Spring MVC란?

Spring MVC는 **Spring Framework**에서 제공하는 MVC 패턴의 구현체입니다. 기본적인 MVC 패턴을 기반으로 하고 있지만, 웹 애플리케이션 개발에 적합하도록 여러 가지 부가적인 기능과 설정을 제공합니다. 즉, Spring MVC는 단순히 MVC 패턴을 사용하는 것이 아니라, **Spring 프레임워크의 IoC(제어의 역전) 컨테이너와 결합하여 더 강력한 기능**을 제공하는 구조입니다.

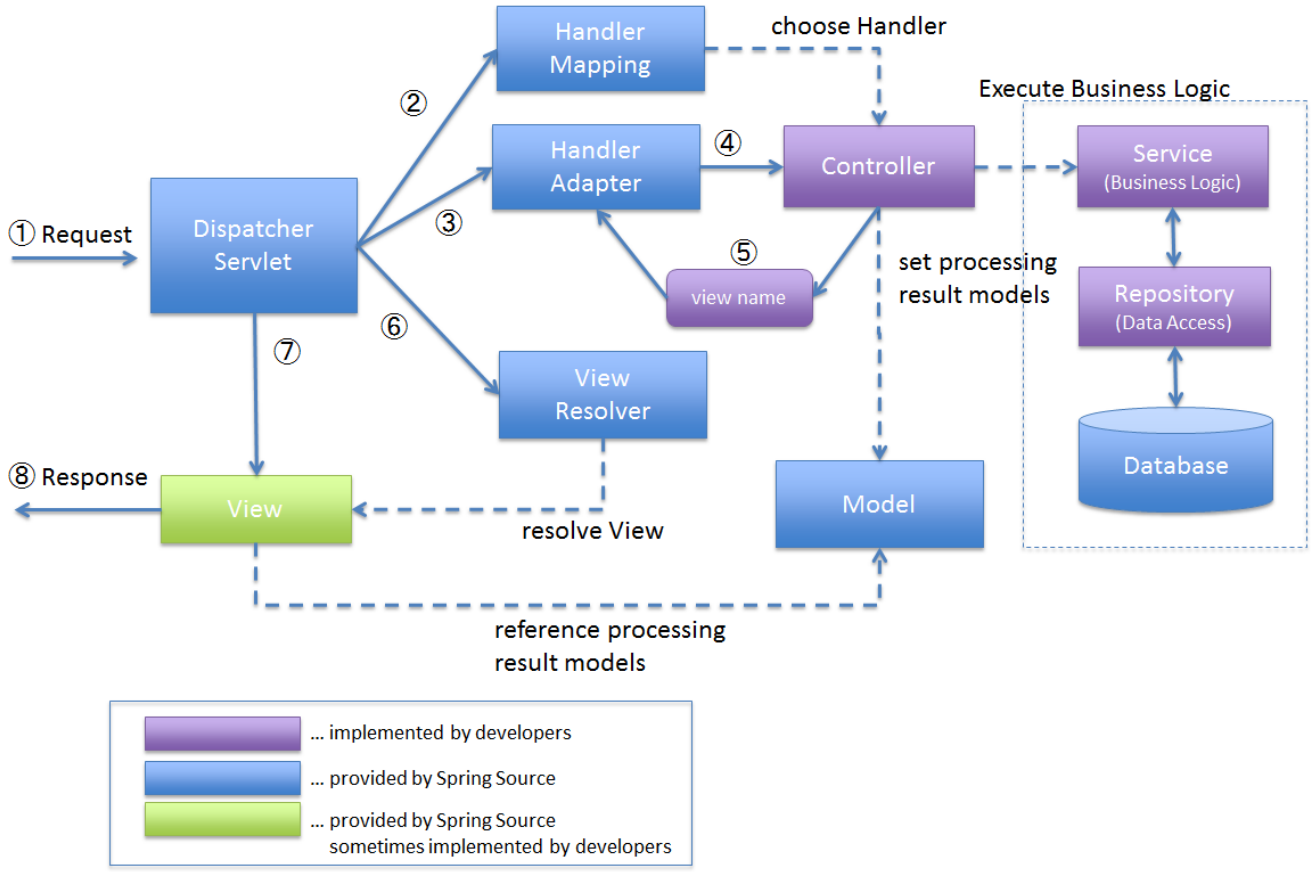
- **Spring MVC의 주요 특징**:
  - `DispatcherServlet`이라는 프론트 컨트롤러가 모든 요청을 일괄적으로 받아 적절한 컨트롤러로 요청을 분배합니다.
  - 어노테이션 기반의 설정(`@Controller`, `@RequestMapping`)을 통해 더 직관적이고 간단한 웹 애플리케이션 구성이 가능합니다.
  - 데이터 바인딩, 폼 유효성 검사, 인터셉터, 메시지 변환 등 다양한 기능을 지원합니다.
  - RESTful 웹 서비스 구축을 위한 다양한 기능을 제공하여, 단순한 페이지 반환뿐만 아니라, JSON/XML 기반의 API 구현도 용이합니다.

