

TWÓJ PRZEWODNIK PO SPRINGU!

Apress

Wprowadzenie do **Spring** **Framework** dla programistów Java

Felipe Gutierrez

Helion 

Tytuł oryginału: Introducing Spring Framework: A Primer

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-0439-0

Original edition copyright © 2014 by Felipe Gutierrez.
All rights reserved.

Polish edition copyright © 2015 by HELION SA.
All rights reserved.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/wprsfj.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/wprsfj>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

	O autorze	11
	O korektorze merytorycznym	12
	Wprowadzenie	13
Część I	Podstawy systemu szkieletowego Spring	15
Rozdział 1.	Pierwsza aplikacja Spring	17
	Wymagania wstępne	18
	Witaj, świecie	18
	Podsumowanie	25
Rozdział 2.	Klasy i zależności	27
	Aplikacja Spring: Moje dokumenty	27
	Wymagania do programu Moje dokumenty	27
	Definiowanie klas i zależności	28
	Praca z systemem Spring	32
	System szkieletowy Spring a wstrzykiwanie zależności	34
	Podsumowanie	36
Rozdział 3.	Stosowanie różnych konfiguracji	37
	Konfiguracja XML programu Moje dokumenty	37
	Używanie adnotacji Spring	44
	Stereotypy Spring	47
	Konfiguracja w postaci klasy ziarna Java	48
	Użycie klasy GroovyBeanDefinitionReader	51
	Który rodzaj konfiguracji wybrać	53
	Podsumowanie	54
Rozdział 4.	Zakresy ziaren	55
	Zakresy	55
	Adnotacja @Scope	58
	Zakresy ziaren w klasie GroovyBeanDefinitionReader	60
	Podsumowanie	61

Rozdział 5.	Kolekcje i typy własne	63
	Listy, słowniki i własności	63
	Kolekcje w konfiguracji XML	65
	Scalanie kolekcji	67
	Podsumowanie	69
Rozdział 6.	Pliki zasobów	71
	Pliki własności	76
	Używanie innego języka — czy mówisz po hiszpańsku?	79
	Podsumowanie	82
Rozdział 7.	Testowanie aplikacji Spring	83
	Testowanie przy użyciu adnotacji	83
	Profile	85
	Inne adnotacje testowe	90
	Podsumowanie	92
Część II	System szkieletowy Spring	93
Rozdział 8.	Rady dla aplikacji Spring	95
	Rady dla aplikacji Moje dokumenty	95
	Na ratunek programowanie aspektowe	96
	Podstawowe pojęcia programowania aspektowego	97
	Rada before	101
	Rada after	103
	Rada around	105
	Rada AfterThrowing	107
	Zastosowanie technik AOP z adnotacjami	111
	Podsumowanie	113
Rozdział 9.	Przechowywanie danych aplikacji Spring	115
	Dodawanie mechanizmu przechowywania danych	115
	Osadzanie bazy danych	128
	Nowa metoda gromadzenia danych — JdbcTemplate i RowMapper	130
	Podsumowanie	134
Rozdział 10.	Publikowanie aplikacji Spring w internecie	135
	Warstwa przechowywania danych	135
	Wzorzec MVC w systemie Spring	137
	Internacjonalizacja	142
	Podsumowanie	145
Rozdział 11.	Integracja aplikacji Spring z systemami zewnętrznymi	147
	Java Message Service	148
	Spring JMS	148
	RabbitMQ i Spring Rabbit	158
	Podsumowanie	165
Rozdział 12.	Udostępnianie API typu REST	167
	Interfejs API typu RESTful	167
	HTTP GET	174
	HTTP POST	175

HTTP PUT	176
HTTP DELETE	177
Podsumowanie	178
Rozdział 13. Zadania e-mail i planowanie zadań	179
Wysyłanie wiadomości e-mail	179
Zadania asynchroniczne	182
Planowanie zadań	185
Podsumowanie	187
Część III Zaawansowane techniki programowania przy użyciu systemu szkieletowego Spring	189
Rozdział 14. Używanie dynamicznych języków programowania	191
Bądź dynamiczny	191
Zastosowanie języka Groovy	192
Zastosowanie języków JRuby i BeanShell	196
Podsumowanie	198
Rozdział 15. Dane Spring w aplikacjach Spring	199
Bazy danych NoSQL	199
Implementacja klasy DocumentDAO	202
Testowanie MongoDB	205
Kompletny test DocumentDAO	207
Podsumowanie	210
Rozdział 16. Przesyłanie wiadomości w aplikacji Spring	211
Zastosowanie programu RabbitMQ	211
RabbitMQ — wymiany, powiązania i kolejki	212
Zastosowanie marshallera XML do przekształcania wiadomości	214
<rabbit:connection-factory/>	218
<rabbit:template/>	219
<rabbit:direct-exchange/> i <rabbit:binding/>	219
<rabbit:queue/>	219
<rabbit:listener-container/> i <rabbit:listener/>	219
Testowanie	219
Podsumowanie	221
Rozdział 17. Media społecznościowe i mobilne	223
Moduł Spring Social	223
Spring Social Twitter	223
Rejestrowanie aplikacji w Twitterze	224
Załącznik	231
Wysyłanie tweetów z aplikacji Spring	234
Podsumowanie	236

Część IV	Nowy system wejścia-wyjścia Spring	237
Rozdział 18.	Spring i Groovy	239
	Napiszmy coś w języku Groovy	239
	Testowanie kodu w języku Groovy	243
	Składnia języka DSL	244
	Podsumowanie	247
Rozdział 19.	Upraszczenie wszystkiego przy użyciu Spring Boot	249
	Spring Boot	249
	Wdrażanie aplikacji	253
	Tworzenie wykonywalnego pliku JAR	253
	Tworzenie pliku WAR	254
	Spring Boot i Groovy	256
	Podsumowanie	260
Rozdział 20.	Pierwsza aplikacja Spring XD	261
	Instalowanie modułu Spring XD	261
	Archiwum ZIP z modulem Spring XD	261
	Instalacja w OSX przy użyciu menedżera Homebrew	261
	Spring XD	262
	Moduły	262
	Podsluchy	262
	Zastosowanie Spring XD w aplikacji Moje dokumenty	263
	Analiza	266
	Podsumowanie	269
Dodatki		271
Dodatek A	Instalacja narzędzi	273
	Instalacja Javy	273
	Instalacja Javy w systemie Windows	274
	Ustawianie zmiennych środowiskowych	276
	Instalowanie Javy w systemie OS X	279
	Narzędzia dla systemu Mac OS X	281
	Instalacja programu Homebrew	281
	Instalacja GVM	282
	Instalacja Gradle	282
	Instalacja programu Gradle w systemie Windows	283
	Instalacja Gradle w systemach Mac OS X i Linux	284
	Instalacja interpretera języka Groovy	285
	Instalacja Groovy w systemie Windows	285
	Instalacja Groovy w systemach Mac OS X i Linux	287
	Instalacja MongoDB	287
	Instalacja MongoDB w systemie Windows	287
	Uruchamianie serwera MongoDB	288
	Zatrzymywanie serwera MongoDB	289
	Instalacja MongoDB w systemie Mac OS X	289
	Uruchamianie serwera MongoDB	289
	Zatrzymywanie serwera MongoDB	289
	Uruchamianie serwera MongoDB przy logowaniu	289

Instalacja brokera Apache Active MQ	290
Instalacja programu Apache Active MQ w systemie Windows	290
Uruchamianie programu ActiveMQ	290
Zatrzymywanie programu ActiveMQ	290
Otwieranie konsoli sieciowej hawtio programu ActiveMQ	291
Instalacja programu ActiveMQ w systemie Mac OS X	291
Uruchamianie programu ActiveMQ	291
Zatrzymywanie brokera ActiveMQ	291
Instalacja programu RabbitMQ	292
Instalacja programu RabbitMQ w systemie Windows	292
Instalacja programu RabbitMQ w systemie Mac OS X	293
Przydatne polecenia do obsługi programu RabbitMQ	293
Instalacja systemu Spring Boot	294
Instalacja Spring Boot w systemie Windows	294
Instalacja Spring Boot w systemach Mac OS X i Linux	295
Instalacja Spring XD	295
Instalacja Spring XD w systemie Windows	295
Instalacja Spring XD w systemie Mac OS X	296
Podsumowanie	296
Skorowidz	297

ROZDZIAŁ 3



Stosowanie różnych konfiguracji

System szkieletowy Spring umożliwia stosowanie różnych technik konfiguracji kontenera. W tym rozdziale znajduje się opis metody opartej na formacie XML, z której skorzystaliśmy w poprzednim rozdziale. Ponadto czytając ten rozdział, poznasz inne techniki obsługi konfiguracji, takie jak adnotacje Spring, klasy konfiguracyjne ziaren Java oraz nowa klasa `GroovyBeanDefinitionReader`.

