

# Camel for Spring Boot 4.10 トレーニング

第1章: イントロダクション	4
● 1.1 本トレーニングの目的と構成	4
● 1.2 対象とする技術スタックとバージョン	4
● Apache Camel 4.10	4
● Spring Boot 3.x	4
● Spring XML DSL	4
● 1.3 前提知識と学習ゴール	5
第2章: KarafとSpring Bootの違いと移行観点	6
● 2.1 Karaf(OSGi)とSpring Bootの構造的違い	6
○ バンドル管理 vs fat JAR	6
○ Blueprint DSL vs Spring XML DSL	6
○ JMX, コンソール, ログ管理の違い	7
● 2.2 Camelコンテキストの起動・停止管理の違い	7
● 2.3 移行時の注意点	7
○ OSGi依存コードの排除	7
○ ServiceMix・Features定義の削除	8
○ デプロイ方法の変更(Karaf deploy → Spring Boot JAR実行)	8
第3章: Camel 4.10 のアーキテクチャ変更点	9
● 3.1 Camel 3 → 4 の主な変更点	9
● 3.2 削除されたコンポーネントと代替策	9
● 3.3 XML DSL構文変更(例: <description>の属性化など)	10
● 3.4 ストリームキャッシュとスプール設定	11
● 3.5 CamelContextの構成と拡張の変化(ExtendedCamelContextなど)	12
第4章: プロジェクトセットアップと構成管理	14
● 4.1 プロジェクト構成(Maven)	14
● 4.2 Spring BootによるCamelContextの起動方法	15
● 4.3 Spring XML DSLの読み込みとルート登録	16
XMLファイルの配置	16
● 4.4 application.propertiesによる構成管理とKarafとの比較	16
代表的な構成ファイル	17
設定例	17
特徴比較	17
第5章: ルート開発(XML DSL)	19
● 5.1 基本構文と構造	19
最小構成の例	19
● 5.2 Karaf Blueprint DSLとの相違点	19
● 5.3 EIPの使い方と変更点(InOnly削除など)	20
代表的なEIPの例	20
変更点: InOnlyの削除	21
● 5.4 errorHandler/onExceptionの定義方法	21
グローバルエラーハンドラ	21
● 5.5 REST DSLと<jetty>等の代替に関する実装注意点	22
REST DSLの使用例(XML DSL)	22
注意点	22

<b>第6章: テストとトラブルシューティング</b>	<b>24</b>
● 6.1 camel-test-spring + JUnit5 の活用	24
依存関係 (pom.xml)	24
テスト例	24
● 6.2 Spring Bootログ構成 (access, gc, consoleログなど)	25
標準構成ファイル: logback-spring.xml	25
ログ種別と対応表	25
● 6.3 Actuatorによるヘルスチェック	26
依存追加 (pom.xml)	26
設定 (application.properties)	26
出力例	26
● 6.4 BacklogTracer／Tracerの使い方と変更点	27
BacklogTracer: ルートの「最後のメッセージ」を記録	27
Tracer: 全メッセージの逐次トレース (パフォーマンスへの影響大)	28
注意点 (Camel 4.10での変更)	28
Spring Bootでの有効化例 (application.properties)	28
<b>第7章: 運用・監視・ビルド・デプロイ</b>	<b>29</b>
● 7.1 Fat JARの作成と実行方法	29
Fat JAR作成手順 (Maven)	29
● 7.2 Mavenビルドと依存関係管理	30
依存の定義例 (HTTP, CXF, XML DSL)	30
● 7.3 外部プロパティ管理 (ConfigMap非使用前提)	31
プロファイルごとのプロパティファイル	31
実行時にプロファイル指定	31
コマンドラインからの上書きも可能	31
● 7.4 JMX監視とHawtio (Spring Bootでの使い方)	31
JMX有効化	31
Hawtio導入 (埋め込み)	32
<b>第8章: 移行演習</b>	<b>34</b>
● 8.1 Blueprintルート → Spring XML DSLへの変換演習	34
Blueprint DSL での例 (before)	34
Spring XML DSL での例 (after)	34
変換ポイント	35
● 8.2 cxf: コンポーネントを含むSOAPルートの移行	35
Blueprint DSL の例 (before)	35
Spring XML DSL の例 (after)	36
注意点	36
● 8.3 ルーティングとException処理の構造再現演習	37
Blueprint DSL の例 (before)	37
Spring XML DSL の例 (after)	37
ポイント	37
<b>第9章: まとめと今後の運用設計</b>	<b>39</b>
● 9.1 Camel for Spring Bootの導入効果	39
● 9.2 段階的移行戦略 (モジュール単位移行、ゼロダウンタイム等)	40

