Red Hat build of Quarkus 3.8

使用属性文件配置红帽构建的 Quarkus 应用程序

Last Updated: 2024-05-17

Red Hat build of Quarkus 3.8 使用属性文件配置红帽构建的 Quarkus 应用程序

法律通告

摘要

本指南论述了如何使用属性文件配置红帽构建的 Quarkus 应用程序。

目录

提供有关红帽构建的 QUARKUS 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中有问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 使用属性文件配置红帽构建的 QUARKUS 应用程序

作为应用程序开发人员，您可以使用 Red Hat build of Quarkus 创建使用 Java 在 OpenShift 和无服务器环境中运行的基于微服务的应用程序。编译到原生可执行文件的应用程序具有较少的内存占用率和快速启动时间。

您可以使用以下方法之一配置 Quarkus 应用程序：

- 在 **application.properties** 文件中设置属性
- 通过更新 **application.yaml** 文件以 YAML 格式应用结构化配置

您还可以通过执行以下操作来扩展和自定义应用程序的配置：

- 使用属性表达式替换和组合配置属性值
- 使用自定义配置源转换器实施符合 MicroProfile 规范的类，从不同的外部来源读取配置值
- 使用配置配置集为您的开发、测试和生产环境保留一组独立的配置值

该流程包括使用 Quarkus **config-quickstart** exercise 创建的配置示例。

先决条件

- 已安装 OpenJDK 17 或 21，并设置 **JAVA_HOME** 环境变量来指定 Java SDK 的位置。
  - 要下载红帽构建的 OpenJDK，请登录到红帽客户门户网站，再进入 Software Downloads。
- 已安装 Apache Maven 3.8.6 或更高版本。
  - 从 Apache Maven Project 网站下载 Maven。
- 您已将 Maven 配置为使用 Quarkus Maven 存储库中的 工件。
  - 要了解如何配置 Maven 设置，请参阅开始使用 Quarkus。

1.1. 红帽配置选项

您可以使用配置选项在单个配置文件中更改应用程序的设置。Red Hat build of Quarkus 支持用于对相关属性进行分组的配置文件，并根据需要在配置集间切换。

默认情况下，Quarkus 从 **src/main/resources** 目录中的 **application.properties** 文件中读取属性。您还可以将 Quarkus 配置为从 YAML 文件中读取属性。

当您将 **quarkus-config-yaml** 依赖项添加到项目 **pom.xml** 文件时，您可以在 **application.yaml** 文件中配置和管理应用程序属性。如需更多信息，请参阅 添加 YAML 配置支持。

红帽构建的 Quarkus 还支持 MicroProfile Config，可用于从其他来源加载应用的配置。

您可以使用 Eclipse MicroProfile 项目的 MicroProfile Config 规范，将配置属性注入到应用中，并使用代码中定义的方法访问它们。

Quarkus 也可以从不同的来源读取应用程序属性，包括以下源：

- 文件系统
- 数据库

- Kubernetes 或 OpenShift Container Platform **ConfigMap** 或 Secret 对象
- Java 应用程序可以加载的任何源

1.2. 创建配置快速入门项目

使用 **config-quickstart** 项目，您可以使用 Apache Maven 和 Quarkus Maven 插件通过简单的 Quarkus 应用程序启动并运行。以下流程描述了如何创建 Quarkus Maven 项目。

先决条件

- 已安装 OpenJDK 17 或 21，并设置 **JAVA_HOME** 环境变量来指定 Java SDK 的位置。
  - 要下载红帽构建的 OpenJDK，请登录到红帽客户门户网站，再进入 Software Downloads。
- 已安装 Apache Maven 3.8.6 或更高版本。
  - 从 Apache Maven Project 网站下载 Maven。

流程

1. 验证 Maven 使用 OpenJDK 17 或 21，并且 Maven 版本是否为 3.8.6 或更高版本：

```
mvn --version
```

2. 如果 **mvn** 命令没有返回 OpenJDK 17 或 21，请确保在系统上安装 OpenJDK 17 或 21 的目录包含在 **PATH** 环境变量中：

```
export PATH=$PATH:<path_to_JDK>
```