W poprzednim rozdziale zdefiniowaliśmy aplikację Spring o nazwie *Moje dokumenty*, a także za pomocą pliku konfiguracyjnego XML wstrzyknęliśmy naszą implementację interfejsu `SearchEngine`. W tym rozdziale będziemy używać tej samej konfiguracji XML, dowiesz się również, jak używać pozostałych technik konfiguracji.

Konfiguracja XML programu *Moje dokumenty*

Na początek przypomnimy sobie aktualnie posiadaną konfigurację XML aplikacji *Moje dokumenty* i dokładnie ją przeanalizujemy (listing 3.1).

Listing 3.1. Zawartość pliku *mydocuments-context.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<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="engine" class="com.apress.isf.java.service.MySearchEngine" />

    <bean id="documentType" class="com.apress.isf.java.model.Type">
        <property name="name" value="WEB" />
        <property name="desc" value="Łącze sieciowe" />
        <property name="extension" value=".url" />
    </bean>

</beans>
```

Na listingu 3.1 pokazano treść pliku konfiguracyjnego w formacie XML zawierającego potrzebne kontenerowi Spring informacje na temat klas i ich zależności. W pliku tym informujemy kontener, że nasza implementacja interfejsu `SearchEngine` będzie ziarnem `MySearchEngine` o identyfikatorze `engine`. Innymi słowy, przypisaliśmy identyfikator zdefiniowanemu przez nas ziarnu. Ponadto stworzymy egzemplarz klasy `Type` w postaci ziarna o identyfikatorze `documentType`.

Ale chwileczkę! Ziarno? A co to jest ziarno? W Javie pojęcie ziarna (ang. *bean*) występuje od początku istnienia tego języka programowania i dlatego w systemie Spring przyjęto podobną nazwę. Ziarna Javy mają pewne standardowe cechy, np. znormalizowane nazwy metod (z przedrostkami `set`, `get` i `is`), odpowiednie konstruktory, zachowania itd., dzięki którym mogą współpracować z innymi ziarnami oraz klasami. Później w społeczności programistów Javy pojęcie ziarna przekształciło się w obiekt POJO (ang. *plain old java object*).

Dzięki tym konwencjom system szkieletowy Spring rozpoznaje, tworzy, wstrzykuje, obsługuje, a nawet niszczy wszystkie klasy zadeklarowane na kontenerze.

Ziarna deklaruje się przy użyciu elementu `<bean/>` w pliku XML. Mogą one mieć kilka atrybutów, opisanych w tabeli 3.1.

Tabela 3.1. Atrybuty elementu *bean*

Atrybut	Opis
<code>id</code>	Identyfikator ziarna. Każdy identyfikator może być zdefiniowany tylko raz.
<code>class</code>	Wskazuje pełną nazwę wraz z nazwą pakietu konkretnej klasy.
<code>scope</code>	Informuje kontener Spring o sposobie tworzenia ziarna. Jeśli atrybut ten jest niezdefiniowany, ziarno jest domyślnie egzemplarzem singletonowym. Poza tym można ustawić zakres <code>prototype</code> (dla każdego żądania ziarna tworzony jest egzemplarz), <code>request</code> (dla każdego żądania sieciowego HTTP tworzony jest pojedynczy egzemplarz) oraz <code>session</code> (ziarno jest tworzone i istnieje przez czas trwania sesji HTTP).
<code>init-method</code>	Nazwa metody wywoływanej po utworzeniu ziarna. Metoda ta jest przydatna, gdy trzeba ustawić stan po utworzeniu obiektu.
<code>factory-method</code>	Nazwa metody służącej do tworzenia ziarna. Innymi słowy, programista powinien dostarczyć metodę tworzącą egzemplarz obiektu. Metoda ta powinna mieć parametry.
<code>destroy-method</code>	Nazwa metody wywoływanej po usunięciu ziarna.
<code>lazy-init</code>	Parametr ten należy ustawić na wartość <code>true</code> , jeśli chcemy, aby kontener tworzył ziarno przy jego wywołaniu lub użyciu przez programistę (poprzez wywołanie metody <code>getBean()</code> bądź później w innej klasie wymagającej tego obiektu).

W systemie Spring informacje na temat klas i ich zależności oraz sposobów interakcji między nimi można dodawać na różne sposoby. O tym wszystkim opowiem w tej książce przy okazji omawiania dodawania różnych funkcji do aplikacji *Moje dokumenty*.

Na listingu 3.2 pokazano implementację interfejsu `SearchEngine` o nazwie `MySearchEngine` z poprzedniego rozdziału. Wydaje się, że to dużo kodu, ale zawiera on sporo danych wpisanych na stałe. Jak w takim razie sobie poradzić, gdy trzeba będzie dodać więcej typów lub metod? W takim przypadku konieczne jest zmodyfikowanie i ponowne skompilowanie kodu. To za dużo pracy!

Listing 3.2. Zawartość pliku *MySearchEngine.java*

```
package com.apress.isf.java.service;

import java.util.ArrayList;
import java.util.List;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;

public class MySearchEngine implements SearchEngine {

    @Override
    public List<Document> findByType(Type documentType) {
```

```

List<Document> result = new ArrayList<Document>();
for(Document document : storage()){
    if(document.getType().getName().equals(documentType.getName()))
        result.add(document);
}
return result;
}

@Override
public List<Document> listAll() {
    return storage();
}

private List<Document> storage(){
    List<Document> result = new ArrayList<Document>();

    Type type = new Type();
    type.setName("PDF");
    type.setDesc("Portable Document Format");
    type.setExtension(".pdf");

    Document document = new Document();
    document.setName("Szablon książki");
    document.setType(type);
    document.setLocation("/Users/felipeg/Documents/Random/Book Template.pdf");

    result.add(document);

    document = new Document();
    document.setName("Przykładowa umowa");
    document.setType(type);
    document.setLocation("/Users/felipeg/Documents/Contracts/Sample Contract.pdf");

    result.add(document);

    type = new Type();
    type.setName("NOTE");
    type.setDesc("Notatki tekstowe");
    type.setExtension(".txt");

    document = new Document();
    document.setName("Clustering with RabbitMQ");
    document.setType(type);
    document.setLocation("/Users/felipeg/Documents/Random/Clustering with RabbitMQ.txt");

    result.add(document);
    type = new Type();
    type.setName("WEB");
    type.setDesc("Łącze sieciowe");
    type.setExtension(".url");

    document = new Document();
    document.setName("Pro Spring Security Book");
    document.setType(type);
    document.setLocation("http://www.apress.com/9781430248187");

    result.add(document);

```

```

        return result;
    }
}

```

Aby wyeliminować opisane niedogodności, zaimplementujemy interfejs `SearchEngine` od nowa. Do tej implementacji wstrzykniemy typy za pomocą metody ustawiającej. Na listingu 3.3 znajduje się nowa klasa, `SearchEngineService`.

Listing 3.3. Zawartość pliku *SearchEngineService.java*

```

package com.apress.isf.spring.service;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;
import com.apress.isf.spring.data.DocumentDAO;

public class SearchEngineService implements SearchEngine {

    private DocumentDAO documentDAO;

    public DocumentDAO getDocumentDAO() {
        return documentDAO;
    }

    public void setDocumentDAO(DocumentDAO documentDAO) {
        this.documentDAO = documentDAO;
    }

    public List<Document> findByType(Type documentType) {
        List<Document> result = new ArrayList<Document>();
        for(Document doc : listAll()){
            if(doc.getType().getName().equals(documentType.getName()))
                result.add(doc);
        }
        return result;
    }

    public List<Document> listAll() {
        return Arrays.asList(documentDAO.getAll());
    }

}

```

Porównamy kod znajdujący się na listingu 3.2 z kodem z listingu 3.3. Na tym ostatnim brak sztywnych fragmentów kodu i metody `storage`, natomiast dodano nowy atrybut o nazwie `documentDAO`, który będzie wstrzykiwany przez metodę ustawiającą, tylko trzeba poinformować kontener Spring o tym nowym atrybucie i klasie zawierającej dane.

Teraz zdefiniujemy nowe klasy: `DocumentDAO` i `DocumentRepository` (listingi 3.4 i 3.5). Pokazana na listingu 3.4 klasa `DocumentDAO` przechowuje wszystkie informacje na temat dokumentów, które na razie będą zapisywane w pamięci. Natomiast widoczna na listingu 3.5 klasa `DocumentRepository` to jej implementacja.

Listing 3.4. Zawartość pliku *DocumentDAO.java*