推奨される段階的移行アプローチ	40
● 9.3 継続的改善に向けた運用設計のポイント	40
1. 監視と可視化の徹底	40
2. 設定の一元管理	40
3. CI/CDパイプラインとの統合	41
4. Camelアップデートと互換性対応	41

# 第1章: イントロダクション

## ● 1.1 本トレーニングの目的と構成

本トレーニングの目的は、**Apache Camel 4.10** と **Spring Boot 3.x** をベースにした **Spring XML DSL** によるアプリケーション構築方法を習得し、従来の **Fuse/Karaf (Blueprint DSL)** 環境からの移行に必要な知識と実践スキルを身に付けることです。

また、KarafからSpring Bootベースへの移行時に遭遇する構成・起動・運用の違いを理解し、Camel 4.x の新機能や変更点に対応できるようになることを目的としています。

本トレーニングでは以下の構成で学習を進めます：

- 第1章: イントロダクション (目的・前提・技術スタック)
- 第2章: KarafとSpring Bootの比較と移行観点
- 第3章: Camel 4.10の技術的な変更点
- 第4章: プロジェクトセットアップと構成管理
- 第5章: Spring XML DSLによるルート開発
- 第6章: テスト・トラブルシューティング
- 第7章: 運用・監視・デプロイ
- 第8章: 演習: Blueprint → Spring XML DSLへの移行実践
- 第9章: まとめと今後の運用設計

各章では、実践的な構成例やコードスニペットを交え、移行時に特に注意すべき点を重点的に解説します。

## ● 1.2 対象とする技術スタックとバージョン

このトレーニングでは以下の技術スタックを採用します：

- **Apache Camel 4.10**
  - 最新のルーティングエンジン、リファクタリングされたモジュール体系
  - Blueprint DSL非対応、Spring DSLの明示的な採用が必要
- **Spring Boot 3.x**
  - Jakarta EE 10対応 (`javax.*` → `jakarta.*`)
  - 自動構成、Spring Actuator、プロファイルなどの運用支援機能
- **Spring XML DSL**
  - Camelルートの記述に <http://camel.apache.org/schema/spring> 名前空間を使用
  - Spring ApplicationContextに統合され、Blueprint DSLに類似した宣言的記述が可能

- `<camelContext>`, `<route>`, `<bean>`, `<errorHandler>`, `<onException>` 等の標準要素が利用可能

## ● 1.3 前提知識と学習ゴール

### ■ 前提知識

このトレーニングの内容を効果的に理解するために、以下の知識を有していることが望めます：

- Java / Maven ベースの業務アプリケーション開発経験
- Apache Camelの基本構文とEIPの理解(split、choice、multicastなど)
- Karaf(OSGiベース)およびBlueprint DSLによるCamelルートの開発経験
- Spring Framework(DI、ApplicationContext)の基礎知識

### ■ 学習ゴール

トレーニングの完了時点で、以下の状態に到達していることを目指します：

- Spring Boot上でCamelルートをSpring XML DSLで定義・動作させられる
  - Blueprint DSLで書かれたルートをSpring XML DSLへ移行できる
  - Spring Bootにおける構成・起動・依存解決の方式を理解している
  - Camelルートの例外処理、プロパティ定義、外部連携を適切に実装・検証できる
-

