

Spring BOOT

Andy

1

Spring boot: initializer

The screenshot shows the Spring Initializr web application interface. The browser address bar shows 'start.spring.io'. The page has a sidebar on the left with sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The main content area has tabs for 'Maven Project' and 'Gradle Project', with 'Maven Project' selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', version '2.1.2' is selected. The 'Project Metadata' section has input fields for 'Group' and 'Artifact', and a section for 'Options'. The 'Dependencies' section has a search bar and a list of 'Search dependencies to add' (Web, Security, JPA, Actuator, Devtools...). At the bottom, there are buttons for 'Generate - Ctrl + G', 'Explore - Ctrl + Space', and 'Share...'. A footer note states: '© 2013-2020 Pivotal Software start.spring.io is powered by Spring Initializr and Pivotal Web Services'.

Create a spring boot project

Add dependency: web, actuator, JPA

2

Starter

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>myproject</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <!-- Inherit defaults from Spring Boot -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
  </parent>

  <!-- Add typical dependencies for a web application -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

  <!-- Package as an executable jar -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Common Starters

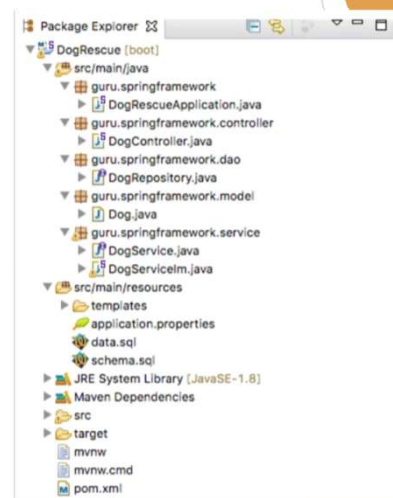
spring-boot-starter-data-jpa
spring-boot-starter-security
spring-boot-starter-test
spring-boot-starter-web

Question:
How does Spring boot determine the
dependency version?

3

Code Layout

```
com
+- example
  +- myapplication
    +- Application.java
    |
    +- customer
      +- Customer.java
      +- CustomerController.java
      +- CustomerService.java
      +- CustomerRepository.java
    |
    +- order
      +- Order.java
      +- OrderController.java
      +- OrderService.java
      +- OrderRepository.java
```



4

Application.java

```
package com.example.myapplication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

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

}
```

How it works:

1. @SpringBootApplication tells this is a spring boot application file and it starts from here
2. Main -> SpringApplication.run() bootstrap spring and application and server/tomcat
3. Application.class - this is the primary spring component

5

Annotations

- @SpringBootApplication
 - = @EnableAutoConfiguration
 - + @ComponentScan
 - (@Component vs @Bean)
 - + @Configuration

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

6

@EnableAutoConfiguration

- ▶ Enable auto-configuration of the Spring Application Context, attempting to guess and configure beans that you are likely to need. Auto-configuration classes are usually applied based on your classpath and what beans you have defined.

```
@Configuration
@EnableJpaRepositories
@EnableTransactionManagement
public class ApplicationConfig {
    @Bean
    public DataSource dataSource() {
        ...
    }

    @Bean
    public EntityManagerFactory entityManagerFactory() {
        ...
        factory.setDataSource(dataSource());
        return factory.getObject();
    }

    @Bean
    public PlatformTransactionManager transactionManager() {
        JpaTransactionManager txManager = new JpaTransactionManager();
        txManager.setEntityManagerFactory(entityManagerFactory());
        return txManager;
    }
}
```



```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

7

@Component vs @Bean

- ▶ @Component (@Component, @Controller, @Service, @Repository) - class level
- ▶ @Bean - method level

```
@Configuration
@ComponentScan({"com.logicbig.example.di.cons"})
public class ConstBasedDI_WireByType {

    @Bean(name = "a")
    public OrderService orderServiceByProvider1 () {
        return new OrderServiceImpl1();
    }

    @Bean(name = "b")
    public OrderService orderServiceByProvider2 () {
        return new OrderServiceImpl2();
    }

    public static void main (String... strings) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(
            ConstBasedDI_WireByType.class);
        OrderServiceClient bean = context.getBean(ConstBasedDI_WireByType.OrderServiceClient.class);
        bean.showPendingOrderDetails();
    }
}
```

