Red Hat Integration 2022.Q1

Camel Extensions for Quarkus のスタートガイド

Camel Extensions for Quarkus のスタートガイド

Last Updated: 2023-03-25

Red Hat Integration 2022.Q1 Camel Extensions for Quarkus のスタートガイド

Camel Extensions for Quarkus のスタートガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

このガイドでは、Camel Extensions for Quarkus を紹介し、Camel Extensions for Quarkus を使用してアプリケーションを作成およびデプロイする各種方法を説明します。

目次

はじめに

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

第1章 CAMEL EXTENSIONS FOR QUARKUS のスタートガイド

このガイドでは、Camel Extensions for Quarkus、プロジェクトを作成するさまざまな方法、および Camel Extensions for Quarkus を使用してアプリケーションの構築を開始する方法を紹介します。

- 「Quarkus の概要のための Camel Extensions」
- 「ツール」
- 「Camel Extensions for Quarkus を使用した最初のプロジェクトのビルド」

注記

Red Hat は、製品に同梱されているコンテンツをホストする Maven リポジトリーを提供します。これらのリポジトリーは、ソフトウェアのダウンロードページからダウンロードできます。

Camel Extensions for Quarkus の場合、次のリポジトリーが必要です。

- rhi-camel-extensions-for-quarkus

このリリースでは、オフラインモードでの Camel Extensions for Quarkus のインストールはサポートされていません。

Camel Quarkus に Apache Maven リポジトリーを使用する方法は、Chapter 2.2 を参照してください。『**Apache Maven を使用した Quarkus アプリケーションの開発およびコンパイル**』の「**Quarkus Maven リポジトリーの** ダウンロードおよび設定」

1.1. QUARKUS の概要のための CAMEL EXTENSIONS

Camel Extensions for Quarkus は、Apache Camel とその vast コンポーネントライブラリーの統合機能を Quarkus ランタイムに提供します。

Camel Extensions for Quarkus を使用する利点には、以下が含まれます。

- ユーザーはパフォーマンス上の利点、Developer Joy、および Quarkus が提供するコンテナーの最初の ethos を利用できるようにします。
- Apache Camel コンポーネントの多くに Quarkus エクステンションが含まれます。
- Camel 3 でパフォーマンスが向上された機能を活用し、メモリーフットプリントが少なくなり、リフレクションへの依存が少なくなり、起動時間が短縮されます。
- Java DSL を使用して Camel ルートを定義できます。

1.2. ツール

重要

Red Hat は、これらの開発者ツールのサポートを提供しません。

1.2.1. IDE プラグイン

Quarkusには、Quarkus言語のサポート、コード/設定補完、プロジェクト作成ウィザードなどを提供する一般的な開発IDE用のプラグインが含まれています。プラグインは、それぞれの IDE マーケットプレイスで利用できます。

- Eclipse プラグイン
- IntelliJ プラグイン
- VSCode プラグイン

プラグインのドキュメントを参照して、希望する IDE のプロジェクトを作成する方法を検出します。

1.2.2. Camel コンテンツアシスト

以下のプラグインは、Camel ルートおよび **application.properties** の編集時にコンテンツアシストをサポートします。

- Camel の VS Code 言語のサポート Camel エクステンションパックの一部
- Camel の Eclipse デスクトップ言語サポート - JBoss Tools と CodeReady Studio の一部
- Apache Camel IDEA プラグイン （常に最新の状態ではありません）
- Language Server Protocol をサポートする他の IDE のユーザーは、Camel Language Server を手動でインストールおよび設定することができます。

1.3. CAMEL EXTENSIONS FOR QUARKUS を使用した最初のプロジェクトのビルド

1.3.1. 概要

code.quarkus.redhat.com を使用して Quarkus Maven プロジェクトを生成し、アプリケーションで使用するエクステンションを自動的に追加および設定できます。

本セクションでは、以下を含む Camel Extensions for Quarkus を使用して Quarkus Maven プロジェクトを作成するプロセスについて説明します。

- code.quarkus.redhat.comを使用したスケルトンアプリケーションの作成
- 簡単な Camel ルートの追加
- アプリケーションコードの使用
- 開発モードでのアプリケーションのコンパイル
- アプリケーションのテスト

1.3.2. スケルトンアプリケーションの生成

プロジェクトのブートストラップおよび生成は、code.quarkus.redhat.com から可能です。Camel Extensions for Quarkus 拡張機能は、見出し「Integration」の下にあります。

