Ważniejsze

# Adnotacje Java Spring Boot

Autor: Artur Pelo

# @adnotacje

01

Adnotacje związane z zarządzaniem komponentami

02

Adnotacje związane z wstrzykiwaniem zależności

03

Adnotacje konfiguracji aplikacji

04

Adnotacje Spring MVC

05

Adnotacje związane z transakcjami

06

Adnotacje bezpieczeństwa (Spring Security)

# Adnotacje związane z zarządzaniem komponentami

@Component, @Service , @Repository, @Controller, @RestController

# @Component

- Oznacza klasę jako komponent zarządzany przez kontener Springa.
- Jest to ogólna adnotacja dla komponentów, takich jak serwisy czy repozytoria.

```
@Component
public class MyComponent {
    public void doSomething() {
        System.out.println("Hello from MyComponent!");
    }
```

# @Service

- Specjalizacja **@Component** przeznaczona dla warstwy serwisowej.
- Używana do oznaczania klas zawierających logikę biznesową.

```
@Service
public class MyService {
    public String getMessage() {
        return "Hello from MyService!";
    }
}
```

# @Repository

- Specjalizacja `@Component` przeznaczona dla warstwy dostępu do danych (DAO, repository).
- Może obsługiwać wyjątki związane z bazą danych i konwertować je na DataAccessException.

```java
@Repository
public class MyRepository {
    public List<String> getData() {
        return List.of("One", "Two", "Three");
    }
}
```

DAO (Data Access Object) to wzorzec projektowy stosowany do oddzielenia warstwy dostępu do bazy danych od logiki biznesowej aplikacji. DAO ułatwia zarządzanie operacjami CRUD (Create, Read, Update, Delete) i pomaga w utrzymaniu kodu.

# @Controller

- Specjalizacja `@Component` używana w Spring MVC dla kontrolerów obsługujących żądania HTTP.

```java
@Controller
public class MyController {
    @GetMapping("/hello")
    public String hello() {
        return "hello";
    }
}
```

**Spring MVC** (Model-View-Controller) to część frameworka Spring, która umożliwia tworzenie aplikacji webowych w oparciu o wzorzec MVC. Jest to jeden z najczęściej używanych frameworków do budowy aplikacji internetowych w Java, który obsługuje żądania HTTP i pozwala na łatwe tworzenie REST API.

Model – reprezentuje dane aplikacji (np. obiekty JPA).

View – generuje widok dla użytkownika (np. Thymeleaf).

Controller – obsługuje żądania HTTP i zwraca odpowiedź.

# @RestController

- Kombinacja `@Controller` i `@ResponseBody`.
- Każda metoda zwraca bezpośrednio dane JSON/XML.

```
@RestController
public class MyRestController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }
}
```

# Adnotacje związane z wstrzykiwaniem zależności

@Autowired, @Qualifier, @Primary, @Value

# @Autowired

- Wstrzykuje zależności do komponentów zarządzanych przez Springa.

```java
@Service
public class MyService {
    public String serve() {
        return "Service is working!";
    }
}
@RestController
public class MyController {
    private final MyService myService;                  <-Uwaga, adnotacja również może poprzedzać tą linię kodu (niewskazane)
    @Autowired                                          <-Uwaga adnotacja może być również w setterze (niewskazane)
    public MyController(MyService myService) {
        this.myService = myService;
    }
    @GetMapping("/service")
    public String callService() {
        return myService.serve();
    }
}
```

# @Qualifier

- Wskazuje konkretną implementację beana, gdy istnieje więcej niż jedna.

```
@Component("firstService")
public class FirstService implements MyService { }

@Component("secondService")
public class SecondService implements MyService { }

@RestController
public class MyController {
    private final MyService myService;

@Autowired
    public MyController(@Qualifier("secondService") MyService myService) {
        this.myService = myService;
    }
}
```

# @Primary

- Określa domyślną implementację beana, gdy istnieje wiele możliwych.

```
@Primary
@Component
public class DefaultService implements MyService { }
```

# @Value

- Wstrzykuje wartości z plików konfiguracyjnych.

```
@Value("${app.name}")
private String appName;
```

# Adnotacje konfiguracji aplikacji

@Configuration, @Bean, @PropertySource

15.03.2025

# @Configuration

- Oznacza klasę jako źródło konfiguracji Springa.

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

# @Bean

- Tworzy i rejestruje beana w kontekście Springa.

```
@Bean
public MyRepository myRepository() {
    return new MyRepository();
}
```

# @PropertySource

- Wskazuje plik z właściwościami konfiguracyjnymi.

```
@PropertySource("classpath:application.properties")
public class Config { }
```

# Adnotacje Spring MVC

@RequestMapping, [@GetMapping, @PostMapping, @PutMapping, @DeleteMapping],

@PathVariable, @RequestParam, @RequestBody, @ResponseBody

# @RequestMapping

- Mapuje żądania HTTP na metody kontrolera.

```
@RequestMapping("/api")
@RestController
public class ApiController {
    @RequestMapping("/hello")
    public String hello() {
        return "Hello, API!";
    }
}
```

# @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

- Skrócone wersje @RequestMapping dla różnych metod HTTP.

```
@GetMapping("/users")
public List<User> getUsers() { ... }

  @PostMapping("/users")
public void createUser(@RequestBody User user) { ... }
```

# @PathVariable

- Pobiera wartość z adresu URL.

```
@GetMapping("/user/{id}")
public User getUser(@PathVariable Long id) { ... }
```