```
package com.apress.isf.spring.data;

import com.apress.isf.java.model.Document;

public interface DocumentDAO {
    public Document[] getAll();
}
```

Na listingu 3.5 przedstawiono klasę `DocumentRepository` zawierającą cztery własności typu `Document` z własnymi metodami ustawiającymi i pobierającymi. Tak, będziemy wstrzykiwać cztery dokumenty za pomocą ich metod ustawiających.

Listing 3.5. Zawartość pliku *DocumentRepository.java*

```
package com.apress.isf.spring.data;

import com.apress.isf.java.model.Document;

public class DocumentRepository implements DocumentDAO {

    private Document doc1;
    private Document doc2;
    private Document doc3;
    private Document doc4;

    public Document getDoc1() {
        return doc1;
    }

    public void setDoc1(Document doc1) {
        this.doc1 = doc1;
    }

    public Document getDoc2() {
        return doc2;
    }

    public void setDoc2(Document doc2) {
        this.doc2 = doc2;
    }

    public Document getDoc3() {
        return doc3;
    }

    public void setDoc3(Document doc3) {
        this.doc3 = doc3;
    }

    public Document getDoc4() {
        return doc4;
    }

    public void setDoc4(Document doc4) {
        this.doc4 = doc4;
    }
}
```

```

    }

    public Document[] getAll() {
        return new Document[] { doc1, doc2, doc3, doc4 };
    }
}

```

To wygląda już trochę lepiej. Oddzieliliśmy metodę wydobywania danych, ale jak to robimy? Przypomnę, że wcześniej do pobierania informacji używaliśmy metody `storage` (listing 3.2). Później postanowiliśmy od nowa zaimplementować interfejs `SearchEngine`. W tej nowej implementacji zmieniliśmy sposób pobierania danych i utworzyliśmy interfejs, który będzie wstrzykiwany niezależnie od swojej implementacji, dzięki czemu klasa stanie się solidniejsza i łatwiejsza w obsłudze. Ale zobaczmy teraz, co trzeba zmienić w pliku XML, aby poinformować kontener Spring o tych wszystkich nowych modyfikacjach. Nowa wersja pliku `mydocuments-context.xml` jest przedstawiona na listingu 3.6. Znajdują się w nim wszystkie informacje dotyczące implementacji klasy `DocumenDAO` (`DocumentRepository`) oraz sposobu jej wstrzykiwania do implementacji `SearchEngine`.

Listing 3.6. Zawartość pliku `mydocuments-context.xml`

```

<?xml version="1.0" encoding="UTF-8"?>
<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="engine" class="com.apress.isf.spring.service.ServiceSearchEngine">
        <property name="documentDAO" ref="documentDAO"/>
    </bean>
    <bean id="documentDAO" class="com.apress.isf.spring.data.DocumentRepository">
        <property name="doc1" ref="doc1"/>
        <property name="doc2" ref="doc2"/>
        <property name="doc3" ref="doc3"/>
        <property name="doc4" ref="doc4"/>
    </bean>
    <bean id="doc1" class="com.apress.isf.java.model.Document">
        <property name="name" value="Szablon książki"/>
        <property name="type" ref="pdfType"/>
        <property name="location" value="/Users/felipeg/Documents/Random/Book Template.pdf"/>
    </bean>
    <bean id="doc2" class="com.apress.isf.java.model.Document">
        <property name="name" value="Przykładowa umowa"/>
        <property name="type">
            <bean id="pdfType" class="com.apress.isf.java.model.Type">
                <property name="name" value="PDF" />
                <property name="desc" value="Portable Document Format" />
                <property name="extension" value=".pdf" />
            </bean>
        </property>
        <property name="location" value="/Users/felipeg/Documents/Contracts/Sample Contract.pdf"/>
    </bean>
    <bean id="doc3" class="com.apress.isf.java.model.Document">
        <property name="name" value="Clustering with RabbitMQ"/>
        <property name="type" ref="noteType"/>
        <property name="location" value="/Users/felipeg/Documents/Random/Clustering with
RabbitMQ.txt"/>
    </bean>
    <bean id="doc4" class="com.apress.isf.java.model.Document">
        <property name="name" value="Pro Spring Security Book"/>
        <property name="type" ref="webType"/>
    </bean>

```

```

    <property name="location" value="http://www.apress.com/9781430248187"/>
  </bean>
  <bean id="webType" class="com.apress.isf.java.model.Type">
    <property name="name" value="WEB" />
    <property name="desc" value="Łącze sieciowe" />
    <property name="extension" value=".url" />
  </bean>
  <bean id="pdfType" class="com.apress.isf.java.model.Type">
    <property name="name" value="PDF" />
    <property name="desc" value="Portable Document Format" />
    <property name="extension" value=".pdf" />
  </bean>
  <bean id="noteType" class="com.apress.isf.java.model.Type">
    <property name="name" value="NOTE" />
    <property name="desc" value="Notatki tekstowe" />
    <property name="extension" value=".txt" />
  </bean>
</beans>

```

Analizując kod przedstawiony na listingu 3.6, można zauważyć, że do przypisywania wartości użyto referencji, takich jak atrybut `ref`. Ponadto w kodzie znajduje się nowa deklaracja klasy `ServiceSearchEngine`, ustawiono także własność `documentDao` i odniesiono jego wartość do innego ziarna o identyfikatorze `documentDAO`.

Spójrz też na ziarno o identyfikatorze `doc2`. Osadzamy nowe ziarno jako wartość, co jest zgodne z zasadami konfiguracji Spring.

Jak widać, wszystkie dane dotyczące typów i dokumentów zostały umieszczone w pliku XML. Może istnieje jeszcze lepszy sposób, ale na razie zajmijmy się utworzeniem testu jednostkowego. Na listingu 3.7 pokazano zmodyfikowaną wersję naszego testu.

Listing 3.7. Zawartość pliku *MyDocumentsTest.java*

```

package com.apress.isf.spring.test;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;

public class MyDocumentsTest {

    private ClassPathXmlApplicationContext context;
    private SearchEngine engine;
    private Type webType;

    @Before
    public void setup(){
        context = new ClassPathXmlApplicationContext("META-INF/spring/mydocuments-context.xml");
        engine = context.getBean(SearchEngine.class);
        webType = context.getBean("webType",Type.class);
    }
}

```

```

@Test
public void testWithSpringFindByType() {
    List<Document> documents = engine.findByType(webType);
    assertNotNull(documents);
    assertTrue(documents.size() == 1);
    assertEquals(webType.getName(), documents.get(0).getType().getName());
    assertEquals(webType.getDesc(), documents.get(0).getType().getDesc());

    assertEquals(webType.getExtension(), documents.get(0).getType().getExtension());
}

@Test
public void testWithSpringListAll() {
    List<Document> documents = engine.listAll();
    assertNotNull(documents);
    assertTrue(documents.size() == 4);
}
}

```

W metodzie `setup()` (metoda ta jest uruchamiana przed wykonaniem każdej metody w klasie) używamy klasy `ClassPathXmlApplicationContext`, która uruchamia kontener Spring przez utworzenie i powiązanie wszystkich egzemplarzy oraz przygotowanie ich do użytku w momencie, gdy będą potrzebne.

Teraz uruchom ten test za pomocą poniższego polecenia:

```
gradle test
```

W katalogu głównym projektu można użyć polecenia:

```
gradle :r03:test
```

■ **Uwaga** Każdy rozdział zawiera kilka plików z testami jednostkowymi, więc poniżej przedstawiam polecenie wykonujące jeden konkretny test:

```
gradle -Dtest.single=MyDocumentsTest test
```

Na razie pokazałem Ci, jak skonfigurować kontener Spring poprzez dodanie ziaren i utworzenie do nich odwołań, aby kontener wiedział, jak je tworzyć i jakie łączyć je relacje, oraz by mógł je przygotować, gdy będą potrzebne. Ale przypominę, że w systemie Spring konfigurację można tworzyć także innymi sposobami, i dlatego w następnym podrozdziale pokazuję, jak utworzyć taką samą konfigurację jak wcześniej przy użyciu adnotacji.

Używanie adnotacji Spring

Adnotacje do języka Java wprowadzono w jego wersji 5. Było to znakomite posunięcie ze strony twórców Javy, ponieważ za pomocą adnotacji do klas można dodawać metadane stosowane zarówno podczas kompilacji, jak i działania programu, co stwarza nowe możliwości dla programistów. Programiści systemu Spring postanowili wykorzystać tę okazję do budowy mechanizmu konfiguracyjnego opartego na adnotacjach. Mechanizm ten pojawił się w wersji 2.5 systemu.

Ale wystarczy tego gadania. Czas się wziąć za kod źródłowy i sprawdzić, co trzeba zmienić, aby zastosować konfigurację opartą na adnotacjach. Spójrz na listing 3.8.

Listing 3.8. Zawartość pliku `AnnotatedSearchEngine.java`