'search' フィールドを使用して、必要なエクステンションを見つけます。

使用するコンポーネントエクステンションを選択し、「Generate your application」ボタンをクリックして基本的なスケルトンプロジェクトをダウンロードします。また、プロジェクトを直接 GitHub にプッシュするオプションもあります。

code.quarkus.redhat.com を使用した Quarkus Maven プロジェクトの生成に関する詳細は、**Quarkus スタートガイド** の code.quarkus.redhat.com を使用した Quarkus Maven プロジェクトの作成 を参照してください。

手順

1. **code.quarkus.redhat.com** の Web サイトを使用して、この例の次のエクステンションを選択します。
   - **camel-quarkus-rest**
   - **camel-quarkus-jackson**

注記

本書の一部としてこのタスクを実行するため、上記の手順の最終ステップに記載されているアプリケーションをコンパイルしないでください。

2. 直前の手順で生成されたプロジェクトファイルを展開したディレクトリーに移動します。

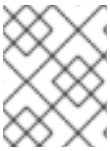
```
$ cd <directory_name>
```

1.3.3. アプリケーションコードの使用

このアプリケーションには、**<dependencyManagement>** でインポートされた **com.redhat.quarkus.platform:quarkus-camel-bom** 内で管理されるコンパイル依存関係が 2 つあります。

pom.xml

```
<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
<quarkus.platform.version>
    <!-- The latest 2.2.x version from
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-bom -->
</quarkus.platform.version>
...
<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>${quarkus.platform.artifact-id}</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>quarkus-camel-bom</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```



注記

BOM 依存関係管理の詳細は、『Developing Applications with Camel Extensions for Quarkus』を参照してください。

アプリケーションは、**src/main/resources/application.properties** 内で定義されたプロパティーで設定できます。たとえば、**camel.context.name** を設定できます。

1.3.4. 簡単な Camel ルートの追加

手順

1. **src/main/java/org/acme/** サブフォルダーに **Routes.java** という名前のファイルを作成します。
2. 以下のコードスニペットのように Camel Rest ルートを追加します。

Routes.java

```
package org.acme;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.CopyOnWriteArrayList;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.rest.RestBindingMode;

import io.quarkus.runtime.annotations.RegisterForReflection;

public class Routes extends RouteBuilder {
    private final List<Fruit> fruits = new CopyOnWriteArrayList<>(Arrays.asList(new
Fruit("Apple")));

    @Override
    public void configure() throws Exception {
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/fruits")
                .get()
                .route()
                .setBody(e -> fruits)
                .endRest()

                .post()
                .type(Fruit.class)
                .route()
                .process().body(Fruit.class, (Fruit f) -> fruits.add(f))
                .endRest();

    }
```

```
    @RegisterForReflection // Let Quarkus register this class for reflection during the native
build
    public static class Fruit {
        private String name;

        public Fruit() {
        }

        public Fruit(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @Override
        public int hashCode() {
            return Objects.hash(name);
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null)
                return false;
            if (getClass() != obj.getClass())
                return false;
            Fruit other = (Fruit) obj;
            return Objects.equals(name, other.name);
        }

    }

}
```

1.3.5. 開発モード

```
$ mvn clean compile quarkus:dev
```

このコマンドでは、プロジェクトのコンパイル、アプリケーションの起動、Quarkus ツールでのワークスペースの変更監視などを行います。プロジェクトの変更は自動的に実行中のアプリケーションに適用されます。

ブラウザーでアプリケーションを確認します（例：**rest-json** 例のhttp://localhost:8080/fruits）。

アプリケーションコードを変更する場合（例： 'Apple' を 'Orange' に変更）、アプリケーションは自動的に更新されます。変更の適用を確認するには、ブラウザーを更新するだけです。

開発モードの詳細は、Quarkus のドキュメントを参照してください。

1.3.6. テスト

1.3.6.1. JVM モード

JVM モードで作成した Camel Rest ルートをテストするには、以下のようにテストクラスを追加します。

手順

1. **src/test/java/org/acme/** サブフォルダーに **RoutesTest.java** という名前のファイルを作成します。
2. 以下のコードスニペットのように **RoutesTest** クラスを追加します。

RoutesTest.java

```
package org.acme;

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

import static io.restassured.RestAssured.given;
import org.hamcrest.Matchers;

@QuarkusTest
public class RoutesTest {

    @Test
    public void testFruitsEndpoint() {

        /* Assert the initial fruit is there */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(1),
                        "name", Matchers.contains("Orange"));