1. Spring MVC의 아키텍처

Spring MVC는 **Model, View, Controller**의 세 가지 주요 구성 요소로 나뉩니다. 각 구성 요소는 특정 역할을 담당하여 웹 애플리케이션의 흐름을 제어하고, 데이터를 처리하고, 사용자에게 적절한 응답을 제공합니다.

- **Model**: 애플리케이션의 데이터와 비즈니스 로직을 담당합니다. 데이터베이스와의 상호작용이나 서비스 로직, 데이터 처리가 포함됩니다.
- **View**: 사용자가 보는 화면을 구성합니다. HTML, JSP, Thymeleaf 등의 템플릿 엔진을 이용하여 사용자에게 데이터를 시각적으로 전달합니다.
- **Controller**: 클라이언트 요청을 받고, 요청을 처리하여 적절한 모델 데이터를 생성하고 뷰에 전달하는 역할을 합니다. 주로 `@Controller`와 `@RequestMapping` 어노테이션을 사용하여 URL 요청을 처리합니다.

1. **DispatcherServlet 아키텍처 흐름*



Spring MVC 아키텍처는 Model-View-Controller 패턴을 기반으로 하여 웹 애플리케이션에서 요청을 처리하고 응답을 생성하는 구조입니다.

클라이언트 요청 (Client Request)

- 사용자가 브라우저에서 특정 URL로 요청을 보냅니다. 이 요청은 HTTP 프로토콜을 통해 전달됩니다.

Dispatcher Servlet

- **역할**: 모든 요청의 중앙 처리 지점입니다.
- 클라이언트의 요청을 받으면, 이를 분석하고 적절한 핸들러(컨트롤러)를 선택합니다.
- Spring 프레임워크에서 설정된 URL 매핑 정보를 기반으로 요청을 처리할 핸들러를 결정합니다.

Handler Mapping

- **역할**: 요청 URL에 기반하여 적절한 핸들러(컨트롤러)를 찾아 매핑합니다.
- Dispatcher Servlet은 Handler Mapping을 사용하여 요청에 적합한 컨트롤러를 선택합니다.
- 각 요청 URL에 대해 어떤 컨트롤러가 처리할지 매핑 정보가 저장되어 있습니다.

Handler Adapter

- **역할**: 선택된 핸들러를 호출할 수 있도록 지원하는 컴포넌트입니다.
- Handler Mapping에 의해 선택된 핸들러(컨트롤러)를 호출하기 위해 Handler Adapter가 사용됩니다.
- 이 어댑터는 컨트롤러의 메소드를 실행하고, 필요 시 요청 및 응답 객체를 전달합니다.

Controller

- **역할**: 비즈니스 로직을 처리하는 중심적인 역할을 수행합니다.
- 요청을 받고, 해당 요청에 대한 비즈니스 로직을 수행합니다. 필요한 경우 서비스 계층을 호출하여 데이터 처리를 합니다.
- 작업이 완료되면, 결과 데이터 모델과 함께 응답할 뷰의 이름을 반환합니다.