3. 输入以下命令来生成项目：

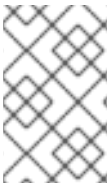
```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.8.4.redhat-00002:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=config-quickstart \
    -DplatformGroupId=com.redhat.quarkus.platform \
    -DplatformVersion=3.8.4.redhat-00002 \
    -DclassName="org.acme.config.GreetingResource" \
    -Dpath="/greeting"
cd config-quickstart
```

验证

前面的 **mvn** 命令在 **config-quickstart** 目录中创建以下项目：

- Maven 项目目录结构
- **org.acme.config.GreetingResource** 资源
- 在启动应用程序后，您可以访问 **http://localhost:8080** 的登录页面
- 用于以原生模式和 JVM 模式测试应用程序的相关单元测试
- **src/main/docker** 子目录中的 **Dockerfile.jvm** 和 **Dockerfile.native** 文件示例

- 应用程序配置文件



注意

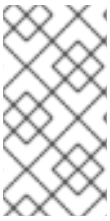
或者，您可以从 Quarkus Quickstarts 归档下载在此教程中使用的 Quarkus Maven 项目，或克隆 Quarkus Quickstarts Git 存储库。Quarkus **config-quickstart** 练习位于 **config-quickstart** 目录中。

1.3. 将配置值注入红帽构建的 QUARKUS 应用程序

红帽构建的 Quarkus 使用 MicroProfile Config 功能将配置数据注入应用。您可以使用上下文和依赖项注入(CDI)或代码中定义方法来访问配置。

使用 **@ConfigProperty** 注释，将对象属性映射到应用的 **MicroProfile Config Sources** 文件中的键。

以下流程和示例演示了如何使用红帽构建的 Quarkus 应用程序配置文件 **application.properties** 将单个属性配置注入 Quarkus **config-quickstart** 项目。



注意

您可以通过与使用 **application.properties** 文件相同的方式使用 MicroProfile Config 配置文件 **src/main/resources/META-INF/microprofile-config.properties**。

使用 **application.properties** 文件是首选的方法。

先决条件

您已创建了 Quarkus **config-quickstart** 项目。

流程

1. 打开 **src/main/resources/application.properties** 文件。
2. 在您的配置文件中添加配置属性，其中 &lt **;property_name&** gt; 是属性 name，**<** value> 是属性值：

```
<property_name>=<value>
```

以下示例演示了如何在 Quarkus **config-quickstart** 项目中为 **greeting.message** 和 **greeting.name** 属性设置值：

application.properties 文件示例

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

重要

当您配置应用程序时，请不要使用字符串 **quarkus** 为特定于应用程序的属性添加前缀。**quarkus** 前缀被保留在框架级别上配置 Quarkus。使用 **quarkus** 作为应用程序特定属性的前缀，可能会导致应用程序运行时出现意外结果。

3. 查看项目中的 **GreetingResource.java** Java 文件。文件包含带有 **hello （）** 方法的 **GreetingResource** 类，该方法在 **/greeting** 端点上发送 HTTP 请求时返回一条消息：

GreetingResource.java 文件示例

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/greeting")
public class GreetingResource {

    String message;
    Optional<String> name;
    String suffix;


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

在提供的示例中，**hello （）** 方法中的 **message** 和 **name** string 的值不会被初始化。当调用端点并在此状态成功启动时，应用会抛出 **NullPointerException**。

4. 定义 **消息**、**name** 和 **suffix** 字段，并使用 **@ConfigProperty** 标注，与您为 **greeting.message** 和 greeting **.name 属性定**义 的值匹配。使用 **@ConfigProperty** 注释来注入每个字符串的配置值。例如：

GreetingResource.java 文件示例

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/greeting")
public class GreetingResource {

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

```
    String message;

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

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

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

1. 如果您没有为 **greeting.message** 字符串配置值，应用程序会失败并抛出以下异常：**jakarta.enterprise.inject.spi.DeploymentException: io.quarkus.runtime.configuration.ConfigurationException: Failed to load config value of type class java.lang.String for: greeting.message**
2. 如果没有为 **greeting.suffix** 配置值，则 Quarkus 会将它解析为默认值。
3. 如果您没有定义 **greeting.name** 属性，则 **name** 的值不可用。即使这个值不可用，您的应用程序仍会运行，因为您在 **名称** 上设置 **Optional** 参数。

注意

要注入配置的值，您可以使用 **@ConfigProperty**。您不需要为使用 **@ConfigProperty** 标注的成员包含 **@Inject** 注释。

5. 以开发模式编译并启动应用程序：

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

6. 在新终端窗口中输入以下命令，以验证端点是否返回消息：

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

这个命令返回以下输出：

