@Controller와 @RestController 차이

1. @Controller

Spring MVC에서 사용하는 어노테이션으로, 주로 **뷰를 반환하는 컨트롤러**를 만들 때 사용됩니다. 이 어노테이션이 붙은 클래스는 HTTP 요청을 처리하는 메서드를 정의하고, 일반적으로 **HTML, JSP, Thymeleaf 등의 뷰 페이지를 반환**하는 역할을 합니다.

특징:

- **뷰를 반환**: `@Controller`가 사용된 클래스는 주로 뷰 템플릿을 반환하는 메서드를 정의합니다.
- `ModelAndView` 또는 `Model` 객체를 사용하여 데이터를 뷰로 전달할 수 있습니다.
- `@RequestMapping` 또는 `@GetMapping`, `@PostMapping`과 같은 메서드 단위의 어노테이션을 사용하여 요청을 처리합니다.

ex)

```
@Controller
public class MyController {

    @GetMapping("/home")
    public String home(Model model) {
        model.addAttribute("message", "Hello, World!");
        return "home";  // home.html 또는 home.jsp 같은 뷰 이름을 반환
    }
}
```

---

2. @RestController

RESTful 웹 서비스를 구축할 때 사용하는 어노테이션입니다. 기본적으로 `@Controller`와 `@ResponseBody`의 조합으로, HTTP 요청에 대한 **JSON, XML, 텍스트 등**을 반환합니다. 즉, 클라이언트에 데이터를 직접 전달하는 API 서버를 구현할 때 사용됩니다.

특징:

- **데이터를 반환**: `@RestController`가 사용된 클래스는 뷰를 반환하는 것이 아니라, 데이터(주로 JSON 형식)를 반환합니다.
- `@ResponseBody`가 기본적으로 적용되어 있기 때문에, 메서드에서 반환하는 객체는 HTTP 응답의 바디에 자동으로 직렬화되어 전송됩니다.
- REST API를 구축할 때 사용됩니다.

ex)

```
@RestController
public class MyRestController {

    @GetMapping("/api/message")
    public String getMessage() {
        return "Hello, World!";  // JSON 형식으로 응답됨
    }
}
```

---

3. `@Controller`와 `@RestController`의 응답 처리 메커니즘

`@Controller`와 `@RestController`는 기본적으로 **응답을 처리하는 방식**에서 차이가 있음.

`@Controller`에서의 응답 처리

`@Controller`에서 `@ResponseBody`를 사용하지 않으면, 메서드가 반환하는 값은 View **이름**으로 간주됨. 즉, Spring은 반환된 문자열을 뷰 템플릿과 연결하여 렌더링하려고 함. View 이름을 반환하고, 실제 View(HTML, JSP 등)를 렌더링하는 방식.

단 `@ResponseBody`를 사용하면, 뷰 템플릿을 렌더링하지 않고 반환된 데이터를 **HTTP 응답 바디에 직접 담아서** 클라이언트에 전달. 이때, 반환되는 객체는 **자동으로 JSON** 또는 **XML**로 변환.

`@RestController`에서의 응답 처리

`@RestController`는 `@ResponseBody`가 기본적으로 적용되므로, 메서드가 반환하는 객체는 **자동으로 직렬화되어 HTTP 응답 바디**로 전송됨. 기본적으로 **JSON** 형식으로 변환되며, XML이나 다른 형식으로 변환하려면 적절한 설정이 필요.

---

4. 응답 형식 처리

`@RestController`에서 반환되는 데이터는 자동으로 **직렬화**되어 HTTP 응답에 포함됨. 기본적으로 **JSON** 형식으로 처리되지만, 사용자가 `Accept` 헤더를 통해 다른 형식(XML 등)을 요청하면 이를 처리할 수 있다. 이를 위해 `@RequestMapping`이나 `@GetMapping`에 `produces` 속성을 사용이 가능하다.

예시: `produces` 속성 사용

```
@RestController
public class ApiController {

    @GetMapping(value = "/api/message", produces = "application/xml")
    public Message getMessage() {
        return new Message("Hello, World!");
    }
}
```

이렇게 설정하면 클라이언트가 `Accept: application/xml` 헤더를 보낼 경우, 응답이 **XML**로 반환됨.

---

5. 컨트롤러와 HTTP 메서드

`@Controller`와 `@RestController`에서 처리하는 **HTTP 메서드**(`GET`, `POST`, `PUT`, `DELETE`)는 동일하지만, 이들이 **응답 처리 방식**에서 다르게 동작할 수 있다는 점을 강조할 수 있다.

