Spring Boot

Marta Dychała, Piotr Libucha

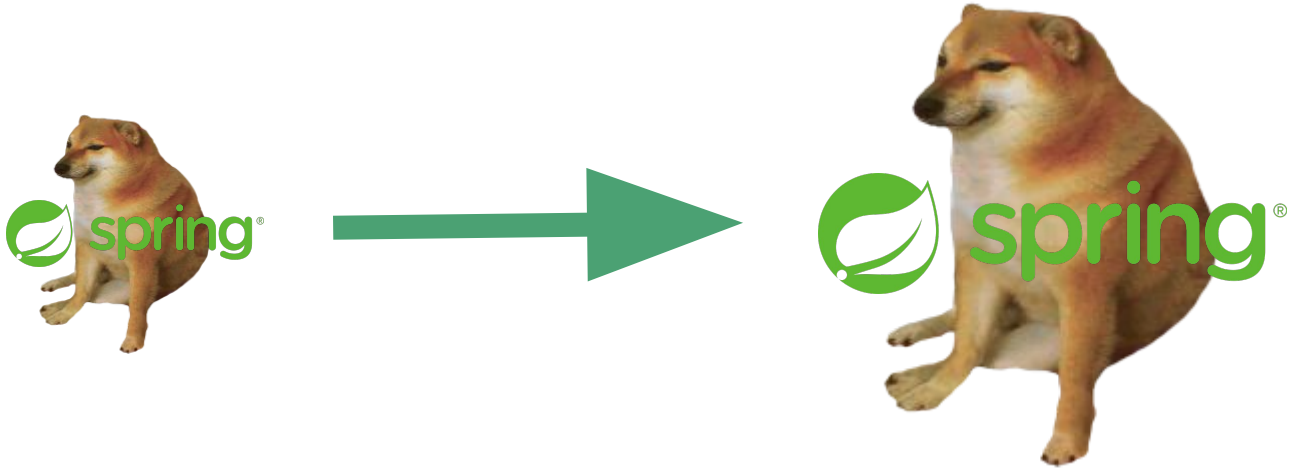
Spis treści / agenda

1. Czym jest Spring oraz Spring Boot
2. Geneza Spring Boot
3. Główne funkcjonalności
4. Popularne startery
5. Pojęcia związane ze Spring Boot (IoC, DI)
6. Przepływ danych w aplikacji Spring Boot
7. Podstawowe adnotacje

Czym jest Spring i Spring Boot?

Spring Boot - geneza

TLDR: Spring się rozrósł i to samo zrobiła jego konfiguracja



Porównanie konfiguracji - Spring

Aby skonfigurować aplikację JSP w Spring'u, należy zdefiniować komponenty aplikacji poprzez plik web.xml bądź poprzez klasę inicjalizującą...

```
public class MyWebAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext container) {
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();
        context.setConfigLocation("com.example");
        container.addListener(new ContextLoaderListener(context));
        ServletRegistration.Dynamic dispatcher = container
          .addServlet("dispatcher", new DispatcherServlet(context));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/");
    }
}
```

Porównanie konfiguracji - Spring

...a następnie należy dodać adnotację `@EnableWebMvc` wraz z adnotacją `@Configuration` i zaimplementować mechanizm rozpoznawania widoków przetwarzający widoki zwracane przez kontrolery.

```
@EnableWebMvc
@Configuration
public class ClientWebConfig implements WebMvcConfigurer {
   @Bean
   public ViewResolver viewResolver() {
      InternalResourceViewResolver bean = new InternalResourceViewResolver();
      bean.setViewClass(JstlView.class);
      bean.setPrefix("/WEB-INF/view/");
      bean.setSuffix(".jsp");
      return bean;
   }
}
```

Porównanie konfiguracji - Spring Boot

Dla porównania, aby skonfigurować aplikację JSP w Spring Boot’cie, w pliku application.properties wystarczy dodać dwa wpisy:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Funkcjonalności Spring Boot

- Wbudowany serwer HTTP (Tomcat/Jetty)
- Automatyczna konfiguracja bibliotek
- Metryki
- Gotowe konfiguracje - Spring Boot Starters
- Wsparcie dla silników szablonów (np. Thymeleaf)
- Wsparcie dla narzędzi ORM (np. Hibernate)
- Logi konfiguracji
- Ułatwione zarządzanie zależnościami