```
hello quarkus!
```

7. 要停止应用程序，请按 Ctrl+C。

1.4. 更新功能测试以验证配置更改

在测试应用程序的功能前，您必须更新功能测试，以反映您对应用程序端点所做的更改。以下流程演示了如何在 Quarkus **config-quickstart** 项目中更新 **testHelloEndpoint** 方法。

流程

1. 打开 **GreetingResourceTest.java** 文件。

2. 更新 **testHelloEndpoint** 方法的内容：

```
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
    }

}
```

1.5. 设置配置属性

默认情况下，Quarkus 从 **src/main/resources** 目录中的 **application.properties** 文件中读取属性。如果更改了构建属性，请确保重新打包应用程序。

Quarkus 在构建期间配置大多数属性。扩展可以在运行时定义属性，例如数据库 URL、用户名和密码，这些属性可特定于您的目标环境。

先决条件

- 您已创建了 Quarkus **config-quickstart** 项目。
- 您已在项目的 **application.properties** 文件中定义 **greeting.message** 和 **greeting.name** 属性。

流程

1. 要打包 Quarkus 项目，请输入以下命令：

```
./mvnw clean package
```

2. 使用以下方法之一设置配置属性：

- 设置系统属性：
  输入以下命令，其中 &lt **;property_name** > 是您要添加的配置属性的名称，&lt **;value** > 是属性值：

```
java -D<property_name>=<value> -jar target/quarkus-app/quarkus-run.jar
```

  例如，要设置 **quarkus.datasource.password** 属性的值，请输入以下命令：

```
java -Dgreeting.suffix=? -jar target/quarkus-app/quarkus-run.jar
```

- 设置环境变量：
  输入以下命令，其中 &lt **;property_name** > 是您要设置的配置属性的名称，&lt **;value** > 是属性值：

```
export <property_name>=<value> ; java -jar target/quarkus-app/quarkus-run.jar
```

注意

环境变量名称遵循 Eclipse MicroProfile 的转换规则。将名称转换为大写，并将不是字母数字字符的任何字符替换为下划线(_)。

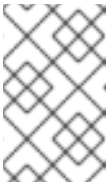
- 使用环境文件：
  在当前工作目录中创建 **.env** 文件并添加配置属性，其中 < **PROPERTY_NAME&** gt; 是属性 name，&lt **;value** > 是属性值：

```
<PROPERTY_NAME>=<value>
```

注意

在开发模式中，此文件位于项目的根目录中。不要跟踪版本控制中的文件。如果您在项目的根目录中创建 **.env** 文件，您可以定义程序读取为属性的键和值。

- 使用 **application.properties** 文件：
  将配置文件放在运行应用程序的 **$PWD/config/application.properties** 目录中，以便该文件中定义的任何运行时属性都覆盖默认配置。



注意

您还可以在开发模式中使用 **config/application.properties** 功能。将 **config/application.properties** 文件放在 **目标** 目录中。来自构建工具的任何清理操作（例如 **mvn clean** ）也会删除 **配置目录**。

1.6. 高级配置映射

以下高级映射流程是特定于红帽构建的 Quarkus 的扩展，且位于 MicroProfile Config 规范之外。

1.6.1. 注解带有 @ConfigMapping 的接口

使用 **@io.smallrye.config.ConfigMapping** 注解来分组配置属性，而不是单独注入多个相关配置值。以下流程描述了如何在 Quarkus **config-quickstart** 项目中使用 **@ConfigMapping** 注释。

先决条件

- 您已创建了 Quarkus **config-quickstart** 项目。
- 您已在项目的 **application.properties** 文件中定义 **greeting.message** 和 **greeting.name** 属性。

流程

1. 查看项目中的 **GreetingResource.java** 文件，并确保该文件包含以下示例中显示的内容。要使用 **@ConfigPoperties** 注释，将另一个配置源的配置属性注入此类，您必须导入 **java.util.Optional** 和 **org.eclipse.microprofile.config.inject.ConfigProperty** 软件包。

   GreetingResource.java 文件示例

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/greeting")
public class GreetingResource {

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

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

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

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

2. 在 **src/main/java/org/acme/config** 目录中创建一个 **GreetingConfiguration.java** 文件。将 **ConfigMapping** 和 **Optional** 的导入语句添加到文件中：

   GreetingConfiguration.java 文件示例

