
spring
PROFISIEN
Spring Framework
Fast Track 2019
A complete of essential concepts and examples
Author: Erfin Feluzy & Wahyu Sudrajat
March 2019

Profile

ERFIN FELUZY

- 11+ years professional experience in Java Programming (10+ years spring framework)
- General Manager in Software Solution
- Sr Middleware Solution Architect

WAHYU SUDRAJAT

- 8+ years professional experience in Spring
- Linux master & Mathematician
- Solution Architect in Software Solution

Agenda Day 1 (March 9, 2019)

- 09:00 – 10:00 : Spring Framework intro
- 10:00 – 10:30 : Spring Boot intro (Hands-On)
- 10:30 – 11:00 : Basic build with Maven (Hands-On)
- 11:00 – 12:00 : Hands-On prerequisite setup
- **12:00 – 13:00 : Ishoma**
- 13:00 – 14:00 : Spring web MVC (Hands-On)
- 14:00 – 15:00 : Spring Data Database (Hands-On)
- 15:00 – 16:00 : Spring Web Service REST producer (Hands-On)
- 16:00 – 17:00 : Spring Web Service SOAP producer (Hands-On)

Agenda Day 2 (March 16, 2019)

- 09:00 – 10:00 : Spring Web Service REST consumer (Hands-On)
- 10:00 – 11:00 : Spring Web Service SOAP consumer (Hands-On)
- 11:00 – 12:00 : Spring JMS Publisher & Subscriber (Hands-On)
- **12:00 – 13:00 : Ishoma**
- 13:00 – 14:30 : Lab : Use Case 1
- 14:30 – 16:00 : Lab : Use Case 2
- 16:00 – 17:00 : Recap & Closing

What is Spring?

Back then, enterprise application commonly use a technology called **Enterprise Java Bean (EJB)** which relatively heavy and tightly depend on enterprise solution such as: Application Web Server(s).

What is Spring?

- An open-source framework.
- An alternative to heavier enterprise Java technologies.
- Addresses the complexity of enterprise application development, thus it “Simplifies Java development ”

https://www.youtube.com/watch?v=a14xfNV-UIs

Spring is Non-Invasive

What does that mean?

- You are not forced to import or extend any Spring APIs
- An invasive framework takes over your code.
- Anti-pattern: (sample of invasive framework)
  - **EJB** forces you to use **Java Naming and Directory Interface (JNDI)**, commonly found at enterprise application framework
  - **Struts** forces you to extend **Action**, **ActionForm**

Invasive frameworks are inherently difficult to test. You have to stub the runtime that is supplied by the application server.

https://www.slideshare.net/pmanvi/spring-framework-overview-ppt-3068245

Benefit of Using Spring

Spring is a Framework which help to solve common software developing problems with Java programming language, such as:

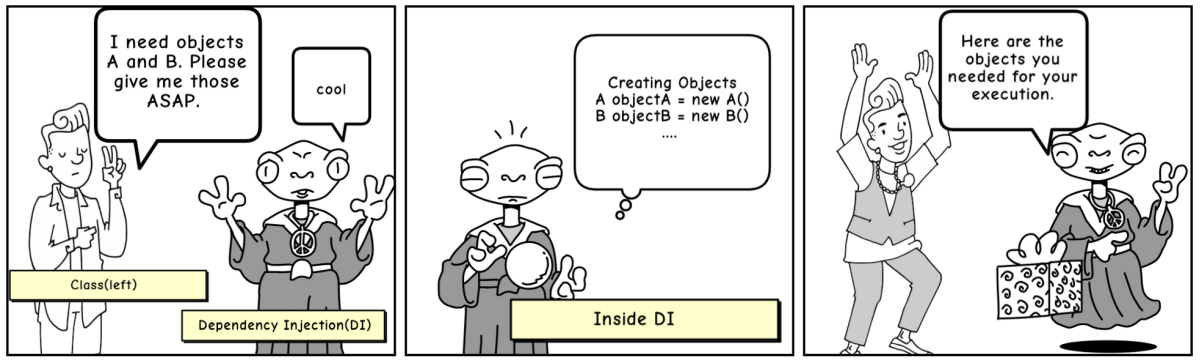
- Manage lifecycle of Java classes (called beans)
- Dependency Injection framework
- Provide Data Access which commonly works through Java Database Connectivity (JDBC), Object-Relational Mapping (ORM), etc.
- Provide Spring-MVC which standardize patterns and structure to develop Web Application
- more…

https://www.youtube.com/watch?v=gq4S-ovWVlM

Dependency Injection (DI)

Wikipedia definition of DI:

*"In software engineering, **dependency injection** is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a service)"*



https://medium.freecodecamp.org/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f

What is Spring Boot?

