



Red Hat build of Apache Camel Extensions for Quarkus 2.7

Camel Extensions for Quarkus によるアプリ ケーション開発

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

Red Hat build of Apache Camel Extensions for Quarkus 2.7 Camel Extensions for Quarkus によるアプリケーション開発

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

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

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Developing_Applications_with_Camel_Extensions_for_Quarkus.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本ガイドは、Camel Extensions for Quarkus の上に Camel アプリケーションを作成する開発者向けです。

目次

はじめに	3
多様性を受け入れるオープンソースの強化	3
第1章 CAMEL EXTENSIONS FOR QUARKUS を使用したアプリケーションの開発の概要	4
第2章 依存関係の管理	5
2.1. 新規プロジェクトを起動する QUARKUS ツール	5
第3章 CAMEL ルートの定義	7
3.1. JAVA DSL	7
3.1.1. エンドポイント DSL	7
第4章 設定	8
4.1. CAMEL コンポーネントの設定	8
4.1.1. application.properties	8
4.1.2. CDI	8
4.1.2.1. @Named コンポーネントインスタンスの生成	9
4.2. 規則による設定	10
4.3. メータリングラベル	10
第5章 CAMEL QUARKUS の CONTEXTS AND DEPENDENCY INJECTION (CDI)	11
5.1. CAMELCONTEXT へのアクセス	11
5.2. CDI および CAMEL BEAN コンポーネント	12
5.2.1. 名前による Bean の参照	12
第6章 可観測性	13
6.1. ヘルス/LIVENESS チェック	13
6.2. メトリクス	13
第7章 ネイティブモード	14
7.1. 文字エンコーディング	14
7.2. LOCALE	14
7.3. ネイティブ実行可能ファイルへのリソースの埋め込み	14
7.4. ネイティブモードでの ONEXCEPTION 句の使用	15
7.5. リフレクションのためのクラスの登録	15
7.6. シリアル化のためのクラスの登録	15

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 を使用したアプリケーションの開発の概要

本ガイドは、Camel Extensions for Quarkus の上に Camel アプリケーションを作成する開発者向けです。

Camel Extensions for Quarkus でサポートされる Camel コンポーネントには、関連する Camel Extensions for Quarkus エクステンションがあります。このディストリビューションでサポートされる Camel Extensions for Quarkus エクステンションに関する詳細は、[Camel Extensions for Quarkus](#) リファレンスガイドを参照してください。

第2章 依存関係の管理

2.1. 新規プロジェクトを起動する QUARKUS ツール

特定の Camel Extensions for Quarkus リリースは、特定の Quarkus リリースでのみ動作するはずです。

新規プロジェクトで適切な依存関係バージョンを取得する最も簡単で分かりやすい方法は、Quarkus ツールのいずれかを使用することです。

- [code.quarkus.redhat.com](https://code.quarkus.io): オンラインプロジェクトジェネレーター
- [Quarkus Maven プラグイン](#)

これらのツールを使用すると、エクステンションを選択し、新しい Maven プロジェクトの基礎を固めることができます。

生成される **pom.xml** は以下のようになります。

```
<project>
...
<properties>
  <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>
  ...
</properties>
<dependencyManagement>
  <dependencies>
    <!-- The BOMs managing the dependency versions -->
    <dependency>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>quarkus-bom</artifactId>
      <version>${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>
  </dependencies>
</dependencyManagement>

<dependencies>
  <!-- The extensions you chose in the project generator tool -->
  <dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-sql</artifactId>
    <!-- No explicit version required here and below -->
```

```

    </dependency>
    ...
</dependencies>
...
</project>

```

ヒント

利用可能なエクステンションの領域は、Quarkus Core、Camel Quarkus、およびその他のサードパーティー参加プロジェクト (Hazelcast、Cassandra、Kogito、OptaPlanner など) にまたがるものです。