## 第2章: KarafとSpring Bootの違いと移行観点

Apache Camelのルートは、Karaf上でもSpring Boot上でも動作しますが、それぞれのプラットフォームはアーキテクチャ的に大きく異なります。本章では、Karaf(OSGiコンテナ)とSpring Bootの構造的な違いを理解した上で、Camel 4.10+ Spring XML DSLへの移行時に考慮すべき要点を解説します。

### ● 2.1 Karaf(OSGi)とSpring Bootの構造的違い

#### ○ バンドル管理 vs fat JAR

**Karaf(OSGi)** は、機能ごとに分割されたバンドル(JAR)を動的に読み込み・起動・停止するモジュール指向のプラットフォームです。各バンドルは`MANIFEST.MF`内でインポート/エクスポートされるパッケージを明示的に宣言し、サービスレジストリを介して連携します。

**Spring Boot** は、すべての依存JARとアプリケーションコードを1つの**Fat JAR**にビルドし、`java -jar`で実行する単一プロセス指向のランタイムです。OSGiのようなバンドル分離やクラスローダ制御は存在せず、クラスパースペースの依存管理が前提です。Fat JARは、すべての依存関係を内包するため、コンテナイメージのビルドを簡素化し、どこでも同じように動作する不変のデプロイ単位を提供します

#### ○ Blueprint DSL vs Spring XML DSL

**Blueprint DSL(Karaf)** は、<http://www.osgi.org/xmlns/blueprint/v1.0.0>名前空間に基づき、CamelルートとJavaBeanを宣言的に記述します。

**Spring XML DSL(Spring Boot)** は、<http://camel.apache.org/schema/spring>名前空間を使用し、CamelルートとBeanをSpring ApplicationContextに統合して定義します。

両者は構文が非常に類似しており、構成要素(`<camelContext>`、`<route>`、`<bean>`、`<onException>`など)に共通性がありますが、Spring XML DSLでは**Blueprint**特有のID参照方式や**OSGi**サービス参照が使用できません。

- JMX, コンソール, ログ管理の違い

**Karaf** は、`karaf shell` によるOSGiコンソール、`log:tail`等のリアルタイムログ確認、`jconsole`によるJMX接続が統合されています。

**Spring Boot** は、`Actuator`エンドポイント(例: `/actuator/health`, `/actuator/logfile`)や`logback-spring.xml`によるログ制御を用い、JMX接続は`spring.jmx.enabled=true`で明示的に有効化されます。

移行後は、Karafコンソールに相当する操作はHawtioやSpring Boot Admin、Actuator経由の管理が基本となります。

- 2.2 Camelコンテキストの起動・停止管理の違い

**Karaf** では、Camelコンテキストは各バンドル内の`blueprint.xml`定義をもとに、自動的にOSGiフレームワークから起動・停止されます。

**Spring Boot** では、CamelContextは`@SpringBootApplication`により起動される`SpringApplication.run()`の中でライフサイクルを管理します。

- 起動時に`camel-spring-boot`により自動的にCamelContextがインスタンス化され、Spring XML DSLに定義されたルートが構成に追加されます。
- 停止は`Ctrl+C`や`ApplicationContext.close()`により行われ、CamelContextも連動して停止されます。

起動制御のカスタマイズは、Karafではバンドルの状態依存、Spring Bootでは`MainConfiguration`や`CamelContextConfiguration`による明示的な設定が必要になります。

- 2.3 移行時の注意点

- OSGi依存コードの排除

- Blueprint DSLでは、`<reference interface="..." />`や、OSGi Declarative Services(DS)に基づく`org.osgi.service`系のサービス



参照が使われることがあります。

- Spring Bootでは、これらのOSGiサービス連携は利用できないため、明示的なDIまたはSpring管理Beanへの書き換えが必要です。

- ServiceMix・Features定義の削除

- Karafでは、`features.xml`によってCamelコンポーネントや依存ライブラリを動的に解決・インストールしていました。
- Spring Bootでは、`pom.xml`の`<dependencies>`内で依存ライブラリを明示的に指定する必要があります。

