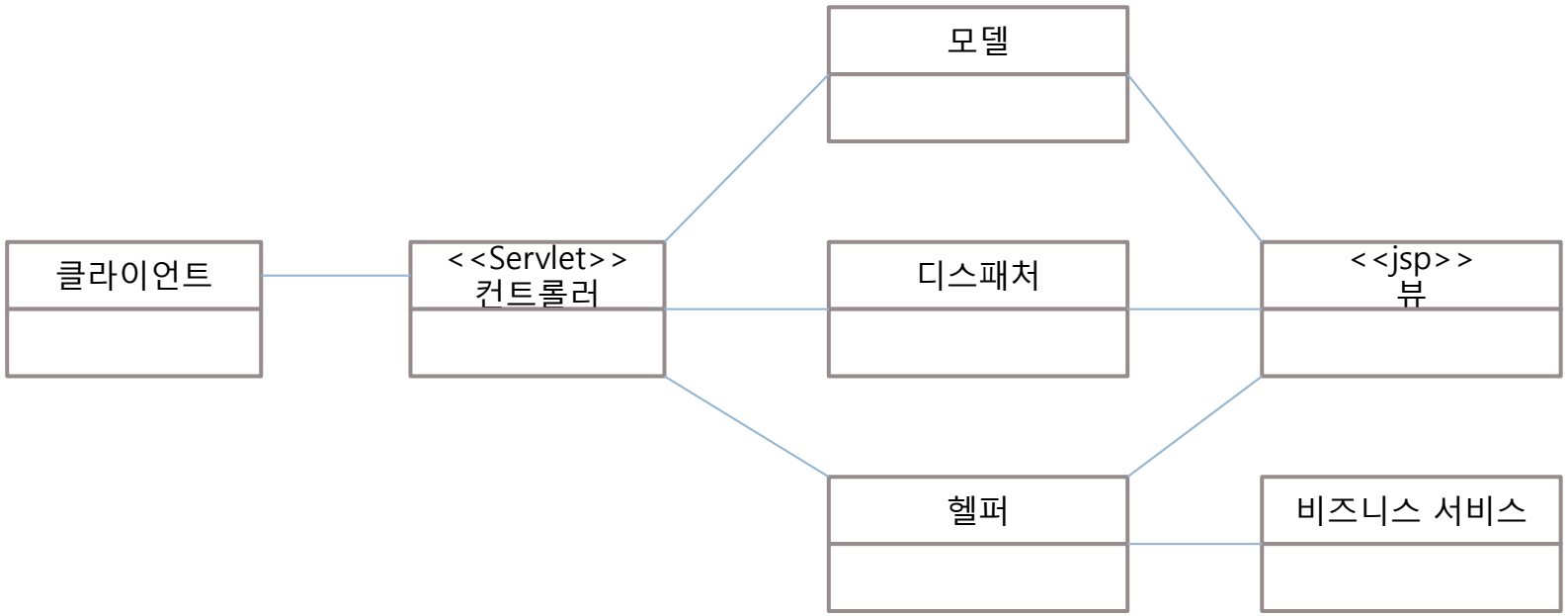
# 웹 애플리케이션 MVC

- 뷰가 클라이언트에 응답하는 구조



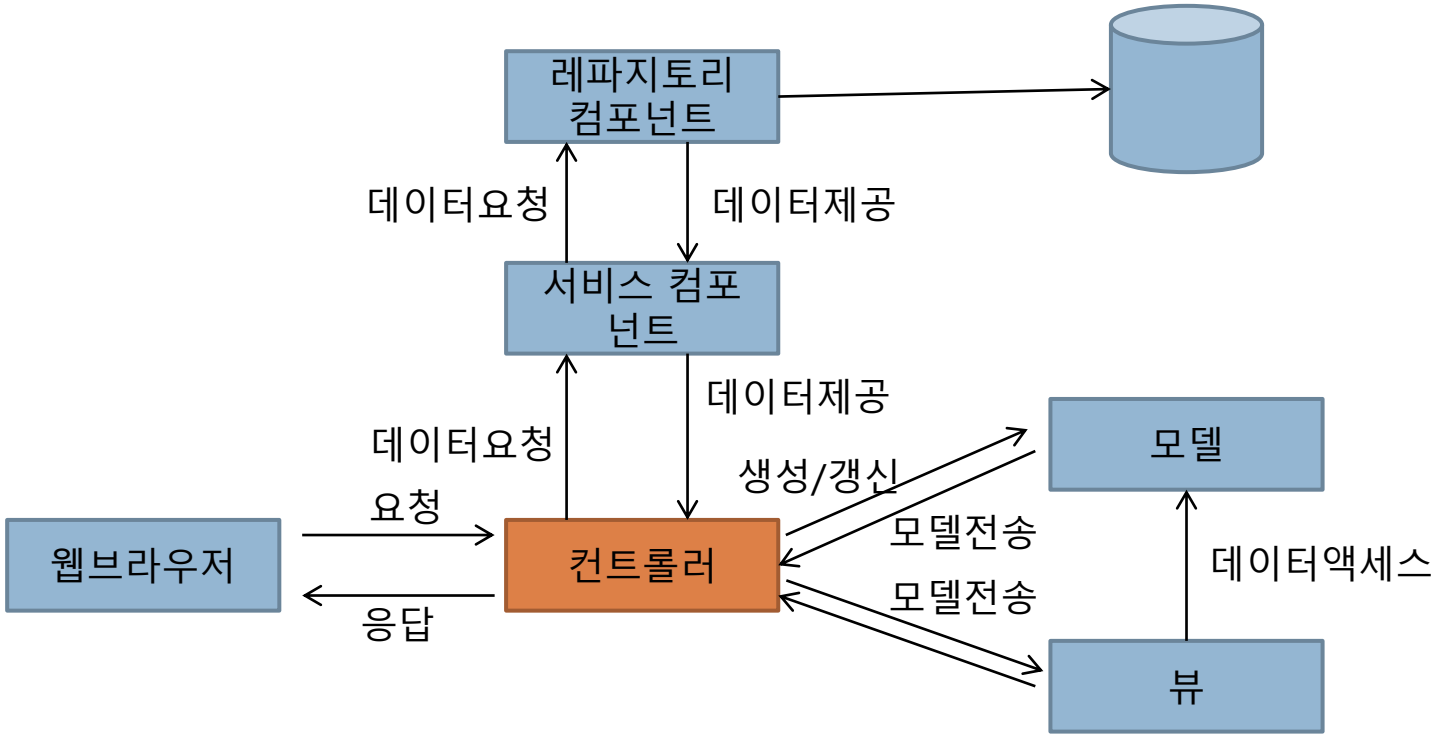
# MVC + Service to Worker 패턴의 클래스 다이어그램

- 모델과 비즈니스 서비스 클래스 추가
- 컨트롤러- Front Controller 패턴과 마찬가지로 요청 처리하는 시작 접속점으로서 인증과 권한 같은 보안 서비스를 호출, 업무처리 위임, 적절한 뷰 선택 관리 에러처리, 콘텐츠 생성전략 선택 관리 등 요청처리 관리
- 디스패처- 뷰의 관리와 이동에 대한 책임, 사용자에게 다음에 보여줄 뷰 선택 관리, 이 리소스로 제어를 인도하는 매커니즘 제공
- 헬퍼 – 뷰나 컨터롤러가 작업 처리를 완료할 수 있도록 도와 줌



