WebSphere Liberty Profile のアプリケーション起動問題

Spring Boot アプリケーションをバージョン 2.7.18 にダウングレードし、WebSphere Liberty Profile (WLP) にデプロイした後、アプリケーションがデフォルトのタイムアウト期間内に起動しない問題が発生しているようです。これは、`console.log` に表示されるメッセージ “the application myapp has not started in 30.xx seconds”に示されているように、アプリケーションが初期化に時間がかかっているか、全く起動しないことを示唆しています。以下に、この問題を診断し修正するためのステップバイステップのプロセスを案内します。

ステップ 1: サーバーログを調査してヒントを得る

まず、サーバーログを詳細に調査して起動失敗の根本原因を特定します。`console.log` ファイルにはタイムアウトメッセージが含まれていますが、全体像を示していないかもしれません。WLP サーバーのログディレクトリ (`/opt/ibm/wlp/usr/servers/myServer/logs/`) で以下のログファイルを確認してください：

- `messages.log`: このファイルには、依存関係が欠けている、設定エラー、または起動中の例外など、問題を特定するための ERROR または WARNING メッセージが含まれていることが多いです。
- `trace.log`: 詳細なトレースが有効になっている場合、このファイルにはデプロイ中の詳細な情報が含まれているかもしれません。

以下を確認してください：- スタックトレースや例外(例：`ClassNotFoundException`、`NoSuchBeanDefinitionException`)。- リソースが欠けているか、互換性のないライブラリに関するメッセージ。- アプリケーションコンテキストが初期化に失敗したことを示す指示。

詳細が足りない場合、`server.xml` ファイルを修正して WLP のログレベルを増やすことができます。以下のように `<logging>` 要素を追加または更新します：

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

この変更を行った後、サーバーを再起動し、アプリケーションを再デプロイして、さらに情報を得るためにログを再度確認します。

ステップ 2: ログを使用してアプリケーションの起動を確認する

この問題は、アプリケーションコンテキストが初期化に失敗しているため、Spring Boot アプリケーションに関連しているかもしれません。起動プロセスがどれだけ進んでいるかを確認するために、メインアプリケーションクラスに簡単なログステートメントを追加します。以下に例を示します：

```
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");
    }
}
```

- アプリケーションを再構築します (`mvn clean package`)。
- WAR ファイルを WLP の `dropins` ディレクトリに再デプロイします。
- `console.log` でメッセージ `"Application context initialized"` を確認します。

このメッセージが表示される場合、アプリケーションコンテキストは正常に読み込まれており、問題はウェブコンポーネントやサーブレットの初期化に関連しているかもしれません。表示されない場合、問題はコンテキストの初期化の早い段階で発生していることを示します。

ステップ 3: Spring Boot でデバッグログを有効にする

Spring Boot の起動プロセスの可視性を高めるために、デバッグログを有効にします。構成ファイルを作成または編集して、`src/main/resources/application.properties` に以下を追加します：

```
debug=true
```

- アプリケーションを再構築し、再デプロイします。
- `console.log`（または他のログ）で Spring Boot からの詳細なデバッグ出力を確認します。

これにより、ビーンの作成、自動構成、起動中のエラーなどの情報がログに記録されます。起動が遅れているか失敗している原因を示す手がかりを探します。

ステップ 4: WAR ファイルと依存関係の設定を確認する

WLP にデプロイするため、WAR ファイルが外部サーバー用に正しく設定されていることを確認します：

- **WAR パッケージング**: `pom.xml` でパッケージングが `war` に設定されていることを確認します：

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

- **Tomcat を提供**: WLP がサーブレットコンテナを提供するため、WAR ファイルから埋め込み Tomcat を除外します。`pom.xml` で以下を確認します：

```
<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>
```

- **サーブレット API の互換性**: Spring Boot 2.7.18 は `javax.servlet:javax.servlet-api:4.0.1` を使用しており、これは WLP の `javaee-8.0` 特性（サーブレット 4.0）と互換性があります。予期しないサーブレット API バージョン（例：`jakarta.servlet-api`、Spring Boot 3.x で使用され、`javaee-8.0` と互換性がない）がないことを確認するために、以下を実行します：

```
mvn dependency:tree
```