```

package com.apress.isf.spring.annotated.service;

import java.util.ArrayList;

```



```

import java.util.Arrays;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;
import com.apress.isf.spring.data.DocumentDAO;

@Service("engine")
public class AnnotatedSearchEngine implements SearchEngine {
    @Autowired
    private DocumentDAO documentDAO;

    public List<Document> findByType(Type documentType) {
        List<Document> result = new ArrayList<Document>();
        for(Document doc : listAll()){
            if(doc.getType().getName().equals(documentType.getName()))
                result.add(doc);
        }
        return result;
    }

    public List<Document> listAll() {
        return Arrays.asList(documentDAO.getAll());
    }
}

```

Na powyższym listingu przedstawiono nową implementację interfejsu `SearchEngine` w postaci klasy `AnnotatedSearchEngine`. Jak widać, zostały w niej użyte adnotacje `@Service("engine")` i `@Autowired`. System Spring obsługuje kilka adnotacji, które zostały wymienione w tabeli 3.2. Są one jedynie markerami albo stereotypami, podobnie jak adnotacja `@Service`. Adnotacja ta może mieć wartość. W tym przypadku jest to wartość `engine`, która oznacza, że kontener utworzy ziarno o identyfikatorze `engine`. W tym przykładzie wskazywana jest klasa `AnnotatedSearchEngine`. To tak samo, jakbyśmy w pliku XML wpisali poniższy element: `<bean id="engine" class="com.apress.isf.spring.annotated.service AnnotatedSearchEngine" />`

Tabela 3.2. Stereotypy

Stereotyp/marker	Opis
<code>@Component</code>	Jest to marker, ogólny stereotyp, który przez system Spring jest rozpoznawany jako zarządzany przez niego składnik.
<code>@Repository</code>	Jest to specjalizacja adnotacji <code>@Component</code> dotycząca obiektu obsługującego dostęp do danych. Klasy oznaczone tą adnotacją mogą być przetwarzane przez inne narzędzia, a nawet aspekty w kontenerze Spring.
<code>@Service</code>	Jest to specjalizacja adnotacji <code>@Component</code> dotycząca warstwy usługowej.
<code>@Controller</code>	To również jest specjalizacja adnotacji <code>@Component</code> , której zazwyczaj używa się w kontekście sieciowym.

Zastosowaliśmy też adnotację `@Autowired`. Nakazuje ona systemowi Spring utworzenie egzemplarza i przypisanie go do zadeklarowanej zmiennej. Ma takie samo znaczenie jak poniższy element XML:

```
<property name="documentDAO" ref="documentDAO" />
```

Podsumowując, konfiguracja klasy `AnnotatedSearchEngine` będzie wyglądała następująco:

```
<bean id="engine" class="com.apress.isf.spring.annotated.service AnnotatedSearchEngine">
    <property name="documentDAO" ref="documentDAO" />
</bean>
```

Na listingu 3.9 znajduje się kod źródłowy klasy `AnnotatedDocumentRepository`. Zawiera ona marker `@Repository` i będzie wstrzykiwana do implementacji `SearchEngine` dzięki adnotacji `@Autowired` (listing 3.8).

Listing 3.9. *AnnotatedDocumentRepository.java*

```
package com.apress.isf.spring.annotated.data;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.spring.data.DocumentDAO;

@Repository("documentDAO")
public class AnnotatedDocumentRepository implements DocumentDAO {

    public Document[] getAll() {
        return storage();
    }

    private Document[] storage(){
        List<Document> result = new ArrayList<Document>();

        Type type = new Type();
        type.setName("PDF");
        type.setDesc("Portable Document Format");
        type.setExtension(".pdf");

        Document document = new Document();
        document.setName("Szablon książki");
        document.setType(type);
        document.setLocation("/Users/felipe/Documents/Random/Book Template.pdf");

        result.add(document);

        document = new Document();
        document.setName("Przykładowa umowa");
        document.setType(type);
        document.setLocation("/Users/felipe/Documents/Contracts/Sample Contract.pdf");

        result.add(document);

        type = new Type();
        type.setName("NOTE");
        type.setDesc("Notatki tekstowe");
        type.setExtension(".txt");

        document = new Document();
        document.setName("Clustering with RabbitMQ");
```

```

        document.setType(type);
        document.setLocation("/Users/felipeg/Documents/Random/Clustering with RabbitMQ.txt");

        result.add(document);

        type = new Type();
        type.setName("WEB");
        type.setDesc("Łącze sieciowe");
        type.setExtension(".url");

        document = new Document();
        document.setName("Pro Spring Security Book");
        document.setType(type);
        document.setLocation("http://www.apress.com/9781430248187");

        result.add(document);

        return result.toArray(new Document[result.size()]);
    }
}

```

Teraz skierujemy uwagę na listing 3.10. Znajduje się na nim treść pliku konfiguracyjnego XML zawierającego elementy nakazujące kontenerowi Spring poszukać klas z adnotacjami i ich adnotacji. W pliku tym użyty został też specjalny element `<context:component-scan-base-package/>`. Należy on do tej samej przestrzeni nazw XML co nasza konfiguracja. W przestrzeni tej będzie dodanych jeszcze więcej znaczników, o których przeczytasz w dalszych rozdziałach tej książki. Na razie wystarczy wiedzieć, że opisywany element nakazuje kontenerowi Spring wyszukiwanie klas z adnotacjami z podanego pakietu, tu `com.apress.isf.spring.annotated`, i wszystkich podpakietów.

Listing 3.10. Zawartość pliku *Mydocuments-annotations-context.xml*

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.0.xsd">

    <context:component-scan base-package="com.apress.isf.spring.annotated"/>

    <bean id="webType" class="com.apress.isf.java.model.Type">
        <property name="name" value="WEB" />
        <property name="desc" value="Łącze sieciowe" />
        <property name="extension" value=".url" />
    </bean>
</beans>

```

Stereotypy Spring

Stereotypy są markerami pomagającymi kontenerowi Spring w zidentyfikowaniu składników zarządzanych przez system Spring. Markerów tych można używać do oznaczania zewnętrznych narzędzi przetwarzających albo jako referencji dla punktów przecięcia (ang. *pointcut*) w programowaniu aspektowym. Obecnie stereotypy

znajdujące się w kodzie źródłowym pomagają programistom zrozumieć ten kod, ponieważ sprawiają, że jest on bardziej czytelny oraz że są w nim zaznaczone warstwy strukturalne i architektoniczne. Do najczęściej wykorzystywanych stereotypów (w społeczności skupionej wokół systemu Spring i oczywiście w naszej aplikacji) zaliczają się te, których opis znajduje się w tabeli 3.2.

Konfiguracja w postaci klasy ziarna Java

W wersji 3.0 systemu Spring wprowadzono nową technikę konfigurowania kontenera Spring polegającą na użyciu nowej klasy konfiguracyjnej w postaci ziarna Java. Początkowo był to osobny projekt, ale postanowiono wcielić go do rdzenia systemu w wersji 3.0. Aktualnie jest to jedna z zalecanych metod konfigurowania kontenera, ponieważ daje wyraźny obraz relacji występujących między klasami i pokazuje interakcje pomiędzy nimi. A w niektórych przypadkach pomaga nawet uniknąć całego tego bałaganu związanego z plikami XML.

Na listingu 3.11 znajduje się kod źródłowy klasy konfiguracyjnej w Javie. Klasa ta jest równoważna z pokazanym na listingu 3.6 kodem konfiguracyjnym w formacie XML. Każda definicja ziarna z tego pliku ma odpowiednik w postaci adnotacji `@Bean` w metodzie.

