使用带有 JSP 和 WAR 存档的 Spring Boot 创建 MVC Web 应用程序

韩博孙

2018 年 10 月 18 日 麻省理工学院

在本文中，我喜欢讨论如何设置打包为 WAR 存档并支持 Spring MVC 并以 JSP 作为视图的 Spring Boot 应用程序。

下载源 - 6.7 KB

介绍

这将是我关于 Spring Boot 的第一个教程，我想在其中讨论设置 Spring MVC 应用程序的方法。为了使用 JSP 页面作为视图模板，一种方法是将应用程序打包为 WAR 存档。成品可以作为独立应用程序运行，并作为 MVC 应用程序处理用户请求。

Spring Boot 是一个非常棒的应用程序开发框架。该框架的唯一重点是创建可以作为独立应用程序运行的各种程序。我指的不仅仅是可以部署到应用程序容器（如 Tomcat、Jetty 或 Glassfish）中的任何 Web 应用程序；还有可以与 RabbitMQ 或 Kafka 等消息传递代理交互的应用程序；以及可以使用石英调度程序运行的程序。我想大多数人可能会使用这个框架来创建基于 Web 的应用程序——MVC 应用程序或基于 RESTFul API 的应用程序。这些应用程序可以部署到 Docker 容器中，作为微服务。

本教程将向您展示使用 Spring Boot 创建基于 Web 的应用程序是多么容易。完成后的应用程序是一个独立的 Java 应用程序。它有一个嵌入式 Tomcat 应用服务器。它可以处理 Web 请求和静态 Web 内容。正如我在过去的文章中多次提到的，每个新项目最糟糕的部分是设置项目。Spring Boot 存在一些困难。但是与创建 Spring v3/v4 MVC 应用程序相比，它更容易使用。你会看见。

文件结构

在我们进入示例项目的细节之前，我喜欢展示项目的目录和文件结构。这里是：

XML 复制代码

```
<base-dir>/src/main/java/org/hanbo/boot/app/controllers/HelloController.java
<base-dir>/src/main/java/org/hanbo/boot/app/App.java
<base-dir>/src/main/resources/application.properties
<base-dir>/src/main/resources/static/test.html
<base-dir>/src/main/resources/static/assets/css/index.css
<base-dir>/src/main/resources/static/assets/js/test.js
<base-dir>/src/main/webapp/WEB-INF/jsp/testme.jsp
```

我们得到了各种各样的文件。他们是：

- 两个 Java 文件。一是程序执行入口。另一个是MVC控制器。

- 一个包含一些配置值的属性文件。
- 三个静态文件，可在请求时直接提供给用户。
- 一个 JSP 文件，用作 MVC 应用程序的视图模板。

正如您在此处看到的，设置 Spring MVC 应用程序既快速又简单。如果您知道设置 Web 应用程序的旧方法，那么您将需要至少两个用于配置的 XML 文件。或者如果你在Spring应用程序中使用Java注解，那么你需要知道如何将XML配置中的配置转换为使用Java注解。如果扩展本教程中的示例应用程序，则必须使用 Java 注释进行配置。它将类似于 XML 配置的工作方式，但对于入门应用程序来说，这非常简单。

POM XML 文件

让我们从 POM XML 文件开始。POM XML 文件用于 Maven 构建。它指定了如何编译和打包项目。这是这个文件的内容：

XML 缩小▲ 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <artifactId>boot-war</artifactId>
    <packaging>war</packaging>
    <name>Hanbo Boot War Sample App</name>
    <description>An example of Spring Boot, JSP and WAR</description>
    <version>1.0.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
    </parent>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

关于这个 POM 文件有一些重要的事情。第一个是指定 maven 构建将创建一个 WAR 存档的行：

XML 复制代码

```xml
<packaging>war</packaging>
```

第二个是 POM 文件具有父 POM 依赖项。这允许下载大量 Spring 和非 Spring 依赖项并将其链接到该项目中：

XML 复制代码

```xml
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.5.RELEASE</version>
</parent>
```

第三个是将 Java 编译设置为使用 JDK 1.8 的属性定义：

XML 复制代码

```xml
<properties>
   <java.version>1.8</java.version>
</properties>
```

最后是使用 Spring Boot Maven 插件进行编译和打包：

XML 复制代码

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

依赖项部分定义了此应用程序所需的额外依赖项。我需要的是 Spring MVC，并作为 J2EE Web 应用程序运行。添加的依赖项用于编译 JSP 视图和运行嵌入式应用服务器。