- 例:`camel-cxf`, `camel-jackson`, `camel-kafka` など

- デプロイ方法の変更 (Karaf deploy → Spring Boot JAR実行)

- Karaf: `deploy/`ディレクトリにバンドルを配置
- Spring Boot: `mvn package`でfat JARを作成し、`java -jar app.jar`で起動
- 実行方法、ログ取得、プロファイル切替などの運用方法も大きく異なるため、運用部門への引き継ぎ資料も必要となります。

この章では、KarafとSpring Bootの基本的な思想と構造の違いを理解し、設計・構築・運用時に必要な移行対応の全体像を把握しました。次章では、Camel 4.10における技術的な変更点と、Blueprint DSLからの移行に影響する仕様を確認していきます。

---

## 第3章: Camel 4.10 のアーキテクチャ変更点

Apache Camel 4系は、Camel 3系から大幅な内部構造の見直しとAPIの整理が行われたバージョンです。特にBlueprint DSLの削除、コンポーネント構成の分割、CamelContextの抽象化など、移行プロジェクトに影響を及ぼす変更点が多数存在します。本章では、Camel 4.10への移行時に把握すべき主なアーキテクチャ変更点について解説します。

### ● 3.1 Camel 3 → 4 の主な変更点

#### Blueprint DSLの削除

Camel 4では <http://camel.apache.org/schema/blueprint> 名前空間のDSLが完全に削除されました。Blueprint XMLで構築されたKaraf環境のルートは、**Spring XML DSL**または**Java DSL**への移行が必須です。

#### APIの整理とスリム化

内部APIが整理され、非推奨だった多くのメソッド・インターフェースが削除されました。特に `Exchange.setException(Throwable)` の扱いや、ルート内での型変換の自動化に関する挙動がCamel 3とは異なります。

#### 依存ライブラリの最小化

Camel 4はコンポーネント単位での依存により、不要なライブラリを含めずにすむ設計に変更されました。これにより、Mavenで使用する依存は明示的に定義する必要があります。

#### Jakarta EE への移行対応

Camel 4では、Spring Boot 3に合わせて `javax.*` パッケージの使用が廃止され、すべて `jakarta.*` へ移行しています。独自コードやライブラリもこれに対応する必要があります。

### ● 3.2 削除されたコンポーネントと代替策

Camel 4では、以下のようなコンポーネントが削除されています：

削除されたコンポーネント	代替手段
<code>camel-jetty</code>	<code>camel-undertow</code> または REST DSL + embedded server

<code>camel-saxon</code>	JavaベースのXML処理または <code>camel-xslt</code>
<code>camel-servlet</code>	Spring Boot組み込みサーバ(Tomcat等)
<code>camel-netty</code>	<code>camel-netty4</code> または <code>camel-grpc</code> 等
<code>camel-twitter</code>	REST API を使用して独自実装へ

Blueprint DSL自体もコンポーネントとして削除されており、ルート定義は `camel-spring-xml` に統一されます。

✓ 対策: `pom.xml` にて使用するすべてのコンポーネントを明示的に指定し、Blueprint依存のルートや設定は段階的にSpring XML形式へリファクタリングする必要があります。

### ● 3.3 XML DSL構文変更(例: `<description>`の属性化など)

Camel 4では、XML DSLの構文にも以下のような変更があります:

- `<description>` 要素の記法変更  
Camel 3までは `<description>xxx</description>` の形式でしたが、Camel 4では `<description text="xxx"/>` のように属性形式に統一されました。
- `<bean>` の参照方式の明確化  
Blueprint DSLで可能だった `id-ref` 形式は使用できず、`ref="beanName"` に統一されます。
- `<route>` や `<from>` 要素内での`uri`属性の記法も一部で厳密化され、スペースや解釈の曖昧さを排除しています。