Listing 3.11. Zawartość pliku *MyDocumentsContext.java*

```
package com.apress.isf.spring.config;

import java.util.HashMap;
import java.util.Map;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;
import com.apress.isf.spring.data.DocumentDAO;
import com.apress.isf.spring.data.DocumentRepository;
import com.apress.isf.spring.service.ServiceSearchEngine;

@Configuration
public class MyDocumentsContext {

    private Map<String,Document> documents = new HashMap<String,Document>();
    private Map<String,Type> types = new HashMap<String,Type>();

    @Bean
    public Type webType(){
        return getTypeFromMap("web");
    }

    @Bean
    public SearchEngine engine(){
        ServiceSearchEngine engine = new ServiceSearchEngine();
        engine.setDocumentDAO(documentDAO());
        return engine;
    }

    public MyDocumentsContext(){
        Type type = new Type();
        type.setName("PDF");
        type.setDesc("Portable Document Format");
        type.setExtension(".pdf");
    }
}
```

```

Document document = new Document();
document.setName("Szablon książki");
document.setType(type);
document.setLocation("/Users/felipeg/Documents/Random/Book Template.pdf");

documents.put("doc1", document);
types.put("pdf", type);

document = new Document();
document.setName("Przykładowa umowa");
document.setType(type);
document.setLocation("/Users/felipeg/Documents/Contracts/Sample Contract.pdf");

documents.put("doc2", document);

type = new Type();
type.setName("NOTE");
type.setDesc("Notatki tekstowe");
type.setExtension(".txt");

document = new Document();
document.setName("Clustering with RabbitMQ");
document.setType(type);
document.setLocation("/Users/felipeg/Documents/Random/Clustering with RabbitMQ.txt");

documents.put("doc3", document);
types.put("note", type);

type = new Type();
type.setName("WEB");
type.setDesc("Łącze sieciowe");
type.setExtension(".url");

document = new Document();
document.setName("Pro Spring Security Book");
document.setType(type);
document.setLocation("http://www.apress.com/9781430248187");

documents.put("doc4", document);
types.put("web", type);
}

private DocumentDAO documentDAO(){
    DocumentRepository documentDAO = new DocumentRepository();
    documentDAO.setDoc1(getDocumentFromMap("doc1"));
    documentDAO.setDoc2(getDocumentFromMap("doc2"));
    documentDAO.setDoc3(getDocumentFromMap("doc3"));
    documentDAO.setDoc4(getDocumentFromMap("doc4"));
    return documentDAO;
}

private Document getDocumentFromMap(String documentKey){
    return documents.get(documentKey);
}

private Type getTypeFromMap(String typeKey){
    return types.get(typeKey);
}
}

```

Na początku kodu tej klasy dodaliśmy adnotację `@Configuration`, a metodom przypisaliśmy adnotację `@Bean`. Użycie adnotacji `@Configuration` jest jak przekazanie kontenerowi Spring następującej informacji: „Tutaj znajdują się definicje moich ziaren”. Natomiast adnotacja `@Bean` przed metodą jest równoznaczna z utworzeniem elementu `<bean/>` i ustawieniem jego własności. A zatem powyższa klasa poinformuje kontener Spring o tym, jakie są ziarna i jak będą powiązane.

Teraz wykorzystamy tę nową konfigurację w teście jednostkowym (listing 3.12).

Listing 3.12. Zawartość pliku *MyDocumentsBeanConfigurationTest.java*

```
package com.apress.isf.spring.test;

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;

import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;
import com.apress.isf.spring.config.MyDocumentsContext;

public class MyDocumentsBeanConfigurationTest {

    private ApplicationContext context;
    private SearchEngine engine;
    private Type webType;

    @Before
    public void setup() {
        context = new AnnotationConfigApplicationContext(MyDocumentsContext.class);
        engine = context.getBean(SearchEngine.class);
        webType = context.getBean(Type.class);
    }

    @Test
    public void testWithBeanConfigurationFindByType() {
        List<Document> documents = engine.findByType(webType);
        assertNotNull(documents);
        assertTrue(documents.size() == 1);
        assertEquals(webType.getName(), documents.get(0).getType().getName());
        assertEquals(webType.getDesc(), documents.get(0).getType().getDesc());
        assertEquals(webType.getExtension(), documents.get(0).getType().getExtension());
    }

    @Test
    public void testWithBeanConfigurationListAll() {
        List<Document> documents = engine.listAll();
        assertNotNull(documents);
        assertTrue(documents.size() == 4);
    }
}
```

W tym kodzie do załadowania konfiguracji użyliśmy klasy `Java AnnotationConfigApplicationContext`. W ten sposób informujemy kontener Spring, że stosowana jest konfiguracja klasowa, a nie w postaci pliku XML. Aby przeprowadzić ten test jednostkowy, należy wykonać polecenie `gradle test`:

```
gradle test
```

Użycie klasy `GroovyBeanDefinitionReader`

W wersji 4. systemu Spring wprowadzono nowy sposób konfigurowania ziaren przy użyciu języka programowania Groovy. Nowość tę ściągnięto z systemu Grails, w którym do tworzenia ziaren zaczęto używać języka specjalistycznego (ang. *domain-specific language* — DSL).

Spójrz na przykładowy kod przedstawiony na listingu 3.13.

Listing 3.13. Zawartość pliku `mydocuments.groovy`

```
import com.apress.isf.spring.service.ServiceSearchEngine
import com.apress.isf.spring.data.DocumentRepository
import com.apress.isf.java.model.Document

beans {

    engine(ServiceSearchEngine){
        documentDAO = ref("documentDAO")
    }

    documentDAO(DocumentRepository){
        doc1 = ref("doc1")
        doc2 = ref("doc2")
        doc3 = ref("doc3")
        doc4 = ref("doc4")
    }

    doc1(Document){
        name = "Szablon książki"
        type = ref("pdfType")
        location = "/Users/felipeg/Documents/Random/Book Template.pdf"
    }

    doc2(Document){
        name = "Przykładowa umowa"
        type = ref("pdfType")
        location = "/Users/felipeg/Documents/Contracts/Sample Contract.pdf"
    }

    doc3(Document){
        name = "Clustering with RabbitMQ"
        type = ref("noteType")
        location = "/Users/felipeg/Documents/Random/Clustering with RabbitMQ.txt"
    }

    doc4(Document){
        name = "Pro Spring Security Book"
        type = ref("webType")
        location = "http://www.apress.com/9781430248187"
    }
}
```

```

webType(com.apress.isf.java.model.Type){
    name = "WEB"
    desc = "Łączy sieciowe"
    extension = ".url"
}

pdfType(com.apress.isf.java.model.Type){
    name = "PDF"
    desc = "Portable Document Format"
    extension = ".url"
}

noteType(com.apress.isf.java.model.Type){
    name = "NOTE"
    desc = "Notatki tekstowe"
    extension = ".txt"
}
}

```

Na listingu tym przedstawiono nowy sposób definiowania konfiguracji za pomocą języka Groovy zamiast XML (listing 3.6), adnotacji (listing 3.9) lub klasy Java (listing 3.10). Na razie nie przejmuj się składnią, bo jej opis znajduje się w rozdziale 18.

Teraz utworzymy test jednostkowy (listing 3.14), w którym użyjemy nowej klasy `GroovyBeanDefinitionReader` do załadowania definicji ziaren. Klasa ta będzie łaadowała plik *mydocuments.groovy*, którego zawartość pokazano na listingu 3.13.

Listing 3.14. Zawartość pliku *MyDocumentsBeanDefinitionReaderTest.java*

```

package com.apress.isf.spring.test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;

import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.GenericGroovyApplicationContext;

import com.apress.isf.java.model.Document;
import com.apress.isf.java.model.Type;
import com.apress.isf.java.service.SearchEngine;

public class MyDocumentsBeanDefinitionReaderTest {

    private ApplicationContext context;
    private SearchEngine engine;
    private Type webType;

    @Before
    public void setup(){
        context = new GenericGroovyApplicationContext("META-INF/spring/mydocuments.groovy");
        engine = context.getBean(SearchEngine.class);
        webType = context.getBean("webType", Type.class);
    }
}

```



```

@Test
public void testWithGroovyFindByType() {
    List<Document> documents = engine.findByType(webType);
    assertNotNull(documents);
    assertTrue(documents.size() == 1);
    assertEquals(webType.getName(), documents.get(0).getType().getName());
    assertEquals(webType.getDesc(), documents.get(0).getType().getDesc());
    assertEquals(webType.getExtension(), documents.get(0).getType().getExtension());
}

@Test
public void testWithGroovyListAll() {
    List<Document> documents = engine.listAll();
    assertNotNull(documents);
    assertTrue(documents.size() == 4);
}
}

```

Za pomocą klasy `GroovyBeanDefinitionReader` załadowaliśmy skrypt w języku Groovy, podając ścieżkę do niego (`META-INF/spring/mydocuments.groovy`). Klasa ta uruchomi kontener Spring, tworząc wszystkie potrzebne egzemplarze, wiążąc ze sobą nasze klasy oraz przygotowując je do użytku, gdy będą potrzebne. Pamiętaj, że tym razem do konfiguracji kontenera Spring wykorzystaliśmy język programowania Groovy!

