部署 Spring Boot 到 WebSphere Liberty Profile

这是一个在 WebSphere Liberty Profile (WLP) 上运行简单 Spring Boot 应用程序的完整指南。本示例创建了一个包含基本 REST 端点的 Spring Boot 应用程序，该端点返回“Hello World!”，并将其部署到 WLP 的 `dropins` 目录中。我们假设您已经启用了 `javaee-8.0` 功能的 WLP，因为它提供了必要的 Servlet 支持。

分步指南

1. 创建 Spring Boot 应用程序 首先，使用 Spring Initializr（或您喜欢的方法）创建一个新的 Spring Boot 项目。我们将使用 Maven、Java 和 Spring Web 依赖项来构建一个简单的 Web 应用程序。

- **生成项目：**
  - 访问 Spring Initializr。
  - 配置以下内容：
    - **项目：** Maven
    - **语言：** Java
    - **Spring Boot 版本：** 2.7.x（或最新的稳定版本）
    - **组：** `com.example`
    - **工件：** `demo`
    - **依赖项：** Spring Web
  - 点击“生成”下载项目 ZIP 文件，然后解压并在 IDE 中打开。
- **添加简单的 REST 控制器：**在 `src/main/java/com/example/demo` 中创建一个名为 `HelloController.java` 的文件，内容如下：

```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {
    @GetMapping("/")
    public String hello() {
        return "Hello World!";
    }
}
```

这将在根路径（`/`）创建一个 REST 端点，返回“Hello World!”作为纯文本。

2. 配置应用程序进行 WAR 部署 默认情况下，Spring Boot 将应用程序打包为包含嵌入式服务器（例如 Tomcat）的 JAR 文件。要在 WLP 上部署，我们需要将其打包为 WAR 文件，并配置其与 WLP 的 Servlet 容器一起工作。

- **修改主应用程序类：**编辑 `src/main/java/com/example/demo/DemoApplication.java` 以扩展 `SpringBootServletInitiali` 这允许应用程序在外部 Servlet 容器（如 WLP）中运行：

```
package com.example.demo;

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 DemoApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(DemoApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

- **更新 `pom.xml` 进行 WAR 打包：**打开 `pom.xml` 并进行以下更改：
  - 通过在顶部（`<modelVersion>` 之后）添加以下行将打包设置为 WAR：

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

  - 将嵌入式 Tomcat 依赖项标记为 `provided`，以便它不会包含在 WAR 中（WLP 提供自己的 Servlet 容器）。修改 `spring-boot-starter-web` 依赖项（其中包含 Tomcat）如下：

    ```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    ```

    在其下方添加：

    ```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
    ```

```
        <scope>provided</scope>
    </dependency>
```

现在，您的 `pom.xml` 依赖项部分应如下所示：

```
<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>
  <!-- 其他依赖项如 spring-boot-starter-test 可能保持不变 -->
</dependencies>
```

3. 构建 WAR 文件 使用 Maven 将应用程序编译并打包为 WAR 文件。

- **运行构建命令：**从项目根目录（`pom.xml` 所在的位置）执行：

  ```
  mvn clean package
  ```

  这将在 `target` 目录中生成 WAR 文件，例如 `target/demo-0.0.1-SNAPSHOT.war`。

- **重命名 WAR 文件（可选）：**为了更清晰的 URL，将 WAR 文件重命名为 `myapp.war`：

  ```
  mv target/demo-0.0.1-SNAPSHOT.war target/myapp.war
  ```

  这将上下文根简化为 `/myapp` 而不是 `/demo-0.0.1-SNAPSHOT`。

4. 在 WLP 上部署 WAR 文件 使用 `dropins` 目录将 WAR 文件部署到 WLP，这将启用自动部署。

- **定位 `dropins` 目录：**找到 WLP 服务器的 `dropins` 目录。如果 WLP 安装在 `/opt/ibm/wlp` 并且服务器名为 `myServer`，则路径为：

  ```
  /opt/ibm/wlp/usr/servers/myServer/dropins
  ```

- **复制 WAR 文件：**将 WAR 文件移动到 `dropins` 目录：

  ```
  cp target/myapp.war /opt/ibm/wlp/usr/servers/myServer/dropins/
  ```

- **启动服务器（如果未运行）：**如果 WLP 未运行，请启动它：

  ```
  /opt/ibm/wlp/bin/server start myServer
  ```

如果它已经在运行，它将自动检测并部署 WAR 文件。

- **验证部署：**检查服务器日志或控制台，查找类似于以下的消息：

```
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/myapp/
```

  - 日志位于 `/opt/ibm/wlp/usr/servers/myServer/logs/console.log`（后台模式）或在终端中显示（前台模式与 `./server run myServer`）。

5. 访问应用程序 在浏览器中测试部署的 Spring Boot 应用程序。

- **打开浏览器：**导航到：

```
http://localhost:9080/myapp/
```

  - `9080` 是 WLP 的默认 HTTP 端口。
  - `/myapp` 是从 WAR 文件名称派生的上下文根。
  - `/` 匹配控制器中的 `@GetMapping("/")`。

- **预期结果：**您应该看到：

```
Hello World!
```

  作为纯文本显示。

注意事项

- **上下文根：**上下文根（`/myapp`）派生自 WAR 文件名称。通过重命名 WAR 文件进行调整。
- **端口号：** WLP 默认使用 `9080` 作为 HTTP 端口。如果您的服务器使用不同的端口，请相应更新 URL。
- **Java 版本：**确保 WLP 和您的构建环境使用兼容的 Java 版本（例如，对于 Spring Boot 2.7.x，使用 Java 8 或 11）。
- **WLP 功能：** `javaee-8.0` 功能包括 Servlet 4.0 支持，这与 Spring Boot 2.7.x 的要求兼容（Servlet 3.1+）。
- **故障排除：**
  - 如果页面未加载，请检查 `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` 中的错误。
  - 验证 WAR 文件是否在 `dropins` 中，并且服务器正在运行。

总结

本指南演示了如何：1. 创建一个包含 REST 端点的 Spring Boot 应用程序。2. 通过扩展 `SpringBootServletInitializer` 并将 Tomcat 设置为 `provided` 进行 WAR 部署的配置。3. 构建并将 WAR 文件部署到 WLP 的 `dropins` 目录。4. 在 `http://localhost:9080/myapp/` 访问“Hello World!”输出。

这种方法利用了 WLP 的自动部署功能和 Spring Boot 的灵活性，提供了一个简单但功能强大的 Web 应用程序在 WebSphere Liberty Profile 上。