Quarkus Extension for Spring DI API.
https://quarkus.io/guides/spring-data-jpa

Quarkus Extension for Spring Web API.
https://quarkus.io/guides/spring-web

Extension for Spring Data API.
https://quarkus.io/guides/spring-di

如何将 SpringBoot 应用迁移到 Quarkus

本文英文原文出处:
https://developers.redhat.com/blog/2020/04/10/migrating-a-spring-boot-microservices-application-to-quarkus/

尽管 Spring Boot 长期以来一直是使用 Java 开发基于容器的应用程序的实际框架，但 Kubernetes 原生框架的性能优势却难以忽视。在本文中，我将向您展示如何快速将 Spring Boot 微服务应用程序迁移到 Quarkus。迁移完成后，我们将测试该应用程序，并比较原始 Spring Boot 应用程序和新 Quarkus 应用程序之间的启动时间。

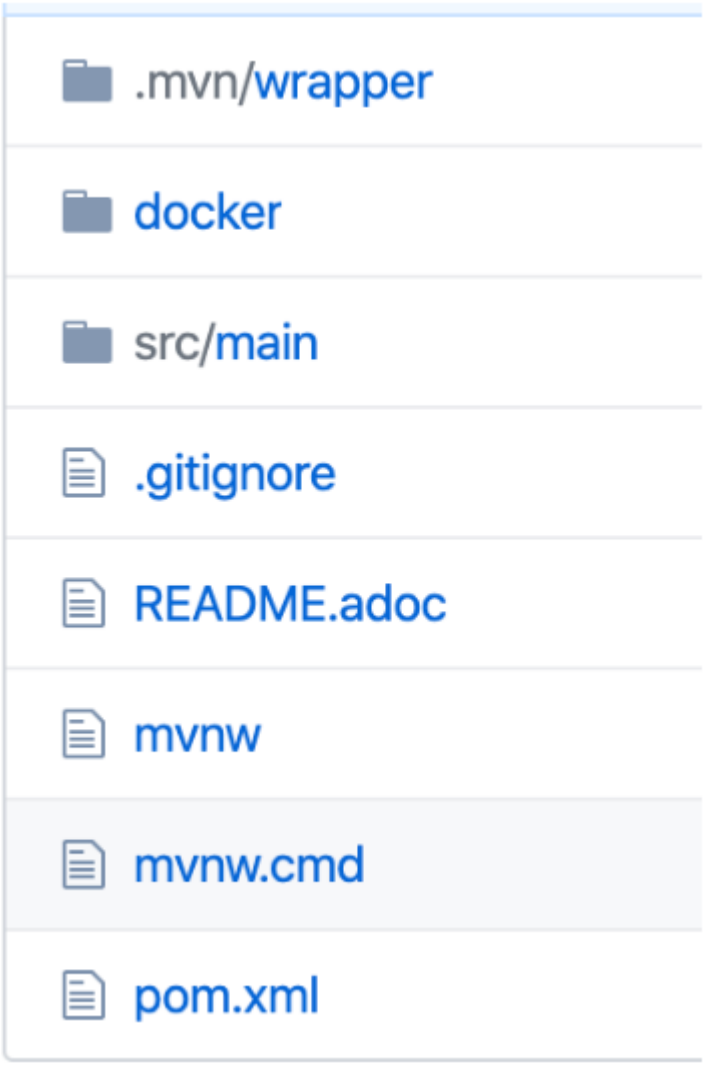
注意：对于有兴趣从 Spring Boot 迁移到 Quarkus 的开发人员，重要的是要知道 Quarkus 不支持 Spring Boot 的所有扩展和功能。作为一个示例，它仅支持 Java EE 上下文和依赖注入（CDI）API 的子集。将微服务或基于容器的应用程序迁移到 Quarkus 比迁移单片应用程序容易。

Quarkus 是 Kubernetes 原生的 Java 框架，专为 Java 虚拟机（JVM）（例如 GraalVM 和 HotSpot）量身定制。Kubernetes 原生意味着 Quarkus 采用容器优先的方法进行 Java 应用程序开发。容器优先开发固有的较小占用空间使 Quarkus 成为当今在 Kubernetes 和无服务器平台上运行 Java 应用程序的最佳选择之一。