Although Spring does simplify Java development process, it can give programmers an adversity while developing an application.

- Boiler plate configuration
  - Programmer writes a lot of codes (configurations) to do minimal task
- Takes time to have Spring application up and running

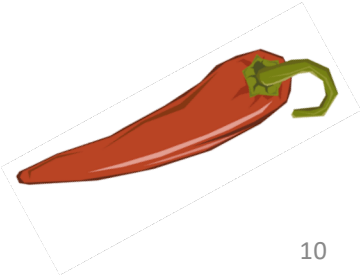
Spring Boot offers the solution for those problems. Spring Boot is a suite of pre-configured framework and technologies and it is a shortest way to have Spring application up and running.

https://www.youtube.com/watch?v=LLfZ2g2JaIc

Hand-On Prerequisites Setup

- **Oracle SDK 8+**, you might also use **OpenJDK**, make sure to have JDK installed instead of JRE
- Java IDE, **Eclipse 2018-12** is preferred
  - Eclipse by default comes with Apache Maven embedded
- Properly setup Java Runtime Environment (JRE) on Environment Variable (Windows), or PATH variable (Unix based)
- Lombok Project for Simplify POJO

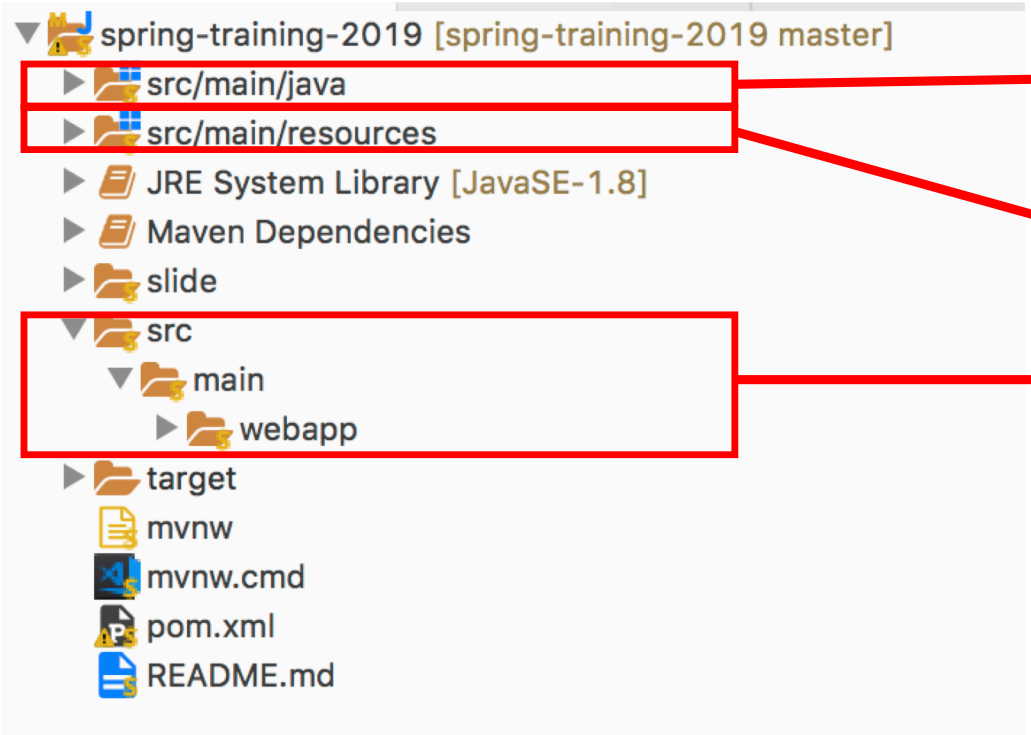
OpenJDK eclipse Maven



Spring Boot Hands-On

Spring Boot Intro

Maven Common Project Structure



Java files here

Application properties files (config) here

Web files here (jsp or html)

