

Presentation Layer

● 웨이터로 이해하는 Spring 표현 계층의 모든 것

■ Spring의 현관문을 열어보세요!



Spring 애플리케이션에서 **Presentation Layer** 는 외부 세계와 우리 시스템을 연결하는 현관문 역할을 합니다.

🍽️ 레스토랑 비유로 이해하기

"Presentation Layer는 레스토랑의 웨이터와 같아요!"



손님 (Client)

웹 브라우저, 모바일 앱
주문을 합니다



웨이터 (Controller)

Presentation Layer
주문을 받고 서빙합니다



주방 (Service)

Business Layer
실제 요리를 만듭니다

주문 요청



주문 접수



요리 제작



서빙



완료



핵심 컴포넌트



@Controller

HTML 페이지 반환 담당

서버사이드 렌더링

JSP

Thymeleaf

웹사이트의 화면을 만들어서 보여주는 역할을 해요!



@RestController

API 데이터(JSON) 반환 담당

REST API

JSON

Mobile App

모바일 앱이나 프론트엔드에 데이터를 전달하는 역할을 해요!



언제 사용하나요?



실시간 상호작용

외부 클라이언트가 우리 애플리케이션의 **기능에 접근해야 할 때 항상** 사용됩니다!

웹사이트 버튼 클릭

사용자가 "로그인", "회원가입" 버튼을 누를 때

모바일 앱 데이터 요청

앱이 서버로부터 사용자 정보를 가져올 때

시스템 간 API 호출

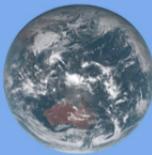
다른 서버가 우리 API를 호출할 때

실시간 업데이트

채팅, 알림 등 실시간 데이터 처리할 때



어디서 사용하나요?



외부 세계



Presentation
Layer



내부 시스템

애플리케이션 아키텍처의 **가장 바깥쪽 계층**에 위치합니다.

외부 클라이언트와 내부 비즈니스 계층 사이의 **중간 다리** 역할을 하며, 시스템의 '현관' 또는 '안내 데스크'라고 생각할 수 있어요!



어떻게 사용하나요?



어노테이션 기반 개발

주로 어노테이션(@)을 사용하여 요청을 처리하고 응답을 반환하는 방식으로 동작합니다.

```
// 🔍 이 클래스가 API 요청을 처리하는 Controller임을 선언
@RestController
public class UserController {

    // 💼 비즈니스 로직을 처리할 UserService 연결 (의존성 주입)
    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    // 📄 전체 사용자 목록을 조회하는 API
    @GetMapping("/api/users")
    public List<User> getAllUsers() {
        // 실제 데이터 조회는 Service에게 위임
        return userService.findAllUsers();
    }

    // 🔎 특정 사용자 조회 API
    @GetMapping("/api/users/{id}")
    public User getUserById(@PathVariable Long id) {
        // Service에게 id를 전달하여 데이터 조회 위임
        return userService.findUserById(id);
    }
}
```

1

@RestController로 Controller 클래스 지정

2

@GetMapping, @PostMapping으로 URL 매핑

3

@PathVariable, @RequestBody로 데이터 수신

4

Service 메서드 호출로 로직 위임

🤝 Controller의 핵심 역할

● 웨이터의 4가지 업무



요청 접수

Endpoint Mapping

손님의 주문을 듣고 어떤 요리인지 파악하는 것처럼, URL과 HTTP Method를 보고 어떤 메서드가 처리할지 결정해요.



데이터 검증

Data Binding & Validation

주문이 정확한지 확인하는 것처럼, 사용자가 보낸 데이터를 자바 객체로 변환하고 유효성을 검사해요.



주방에 전달

Delegate to Service

웨이터가 직접 요리하지 않는 것처럼, Controller도 비즈니스 로직을 Service 계층에 위임해요.



결과 서빙

Return Response

완성된 요리를 손님에게 서빙하는 것처럼, 처리 결과를 JSON이나 HTML 형태로 변환해서 반환해요.



@Controller vs @RestController

구분

@Controller

@RestController

 주요 목적
MVC 패턴의 **View(화면)** 반환REST API의 데이터(**JSON** 등) 반환
 메서드 반환 값

View 이름 (String)

객체, 데이터 (자동으로 JSON 변환)

 @ResponseBody
데이터 반환 시, 메서드에 **별도 추가 필요**클래스에 **자동으로 포함되어** 불필요
 사용 사례

서버사이드 렌더링 (JSP, Thymeleaf) 웹 페이지 개발

모바일 앱, 프론트엔드(React, Vue)와 연동하는 API 개발

 핵심 차이점: **@RestController = @Controller + @ResponseBody**
API 개발시에는 **@RestController**를, 전통적인 웹 개발시에는 **@Controller**를 사용하세요!

