Red Hat build of Quarkus 1.3

Quarkus アプリケーションの設定

Last Updated: 2020-11-18

法律上の通知

Copyright © 2020 Red Hat, Inc.

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

本ガイドでは、Quarkus アプリケーションの設定方法について説明します。

目次

前書き ... 3

第1章 RED HAT ビルドの QUARKUS 設定オプション ... 4

第2章 設定クイックスタートプロジェクトの作成 ... 5

第3章 QUARKUS アプリケーションの設定ファイルのサンプルの生成 ... 6

第4章 QUARKUS アプリケーションへの設定値のインジェクション ... 7
4.1. @CONFIGPROPERTIES でのクラスのアノテーション 8
4.2. ネストされたオブジェクト設定の使用 10
4.3. @CONFIGPROPERTIES でのインターフェースのアノテーション 11

第5章 コードからの設定へのアクセス ... 14

第6章 設定プロパティーの設定 ... 15

第7章 設定プロファイルの使用 ... 17
7.1. カスタム設定プロファイルの設定 17

第8章 カスタム設定ソースの設定 ... 19

第9章 カスタム設定コンバーターの設定値としての使用 ... 21
9.1. カスタムコンバーターの優先度の設定 21

第10章 YAML 設定サポートの追加 ... 23
10.1. YAML を使用したネストされたオブジェクト設定の使用 23
10.2. YAML を使用したカスタム設定プロファイルの設定 24
10.3. 設定キーの競合の管理 24

第11章 機能テストの更新による設定の変更の検証 ... 26

第12章 QUARKUS アプリケーションのパッケージ化および実行 ... 27

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、Eclipse MicroProfile Config メソッドまたは YAML 形式を使用して Quarkus アプリケーションを設定する方法を説明します。この手順には、Quarkus の **config-quickstart** 演習を使用して作成された設定例が含まれます。

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。
- Maven の設定が、Quarkus Maven リポジトリー のアーティファクトを使用するよう設定されていること。Maven 設定の設定方法は、『Red Hat ビルドの Quarkus のスタートガイド』 を参照してください。

第1章 RED HAT ビルドの QUARKUS 設定オプション

設定オプションを使用すると、1 つの設定ファイルでアプリケーションの設定を変更できます。Quarkus は、必要に応じて関連するプロパティーをグループ化し、プロファイルの切り替えを可能にする設定プロファイルをサポートします。

Eclipse MicroProfile プロジェクトの MicroProfile Config 仕様を使用して、設定プロパティーをアプリケーションにインジェクトし、コードに定義されたメソッドを使用して設定できます。デフォルトでは、Quarkus は **src/main/resources** ディレクトリーにある **application.properties** ファイルからプロパティーを読み取ります。

config-yaml 依存関係をプロジェクト **pom.xml** ファイルに追加することにより、YAML 形式を使用して **application.yaml** ファイルにアプリケーションプロパティーを追加できます。

Quarkus は、さまざまなソースからアプリケーションプロパティーを読み取ることもできます (例: ファイルシステム、データベース、または Java アプリケーションによってロードできるソースなど)。

第2章 設定クイックスタートプロジェクトの作成

config-quickstart プロジェクトでは、Apache Maven および Quarkus Maven プラグインを使用して、簡単な Quarkus アプリケーションを使い始めることができます。以下の手順では、Quarkus Maven プロジェクトの作成方法を説明します。

手順

1. コマンドターミナルで以下のコマンドを入力し、Maven が JDK 11 を使用していること、そして Maven のバージョンが 3.6.2 以上であることを確認します。

```
mvn --version
```

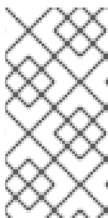
2. 上記のコマンドで JDK 11 が返されない場合は、JDK 11 へのパスを PATH 環境変数に追加し、上記のコマンドを再度入力します。

3. プロジェクトを生成するには、以下のコマンドを入力します。

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=config-quickstart \
    -DplatformGroupId=com.redhat.quarkus \
    -DplatformVersion=1.3.4.Final-redhat-00004 \
    -DclassName="org.acme.config.GreetingResource" \
    -Dpath="/greeting"
cd config-quickstart
```

このコマンドは、**./config-quickstart** ディレクトリーに以下の要素を作成します。

- Maven の構造
- **org.acme.config.GreetingResource** リソース
- アプリケーションの起動後に **http://localhost:8080** でアクセス可能なランディングページ
- **src/main/docker** の **Dockerfile** ファイルの例
- アプリケーション設定ファイル
- 関連するテスト



注記

このチュートリアルで使用する Quarkus Maven プロジェクトは、Quarkus quickstart archive から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。この演習は **config-quickstart** ディレクトリーにあります。

第3章 QUARKUS アプリケーションの設定ファイルのサンプルの生成

すべての利用可能な設定値と、アプリケーションが使用するように設定されたエクステンションのドキュメントで **application.properties.example** ファイルを作成できます。新しいエクステンションのインストール後にこの手順を繰り返し、追加された設定オプションを確認できます。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

- **application.properties.example** ファイルを作成するには、以下のコマンドを入力します。