BOM は Bill of Materials を指します。この **pom.xml** の主目的は、アーティファクトのバージョンを管理することです。これにより、BOM をプロジェクトにインポートするエンドユーザーは、互いに機能するアーティファクトの特定バージョンを気にする必要はありません。つまり、**pom.xml** の **<dependencyManagement>** セクションに BOM をインポートすると、指定の BOM によって管理される依存関係のバージョンを指定する必要はありません。

pom.xml に含まれる特定の BOM は、整合性のある BOM の最小セットを選択するよう設定されたジェネレーターツールを使用して選択するエクステンションによって異なります。

pom.xml ファイルのいずれかの BOM で管理されていないエクステンションを後で追加する場合、適切な BOM を手動で検索する必要はありません。**quarkus-maven-plugin** を使用すると、エクステンションを選択でき、ツールは必要に応じて適切な BOM を追加します。**quarkus-maven-plugin** を使用して、BOM バージョンをアップグレードすることもできます。

com.redhat.quarkus.platform BOM は相互に調整されるため、アーティファクトが複数の BOM で管理されている場合は、常に同じバージョンで管理されることになります。これには、アプリケーション開発者が、独立した各種プロジェクトからの個々のアーティファクトの互換性に注意する必要がないという利点があります。

第3章 CAMEL ルートの定義

Camel Extensions for Quarkus は、Camel ルートを定義する Java DSL 言語をサポートします。

3.1. JAVA DSL

org.apache.camel.builder.RouteBuilder を拡張し、そこで利用可能な Fluent Builder メソッドを使用するのが、Camel ルートを定義する最も一般的な方法です。以下は、タイマーコンポーネントを使用したルートの簡単な例です。

```
import org.apache.camel.builder.RouteBuilder;

public class TimerRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("timer:foo?period=1000")
            .log("Hello World");
    }
}
```

3.1.1. エンドポイント DSL

Camel 3.0 以降、Fluent ビルダーを使用して Camel エンドポイントを定義することもできます。以下は前述の例と同等です。

```
import org.apache.camel.builder.RouteBuilder;
import static org.apache.camel.builder.endpoint.StaticEndpointBuilders.timer;

public class TimerRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from(timer("foo").period(1000))
            .log("Hello World");
    }
}
```



注記

すべての Camel コンポーネントの Builder メソッドは **camel-quarkus-core** で利用できますが、ルートが適切に機能するように、指定のコンポーネントのエクステンションを依存関係として追加する必要があります。上記の例では、**camel-quarkus-timer** になります。

第4章 設定

Camel Quarkus は、デフォルトでは Quarkus アプリケーションライフサイクルに応じて開始/停止される Camel Context Bean を自動的に設定およびデプロイします。設定ステップは、Quarkus の拡張フェーズ中のビルド時に実行され、Camel Quarkus 固有の **quarkus.camel.*** プロパティを使用して調整できる Camel Quarkus エクステンションによって実行されます。



注記

quarkus.camel.* 設定プロパティは、個別のエクステンションページに記載されています ([Camel Quarkus Core](#) 等を参照)。

設定が完了すると、最小の Camel Runtime がアセンブルされ、**RUNTIME_INIT** フェーズで起動します。

4.1. CAMEL コンポーネントの設定

4.1.1. application.properties

プロパティによって Apache Camel のコンポーネントおよびその他の要素を設定するには、アプリケーションが **camel-quarkus-core** に直接、または推移的な推移的に依存するようにしてください。ほとんどの Camel Quarkus エクステンションは **camel-quarkus-core** に依存するため、通常は明示的に追加する必要はありません。

camel-quarkus-core は、Camel Main から Camel Quarkus への機能を提供します。

以下の例では、**application.properties** 経由で **LogComponent** に特定の **ExchangeFormatter** を設定します。

```
camel.component.log.exchange-formatter =
#class:org.apache.camel.support.processor.DefaultExchangeFormatter
camel.component.log.exchange-formatter.show-exchange-pattern = false
camel.component.log.exchange-formatter.show-body-type = false
```