추가 핵심 컴포넌트들



@ControllerAdvice

전역 예외 처리

여러 Controller에서 발생하는 예외들을 한 곳에서 공통으로 처리! 코드 중복을 줄이고 예외 관리를 중앙화해요.



@ExceptionHandler

특정 예외 처리

특정 예외가 발생했을 때 실행될 메서드를 지정. 예를 들어 NullPointerException 발생 시 특정 에러 페이지를 보여줘요.



Filter

요청/응답 전처리

Spring 프레임워크 바깥 Servlet 단계에서 동작. 인코딩 변환, 보안 검사 등을 수행해요.



Interceptor

Spring MVC 전후처리

Spring MVC 내부에서 동작. 사용자 인증 확인, 로깅, 추가 데이터 처리 등 비즈니스 로직에 가까운 작업을 담당해요.



왜 이렇게 나누나요?



관심사의 분리 (Separation of Concerns, SoC)

시스템을 더 체계적이고 유연하게 만들기 위한 핵심 원칙이에요!

역할과 책임 명확화

Presentation Layer는 '웹 요청 처리'만, Business Layer는 '핵심 로직 처리'만 담당해서 코드 이해가 쉬워져요!

유지보수 용이성

프론트엔드 데이터 형식을 바꿔야 할 때, 다른 로직은 건드리지 않고 Presentation Layer만 수정하면 돼요!

유연성 및 확장성

웹 서비스에서 모바일 앱으로 확장할 때, 기존 비즈니스 로직은 그대로 두고 새로운 Controller만 추가하면 돼요!

테스트 용이성

각 계층을 독립적으로 테스트할 수 있어서 버그를 빠르게 찾고 수정할 수 있어요!



실전 시나리오로 이해하기



사용자가 "내 프로필 보기" 버튼을 클릭했을 때

1 클릭!



2 HTTP GET 요청



3 Controller 매핑



4 Service 호출



5 JSON 응답

상세 흐름

```
// ❶ 사용자 클릭 → HTTP GET /api/users/123

// ❷ Controller가 요청을 받음
@GetMapping("/api/users/{id}")
public User getUserProfile(@PathVariable Long id) {

    // ❸ 입력값 검증 (id가 유효한지 확인)
    if (id == null || id <= 0) {
        throw new IllegalArgumentException("Invalid user ID");
    }

    // ❹ Service에게 실제 작업 위임
    User user = userService.findUserById(id);

    // ❺ 결과를 JSON으로 변환해서 반환
    return user; // Spring이 자동으로 JSON 변환
}

// ❻ 프론트엔드가 JSON 데이터를 받아서 화면에 표시
```



실무 팁 & 베스트 프랙티스

⚠ 하지 말아야 할 것

- Controller에서 직접 데이터베이스 접근
- 복잡한 비즈니스 로직을 Controller에 작성

✓ 해야 할 것

- 입력값 검증과 응답 형태 변환에만 집중
- @ControllerAdvice로 전역 예외 처리

- 예외 처리를 각 Controller마다 중복으로 작성

- 명확하고 RESTful한 URL 설계

성능 최적화 팁

@RequestMapping 보다는 @GetMapping, @PostMapping 등 구체적인 어노테이션 사용

@Valid를 활용한 자동 입력값 검증으로 코드 간소화

ResponseBody로 HTTP 상태 코드까지 정확하게 관리



마무리: Presentation Layer의 핵심

"웨이터는 요리를 하지 않지만, 레스토랑에서 가장 중요한 역할을 해요!"

Presentation Layer의 가치

비록 복잡한 비즈니스 로직을 직접 처리하지는 않지만, 외부 세계와 내부 시스템을 연결하는 핵심 다리 역할을 합니다.

이 계층이 잘 설계되어 있으면 확장성, 유지보수성, 테스트 용이성이 모두 향상되어 더 좋은 소프트웨어를 만들 수 있어요!



다음에 배울 것

Service Layer (비즈니스 계층)

실제 비즈니스 로직이 처리되는 '주방'에서는 어떤 일이 일어나는지 알아보아요!



실습 추천

간단한 CRUD API 만들기

회원 관리 시스템을 만들어보면서 Presentation Layer를 직접 체험해보세요!



Spring Framework를 마스터하는 여정, 함께 해요! ❤

#Spring

#백엔드개발

#Java

#웹개발

#API개발

#PresentationLayer