Spring Boot 应用程序
对于我们的示例应用程序，我们将使用我上一篇文章“带 Red Hat AMQ Streams 的基于事件的微服务”中的 AccountBalance 微服务应用程序。AccountBalance 服务具有自己的 MongoDB 数据库，其中包含帐户余额信息。其他服务（例如 EventCorrelator 服务）也调用该数据库。

您可以在 GitHub 上找到示例应用程序的源代码。我将指导您完成从 Spring Boot 迁移到 Quarkus 的每个步骤。

下面显示了示例源代码的内容。这是典型的标准 Java 项目文件结构。mvnw 是我们生成的 Maven 包装器插件。我们将需要在 src / main 下修改 pom.xml 和源代码。



步骤 1：为您的应用程序修改 pom.xml

从 Spring Boot 迁移到 Quarkus 的最简单方法是引导一个示例 Quarkus 应用程序，并使用该应用程序的 pom.xml 作为模板来修改 Spring Boot 应用程序中的相同文件。

修改 Spring Boot pom.xml。我们将从删除不再需要的 Spring Boot 配置开始，然后将这些元素替换为 Quarkus 应用程序的相应配置。

删除 Spring Boot 配置

首先，我们从示例应用程序的 pom.xml 中删除打包配置。不再需要此操作，因为 pom.xml 配置中的<build>部分已对其进行了处理。

```
<!--- Remove the packing configuration -->

<packaging>jar</packaging>
```

spring-boot-starter-parent 用于 Spring Boot 应用程序，这里不再需要。

```
<!--- Remove spring-boot-starter-parent -->

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
</parent>
```

对 spring-cloud-dependencies 也是如此：

```
<!-- Remove the following from the <dependencyManagement> -->

<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>Brixton.SR4</version>
```

现在，我们可以删除所有剩余的 Spring Boot 依赖项：

```
<!-- Remove all Spring Boot-related dependencies -->

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-mongodb</artifactId>
 </dependency>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-commons</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-messaging</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-context</artifactId>
```

```
</dependency>
<dependency>
  <groupId>javax.ws.rs</groupId>
  <artifactId>javax.ws.rs-api</artifactId>
  <version>2.0.1</version>
</dependency>
<dependency>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven-model</artifactId>
  <version>3.3.9</version>
</dependency>
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.2.11</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.2.11</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.2.11</version>
</dependency>
<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>
```

我们还可以在构建部分中取出 spring-boot-maven-plugin：

```
<!-- Remove the spring-boot-maven plugin from the build section -
->

<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

将 Quarkus 元素添加到 pom.xml

接下来，我们将从引导的 Quarkus pom.xml 复制属性和依赖项，并将其粘贴到示例应用程序 pom.xml 的顶部。这部分对告诉编译器我们要在 Quarkus 中使用哪个版本的组件很重要。Quarkus 发行进展迅速，您可能需要检查<quarkus.platform.version>的最新版本。

```
<!-- Place this at the top of the pom.xml -->

<properties>
  <compiler-plugin.version>3.8.1</compiler-plugin.version>
  <maven.compiler.parameters>true</maven.compiler.parameters>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
  <quarkus-plugin.version>1.2.1.Final</quarkus-plugin.version>
  <quarkus.platform.artifact-id>quarkus-universe-
bom</quarkus.platform.artifact-id>
  <quarkus.platform.group-id>io.quarkus</quarkus.platform.group-
id>
  <quarkus.platform.version>1.2.1.Final</quarkus.platform.version>
  <surefire-plugin.version>2.22.1</surefire-plugin.version>
</properties>
```

将其添加到 dependencyManagement 下：

```
<!-- Add this under dependencyManagement -->

<dependencyManagement>
  <dependencies>
    <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>
  </dependencies>
</dependencyManagement>
```

现在，我们可以从引导文件中复制 Quarkus 依赖项并将其粘贴到 pom.xml 中。请注意，在下面的代码中，我还手动添加了示例应用程序所需的 MongoDB Panache 依赖项。我之所以手动添加了此依赖关系，是因为我以前没有使用 Web UI 对其进行引导。还要注意，AccountBalance 使用存储库连接到 MongoDB 数据库并查询它。Panache 扩展允许我们以最小的更改迁移现有代码。（您可以在 GitHub 上找到 Panache 的源代码。）

首先将以下依赖项添加到 pom.xml 中：