```
./mvnw quarkus:generate-config
```

  このコマンドにより、**src/main/resources/** ディレクトリーに **application.properties.example** ファイルが作成されます。このファイルには、インストールしたエクステンションで公開されるすべての設定オプションが含まれます。これらのオプションはコメントアウトされ、該当する場合はデフォルト値があります。

  以下の例は、**application.properties.example** ファイルからの HTTP ポート設定エントリーを示しています。

```
#quarkus.http.port=8080
```

その他のリソース

- Quarkus 設定オプションの完全な一覧は Quarkus-All Configuration Options を参照してください。

第4章 QUARKUS アプリケーションへの設定値のインジェクション

Red Hat ビルドの Quarkus は MicroProfile Config 機能 を使用して、設定データをアプリケーションにインジェクトします。コンテキストおよびディペンデンシーインジェクション (CDI) を使用するか、コードに定義されたメソッドを使用して、設定にアクセスできます。

@ConfigProperty アノテーションを使用して、オブジェクトプロパティーをアプリケーションの MicroProfile ConfigSources ファイルのキーにマップできます。この手順では、個別のプロパティー設定を Quarkus **config-quickstart** プロジェクトにインジェクトする方法を説明します。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

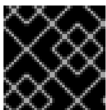
手順

1. **src/main/resources/application.properties** ファイルを開きます。
2. 設定プロパティーを設定ファイルに追加します。**<KEY>** はプロパティー名に、**<VALUE>** はプロパティーの値に置き換えます。

```
<KEY>=<VALUE>
```

以下の例は、Quarkus **config-quickstart** プロジェクトの **greeting.message** プロパティーおよび **greeting.name** プロパティーの値を設定する方法を示しています。

```
greeting.message = hello
greeting.name = quarkus
```



重要

quarkus を Quarkus プロパティーの接頭辞として使用します。

3. **GreetingResource.java** ファイルを確認し、以下の import ステートメントが含まれることを確認します。

```
import org.eclipse.microprofile.config.inject.ConfigProperty;
import java.util.Optional;
```

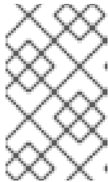
4. 以下の構文でアノテーションを付けて、同等のプロパティーを定義します。

```
@ConfigProperty(name = "greeting.message") (1)
String message;

@ConfigProperty(name = "greeting.suffix", defaultValue="!") (2)
String suffix;

@ConfigProperty(name = "greeting.name")
Optional<String> name; (3)
```

1. このプロパティーの値を指定しないと、アプリケーションは失敗し、以下の例外メッセージをスローします。
2. **greeting.suffix** の値を指定しない場合、Quarkus はこれをデフォルト値に解決します。
3. **Optional** パラメーターに値がない場合は、**greeting.name** に対する値は返されません。



注記

設定した値をインジェクトするには、**@ConfigProperty** を使用できます。**@ConfigProperty** アノテーションが付けられたメンバーには、**@Inject** アノテーションを付ける必要はありません。

5. **hello** メソッドを編集して、以下のメッセージを返します。

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return message + " " + name.orElse("world") + suffix;
}
```

6. 開発モードで Quarkus アプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```

7. エンドポイントがメッセージを返すことを確認するには、新しいターミナルウィンドウに以下のコマンドを入力します。

```
curl http://localhost:8080/greeting
```

このコマンドは、以下の出力を返します。

```
hello quarkus!
```

8. アプリケーションを停止するには、**CTRL+C** を押します。

4.1. @CONFIGPROPERTIES でのクラスのアノテーション

複数の関連する設定値を個別にインジェクトする代わりに、**@io.quarkus.arc.config.ConfigProperties** アノテーションを使用して設定プロパティーをグループ化できます。以下の手順では、Quarkus **config-quickstart** プロジェクトでの **@ConfigProperties** アノテーションの使用方法を説明しています。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. **GreetingResource.java** ファイルを確認し、以下の import ステートメントが含まれることを確認します。

```
package org.acme.config;

import java.util.Optional;
import javax.inject.Inject;
```

