

spring-boot-starter-example (0.0.1-SNAPSHOT)

Maksim Kostromin

Version 0.0.1-SNAPSHOT, 2018-06-27 12:32:13 UTC

Table of Contents

1. Introduction	2
2. Implementation	3
2.1. hello-service	3
2.2. hello-service-autoconfigure	4
2.3. spring-boot-starter-hello	5
3. Testing	9
4. Links	10
5. Other links	11

Travis CI status: [\[Build Status\]](#)

Chapter 1. Introduction

spring-boot magic...

Read github pages [reference documentation](#)

generated by [generator-jvm](#) yeoman generator (java-spring-boot)

Chapter 2. Implementation

To create starter we need minimal projects structure:

- service with functionality we wanna expose / integrate: `hello-service`
- module which will be automatically configure that service as far starter was added as a dependency: `hello-service-autoconfigure`
- starter module, containing everything needed (hello-service + auto-config + other dependencies): `spring-boot-starter-hello`



we will be using and testing out starter in project: `spring-boot-starter-hello-tests`

2.1. hello-service

First, create service with functionality you wanna to share with the world

```
mkdir hello-service
touch hello-service/pom.xml
# ...
```

That module contains `HelloService` we wanna expose:

HelloService interface:

```
/**
 * Super complex greeting service!
 */
public interface HelloService {

    /**
     * Some javadoc...
     * @param whom who, we salute?
     * @return greeting message
     */
    String sayHello(final String whom);
}
```

HelloServiceImpl interface:

```
@RequiredArgsConstructor
public class HelloServiceImpl implements HelloService {

    final String prefix;
    final String suffix;

    @Override
    public String sayHello(String whom) {
        return format("%s %s%s", prefix, whom, suffix);
    }
}
```

2.2. hello-service-autoconfigure

Next, create auto-configuration module for hello-service

```
mkdir hello-service-autoconfigure
touch hello-service-autoconfigure/pom.xml
# ...
```

That module will depend on `hello-service` module and spring-boot auto-configuration dependencies

file hello-service-autoconfigure/pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-autoconfigure</artifactId>
  </dependency>

  <dependency>
    <optional>true</optional>
    <groupId>com.github.daggerok</groupId>
    <artifactId>hello-service</artifactId>
  </dependency>
</dependencies>
```

Here we are creating auto-configuration which is basically will be picked up if `HelloService` class in classpath

file ./hello-service-

autoconfigure/src/main/java/com/github/daggerok/hello/config/HelloServiceAutoConfiguration.java:

```
/**
 * Apply auto configuration only if {@link HelloService} class is in classpath.
 */
@Configuration
@RequiredArgsConstructor
@ConditionalOnClass(HelloService.class)
@EnableConfigurationProperties(HelloProperties.class)
public class HelloServiceAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public HelloService helloService(final HelloProperties properties) {
        final Hello prop = properties.getHello();
        final HelloServiceImpl helloService = new HelloServiceImpl(prop.getPrefix(), prop
.getSuffix());
        return helloService;
    }
}
```

To make it happens, we need provide `spring.factories` file, which spring-boot will identify and create needed auto-configurations for us if starter in classpath according to conditions

file ./hello-service-

autoconfigure/src/main/java/com/github/daggerok/hello/config/HelloServiceAutoConfiguration.java:

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.github.daggerok.hello.config.HelloServiceAutoConfiguration
```

2.3. spring-boot-starter-hello

Now we are ready to go create starter itself

```
mkdir spring-boot-starter-hello
touch spring-boot-starter-hello/pom.xml
# ...
```

That starter will define in dependencies everything needed

file `spring-boot-starter-hello/pom.xml`:

```
<artifactId>spring-boot-starter-hello</artifactId>
<packaging>jar</packaging>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>com.github.daggerok</groupId>
    <artifactId>hello-service</artifactId>
  </dependency>

  <dependency>
    <groupId>com.github.daggerok</groupId>
    <artifactId>hello-service-autoconfigure</artifactId>
  </dependency>
</dependencies>
```

that module also has auto-configuration `HelloStarterAutoConfiguration.java`:

```
@ConditionalOnClass(HelloServiceAutoConfiguration.class)
@ImportAutoConfiguration({ HelloServiceAutoConfiguration.class })
@Configuration
public class HelloStarterAutoConfiguration { }
```

and `spring.factories` file:

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.github.daggerok.starter.HelloStarterAutoConfiguration
```

it's very important test your auto configuration if it's properly works:

```
public class HelloStarterAutoConfigurationTests {
```



```

private ConfigurableApplicationContext context;

public static class EnvironmentTestUtils {
    public static void addEnvironment(final ConfigurableApplicationContext context,
final String... pairs) {
        TestPropertyValues.of(pairs).applyTo(context);
    }
}

private void load(Class<?> config, String... environment) {
    final AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext();
    ctx.register(config);
    EnvironmentTestUtils.addEnvironment(ctx, environment);
    ctx.refresh();
    this.context = ctx;
}

@Rule
public ExpectedException expectedException = ExpectedException.none();

@Rule
public OutputCapture output = new OutputCapture();

@After
public void closeContext() {
    Optional.ofNullable(context)
        .ifPresent(ConfigurableApplicationContext::close);
}

@Configuration
@ImportAutoConfiguration(HelloStarterAutoConfiguration.class)
static class EmptyConfiguration { }

@Test
public void serviceBeanWithEmptyContextIsAutoConfigured() {
    load(EmptyConfiguration.class);

    assertThat(context.getBeansOfType(HelloService.class))
        .hasSize(1);

    // default prefix: 'Hello, ', default suffix: '!'
    assertThat(context.getBean(HelloService.class).sayHello("Maksimko"))
        .isEqualTo("Hello, Maksimko!");
}

@Configuration
@ImportAutoConfiguration(HelloStarterAutoConfiguration.class)
public static class UserConfiguration {

```

```
@Bean
public HelloService helloService() {
    return new HelloServiceImpl("<", ">");
}

@Test
public void defaultBeanIsNotCreatingIfUserProvidedOne() {
    load(UserConfiguration.class);

    assertThat(context.getBeansOfType(HelloService.class))
        .hasSize(1);

    assertThat(context.getBean(HelloService.class).sayHello("ololo"))
        .isEqualTo("< ololo >");
}
```



Here we are testing that HelloService bean was properly instantiated and found in application context.

Chapter 3. Testing

To test starter, all you need to do is:

1. create module for it:

```
mkdir spring-boot-starter-hello-tests
touch spring-boot-starter-hello-tests/pom.xml
# ...
```

2. add to your pom.xml started:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>

  <dependency>
    <groupId>com.github.daggerok</groupId>
    <artifactId>spring-boot-starter-hello</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
</dependencies>
```

3. and use it like any other spring-boot starters:

```
@Log4j2
@SpringBootApplication
public class HelloStarterTestApplication {
  public static void main(String[] args) {
    final ConfigurableApplicationContext context = SpringApplication.run
(HelloStarterTestApplication.class, args);
    final HelloService helloService = context.getBean(HelloService.class);
    log.info(() -> helloService.sayHello("ololo-trololo"));
  }
}
```

Chapter 4. Links

- [GitHub repo](#)
- [GitHub pages](#)

Chapter 5. Other links

- [YouTube: It's a kind of magic: under the covers of Spring Boot - Stéphane Nicoll & Andy Wilkinson](#)
- [GitHub: snicoll-demos/hello-service-auto-configuration](#)
- [other GitHub repo](#)