依存関係の問題が疑われる場合、WAR ファイルを解凍して `WEB-INF/lib` を確認し、予期しないサーブレット関連の JAR が含まれていないことを確認します。

ステップ 5: ローカルで問題を特定するためにテストする

問題が WLP に特有か、アプリケーション自体に関連しているかを確認するために、埋め込み Tomcat を使用してアプリケーションをローカルでテストします：

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

成功してアクセスできるエンドポイントがある場合（例：シンプルな `"Hello World!"` REST コントローラー）、問題は WLP のデプロイに関連している可能性が高いです。

ステップ 6: WLP の起動タイムアウトを調整する（一時的な対処法）

ログがアプリケーションが起動しているが 30 秒を超えてかかっていることを示している場合、WLP の `server.xml` で起動タイムアウトを増やすことができます：

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

- アプリケーションを再デプロイし、ログを監視します。
- タイムアウトが延長された後に起動する場合、これは遅い起動プロセスを示しており、アプリケーションを最適化する必要があります（例：コンポーネントのスキャンや初期化タスクを減らす）。

ただし、これは一時的な対処法です。理想的には、シンプルなアプリケーションは 30 秒以内に起動するべきですので、根本原因を調査し続けます。

ステップ 7: 新しいプロジェクトで簡略化して比較する

問題が解決しない場合、WLP にデプロイするための最小限の Spring Boot 2.7.18 プロジェクトを作成します：1. Spring Initializr を使用して以下を設定します：- Spring Boot 2.7.18 - Java（WLP バージョンに一致、例：8 または 11）- 依存関係：Spring Web 2. 基本的な REST コントローラーを追加します：

```
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. WAR デプロイ用に設定します（前述のように `SpringBootServletInitializer` を拡張）。
4. WAR ファイルをビルドします (`mvn clean package`) そして、WLP の `dropins` ディレクトリにデプロイします。

この新しいプロジェクトが正常に起動する場合、その `pom.xml`、`main` クラス、設定を元のプロジェクトと比較して、失敗の原因となる違いを特定します。

ステップ 8: 追加の WLP トラブルシューティング

上記のいずれかが問題を解決しない場合、以下の WLP 特有のチェックを検討してください：- **作業ディレクトリをクリーン**: デプロイに干渉するキャッシュデータを削除します：`bash rm -rf /opt/ibm/wlp/usr/servers/myServer/workarea` サーバーを再起動し、再デプロイします。- **WLP バージョンの確認**: `/opt/ibm/wlp/bin/productInfo version` を実行して、WLP バージョンが Spring Boot 2.7.x デプロイをサポートしていることを確認します。- **Java バージョン**: WLP で使用される Java バージョン（WLP 環境で `java -version` を実行して確認）が Spring Boot 2.7.18 でサポートされているバージョン（Java 8、11、または 17）と一致していることを確認します。

推奨される修正

提供された情報に基づくと、問題はアプリケーションコンテキストの初期化中に発生しているか、WLP との微妙な設定の不一致である可能性が高いです。以下のように修正します：

1. **ログから始める**:
   - `@PostConstruct` ログ（ステップ 2）を追加し、`debug=true` を有効にします（ステップ 3）。
   - 再デプロイし、`console.log` と `messages.log` をエラーや起動が停止している場所を確認します。
2. **設定の検証**:
   - WAR ファイルの設定と依存関係を確認します（ステップ 4）。
   - ローカルでテストします（ステップ 5）でアプリケーションレベルの問題を排除します。
3. **繰り返しと簡略化**:

- 必要に応じて、問題を特定するために最小限のプロジェクトをテストします（ステップ 7）。

これらのステップに従うことで、問題の原因を特定し、依存関係の競合、遅い起動コンポーネント、または WLP の設定問題であるかを解決することができます。問題が解決しない場合は、ログから具体的なエラーメッセージを共有して、さらにサポートを受けます！