2. **src/main/java/org/acme/config** ディレクトリーに **GreetingConfiguration.java** ファイルを作成します。

3. **@ConfigProperties** および **@Optional** の import を **GreetingConfiguration.java** ファイルに追加します。

```
package org.acme.config;

import io.quarkus.arc.config.ConfigProperties;
import java.util.Optional;
```

4. **GreetingConfiguration.java** ファイルに、**greeting** プロパティーの **GreetingConfiguration** クラスを作成します。

```
@ConfigProperties(prefix = "greeting") (1)
public class GreetingConfiguration {

    private String message;
    private String suffix = "!"; (2)
    private Optional<String> name;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getSuffix() {
        return suffix;
    }

    public void setSuffix(String suffix) {
        this.suffix = suffix;
    }

    public Optional<String> getName() {
        return name;
    }

    public void setName(Optional<String> name) {
        this.name = name;
    }
}
```

(1) **prefix** はオプションです。接頭辞を設定しないと、クラス名により決定されます。この例では、**greeting** になります。

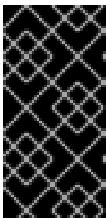
2 **greeting.suffix** が設定されていない場合、**!** がデフォルト値になります。

5. コンテキストおよびディペンデンシーインジェクション (CDI) **@Inject** アノテーションを使用して、属性を **GreetingResource** クラスにインジェクトします。

```
@Inject
GreetingConfiguration greetingConfiguration;
```

6. 開発モードでお使いのアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```



重要

クラスプロパティーの値を指定しない場合、アプリケーションは失敗し、**javax.enterprise.inject.spi.DeploymentException** がスローされ、値が指定されていないことを示します。これは、**Optional** フィールドおよびデフォルト値のフィールドには適用されません。

4.2. ネストされたオブジェクト設定の使用

既存のクラス内でネストされたクラスを定義できます。この手順では、Quarkus **config-quickstart** プロジェクトでネストされたクラスの設定を作成する方法を説明します。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. **GreetingConfiguration.java** ファイルを確認し、以下の import ステートメントが含まれることを確認します。

```
import io.quarkus.arc.config.ConfigProperties;
import java.util.Optional;
import java.util.List;
```

2. **@ConfigProperties** アノテーションを使用して **GreetingConfiguration.java** ファイルに設定を追加します。
以下の例は、**GreetingConfiguration** クラスおよびそのプロパティーの設定について示しています。

```
@ConfigProperties(prefix = "greeting")
public class GreetingConfiguration {

    public String message;
    public String suffix = "!";
    public Optional<String> name;
}
```

3. 以下に示す例のようなネストされたクラス設定を追加します。

```
@ConfigProperties(prefix = "greeting")
public class GreetingConfiguration {

    public String message;
    public String suffix = "!";
    public Optional<String> name;
    public HiddenConfig hidden;

    public static class HiddenConfig {
        public Integer prizeAmount;
        public List<String> recipients;
    }
}
```

この例では、ネストされたクラス **HiddenConfig** を示しています。フィールドの名前 (ここでは **hidden**) が、オブジェクトにバインドされるプロパティーの名前を決定します。

4. 同等の設定プロパティーを **application.properties** ファイルに追加します。
   以下の例は、**GreetingConfiguration** クラスおよび **HiddenConfig** クラスのプロパティーの値を示しています。

```
greeting.message = hello
greeting.name = quarkus
greeting.hidden.prize-amount=10
greeting.hidden.recipients=Jane,John
```

5. 開発モードでお使いのアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```

注記

@ConfigProperties のアノテーションが付けられたクラスは、以下に示す例のような Bean Validation アノテーションを付けることができます。

```
@ConfigProperties(prefix = "greeting")
public class GreetingConfiguration {

    @Size(min = 20)
    public String message;
    public String suffix = "!";
}
```

プロジェクトには **quarkus-hibernate-validator** 依存関係が含まれている必要があります。

4.3. @CONFIGPROPERTIES でのインターフェースのアノテーション

プロパティーを管理する代替方法は、インターフェースとして定義することです。インターフェースに **@ConfigProperties** のアノテーションを付けると、そのインターフェースは他のインターフェースを拡張でき、インターフェース階層全体のメソッドを使用してプロパティーをバインドできます。