✓ 対策: 既存のBlueprint DSLをSpring XML DSLに移行する際は、構文チェックとXSDベースのバリデーションを活用することを推奨します。

## ● 3.4 ストリームキャッシュとスプール設定

Camel 4では、大きなメッセージのメモリ圧迫を防ぐためのスプール(**spooling**)設定が強化されています。


- デフォルトでは、Camelはメッセージボディの一部をストリームとして処理します(例: HTTP、ファイルなど)。
- ストリームキャッシュが有効な場合、一定サイズ以上のメッセージはメモリに保持されず、一時ファイルにスプールされます。

設定例(Spring Bootの`application.properties`):

```
Unset
camel.springboot.streamCachingEnabled=true
camel.springboot.streamCachingSpoolEnabled=true
camel.springboot.streamCachingSpoolThreshold=4096
```

また、XML DSL内でも以下のように定義可能です:

```
XML
<camelContext streamCache="true"
xmlns="http://camel.apache.org/schema/spring">
  <streamCaching spoolDirectory="/tmp/spool" spoolThreshold="4096" />
</camelContext>
```

 推奨: スプール先ディレクトリのマウント領域やパーミッションに注意し、ログ出力でスプール使用量を定期的を確認すること。

## ● 3.5 CamelContextの構成と拡張の変化(ExtendedCamelContextなど)

Camel 4では、`ExtendedCamelContext`は廃止され、`CamelContext`インターフェース自体が拡張されています。そのため、従来のように`ExtendedCamelContext`を明示的に取得して設定することは不要です。

例(旧方式):

```
Java
ExtendedCamelContext ecc = (ExtendedCamelContext) camelContext;
ecc.setXXX(...);
```

例(新方式):

```
Java
camelContext.setXXX(...); // CamelContextインターフェースが直接提供
```

また、CamelContextの構成要素(TypeConverter、Registry、ComponentResolverなど)はより明確に分離され、プログラムの構成しやすくなっています。

Spring Bootでは、[CamelContextConfiguration](#)を使ってBean定義からの拡張が可能です:

```
Java
@Bean
CamelContextConfiguration contextConfiguration() {
    return camelContext -> {
        camelContext.setStreamCachingStrategy(...);
    };
}
```

✅ ポイント: 従来の拡張手法を見直し、[CamelContext](#)の直接APIかSpring DI経由の構成を採用する。

本章では、Camel 4.10における技術的な変更点を整理しました。次章では、Spring BootプロジェクトにおけるCamel構成と、Spring XML DSLのルート定義方法を実践的に解説します。

## 第4章:プロジェクトセットアップと構成管理

本章では、Camel 4.10 と Spring Boot 3.x をベースとしたアプリケーションのプロジェクト構成、CamelContext の起動方法、Spring XML DSL の読み込み方法、構成ファイルの扱いについて解説します。Karaf(OSGi)とは異なり、Spring Bootではプロジェクト起動・構成・ルート登録のアプローチが統一されています。

### ● 4.1 プロジェクト構成(Maven)

Camel for Spring Boot プロジェクトは、通常の Spring Boot アプリケーションと同様に Maven プロジェクトとして構成されます。以下は代表的な `pom.xml` の例です。

```
XML
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>camel-springboot-app</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <properties>
    <java.version>21</java.version>
    <camel.version>4.10.3.redhat-00019</camel.version>
    <spring-boot.version>3.4.5</spring-boot.version>
  </properties>

  <dependencies>
    <!-- Spring Boot Starter -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <!-- Camel Spring Boot Starter -->
    <dependency>
      <groupId>org.apache.camel.springboot</groupId>
      <artifactId>camel-spring-boot-starter</artifactId>
    </dependency>

    <!-- XML DSL サポート -->
    <dependency>
      <groupId>org.apache.camel.springboot</groupId>
      <artifactId>camel-spring-boot-xml-starter</artifactId>
    </dependency>
  </dependencies>
</project>
```

```

<!-- 使用するコンポーネント例（例：HTTP） -->
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-http-starter</artifactId>
</dependency>
</dependencies>

<build>
  <plugins>
    <!-- Spring Boot Plugin -->
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

## ● 4.2 Spring BootによるCamelContextの起動方法

Spring Bootでは、`@SpringBootApplication` により `ApplicationContext` が起動され、それに付随して `CamelContext` も自動的に起動されます。開発者が明示的に `CamelContext` を生成・起動する必要はありません。