Teraz możesz uruchomić test za pomocą poniższego polecenia:

```
gradle test
```

Który rodzaj konfiguracji wybrać

Każdy typ konfiguracji różni się czymś od pozostałych. Różnice te zostały opisane w tabeli 3.3.

Tabela 3.3. Różnice w zastosowaniu różnych rodzajów konfiguracji

Typ konfiguracji	Zastosowanie
XML	Nadaje się do użycia z zewnętrznymi bibliotekami i różnymi środowiskami programistycznymi. Jest czytelna, ale może być bardzo rozwlekła. Wprawdzie można ją podzielić na kilka plików, jednak trzeba nad nimi zapanować.
Adnotacje	W tym typie konfiguracji wiąże się kontekst Spring z aplikacją. Można tego uniknąć przy użyciu własnych dziedzinowych adnotacji.
Ziarno Java	Jest to jedna z aktualnie zalecanych metod dla programistów, którzy nie lubią formatu XML. Może być stosowana w przypadku ziaren i składników, które niewiele się zmieniają.
Konfiguracja w języku Groovy	Nowa technika umożliwiająca wykorzystanie języka programowania Groovy do definiowania konfiguracji. Jest prosta i oszczędna, jeśli chodzi o kod źródłowy.

Wybór typu konfiguracji zależy od potrzeb i sposobu zarządzania cyklem programistycznym. Czasami dyktują go też preferencje zespołu programistycznego, któremu łatwiej może być zarządzać projektem i osiągać cele przy użyciu określonej technologii.

Podsumowanie

W tym rozdziale poznałeś różne sposoby tworzenia ziaren w celu uzyskania tego samego efektu. Poznałeś też różnorodne klasy do ładowania ziaren, takie jak `ClassPathXmlApplicationContext`, `AnnotationConfigApplicationContext` oraz nowa klasa `GenericGroovyApplicationContext`.

Nauczyłeś się oznaczać klasy adnotacjami w taki sposób, by poinformować system Spring o tym, jakich ziaren ma używać, jakie są ich zależności oraz jakie relacje je łączą. Dowiedziałeś się, że adnotacja `@Repository` oznacza klasę jako zarządzaną przez Spring składnik pełniący funkcję obiektu dostępowego do danych.

Ponadto dowiedziałeś się, jak używać konfiguracyjnej klasy w Javie oraz jak za pomocą jej adnotacji `@Configuration` i `@Bean` tworzyć ziarna i relacje między nimi. Poznałeś też technikę tworzenia konfiguracji przy użyciu języka programowania Groovy.

W następnym rozdziale dodamy nowe funkcje do aplikacji *Moje dokumenty*. Dowiesz się, w jaki sposób system Spring inicjuje klasy, oraz poznasz różne metody tworzenia różnych egzemplarzy tej samej klasy.

Skorowidz

A

adnotacja, 44

- @ActiveProfiles, 86, 87
- @Around, 112
- @Aspect, 112
- @Async, 182, 183
- @Autowired, 45, 75, 84, 138
- @Bean, 21, 48
- @Component, 45
- @ComponentScan, 256
- @Configuration, 50, 256
- @ContextConfiguration, 84, 220
- @Controller, 22, 45, 138, 139
- @DBRef, 201, 202
- @EnableAutoConfiguration, 256
- @EnableRabbitMessaging, 268
- @FixMethodOrder, 158
- @Id, 201
- @Ignore, 210
- @PathVariable, 172
- @Profile, 87
- @ProfileValueSourceConfiguration, 90
- @Repeat, 92
- @Repository, 45, 128
- @RequestMapping, 138
- @ResponseBody, 172
- @RunWith, 84
- @Scheduled, 185, 186
- @Scope, 58, 59
- @Service, 45
- @Timed, 92

adnotacje testowe dodatkowe, 90

aliasy, 241

AMQP, Advanced Message Queuing Protocol, 147, 158, 211

AOP, aspect-oriented programming, 93, 111

Apache Active MQ, 290

API REST, 223, 236

API typu RESTful, 167

aplikacja Moje dokumenty, 27, 95, 115

aspekt, aspect, 97

atrybut

- documentDAO, 40
- elementu bean, 38
- profile, 86
- ref, 43
- zakresu, 58

B

baza danych, 124, 128, 207

- MongoDB, 287
- NoSQL, 199

biblioteka

- AspectJ, 111
- XStream, 152, 214

błąd, 92

broker

- Apache Active MQ, 290
- RabbitMQ, 159
- wiadomości, 292
- wiadomości RabbitMQ, 158

C

cykl życia kontenera, 35

D

dane aplikacji, 115

diagram UML, 28

dokument informacyjny, 148

DSL, domain-specific language, 51, 192, 239, 244
 dynamiczne
 języki programowania, 191
 strony internetowe, 139
 działanie profili, 85
 dziennik, 55

E

element
 <bean/>, 50
 <beans/>, 86
 <context:component-scan/>, 86, 128
 <context:component-scan-base-package/>, 47
 <Document/>, 217
 <jdbc:embedded-database/>, 138
 <jms:listener-container/>, 164
 <lang:groovy/>, 194
 <lang:jruby/>, 197
 <list/>, 66
 <map/>, 66
 <mongo/>, 241
 <props/>, 66
 <rabbit:admin/>, 161
 <rabbit:connection-factory/>, 161, 218
 <rabbit:direct-exchange/>, 219
 <rabbit:listener-container/>, 161, 219
 <rabbit:queue/>, 161, 219
 <rabbit:template/>, 161, 219
 <set/>, 66
 <task:annotation-driven/>, 183
 elementy XML, 66

F

format
 JSP, 139
 JSPX, 139
 XML, 216

G

GVM, Groovy enVironment Manager, 282

H

hiperłącza, 167
 Homebrew, 261
 HTTP DELETE, 177
 HTTP GET, 174
 HTTP POST, 175
 HTTP PUT, 176

I

identyfikator URI, 167
 implementacja
 interfejsu, 20
 interfejsu SearchEngine, 40
 klasy DocumentDAO, 202
 informacja o wysłaniu wiadomości, 150
 instalacja
 ActiveMQ, 291
 Apache Active MQ, 290
 Gradle, 282
 Gradle w systemie Linux, 284
 Gradle w systemie Mac OS X, 284
 Gradle w systemie Windows, 283
 Groovy w systemie Linux, 287
 Groovy w systemie Mac OS X, 287
 Groovy w systemie Windows, 285
 GVM, 282
 Homebrew, 281
 interpretera języka Groovy, 285
 Javy, 273
 Javy w systemie OS X, 279
 Javy w systemie Windows, 274
 MongoDB, 287
 MongoDB w systemie Mac OS X, 289
 MongoDB w systemie Windows, 287
 narzędzi, 273
 RabbitMQ, 292
 RabbitMQ w systemie Mac OS X, 293
 RabbitMQ w systemie Windows, 292
 Spring Boot, 294
 Spring XD, 261, 295, 296
 instrukcja INSERT, 124
 integrowanie aplikacji, 147
 interfejs
 AfterReturningAdvice, 103
 API, 167
 documentDAO, 120
 javax.jms.MessageListener, 151
 JDBC, 120, 121, 134
 Logger, 56
 Login, 191
 Marshaller, 214
 MessageService.java, 19
 MethodBeforeAdvice, 101, 102
 RabbitMQConsumer, 213
 SearchEngine, 29, 34, 40
 TypeDAO, 204
 TypeDataDAO, 64
 internacjonalizacja, 142
 internetowe typy danych, 167

J

JDBC, 120, 128, 131, 134, 251
 JDK, Java Development Kit, 273
 język
 BeanShell, 196, 197
 DSL, 239, 244
 Groovy, 23, 52, 192, 239, 243, 256, 285
 JRuby, 196
 języki
 dynamiczne, 191
 dziedziczne, 239
 JMS, Java Message Service, 147, 148, 290
 JPA, Java Persistence API, 210
 JSTL, Java Standard Tag Library, 141

