Red Hat build of Quarkus 1.3

Quarkus Getting Started プロジェクトのテスト

Last Updated: 2020-11-18

Red Hat build of Quarkus 1.3 Quarkus Getting Started プロジェクトのテスト

法律上の通知

概要

本ガイドでは、Quarkus Getting Started プロジェクトをテストする方法について説明します。

目次

前書き

アプリケーション開発者は、Red Hat ビルドの Quarkus を使用して、サーバーレス環境および OpenShift 環境で実行される Java で書かれたマイクロサービスベースのアプリケーションを作成できます。これらのアプリケーションのメモリーフットプリントは小さくなり、起動時間は高速化されます。

本ガイドでは、Apache Maven を使用して JVM モードで Quarkus Getting Started プロジェクトをテストする方法と、リソースをテストにインジェクトする方法を説明します。**『Red Hat ビルドの Quarkus のスタートガイド』** で作成したテストを展開します。

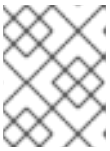
前提条件

- OpenJDK (JDK) 11 がインストールされ、**JAVA_HOME** 環境変数が Java SDK の場所を指定していること。Red Hat ビルドの Open JDK は、Red Hat カスタマーポータルの Software Downloads ページから入手可能です (ログインが必要です)。
- Apache Maven 3.6.2 以降がインストールされていること。Maven は Apache Maven Project の Web サイトから入手できます。
- Quarkus Getting Started プロジェクトを完了していること。
  - Quarkus Getting Started プロジェクトのビルド手順は、**『Red Hat ビルドの Quarkus のスタートガイド』** を参照してください。
  - このチュートリアルで使用する Quarkus Maven プロジェクトの完全なサンプルについては、Quarkus quickstart archive をダウンロードするか、**Quarkus Quickstarts** Git リポジトリーをクローンしてください。この例は **getting-started** ディレクトリーにあります。

第1章 テスト依存関係の確認

このチュートリアルでは、Quarkus Getting Started プロジェクトが完了済みで、プロジェクト **pom.xml** ファイルに **quarkus-junit5** および **rest-assured** 依存関係が含まれている必要があります。Quarkus Getting Started の演習を完了した場合、または完了済みのサンプルをダウンロードした場合に、これらの依存関係が表示されます。

- **quarkus-junit5** 依存関係は、テストフレームワークを制御する **@QuarkusTest** アノテーションを提供するため、テストに必要です。
- **rest-assured** 依存関係は必要ありませんが、HTTP エンドポイントをテストするための便利な方法として使用できます。



注記

Quarkus は正しい URL を自動的に設定するインテグレーションを提供するため、設定は必要ありません。

手順

1. Getting Started プロジェクトの **pom.xml** ファイルを開きます。
2. 以下の依存関係がファイルに含まれていることを確認し、必要な場合は追加します。

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <scope>test</scope>
</dependency>
```

3. **pom.xml** ファイルに **maven-surefire-plugin** が含まれることを確認します。このチュートリアルでは JUnit 5 フレームワークを使用することから、**maven-surefire-plugin** のバージョンを設定する必要があります (デフォルトのバージョンでは JUnit 5 に対応していないため)。

```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${surefire-plugin.version}</version>
    <configuration>
      <systemProperties>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
      </systemProperties>
    </configuration>
</plugin>
```

4. **java.util.logging.manager** システムプロパティーを設定し、テスト用に適切なログマネージャーを使用します。
5. **GreetingResourceTest.java** ファイルに以下の内容が含まれていることを確認し、必要な場合は追加します。

```
package org.acme.quickstart;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import java.util.UUID;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/hello")
          .then()
             .statusCode(200)
             .body(is("hello"));
    }

    @Test
    public void testGreetingEndpoint() {
        String uuid = UUID.randomUUID().toString();
        given()
          .pathParam("name", uuid)
          .when().get("/hello/greeting/{name}")
          .then()
            .statusCode(200)
            .body(is("hello " + uuid));
    }