Service Layer

- **역할**: 비즈니스 로직을 담당합니다.
- Controller로부터 받은 요청을 처리하고, 필요한 데이터에 대한 CRUD 작업을 수행하기 위해 Repository를 호출합니다.
- 필요한 로직을 수행하고 결과를 컨트롤러에 반환합니다.

Repository Layer

- **역할**: 데이터베이스와의 상호작용을 처리합니다.
- 데이터베이스에 접근하여 필요한 데이터를 가져오거나 저장하는 역할을 합니다.
- JPA, MyBatis 등과 같은 데이터 접근 기술을 사용하여 데이터베이스와 통신합니다.

Model

- **역할**: 컨트롤러가 처리한 결과를 담는 데이터 객체입니다.
- 뷰에 전달할 데이터를 포함합니다. 이 데이터는 사용자가 요청한 정보일 수 있습니다.

View Resolver

- **역할**: 컨트롤러가 반환한 뷰 이름을 실제 뷰로 변환합니다.
- 뷰 이름을 기반으로 적절한 뷰 파일의 경로를 결정하고, 그에 따라 뷰를 생성합니다.
- JSP, Thymeleaf, FreeMarker 등 다양한 템플릿 엔진과 함께 사용할 수 있습니다.

View

- **역할**: 최종적으로 클라이언트에게 전달될 HTML 등의 결과를 생성합니다.
- View Resolver가 반환한 뷰 파일을 사용하여, 모델 데이터를 포함한 최종 결과를 생성합니다.
- 생성된 HTML은 Dispatcher Servlet을 통해 클라이언트에게 응답으로 전달됩니다.