```
<dependency>
  <groupId>io.quarkus</groupId>

  <!-- notice this app is using jsonb -->

  <artifactId>quarkus-resteasy-jsonb</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-mongodb-panache</artifactId>
</dependency>
```

在构建配置下添加 quarkus-maven-plugin：

```
<dependency>
  <groupId>io.quarkus</groupId>

  <!-- notice this app is using jsonb -->

  <artifactId>quarkus-resteasy-jsonb</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>
```

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-mongodb-panache</artifactId>
</dependency>
```

在构建配置下添加 quarkus-maven-plugin：

```
<!-- Add the following plugin under the build configuration -->

<build>
  <finalName>${project.artifactId}-${project.version}</finalName>
  <plugins>
    <plugin>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
        <version>${compiler-plugin.version}</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemProperties>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.
util.logging.manager>
        </systemProperties>
      </configuration>
    </plugin>
  </plugins>
</build>
```

最后，添加配置文件设置：

```
<!-- Add the following profiles settings -->

<profiles>
  <profile>
    <id>native</id>
    <activation>
      <property>
        <name>native</name>
      </property>
    </activation>
    <build>
      <plugins>
        <plugin>
          <artifactId>maven-failsafe-plugin</artifactId>
            <version>${surefire-plugin.version}</version>
            <executions>
              <execution>
                <goals>
                  <goal>integration-test</goal>
                  <goal>verify</goal>
                </goals>
                <configuration>
                  <systemProperties>

<native.image.path>${project.build.directory}/${project.build.fin
alName}-runner</native.image.path>
                  </systemProperties>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
     </build>
     <properties>
       <quarkus.package.type>native</quarkus.package.type>
     </properties>
  </profile>
</profiles>
```

步骤 2：迁移 Spring Boot 应用程序代码

Maven POM 已准备就绪。现在，我们准备将 Spring Boot 应用程序代码迁移到 Quarkus。

首先，删除 Application.java 类。我们不再需要它了。您还可以修改应用程序端口和 MongoDB 客户端属性文件的 application.properties，如下所示

```
### --- Remove the Spring Boot properties
# spring.application.name=accountbalance-service
# spring.data.mongodb.host=localhost
# spring.data.mongodb.port=27017
# spring.data.mongodb.username=checkbalance
# spring.data.mongodb.password=checkbalance
# spring.data.mongodb.database=checkbalance
# server.port=8082
### --- Replace with equivalent Quarkus MongoDB
properties
quarkus.mongodb.connection-
string=mongodb://checkbalance:checkbalance@localhost:2701
7/checkbalance
# --- Note the wrong context of database name ...
planning and design before coding is so important.
quarkus.mongodb.database=checkbalance

quarkus.http.port=8082。。s.s;sxs
```

实体 bean 代码

接下来，我们将在 Balance.java 中更改实体 bean 代码。首先，删除与 Spring Boot 相关的 import 语句。在下一节中，我们将用 Quarkus 等效组件替换这些组件。

```
/// --- Remove the Spring Boot-related import statements

import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```

为 Quarkus 添加等效的 import 语句。Quarkus 框架通过 MongoDB Panache 扩展为 MongoDB 提供了自己支持的组件。Quarkus 还提供了自己的 CDI 注释版本。我们在这里为这些提供正确的导入语句。

```
/// --- Add the following Quarkus-related import statements

import io.quarkus.mongodb.panache.MongoEntity;
import io.quarkus.mongodb.panache.PanacheMongoEntity;
import org.bson.codecs.pojo.annotations.BsonProperty;
import org.bson.codecs.pojo.annotations.BsonId;
import io.quarkus.mongodb.panache.PanacheQuery;
```

用@MongoEntity 替换现有的@Document 批注：