```
package org.acme.config;

import io.smallrye.config.ConfigMapping;
import io.smallrye.config.WithDefault;
import java.util.Optional;

@ConfigMapping(prefix = "greeting") 1
public interface GreetingConfiguration {
    String message();

    @WithDefault("!") 2
    String suffix();
```

```
    Optional<String> name();
}
```

1. **prefix** 属性是可选的。例如，在这种情况下，前缀是 **问候语**。
2. 如果没有设置 **greeting.suffix**，则使用 **!** 作为默认值。

3. 使用 **@Inject** 注释将 **GreetingConfiguration** 实例注入 **GreetingResource** 类，如下所示：

注意

此片段取代了在 **config-quickstart** 项目的初始版本中标上 **@ConfigProperty** 的三个字段。

GreetingResource.java 文件示例

```
package org.acme.config;

import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingConfiguration config;

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

4. 以开发模式编译并启动应用程序：

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

重要

如果没有为类属性提供值，应用程序将无法编译，并返回 **io.smallrye.config.ConfigValidationException** 错误来指示缺少值。这不适用于带有默认值的可选字段或字段。

5. 要验证端点是否返回信息，请在一个新的终端窗口中输入以下命令：

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

6. 您会收到以下信息：

```
hello quarkus!
```

7. 要停止应用程序，请按 Ctrl+C。

1.6.2. 使用嵌套对象配置

您可以定义嵌套在另一个接口中的接口。此流程演示了如何在 Quarkus **config-quickstart** 项目中创建和配置嵌套接口。

先决条件

- 您已创建了 Quarkus **config-quickstart** 项目。
- 您已在项目的 **application.properties** 文件中定义 **greeting.message** 和 **greeting.name** 属性。

流程

1. 查看项目中的 **GreetingResource.java**。文件包含带有 **hello （）** 方法的 **GreetingResource** 类，该方法在 **/greeting** 端点上发送 HTTP 请求时返回一条消息：

GreetingResource.java 文件示例

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/greeting")
public class GreetingResource {

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

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

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

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

2. 使用 **GreetingConfiguration.java** 实例创建一个 **GreetingConfiguration**.java 类文件。此类包含 **GreetingResource** 类中定义的 **hello （）** 方法的外部化配置：

GreetingConfiguration.java 文件示例

```
package org.acme.config;

import io.smallrye.config.ConfigMapping;
import io.smallrye.config.WithDefault;
import java.util.Optional;

@ConfigMapping(prefix = "greeting")
public interface GreetingConfiguration {
    String message();

    @WithDefault("!")
    String suffix();

    Optional<String> name();
}
```

3. 创建嵌套在 **GreetingConfiguration** 实例中的 **ContentConfig** 接口，如下例所示：

GreetingConfiguration.java 文件示例

```
package org.acme.config;

import io.smallrye.config.ConfigMapping;
import io.smallrye.config.WithDefault;

import java.util.List;
import java.util.Optional;

@ConfigMapping(prefix = "greeting")
public interface GreetingConfiguration {
    String message();

    @WithDefault("!")
    String suffix();

    Optional<String> name();

    ContentConfig content();

    interface ContentConfig {
        Integer prizeAmount();

        List<String> recipients();
    }
}
```

注意

ContentConfig 接口的方法名称为 **content**。要确保您将属性绑定到正确的接口，当您为此类定义配置属性时，请使用前缀 **中的内容**。这样，您还可以防止属性名称冲突和意外的应用程序行为。

4. 在 **application.properties** 文件中定义 **greeting.content.prize-amount** 和 **greeting.content.recipients** 配置属性。
以下示例显示了为 **GreetingConfiguration** 实例和 **ContentConfig** 接口定义的属性：

application.properties 文件示例

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

5. 使用 **@Inject** 注释，将 **GreetingConfiguration** 实例注入 **GreetingResource** 类，而不是三个 **@ConfigProperty** 字段注释，如下例所示。另外，您必须更新 **/greeting** 端点返回的消息字符串，并带有您为新的 **greeting.content.prize-amount** 和 **greeting.content.recipients** 属性设置的值。