この手順では、Quarkus **config-quickstart** プロジェクトのインターフェースとしての **GreetingConfiguration** クラスの実装について説明します。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. **GreetingConfiguration.java** ファイルを確認し、以下の import ステートメントが含まれることを確認します。

```
package org.acme.config;

import io.quarkus.arc.config.ConfigProperties;
import org.eclipse.microprofile.config.inject.ConfigProperty;
import java.util.Optional;
```

2. **GreetingConfiguration** クラスをインターフェースとして **GreetingConfiguration.java** ファイルに追加します。

```
@ConfigProperties(prefix = "greeting")
public interface GreetingConfiguration {

    @ConfigProperty(name = "message") 1
    String message();

    @ConfigProperty(defaultValue = "!")
    String getSuffix(); 2

    Optional<String> getName(); 3
}
```

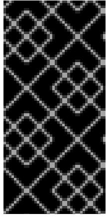
1 設定プロパティーの名前が getter メソッド命名規則に準拠しないため、**@ConfigProperty** アノテーションを設定する必要があります。

2 この例では、**name** は設定されていなかったので、対応するプロパティーが **greeting.suffix** になります。

3 メソッド名は getter メソッドの命名規則 (対応するプロパティーは **greeting.name**) に従い、デフォルト値が必要ないため、**@ConfigProperty** アノテーションを指定する必要はありません。

3. 開発モードでお使いのアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```



重要

interface フィールドの値を指定しないと、アプリケーションが失敗し、**javax.enterprise.inject.spi.DeploymentException** がスローされ、値が指定されていないことを示します。これは、**Optional** フィールドおよびデフォルト値のフィールドには適用されません。

第5章 コードからの設定へのアクセス

コードに定義されたメソッドを使用すると、設定にアクセスできます。CDI Bean リソースまたは JAX-RS リソースではないクラスから、動的ルックアップを実行したり、設定した値を取得したりできます。

org.eclipse.microprofile.config.ConfigProvider.getConfig() メソッドを使用して設定にアクセスできます。**Config object** の **getValue** メソッドは、設定プロパティーの値を返します。

前提条件

Quarkus Maven プロジェクトがあること。

手順

以下のオプションのいずれかを使用して設定にアクセスします。

- **application.properties** ファイルですでに定義されているプロパティーの設定にアクセスするには、以下の構文を使用します。**DATABASE.NAME** は、**databaseName** 変数に割り当てられたプロパティーの名前に置き換えます。

  ```
  String databaseName = ConfigProvider.getConfig().getValue("DATABASE.NAME",
  String.class);
  ```

- **application.properties** ファイルに定義されていない可能性のあるプロパティーの設定にアクセスするには、以下の構文を使用します。

  ```
  Optional<String> maybeDatabaseName =
  ConfigProvider.getConfig().getOptionalValue("DATABASE.NAME", String.class);
  ```

第6章 設定プロパティーの設定

デフォルトでは、Quarkus は **src/main/resources** ディレクトリーにある **application.properties** ファイルからプロパティーを読み取ります。ビルドプロパティーを変更する場合は、アプリケーションを必ず再パッケージしてください。

Quarkus はビルド時にほとんどのプロパティーを設定します。エクステンションはプロパティーを実行時に上書き可能であると定義できます (例: データベース URL、ユーザー名、ターゲット環境に固有のパスワード)。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

1. Quarkus プロジェクトをパッケージ化するには、以下のコマンドを入力します。

```
./mvnw clean package
```

2. 以下のメソッドのいずれかを使用して、設定プロパティーを設定します。

- システムプロパティーの設定
以下のコマンドを入力します。**<KEY>** は追加する設定プロパティーの名前に、**<VALUE>** はプロパティーの値に置き換えます。

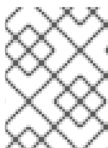
```
java -D<KEY>=<VALUE> -jar target/myapp-runner.jar
```

たとえば、**quarkus.datasource.password** プロパティーの値を設定するには、以下のコマンドを入力します。

```
java -Dquarkus.datasource.password=youshallnotpass -jar target/myapp-runner.jar
```

- 環境変数の設定
以下のコマンドを入力します。**<KEY>** は設定する設定プロパティーの名前に、**<VALUE>** はプロパティーの値に置き換えます。

```
export <KEY>=<VALUE> ; java -jar target/myapp-runner.jar
```

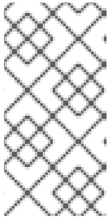


注記

環境変数名は、Eclipse MicroProfile の変換ルールに従います。名前を大文字に変換し、英数字以外の文字をすべてアンダースコア (_) に置き換えます。