```
/// --- Replace @Document annotation with @MongoEntity

// --- @Document(collection = "balance") <--- Remove this
@MongoEntity(collection="balance")
public class Balance extends PanacheMongoEntity{
/// --- More codes here are omitted ...
...
```

用 Quarkus 的@BsonId 替换 Spring Boot 的@Id 注释：

```
/// --- Replace @Id annotation with @BsonId

// @Id <--- Remove this
@BsonId  //In fact, we do not need this. Just leave it here for
now.
private String _id;
```

存储库类

我们还需要修改 Spring Boot 存储库类 BalanceRepository.java。首先，删除以下导入语句：

```
/// --- Remove the following import statement

import
org.springframework.data.mongodb.repository.MongoRepository;
```

Replace it with this one:

```
/// --- Add the following import statements

import io.quarkus.mongodb.panache.PanacheMongoRepository;
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.mongodb.panache.PanacheQuery;
```

然后再进行三个快速更改：

```
/// --- Change the BalanceRepository to implement
PanacheMongoRepository
// --- Change the BalanceRepository from interface to class
// --- Drop in the @ApplicationScoped annotation

@ApplicationScoped
public class BalanceRepository implements
PanacheMongoRepository<Balance> {
   // --- Change the findByAccountId(String accountId) to the
```

```
following implementation
   public Balance findByAccountId(String accountId){
      return find("accountId", accountId).firstResult();
   }
}
```

REST 类

我们还将修改 AccountBalance.java。首先删除所有与 Spring Boot 相关的 import 语句：

```
/// --- Remove all the Spring Boot-related import statements

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.core.env.Environment;
import org.springframework.data.domain.Example;
import org.springframework.data.repository.Repository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.beans.factory.annotation.Autowired;
```

为 Quarkus 添加以下 import 语句。（我们没有在.java 文件中使用所有这些导入，但现在可以将其保留。）

```
/// --- Add the following import statements.

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;
import javax.inject.Inject;
```

删除 Spring Boot @RequestMapping 和@RestController 注释：

```
/// --- Remove the @RequestMapping and @RestController and
update as follows
```

```
// @RequestMapping("/ws/pg")      <--- Remove this
// @RestController      <--- Remove this
@Path("/ws/pg")        // <---- Add this
@Produces(MediaType.APPLICATION_JSON)        // <---- Add this
@Consumes(MediaType.APPLICATION_JSON)       // <---- Add this
public class AccountBalance{
/// --- I have omitted more code here
...
```

您还需要删除@Autowire 和@Inject 批注：

```
/// --- Remove @Autowire with @Inject

// @Autowired      <--- Remove this
@Inject
private BalanceRepository repository;
```

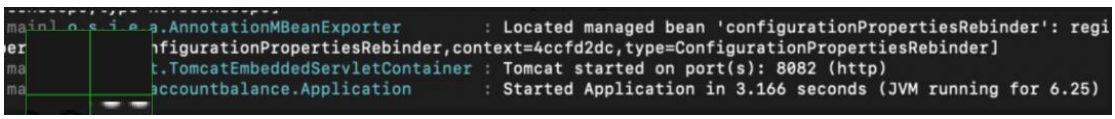
请注意，我们将@RequestMapping 批注更改为@Path，
并且还删除了@ResponseBody。
最后一步，我们需要将@PathVariable 更改为@PathParam，如下所示。