```

Java
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

```

`CamelContext`の設定を追加する場合は、`CamelContextConfiguration` を使って以下のよう  
に記述します：

```

Java
@Bean
CamelContextConfiguration contextConfiguration() {
    return camelContext -> {
        camelContext.setStreamCaching(true);
    };
}

```

```
// 必要な設定を追加
};
}
```

## ● 4.3 Spring XML DSLの読み込みとルート登録

Karafの Blueprint DSL では `blueprint.xml` に定義することで自動的にルートが読み込まれていましたが、Spring Boot では `resources` 配下に配置された XML ファイルを自動でスキャンして読み込む形になります。

### XMLファイルの配置

以下のようなXMLを `src/main/resources/camel/routes.xml` に配置します：

```
XML
<routes xmlns="http://camel.apache.org/schema/spring">
  <route id="example-route">
    <from uri="timer:foo?period=5s"/>
    <setBody>
      <constant>Hello Camel 4.10</constant>
    </setBody>
    <log message="Received: ${body}"/>
  </route>
</routes>
```

`application.properties` による有効化（詳細は次節）

```
Unset
camel.springboot.xmlRoutes=classpath:camel/routes.xml
```

## ● 4.4 `application.properties`による構成管理とKarafとの比較

Karaf環境では、構成管理は以下のような形で分散されていました：

- OSGi Config Admin(`etc/*.cfg` ファイル)
- Blueprint XML による `<property-placeholder>`
- フラグメントバンドルでの動的設定提供



一方、Spring Bootでは、以下に統一されます：

代表的な構成ファイル

- `src/main/resources/application.properties`  
または
- `application.yaml`

設定例

```
Unset
# Camel XML DSL ルート読み込み
camel.springboot.xmlRoutes=classpath:camel/routes.xml

# ストリームキャッシュの有効化
camel.springboot.streamCachingEnabled=true

# コンテキスト名の定義
camel.springboot.name=camel-springboot-app

# Spring Bootのサーバ設定
server.port=8080
```

特徴比較

観点	Karaf	Spring Boot
設定管理方法	etc/*.cfg, Blueprint	application.properties / yaml
プロファイル切替	Fabric8 / ConfigAdmin	Spring profiles( <code>--spring.profiles.active</code> )
ランタイム変更の 反映	OSGiバンドル再起動/reload	再起動/Spring Cloud Configを併用 可能
複数設定の管理	Feature定義、フラグメントバンドル	プロファイル別設定ファイル

この章では、Spring Boot上でのCamelアプリケーションのセットアップ手順、XML DSLルートの登録方法、Karaf環境からの構成管理の移行方法を整理しました。

次章では、Spring XML DSLでのルート定義方法や主要EIPの記述方法について詳しく解説します。

---

## 第5章: ルート開発(XML DSL)

本章では、Camel 4.10においてSpring XML DSL(<http://camel.apache.org/schema/spring>)を使用してルートを定義する方法について解説します。特にKaraf環境で使用されていたBlueprint DSLとの違いや、EIP(エンタープライズ統合パターン)の記述方式の変化、例外処理やREST定義の実装注意点に焦点を当てます。

### ● 5.1 基本構文と構造

Spring XML DSLでは、Camelのルートは<camelContext>の中に<route>を定義することで記述されます。

最小構成の例

```
XML
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         https://www.springframework.org/schema/beans/spring-beans.xsd
         http://camel.apache.org/schema/spring
         https://camel.apache.org/schema/spring/camel-spring.xsd">

