SPRING

Konfiguracja za pomocą XML

W celu zadeklarowania komponentu w XML należy skorzystać z tagu <bean />.
Posiada on wiele atrybutów, które pozwalają na wiele możliwości oraz modyfikacji. Dzięki tym atrybutom jesteśmy w stanie kontrolować inicjalizacje poszczególnych komponentów i zarządzać nimi.

Listing 3-30. chapter3/src/test/resources/musicservicetest.xml

```
<bean id="musicServiceTests" class="com.bsg5.chapter3.MusicServiceTests" />
```

Przykładowe atrybuty tagu <bean />

Atrybut	Znaczenie	Przykład
name	Nazwa tagu bean	<bean name=”Foo” />
class	Obowiązkowy atrybut nadający klasę tagu.	<bean class=”Class1” />
scope	Określa przestrzeń komponentu	<bean scope=”...” />
lazy-init	Inicjalizacja z opóźnieniem. Bean zostanie zainicjalizowany w przypadku wywołania.	<bean lazy-init=”true” />
depends-on	Lista komponentów beans, które muszą być zainicjalizowane przed danym komponentem	<bean depends-on="otherBeanId" />

<beans />

Tag <beans> służy do deklaracji kontenera na pozostałe komponenty typu bean

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="bar" class="com.bsg5.chapter3.mem03.SimpleNormalizer" />
    <bean id="foo" class="com.bsg5.chapter3.mem03.CapLeadingNormalizer" />
</beans>
```

<property /> - łączenie komponentów

Tag <property/> pozwala na utworzenie referencji do innego komponentu z wykorzystaniem atrybutu **ref**

```
<bean name="musicService" class="com.bsg5.chapter3.mem03.MusicService3">
    <property name="artistNormalizer" ref="foo" />
    <property name="songNormalizer" ref="bar" />
</bean>
```

Atrybut	Znaczenie
name	Nazwa tagu property
value	Wartość argumentu, który nie może współistnieć z argumentem ref
ref	To referencja do innego komponentu bean po nazwie

Konfiguracja i Deklaracja Beanów

W Spring istnieje możliwość zdefiniowania beana na kilka sposobów, poprzez:

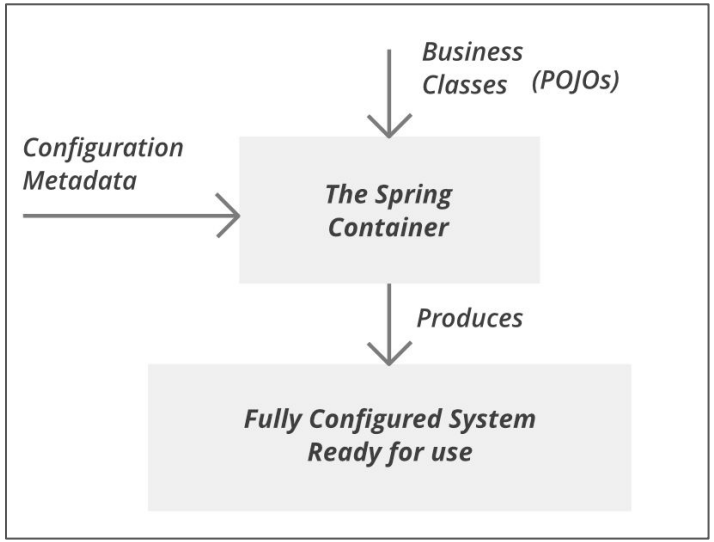
- Poprzez wykorzystanie adnotacji.
- Powoływanie ich instancji w metodach klasy konfiguracyjnej.
- Z wykorzystaniem konfiguracji XML.

Kontener Springa

Kontener Springa jest odpowiedzialny za tworzenie obiektów (beanów) oraz dostarczanie ich w odpowiednim momencie do danej klasy. Dotychczas aby utworzyć nowy obiekt niezbędne było użycie słowa kluczowego “new”:

