@Controller와 @RestController의 차이

> HTTP Response Body가 생성되는 방식의 차이

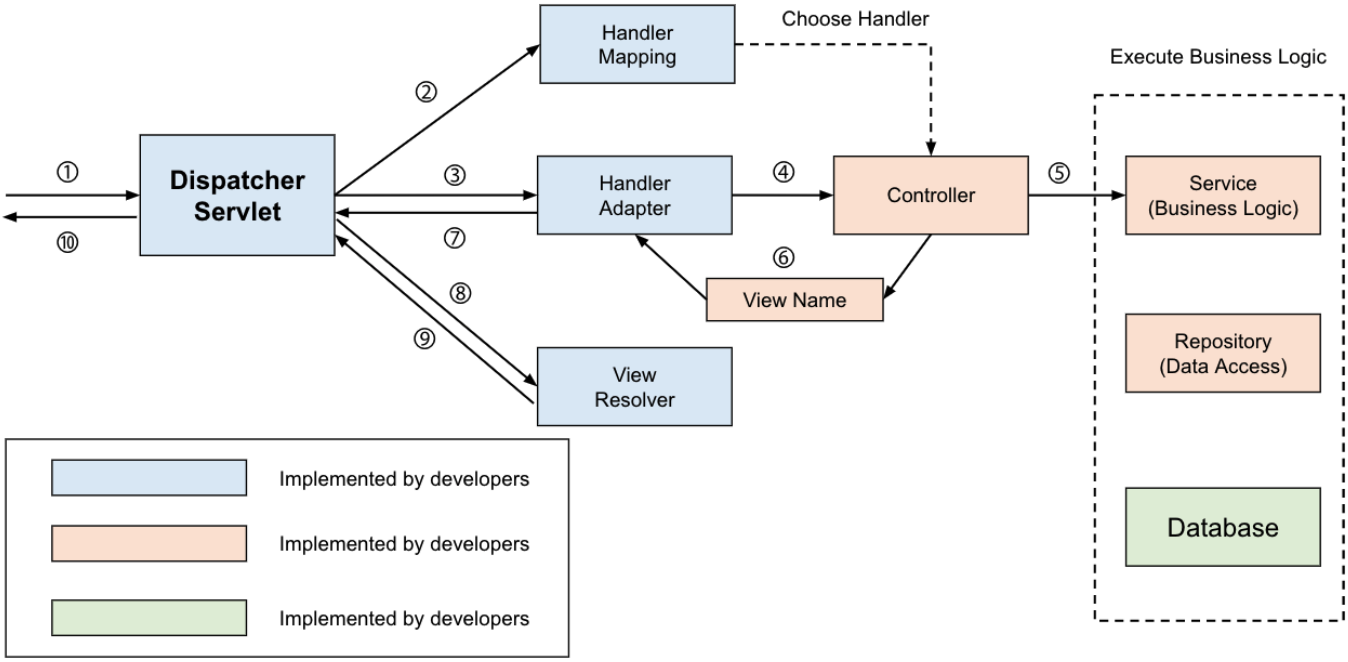
> @Controller : spring MVC 컨트롤러

> @RestController : Restful 웹 서비스의 컨트롤러

@Controller

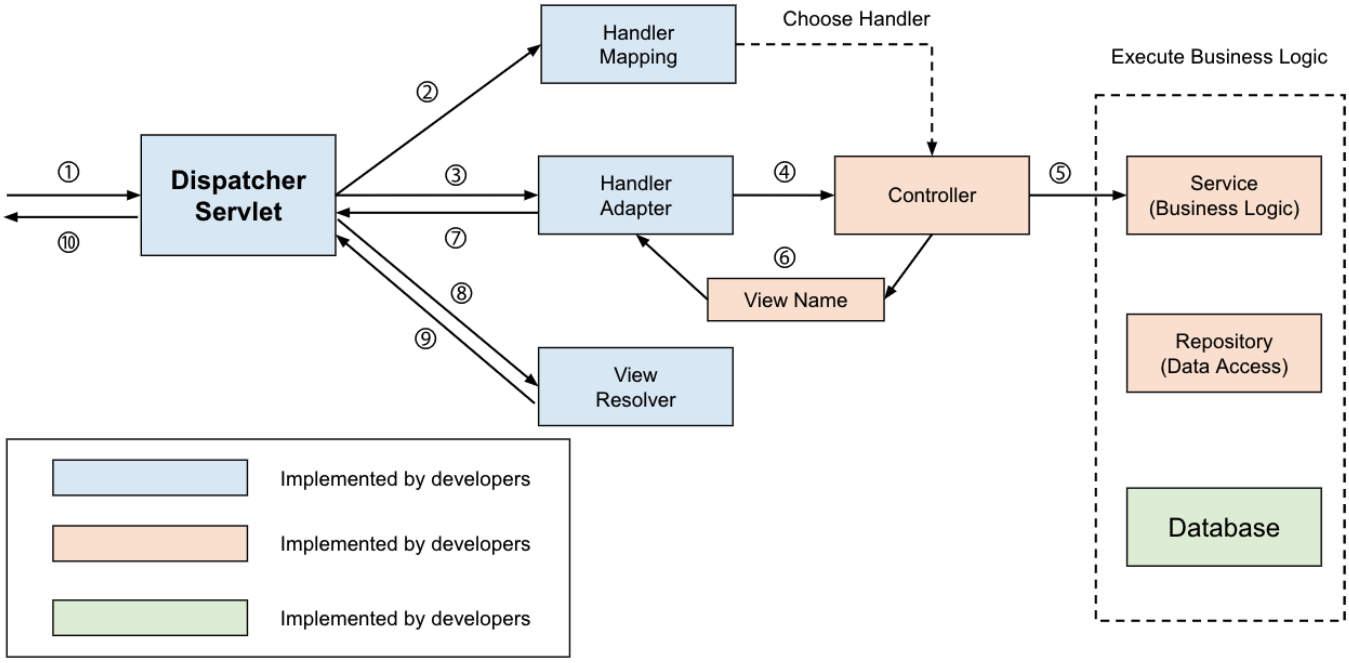
Controller로 View 반환하기

- 주로 클라이언트의 요청으로 부터 View를 반환하기위해 사용



1. Client는 URI 형식으로 웹 서비스에 요청을 보냄
2. DispatcherServlet이 요청을 처리할 대상을 찾음
3. HandlerAdapter를 통해 요청을 Controller로 위임
4. Controller는 요청을 처리 후, ViewName을 반환
5. DispatcherServlet은 ViewResolver를 통해 ViewName에 해당하는 View를 찾아 사용자에게 반환

> 반환한 뷰의 이름으로 부터 View를 렌더링 하기 위해선 ViewResolver가 사용됨. ViewResolver 설저에 맞게 View를 찾아 렌더링



1. Client는 URI 형식으로 웹 서비스에 요청을 보낸다.
2. DispatcherServlet이 요청을 처리할 대상을 찾는다.
3. HandlerAdapter을 통해 요청을 Controller로 위임한다.
4. Controller는 요청을 처리한 후에 객체를 반환한다.
5. 반환되는 객체는 Json으로 Serialize되어 사용자에게 반환된다.

- ResponseEntity로 감싸서 반환
- 객체 반환은 ViewResolver 대신 HttpMessageConverter가 동작
    - 반환데이터에 따라 여러 Converter를 골라서 동작
    - 문자열 : StringHttpMessageConverter
    - 객체 : MappingJackson2HttpMessageConverter
- Spring은 클라이언트의 HTTP Accept 헤더와, 서버의 컨트롤러 반환 타입 정보 둘을 조합해 HttpMessageConverter를 선택해 처리

```
@Controller
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @GetMapping(value = "/users")
    public @ResponseBody ResponseEntity<User>
findUser(@RequestParam("userName") String userName){
        return ResponseEntity.ok(userService.findUser(user));
    }

    @GetMapping(value = "/users/detailView")
    public String detailView(Model model, @RequestParam("userName") String
userName){
        User user = userService.findUser(userName);
        model.addAttribute("user", user);
        return "/users/detailView";
```