  <camel:camelContext id="camelContext">
    <camel:route id="hello-route">
      <camel:from uri="timer:hello?period=5s"/>
      <camel:setBody>
        <camel:constant>Hello from Camel XML DSL</camel:constant>
      </camel:setBody>
      <camel:log message="Message: ${body}"/>
    </camel:route>
  </camel:camelContext>
</beans>
```

構成ファイルは `application.properties` で `camel.springboot.xmlRoutes` に登録することで自動的に読み込まれます。

### ● 5.2 Karaf Blueprint DSLとの相違点

Spring XML DSLとBlueprint DSLは構文が似ているため、移行は比較的スムーズです。ただし、以下の点に注意が必要です:

比較項目	Karaf Blueprint DSL	Spring XML DSL
------	---------------------	----------------

名前空間	<code>http://camel.apache.org/schema/blueprint</code>	<code>http://camel.apache.org/schema/spring</code>
コンテキストルート定義	<code>&lt;camelContext id="..."&gt;</code>	<code>&lt;camelContext id="..." xmlns="camel"&gt;</code>
Bean定義	OSGi参照( <code>&lt;reference interface="..." /&gt;</code> など)	Spring Beans( <code>&lt;bean class="..." /&gt;</code> )
Property参照	<code>&lt;property-placeholder&gt;</code> (Blueprint定義)	Spring Boot標準の構成管理を使用
OSGiサービス利用	<code>&lt;reference&gt;</code> でOSGiサービス参照が可能	Spring DIのみ利用可能

Blueprint固有の構文やOSGi依存のサービス参照は、すべてSpringコンテキストベースに書き換える必要があります。

### ● 5.3 EIPの使い方と変更点(InOnly削除など)

Camel 4では一部EIPの仕様や記法が変更されています。

代表的なEIPの例

XML

```
<route id="content-based-router">
  <from uri="direct:start"/>
  <choice>
    <when>
      <simple>${body} contains 'Camel'</simple>
      <to uri="log:camel"/>
    </when>
    <otherwise>
      <to uri="log:other"/>
    </otherwise>
  </choice>
</route>
```

```
</choice>
</route>
```

### 変更点: InOnlyの削除

Camel 4.10では `<inOnly>` および `<inOut>` 要素は **XML DSL**から削除され、代わりに `<exchangePattern pattern="InOnly"/>` 属性を使用する必要があります。

例(InOnlyの代替):

```
XML
<to uri="seda:asyncQueue" pattern="InOnly"/>
```

✅ Blueprint XMLで `<inOnly>` を使っていた場合は、`<to>` 要素に `pattern="InOnly"` を付与するように変換してください。

## ● 5.4 errorHandler/onExceptionの定義方法

Camelでは、例外処理をルートごとまたはグローバルに定義できます。

### グローバルエラーハンドラ

```
XML
<camelContext id="ctx" xmlns="http://camel.apache.org/schema/spring">
  <errorHandler id="defaultErrorHandler" type="DefaultErrorHandler">
    <redeliveryPolicy maximumRedeliveries="3" redeliveryDelay="2000"/>
  </errorHandler>

  <route id="example">
    <from uri="direct:start"/>
    <to uri="bean:someProcessor"/>
  </route>
</camelContext>
```

### onExceptionの例

XML

```
<onException>
  <exception>java.lang.Exception</exception>
  <handled>
    <constant>true</constant>
  </handled>
  <log message="Exception occurred: ${exception.message}"/>
  <setBody>
    <simple>Error: ${exception.message}</simple>
  </setBody>
</onException>
```

✓ **handled** や **continued** は Camel 4でも利用可能で、より柔軟なフロー制御が可能です。

## ● 5.5 REST DSLと<jetty>等の代替に関する実装注意点

Camel 4.10では、Karafで使用されていた <jetty> コンポーネントや <servlet> コンポーネントは削除または非推奨になっており、代替として **Spring Boot**の組み込みサーバ(Tomcatなど)や REST DSL を使用します。

### REST DSLの使用例(XML DSL)

XML