        /* Add a new fruit */
        given()
                .body("{\"name\": \"Pear\"}")
                .header("Content-Type", "application/json")
                .when()
                .post("/fruits")
                .then()
                .statusCode(200);

        /* Assert that pear was added */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(2),
```

```
                "name", Matchers.contains("Orange", "Pear"));
    }

}
```

JVM モードテストは、**Maven** フェーズで **maven-surefire-plugin** によって実行されます。

```
$ mvn clean test
```

1.3.6.2. ネイティブモード

ネイティブモードで作成した Camel Rest ルートをテストするには、以下のようにテストクラスを追加します。

手順

1. **src/test/java/org/acme/** サブフォルダーに **NativeRoutesIT.java** という名前のファイルを作成します。
2. 以下のコードスニペットのように **NativeRoutesIT** クラスを追加します。

NativeRoutesIT.java

```
package org.acme;

import io.quarkus.test.junit.NativeImageTest;

@NativeImageTest
public class NativeRoutesIT extends RoutesTest {

    // Execute the same tests but in native mode.
}
```

ネイティブモードテストは、**verify** フェーズで **maven-failsafe-plugin** を使用して検証されます。**ネイティブ** プロパティーを指定して、テストを実行するプロファイルをアクティベートします。

```
$ mvn clean verify -Pnative
```

1.3.7. アプリケーションのパッケージ化および実行

1.3.7.1. JVM モード

mvn パッケージ は、ストック JVM で実行するためにシン形式の **jar** を作成します。

```
$ mvn clean package
$ ls -lh target/quarkus-app
...
-rw-r--r--. 1 user user 238K Oct 11 18:55  quarkus-run.jar
...
```

これは、以下のように実行できます。

```
$ java -jar target/quarkus-app/quarkus-run.jar
...
[io.quarkus] (main) Quarkus started in 1.163s. Listening on: http://[::]:8080
```

起動時間の周りに 2 回目に着目することに注意してください。

シン形式の **jar** にはアプリケーションコードのみが含まれます。これを実行するには、**target/quarkus-app/lib** の依存関係も必要です。

1.3.7.2. ネイティブモード

注記

ネイティブ実行可能ファイルの準備に関する追加情報は、**Quarkus アプリケーションのネイティブ実行可能ファイルへのコンパイル**ガイドのネイティブ実行可能ファイルの作成を参照してください。

ネイティブ実行可能ファイルを作成するには、以下のコマンドを実行します。

```
$ mvn clean package -Pnative
$ ls -lh target
...
-rwxr-xr-x. 1 user user  46M Oct 11 18:57  code-with-quarkus-1.0.0-SNAPSHOT-runner
...
```

上記の一覧の **ランナー** には **.jar** 拡張がなく、**x** (実行可能) パーミッションが設定されていることに注意してください。そのため、以下を直接実行できます。

```
$ ./target/*-runner
...
[io.quarkus] (main) Quarkus started in 0.013s. Listening on: http://[::]:8080
...
```

アプリケーションはたった 13 ミリ秒以内に起動していることに注意してください。メモリーを効率的に処理する方法を確認するには、以下のコマンドを入力します。

