Spring Boot cung cấp một số tiện ích và chú thích để trợ giúp khi thử nghiệm ứng dụng của bạn. Hỗ trợ thử nghiệm được cung cấp bởi hai mô-đun: spring-boot-test chứa các mục cốt lõi và spring-boot-test-autoconfigure hỗ trợ cấu hình tự động cho các thử nghiệm.

Hầu hết các nhà phát triển đều sử dụng spring-boot-starter-test, which imports both Spring Boot test modules as well as JUnit, AssertJ, Hamcrest và một số thư viện hữu ích khác.

Test Scope Dependencies

TIẾNG VIỆT

**Phạm vi kiểm tra phụ thuộc**

* [JUnit](http://junit.org/): The de-facto standard for unit testing Java applications.
* [Spring Test](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/testing.html#integration-testing) & Spring Boot Test: Utilities and integration test support for Spring Boot applications.
* [AssertJ](https://joel-costigliola.github.io/assertj/): A fluent assertion library.
* [Hamcrest](http://hamcrest.org/JavaHamcrest/): A library of matcher objects (also known as constraints or predicates).
* [Mockito](http://mockito.org/): A Java mocking framework.
* [JSONassert](https://github.com/skyscreamer/JSONassert): An assertion library for JSON.
* [JsonPath](https://github.com/jayway/JsonPath): XPath for JSON.

**Testing Spring Applications**

Một trong những ưu điểm chính của tiêm phụ thuộc là nó sẽ làm cho mã của bạn dễ dàng hơn để kiểm tra đơn vị. Bạn có thể khởi tạo các đối tượng bằng cách sử dụng toán tử mới mà không cần liên quan đến Spring. Bạn cũng có thể sử dụng các đối tượng giả thay vì phụ thuộc thực.

…

## Testing Spring Boot Applications

@SpringBootTest annotation thay thế cho spring-test @ContextConfiguration annotation khi sử dụng Spring Boot.

Nó tạo Application-context sử dụng trong test thông qua SpringApplication.

Ngoài @SpringBootTest một số chú thích khác cũng được cung cấp để thử nghiệm các lát cụ thể hơn của một ứng dụng.

Don’t forget to also add @RunWith(SpringRunner.class) to your test, otherwise the annotations will be bỏ qua.

Theo mặc định, @SpringBootTest sẽ không khởi động server. Bạn có thể sử dụng thuộc tính webEnvironment của @SpringBootTest để tinh chỉnh thêm cách chạy thử nghiệm của bạn:

MOCK(Default) : Loads a web ApplicationContextand provides a mock web environment. Server được nhúng không được bắt đầu khi sử dụng chú thích này. Nếu môi trường web không có sẵn trên classpath của bạn, chế độ này sẽ quay trở lại một cách minh bạch để tạo một ApplicationContext không thường xuyên trên web.

Nó có thể được sử dụng kết hợp với @AutoConfigureMockMvc hoặc @AutoConfigureWebTestClient để thử nghiệm dựa trên mô hình ứng dụng web của bạn.

RANDOM\_PORT: Tải một WebServerApplicationContext và cung cấp một môi trường web thực sự. Các máy chủ nhúng được bắt đầu và lắng nghe trên một cổng ngẫu nhiên.

DEFINED\_PORT: Tải một WebServerApplicationContext và cung cấp một môi trường web thực sự. Máy chủ nhúng được bắt đầu và lắng nghe trên một cổng được xác định (từ ứng dụng của bạn.properties hoặc trên cổng mặc định là 8080).

NONE: Tải một ApplicationContext bằng cách sử dụng SpringApplication nhưng không cung cấp bất kỳ môi trường web nào (giả hoặc cách khác).

Nếu test của bạn là @Transactional, nó sẽ quay lại giao dịch ở cuối mỗi phương pháp test theo mặc định. Tuy nhiên, khi sử dụng sắp xếp này với RANDOM\_PORT hoặc DEFINED\_PORT ngầm cung cấp một môi trường servlet thực, máy khách và máy chủ HTTP chạy trong các luồng riêng biệt và, do đó, trong các giao dịch riêng biệt. Bất kỳ giao dịch nào được khởi chạy trên máy chủ đều không quay lại trong trường hợp này.

### Detecting Web Application Type

Nếu Spring MVC có sẵn, một bối cảnh ứng dụng dựa trên MVC thông thường được cấu hình. Nếu bạn chỉ có Spring WebFlux, chúng tôi sẽ phát hiện và định cấu hình ngữ cảnh ứng dụng dựa trên WebFlux thay thế.

