使い方は「pom.xml」に以下の依存関係を追加するだけです。

【pom.xml】依存関係の追加

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

この依存関係を追加してアプリケーションを起動すると、以下のようなログが出力されることを確認してください。

【ターミナル】出力されたログ

```
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/heapdump (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/mappings (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/metrics/{name:.*}]
(略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/metrics (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/info (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/configprops (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/env/{name:.*}] (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/env (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/dump (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/health (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/beans (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/autoconfig (略)
(略) o.s.b.a.e.mvc.EndpointHandlerMapping  : Mapped "{[/trace (略)
```

HTTP でアクセスできる「エンド・ポイント」が表示されています。

*

代表的な「エンド・ポイント」について次の表で説明します。

代表的な「エンド・ポイント」

パス	取得内容
/metrics	「アクセス・カウンタ」や「レスポンス・タイム」「JVM の状態」(「ヒープ」「GC 回数」など) など。
/health	各種「データソース」の「ヘルス・チェック」。
/dump	スレッド・ダンプ。
/configprops	プロパティの設定値。
/env	環境変数やシステム・プロパティ。

*

依存関係を追加した後、再ビルドして、「Cloud Foundry」にデプロイし、このアプリケーションに対して、「ヘルス・チェック」の API を実行してみます。

【ターミナル】「ヘルス・チェック」の API を実行

```
$ curl https://hajiboot-rest-maki.cfapps.io/health -XGET
{"status":"UP","diskSpace":{"status":"UP","total":1056858112,"free":8
94726144,"threshold":10485760},"db":{"status":"UP","database":"MySQL"
,"hello":1}}j
```

データベースの状態が確認できました。

正常に起動している場合は「UP」、障害が発生している場合は「DOWN」が返ります。

*

本書では省略しますが、他の「エンド・ポイント」も活用して、運用時の役に立ててください。

第5章 「Spring Boot」におけるテスト

本書の最後に、「Spring Boot」を用いたテストの方法を説明します。

これまで「Spring Boot」を使ったアプリが「組み込みサーバ」を立ち上げて動作することを見てきました。

Java でテストを書くときは「JUnit」を使うのが一般的です。「JUnit」を使ったテストでサーバを立ち上げ、DB まで接続した結合テストが簡単にできたら便利ではないでしょうか。

「Spring Boot」は、このような「組み込みサーバ」を使った「結合テスト」を実施する仕組みを提供しています。

本章では、このような「結合テスト」の実施方法について説明します。

*

なお、「モック」を利用した単体テストについては本書では扱いません。ここでは説明しませんが、「Spring Framework」にはもともと「MockMVC」[1] という、「サーバを立ち上げなくても Controller をテストできる仕組み」が用意されています。

5.1 「Hello World」アプリの結合テスト

「[1.3] はじめての「Spring Boot」」で最初に作った、「HelloWorld アプリ」のテストを書いてみましょう。

「Spring Initializr」で作ったプロジェクトには、あらかじめ「テスト・コード」の雛形が用意されています。

「src/test/java/com/example/HajibootApplicationTests.java」を開いて、次のコードを書いてください。

HajibootApplicationTests クラス

```
package com.example;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.embedded.LocalServerPort;
import org.springframework.boot.test.context.SpringBootTest;
```

[1] http://docs.spring.io/spring/docs/4.3.2.RELEASE/spring-framework-reference/html/testing.html#spring-mvc-test-framework

```
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;

import static org.assertj.core.api.Assertions.*;

@RunWith(SpringRunner.class) // (1)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RAND
OM_PORT) // (2)
public class HajibootApplicationTests {
	@LocalServerPort // (3)
	int port;
	@Autowired
	TestRestTemplate restTemplate; // (4)

	@Test
	public void contextLoads() {
		ResponseEntity<String> response = restTemplate.getForEntity(
				"http://localhost:" + port, String.class);
                                                        // (5)

		// ResponseEntity<String> response = restTemplate.getFor
Entity("/", String.class);でも可 // (6)

		assertThat(response.getStatusCode()).isEqualTo(HttpStat
us.OK); // (7)
		assertThat(response.getBody()).isEqualTo("Hello Wor
ld!"); // (7)
	}

}
```