```
$ ps -o rss,command -p $(pgrep code-with)
  RSS COMMAND
65852 ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
```

上記の例では、アプリケーションは 65 MB のメモリーのみを使用します。

ヒント

Quarkus ネイティブの実行可能ガイド には、コンテナーイメージを作成する手順 などの詳細情報が記載されています。

第2章 QUARKUS アプリケーションのデプロイ

以下のビルドストラテジーのいずれかを使用して、Quarkus アプリケーションを OpenShift にデプロイできます。

- Docker ビルド
- S2I バイナリー
- Source S2I

各ビルドストラテジーの詳細は、「 」を参照してください。『OpenShift ビルドストラテジーおよび OpenShift への Quarkus の デプロイ』の「OpenShift ビルドストラテジー」および「Quarkus 」

注記

OpenShift Docker ビルドストラテジーは、JVM 対象の Quarkus アプリケーションおよびネイティブ実行可能ファイルにコンパイルされた Quarkus アプリケーションをサポートする推奨されるビルドストラテジーです。**quarkus.openshift.build-strategy** プロパティーを使用して、デプロイメントストラテジーを設定できます。

第3章 例

以下の表に記載されているクイックスタートの例は、Git リポジトリーの Camel Quarkus Examples からクローンまたはダウンロードできます。

例の数： 2

例	説明
Bindy および FTP を持つファイルコンシューマー	CSV ファイルを消費する方法、データのマーシャリングおよびアンマーシャリング、FTP 経由で送信する方法を示しています。
Kafka の例	Strimzi Operator を使用して Kafka トピックでメッセージを生成および消費する方法を示しています。

3.1. ファイルコンシューマークイックスタートの例の使用

クイックスタートは、Git リポジトリーの Camel Quarkus Examples からダウンロードまたはクローンできます。この例は **file-bindy-ftp** ディレクトリーにあります。

zip ファイルの内容を抽出するか、リポジトリーのクローンをローカルディレクトリー (例: **quickstarts**) に展開します。

このサンプルは、ローカルマシンのコマンドラインで実行できます。開発モードを使用すると、開発段階にてインテグレーションで繰り返しを迅速に行うことができ、コードに関するフィードバックを即座に受け取ることができます。詳細は、Camel Quarkus User Guide の Development mode セクションを参照してください。

注記

コンテナーのリソース制限を設定したり、Quarkus Kubernetes クライアントが自己署名証明書を信頼できるようにする必要がある場合は、これらの設定オプションを **src/main/resources/application.properties** ファイルで確認できます。

前提条件

- OpenShift クラスターにアクセス可能な **cluster admin** 権限が必要です。
- SFTP サーバーにアクセスでき、アプリケーションプロパティー設定ファイル **src/main/resources/application.properties** のサーバープロパティー (**ftp** が最初に付けられる) が設定されている。

手順

1. Maven を使用して、開発モードでサンプルアプリケーションをビルドします。

```
$ cd quickstarts/file-bindy-ftp
$ mvn clean compile quarkus:dev
```

アプリケーションは 10 秒ごとにタイマーコンポーネントをトリガーし、無作為に「ブック」データを生成し、一時ディレクトリーにエントリーを 100 個含めて CSV ファイルを作成します。コンソールに以下のメッセージが表示されます。

> [route1] (Camel (camel-1) thread *#3 - timer://generateBooks) Generating randomized books CSV data*

次に、CSV ファイルはファイルコンシューマーによって読み取られ、Bindy は個別のデータ行を使用して Book オブジェクトにマーシャリングします。

> [route2] (Camel (camel-1) thread *#1 - file:///tmp/books) Reading books CSV data from 89A0EE24CB03A69-0000000000000000*

次に、Book オブジェクトのコレクションが個別の項目に分割され、genre プロパティーに基づいて集約されます。

> [route3] (Camel (camel-1) thread *#0 - AggregateTimeoutChecker) Processed 34 books for genre 'Action'*
> [route3] (Camel (camel-1) thread *#0 - AggregateTimeoutChecker) Processed 31 books for genre 'Crime'*
> [route3] (Camel (camel-1) thread *#0 - AggregateTimeoutChecker) Processed 35 books for genre 'Horror'*

最後に、集約されたブックコレクションは CSV 形式に戻され、テスト FTP サーバーにアップロードされます。

> [route4] (Camel (camel-1) thread *#2 - seda://processed) Uploaded books-Action-89A0EE24CB03A69-0000000000000069.csv*
> [route4] (Camel (camel-1) thread *#2 - seda://processed) Uploaded books-Crime-89A0EE24CB03A69-0000000000000069.csv*
> [route4] (Camel (camel-1) thread *#2 - seda://processed) Uploaded books-Horror-89A0EE24CB03A69-0000000000000069.csv*

2. JVM モードでアプリケーションを実行するには、以下のコマンドを入力します。

```
$ mvn clean package -DskipTests
$ java -jar target/*-runner.jar
```

3. 以下のコマンドを入力して、サンプルアプリケーションをビルドして OpenShift にデプロイできます。

```
$ mvn clean package -DskipTests -Dquarkus.kubernetes.deploy=true
```

4. Pod が実行されていることを確認します。

```
$oc get pods

NAME                                        READY   STATUS      RESTARTS   AGE
camel-quarkus-examples-file-bindy-ftp-1-d72mb    1/1     Running     0          5m15s
ssh-server-deployment-5f6f685658-jtr9n           1/1     Running     0          5m28s
```

5. オプション： 以下のコマンドを入力してアプリケーションログを監視します。

```
oc logs -f camel-quarkus-examples-file-bindy-ftp-5d48f4d85c-sjl8k
```

関連情報

- Camel Extensions for Quarkus によるアプリケーション開発
- Camel Quarkus User ガイド