GreetingResource.java 文件示例

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import jakarta.inject.Inject;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingConfiguration config;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return config.message() + " " + config.name().orElse("world")
                + config.suffix() + "\n" + config.content().recipients()
                + " receive total of candies: "
                + config.content().prizeAmount();
    }
}
```

6. 以开发模式编译并启动应用程序：

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

重要

如果没有为类属性提供值，应用程序将无法编译，您会收到一个 **jakarta.enterprise.inject.spi.DeploymentException** 异常来指示缺少值。这不适用于带有 **默认值的** Optional 字段和字段。

7. 要验证端点是否返回信息，请打开一个新的终端窗口并输入以下命令：

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

这时将显示一条消息，其中包含两行输出。第一行显示问候语，第二行报告 prize 收件人以及奖品量，如下所示：

```
hello quarkus!
Jane,John receive total of candies: 10
```

8. 要停止应用程序，请按 Ctrl+C。

注意

您可以使用类似以下示例的 bean 验证注解标注了 **@ConfigMapping** 的类：

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

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

您的项目必须包含 **quarkus-hibernate-validator** 依赖项。

1.7. 以编程方式访问配置

您可以在代码中定义方法，以检索应用中的配置属性值。这样，您可以动态查找配置属性值，或者从 CDI Bean 或 Jakarta REST （以前称为 JAX-RS）资源的类中检索配置属性值。

您可以使用 **org.eclipse.microprofile.config.ConfigProvider.getConfig （）** 方法访问配置。**config** 对象的 **getValue （）** 方法返回配置属性的值。

先决条件

- 您有一个 Quarkus Maven 项目。

流程

- 使用方法访问应用程序代码中任何类或对象的配置属性值。根据您要检索的值是否在项目中的配置源中设置，您可以使用以下方法之一：
  - 要访问项目中配置源中设置的属性值，例如在 **application.properties** 文件中，使用 **getValue （）** 方法：

```
String <variable-name> = ConfigProvider.getConfig().getValue("<property-name>",
<data-type-class-name>.class);
```

例如，要检索带有数据类型 **String** 的 **greeting.message** 属性的值，并分配给代码中的 **message** 变量，请使用以下语法：

```
String message = ConfigProvider.getConfig().getValue("greeting.message",String.class);
```

- 当您要检索是可选的或默认值，且可能没有在应用程序的 **application.properties** 文件或其他配置源中定义时，请使用 **getOptionalValue （）** 方法：

```
Optional_<String>_ <variable-name> =
ConfigProvider.getConfig().getOptionalValue("<property-name>", <data-
type-class-name>.class);
```

例如，要检索 **greeting.name** 属性的值，它是可选的，具有数据类型 **String**，并分配给代码中的 **name** 变量，请使用以下语法：

```
Optional_<String>_ name =
ConfigProvider.getConfig().getOptionalValue("greeting.name", String.class);
```

以下片段显示了使用 programmatic 访问配置，使用上述 **GreetingResource** 类的变体：

src/main/java/org/acme/config/GreetingResource.java

```
package org.acme.config;

import java.util.Optional;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.Config;
import org.eclipse.microprofile.config.ConfigProvider;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/greeting")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        Config config = ConfigProvider.getConfig();
        String message = config.getValue("greeting.message", String.class);
        Optional_<String>_ suffix = config.getOptionalValue("greeting.suffix", String.class).orElse("!");
        Optional_<String>_ name = config.getOptionalValue("greeting.name", String.class);

        return message + " " + name.orElse("world") + suffix;
    }
}
```

1.8. 属性表达式

您可以将属性引用和文本字符串合并到属性表达式中，并将这些表达式用作 Quarkus 配置中的值。

与变量一样，属性表达式会动态替换配置属性值，避免了硬编码的值。

您可以在一个配置源中扩展表达式，并在另一个配置源中定义一个值。

当 **java.util.Properties** 从配置源读取属性值时，应用程序会解析属性表达式：在编译时从编译时读取，并在运行时覆盖（如果在该点被覆盖）。

如果应用无法解析表达式中属性值，并且属性没有默认值，则应用会抛出 **NoSuchElementException** 错误。

1.8.1. 属性表达式的使用示例

要在配置 Quarkus 应用程序时获得灵活性，您可以使用属性表达式，如下例所示。

- 替换配置属性值：
  为了避免配置中硬编码属性值，您可以使用属性表达式。使用 **${<property_name>}** 语法编写引用配置属性的表达式，如下例所示：

  application.properties 文件示例