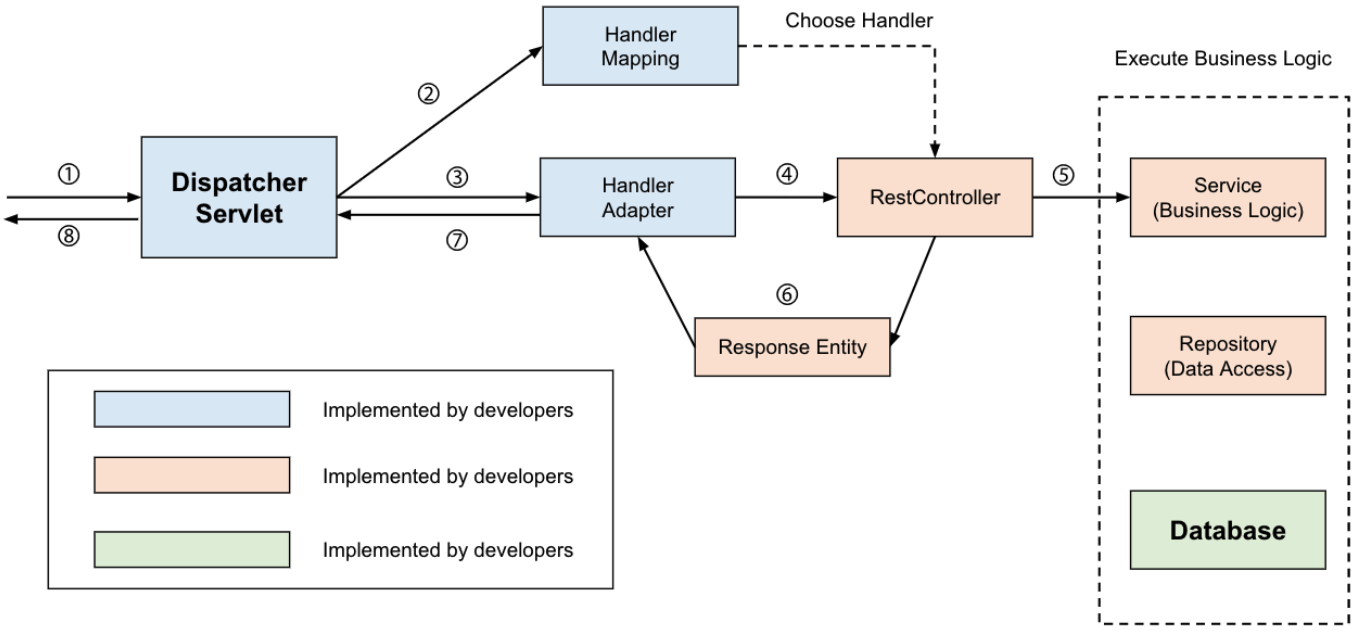
```
    }
}
```

@RestController

> @Controller에 @ResponseBody가 추가된 것

> Json형태로 객체 데이터 반환

> @Controller에 @ResponseBody가 추가된 것과 완벽히 동일하게 동작



1. Client는 URI 형식으로 웹 서비스에 요청을 보낸다.
2. DispatcherServlet이 요청을 처리할 대상을 찾는다.
3. HandlerAdapter을 통해 요청을 Controller로 위임한다.
4. Controller는 요청을 처리한 후에 객체를 반환한다.
5. 반환되는 객체는 Json으로 Serialize되어 사용자에게 반환된다.

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

    private final UserService userService;

    @GetMapping(value = "/users")
    public User findUser(@RequestParam("userName") String userName){
        return userService.findUser(user);
    }

    @GetMapping(value = "/users")
    public ResponseEntity<User>
findUserWithResponseEntity(@RequestParam("userName") String userName){
        return ResponseEntity.ok(userService.findUser(user));
```

```
    }
}
```

- 다만 User객체를 그대로 반환하므로, 응답 상태를 할 수 없다.
- 때문에 적당히 ResponseEntity로 감싸줘야함
- @Component
    - @Controller, @Service, @Repository, @Configuration

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Configuration {

    @AliasFor(annotation = Component.class)
    String value() default "";

    boolean proxyBeanMethods() default true;

}
```

@Bean, @Configuration, @Component 차이 및 비교 요약

@Bean, @Configuration

- 수동으로 스프링 컨테이너에 빈을 등록하는 방법
- 개발자가 직접 제어가 불가능한 라이브러리를 빈으로 등록할 때 불가피하게 사용
- 유지보수성을 높이기 위해 애플리케이션 전범위적으로 사용되는 클래스나 다형성을 활용하여 여러 구현체를 빈으로 등록 할 때 사용
- 1개 이상의 @Bean을 제공하는 클래스의 경우 반드시 @Configuration을 명시해 주어야 싱글톤이 보장됨

@Component

- 자동으로 스프링 컨테이너에 빈을 등록하는 방법
- 스프링의 컴포넌트 스캔 기능이 @Component 어노테이션이 있는 클래스를 자동으로 찾아서 빈으로 등록함
- 대부분의 경우 @Component를 이용한 자동 등록 방식을 사용하는 것이 좋음
- @Component 하위 어노테이션으로 @Configuration, @Controller, @Service, @Repository 등이 있음