プログラム解説

項 番	説 明
(1)	「JUnit」の「@RunWith」に「SpringRunner.class」を指定することで、「JUnit」のテスト内で「Spring」の機能を使うことができる。
(2)	「@SpringBootTest」で「Spring Boot」のテスト機能を有効にする。「webEnvironment」属性に「RANDOM_PORT」を指定することで、空いているポートに「組み込みサーバ」が立ち上がる。
(3)	「@LocalServerPort」を付けることで、(2) で立ち上がったサーバのポート番号をインジェクションできる。
(4)	立ち上げた「組み込みサーバ」にアクセスするための「HTTP クライアント」を用意する。 「TestRestTemplate」は通常の「RestTemplate」と比べて、(a)「エラーが発生しても処理を続行」したり、(b)「Basic 認証の設定が簡単にできる」――など、「テスト用の設定」が追加されている。 「RestTemplate」とほぼ同じインターフェイスをもつが、「RestTemplate」を継承しているわけではない。

(5)	「TestRestTemplate」の「getForEntity」で HTTP の「GET」に相当するリクエストを行なう。 「第二引数」に「レスポンス・ボディ」を、「第三引数」に「シリアライズする型」を指定する。
(6)	「TestRestTemplate」を使用する場合は、「プロトコル :// ホスト名 : ポート番号」を省略してパスから指定できる。
(7)	「ResponseEntity」に「HTTP レスポンス」の「ステータス・コード」や「ヘッダ」「ボディ」が格納されている。 「assertThat」は「AssertJ」のメソッドであり、「流れるインターフェイス」でテスト結果を確認できる。

実 行

このテストを実行してみましょう。

【ターミナル】実行結果

```
2016-08-24 02:59:53.751  INFO 74997 --- [           main] s.b.c.e.t.
TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 0
(http)
2016-08-24 02:59:53.773  INFO 74997 --- [           main] o.apache.
catalina.core.StandardService   : Starting service Tomcat
2016-08-24 02:59:53.775  INFO 74997 --- [           main] org.apac
he.catalina.core.StandardEngine  : Starting Servlet Engine: Apache
Tomcat/8.5.4
2016-08-24 02:59:53.883  INFO 74997 --- [ost-startStop-1] o.a.c.c.C.
[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApp
licationContext
(略)
2016-08-24 02:59:54.833  INFO 74997 --- [           main] s.b.c.e.t
.TomcatEmbeddedServletContainer : Tomcat started on port(s): 56034
(http)
```