Popularne startery

- spring-boot-starter-**web**
- spring-boot-starter-**data-jpa**
- spring-boot-starter-**jdbc**
- spring-boot-starter-**security**
- spring-boot-starter-**test**
- spring-boot-starter-**thymeleaf**
- spring-boot-starter-**websocket**
- spring-boot-starter-**actuator**
- spring-boot-starter-**cache**
- spring-boot-starter-**oauth2-client**
- spring-boot-starter**-graphql**

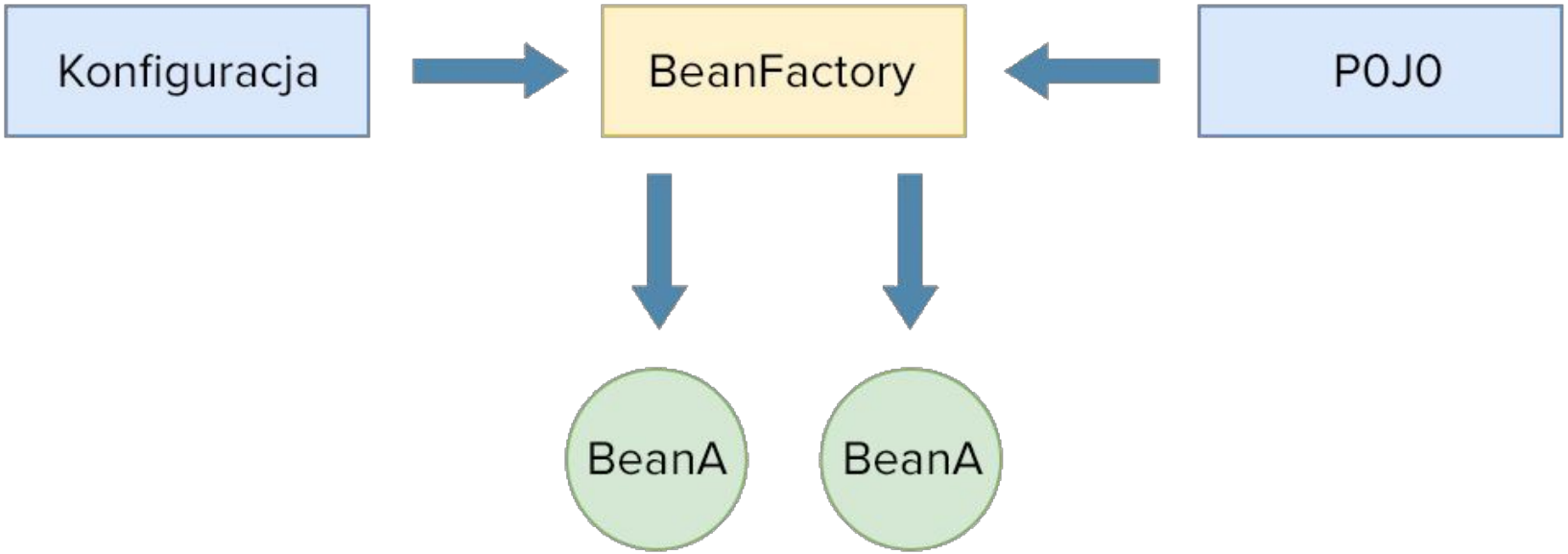
Inversion of Control (IoC)

- Odwrócenie kontroli (IoC) polega na przekazaniu kontroli aplikacji kontenerowi Springa.
- Spring Boot zarządza czasem życia obiektów - odpowiada za ich tworzenie, wykorzystanie i usuwanie.
- Spring Boot automatycznie konfiguruje zależności między serwisami, kontrolerami i repozytoriami.
- Kontrola aplikacji możliwa jest przy użyciu pliku *application.properties* bądź adnotacji za pomocą których Spring Boot automatycznie tworzy i implementuje konfigurację.

Inversion of Control (IoC) cd.