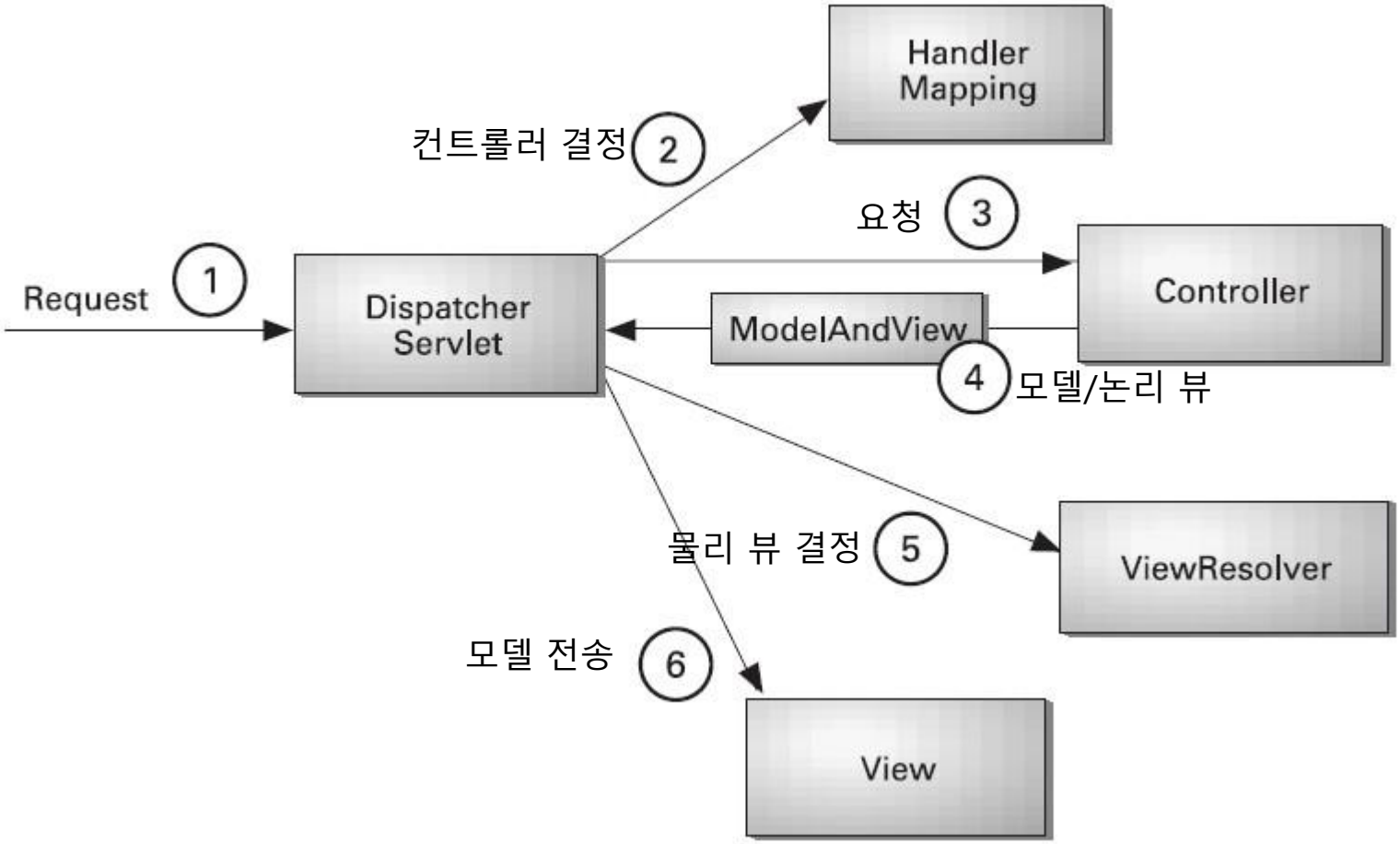
# 웹 애플리케이션 MVC

- **MVC 패턴 + Service to worker + Data Access Object 패턴 추가**
  - 비즈니스 서비스로부터 데이터 액세스 로직을 담당하는 DAO(레파지토리) 서비스를 분리



# Spring MVC 아키텍처 개요

- **Spring MVC 매카니즘**



# Spring MVC 시작

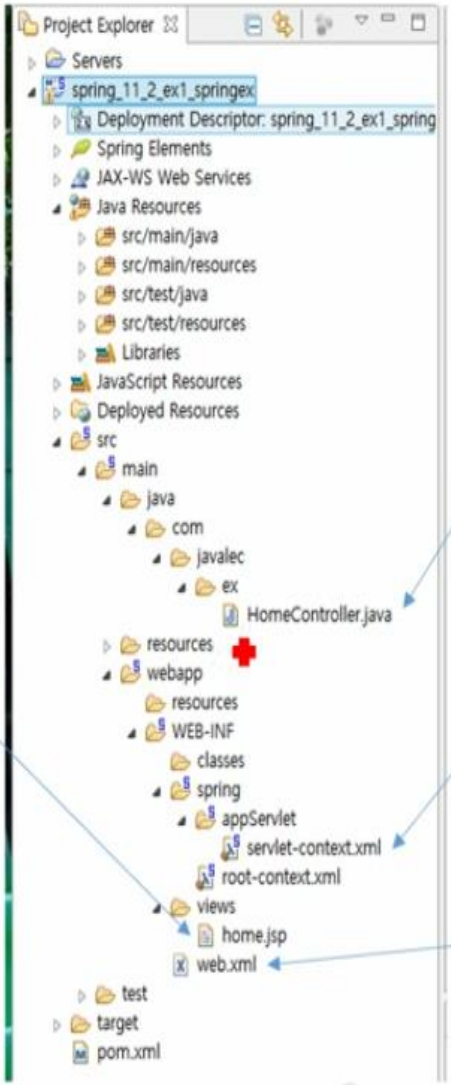
## 스프링 프로젝트 작성 두 가지 방법

- Spring Boot를 이용하는 방법
  - File -> new -> Spring Start Project
- Spring 템플릿 프로젝트를 이용하는 방법
  - File>New>Spring Project
  - New Spring Project 위저드의 Spring Project 화면에서 Spring MVC Project 항목을 선택, 프로젝트 이름: SpringProject01로 지정하고 next 선택
  - Project Setting>Spring MVC Project화면에서 패키지명을 com.miya.project01로 지정, finish 선택