K

klasa

ActiveMQConnectionFactory, 154
 ApplicationContext, 21
 AroundLoggingModule, 105
 BeforeLoggingModule, 102
 CachingModule, 109
 ClassPathXmlApplicationContext, 34, 44
 ClassPathXMLApplicationContext, 92
 Connection, 121
 CookieLocaleResolver, 143
 CustomProfile, 90
 DataSource, 121
 DispatcherServlet, 138
 Document, 28, 118
 DocumentDAO, 40, 135, 202
 DocumentFacade, 234
 DocumentJdbcTemplateRepository, 132, 133
 DocumentRepository, 40, 100, 171
 DocumentRowMapper, 131
 DocumentService, 168
 DocumentServiceFacade, 169, 220
 DocumentTweet, 233, 235
 EmailService, 182
 FileSearchEngineService, 88
 GroovyBeanDefinitionReader, 37, 51–53, 60
 JavaMailSenderImpl, 180
 JdbcTemplate, 131
 JMSConsumer, 153
 Login, 77
 MailSender, 180
 Menu, 74
 MethodInvocation, 105
 MyDocumentsController, 259

MySearchEngine, 29, 32, 34
 org.springframework.aop, 102
 PdfDocumentConsumer, 213
 PropertyPlaceholderConfigurer, 78, 82
 Proxy, 102
 rabbitmqProducer, 162
 RabbitMQProducer, 211
 ReloadableResourceBundleMessageSource, 143
 Repository, 258
 Resource, 72
 ResourceBundleMessageSource, 79
 ResultSet, 121
 RowMapper, 131, 136
 Scheduling, 182
 SearchEngine, 99
 SearchEngineService, 100
 SearchEngineService, 102
 SecurityServiceFacade, 192
 ServiceSearchEngine, 43, 56
 SimpleMailMessage, 180
 SpringBootTestInitializer, 256
 SpringJUnit4ClassRunner, 84
 Statement, 121
 Tasks, 182
 TextDocumentsConsumer, 213
 Type, 29, 117
 Utility, 151
 XStreamMarshaller, 217, 242
 klasy konfiguracyjne, 48
 klucz trasowania, 212
 kolejka, 212, 220
 ActiveMQ, 155
 mydocumentsQueue, 148, 149, 157
 kolejki w RabbitMQ, 163
 kolekcje, 63, 66, 207
 scalanie, 67
 typów, 207
 w konfiguracji XML, 65
 kompilator Gradle, 19, 20
 komunikacja między systemami, 147
 konfiguracja
 JDBC, 251
 XML dla JDBC, 216
 XML dla MongoDB, 216
 konfiguracje
 adnotacje, 44, 53
 DSL, 247
 kontenera, 37
 testowe, 84
 w języku Groovy, 51, 53
 XML, 37, 53, 247
 ziarna Java, 48, 53

konsola
 hawtio, 291
 RabbitMQ, 266, 292
 kontener wstrzykiwania zależności, 35
 kontrola jakości, 87
 kontroler frontowy, 138
 konwertowanie dokumentów, 216

L

lista narzędzi, 273
 listy, 63
 lokalizacje, 79

M

magazynowanie danych, 115
 mechanizm
 przechowywania danych, 115
 tworzenia widoków, 139
 media społecznościowe, 223
 menedżer
 pakietów, 261
 środowiska języka Groovy, 282
 metoda
 afterThrowing, 107
 context.getBean, 21, 34
 findByLocation, 99
 findByType, 110
 HTTP DELETE, 177
 HTTP GET, 174
 HTTP POST, 175
 HTTP PUT, 176
 initialize, 128
 JmsTemplate.send, 156
 listAll, 30
 rabbitTemplate.send, 160
 send, 179
 setup, 44
 SpringApplication.run, 256
 storage, 30, 42
 testSpringRabbitMQ_1(), 162
 tweet, 233
 urlCheck, 187
 metody
 konfigurowania kontenera, 48
 protokołu HTTP, 167
 prywatne, 30
 model, 137
 moduł
 Spring AMQP, 158, 211
 Spring Data, 199, 210

Spring Rabbit, 158
 Spring Social, 223
 Spring Social Core, 223
 Spring Social Facebook, 223
 Spring Social Twitter, 223
 Spring XD, 261, 295

moduły
 przetwórcze, processor, 262
 ujściowe, sink, 262
 źródłowe, source, 262
 MongoDB, 199, 204, 205
 MVC, 135, 137, 141

N

narzędzia, 18, 273
 analityczne, 266
 dla systemu Mac OS X, 281
 nawias
 klamrowy, 244
 prostokątny, 245

O

obiekt POJO, 38
 obsługa
 baz danych, 199
 RabbitMQ, 293
 symboli zastępczych, 82
 osadzanie bazy danych, 128

P

pakiet JDK, 18, 273
 pierwsza aplikacja
 Spring, 17
 Spring XD, 261
 planowanie zadań, 185
 plik
 AfterLoggingModule.java, 103
 all.jsp, 139
 AnnotatedDocumentRepository.java, 46, 128
 AnnotatedSearchEngine.java, 44, 59
 app.groovy, 24, 259
 Application.java, 20
 AroundLoggingModule.java, 105
 BeforeLoggingModule.java, 101
 build.gradle, 19, 31, 140, 249, 254
 Caching.java, 111
 CachingModule.java, 108
 controller.groovy, 258
 CustomProfile.java, 88

data.sql, 124
 dictionary.properties, 80
 Document.java, 116, 199
 DocumentController.java, 171
 DocumentDAO.java, 41
 DocumentFacade.java, 234
 DocumentJdbcTemplateRepository.java, 131
 DocumentRepository.java, 41, 63, 120, 124, 135, 152, 169
 DocumentRowMapper.java, 136
 DocumentScheduler.java, 185, 186
 DocumentService.java, 168
 DocumentServiceFacade.java, 168
 DocumentTweet.java, 232
 email.properties, 181
 EmailService.java, 179, 182
 env_dev.properties, 77
 FileSearchEngineService.java, 87
 groovylogin.groovy, 193
 jdbc.properties, 121
 JDBCConfig.java, 250
 jdbc-context.xml, 251
 JMSConsumer.java, 151
 JMSProducer.java, 155
 Login.java, 77, 191
 LoginService.java, 77, 191
 Menu.java, 73
 menu.txt, 71
 messages.properties, 143
 model.groovy, 257
 MongoDocumentRepository.java, 202
 MongoTypeRepository.java, 204
 MyDocumentAOPTest.java, 110
 mydocuments.groovy, 51, 61, 240
 Mydocuments-annotations-context.xml, 47
 mydocuments-aop-annotated-context.xml, 112
 mydocuments-aop-context.xml, 99, 102–109
 MyDocumentsAOPTest.java, 100
 MyDocumentsApp.java, 252
 MyDocumentsBeanConfigurationTest.java, 50
 MyDocumentsBeanDefinitionReaderTest.java, 52
 MyDocumentsContext.java, 48, 59
 mydocuments-context.xml, 32, 37, 42, 57, 65, 153, 156, 160, 163, 180, 183, 193–197, 204, 231, 239
 mydocuments-custom-profiles-context.xml, 88
 mydocuments-i18n-context.xml, 80
 MyDocumentsI18nTest.java, 81
 mydocumentsJDBC.groovy, 240
 mydocuments-jdbc-context.xml, 121, 127, 215
 MyDocumentsJDBCEmbeddedAnnotatedTest.java, 130
 mydocuments-jdbc-embedded-context.xml, 128
 MyDocumentsJDBCTemplateTest.java, 133
 mydocuments-jdb-template-context.xml, 132
 mydocuments-login-context.xml, 78
 mydocumentsMongo.groovy, 241
 mydocuments-mongo-context.xml, 216
 MyDocumentsMoreAnnotationsTest.java, 91
 MyDocumentsTest.java, 30, 235
 mydocumentsOXM.groovy, 242
 mydocuments-oxm-context.xml, 216
 mydocuments-profiles-context.xml, 85
 mydocumentsRabbitMQ.groovy, 242
 mydocuments-rabbitmq-context.xml, 217
 mydocuments-resource-injection-context.xml, 73
 mydocuments-servlet.xml, 138, 142, 172
 MyDocumentsTest.groovy, 243
 MyDocumentsTest.java, 43, 71, 83, 154, 181, 184, 194, 205, 208, 219, 233
 MyDocumentsTestWithSpring.java, 33
 mydocuments-util-context.xml, 68
 MyDocumentsWebApp.java, 255
 MyDocumentsWithCustomProfilesTest.java, 88
 MyDocumentsWithLoginTest.java, 78
 MyDocumentsWithProfilesTest.java, 86
 MyDocumentsWithResourceInjectionTest.java, 74
 MyDocumentsWithResourceLoaderInjectionTest.java, 76
 MyDocumentTest.java, 56, 157
 MySearchEngine.java, 38
 PdfDocumentsConsumer.java, 213
 RabbitMQConsumer.java, 164, 212
 RabbitMQProducer.java, 158, 211
 rabbitmqStream.groovy, 267
 repo.groovy, 257
 ResourceLoaderMenu.java, 75
 schema.sql, 123
 SearchController.java, 137
 SearchEngine.java, 98
 SearchEngineService.java, 40, 55, 95
 SecurityServiceFacade.java, 192
 service.groovy, 258
 TextDocumentsConsumer.java, 213
 ThrowsLoggingModule.java, 107
 twitter.properties, 231
 Type.java, 117, 201
 User.java, 118
 web.xml, 138
 WebDocumentsConsumer.java, 213
 XmlUtils.java, 151
 pliki
 konfiguracyjne, 23
 własności, 76
 wykonywalne, 253, 259
 zasobów, 71