}
```

6. テストを実行するには、以下のコマンドを入力します。

```
./mvnw clean verify
```

IDE から直接テストを実行することもできます。

注記

このテストでは、HTTP を使用して REST エンドポイントを直接テストします。テストがトリガーされると、テストの実行前にアプリケーションが起動します。

第2章 テストポートの指定

デフォルトでは、Quarkus テストはポート **8081** で実行され、実行中のアプリケーションとの競合が発生しないようにします。これにより、アプリケーションを並行して実行中にテストを実行できます。

手順

- プロジェクトのテスト中に使用するポートを指定するには、プロジェクト **application.properties** ファイルに **quarkus.http.test-port** プロパティーを設定します。**<PORT>** は、テストするポートに置き換えます。

```
quarkus.http.test-port=<PORT>
```

注記

Quarkus は、テストの実行前に RestAssured が使用するデフォルトポートを更新する RestAssured インテグレーションを提供するため、追加の設定は必要ありません。

第3章 テストへの URL のインジェクト

別のクライアントを使用する場合は、Quarkus **@TestHTTPResource** アノテーションを使用して、テスト予定のアプリケーションの URL を test クラスのフィールドに直接インジェクトします。このフィールドには、タイプ **string**、**URL**、または **URI** を使用できます。このアノテーションにテストパスを指定することもできます。この演習では、静的リソースをロードする簡単なテストを作成します。

手順

1. 以下の内容を含む **src/main/resources/META-INF/resources/index.html** ファイルを作成します。

```
<html>
    <head>
        <title>Testing Guide</title>
    </head>
    <body>
        Information about testing
    </body>
</html>
```

2. 以下の内容で **StaticContentTest.java** ファイルを作成し、**index.html** に適切なサービスが提供されていることをテストします。

```
package org.acme.quickstart;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.nio.charset.StandardCharsets;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import io.quarkus.test.common.http.TestHTTPResource;
import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class StaticContentTest {

    @TestHTTPResource("index.html") (1)
    URL url;

    @Test
    public void testIndexHtml() throws Exception {
        try (InputStream in = url.openStream()) {
            String contents = readStream(in);
            Assertions.assertTrue(contents.contains("<title>Testing Guide</title>"));
        }
    }

    private static String readStream(InputStream in) throws IOException {
        byte[] data = new byte[1024];
        int r;
```

```
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        while ((r = in.read(data)) > 0) {
            out.write(data, 0, r);
        }
        return new String(out.toByteArray(), StandardCharsets.UTF_8);
    }
}
```

1 **@TestHTTPResource** アノテーションを使用すると、Quarkus インスタンスの URL を直接インジェクトできます。アノテーションの値は URL の path コンポーネントです。

第4章 CDI BEAN のテストへのインジェクション

ユニットテストを実行し、CDI Bean を直接テストできます。Quarkus により、**@Inject** アノテーションを使用して CDI Bean をテストにインジェクトできます。実際、Quarkus のテストでは完全な CDI Bean であるため、完全な CDI 機能を使用できます。

注記

ネイティブテストでインジェクションを使用することはできません。

手順

- 以下の内容を含む **GreetingServiceTest.java** ファイルを作成します。

```
package org.acme.quickstart;

import javax.inject.Inject;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class GreetingServiceTest {

    @Inject (1)
    GreetingService service;

    @Test
    public void testGreetingService() {
        Assertions.assertEquals("hello Quarkus", service.greeting("Quarkus"));
    }
}
```

(1) **GreetingService** Bean はテストにインジェクトされます。

第5章 テストへのインターセプターの適用

Quarkus テストは完全な CDI Bean であるため、通常通りに CDI インターセプターを適用できます。たとえば、トランザクションのコンテキスト内であるテスト方法を実行する場合は、**@Transactional** アノテーションをその方法に適用できます。また、独自のテストステレオタイプを作成することもできます。

手順

1. **quarkus-narayana-jta** 依存関係を **pom.xml** ファイルに追加します。

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-narayana-jta</artifactId>
</dependency>
```

2. 以下の import ステートメントが **TransactionalQuarkusTest.java** に含まれることを確認してください。

```
package org.acme.quickstart;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.enterprise.inject.Stereotype;
import javax.transaction.Transactional;

import io.quarkus.test.junit.QuarkusTest;
```

3. **@TransactionalQuarkusTest** アノテーションを作成します。

```
@QuarkusTest
@Stereotype
@Transactional
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface TransactionalQuarkusTest {
}
```