Spring Boot's pom.xml

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.3.RELEASE</version>
</parent>


<dependencies>
    <!-- Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

Spring boot parent artifact

Based on web MVC

Spring Boot Starter

Application.java

```
1  package com.erfinfeluzy.training;
2
3  import org.springframework.boot.SpringApplication;
6
7
8  @SpringBootApplication(scanBasePackages = { "com.erfinfeluzy" })
9  @PropertySource({
10     "classpath:apps.properties"
11 })
12 public class Application {
13
14     public static void main(String[] args) {
15
16         SpringApplication.run(Application.class, args);
17     }
18
19 }
20
21
```

Scan all spring bean inside base package

Spring boot custom configuration.
Eg: server port, database, etc

Spring boot custom configuration.
Eg: server port, database, etc


Maven Hands-On
Basic Build with Apache Maven

Apache Maven

Maven is a software management and comprehension tool based on the concept of Project Object Model(POM). It can manage project build, reporting and documentation, from a central piece of information. Basically, this tool can be used as both, a build tool (just like ANT tool), as well as a project management tool. However, in this article we will use it just for building our project.

Basic usage of maven are:

```
$> mvn clean
$> mvn compile
$> mvn install
```

https://medium.com/@himanshuagarwal1395/getting-started-with-apache-maven-hello-world-eccb278a262a

Spring Web MVC Hand-On

Spring Boot + Spring Web

Spring MVC (Model-View-Controller)

Customer.java

```
1  package com.erfinfeluzy.training.spring.model;
2
3  import java.util.Date;
12
13 @Data
14 @Entity
15 @Table(name = "tbl_customer")
16 public class Customer {
17
18     @Id
19     @GeneratedValue
20     private Long id;
21
22     @Column(name="username")
23     private String username;
24
25     @Column(name="first_name")
26     private String firstName;
27
28     @Column(name="last_name")
29     private String lastName;
30
31     @Column(name="birth_date")
32     private Date birthdate;
33
34
35
36 }
37
```

model

```
@Controller
public class HomeController {

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home() {

        System.out.println("masuk : / ");

        return "home";
    }
```

controller

home.jsp

```
1  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2  <html>
3
4      <head>
5          <title>Training</title>
6      </head>
7
8      <body>
9
10         <h2>Halo Trainee</h2>
11
12         <p>greetings: ${greetings}</p>
13
14     </body>
15
16 </html>
```

view

Spring Data Hand-On

Spring Boot + Spring Data

Spring Boot Data (config)

```
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;INIT=CREATE SCHEMA IF NOT EXISTS USERS
spring.datasource.username=sa
spring.datasource.password=sa
spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.show_sql=true

spring.h2.console.enabled=true
spring.h2.console.path=/console
```

Spring Boot Data (custom SQL query)

CustomerRepository.java

```
1  package com.erfinfeluzy.training.spring.service;
2
3  import java.util.List;
11
12 @Transactional
13 public interface CustomerRepository extends JpaRepository<Customer, Long> {
14
15     @Query(value = "select * from tbl_customer where first_name like :firstname",
16       nativeQuery = true)
17     List<Customer> findAllCustomerNativeSQL(
18             @Param("firstname") String firstname);
19
20 }
21
```

Invoke JDBC Service - Create

HomeController.java

```
26
27    @Autowired
28    CustomerRepository customerRepository;
29
30    @RequestMapping(value = "/customer/add", method = RequestMethod.GET)
31    public String customerAdd() {
32
33        System.out.println("/customer/add");
34
35        Customer cust = new Customer();
36        cust.setFirstName("erfin");
37        cust.setLastName("feluzy");
38        cust.setBirthdate(new Date());
39        cust.setUsername("erfinfeluzy");
40
41        customerRepository.save(cust);
42
43        return "add_customer";
44    }
45
```

Dependency Injection (Repository)

Create New Object

Save new Customer to Database

Invoke JDBC Service - Read

```
@RequestMapping(value = "/customer/all", method = RequestMethod.GET)
public String customerList(Model model) {
    List<Customer> customers = customerRepository.findAll();
    model.addAttribute("customers", customers);

    return "customers";
}
```