podsluchy, tap, 262
 podział funkcjonalności programu, 96
 pojęcia programowania aspektowego, 98
 POJO, plain old java object, 38
 polecenie stream list, 265
 powiązania, 212, 221
 powłoka Spring XD, 264
 profile, 85
 program
 ActiveMQ, 148, 290
 Apache Active MQ, 290
 Gradle, 18, 24, 282
 Homebrew, 281
 Maven, 19
 RabbitMQ, 211, 266, 292
 programowanie aspektowe, AOP, 93, 96, 98
 protokół
 AMQP, 158, 212, 292
 HTTP, 171
 przechowywanie danych, 115
 przechwytywacz LocaleChangeInterceptor, 143
 przekształcanie wiadomości, 214
 przestrzenie nazw RabbitMQ, 161
 przesyłanie
 danych, 167
 wiadomości, 211
 publikowanie aplikacji, 135
 punkt przecięcia, pointcut, 47, 97

R

RabbitMQ, 158
 rada, advice, 97
 after, 103
 AfterThrowing, 107
 around, 105
 before, 101
 rady dla aplikacji, 95
 rejestratory, 99
 rejestrowanie
 aktywności, 55
 aplikacji w Twitterze, 224
 relacje w aplikacji, 115
 renderowanie strony, 141
 reprezentacja zasobów, 167
 REST, Representational State Transfer, 167
 RESTClient, 176
 rodzaje konfiguracji, 53
 rozszerzenie
 AOP, 111
 Groovy Boot, 24
 MVC, 137, 141

Spring AMQP, 212
 Spring Boot, 21, 24
 Spring Social, 223

S

SaaS, Software as a Service, 223
 scalanie kolekcji, 67
 schemat bazy danych, 124
 serwer MongoDB, 205, 220
 uruchamianie, 288, 289
 zatrzymywanie, 289
 serwer RabbitMQ, 220
 skanowanie pakietów, 112
 składnia języka DSL, 244
 skrypty, 259
 słownik, 63
 hiszpański, 81
 polski, 80
 słowo kluczowe
 beans, 240, 244
 providedRuntime, 255
 public, 24
 Spring Boot, 249, 256, 294
 Spring JMS, 148
 Spring Rabbit, 158
 Spring Social, 223
 Spring Social Twitter, 223
 Spring XD, 261
 dane analityczne, 267
 licznik, 267
 podsluchy, 262
 powłoka, 263, 264
 rodzaje modułów, 262
 tryb multi, 262
 tryb pojedynczy, 262
 SQL, 124
 stereotypy, 45, 47
 strona brokera RabbitMQ, 159
 struktura folderów, 18
 strumień rabbit, 264
 system
 baz danych, 199
 JUnit, 84
 Spring Boot, 294
 szkieletowy, 93
 szkieletowy Spring, 15
 wejścia-wyjścia, 237
 systemy zewnętrzne, 147
 szablon
 MongoOperations, 210
 widoku, 137

Ś

środowisko wykonawcze JSTL, 141

T

test jednostkowy, 34, 52, 68, 78, 81, 90, 108, 122, 207, 219, 243, 244

testowanie

- aplikacji, 83
- DocumentDAO, 207
- MongoDB, 205
- przy użyciu adnotacji, 83

trasowanie, 212

Twitter, 224

- formularz logowania, 224
- formularz tworzenia aplikacji, 225
- informacje o aplikacji, 227
- klucze dostępu, 230
- narzędzia dla programistów, 226
- tworzenie nowej aplikacji, 225
- tworzenie tokenu dostępowego, 230
- uprawnienia, 228
- ustawienia aplikacji, 227, 231
- ustawienia dostępu, 229

tworzenie

- aplikacji, 23, 225
- dynamicznych stron, 139
- kolejki, 148, 149
- liczników i mierników, 266
- pliku JAR, 253, 259
- pliku WAR, 254
- podsluchu, 264
- strumienia, 264
- widoków, 139
- ziaren, 37, 51, 54

typy własne, 63

U

uruchamianie

- ActiveMQ, 290, 291
- aplikacji, 23
- powłoki, 264
- serwera MongoDB, 288, 289

usługa

- JMS, 148
- sieciowa RESTful, 168

używanie adnotacji, 44

W

warstwa przechowywania danych, 135

wczytywanie plików zasobów, 127

wdrażanie

- aplikacji, 253
- strumieni, 265

wersje językowe serwisu, 79

wiadomość

- e-mail, 179
- XML, 149

widok, 137

wklejanie wiadomości XML, 149

własności, 63

własność

- basename, 81
- documentDao, 43
- interceptorNames, 107
- username, 78

wstrzykiwanie zależności, 21, 34, 35

wtyczka spring-boot, 250, 253

wybór

- konfiguracji, 53
- wersji językowej, 79

wyjątek UnsupportedOperationException, 107

wymagania do programu Moje dokumenty, 27

wymiany, 212

wysyłanie

- tweetów, 234
- wiadomości, 160, 161
- wiadomości e-mail, 179

wyszukiwanie klas, 47

wzorzec projektowy MVC, 135, 137

Z

zadania

- asynchroniczne, 182
- e-mail, 179

zakres

- globalSession, 58
- prototype, 58
- request, 58
- session, 58
- singleton, 58

zakresy ziaren, 55, 58

- w klasie GroovyBeanDefinitionReader, 60

zależności JAR, 112

zastosowanie technik AOP, 111

zatrzymywanie

 brokera ActiveMQ, 290, 291

 serwera MongoDB, 289

ziarna

 Javy, 38, 48

 zarządzane, 34

ziarno, bean, 38

 anotherTypeDAO, 67

 beforeLogging, 102

 engineProxy, 107

 xstreamMarshaller, 217

zmienna środowiskowa, 276

 JAVA_HOME, 278

 Path, 278

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

Wprowadzenie do Spring Framework dla programistów Java

Pojawienie się Springa na zawsze zmieniło świat Javy. Ten przemyślany, dobrze zaprojektowany, dający programistom mnóstwo możliwości framework został przez nich wyjątkowo ciepło przyjęty. Obecnie jest on prawdopodobnie najczęściej wybieranym narzędziem do tworzenia aplikacji — niezależnie od stopnia ich skomplikowania. Za Springiem stoją ogromna społeczność oraz przepastne zasoby dokumentacji, przykładów i instrukcji. Już teraz dowiedz się, dlaczego Spring jest numerem 1!

Książka ta wprowadzi Cię w świat Springa oraz zagadnień z nim związanych. Sięgnij po nią, by skonfigurować swój pierwszy projekt i przetestować aplikację. W kolejnych rozdziałach odkryjesz sposoby wykorzystania programowania aspektowego i wzorca MVC oraz integrowania aplikacji Spring z systemami zewnętrznymi. Ponadto przygotujesz swoje pierwsze API typu REST oraz zapiszesz dane w bazie MongoDB. Książka ta jest doskonałą lekturą dla osób znających język Java, chcących poznać legendarne możliwości Spring Framework.

Dzięki Spring Framework:

- błyskawicznie przygotujesz rozbudowaną aplikację
- bezproblemowo wykorzystasz JMS do integracji aplikacji
- z łatwością skorzystasz z baz danych NoSQL
- zbudujesz przejrzyste REST API

sięgnij po WIĘCEJ



KOD KORZYŚCI

Helion

29967 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nawosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. +32 230 98 43
e-mail: helion@helion.pl
<http://helion.pl>

ISBN 978-83-283-0439-0



9 788328 304390

Informatyka w najlepszym wydaniu

cena: 57,00 zł

Apress