## 디렉토리 구조 살펴보기

- src/main/java - controller
- resources - 자원
- webapp - view

# Spring MVC 구조




뷰(.jsp)

컨트롤러

Dispatcher에서 전달된 요청을 처리

servlet-context.xml

스프링 컨테이너 설정 파일

web.xml

1) DispatcherServlet 서블릿 맵핑
2) 스프링 설정 파일 위치 정의

DispatcherServlet

1) 클라이언트의 요청을 최초 받아
2) 컨트롤러에게 전달

# Spring MVC 애플리케이션 설정

- **web.xml 살펴보기**
  - Servlet 지정

```
<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 매핑

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

# 프론트 컨트롤러 DispatcherServlet

- DispatcherServlet
  - HttpServlet 클래스에서 파생된 서블릿
  - 매핑된 URL의 모든 HTTP 요청을 처리
  - 각 DispatcherServlet은 자기 자신의 웹 애플리케이션용 IoC 컨테이너인 WebApplicationContext가 생성
  - WebApplicationContext는 Spring MVC 웹 어플리케이션에 필요한 Spring 빈의 인스턴를 생성. 관리
  - 생성된 Spring 빈은 웹 애플리케이션에서 정의한 컨트롤러, HTTP 요청을 컨트롤러와 매핑시켜주는 HandlerMapping과 물리적인 뷰를 결정하는 ViewResolver도 포함.

## servlet-context.xml

- DispatchServlet의 컨텍스트 파일

- 어노테이션 지원을 위한 태그 설정

```
<annotation-driven />
```

  - HandlerMapping 대한 설정-HTTP 요청을 처리할 컨트롤러 선택
  - RequestMappingHandlerMapping 클래스가 HandlerMapping을 처리하는 Spring 빈으로 등록
  - @RequestMapping 어노테이션 사용하여 @Controller 어노테이션이 지정된 컨트롤러의 메서드가 HTTP 요청을 처리할 수 있도록 매핑

# servlet-context.xml-2

- 물리적 뷰 결정-ViewResolver Spring 빈 설정

```
<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>
```

- 정적인 리소스에 대한 설정

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