4.1.2. CDI

CDI を使用して、コンポーネントをプログラムで設定することもできます。

推奨の方法は、**ComponentAddEvent** を監視し、ルートおよび **CamelContext** を起動する前にコンポーネントを設定することです。

```
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.event.Observes;
import org.apache.camel.quarkus.core.events.ComponentAddEvent;
import org.apache.camel.component.log.LogComponent;
import org.apache.camel.support.processor.DefaultExchangeFormatter;

@ApplicationScoped
public static class EventHandler {
    public void onComponentAdd(@Observes ComponentAddEvent event) {
        if (event.getComponent() instanceof LogComponent) {
            /* Perform some custom configuration of the component */
            LogComponent logComponent = ((LogComponent) event.getComponent());
```

```

DefaultExchangeFormatter formatter = new DefaultExchangeFormatter();
formatter.setShowExchangePattern(false);
formatter.setShowBodyType(false);
logComponent.setExchangeFormatter(formatter);
    }
}
}

```

4.1.2.1. @Named コンポーネントインスタンスの生成

または、**@Named** プロデューサーメソッドでコンポーネントを作成し、設定できます。これは、Camel がコンポーネントの URI スキームを使用してレジストリーからコンポーネントをルックアップするため機能します。たとえば、**LogComponent** Camel の場合は bean という名前の **log** を検索します。



警告

@Named コンポーネント Bean の生成は通常は機能しますが、一部のコンポーネントで少し問題が発生する可能性があることに注意してください。

Camel Quarkus エクステンションは、以下のいずれかを行う場合があります。

- デフォルトの Camel コンポーネントタイプのカスタムサブタイプを渡します。[Vert.x WebSocket エクステンション](#) の例を参照してください。
- コンポーネントの Quarkus 固有のカスタマイズをいくつか実施します。[JPA エクステンション](#) の例を参照してください。

これらのアクションは、独自のコンポーネントインスタンスの作成時に実行されないため、オブザーバーメソッドでコンポーネントを設定することが推奨される方法です。

```

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

import org.apache.camel.component.log.LogComponent;
import org.apache.camel.support.processor.DefaultExchangeFormatter;

@ApplicationScoped
public class Configurations {
    /**
     * Produces a {@link LogComponent} instance with a custom exchange formatter set-up.
     */
    @Named("log") ❶
    LogComponent log() {
        DefaultExchangeFormatter formatter = new DefaultExchangeFormatter();
        formatter.setShowExchangePattern(false);
        formatter.setShowBodyType(false);

        LogComponent component = new LogComponent();
    }
}

```

```

        component.setExchangeFormatter(formatter);

        return component;
    }
}

```

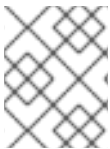
- 1 メソッドの名前が同じであれば、**@Named** アノテーションの **"log"** 引数は省略できます。

4.2. 規則による設定

camel-quarkus-core プロパティによる Camel の設定をサポートする他に、規則を使用して Camel 動作を設定できます。たとえば、CDI コンテナに単一の **ExchangeFormatter** インスタンスがある場合、その Bean を **LogComponent** に自動的に接続します。

4.3. メータリングラベル

OpenShift Container Platform バージョン 4.8 以前では、Metering Operator を使用してラベルを Pod に適用できます。バージョン 4.9 以降は、Metering Operator は直接置き換えなしには利用できなくなりました。



注記

メータリングラベルは、Operator またはテンプレートがデプロイおよび管理する Pod に追加しないでください。

Camel Quarkus は以下のメータリングラベルを使用できます。

- **com.company: Red_Hat**
- **rht.prod_name: Red_Hat_Integration**
- **rht.prod_ver: 2021.Q4**
- **rht.comp: "Camel-Quarkus"**
- **rht.comp_ver: 2.2.0**
- **rht.subcomp: {sub-component-name}**
- **rht.subcomp_t: application**

関連情報

- [OpenShift Container Platform でのメータリングの設定および使用](#)

第5章 CAMEL QUARKUS の CONTEXTS AND DEPENDENCY INJECTION (CDI)

CDI は Quarkus の中心的なロールを果たします。Camel Quarkus は優れたサポートを提供します。

たとえば、**@Inject**、**@ConfigProperty**、および同様のアノテーションを使用して、Bean と設定値を Camel **RouteBuilder** に注入することができます。以下に例を示します。

```
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import org.apache.camel.builder.RouteBuilder;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@ApplicationScoped ❶
public class TimerRoute extends RouteBuilder {

    @ConfigProperty(name = "timer.period", defaultValue = "1000") ❷
    String period;

    @Inject
    Counter counter;

    @Override
    public void configure() throws Exception {
        fromF("timer:foo?period=%s", period)
            .setBody(exchange -> "Incremented the counter: " + counter.increment())
            .to("log:cdi-example?showExchangePattern=false&showBodyType=false");
    }
}
```

