

JUnitの利用

JUnitは単体テストのフレームワークで、テストケースをロジックで記述し、Java開発におけるスタンダードな技術です。「spring-boot-starter-test」を依存関係に追加する事で使用することができます。

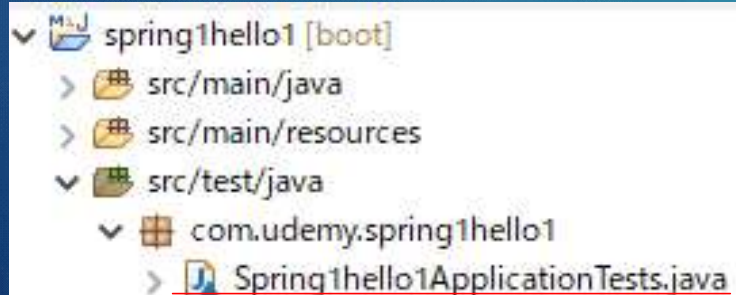
pom.xml

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

説明

Spring Initializerから生成すれば、デフォルトで追加されています。

ソースコード



説明

- src/main/java . . . 作成したクラスを格納する階層
- src/**test**/java . . . 作成したテストクラスを格納する階層
- ✓ テスト対象クラスの同一パッケージ配下にテストクラスを作成
- ✓ テスト対象クラスに対してテストクラスを作成
- ✓ デフォルトでSpringBootApplicationクラスに対応するテストクラスができてい
る (テストの内容は無い)
- ✓ テストクラスの命名は、「テスト対象クラス名 + Test」 となる

spring-boot-starter-test

spring-boot-starter-testを追加する事で使用できるライブラリを一部紹介します

ライブラリ	説明
JUnit5	以下モジュールで構成される、単体テストフレームワーク JUnit Platform : テストフレームワークを起動するための基盤 JUnit Jupiter : JUnit5の事を指し、Test Engineを提供
Spring Test & Spring Boot Test	Spring Bootベースのアプリケーションのテストを実行するために必要なライブラリ群 MockMVCという、Tomcatを起動させない状態でも、Springの動作を再現することができるフレームワークを持つ。
AssertJ	JUnitで値を検証する時に使用するライブラリ

Spring BootのJUnit Testケースの書き方(基本)

@SpringBootTestを利用する事で、SpringBootアプリケーションの検証を行う事ができます。JUnit5の書き方に従い、AssertJを利用した値の検証を記述する事ができます。

ソースコード

```
@SpringBootTest
class Spring3itemApplicationTests {

    @Autowired
    private ItemController itemController;

    // アプリケーションがSpringコンテキストを正常にロードできたかどうかを検
    @Test
    void contextLoads() {
        // AssertJを利用した検証を実装する
        // assertThatの引数に検証の値を入れる
        // 続けてメソッドにて期待値を指定。この場合はNullでないこと。
        assertThat(itemController).isNotNull();
    }
}
```

ポイント

● テストクラスにて、SpringBootの機能が利用できるようになります。コンソール上、SpringBootが起動しているように見えますが、Tomcatサーバは起動していません。

● JUnitがテストケース(テスト項目表の1テストケースに対応するイメージ)として管理します。

● AssertJの機能を用いて、値を検証します。
assertThat(テスト項目).メソッド(期待値) という形式で、テスト項目が期待値通りであることを確認します。左記の例はテスト項目がNULLでないことを確認しています。
期待値と異なっていた場合は、単体テスト項目NGとしてJUnitが管理します。

MockMvcを用いたTestケースの書き方

MockMvcを利用すると、アプリケーションサーバが起動していない状態においても、Spring アプリケーションの動作を再現させる事ができます。Controllerのテストを行う時に利用します。

ソースコード

```
@SpringBootTest
@AutoConfigureMockMvc
class ItemControllerTest {

    @Autowired
    private MockMvc mvc;

    // 検索結果が想定通りであることを確認するテスト
    @Test
    void testGetItem() throws Exception {
        int itemId = 1;
        String responseJsonString = mvc.perform(get("/items/{itemId}", itemId)
            .contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.APPLICATION_JSON_UTF8_VALUE) // 非推奨であるが現時点
            .characterEncoding("UTF-8"))
            .andExpect(status().isOk()) // ステータスコードチェック
            .andReturn().getResponse().getContentAsString();
```

ポイント

@AutoConfigureMockMvcアノテーションを付加すると、MockMvcが自動構成され、@Autowiredアノテーションで利用することができます。

mvc.performメソッドを利用して、HTTP GET等のAPI呼出しを再現し、動作を検証することができます。

httpステータスコードが200(OK)である事を検証することができます。

実機演習の流れ

- 1) spring3itemプロジェクトのSpring3itemApplicationTests.java にテストコードを追加します
- 2) JUnitを実行し、1の結果を確認します
- 3) ItemController.java に対する、JUnitテストクラス ItemControllerTest.javaを作成します
- 4) JUnitを実行し、3の結果を確認します