- 環境ファイルの使用
現在の作業ディレクトリーに **.env** ファイルを作成し、設定プロパティーを追加します。**<KEY>** はプロパティー名に、**<VALUE>** はプロパティーの値に置き換えます。

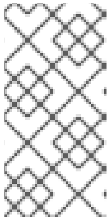
```
<KEY>=<VALUE>
```



注記

開発モードでは、このファイルはプロジェクトの root ディレクトリーに置かれますが、バージョン管理でファイルを追跡しないことが推奨されます。プロジェクトのルートディレクトリーに **.env** ファイルを作成する場合は、プログラムがプロパティーとして読み取るキーおよび値を定義できます。

- **application.properties** ファイルの使用
  アプリケーションが実行される **$PWD/config/application.properties** ディレクトリーに設定ファイルを配置し、そのファイルに定義されたランタイムプロパティーがデフォルト設定を上書きできるようにします。



注記

開発モードで **config/application.properties** 機能を使用することもできます。**target** ディレクトリー内に **config/application.properties** を配置します。ビルドツールからのクリーニング操作 (例: **mvn clean**) は、**config** ディレクトリーも削除します。

第7章 設定プロファイルの使用

お使いの環境に応じて、異なる設定プロファイルを使用できます。設定プロファイルを使用すると、同じファイルに複数の設定を含めることができ、プロファイル名を使用してそれらを選択できます。Red Hat ビルドの Quarkus には 3 つの設定プロファイルがあります。さらに、独自のカスタムプロファイルを作成することもできます。

Quarkus のデフォルトプロファイル

- **dev**: 開発モードでのアクティブ化
- **test**: テストの実行時のアクティブ化
- **prod**: 開発またはテストモードで実行されていない場合のデフォルトプロファイル

前提条件

- Quarkus Maven プロジェクトがあること。

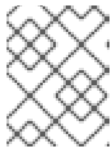
手順

1. Java リソースファイルを開き、以下の import ステートメントを追加します。

```
import io.quarkus.runtime.configuration.ProfileManager;
```

2. 現在の設定プロファイルを表示するには、**ProfileManager.getActiveProfile()** メソッドを呼び出すログを追加します。

```
LOGGER.infof("The application is starting with profile `%s`",
ProfileManager.getActiveProfile());
```



注記

@ConfigProperty("quarkus.profile") メソッドを使用して、現在のプロファイルにアクセスすることはできません。

7.1. カスタム設定プロファイルの設定

設定プロファイルは、必要なだけ作成できます。同じファイルに複数の設定を含めることができ、プロファイル名を使用してそれらを選択できます。

手順

1. カスタムプロファイルを設定するには、**application.properties** ファイルのプロファイル名で設定プロパティーを作成します。**<KEY>** はプロパティー名に、**<VALUE>** はプロパティーの値に、そして **<PROFILE>** はプロファイル名に置き換えます。

```
%<PROFILE>.<KEY>=<VALUE>
```

以下の設定例では、**quarkus.http.port** の値はデフォルトで 9090 で、**dev** プロファイルがアクティブ化されると 8181 になります。

```
quarkus.http.port=9090
%dev.quarkus.http.port=8181
```

2. プロファイルを有効にするには、以下のいずれかの方法を使用します。

   - **quarkus.profile** システムプロパティーの設定

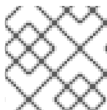
     - **quarkus.profile** システムプロパティーを使用してプロファイルを有効にするには、以下のコマンドを入力します。

       ```
       mvn -D<KEY>=<VALUE> quarkus:<PROFILE>
       ```

   - **QUARKUS_PROFILE** 環境変数の設定

     - 環境変数を使用してプロファイルを有効にするには、以下のコマンドを入力します。

       ```
       export QUARKUS_PROFILE=<PROFILE>
       ```

       注記

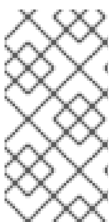
       システムプロパティーの値は環境変数の値よりも優先されます。

3. アプリケーションを再パッケージしてプロファイルを変更するには、以下のコマンドを入力します。

   ```
   ./mvnw package -Dquarkus.profile=<PROFILE>
    java -jar target/myapp-runner.jar
   ```

   以下の例は、**prod-aws** プロファイルをアクティブ化するコマンドを示しています。

   ```
   ./mvnw package -Dquarkus.profile=prod-aws
    java -jar target/myapp-runner.jar
   ```