  ```
  remote.host=quarkus.io
  callable.url=https://${remote.host}/
  ```

  callable.url 属性的值解析为 **https://quarkus.io/**。

- 设置特定于特定配置配置文件的属性值：
  在以下示例中，**%dev** 配置配置集和默认配置配置集被设置为使用带有不同主机地址的数据源连接 URL。

  application.properties 文件示例

  ```
  %dev.quarkus.datasource.jdbc.url=jdbc:mysql://localhost:3306/mydatabase?useSSL=false
  quarkus.datasource.jdbc.url=jdbc:mysql://remotehost:3306/mydatabase?useSSL=false
  ```

  根据用于启动应用程序的配置配置集，您的数据源驱动程序使用您为配置集设置的数据库 URL。

  您可以通过为每个配置配置集的自定义 **application.server** 属性设置不同的值，从而简化相同的结果。然后，您可以引用应用程序的数据库连接 URL 中的属性，如下例所示：

  application.properties 文件示例

  ```
  %dev.application.server=localhost
  application.server=remotehost

  quarkus.datasource.jdbc.url=jdbc:mysql://${application.server}:3306/mydatabase?
  useSSL=false
  ```

  application.server 属性会根据您在运行应用程序时选择的配置集解析为适当的值。

- 设置属性表达式的默认值：

您可以为属性表达式定义默认值。如果扩展表达式所需的属性值没有从任何配置源解析，则 quarkus 使用默认值。您可以使用以下语法为表达式设置默认值：

```
${<property_name>:<default_value>}
```

在以下示例中，数据源 URL 中的属性表达式使用 **mysql.db.server** 作为 **application.server** 属性的默认值：

application.properties 文件示例

```
quarkus.datasource.jdbc.url=jdbc:mysql://${application.server:mysql.db.server}:3306/mydatabas
e?useSSL=false
```

- 嵌套属性表达式：
  您可以通过在另一个属性表达式中嵌套属性表达式来编写属性表达式。当扩展嵌套属性表达式时，首先扩展内部表达式。您可以使用以下语法嵌套属性表达式：

```
${<outer_property_name>${<inner_property_name>}}
```

- 组合多个属性表达式：
  您可以使用以下语法将两个或多个属性表达式接合在一起：

```
${<first_property_name>}${<second_property_name>}
```

- 将属性表达式与环境变量合并：
  您可以使用属性表达式替换环境变量的值。以下示例中的表达式替换了为 **HOST** 环境变量设置的值作为 **application.host** 属性的值：

  application.properties 文件示例

```
remote.host=quarkus.io
application.host=${HOST:${remote.host}}
```

如果没有设置 **HOST** 环境变量，**application.host** 属性将使用 **remote.host** 属性的值作为默认值。

1.9. 使用配置配置集

您可以根据您的环境使用不同的配置配置集。使用配置配置文件时，您可以在同一个文件中有多个配置，并使用配置集名称来选择它们。

Red Hat build of Quarkus 有以下三个默认配置配置集：

- **Dev** ：在开发模式中激活
- **测试** ：运行测试时激活
- **prod** ：不在开发或测试模式下运行的默认配置集

注意

另外，您可以创建自己的自定义配置集。

先决条件

您有一个 Quarkus Maven 项目。

流程

1. 打开 Java 资源文件并添加以下导入语句：

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

2. 要获取当前配置文件的列表，请通过调用 **ConfigUtils.getProfiles （）** 方法来添加日志：

```
LOGGER.infof("The application is starting with profiles `%s`", ConfigUtils.getProfiles());
```

1.9.1. 设置自定义配置配置集

您可以根据需要创建多个配置配置集。同一文件中可以有多个配置，您可以使用配置集名称来选择配置。

流程

1. 要设置自定义配置集，请在 **application.properties** 文件中使用配置集名称创建一个配置属性，其中 < **property_name** > 是属性的名称，< **value&gt**; 是属性值，& **lt;profile** > 是配置集的名称：

   创建配置属性