```
<resources mapping="/resources/**" location="/resources/" />
<resources mapping="/css/**" location="/css/" />
<resources mapping="/js/**" location="/js/" />
```

# 한글처리

- **web.xml에 CharacterEncodingFilte 추가**

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

# 루트 웹 애플리케이션 컨텍스트

- IoC 컨테인 Application Context는 트리 구조를 가짐
- 모든 Application Context는 부모 Application Context를 가질 수 있으며, 계층구조 안에 모든 Application Context는 각각 독립적인 설정 정보를 사용하여 Spring 빈 객체를 생성하고 관리
- 의존성 주입을 위해 Spring 빈을 찾을 때 먼저 자신의 Application Context를 관리하는 Spring 빈 중에서 찾는다
- 자신을 관리하는 Spring 빈 중에 없는 경우 부모 Application Context에게 Spring 빈을 찾아 줄 것을 요청
- 계층구조를 따라 최상위 root Context까지 요청 가능

## 루트 웹 애플리케이션 등록-web-xml

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

- ContextLoaderListener : 루트 웹 애플리케이션 컨텍스트를 시작, 종료

## 컨텍스트 파라미터 지정

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

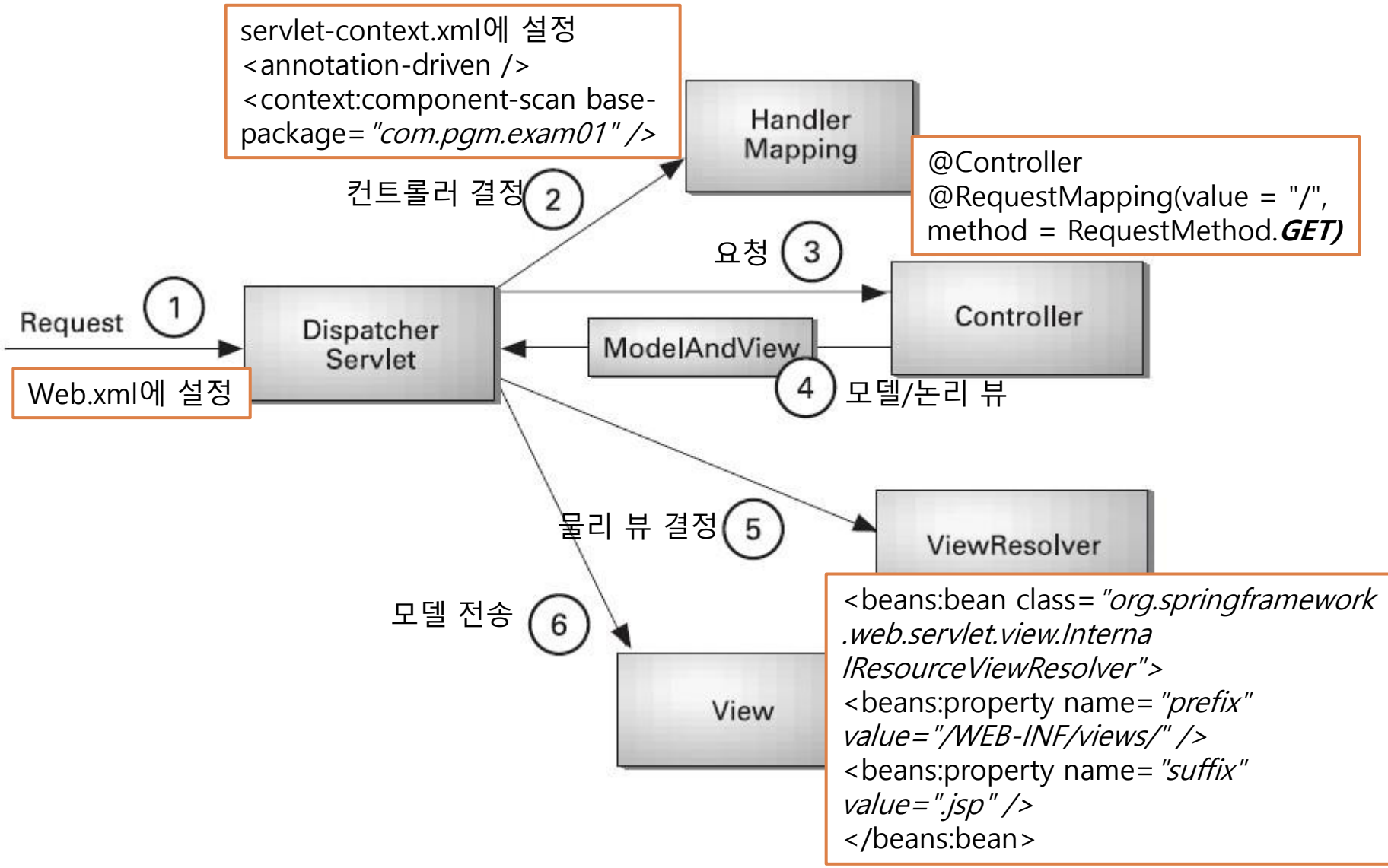
- 루트 컨텍스트가 사용할 Spring 설정 파일의 위치를 지정한 contextConfigLocation 설정

# 2. 컨트롤러 구현

- Home 컨트롤러 실행 과정
- 컨트롤러 클래스
- 모델 클래스
- 요청클래스 핸들러 메서드 매개변수 타입
- 요청핸들러 메서드 반환타입
- 뷰 이름 리턴

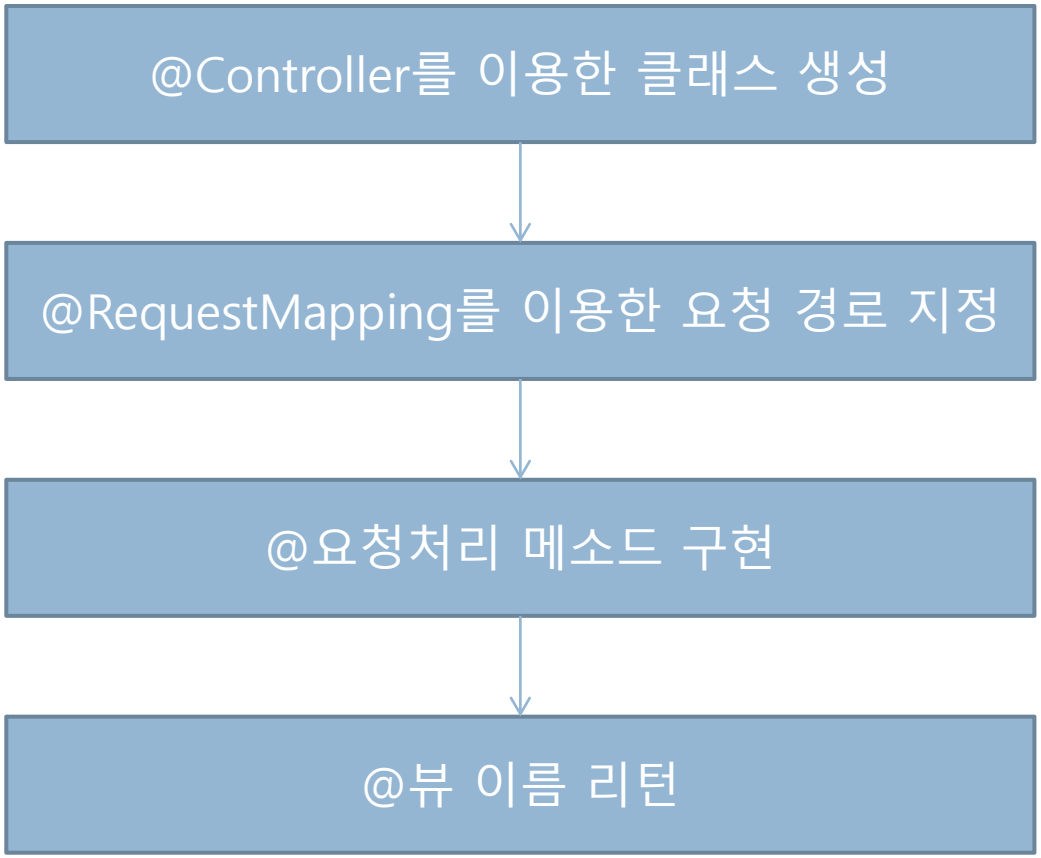
# 2. 컨트롤러 구현

- **Home 컨트롤러 실행 과정**



# 2. 컨트롤러 구현

## 컨트롤러 클래스 제작 순서