注記

デフォルトの Quarkus アプリケーションのランタイムプロファイルは、アプリケーションのビルドに使用されるプロファイルに設定されます。Red Hat ビルドの Quarkus は、環境モードに応じてプロファイルを自動的に選択します。たとえば、アプリケーションの実行中は、Quarkus は **prod** モードになります。

第8章 カスタム設定ソースの設定

デフォルトでは、Quarkus は **application.properties** ファイルからプロパティーを読み取ります。ただし、Quarkus は MicroProfile Config 機能をサポートするため、カスタム設定ソースを導入して別のソースから設定を読み込むことができます。

org.eclipse.microprofile.config.spi.ConfigSource インターフェースと **org.eclipse.microprofile.config.spi.ConfigSourceProvider** インターフェースを実装するクラスを提供することで、設定した値にカスタム設定ソースを導入できます。以下の手順では、Quarkus プロジェクトでカスタム設定ソースを実装する方法を説明します。

前提条件

Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. プロジェクト内に **ExampleConfigSourceProvider.java** ファイルを作成し、以下の import を追加します。

```
package org.acme.config;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import org.eclipse.microprofile.config.spi.ConfigSource;
import org.eclipse.microprofile.config.spi.ConfigSourceProvider;
```

2. **ConfigSourceProvider** インターフェースを実装するクラスを作成し、その **getConfigSources** メソッドを上書きして **ConfigSource** オブジェクトの一覧を返すようにします。
以下の例は、カスタムの **ConfigSourceProvider** クラスおよび **ConfigSource** クラスの実装を示しています。

```
public class ExampleConfigSourceProvider implements ConfigSourceProvider {

    private final int times = 2;
    private final String name = "example";
    private final String value = "value";

    @Override
    public Iterable<ConfigSource> getConfigSources(ClassLoader forClassLoader) {
        InMemoryConfigSource configSource = new
InMemoryConfigSource(Integer.MIN_VALUE, "example config source");
        for (int i = 0; i < this.times; i++) {
            configSource.add(this.name + ".key" + (i + 1), this.value + (i + 1));
        }
        return Collections.singletonList(configSource);
    }

    private static final class InMemoryConfigSource implements ConfigSource {

        private final Map<String, String> values = new HashMap<>();
```

```
        private final int ordinal;
        private final String name;

        private InMemoryConfigSource(int ordinal, String name) {
            this.ordinal = ordinal;
            this.name = name;
        }

        public void add(String key, String value) {
            values.put(key, value);
        }

        @Override
        public Map<String, String> getProperties() {
            return values;
        }

        @Override
        public Set<String> getPropertyNames() {
            return values.keySet();
        }

        @Override
        public int getOrdinal() {
            return ordinal;
        }

        @Override
        public String getValue(String propertyName) {
            return values.get(propertyName);
        }

        @Override
        public String getName() {
            return name;
        }
    }
}
```

3. **META-INF/services/** ディレクトリーに **org.eclipse.microprofile.config.spi.ConfigSourceProvider** サービスファイルを作成します。

4. **org.eclipse.microprofile.config.spi.ConfigSourceProvider** ファイルを開き、カスタム ConfigSourceProvider クラスの完全修飾名を追加します。

```
org.acme.config.ExampleConfigSourceProvider
```

5. 開発モードでアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```

アプリケーションを再起動すると、Quarkus はカスタム設定プロバイダーを選択します。

第9章 カスタム設定コンバーターの設定値としての使用

org.eclipse.microprofile.config.spi.Converter<T> を実装し、その完全修飾クラス名を **META-INF/services/org.eclipse.microprofile.config.spi.Converter** ファイルに追加することで、設定値としてカスタムタイプを保存することができます。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. 以下の例に示すように、**META-INF/services/org.eclipse.microprofile.config.spi.Converter** サービスファイルにコンバーターの完全修飾クラス名を含めます。

```
org.acme.config.MicroProfileCustomValueConverter
org.acme.config.SomeOtherConverter
org.acme.config.YetAnotherConverter
```

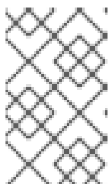
2. converter クラスを実装し、convert メソッドを上書きします。

```
package org.acme.config;

import org.eclipse.microprofile.config.spi.Converter;

public class MicroProfileCustomValueConverter implements
Converter<MicroProfileCustomValue> {

    @Override
    public MicroProfileCustomValue convert(String value) {
        return new MicroProfileCustomValue(Integer.valueOf(value));
    }
}
```