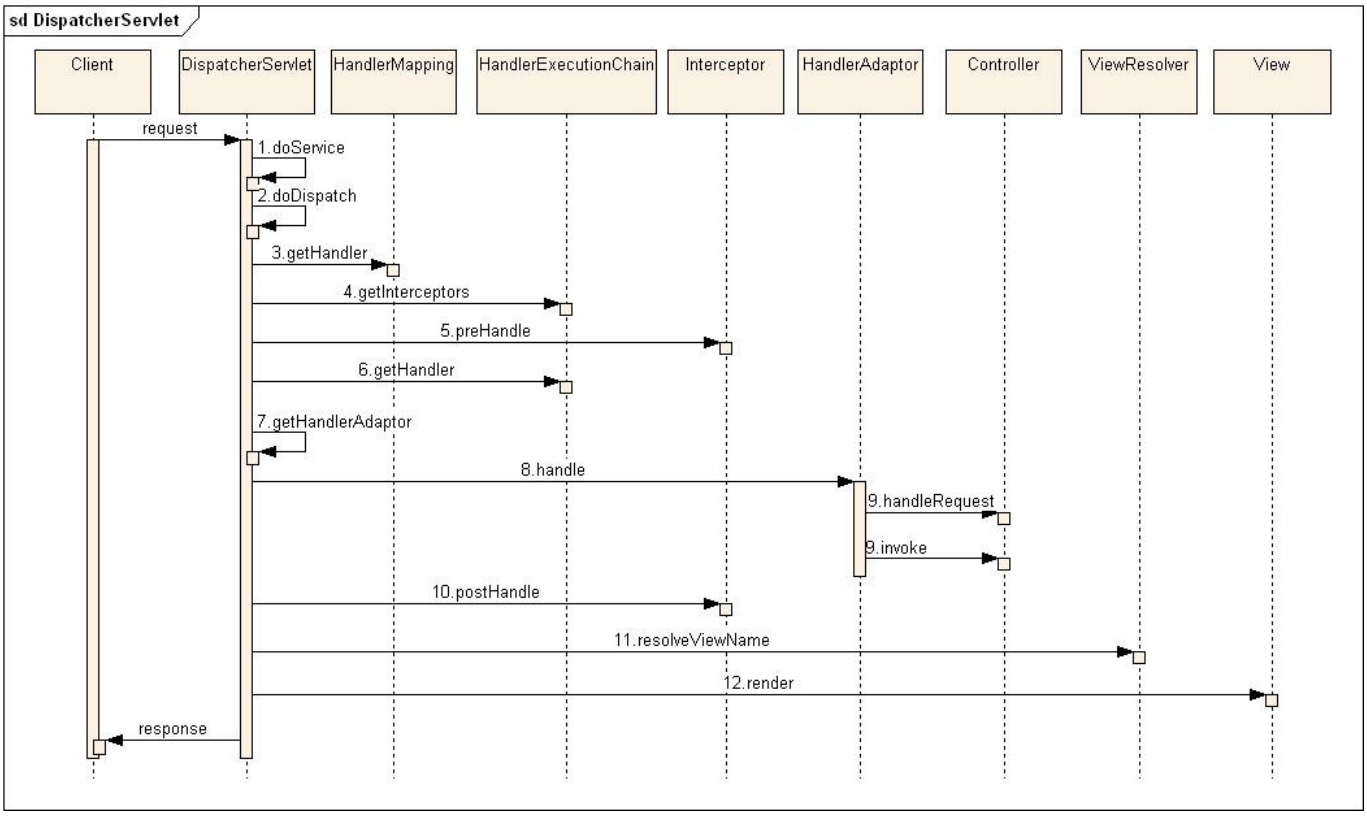
Response

- 최종적으로 클라이언트에게 응답이 전달됩니다. 이 응답은 사용자가 요청한 페이지나 데이터를 포함합니다.

전체 흐름 요약

1. 클라이언트가 HTTP 요청을 보냄.
2. Dispatcher Servlet이 요청을 받고, Handler Mapping을 통해 적절한 핸들러를 찾음.
3. Handler Adapter가 선택된 핸들러(컨트롤러)를 호출.
4. Controller가 비즈니스 로직을 실행하고, 결과 모델과 뷰 이름을 반환.
5. View Resolver가 뷰 이름을 기반으로 뷰를 결정.
6. View가 최종적으로 HTML 응답을 생성.
7. Dispatcher Servlet이 생성된 응답을 클라이언트에게 전달.

3. Spring MVC의 주요 구성 요소



> **Spring MVC의 구조** > Spring MVC는 요청을 처리하기 위해 다양한 컴포넌트들이 서로 협력하여 동작합니다. 각 컴포넌트의 역할은 다음과 같습니다:

```
- **DispatcherServlet**: 프론트 컨트롤러로 모든 요청을 받아 적절한 핸들러에 위임합니다.
- **HandlerMapping**: 특정 요청에 대해 어떤 핸들러가 처리할지 결정합니다.
- **HandlerAdapter**: 핸들러를 호출하고 결과를 반환할 수 있도록 도와줍니다.
- **Controller**: 비즈니스 로직을 처리하고 View 이름을 반환합니다.
- **ViewResolver**: View 이름을 기반으로 실제 View를 찾아 반환합니다.
- **View**: 최종적으로 사용자에게 응답으로 전달될 HTML을 렌더링합니다.
```

Spring MVC에서 자주 사용되는 구성 요소와 어노테이션에 대해 좀 더 자세히 알아보겠습니다.

3.1 `@Controller`

- `@Controller`는 특정 클래스를 Spring MVC의 컨트롤러로 정의합니다. 이 클래스는 클라이언트의 요청을 처리하고, 응답을 생성하는 역할을 합니다.

```java
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home() {
        return "home";  // View name 반환
    }
}
```

3.2 @RequestMapping

- `@RequestMapping`은 특정 URL 패턴을 메소드에 매핑하여, 특정 요청이 발생했을 때 해당 메소드가 호출되도록 설정합니다. 요청 메소드(`GET`, `POST`, 등)와 URL 패턴을 정의할 수 있습니다.

```
@Controller
public class UserController {
    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public String getUser(Model model) {
        model.addAttribute("username", "John");
        return "userView";  // userView.jsp 페이지 반환
    }
}
```

3.3 @RestController

- `@RestController`는 `@Controller`와 `@ResponseBody`를 결합한 어노테이션입니다. 주로 RESTful API를 만들 때 사용하며, 메소드가 반환하는 객체가 JSON이나 XML 등의 형태로 바로 클라이언트에게 전송됩니다.

```
@RestController
public class ApiController {
    @RequestMapping("/api/user")
    public User getUser() {
        return new User("John", "Doe");
    }
}
```