```
%<profile>.<property_name>=<value>
```

   在以下示例配置中，**quarkus.http.port** 的值默认为 **9090**，并在 **dev** 配置集激活时变为 **8181** ：

   配置示例

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

2. 使用以下方法之一启用配置集：

   - 设置 **quarkus.profile** 系统属性。
     - 要使用 **quarkus.profile** 系统属性启用配置集，请输入以下命令：

       使用 quarkus.profile 属性启用配置集

```
mvn -Dquarkus.profile=<value> quarkus:dev
```

   - 设置 **QUARKUS_PROFILE** 环境变量。
     - 要使用环境变量启用配置集，请输入以下命令：

       使用环境变量启用配置集

```
export QUARKUS_PROFILE=<profile>
```

注意

系统属性值优先于环境变量值。

3. 要重新打包应用程序并更改配置集，请输入以下命令：

更改配置集

```
./mvnw package -Dquarkus.profile=<profile>
 java -jar target/quarkus-app/quarkus-run.jar
```

以下示例显示了激活 **prod-aws** 配置集的命令：

激活配置集的命令示例

```
./mvnw package -Dquarkus.profile=prod-aws
 java -jar target/quarkus-app/quarkus-run.jar
```

注意

默认 Quarkus 应用程序运行时配置集被设置为用于构建应用程序的配置集。红帽构建的 Quarkus 根据您的环境模式自动选择配置集。例如，当您的应用程序以 JAR 身份运行时，Quarkus 处于 **prod** 模式。

1.10. 设置自定义配置源

默认情况下，Quarkus 应用从项目的 **src/main/resources** 子目录中的 **application.properties** 文件中读取属性。通过 Quarkus，您可以根据用于外部配置的 MicroProfile 配置规范从其他来源加载应用配置属性。您可以通过定义实现 org.eclipse.microprofile.config. **spi.ConfigSource 和 org.eclipse.microprofile .config.spi.ConfigSource** 的类，从其他来源加载配置属性。此流程演示了如何在 Quarkus 项目中实施自定义配置源。

前提条件

您有 Quarkus **config-quickstart** 项目。

流程

1. 在项目中创建一个类文件，它将实施 **org.eclipse.microprofile.config.spi.ConfigSourceProvider** 接口。要返回 **ConfigSource** 对象列表，您必须覆盖 **getConfigSources （）** 方法。

Example org.acme.config.InMemoryConfigSourceProvider

```
package org.acme.config;

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

import java.util.List;

public class InMemoryConfigSourceProvider implements ConfigSourceProvider {

    @Override
```

```
    public Iterable<ConfigSource> getConfigSources(ClassLoader classLoader) {
    return List.of(new InMemoryConfigSource());
    }
}
```

2. 创建实现 **org.eclipse.microprofile.config.spi.ConfigSource** 接口的 **InMemoryConfigSource** 类：

Example org.acme.config.InMemoryConfigSource

```
package org.acme.config;

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

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

public class InMemoryConfigSource implements ConfigSource {
private static final Map<String, String> configuration = new HashMap<>();


 static {
    configuration.put("my.prop", "1234");
 }

  @Override
  public int getOrdinal() { (1)
    return 275;
  }

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

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

  @Override
  public String getName() {
    return InMemoryConfigSource.class.getSimpleName();
  }
}
```

(1) **getOrdinal （）** 方法返回 **ConfigSource** 类的优先级。因此，当多个配置源定义相同的属性时，Quarkus 可以选择由具有最高优先级的 **ConfigSource** 类定义的相应值。

3. 在项目的 **src/main/resources/META-INF/services/** 子目录中，创建一个名为 **org.eclipse.microprofile.config.spi.ConfigSourceProvider** 的文件，并在您创建的文件中输入实施 **ConfigSourceProvider** 的完全限定名称：

org.eclipse.microprofile.config.spi.ConfigSourceProvider 文件示例：

```
org.acme.config.InMemoryConfigSourceProvider
```

要确保在编译和启动应用程序时注册并安装您创建的 **ConfigSourceProvider**，您必须完成上一步。

4. 编辑项目中的 **GreetingResource.java** 文件，以添加以下更新：

```
@ConfigProperty(name="my.prop") int value;
```

5. 在 **GreetingResource.java** 文件中，扩展 **hello** 方法以使用新属性：

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

6. 要以开发模式编译并启动应用程序，请输入以下命令：

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

7. 要验证 **/greeting** 端点是否返回预期的消息，请打开终端窗口并输入以下命令：

请求示例

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

8. 当应用程序成功读取自定义配置时，命令会返回以下响应：