「組み込みサーバ」が「56034 番ポート」で立ち上がったことが分かります(ポート番号は毎回変わります)。

このサーバに対して「TestRestTemplate」でアクセスして、テストが成功します。

5.2 「REST API」の結合テスト

次に、「[3.2]「REST Web サービス」の開発」で作った「REST API」をテストしましょう。

基本的には「HelloWorld」の場合と同じです。

HajibootRestApplicationTests クラス

```
package com.example;

import com.example.domain.Customer;
import com.example.repository.CustomerRepository;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;

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

import static org.assertj.core.api.Assertions.*;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RAND
OM_PORT,properties = {"spring.datasource.url:jdbc:h2:mem:customers;
DB_CLOSE_ON_EXIT=FALSE"}) // (1)
public class HajibootRestApplicationTests {
    @Autowired
    CustomerRepository customerRepository; // (2)
    @Autowired
    TestRestTemplate restTemplate;
    Customer customer1;
    Customer customer2;

    // (3)
    @Data
    @JsonIgnoreProperties(ignoreUnknown = true)
    static class Page<T> {
        private List<T> content;
        private int numberOfElements;
    }

    // (4)
    @Before
    public void setUp() {
        customerRepository.deleteAll();
        customer1 = new Customer();
        customer1.setFirstName("Taro");
        customer1.setLastName("Yamada");
        customer2 = new Customer();
        customer2.setFirstName("Ichiro");
        customer2.setLastName("Suzuki");

        customerRepository.save(Arrays.asList(customer1, customer2));
    }

    // (5)
    @Test
    public void testGetCustomers() throws Exception {
        ResponseEntity<Page<Customer>> response = restTemplate.exchange(
                "/api/customers", HttpMethod.GET, null /* body,header */,
                new ParameterizedTypeReference<Page<Customer>>() {
                }); // (6)
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().getNumberOfElements()).isEqualTo(2);

        Customer c1 = response.getBody().getContent().get(0);
        assertThat(c1.getId()).isEqualTo(customer2.getId());
        assertThat(c1.getFirstName()).isEqualTo(customer2.getFirstName());
        assertThat(c1.getLastName()).isEqualTo(customer2.getLastName());

        Customer c2 = response.getBody().getContent().get(1);
        assertThat(c2.getId()).isEqualTo(customer1.getId());
        assertThat(c2.getFirstName()).isEqualTo(customer1.getFirstName());
        assertThat(c2.getLastName()).isEqualTo(customer1.getLastName());
    }

    // (7)
    @Test
    public void testPostCustomers() throws Exception {
        Customer customer3 = new Customer();
        customer3.setFirstName("Nobita");
        customer3.setLastName("Nobi");

        ResponseEntity<Customer> response = restTemplate.exchange("/
api/customers",HttpMethod.POST, new HttpEntity<Object>(customer3)
                                                    /* (8) */, Customer.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
        Customer customer = response.getBody();
        assertThat(customer.getId()).isNotNull();
        assertThat(customer.getFirstName()).isEqualTo(customer3.getFi
rstName());
        assertThat(customer.getLastName()).isEqualTo(customer3.getLas
tName());

        assertThat(restTemplate.exchange("/api/customers", HttpMethod.GET,
 null, new ParameterizedTypeReference<Page<Customer>>() {
                })
                .getBody().getNumberOfElements()).isEqualTo(3);
    }

    // (9)
    @Test
    public void testDeleteCustomers() throws Exception {
        ResponseEntity<Void> response = restTemplate.exchange("/api/
customers/{id}" /* (10) */, HttpMethod.DELETE, null /* body,header
 */, Void.class, Collections.singletonMap("id", customer1.getId()));
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NO_
CONTENT);
```

```
        assertThat(restTemplate.exchange("/api/customers", HttpMethod.GET,
null, new ParameterizedTypeReference<Page<Customer>>() {
                })
                .getBody().getNumberOfElements()).isEqualTo(1);
    }

}
```

プログラム解説

項 番	説 明
(1)	「@SpringBootTest」アノテーションの「properties」属性でテスト用にプロパティを上書き可能。 実際のアプリケーションでは、永続可能なデータベースを使っている場合に、「テストのときだけインメモリのデータベースを使う」という使い方ができる。
(2)	「テスト・データ」を投入するために「CustomerRepository」を使う。
(3)	「全件取得 API の」返り値は「org.springframework.data.domain.Page 型」だが、このクラスは「RestTemplate」で取得できない[2]。そのため、テスト用に「レスポンスの JSON」をマッピングする Java クラスを用意する。この準備は、「REST API」のテストの説明としては本質的ではない。 ここでは、後述のテストで使うフィールド（「content」「numberOfElements」）のみを定義する。レスポンスの JSON には、存在する他のフィールドを無視するために、「@JsonIgnoreProperties(ignoreUnknown = true)」を付ける。
(4)	テストの初期化。 「テスト・データ」を全件削除した後、改めて「テスト・データ」を投入する。 「JUnit」のテスト実行順は不定であるため、状態をリセットする必要がある。
(5)	「全件取得 API」のテストを作る。
(6)	「TestRestTemplate」の汎用リクエストメソッドである「exchange」を使って、テスト用のリクエストを送る。 返り値の型がジェネリクス（汎用）である場合は、「ParameterizedTypeReference」を使って型を特定する。
(7)	「新規作成 API」のテストを作る。
(8)	「HttpEntity」クラスを用いて、「リクエスト・ボディ」を作る。
(9)	「一件削除 API」のテストを作る。
(10)	パス中のパラメータは、「プレース・ホルダ」を利用して、埋めることができる。