Nếu cả hai đều có mặt, Spring MVC sẽ được ưu tiên. Nếu bạn muốn thử nghiệm một ứng dụng web phản ứng trong trường hợp này, bạn phải đặt thuộc tính type spring.main.web-application:

*@RunWith(SpringRunner.class)*

*@SpringBootTest(properties = "spring.main.web-application-type=reactive")*

**public** **class** MyWebFluxTests { ... }

### Detecting Test Configuration

Nếu bạn đã quen thuộc với Spring Test Framework, bạn có thể được sử dụng để sử dụng @ContextConfiguration (classes =…) để xác định Spring @Configuration nào cần tải. Ngoài ra, bạn có thể thường xuyên sử dụng các lớp lồng nhau @Configuration trong bài test của bạn.

Khi test các ứng dụng Spring Boot, điều này thường không được yêu cầu. Chú thích @ \*Test của Spring Boot tự động tìm kiếm cấu hình chính của bạn bất cứ khi nào bạn không xác định rõ ràng một chú thích.

Thuật toán tìm kiếm hoạt động từ gói chứa thử nghiệm cho đến khi nó tìm thấy một lớp được chú thích bằng @SpringBootApplication hoặc @SpringBootConfiguration. Miễn là bạn cấu trúc mã của bạn một cách hợp lý, cấu hình chính của bạn thường được tìm thấy.

Nếu bạn sử dụng chú thích kiểm tra để kiểm tra một phần ứng dụng cụ thể hơn, bạn nên tránh thêm cài đặt cấu hình cụ thể cho một khu vực cụ thể trên lớp ứng dụng của phương thức chính. Cấu hình quét thành phần cơ bản của @SpringBootApplication định nghĩa các bộ lọc loại trừ được sử dụng để đảm bảo việc cắt lát hoạt động như mong đợi. Nếu bạn đang sử dụng chỉ thị @ComponentScan rõ ràng trên lớp @ chú thích SpringBootApplication của mình, hãy lưu ý rằng các bộ lọc đó sẽ bị tắt. Nếu bạn đang sử dụng slicing, bạn nên định nghĩa lại chúng.

Nếu bạn muốn tùy chỉnh cấu hình chính, bạn có thể sử dụng lớp @TestConfiguration lồng nhau. Không giống như lớp @Configuration lồng nhau, sẽ được sử dụng thay cho cấu hình chính của ứng dụng của bạn, lớp @TestConfiguration lồng nhau được sử dụng ngoài cấu hình chính của ứng dụng của bạn.

### Excluding Test Configuration

Nếu ứng dụng của bạn sử dụng tính năng quét thành phần (ví dụ: nếu bạn sử dụng @SpringBootApplication hoặc @ComponentScan), bạn có thể tìm thấy các lớp cấu hình cấp cao nhất mà bạn đã tạo chỉ cho các bài kiểm tra cụ thể vô tình được chọn ở mọi nơi.

Như chúng ta đã thấy trước đó, @TestConfiguration có thể được sử dụng trên một lớp bên trong của một bài kiểm tra để tùy chỉnh cấu hình chính. Khi được đặt trên một lớp cấp cao nhất, @TestConfiguration chỉ ra rằng các lớp trong src / test / java không nên được chọn bằng cách quét. Sau đó, bạn có thể nhập lớp đó một cách rõ ràng ở nơi bắt buộc, như được hiển thị trong ví dụ sau:

*@RunWith(SpringRunner.class)*

*@SpringBootTest*

*@Import(MyTestsConfiguration.class)*

**public** **class** MyTests {

*@Test*

**public** **void** exampleTest() {

...

}

}

Nếu bạn trực tiếp sử dụng @ComponentScan (nghĩa là, không phải thông qua @SpringBootApplication), bạn cần đăng ký TypeExcludeFilter với nó. Xem Javadoc để biết chi tiết.

### Testing with a mock environment

Theo mặc định, @SpringBootTest không khởi động máy chủ. Nếu bạn có thiết bị đầu cuối web mà bạn muốn thử nghiệm với môi trường giả này, bạn có thể cấu hình bổ sung MockMvc như trong ví dụ sau:

*@RunWith(SpringRunner.class)*

*@SpringBootTest*

*@AutoConfigureMockMvc*

**public** **class** MockMvcExampleTests {

*@Autowired*

**private** MockMvc mvc;

*@Test*

**public** **void** exampleTest() **throws** Exception {

**this**.mvc.perform(get("/")).andExpect(status().isOk())

.andExpect(content().string("Hello World"));

}

}