```
ItemService itemService = new
ItemService();
```

Obecnie, korzystając z kontenera Springa, w ogóle nie musi się on martwić o wykonanie tej operacji, ponieważ Spring zrobi to niejawnie za niego. aby uzyskać referencje do obiektu tworzonego przez kontener należy “wstrzyknąć” go do naszej klasy poprzez **konstruktor**, **pole** (field) lub **metodę set** (setter). Wiązania tego typu w zdecydowanej większości przypadków oznaczamy adnotacją **@Autowired**



Definiowanie Beana Za Pomocą Adnotacji

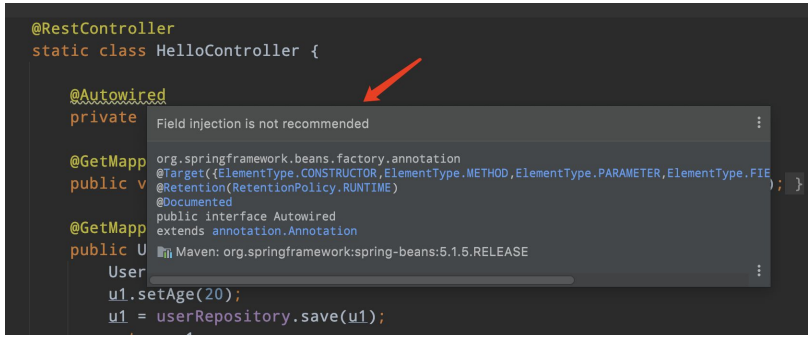
Istnieje szereg **adnotacji** do tworzenia beana. Każda ma swoje przeznaczenie. Do najpopularniejszych adnotacji należą:

- **@Component** – najbardziej ogólna adnotacja, może być wykorzystywany w ramach definiowania beanów DTO.
- **@Repository** – adnotacja dedykowana dla klas, których zadaniem jest przechowywanie, agregowane danych.
- **@Service** – adnotacja sugerowana dla klas, które dostarczają usługi.
- **@Controller/@RestController** – adnotacja przeznaczona dla warstwy prezentacji lub/i dla API aplikacji.

Adnotacje należy dobierać w zależności od odpowiedzialności klasy.

Wstrzykiwanie Zależności w Pole Klasy

ustawiamy adnotację nad polem, co jest nieco rzadziej używane, ze względu na pewne utrudnienie napotykane podczas tworzenia testów. Ciężko jest bowiem utworzyć obiekt klasy w przypadku gdy chcemy zainicjować go od razu naszą własną implementacją intrerfejsu, jeśli konstruktor nie zawiera opcji podania takiego parametru (zakładamy również, że w teście z jakiegoś powodu nie możemy zainicjować kontekstu Springa, co uniemożliwia użycie adnotacji @Autowired):



Jak widać na załączonym rysunku, nawet IDEE nie zaleca wstrzykiwania zależności do pól

Wstrzykiwanie Zależności w Konstruktor Klasy

```java
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import com.javadevsguide.springframework.di.service.MessageService;

@Component
public class ConstructorBasedInjection {
    private MessageService messageService;

    @Autowired
    public ConstructorBasedInjection(@Qualifier("TwitterService")
        MessageService messageService) {
        super();
        this.messageService = messageService;
    }