```
@Component
public static class OrderServiceClient {

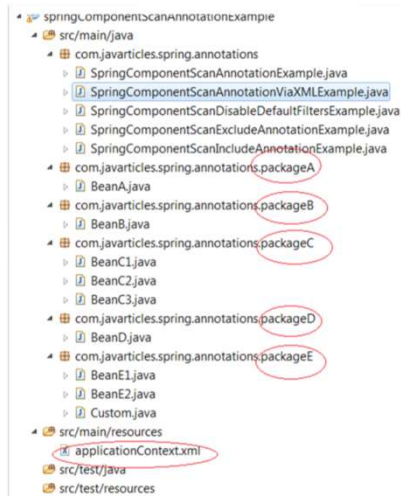
    private final OrderService orderService1;
    private final OrderService orderService2;

    @Autowired
    OrderServiceClient (@Qualifier("a") OrderService orderService1,
        @Qualifier("b") OrderService orderService2) {
        this.orderService1 = orderService1;
        this.orderService2 = orderService2;
    }

    public void showPendingOrderDetails () {
        System.out.println(orderService1.getOrderDetails("100"));
        System.out.println(orderService2.getOrderDetails("100"));
    }
}
```

8

@ComponentScan



```
package com.javarticles.spring.annotations;

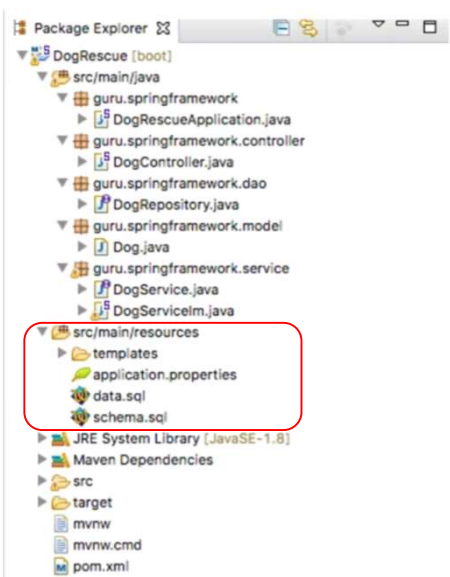
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.javarticles.spring.annotations.packageC.BeanC1;

@Configuration
@ComponentScan(basePackages = {
    "com.javarticles.spring.annotations.packageA",
    "com.javarticles.spring.annotations.packageB",
    "com.javarticles.spring.annotations.packageE" }, basePackageClasses = BeanC1.class)
public class SpringComponentScanAnnotationExample {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
        try {
            ctx.register(SpringComponentScanAnnotationExample.class);
            ctx.refresh();
            System.out.println("SpringComponentScanAnnotationExample: " +
                ctx.getBean("springComponentScanAnnotationExample"));
            System.out.println("BeanA: " +
                ctx.getBean("beanA"));
            System.out.println("BeanB: " +
                ctx.getBean("beanB"));
            System.out.println("BeanC1: " +
                ctx.getBean("beanC1"));
            System.out.println("BeanC2: " +
                ctx.getBean("beanC2"));
            System.out.println("Contains BeanC3: " +
                ctx.containsBean("beanC3"));
            System.out.println("Contains BeanD: " +
                ctx.containsBean("beanD"));
            System.out.println("Contains BeanE1: " +
                ctx.containsBean("beanE1"));
            System.out.println("Contains BeanE2: " +
                ctx.containsBean("beanE2"));
        } finally {
            ctx.close();
        }
    }
}
```

9

Load properties file



- ▶ Application.properties is a default, auto-load file, no special treatment required

- ▶ Foo.properties - put in src/main/resources (classpath)

To load:

@Configuration

@PropertySource{

 "classpath: foo.properties"

 "classpath: foo-\${env}.properties"

}

public class PropertiesWithJavaConfig {}

- ▶ Inject/Use a property

@Value("\${jdbc.url}")

private String jdbcUrl;

OR::

private Environment env;

String keyValue = env.getProperty(key);

OR::

@ConfigurationProperties("app")

POJO class with each field match the key (app.key) in the properties

10

Question

- ▶ 1. why use spring boot
- ▶ 2. steps to create spring boot project
- ▶ 3. what is auto configuration
- ▶ 4. what is starter
- ▶ 5. spring boot version and java version

11

Example

- ▶ Build a Spring boot project with Models corresponding to the database table structures for clinical project
- ▶ Include the view components into spring boot project and finish controller and service layer

12