Find All Customer

Put as attribute on viewer (jsp)

Spring Web Service REST Producer Hand-On

Spring Boot + Spring Web

Web Service REST Config

HelloWorldRestController.java

```java
package com.erfinfeluzy.training.spring.rest;

import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/api")
public class HelloWorldRestController {

    /** Serve as {@link #getHelloWorld()} which constructed as REST producer. */
    @GetMapping(value = "/hello")
    public ResponseEntity<?> getHelloWorld() {

        Hello hello = new Hello();
        hello.setStatus("success");
        hello.setMessage("Hello World!");

        return ResponseEntity.ok().body(hello);
    }

    // define inner class which used specifically by this controller
    @Data
    private class Hello {
        private String status;
        private String message;
    }
}
```

Rest Web Service cd /api/hello

Spring Web Service REST Consumer Hand-On

Spring Boot + Spring Web

Spring Web Service SOAP Producer Hand-On

Spring Boot + Spring Web

Web Service SOAP Producer Config

SOAPConfig.java | JmsConfig.java | Sender.java | Receiver.java | TestJmsRestCont

```
2
3  import javax.xml.ws.Endpoint;
17
18 @SuppressWarnings("deprecation")
19 @Configuration
20 public class SOAPConfig {
21
22     /** SOAP producer **/
23
24     @Bean
25     public ServletRegistrationBean cxfDispatcherServlet() {
26         return new ServletRegistrationBean(new CXFServlet(), "/soap/*");
27     }
28
29     @Bean(name = Bus.DEFAULT_BUS_ID)
30     public SpringBus springBus() {
31         SpringBus springBus = new SpringBus();
32         return springBus;
33     }
34
35     @Bean
36     public Endpoint endpoint() {
37         EndpointImpl endpoint = new EndpointImpl(springBus(), customerSoapService());
38         endpoint.getFeatures().add(new LoggingFeature());
39         endpoint.publish("/CustomerService");
40         return endpoint;
41     }
42
43     @Bean
44     public CustomerSoapService customerSoapService() {
45         return new CustomerSoapServiceImpl();
46     }
47
```

SOAP Web Service:
/soap/CustomerService

Spring Web Service SOAP Consumer Hand-On

Spring Boot + Spring Web

Web Service SOAP Consumer Config

CustomerSoapSer | CustomerSoapSer | SOAPConfig.java | SoapConsumerCon

```
32          return springBus;
33      }
34
35      @Bean
36      public Endpoint endpoint() {
37          EndpointImpl endpoint = new EndpointImpl(springBus(), customerSoapService());
38          endpoint.getFeatures().add(new LoggingFeature());
39          endpoint.publish("/CustomerService");
40          return endpoint;
41      }
42
43      @Bean
44      public CustomerSoapService customerSoapService() {
45          return new CustomerSoapServiceImpl();
46      }
47
48      /** SOAP consumer **/
49
50      @Bean(name = "client")
51      public CustomerSoapService customerServiceProxy() {
52          return (CustomerSoapService) proxyFactoryBean().create();
53      }
54
55      @Bean
56      public JaxWsProxyFactoryBean proxyFactoryBean() {
57          JaxWsProxyFactoryBean proxyFactory = new JaxWsProxyFactoryBean();
58          proxyFactory.setServiceClass(CustomerSoapService.class);
59          proxyFactory.setAddress("http://localhost:8080/soap/CustomerService");
60          return proxyFactory;
61      }
62
63  }
64
```

SOAP Web Service:
/soap/CustomerService

Spring JMS Publisher & Subscriber Hand-On

Spring Boot + Spring JMS

JMS Publisher/Subscriber Java Config

CustomerSoapSer | SOAPConfig.java | SoapConsumerCon | JmsConfig.java | Sender.java

```java
package com.erfinfeluzy.training.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.support.converter.MappingJackson2MessageConverter;
import org.springframework.jms.support.converter.MessageConverter;
import org.springframework.jms.support.converter.MessageType;

@Configuration
@EnableJms
public class JmsConfig {

    // Serialize message content to json using TextMessage
    @Bean("jacksonMessageConverter")
    public MessageConverter jacksonJmsMessageConverter() {
        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
        converter.setTargetType(MessageType.TEXT);
        converter.setTypeIdPropertyName("_type");
        return converter;
    }

}
```