3.4 ModelAndView

- `ModelAndView`는 `Model` 데이터와 `View` 정보를 함께 담고 있는 객체입니다. 컨트롤러가 이 객체를 반환하면, `DispatcherServlet`은 이를 바탕으로 뷰를 렌더링합니다.

```
@Controller
public class ProductController {
    @RequestMapping("/product")
    public ModelAndView getProduct() {
        ModelAndView mav = new ModelAndView("productView");
        mav.addObject("productName", "Laptop");
        return mav;
    }
}
```

3.5 @ModelAttribute

- `@ModelAttribute`는 메소드 인자에 사용하여, 요청 파라미터를 특정 객체에 바인딩하거나, 모델 데이터를 뷰에 전달하는 용도로 사용됩니다.

```
@Controller
public class OrderController {
    @RequestMapping("/order")
    public String createOrder(@ModelAttribute Order order) {
        // Order 객체가 자동으로 요청 파라미터에 바인딩됩니다.
        return "orderView";
    }
}
```

3.6 `@RequestParam`

- `@RequestParam`은 HTTP 요청 파라미터를 메소드 인자로 바인딩할 때 사용됩니다.

```
@RequestMapping("/search")
public String search(@RequestParam("query") String query, Model model) {
    model.addAttribute("result", searchService.search(query));
    return "searchResultView";
}
```

4. Spring MVC의 주요 설정 파일

Spring MVC를 설정할 때는 주로 다음과 같은 파일을 사용하여 웹 애플리케이션의 동작을 정의합니다.

- **`web.xml`**: `DispatcherServlet`을 등록하고 초기화 파라미터를 설정합니다.
- **`servlet-context.xml`**: `DispatcherServlet`의 설정 파일로, 핸들러 매핑, 뷰 리졸버 등을 정의합니다.
- **`applicationContext.xml`**: 서비스, DAO, 비즈니스 로직과 관련된 스프링 빈을 정의하는 파일입니다.

5. Spring Boot와 Spring MVC

Spring Boot에서는 Spring MVC를 좀 더 쉽게 설정할 수 있도록 `@SpringBootApplication`과 자동 설정을 제공합니다. 이를 통해 `DispatcherServlet`이나 `ViewResolver` 설정을 직접 정의하지 않아도 자동으로 필요한 구성을 가져옵니다. `application.properties`나 `application.yml` 파일을 통해 추가적인 설정을 할 수 있습니다.

이와 같이 Spring MVC는 웹 애플리케이션의 기본적인 요청 처리부터 복잡한 데이터 바인딩 및 템플릿 엔진 연동까지 다양한 기능을 제공하여, 유지보수성과 확장성을 높여줍니다. 필요한 부분을 설정하여 더욱 강력한 웹 애플리케이션을 만들 수 있습니다.

결론

Spring MVC 아키텍처는 요청을 효율적으로 처리하고, 비즈니스 로직과 UI를 분리하여 유지보수성을 높이는 구조입니다. 이러한 흐름은 코드의 재사용성과 테스트 용이성을 극대화하며, 확장성과 유연성을 제공합니다.

4. Spring MVC와 일반적인 MVC의 차이점 및 연관성

Spring MVC는 일반적인 MVC 패턴의 원리를 따르지만, **웹 애플리케이션 개발에 특화된 구현체**입니다. 따라서 Spring MVC와 일반적인 MVC 패턴은 설계 원칙은 동일하지만, 구체적인 구현 방식이나 사용되는 기술 스택에서 차이가 있습니다.