```
@Controller
public class HomeController {

}
```

# 뷰에 데이터 전달

## Model 클래스를 이용한 전달

- Model 객체를 파라미터로 받음

```
@RequestMapping("/board/content")
public String content(Model model){
        model.addAttribute("id",30);
        return "board/content";
}
```

- ModelAndView 객체 생성

```
@RequestMapping("/board/reply")
public ModelAndView reply(){
        ModelAndView mv=new ModelAndView();
        mv.addObject("id",20);
        mv.addObject("name", "홍길동");
        mv.setViewName("board/reply");
        return mv;
}
```

# Controller 예제

```
@RequestMapping("/board/view")
public String view(){
    return "board/view";
}
```

```
@RequestMapping("/board/content")
public String content(Model model){
    model.addAttribute("id", 30);
    return "board/content";
}
```

```
@RequestMapping("board/reply")
public ModelAndView reply(){
    ModelAndView mv=new ModelAndView();
    mv.addObject("id", "abcd");
    mv.addObject("pw", "a1234");
    mv.setViewName("board/reply");
    return mv;
}
```

# 2. 컨트롤러 구현

- 클래스에 @RequestMapping 적용

```
7  @Controller
8  @RequestMapping("/board")
9  public class HomeController {
```

클래스에 @RequestMapping 적용

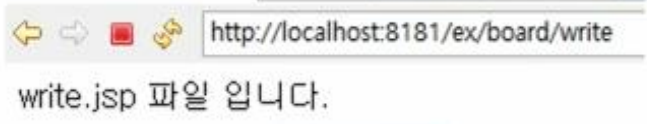
/board

```
@RequestMapping("/write")
public String write(Model model) {

    model.addAttribute("id", 30);

    return "board/write";
}
```

메소드에 @RequestMapping 적용

/write

조합된 요청 경로 : /board/write



# 2. 컨트롤러 구현

## 클래스에 @RequestMapping 적용 예

- Controller 작성

```
@Controller
@RequestMapping(“/board”)
public class HomeController{
        @RequestMapping("/content")
        public String content(Model model){
            model.addAttribute("id", 30);
            return "board/content";
        }
}
```

- View 작성
  - Board 폴더 작성 후 content.jsp작성