主要入口

接下来，我将向您展示程序入口。与在应用程序容器中运行的基于 Spring 的 Web 应用程序不同，基于 Spring Boot 的 Web 应用程序是自托管的。每个此类程序中都有一个静态主条目。这是包含主要条目的类的完整源代码：

爪哇 复制代码

```java
package org.hanbo.boot.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class App extends SpringBootServletInitializer
{
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder appBuilder)
   {
        return appBuilder.sources(App.class);
    }

    public static void main(String[] args) throws Exception
```

```
    {
        SpringApplication.run(App.class, args);
    }
}
```

该类被调用App，它从 class 扩展而来SpringBootServletInitializer。这允许 Spring Framework 识别App可以作为传统 WAR 包初始化和执行的类。它还告诉 Spring Framework 将有*WEB-INF*文件夹和其中的资源以供使用。

在类中App，有一个受保护的方法称为configure(). 它用于指定任何特定于应用程序的配置。它只有一行，它采用 的类类型App并创建一个SpringApplicationBuilder对象并返回。这样做的作用是SpringApplicationBuilder创建的对象将自动扫描App类、它所在的包以及任何带注释的类的子包，以及包含 Spring 配置的注释。然后它会基于这些配置构建一个 Spring 应用程序。这是按惯例集成的经典示例。一切都通过依赖注入耦合。

该static main方法只有一行，它将类的类型App和任何其他命令行参数传递到SpringApplication.run(). 在幕后，这个类做了很多。它会隐式地对当前包和所有子包进行组件扫描。如果需要，开发人员还可以指定其他包进行组件扫描。开发人员可以为所需的任何其他配置添加额外的注释。对于这个简单的程序，只有 MVC 控制器类可以处理用户对页面的请求。

MVC 控制器类

我想展示的下一个类是 MVC 控制器类。这是一个非常简单的类，只有一个方法来处理GET来自用户的 HTTP请求，以及一些查询参数。它通过使用 JSP 页面作为视图来响应。

源代码如下：

爪哇 复制代码

```
package org.hanbo.boot.app.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloController
{
    @RequestMapping(value="/meow", method = RequestMethod.GET)
    public ModelAndView hello(
       @RequestParam("sayit")
       String sayit
    )
   {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("testme");
        retVal.addObject("mymessage", sayit);
        return retVal;
    }
}
```

这门课没有新的惊喜。也就是说，对于在应用程序容器中运行的应用程序，此控制器类的定义与任何 Spring MVC 控制器相同。让我简单总结一下实现：

- 该类用@Controller.
- 该类只有一种处理 HTTPGET请求的方法。它用 注释@RequestMapping。该注解定义了请求的子路径和它可以处理的 HTTP 方法，GETrequests。
- 该方法创建一个ModelAndView对象并返回。视图页面称为“ testme”。数据模型只是一个string将显示在页面上的。
- 该方法接受一个参数，该参数从查询参数中获取，称为“ sayit”。

简单地说，当用户尝试导航到：

复制代码

```
http://localhost:8080/meow?sayit=This+is+pretty+crazy
```

然后用户点击**Enter**，浏览器将显示以下内容：

复制代码

```
What did you say?

I said: "This is pretty crazy."
```

为了让这个控制器按预期工作，需要一些额外的配置。它是在*application.properties*文件中完成的。

Application.properties 文件

我需要 application.properties 文件的原因是我需要指定视图模板文件的前缀和后缀。如果您从过去就知道，有一个创建和配置类型对象的步骤`org.springframework.web.servlet.view.InternalResourceViewResolver`。在这种情况下，我们只需要两行`application.properties`：

XML 复制代码

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

这个配置的作用是设置应用程序在“ *WEB-INF/jsp/* ”文件夹中找到视图模板。文件扩展名为“ *.jsp* ”。现在我们已经看到了主条目、控制器类和内部资源视图解析器的设置。最后一块拼图是视图模板。