注記

カスタムコンバータークラスは **public** で、引数なしのコンストラクター **public** がなければなりません。カスタムコンバータークラスは **abstract** であってはなりません。

3. カスタムタイプを設定値として使用します。

```
@ConfigProperty(name = "configuration.value.name")
MicroProfileCustomValue value;
```

追加リソース:

- プロパティーファイルのコンテンツを **String** からTyped Java タイプに変換するコンバーターがあります。詳細は、**microprofile-config** GitHub リポジトリーの コンバーターの一覧 を参照してください。

9.1. カスタムコンバーターの優先度の設定

すべての Quarkus コアコンバーターのデフォルト優先度は 200 で、他のすべてのコンバーターは 100 です。ただし、**javax.annotation.Priority** アノテーションを使用してカスタムコンバーターにより高い優先度を設定できます。

以下の手順は、優先度が 150 に割り当てられたカスタムコンバーター **MicroProfileCustomValue** の実装を説明しています。これは、優先度が 100 である **MicroProfileCustomValueConverter** よりも優先されます。

前提条件

- Quarkus **config-quickstart** プロジェクトを作成していること。

手順

1. 以下の import ステートメントをサービスファイルに追加します。

```
package org.acme.config;

import javax.annotation.Priority;
import org.eclipse.microprofile.config.spi.Converter;
```

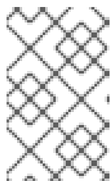
2. クラスに **@Priority** アノテーションを付け、優先度の値を渡して、カスタムコンバーターの優先度を設定します。

```
@Priority(150)
public class MyCustomConverter implements Converter<MicroProfileCustomValue> {

    @Override
    public MicroProfileCustomValue convert(String value) {

        final int secretNumber;
        if (value.startsFrom("OBF:")) {
            secretNumber = Integer.valueOf(SecretDecoder.decode(value));
        } else {
            secretNumber = Integer.valueOf(value);
        }

        return new MicroProfileCustomValue(secretNumber);
    }
}
```



注記

新しいコンバーターを追加する場合は、これを **META-INF/services/org.eclipse.microprofile.config.spi.Converter** サービスファイルに記載する必要があります。

第10章 YAML 設定サポートの追加

Red Hat ビルドの Quarkus は、Eclipse MicroProfile Config の **SmallRye Config** 実装により YAML 設定ファイルをサポートします。**Quarkus Config YAML** エクステンションを追加し、プロパティーの代わりに YAML を使用して設定することができます。Quarkus は、YAML ファイルの名前に **application.yml** と **application.yaml** を使用することをサポートしています。

YAML 設定ファイルは、**application.properties** ファイルよりも優先されます。推奨される方法は、**application.properties** ファイルを削除し、エラーを回避するために 1 種類のみの設定ファイルを使用することです。

手順

- 以下の方法のいずれかを使用して、プロジェクトに YAML エクステンションを追加します。
  - **pom.xml** ファイルを開き、**quarkus-config-yaml** エクステンションを依存関係として追加します。

    ```
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-config-yaml</artifactId>
    </dependency>
    ```

  - コマンドラインから **quarkus-config-yaml** エクステンションを追加するには、プロジェクトディレクトリーから以下のコマンドを入力します。

    ```
    ./mvnw quarkus:add-extension -Dextensions="quarkus-config-yaml"
    ```

10.1. YAML を使用したネストされたオブジェクト設定の使用

既存のクラス内でネストされたクラスを定義できます。以下の YAML 設定ファイルの例は、YAML 形式を使用して Quarkus アプリケーションでネストされたプロパティーを設定する方法を示しています。

前提条件

- YAML 設定ファイルを読み取ることができる既存の Quarkus プロジェクトがあること。

手順

1. 開発モードで Quarkus を起動するには、Quarkus アプリケーションの **pom.xml** ファイルが含まれるディレクトリーで以下のコマンドを入力します。

   ```
   ./mvnw quarkus:dev
   ```

2. YAML 設定ファイルを開きます。
3. 以下に示す構文のようなネストされたクラス設定を追加します。

```
# YAML supports comments
quarkus:
  datasource:
    url: jdbc:postgresql://localhost:5432/some-database
    driver: org.postgresql.Driver
```

```
    username: quarkus
    password: quarkus

# REST Client configuration property
org:
  acme:
    restclient:
      CountriesService/mp-rest/url: https://restcountries.eu/rest

# For configuration property names that use quotes, do not split the string inside the quotes.
quarkus:
  log:
    category:
      "io.quarkus.category":
        level: INFO
```