- ❶ **@ApplicationScoped** アノテーションは **@Inject** および **@ConfigProperty** が **RouteBuilder** で機能するために必要です。**@ApplicationScoped** Bean は CDI コンテナによって管理され、それらのライフサイクルは、単純な **RouteBuilder** のライフサイクルよりも複雑です。つまり、**RouteBuilder** で **@ApplicationScoped** を使用すると、ブート時にデメリットが生じることがあるため、本当に必要なときにのみ **RouteBuilder** に **@ApplicationScoped** のアノテーションを付ける必要があります。
- ❷ **timer.period** プロパティの値は、サンプルプロジェクトの **src/main/resources/application.properties** で定義されます。

ヒント

詳細は、[Quarkus Dependency Injection ガイド](#) を参照してください。

5.1. CAMELCONTEXT へのアクセス

CamelContext にアクセスするには、Bean に注入します。

```
import javax.inject.Inject;
import javax.enterprise.context.ApplicationScoped;
import java.util.stream.Collectors;
import java.util.List;
```

```
import org.apache.camel.CamelContext;

@ApplicationScoped
public class MyBean {

    @Inject
    CamelContext context;

    public List<String> listRouteIds() {
        return context.getRoutes().stream().map(Route::getId).sorted().collect(Collectors.toList());
    }
}
```

5.2. CDI および CAMEL BEAN コンポーネント

5.2.1. 名前による Bean の参照

名前でルート定義で Bean を参照するには、Bean に **@Named("myNamedBean")** および **@ApplicationScoped**、**@RegisterForReflection** アノテーションは、ネイティブモードでは重要です。

```
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;
import io.quarkus.runtime.annotations.RegisterForReflection;

@ApplicationScoped
@Named("myNamedBean")
@RegisterForReflection
public class NamedBean {
    public String hello(String name) {
        return "Hello " + name + " from the NamedBean";
    }
}
```

次に、ルート定義で **myNamedBean** 名を使用できます。

```
import org.apache.camel.builder.RouteBuilder;
public class CamelRoute extends RouteBuilder {
    @Override
    public void configure() {
        from("direct:named")
            .to("bean:namedBean?method=hello");
    }
}
```


第6章 可観測性

6.1. ヘルス/LIVENESS チェック

ヘルス/liveness チェックは [MicroProfile Health](#) エクステンションでサポートされます。

Camel Health API または Quarkus MicroProfile Health を使用して設定できます。

設定されたチェックはすべて標準の MicroProfile Health エンドポイント URL で利用できます。

- <http://localhost:8080/q/health>
- <http://localhost:8080/q/health/live>
- <http://localhost:8080/q/health/ready>

6.2. メトリクス

メトリクスを公開するための [MicroProfile メトリクス](#) を提供します。

一部の基本的な Camel メトリクスは追加設定なしで提供され、ルートに追加メトリクスを設定して補足できます。

メトリクスは、標準の Quarkus メトリクスエンドポイントで利用できます。

- <http://localhost:8080/q/metrics>

第7章 ネイティブモード

ネイティブモードでのアプリケーションのコンパイルおよびテストについての詳細は、**Compiling your Quarkus applications to native executables** ガイドの [Producing a native executable](#) を参照してください。

7.1. 文字エンコーディング

デフォルトでは、すべての **Charset** がネイティブモードで利用できる訳ではありません。

```
Charset.defaultCharset(), US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16
```

アプリケーションでこのセットに含まれていないエンコーディングが必要な場合や、ネイティブモードで **UnsupportedCharsetException** が出力された場合は、以下のエントリを **application.properties** に追加してください。

```
quarkus.native.add-all-charsets = true
```