    public void processMsg(String message) {
        messageService.sendMsg(message);
    }
}
```

Oprócz wstrzykiwania zależności do pól klasy, istnieje również pojęcie wstrzykiwania zależności w konstruktor klasy. Ta metoda ma parę zalet zalet, najważniejszą z nich jest możliwość używania pól oznaczonych jako “final”.

w tym przypadku dodajemy adnotację nad konstruktorem i w ten sposób wstrzykujemy obiekt jako parametr (od wersji 4.3 Spring umożliwia w ogóle pominięcie adnotacji @Autowired, jednak tylko w przypadku gdy klasa posiada zdefiniowany tylko jeden konstruktor):

Wstrzykiwanie Zależności w Setter

Dzięki wstrzyknięciu zależności w setter możemy zmieniać zależności w dowolnym czasie, jednak może się to przerodzić w coś niepożądanego, szczególnie w wielowątkowym środowisku. Może to prowadzić do niespójnej klasy, bo nic nie wymusza żeby tą zależność podać. Aby to osiągnąć trzeba by utworzyć metodę, która sprawdzałaby przed każdym użyciem zależnej funkcji czy ta zależność została ustawiona.

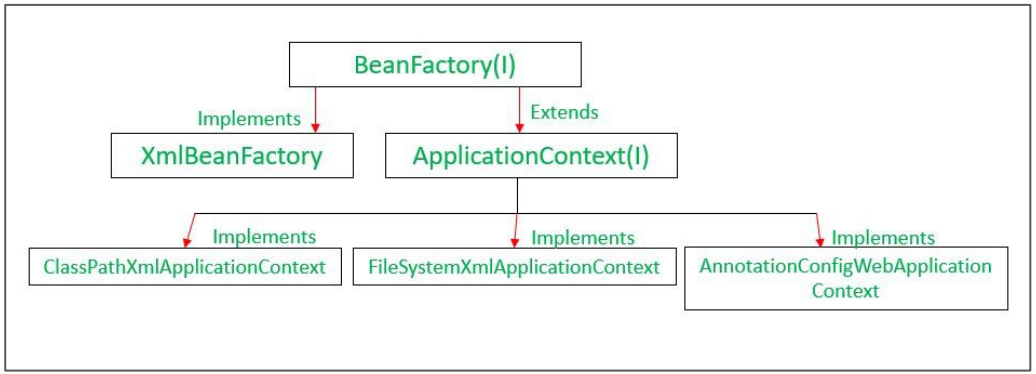
```
@Controller
public class UserController {

    private UserService userService;

    @Autowired
    public void setUserService(UserService userService){
        this.userService = userService;
    }
}
```

Konfiguracja poprzez adnotacje

Konfiguracja przy użyciu adnotacji Springa jest prosta. Trzeba powiedzieć Springowi, że używamy adnotacji, np. za pomocą XML. Obiekty muszą być zarządzane przez kontener, aby adnotacje miały znaczenie. Są dwie kategorie adnotacji: deklaracja komponentu i wtyczki, które są wstrzykiwane przez kontener.

Deklaracja Spring Bean z @Component

W pliku konfiguracji Spring jest jedna ważna linia: <context:component-scan basepackage=... />. Oznacza to, że skanowane będą pakiety i inne pod nim, szukając beanów oznaczonych odpowiednimi adnotacjami, takimi jak @Component, @Service, @Repository i inne. Beany te zostaną potem zarejestrowane w ApplicationContext i będą mogły być używane przez inne składniki Spring zarządzane przez ApplicationContext.
Aby to działało, muszą być utworzone dwie kolejne klasy: jedna implementująca interfejs z adnotacją @Component i druga abstrakcyjna, która będzie ją rozszerzać.



Łączenie Komponentów z użyciem @Autowired

Ta konfiguracja dodaje dwa komponenty i łączy je ze sobą. Tworzymy nową usługę MusicService2, która przesłania transformArtist() i transformSong() w taki sposób, że wywołują wstrzykniętą instancję Normalizera. Tworzymy konkretny typ, który ma służyć jako Normalizator i dodajemy kolejny plik konfiguracyjny oraz dwie dodatkowe klasy: jedną dla testów podstawowych i drugą dla testu wykonywalnego.Normalizer to konkretna realizacja interfejsu, która może być oznaczona jako @Component i tworzona. Można go użyć do obcięcia białych znaków lub zapewnić więcej funkcjonalności.

```
    @Override
    protected String transformArtist(String input) {
        return normalizer.transform(input);
    }

    @Override
    protected String transformSong(String input) {
        return normalizer.transform(input);
    }
}
```

```
@Component
public class MusicService2 extends AbstractMusicService {
    @Autowired
    Normalizer normalizer;
```

Wybór Komponentów z Adnotacją @Qualifier

Wcześniej normalizowaliśmy nazwy piosenek oraz nazwiska artystów, ale co jeśli te dwa aspekty nie powinny być normalizowane w ten sam sposób? Wtedy należy użyć różnych rodzajów normalizerów. Musimy jednak powiedzieć Springowi którego normalizera używać w danym kontekście.

Możemy stworzyć dedykowane interfejsy, np. SongNameNormalizer, i używać go w taki sposób:

@Autowired
SongNameNormalizer songNameNormalizer.

Można również z opcjonalnej wartości dla @Component, która będzie wskazywać jego nazwę, np. @Component(“bluePin”).

Pokażemy to na przykładzie 2 normalizerów, nadając komponentom nazwy “foo” oraz “bar”. Pierwszy będzie tylko usuwał białe znaki, drugi będzie zamieniał pierwsze litery słów na wielkie.