4.1 연관성

- **기본 설계 원칙은 동일**: Spring MVC는 `Model`, `View`, `Controller`의 개념을 동일하게 사용하며, 각 역할의 분리를 통해 애플리케이션의 구조를 명확하게 나누고 있습니다.
- **비즈니스 로직의 분리**: 비즈니스 로직(`Model`)은 일반적인 MVC와 마찬가지로 `Service`나 `DAO` 계층에서 관리됩니다.
- **컨트롤러의 역할**: Spring MVC의 `@Controller`와 `@RequestMapping`을 통해, 일반적인 MVC의 컨트롤러 역할을 수행하며, 요청을 받고 데이터를 가공하여 뷰에 전달하는 역할을 합니다.

4.2 차이점

- **구현체의 차이**:
  - **일반적인 MVC 패턴**은 특정 프레임워크에 의존하지 않으며, 개념적인 설계 패턴입니다. 다양한 언어와 프레임워크에서 구현할 수 있습니다.
  - **Spring MVC**는 Spring 프레임워크의 일부로, Spring의 다양한 모듈(IoC 컨테이너, AOP, 트랜잭션 관리 등)과 통합되어 동작합니다.
- **핵심 요소의 추가**: 일반적인 MVC에는 없는 Spring MVC의 주요 구성 요소가 존재합니다.
  - `DispatcherServlet`: 모든 요청을 중앙에서 제어하는 프론트 컨트롤러로, Spring MVC의 핵심적인 요소입니다.
  - `ViewResolver`: 뷰 이름을 바탕으로 렌더링할 뷰를 찾아주는 역할을 합니다.
  - `HandlerMapping`: 특정 URL 요청을 어떤 컨트롤러가 처리할지 매핑하는 역할을 합니다.
- **어노테이션 사용**: Spring MVC는 어노테이션 기반의 설정(`@Controller`, `@RequestMapping`, `@GetMapping`, `@PostMapping`)을 사용하여 간편하게 설정이 가능합니다. 일반적인 MVC는 설정 방식이 다양하며, 특정 언어에서는 명시적으로 라우터나 컨트롤러를 지정해야 합니다.
- **부가 기능의 지원**: Spring MVC는 기본적인 MVC 패턴의 구현 외에도 다음과 같은 다양한 기능을 제공합니다.
  - **데이터 바인딩**: `@ModelAttribute`나 `@RequestParam`을 통해 요청 데이터를 자동으로 객체에 바인딩합니다.
  - **폼 유효성 검사**: `@Valid`와 같은 어노테이션을 통해 폼 데이터를 쉽게 검증할 수 있습니다.
  - **메시지 변환**: `HttpMessageConverter`를 사용하여 JSON, XML 등의 다양한 포맷으로 데이터를 주고받을 수 있습니다.
  - **인터셉터**: `HandlerInterceptor`를 사용하여 요청 전후에 특정 로직을 수행할 수 있습니다. 일반적인 MVC 패턴에서는 이러한 기능을 직접 구현해야 합니다.
- **REST 지원**: Spring MVC는 일반적인 MVC 패턴과 달리, RESTful API를 보다 쉽게 구현할 수 있도록 `@RestController`, `@ResponseBody` 등의 어노테이션을 제공하여, JSON/XML 형태의 응답을 더 직관적으로 작성할 수 있습니다.

5. 정리

- **Spring MVC와 일반적인 MVC 패턴의 연관성**:
  - 둘 다 `Model`, `View`, `Controller`라는 구조적인 설계 원칙을 따르고 있습니다.
  - 주요 목표는 관심사의 분리와 애플리케이션의 유지보수성, 확장성을 높이는 것입니다.

- **Spring MVC와 일반적인 MVC 패턴의 차이점**:
  - Spring MVC는 단순한 설계 패턴이 아닌, Spring 프레임워크에 내장된 **MVC 구현체**입니다.
  - 일반적인 MVC는 특정 기술 스택이나 프레임워크에 종속되지 않고, 다양한 언어 및 프레임워크에서 사용될 수 있는 반면, Spring MVC는 **Spring의 다른 모듈**과 밀접하게 연관되어 동작합니다.
  - Spring MVC는 `DispatcherServlet`, `HandlerMapping`, `ViewResolver` 등과 같은 추가적인 구성 요소를 통해 **웹 애플리케이션의 복잡한 흐름을 쉽게 관리**할 수 있도록 지원합니다.