Quarkus ドキュメントの [quarkus.native.add-all-charsets](#) も参照してください。

7.2. LOCALE

デフォルトでは、JVM のデフォルトロケールのビルドのみがネイティブイメージに含まれます。Quarkus では、**application.properties** でロケールを設定する方法が提供され、これにより **LANG** および **LC_*** 環境変数に依存しなくても良いようになります。

```
quarkus.native.user-country=US
quarkus.native.user-language=en
```

また、複数のロケールをネイティブイメージに組み込むことや、Mandel コマンドラインオプション - **H:IncludeLocales=fr,en**、**H:+IncludeAllLocales**、および **-H:DefaultLocale=de** を使用してデフォルトのロケールを選択することもサポートされます。これらは、Quarkus **quarkus.native.additional-build-args** プロパティで設定できます。

7.3. ネイティブ実行可能ファイルへのリソースの埋め込み

Class.getResource()、**Class.getResourceAsStream()**、**ClassLoader.getResource()**、**ClassLoader.getResourceAsStream()**などを介してアクセスされるリソースは、ネイティブ実行ファイルに含めるために明示的に一覧表示する必要があります。

これは、以下のように **application.properties** ファイルの Quarkus **quarkus.native.resources.includes** および **quarkus.native.resources.excludes** プロパティを使用して実行できます。

```
quarkus.native.resources.includes = docs/*,images/*
quarkus.native.resources.excludes = docs/ignored.adoc,images/ignored.png
```

上記の例では、**docs/included.adoc** および **images/included.png** という名前のリソースはネイティブ実行可能ファイルに組み込まれ、**docs/ignored.adoc** および **images/ignored.png** のリソースはネイティブの実行ファイルに組み込まれません。

resources.includes および **resources.excludes** は、どちらも Ant-path スタイルの glob パターンのコマ区切りリストです。

詳細は、[Camel Extensions for Quarkus Reference](#) を参照してください。

7.4. ネイティブモードでの ONEXCEPTION 句の使用

ネイティブモードで camel onException 処理を使用する場合、反映させるためにアプリケーション開発者は例外クラスを登録する必要があります。

たとえば、onException 処理のある camel コンテキストは以下のようになります。

```
onException(MyException.class).handled(true);
from("direct:route-that-could-produce-my-exception").throw(MyException.class);
```

リフレクションのために、クラス **mypackage.MyException** を登録する必要があります。詳細は、[リフレクションのためのクラスの登録](#) を参照してください。

7.5. リフレクションのためのクラスの登録

デフォルトでは、動的リフレクションはネイティブモードでは使用できません。リフレクションアクセスが必要なクラスは、コンパイル時にリフレクション用に登録する必要があります。

多くの場合、Quarkus エクステンションはリフレクションを必要とするクラスを検出して自動的に登録できるため、アプリケーション開発者は注意する必要はありません。

ただし、状況によっては、Quarkus のエクステンションはクラスの一部を見逃す場合があるので、アプリケーション開発者が登録を行う必要があります。これには 2 つの方法があります。

1. [@io.quarkus.runtime.annotations.RegisterForReflection](#) アノテーションを使用して、使用するクラスを登録するか、**targets** 属性を使用してサードパーティークラスを登録することもできます。
2. **application.properties** の **quarkus.camel.native.reflection** オプション:

```
quarkus.camel.native.reflection.include-patterns = org.apache.commons.lang3.tuple.*
quarkus.camel.native.reflection.exclude-patterns = org.apache.commons.lang3.tuple.*Triple
```

これらのオプションが適切に機能するには、選択したクラスが含まれるアーティファクトに Jandex インデックス ('META-INF/jandex.idx') が含まれるか、**'application.properties'** の **'quarkus.index-dependency.*'** オプションを使用して、インデックス化のために登録する必要があります。たとえば、

```
quarkus.index-dependency.commons-lang3.group-id = org.apache.commons
quarkus.index-dependency.commons-lang3.artifact-id = commons-lang3
```

7.6. シリアル化のためのクラスの登録

quarkus.camel.native.reflection.serialization-enabled でシリアル化サポートを要求すると、[CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES](#) に一覧表示されているクラスはシリアル化のために自動的に登録されます。

ユーザーは **@RegisterForReflection(serialization = true)** を使用してさらにクラスを登録できます。