```
@Component("bar")
public class CapLeadingNormalizer implements Normalizer {
    @Override
    public String transform(String input) {
        StringJoiner joiner = new StringJoiner(" ");
        Stream
                .of(input.trim().split("\\s"))
                .filter(s -> !s.isBlank())
                .map(s ->
                        Character.toUpperCase(s.charAt(0)) +
                                s.substring(1).toLowerCase()
                )
                .forEach(joiner::add);
        return joiner.toString();
    }
}
```

SimpleNormalizer.java

```
package com.bsg5.chapter3.mem03;

import com.bsg5.chapter3.Normalizer;
import org.springframework.stereotype.Component;

@Component("foo")
public class SimpleNormalizer implements Normalizer {
}
```

Konfiguracja za pomocą języka Java

Spring umożliwia tworzenie konfiguracji za pomocą języka Java za pomocą dwóch sposobów:

- (statyczna) poprzez stworzenie klasy opatrzonej adnotacją @Configuration, w której tworzy się metody opatrzone adnotacją @Bean

- (dynamiczna) wykorzystanie ApplicationContext i ręczne zarejestrowanie komponentu Bardziej popularną metodą konfiguracji jest metoda statyczna

Wstrzykiwanie zależności - statyczna konfiguracja

Wstrzykiwanie zależności:

● wykorzystując adnotację @Autowired jest możliwe wstrzyknięcie w pole beanu, dla którego metoda rejestrująca została zawarta w klasie konfiguracyjnej

● wykorzystując konstruktor klasy w której znajduje się pole do, którego Spring wstrzykuje Adnotacja @Qualifier w klasie konfiguracyjnej, umożliwia rozróżnienie po nazwie metody, który bean ma zostać wybrany przez Spring.

Testowanie konfiguracji z DataProvider

● ma na celu czy konfiguracja pochodzi z klasy opatrzonej metodą @Configuration lub z pliku XML, jeśli nie to jest rzucany wyjątek

● sprawdzanie każdej konfiguracji, dzięki metodzie data provider

● metoda data providera zawiera listę referencji do do każdej konfiguracji do testowania

Konfiguracja za pomocą XML wczytuje się wolniej od konfiguracji z wykorzystaniem klas języka Java. Różnica jest zauważalna tylko przy rejestrowaniu beanów w kontekście Springa.

Podsumowanie

W powyższej prezentacji przedstawiono elementy frameworku Spirint pozwalające na deklarowanie w różny sposób komponentów typu bean. Przedstawiono definicję poprzez adnotację, wywoływanie z instancji metody klasy konfiguracyjnej oraz poprzez konfigurację z wykorzystaniem XML.

Ponadto, przedstawiono sposoby wstrzykiwania zależności min. w:

- pole klasy
- konstruktor klasy
- w setter

oraz łączenie komponentów z użyciem adnotacji

Dalsze kroki…

W kolejnym rozdziale omawiane będą opcje cyklu życia Spring'a, w których będziemuy mieć większą kontrolę nad sposobem tworzenia komponentów bean (poprzez konstruktory, analogicznie do tych przedstawionych w wyżej przedstawionym rozdziale 3),. Ukazane zostaną ponadto sposoby w jakie wywołujemy metody, gdy obiekty Spring Bean
są tworzone lub niszczone. Zobaczymy również, jak tworzyć nowe Spring Bean'y, nie będące obiektami singletonowymi.

Dziękujemy za uwagę

Krzysztof Szczęśniak
Andrzej woźniacki
Aleksander Radwan-Pragłowski
Mateusz Klimczyk
Mateusz Nawrocki
Paweł Pytlowski