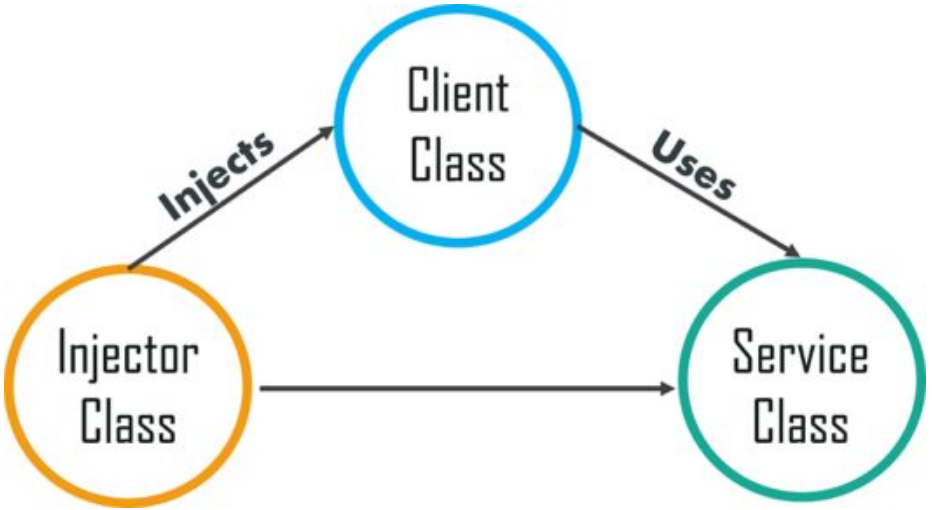
- Kontenerem bazowym IoC w Spring jest `BeanFactory`, odpowiadający za tworzenie beanów.
- Innym kontenerem jest `ApplicationContext` rozszerzający `BeanFactory`, obsługujący adnotacje, zdarzenia i transakcje.
- `WebApplicationContext` to kontener implementujący `ApplicationContext` zawierający funkcje niezbędne do obsługi warstwy aplikacji. Przechowuje beany odpowiedzialne za obsługę żądań sieciowych.

Struktura kontenera BeanFactory



Dependency injection

- Wstrzykiwanie zależności (DI) to technika pozwalająca dostarczyć obiektom wszelkie potrzebne im zależności bez konieczności definiowania ich wewnątrz obiektów
- Spring Boot automatycznie tworzy i zarządza beanami, które są zarejestrowane w kontenerze Springa



Dependency injection w Spring Boot - przykład

Najpopularniejszym rodzajem wstrzykiwania zależności w Spring Boot jest wstrzykiwanie poprzez konstruktor.

Pozostałe rodzaje wstrzykiwania zależności:

- Wstrzykiwanie poprzez interfejs
- Wstrzykiwanie poprzez setter

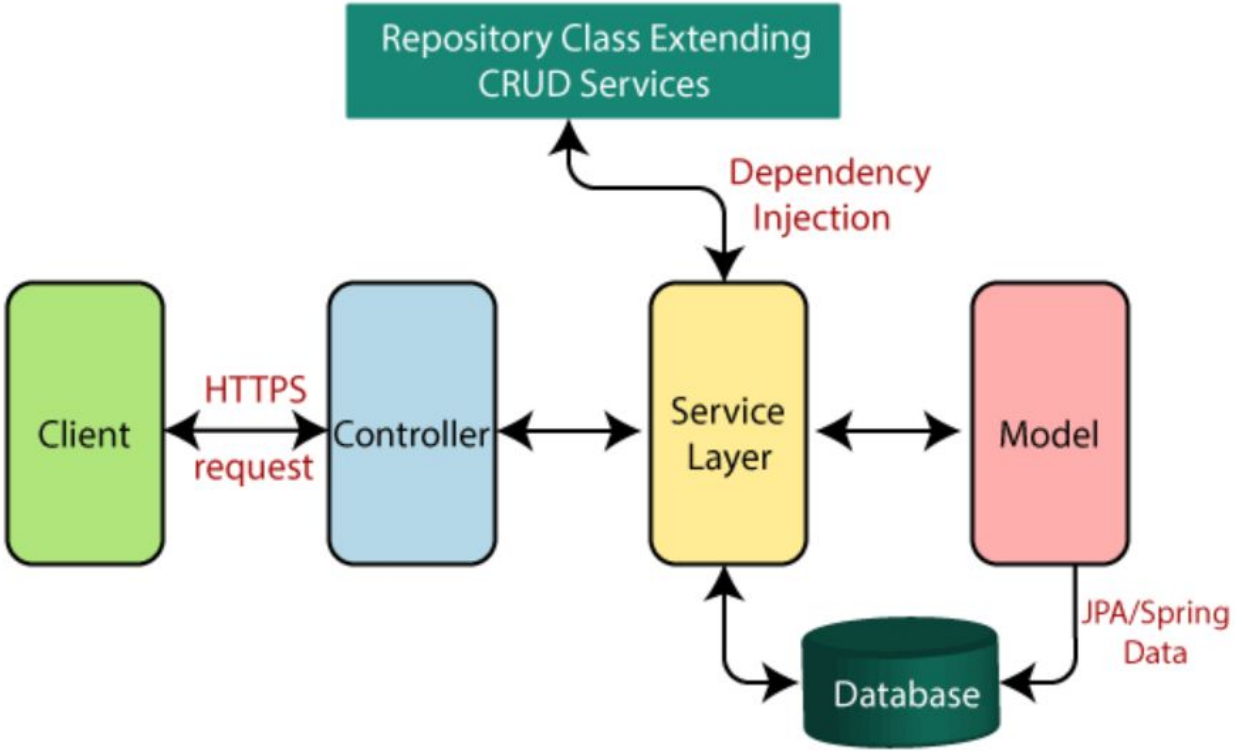
```
@Service
public class UserService {

    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository)
{
        this.userRepository = userRepository;
    }

    // metody UserService
}
```