```
@Path("/balance/{accountid}")
@GET
public Balance get(@PathParam("accountid") String accountId) {
   Balanceresult = repository.findByAccountId(accountId);
   return result;
}
```

尽管我省略了一些步骤以使此练习简短，但完成了迁移。 例如，AccountBalance 应用程序打算部署到 Red Hat OpenShift 上，
因此我为此修改了 Heathz.java 文件。您可以在应用程序源中查看这些更改。

步骤 3：测试 Quarkus 应用程序

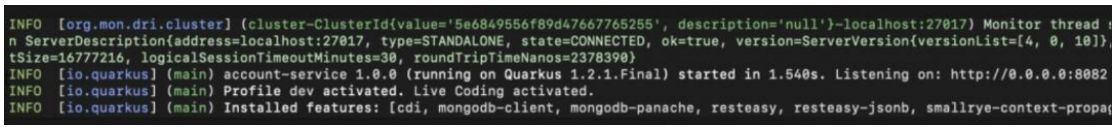
接下来，我们将测试我们的新应用程序。如图显示了迁移之前的 Spring Boot 执行。



Spring Boot 应用程序在 3.166 秒内启动。

让我们看看 Quarkus 应用程序的启动与 Spring Boot 的比较。在命令提示符下，切换到 AccountBalance 目录。执行以下命令以运行迁移的应用程序：

mvn quarkus：dev
请注意，首次运行该应用程序所花费的时间可能会比后续运行所花费的时间更长。Maven 需要更多时间才能将 Quarkus 存储库下载到本地计算机。下图显示了新 Quarkus 应用程序的开始时间。

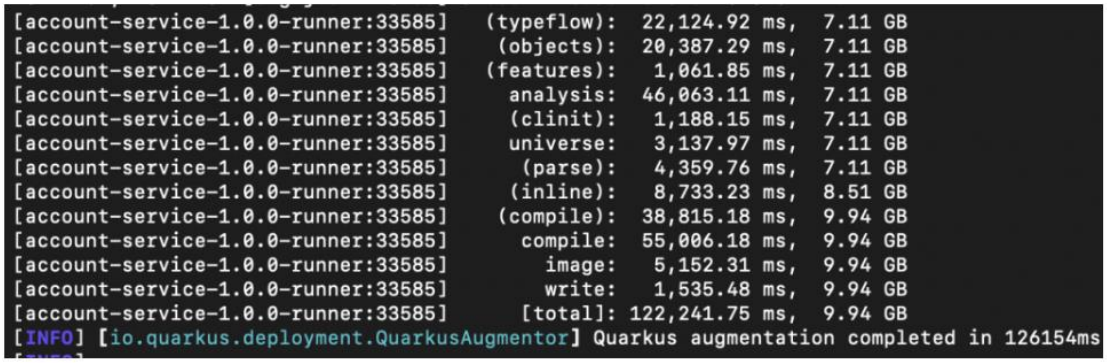


Quarkus 在 1.540 秒内启动。

这是一个进步。现在让我们做一个 Quarkus 原生版本。请记住要从项目的根目录执行本机构建：

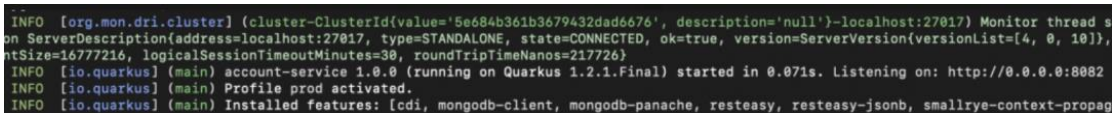
```
/mvnw package -Pnative
```

图 3 的输出表明 Quarkus 本机构建成功。



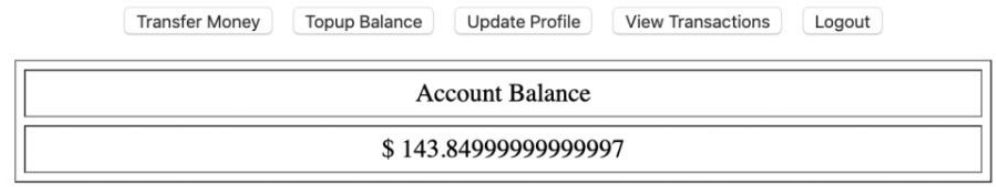
输入以下内容以运行本机二进制文件：

```
./target/account-service-1.0.0-runner
```



Quarkus 本地版本在 0.071 秒内开始。

最后，我创建了一个简单的 UI 来调用 AccountBalance 服务并显示帐户余额。如图所示，迁移的应用程序按预期工作。



将示例应用程序从 Spring Boot 迁移到 Quarkus 是一个非常简单的练习。对于更复杂的应用程序，迁移路径将更加复杂。将我的微服务应用程序从 Spring Boot 迁移到 Quarkus 所面临的最大挑战是确定要使用的注释和库。Quarkus 的文档提供了很好的示例，但省略了 import 语句。许多 IDE 将为您处理这些配置详细信息，但我选择手动迁移。