JSP 视图模板

本教程的视图模板非常简单，只有一个 JSP 文件，用于演示如何显示来自视图模型的数据：

HTML 复制代码

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html lang="en">
<head>
    <c:url value="/assets/css/index.css" var="jstlCss" />
    <link href="${jstlCss}" rel="stylesheet" />
</head>
<body>
   <p>What did you say?</p>
   <p>I said: <span class="text-underline">"${mymessage}."</span></p>
   <script type="text/javascript" src="/assets/js/test.js">
   </script>
</body>
</html>
```

在这个 JSP 文件中，有三件事：

- 有级联样式表文件。它是一个静态文件，通过 JSTL 标记添加。这表明 JSTL 库已正确合并到项目中。
- 视图模型只有一个元素，并通过`${mymessage}`.
- 在 HTML 内容的末尾添加了一个 JavaScript 文件，一旦加载它就会执行。这是为了演示在这个项目中使用静态文件作品。

让示例应用程序提供静态内容非常重要。它提供了超越 Spring MVC 的方法——静态页面和 JavaScript 可用于单页面 Web 应用程序。并且使用 Spring Boot 拥有静态内容非常容易，接下来会解释。

提供静态内容

Spring Boot 提供了很多便利，使开发人员可以快速开发和开发。方便是通过约定。在 Web 应用程序中提供静态内容就是一个很好的例子。当您指定使用 Spring Boot 开发的应用程序作为基于 Web 的应用程序时，您需要做的就是在*src/main/resources*下创建一个名

为“ *static* ”的文件夹。在这个“*静态*”文件夹中，随意添加任何子文件夹，并将任何静态内容文件放入其中，然后您就可以使用它们了。

以示例应用程序为例，我将三个文件指定为静态内容：

- 级联样式表文件——*src/resources/static/assets/css/index.css*
- 一个 JavaScript 文件，它没有做任何有用的事情，而是在 JavaScript 输出控制台上打印出一行 -- *src/resources/static/assets/js/test.js*
- 一个 HTML 文件，只是为了演示在基于 Spring Boot 的 Web 应用程序中不需要 JSP 的事实——*src/resources/static/test.html*

那么，用户如何`static`通过浏览器访问这些内容呢？这很容易。假设应用程序正在运行，假设应用程序运行的网址是 *http://localhost:8080/*（这里没有特定的应用程序上下文），那么用户可以看到这些静态内容文件的实际内容以下网址：

- *http://localhost:8080/assets/js/test.js* - 您会看到 JS 文件的实际内容。
- *http://localhost:8080/assets/css/index.css* - 您会看到级联样式表文件的实际内容。
- *http://localhost:8080/assets/test.html* - 您会看到一个网页。

有多种方法可以配置 Spring Boot 应用程序以在其他位置查找静态内容文件，WEB-INF 和 JSP 位置也是如此，以及应用程序如何初始化，以及 Spring Boot 应用程序如何运行的许多其他方面。这些不是本教程中讨论的主题，但如果对 Spring Boot 有足够的了解，它们就会变得微不足道。我可能会在以后的作品中介绍其中的一些。

如何构建和运行应用程序

在构建项目之前，请转到*src/main/resources/static/assets/js*文件夹并将文件“ *test.sj* ”重命名为“ *test.js* ”。

要构建此应用程序，请在命令行控制台中运行以下命令：

复制代码

```
mvn clean install
```

当您第一次运行它时，它会下载构建此应用程序的所有依赖项，这可能需要一些时间。之后，后续构建将花费更少的时间。

要运行 Web 应用程序，您还可以使用命令行控制台执行此操作：

XML 复制代码

```
java -jar target\boot-war-1.0.0.war
```

如果一切顺利并且您可以构建应用程序，则执行将在最后输出如下内容：

复制代码

```
<small>
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.5.RELEASE)

2018-10-14 22:51:11.356  INFO 6124 --- [           main] org.hanbo.boot.app.App
: Starting App v1.0.0 on U3DTEST-PC with PID 6124 (C:\Users\u3dadmin\workspace-
mars8\SpringBootJspWar\target\boot-war-1.0.0.war started by u3dadmin in
C:\Users\u3dadmin\workspace-mars8\SpringBootJspWar)

....
....
....

2018-10-14 22:51:28.730  INFO 6124 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context
path ''
2018-10-14 22:51:28.745  INFO 6124 --- [           main] org.hanbo.boot.app.App
: Started App in 20.84 seconds (JVM running for 23.398)
</small>
```

兴趣点

正如我之前提到的，使用 Spring Boot 创建 Web 应用程序非常简单、方便。这种便利的全部意义在于去除所有凌乱的配置，以便人们可以专注于开发过程中最重要的部分，设计出适合最终产品愿景的东西。除此之外，Spring Boot 的一个很酷的地方是它可以快速部署到 Docker VM 中。所以它非常适合开发微服务。

对我来说，下一步是编写一个更复杂的 Spring Boot 教程，其中包括数据访问、身份验证和授权。敬请关注。

历史

- 10/14/2018：初稿

执照

分享

关于作者

韩博孙

组长 The Judge Group

美国

手表该会员

没有提供传记

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

-- 本论坛暂无消息 --