```
hello world 1234
```

1.11. 使用自定义配置转换器作为配置值

您可以通过实施 **org.eclipse.microprofile.config.spi.Converter<T>** 并将其完全限定类名称添加到 **META-INF/services/org.eclipse.microprofile.config.spi.Converter** 文件，将自定义类型存储为配置值。通过使用转换器，您可以将值的字符串表示转换为对象。

先决条件

您已创建了 Quarkus **config-quickstart** 项目。

流程

1. 在 **org.acme.config** 软件包中，使用以下内容创建 **org.acme.config.MyCustomValue** 类：

自定义配置值示例

```
package org.acme.config;

public class MyCustomValue {
```

```
    private final int value;

    public MyCustomValue(Integer value) {
        this.value = value;
    }

    public int value() {
        return value;
    }
}
```

2. 实施转换器类，以覆盖转换方法来生成 **MyCustomValue** 实例。

转换类实施示例

```
package org.acme.config;

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

public class MyCustomValueConverter implements Converter<MyCustomValue> {

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

3. 将转换器的完全限定类名称 **org.acme.config.MyCustomValueConverter** 添加到您的 **META-INF/services/org.eclipse.microprofile.config.spi.Converter** 服务文件。

4. 在 **GreetingResource.java** 文件中，注入 **MyCustomValue** 属性：

```
@ConfigProperty(name="custom")
MyCustomValue value;
```

5. 编辑 **hello** 方法以使用此值：

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

6. 在 **application.properties** 文件中，添加要转换的字符串表示：

```
custom=1234
```

7. 要以开发模式编译并启动应用程序，请输入以下命令：

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

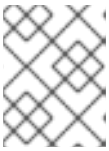
8. 要验证 /**greeting** 端点是否返回预期的消息，请打开终端窗口并输入以下命令：

请求示例

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

9. 当应用程序成功读取自定义配置时，命令会返回以下响应：

```
hello world - 1234
```



注意

您的自定义转换器类必须 **是公共的** 转换器，且必须有一个 **公共** 的非参数构造器。您的自定义转换器类无法 **抽象**。

其他资源：

- **microprofile-config** GitHub 存储库中的转换器列表

1.11.1. 设置自定义转换器优先级

所有 Quarkus 核心转换器的默认优先级都是 200。对于所有其他转换器，默认优先级为 100。您可以使用 **jakarta.annotation.Priority** 注解来增加自定义转换器的优先级。

以下流程演示了自定义转换器 **AnotherCustomValueConverter** 的实现，其优先级为 150。这优先于上一节中的 **MyCustomValueConverter**，其默认优先级为 100。

先决条件

- 您已创建了 Quarkus **config-quickstart** 项目。
- 您已为应用程序创建了自定义配置转换器。

流程

1. 通过使用 **@Priority** 注释注解类并传递优先级值，为您的自定义转换器设置优先级。在以下示例中，优先级值设为 **150**。

   AnotherCustomValueConverter.java 文件示例

```
package org.acme.config;

import jakarta.annotation.Priority;
import org.eclipse.microprofile.config.spi.Converter;

@Priority(150)
public class AnotherCustomValueConverter implements Converter<MyCustomValue> {

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

2. 在项目的 **src/main/resources/META-INF/services/** 子目录下，创建名为 **org.eclipse.microprofile.config.spi.Converter** 的文件，并在您创建的文件中输入实施 **Converter** 的完全限定名称：

org.eclipse.microprofile.config.spi.Converter 文件示例

```
org.acme.config.AnotherCustomValueConverter
```

您必须完成上一步，以确保在编译和启动应用程序时注册并安装了您创建的 **Converter**。

验证

完成所需的配置后，下一步是编译并打包 Quarkus 应用程序。如需更多信息和示例，请参阅 Getting started with Quarkus 指南中的编译和打包部分。

1.12. 其他资源

- 使用 Apache Maven 开发并编译 Quarkus 应用程序
- 将 Quarkus 应用程序部署到 OpenShift Container Platform
- 将 Quarkus 应用程序编译到原生可执行文件

更新于 2024-05-10