# @RequestParam

- Pobiera parametr z zapytania.

```
@GetMapping("/search")
public List<User> search(@RequestParam String name) { ... }
```

# @RequestBody

- Konwertuje ciało żądania HTTP na obiekt Java.

```
@PostMapping("/user")
public void createUser(@RequestBody User user) { ... }
```

# @ResponseBody

- Wskazuje, że metoda zwraca bezpośrednio dane JSON/XML.

```java
@ResponseBody
@GetMapping("/json")
public Map<String, String> json() {
    return Map.of("message", "Hello");
}
```

# Adnotacje związane z transakcjami

@Transactional

15.03.2025

# @Transactional

- Oznacza metodę lub klasę jako transakcyjną.

```
@Service
public class UserService {
    @Transactional
    public void updateUser(User user) { ... }
}
```

# Adnotacje bezpieczeństwa (Spring Security)

@Secured, @PreAuthorize, @RolesAllowed

# @Secured

- Określa role użytkownika wymagane do wykonania metody.

```
@Secured("ROLE_ADMIN")
   public void adminMethod() { ... }
```

# @PreAuthorize

- Definiuje warunki dostępu przed wykonaniem metody.

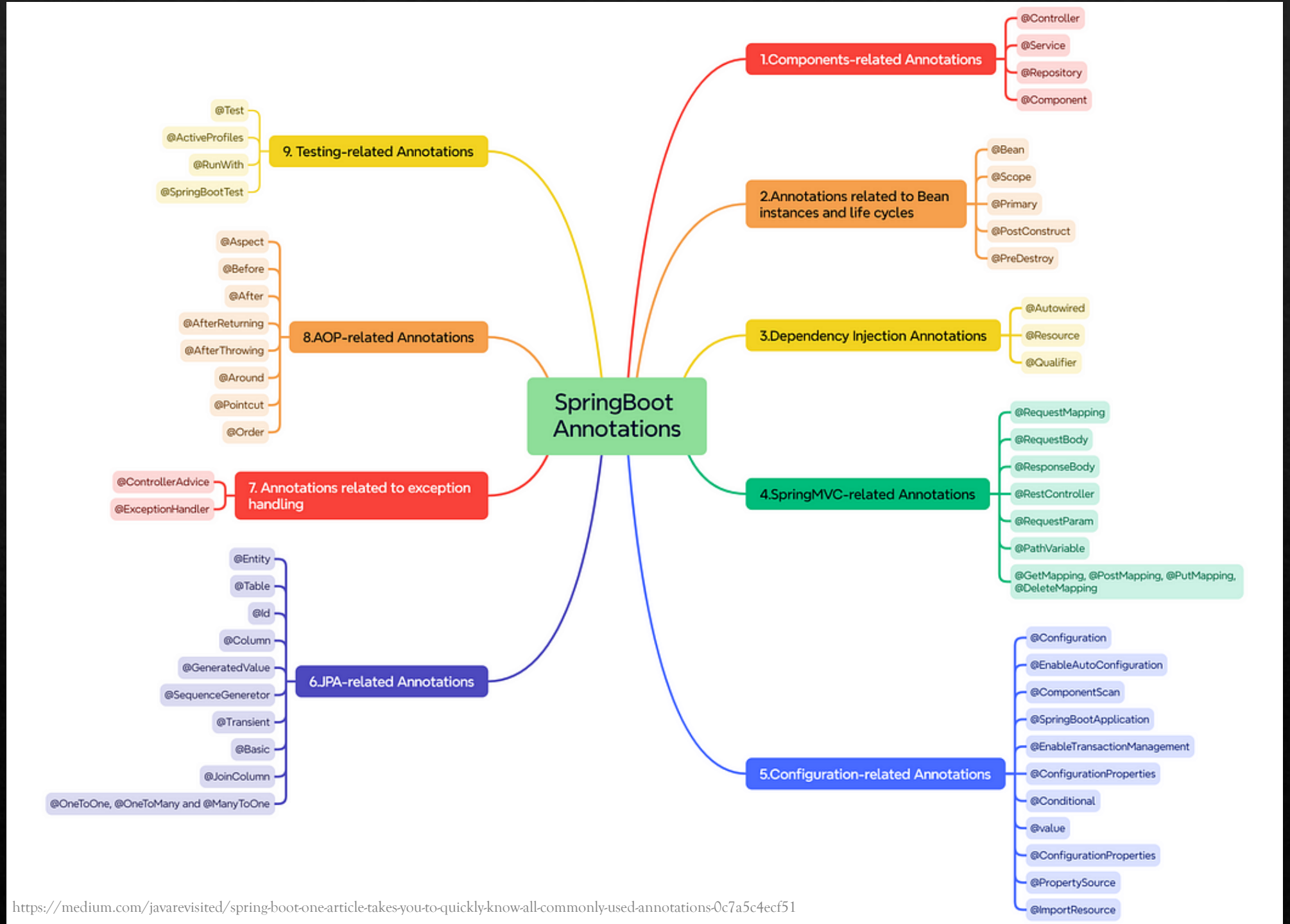
```
@PreAuthorize("hasRole('ADMIN')")
  public void secureMethod() { ... }
```

# @RolesAllowed

- Alternatywa dla @Secured.

```
@RolesAllowed({"ADMIN", "USER"})
  public void multiRoleMethod() { ... }
```

# Diagram adnotacji Spring Boot




https://medium.com/javarevisited/spring-boot-one-article-takes-you-to-quickly-know-all-commonly-used-annotations-0c7a5c4ecf51