Message Converter for Object

JMS Publisher/Subscriber File Config

SOAPConfig.java | apps.properties | JmsConfig.java | Sender.java | Receiver.java | »11

```
1 # see more of default Spring common application.properties
2 # https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html
3
4 server.port=8080
5
6 spring.mvc.view.prefix=/WEB-INF/jsp/
7 spring.mvc.view.suffix=.jsp
8
9 spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;INIT=CREATE SCHEMA IF NOT EXISTS USERS
10 spring.datasource.username=sa
11 spring.datasource.password=sa
12 spring.datasource.driver-class-name=org.h2.Driver
13
14 spring.jpa.properties.hibernate.hbm2ddl.auto=update
15 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
16 spring.jpa.properties.hibernate.show_sql=true
17
18 spring.h2.console.enabled=true
19 spring.h2.console.path=/console
20
21 spring.jms.isPubSubDomain=true
22
23 spring.activemq.broker-url=tcp://localhost:61616
24 spring.activemq.user=admin
25 spring.activemq.password=admin
26
```

Allow pubSub for Topic

JMS Publisher (Sender)

CustomerSoapSer | CustomerSoapSer | SOAPConfig.java | SoapConsumerCon | JmsConfig.java | Sender.java

```java
package com.erfinfeluzy.training.spring.jms;

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

@Component
public class Sender {

    private static final String MAILBOX_QUEUE_DESTINATION = "mailbox.queue";
    private static final String TEXT_QUEUE_DESTINATION = "text.queue";

    private static final String ALERT_TOPIC_DESTINATION = "alert.topic";

    @Autowired
    @Qualifier("jacksonMessageConverter")
    private MessageConverter jacksonMessageConverter;

    @Autowired
    private JmsTemplate jmsTemplate;


    /** Queue **/

    public void sendTextMessage() {
        jmsTemplate.convertAndSend(TEXT_QUEUE_DESTINATION,
                "Test kirim pesan!");
    }

    public void sendEmailMessage() {
        jmsTemplate.setMessageConverter(jacksonMessageConverter);
        jmsTemplate.convertAndSend(MAILBOX_QUEUE_DESTINATION,
                new Email("admin@localhost", "Connection test!"));
    }


    /** Topic **/

    public void sendAlertMessage() {
        jmsTemplate.convertAndSend(ALERT_TOPIC_DESTINATION,
                "System alert!");
    }

}
```

Destination

JMS Queue

JMS Topic

JMS Subscriber (Receiver)

CustomerSoapSer | CustomerSoapSer | SOAPConfig.java | SoapConsumerCon | JmsConfig.java | Sender.java | Receiver.java

```java
package com.erfinfeluzy.training.spring.jms;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

import com.erfinfeluzy.training.spring.model.Email;

@Component
public class Receiver {

    private static final String MAILBOX_QUEUE_DESTINATION = "mailbox.queue";
    private static final String TEXT_QUEUE_DESTINATION = "text.queue";

    private static final String ALERT_TOPIC_DESTINATION = "alert.topic";

    /** Queue **/

    @JmsListener(destination = TEXT_QUEUE_DESTINATION)
    public void receiveTextMessage(String message) {
        System.out.println("Queue Received <" + message + ">");
    }

    @JmsListener(destination = MAILBOX_QUEUE_DESTINATION)
    public void receiveEmailMessage(Email email) {
        System.out.println("Queue Received <" + email + ">");
    }

    /** Topic **/

    @JmsListener(destination = ALERT_TOPIC_DESTINATION)
    public void receiveAlertMessage1(String message) {
        System.out.println("Topic Received (1) <" + message + ">");
    }

    @JmsListener(destination = ALERT_TOPIC_DESTINATION)
    public void receiveAlertMessage2(String message) {
        System.out.println("Topic Received (2) <" + message + ">");
    }

}
```

Destination

JMS Queue

JMS Topic

Lab – Use Case 1

Create Multiple Backend Web Services

Lab – Use Case 2

Create Microservices apps with Spring Boot

Recap & Closing