4. このアノテーションを、**@QuarkusTest** アノテーションおよび **@Transactional** アノテーションの両方を適用したかのように動作するテストクラスに適用します。

```
@TransactionalQuarkusTest
public class TestStereotypeTestCase {

    @Inject
    UserTransaction userTransaction;

    @Test
    public void testUserTransaction() throws Exception {
        Assertions.assertEquals(Status.STATUS_ACTIVE, userTransaction.getStatus());
```

```
    }

}
```

これは、HTTP を使用せずに greeting サービスを直接評価する簡単なテストです。

第6章 CDI BEAN のモック化

Quarkus により、特定のテスト用に一部の CDI Bean をモック化できます。

以下の方法の 1 つを使用して、オブジェクトをモック化できます。

- **src/test/java** ディレクトリーのクラスでモック化する Bean を上書きし、**@Alternative** および **@Priority(1)** のアノテーションを Bean に配置します。
- **io.quarkus.test.Mock** ステレオタイプアノテーションを使用します。**@Mock** アノテーションには **@Alternative**、**@Priority(1)**、および **@Dependent** のアノテーションが含まれます。

以下の手順では、**@Alternative** アノテーションを使用して外部サービスをモック化する方法を説明します。

手順

1. 以下の例に示すような **src/main/java** ディレクトリーに **ExternalService** を作成します。

```
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class ExternalService {

    public String service() {
        return "external";
    }

}
```

2. **src/main/java** ディレクトリーで **ExternalService** を使用する **UsesExternalService** クラスを作成します。

```
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;

@ApplicationScoped
public class UsesExternalService {

    @Inject
    ExternalService externalService;

    public String doSomething() {
        return externalService.service();
    }
}
```

3. 以下の例に示すような **src/test/java** ディレクトリーにテストを作成します。

```
package org.acme.quickstart;
```

```
import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

@QuarkusTest
class UsesExternalServiceTest {

    @Inject
    UsesExternalService usesExternalService;

    @Test
    public void testDoSomething() {
        Assertions.assertEquals("external", usesExternalService.doSomething());
    }
}
```

4. **@Alternative** アノテーションを使用する **src/test/java** に **MockExternalService** を作成します。

```
package org.acme.quickstart;

import javax.annotation.Priority;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Alternative;

@Alternative
@Priority(1)
@ApplicationScoped
public class MockExternalService extends ExternalService { (1)

    @Override
    public String service() {
        return "mock";
    }
}
```

(1) **ExternalService** が使用されている場所に、**MockExternalService** がインジェクトされます。この例では、**MockExternalService** は、**UsesExternalService** で使用されます。

注記

@Alternative、**@Priority(1)** および **@Dependent** のアノテーションの代わりに **@Mock** アノテーションを使用できます。

以下の例は、**@Mock** アノテーションを使用する **MockExternalService** クラスを作成する方法を示しています。

```
import javax.enterprise.context.ApplicationScoped;

import io.quarkus.test.Mock;

@Mock
@ApplicationScoped
public class MockExternalService extends ExternalService {

    @Override
    public String service() {
        return "mock";
    }
}
```

5. テストで、アサートされた文字列を **"external"** から **"mock"** に変更します。

```
package org.acme.quickstart;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

@QuarkusTest
class UsesExternalServiceTest {

    @Inject
    UsesExternalService usesExternalService;

    @Test
    public void testDoSomething() {
        Assertions.assertEquals("mock", usesExternalService.doSomething());
    }
}
```

第7章 その他のリソース

- Maven を使用した Quarkus アプリケーションの作成に関する詳細は、**『Apache Maven を使用した Quarkus アプリケーションの作成』** を参照してください。
- Red Hat OpenShift Container Platform での Quarkus Maven アプリケーションのデプロイに関する詳細は、**『Red Hat OpenShift Container Platform での Quarkus アプリケーションのデプロイ』** を参照してください。
- Maven Surefire プラグインの詳細は、Apache Maven Project の Web サイトを参照してください。
- JUnit 5 テストフレームワークの詳細は、JUnit 5 の Web サイトを参照してください。
- REST-assured の詳細は、REST-assured の Web サイトを参照してください。

改訂日時: 2020-11-18 01:18:38 UTC