10.2. YAML を使用したカスタム設定プロファイルの設定

Red Hat ビルドの Quarkus では、プロファイル依存の設定を作成し、必要に応じてそれらの設定の切り替えを行うことができます。以下の手順は、YAML を使用してプロファイル依存設定を提供する方法を示しています。

前提条件

- YAML 設定ファイルを読み取るように Quarkus プロジェクトが設定されていること。

手順

1. YAML 設定ファイルを開きます。
2. プロファイル依存設定を設定するには、**"%profile"** 構文を使用してキーと値のペアを定義する前にプロファイル名を追加します。
   以下の例では、Quarkus が開発モードで実行される際に、**jdbc:postgresql://localhost:5432/some-database** URL で PostgreSQL データベースが利用可能になるように設定されます。

```
"%dev":
  quarkus:
    datasource:
      url: jdbc:postgresql://localhost:5432/some-database
      driver: org.postgresql.Driver
      username: quarkus
      password: quarkus
```

3. アプリケーションを停止した場合には、以下のコマンドを入力して再起動します。

```
./mvnw quarkus:dev
```

10.3. 設定キーの競合の管理

YAML などの構造化形式は、Configuration 名前空間の候補のサブセットのみをサポートします。以下の手順は、2 つの設定プロパティー (**quarkus.http.cors** と **quarkus.http.cors.methods**) における競合の解決方法を示しています。ここで、1 つのプロパティーはもう 1 つのプロパティーの接頭辞になりま

す。

前提条件

- YAML 設定ファイルを読み取るように Quarkus プロジェクトが設定されていること。

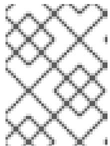
手順

1. YAML 設定ファイルを開きます。
2. YAML プロパティーを別のプロパティーの接頭辞として定義するには、以下の例に示すようにプロパティーの範囲にティルデ (~) を追加します。

```
quarkus:
  http:
    cors:
      ~: true
      methods: GET,PUT,POST
```

3. 開発モードで Quarkus アプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw quarkus:dev
```



注記

YAML キーは設定プロパティー名のアセンブリーに含まれないため、任意のレベルの競合する設定キーに使用できます。

第11章 機能テストの更新による設定の変更の検証

アプリケーションの機能をテストする前に、アプリケーションのエンドポイントに加えた変更を反映するように機能テストを更新する必要があります。以下の手順では、Quarkus **config-quickstart** プロジェクトで **testHelloEndpoint** メソッドを更新する方法を説明します。

手順

1. **GreetingResourceTest.java** ファイルを開きます。
2. **testHelloEndpoint** メソッドの内容を更新します。

```
package org.acme.config;

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

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

@QuarkusTest
public class GreetingResourceTest {

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

}
```

第12章 QUARKUS アプリケーションのパッケージ化および実行

Quarkus プロジェクトをコンパイルしたら、JAR ファイルでパッケージ化し、コマンドラインから実行できます。

前提条件

- Quarkus プロジェクトをコンパイルしていること。

手順

1. Quarkus プロジェクトをパッケージ化するには、**root** ディレクトリーで以下のコマンドを入力します。

   ```
   ./mvnw clean package
   ```

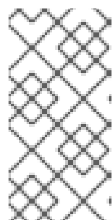
   このコマンドは、以下の JAR ファイルを **/target** ディレクトリーに生成します。

   - **config-quickstart-1.0-SNAPSHOT.jar**: プロジェクトのクラスおよびリソースが含まれます。これは、Maven ビルドで生成される通常のアーティファクトです。
   - **config-quickstart-1.0-SNAPSHOT-runner.jar**: 実行可能な JAR ファイルです。依存関係は **target/lib** ディレクトリーにコピーされるため、このファイルは uber-JAR ファイルではない点に注意してください。

2. 開発モードを実行している場合は、**CTRL+C** を押して開発モードを停止します。停止しないと、ポートの競合が発生します。

3. 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。

   ```
   java -jar target/config-quickstart-1.0-SNAPSHOT-runner.jar
   ```



注記

runner JAR ファイルからの **MANIFEST.MF** ファイルの**Class-Path** エントリーは、**lib** ディレクトリーからの JAR ファイルを明示的に記述します。別の場所からアプリケーションをデプロイする場合は、**runner** JAR ファイルと **lib** ディレクトリーをコピーする必要があります。