Red Hat Integration 2021.Q3

Camel Quarkus を使用したアプリケーションの開発

テクノロジープレビュー - Camel Quarkus を使用したアプリケーションの開発

Last Updated: 2023-04-25

Red Hat Integration 2021.Q3 Camel Quarkus を使用したアプリケーションの開発

テクノロジープレビュー - Camel Quarkus を使用したアプリケーションの開発

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

概要

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

目次

前書き

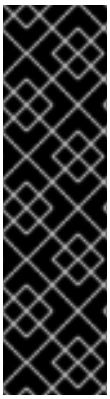
多様性を受け入れるオープンソースの強化

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。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。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 QUARKUS を使用したアプリケーション開発の概要

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

Camel Quarkus でサポートされる Camel コンポーネントには、関連する Camel 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 Quarkus エクステンションに関する詳細は、『Camel Extensions for Quarkus』を参照してください。



重要

Camel Quarkus はテクノロジープレビューの機能です。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat では、これらについて実稼働環境での使用を推奨していません。

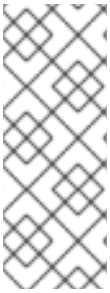
テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

第2章 依存関係の管理

特定の Camel Quarkus リリースは、特定の Quarkus リリースとの動作だけがサポートされます。

2.1. QUARKUS BOM

コア Quarkus 依存関係は、**com.redhat.quarkus.quarkus:quarkus-bom** で管理されます。この BOM は追加の Quarkus エクステンション (Camel Quarkus や Kubernetes など) を管理しないため、インポート **quarkus-bom** が含まれるプロジェクトで使用できる他の BOM もあります。



注記

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 **<depependencyManagement>** 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がありません。

2.2. CAMEL QUARKUS BOM

Quarkus および Camel Quarkus からの依存関係以外を使用する予定がない場合は、サポートされるすべての Camel アーティファクトを管理し、**com.redhat.quarkus.quarkus:quarkus-bom** をインポートする **org.apache.camel.quarkus:camel-quarkus-bom** を使用する必要があります。

警告

本リリースに含まれるサポートされる Camel エクステンションが含まれていないため、このテクノロジープレビュー機能には **com.redhat.quarkus:quarkus-universe-bom** を使用しないでください。

2.3. 他の BOM との統合

camel-quarkus-bom を他の BOM と組み合わせる場合は、インポートの順序が優先順位を定義するため、これらをインポートする順番を慎重に検討してください。

たとえば、**camel-quarkus-bom** の前に **my-foo-bom** がインポートされている場合は、**my-foo-bom** で定義されたアーティファクトバージョンが優先されます。**my-foo-bom** および **camel-quarkus-bom** に重複するアーティファクトがあるかどうかによっては、意図と異なる場合があります。さらに、優先順位の高い **my-foo-bom** のアーティファクトバージョンが、**camel-quarkus-bom** で管理される残りのアーティファクトと互換性があるかどうかを考慮する必要がある場合もあります。

第3章 CAMEL ルートの定義

Camel 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、timer コンポーネントを使用する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 Builder を使用して Camel エンドポイントを定義することも可能です。以下は、前述の例と等価です。

```
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 Quarkus アプリケーションライフサイクルに基づいて起動または起動する Camel Context Bean を自動的に設定し、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.*** プロパティー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.* 設定プロパティーは、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、RUNTIME_INIT フェーズで、最小限の Camel Runtime がアセンブルされ、起動します。

4.1. CAMEL コンポーネントの設定

4.1.1. application.properties

プロパティーによって 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 に機能を提供します。

以下の例では、**application.properties** を使用して **LogComponent** に特定の **ExchangeFormatter** 設定を設定します。

```
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的に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作成

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 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検索する際に機能します。たとえば、**LogComponent** の場合、Camel は **log** の名前が付けられた Bean 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います。

- 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。
- Quarkus 固有のコンポーネントのカスタマイズを実行します。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を作成する際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") 1
    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

プロパティーによる Camel の設定に加え、**camel-quarkus-core** では、規則を使用して Camel の動作を設定することができます。たとえば、CDI コンテナーに単一の **ExchangeFormatter** インスタンスがある場合、その Bean は自動的に **LogComponent** に接続されます。

第5章 CAMEL QUARKUS における CONTEXTS AND DEPENDENCY INJECTION (CDI)

CDI は Quarkus の中心的な役割りを果たし、Camel Quarkus もそれを適切にサポートします。

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 (1)
public class TimerRoute extends RouteBuilder {

    @ConfigProperty(name = "timer.period", defaultValue = "1000") (2)
    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");
    }
}
```

(1) **@Inject** および **@ConfigProperty** が **RouteBuilder** で機能するには、**@ApplicationScoped** アノテーションが必要です。**@ApplicationScoped** Bean は CDI コンテナーによって管理され、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。

(2) **timer.period** プロパティーの値は、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 guide」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、それを Bean に注入するだけです。

```
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

名前でルート定義の Bean を参照するには、Bean に **@Named("myNamedBean")** および **@ApplicationScoped** のアノテーションを付けるだけです。**@RegisterForReflection** アノテーションは、ネイティブモードにとって重要です。

```
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";
    }
}
```

その後、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");
    }
}
```