Przepływ danych w aplikacji Spring Boot



Adnotacja @Configuration

`@Configuration` jest adnotacją używaną do zadeklarowania, że klasa udostępnia jedną lub więcej metod `@Bean` i może być przetwarzana przez kontener Spring IoC.

- Oznaczając klasę adnotacją `@Configuration` mówimy, że klasa będzie zawierać co najmniej jednego beana dla kontekstu aplikacji.
- W klasie oznaczonej tą adnotacją definiujemy metody z adnotacją `@Bean`, które produkują instancje beanów zarządzane przez kontener Springa.

Dla konfiguracji Spring Boot'a używa się wyspecjalizowanej wersji - `@SpringBootConfiguration`.

Adnotacja @Configuration

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserService userService;

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        final CorsConfiguration configuration = new CorsConfiguration();
        configuration.addAllowedOriginPattern("*");
        configuration.setAllowedMethods(ImmutableList.of("HEAD",
                "GET", "POST", "PUT", "DELETE", "PATCH"));
        configuration.setAllowCredentials(true);
        configuration.setAllowedHeaders(ImmutableList.of("Authorization", "Cache-Control", "Content-Type"));
        final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

Adnotacja `@ComponentScan`

`@ComponentScan` służy do włączania skanowania komponentów w poszukiwaniu beanów zarządzanych przez Spring w określonym pakiecie i jego podpakietach. Po napotkaniu takiej adnotacji, pakiet jest skanowany i rejestrowane są wszystkie klasy opatrzone adnotacją `@Component` lub jej stereotypami takimi jak:

- `@Controller`
- `@RestController`
- `@Service`
- `@Repository`

Adnotacja `@ComponentScan`

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = {"com.example.package1", "com.example.package2"})
public class MyApplicationConfig {
    // Configuration class content...
}

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmail(String email);
}
```

Adnotacja `@EnableAutoConfiguration`

`@EnableAutoConfiguration` uruchamia automatyczną konfigurację beanów na podstawie zależności obecnych w ścieżce classpath i konfiguracji aplikacji. Aby dokonać autokonfiguracji Spring Boot:

- Bada zasoby znalezione w classpath w celu rozpoznania zależności i bibliotek używanych w projekcie
- Konfiguruje kontekst Springa, włączając określone funkcje i beany. Konfiguruje beany odpowiedzialne za połączenie z bazą danych, serwery itp.
- Bierze pod uwagę jawnie dostarczone przez twórcę klasy konfiguracji i traktuje je nadrzędnie w stosunku do konfiguracji wygenerowanej automatycznie.

Adnotacja `@SpringBootApplication`

`@SpringBootApplication` jest złożeniem trzech adnotacji, wprowadzonym dla wygody użytkownika. Zazwyczaj opatruje się nią główną klasę aplikacji. `@SpringBootApplicaction` łączy ze sobą poniższe adnotacje:

- `@Configuration`
- `@EnableAutoConfiguration`
- `@ComponentScan`

Adnotacja `@SpringBootApplication`

```java
@SpringBootApplication
public class BackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(BackendApplication.class, args);
    }

    @EventListener(ApplicationReadyEvent.class)
    public void afterStartupActions() throws NoSuchAlgorithmException {
        new SecretKeyGenerator().generateSecretKey();
    }
}
```

Adnotacja `@Autowired`

`@Autowired` służy do automatycznego wstrzykiwania zależności. Mówi ona Spring'owi aby automatycznie podłączał beany określonych typów do zmiennych, metod lub konstruktorów oznaczonych tą adnotacją.

```
@CrossOrigin
@RestController
@RequestMapping("/room")
public class RoomController {
    @Autowired
    private RoomService roomService;

    @Autowired
    private UserService userService;
}
```