WebSphere Liberty Profile 应用启动问题

看起来，在将 Spring Boot 应用程序降级到版本 2.7.18 并部署到 WebSphere Liberty Profile (WLP) 后，您遇到了一个问题，即应用程序在默认超时期限内无法启动，`console.log` 中显示的消息是 “the application myapp has not started in 30.xx seconds”。这表明应用程序初始化时间过长或根本无法启动。以下是诊断和解决此问题的逐步指南。

步骤 1：调查服务器日志以寻找线索

首先，详细检查服务器日志以确定启动失败的根本原因。`console.log` 文件提供了超时消息，但可能不包含完整的信息。检查 WLP 服务器日志目录中的以下日志文件（`/opt/ibm/wlp/usr/servers/myServer/logs/`）：

- `messages.log`：此文件通常包含 ERROR 或 WARNING 消息，可以指明问题，如缺少依赖项、配置错误或启动期间的异常。
- `trace.log`：如果启用了详细跟踪，此文件可能提供有关部署期间发生情况的更多上下文。

查找：- 堆栈跟踪或异常（例如，`ClassNotFoundException`，`NoSuchBeanDefinitionException`）。- 关于缺少资源或不兼容库的消息。- 应用程序上下文未初始化的指示。

如果没有看到足够的详细信息，可以通过修改 `server.xml` 文件在 WLP 中增加日志级别。添加或更新 `<logging>` 元素如下：

```
<logging traceSpecification="*=info:com.ibm.ws.webcontainer*=all" />
```

在进行此更改后重新启动服务器，重新部署应用程序，并再次检查日志以获取更多信息。

步骤 2：使用日志验证应用程序启动

由于这是一个 Spring Boot 应用程序，问题可能与应用程序上下文未初始化有关。为了确定启动过程进展到哪一步，在主应用程序类中添加一个简单的日志语句，使用 `@PostConstruct` 方法。以下是一个示例：

```
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;
```

```
import javax.annotation.PostConstruct;

@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);
    }

    @PostConstruct
    public void init() {
        System.out.println("Application context initialized");
    }
}
```

- 重新构建应用程序（`mvn clean package`）。
- 将 WAR 文件重新部署到 WLP 的 `dropins` 目录。
- 检查 `console.log` 中的消息 `"Application context initialized"`。

如果显示此消息，则应用程序上下文加载成功，问题可能与 Web 组件或 Servlet 初始化有关。如果没有显示，则问题发生在上下文初始化的早期阶段。

步骤 3：在 Spring Boot 中启用调试日志

为了更好地了解 Spring Boot 的启动过程，启用调试日志，方法是添加一个配置文件。创建或编辑 `src/main/resources/application.properties` 文件，内容如下：

```
debug=true
```

- 重新构建并重新部署应用程序。
- 检查 `console.log`（或其他日志）以获取 Spring Boot 的详细调试输出。

这将记录有关 bean 创建、自动配置和启动期间发生的任何错误的信息。查找可能导致挂起或失败的线索。

步骤 4：验证 WAR 文件和依赖项配置

由于您将其部署到 WLP，WLP 提供自己的 Servlet 容器，因此请确保 WAR 文件正确配置为外部服务器：

- **WAR 打包**：在 `pom.xml` 中，确认打包设置为 `war`：

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

- **Tomcat 作为提供的**：确保嵌入式 Tomcat 被排除在 WAR 文件之外，因为 WLP 将提供 Servlet 容器。检查 `pom.xml` 中的：

```
<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>
```

- **Servlet API 兼容性**：Spring Boot 2.7.18 使用 `javax.servlet:javax.servlet-api:4.0.1`，与 WLP 的 `javaee-8.0` 功能（Servlet 4.0）兼容。要确认没有冲突的依赖项，运行：

```
mvn dependency:tree
```

查找任何意外的 Servlet API 版本（例如，`jakarta.servlet-api`，用于 Spring Boot 3.x 并与 `javaee-8.0` 不兼容）。

如果怀疑依赖项问题，解压 WAR 文件并检查 `WEB-INF/lib`，以确保没有意外的 Servlet 相关 JAR 文件包含在内。

步骤 5：本地测试以隔离问题

为了确定问题是特定于 WLP 还是应用程序本身，使用嵌入式 Tomcat 本地测试应用程序：

```
mvn spring-boot:run
```

如果成功启动并且可以访问您的端点（例如，一个简单的 `"Hello World!"` REST 控制器），则问题可能与 WLP 部署而不是应用程序代码有关。

步骤 6：调整 WLP 启动超时（临时解决方案）

如果日志表明应用程序正在启动但需要超过 30 秒，可以在 WLP 的 `server.xml` 中增加启动超时：

```
<applicationMonitor startTimeout="60s" />
```

- 重新部署应用程序并监控日志。
- 如果在延长的超时后启动，这确认了一个缓慢的启动过程，您应该优化应用程序（例如，减少组件扫描或初始化任务）。

然而，这是一个临时解决方案——理想情况下，简单的应用程序应该在 30 秒内启动，因此继续调查根本原因。

步骤 7：简化并与新项目进行比较

如果问题仍然存在，创建一个最小的 Spring Boot 2.7.18 项目以测试在 WLP 上的部署：1. 使用 Spring Initializr 并选择：- Spring Boot 2.7.18 - Java（与 WLP 版本匹配，例如 8 或 11）- 依赖项：Spring Web 2. 添加一个基本的 REST 控制器：

```
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!";
    }
}
```

3. 将其配置为 WAR 部署（如上所述扩展 `SpringBootServletInitializer`）。
4. 构建 WAR 文件（`mvn clean package`）并将其部署到 WLP 的 `dropins` 目录。

如果此新项目成功启动，请将其 `pom.xml`、主类和配置与原始项目进行比较，以确定导致失败的差异。

步骤 8：额外的 WLP 故障排除

如果以上方法都无法解决问题，请考虑这些特定于 WLP 的检查：- **清理工作目录**：删除可能干扰部署的缓存数据：`bash  rm -rf /opt/ibm/wlp/usr/servers/myServer/workarea` 然后重新启动服务器并重新部署。- **检查 WLP 版本**：运行 `/opt/ibm/wlp/bin/productInfo version` 以确认您的 WLP 版本支持使用 `javaee-8.0` 部署 Spring Boot 2.7.x。- **Java 版本**：确保 WLP 使用的 Java 版本（在 WLP 环境中使用 `java -version` 检查）与 Spring Boot 2.7.18 支持的版本（Java 8、11 或 17）匹配。

推荐修复

根据提供的信息，最有可能的问题是应用程序上下文初始化期间的问题或与 WLP 的微妙配置不匹配。以下是如何修复它的方法：

1. **从日志开始**：
   - 添加 `@PostConstruct` 日志（步骤 2）并启用 `debug=true`（步骤 3）。
   - 重新部署并检查 `console.log` 和 `messages.log` 中的错误或启动挂起的位置。
2. **验证配置**：
   - 确认 WAR 文件设置和依赖项（步骤 4）。
   - 本地测试（步骤 5）以排除应用程序级问题。
3. **迭代和简化**：
   - 如果需要，测试一个最小项目（步骤 7）以隔离问题。

通过遵循这些步骤，您应该能够找出问题的原因——无论是依赖项冲突、启动缓慢的组件还是 WLP 配置问题——并相应地解决它。如果仍然遇到困难，请分享日志中的具体错误消息以获得进一步的帮助！