예시: `@Controller`에서 `@ResponseBody`를 사용하는 경우

```
@Controller
public class MyController {

    @GetMapping("/api/data")
    @ResponseBody
    public List<String> getData() {
        List<String> data = new ArrayList<>();
```

```
        data.add("Item 1");
        data.add("Item 2");
        return data;  // JSON으로 응답
    }
}
```

위 예시처럼 `@Controller`에서 `@ResponseBody`를 사용하면, `@RestController`처럼 데이터를 직접 응답 바디에 담을 수 있다.

---

6. 뷰 리졸버 (View Resolver)

`@Controller`를 사용할 때, 반환값이 View 이름이라면 Spring은 이를 View **리졸버(View Resolver)**를 사용하여 실제 View 파일로 매핑. 이 View 리졸버는 반환된 View 이름을 기반으로 적절한 View 파일을 찾아서 클라이언트에 렌더링함.

View 리졸버는 **JSP**, **Thymeleaf**, **Freemarker** 등의 템플릿 엔진을 사용할 때 유용함. 이러한 View 엔진들은 모델 데이터를 템플릿에 삽입하여 최종 HTML 페이지를 생성.

예시: `@Controller`에서 뷰 리졸버 사용

```
@Controller
public class HomeController {

    @GetMapping("/home")
    public String home(Model model) {
        model.addAttribute("message", "Welcome to the Home Page!");
        return "home";  // home.html 또는 home.jsp와 연결된 뷰 파일을 반환
    }
}
```

위 예시에서, `"home"`은 **View 이름**이고, Spring은 이를 `home.html`이나 `home.jsp`로 변환하여 클라이언트에 반환.

---

7. `@RestController`의 직렬화 및 `HttpMessageConverter`

`@RestController`에서 반환되는 객체는 자동으로 `HttpMessageConverter`에 의해 직렬화됨. Spring은 기본적으로 `Jackson` 라이브러리를 사용하여 객체를 JSON 형식으로 변환. 이 외

에도 XML 직렬화와 같은 다른 형식을 지원하기 위해 `Jackson`, `Gson`, `JAXB` 등의 라이브러리를 추가적으로 사용할 수 있음.

예시: `HttpMessageConverter` 활용

```
@RestController
public class ApiController {

    @PostMapping("/api/user")
    public User createUser(@RequestBody User user) {
        // User 객체를 받아 처리하고, 클라이언트에게 JSON 형식으로 응답
        return user;
    }
}
```

위 코드에서 `@RequestBody`로 클라이언트로부터 받은 JSON 데이터를 **자동으로 Java 객체로 변환**하고, `@RestController`는 이 객체를 **JSON 형식으로 응답**.

---

8. 실제 사용 예: 웹 애플리케이션 vs API 서버

`@Controller`와 `@RestController`는 용도에 따라 다르게 사용. 간단한 **웹 애플리케이션**에서 페이지를 렌더링하려면 `@Controller`를 사용하고, **프론트엔드**와 **백엔드**가 분리된 **RESTful API**를 제공하려면 `@RestController`를 사용.

`@Controller` 사용 예 (웹 애플리케이션)

```
@Controller
public class WebController {

    @GetMapping("/home")
    public String showHomePage(Model model) {
        model.addAttribute("welcomeMessage", "Welcome to the Home Pag
e!");
        return "home";  // home.html 또는 home.jsp로 렌더링
```

```
    }
}
```

`@RestController` 사용 예 (API 서버)

```
@RestController
public class ApiController {

    @GetMapping("/api/users/{id}")
    public User getUser(@PathVariable Long id) {
        return userService.getUserById(id);  // User 객체를 JSON으로 반환
    }
}
```

---

9. 비교

특징	@Controller	@RestController
목적	View 페이지 반환 (웹 애플리케이션)	데이터 반환 (RESTful API)
응답 형식	뷰 이름 (HTML, JSP 등)	JSON, XML, 텍스트 등 (HTTP 응답 바디)
어노테이션	`@Controller` 만 사용	`@RestController` 사용 (내부적으로 `@ResponseBody` 포함)
주요 사용처	서버 사이드 렌더링 (Spring MVC)	REST API 서버 (JSON 또는 다른 데이터 반환)

아래의 두 코드는 동일한 동작을 한다.

```
@Controller
@ResponseBody
public class MVCController {
    business logic...
}

@RestController
```

```
public class RestFulController {
    business logic...
}
```

END. 결론

전체적으로 `@Controller`와 `@RestController`의 차이는 **응답 형식**과 **목적**에서 발생. `@Controller`는 View 를 반환하여 웹 애플리케이션을 구축하는 데 사용되고, `@RestController`는 데이터(주로 JSON 형식)를 반환하여 RESTful API를 구축하는 데 사용됨. 이러한 차이를 명확히 이해하고, 각 용도에 맞게 적절한 어노테이션을 선택하면 보다 효율적으로 웹 애플리케이션을 구축할 수 있다.