*

「REST API」のテストも簡単に作ることができました。

「Spring Boot」を利用すると、早い段階で「End to End テスト」を作りやすくなり、安定したアプリケーション開発が期待できます。

*

画面遷移のあるアプリケーションのテストも同様に作れます。

本書では省略しますが、読者の皆さんは作ってみてください。

[2] デフォルトではマッピング対象のクラスに「デフォルト・コンストラクタ」と「セッター / ゲッター」が必要です。「Page」インターフェイスの実装クラスである「org.springframework.data.domain.PageImpl」クラスはこれらをもたないため、「RestTemplate」側でマッピングすることができません。「Spring Data」側での対応が待たれます。

ノート 「Spring Boot」とは直接関係がありませんが、「REST API」のテストを行なう際に、「RestTemplate」よりも便利な「REST-assured」[3] を紹介します。

「pom.xml」に、以下の依存関係を追加してください。

【pom.xml】「REST-assured」の依存関係の追加

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.0.0</version>
    <scope>test</scope>
</dependency>
```

*

先ほどのテストケースは「REST-assured」を使うと、以下のように書き換えられます。

```
package com.example;

import com.example.domain.Customer;
import com.example.repository.CustomerRepository;
import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.embedded.LocalServerPort;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.test.context.junit4.SpringRunner;

import java.util.Arrays;

import static io.restassured.RestAssured.*;
import static org.hamcrest.CoreMatchers.*;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RAND
OM_PORT,properties = {"spring.datasource.url:jdbc:h2:mem:customers;
DB_CLOSE_ON_EXIT=FALSE"})
public class HajibootRestApplicationTests {
    @Autowired
    CustomerRepository customerRepository;
    @LocalServerPort
    int port;
    Customer customer1;
    Customer customer2;

    @Before
    public void setUp() {
        customerRepository.deleteAll();
        customer1 = new Customer();
        customer1.setFirstName("Taro");
```

[3] https://github.com/rest-assured/rest-assured

```
        customer1.setLastName("Yamada");
        customer2 = new Customer();
        customer2.setFirstName("Ichiro");
        customer2.setLastName("Suzuki");

        customerRepository.save(Arrays.asList(customer1, customer2));
        RestAssured.port = port; // (1)
    }

    @Test
    public void testGetCustomers() throws Exception {
        when().get("/api/customers") // (2)
                .then()
                .statusCode(HttpStatus.OK.value())
                .body("numberOfElements", is(2)) // (3)
                .body("content[0].id", is(customer2.getId()))
                .body("content[0].firstName", is(customer2.getFirstName()))
                .body("content[0].lastName", is(customer2.getLastName()))
                .body("content[1].id", is(customer1.getId()))
                .body("content[1].firstName", is(customer1.getFirstName()))
                .body("content[1].lastName", is(customer1.getLastName()));
    }

    @Test
    public void testPostCustomers() throws Exception {
        Customer customer3 = new Customer();
        customer3.setFirstName("Nobita");
        customer3.setLastName("Nobi");

        given().body(customer3) // (4)
                .contentType(ContentType.JSON)
                .and()
                .when().post("/api/customers")
                .then()
                .statusCode(HttpStatus.CREATED.value())
                .body("id", is(notNullValue()))
                .body("firstName", is(customer3.getFirstName()))
                .body("lastName", is(customer3.getLastName()));

        when().get("/api/customers")
                .then()
                .statusCode(HttpStatus.OK.value())
                .body("numberOfElements", is(3));
    }

    @Test
    public void testDeleteCustomers() throws Exception {
        when().delete("/api/customers/{id}", customer1.getId())
                .then()
                .statusCode(HttpStatus.NO_CONTENT.value());

        when().get("/api/customers")
                .then()
                .statusCode(HttpStatus.OK.value())
                .body("numberOfElements", is(1));

    }
}
```


プログラム解説

項　番	説　明
(1)	「REST-assured」で使う「ポート」を設定。
(2)	「GET」メソッドで「HTTP リクエスト」を送る。
(3)	「body」メソッドでレスポンス JSON のフィールドの値を確認。
(4)	「リクエスト・ボディ」を設定。「RestTemplate」を使った場合より、「REST-assured」のほうが「テスト・コード」の見通しがいい。