```
<restConfiguration component="servlet" port="8080"/>

<rest path="/hello">
  <get uri="/{name}">
    <to uri="direct:sayHello"/>
  </get>
</rest>

<route id="say-hello-route">
  <from uri="direct:sayHello"/>
  <setBody>
    <simple>Hello, ${header.name}!</simple>
  </setBody>
</route>
```

### 注意点

- **jetty** や **servlet** を使っていた既存ルートは、**REST DSL + Spring Boot**組み込みサーバ構成へ移行する必要があります。

- URIパスのルーティングやCORS設定などはSpring Boot側で行うことが望ましい場合があります。

✅ Camel 4では `camel-undertow` または REST DSL の利用が公式に推奨されています。

---

この章では、Camel 4.10におけるSpring XML DSLによるルート開発の基本、Blueprintからの差異、主要EIPの書き方、例外処理の定義方法、RESTの構築方法について解説しました。次章では、ルートのテスト、ログ出力、トラブルシュートの方法に進みます。

## 第6章: テストとトラブルシューティング

本章では、Camel for Spring Bootアプリケーションの品質保証および障害解析の手法として、単体テストの実施方法、ログ構成の管理、ヘルスチェックの実装、および**Camel**組み込みのトレース機能について解説します。Karafとは異なり、Spring Bootでは統一されたログおよびテスト基盤を活用できます。

### ● 6.1 camel-test-spring + JUnit5 の活用

Camelルートの単体テストは、**camel-test-spring-junit5** を利用することで容易に実現できます。

依存関係 (**pom.xml**)

```
XML
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-test-spring-junit5</artifactId>
  <scope>test</scope>
</dependency>
```

テスト例

```
Java
@CamelSpringBootTest
@SpringBootTest
public class SampleRouteTest {

    @Autowired
    private ProducerTemplate producerTemplate;

    @Test
    void testRoute() {
        String result = producerTemplate.requestBody("direct:hello", "Keiji",
String.class);
        assertEquals("Hello Keiji", result);
    }
}
```

- **@CamelSpringBootTest** により CamelContext を起動
- **ProducerTemplate** を使ってルートを呼び出し
- 実行結果を **assertEquals()** で検証



✔ Blueprint環境でのテストより簡潔で、DIやMockを使ったテストも容易です。

## ● 6.2 Spring Bootログ構成(access, gc, consoleログなど)

Karafでは `console.log` や `fuse.log` などに分かれていましたが、Spring Bootでは logback (または log4j2) により一元的に管理されます。

標準構成ファイル: `logback-spring.xml`

```
XML
<configuration>
  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} -
        %msg%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="CONSOLE" />
  </root>

  <logger name="org.apache.camel" level="DEBUG" />
</configuration>
```

ログ種別と対応表

目的	Karafログファイル例	Spring Bootでの扱い
アクセスログ	access.log	filterやinterceptorで明示的に出力
Web UIアクセスログ	web_mng_access.log	N/A(外部UIに依存)
GCログ	gc_pid~.log	JVM引数で出力設定(例: <code>-Xlog:gc</code> )

コンソールログ	console.log	標準出力 (logback構成で制御)
Camelログ	fuse.log / camel.log	logback上で <code>org.apache.camel</code> を制御

✓ 特殊ログ (GCなど) は JVM レベルで制御、Camel特有のログは logger 名で細かく出力レベルを調整できます。

## ● 6.3 Actuatorによるヘルスチェック

Spring Bootでは、`spring-boot-starter-actuator` を導入することで `/actuator/health` 等のエンドポイントが有効になり、プローブや監視が容易になります。

依存追加 (`pom.xml`)

```
XML

<dependency>

  <groupId>org.springframework.boot</groupId>

  <artifactId>spring-boot-starter-actuator</artifactId>

</dependency>
```

設定 (`application.properties`)

```
Unset

management.endpoints.web.exposure.include=health,info

management.endpoint.health.show-details=always
```

出力例

```
JSON

{
```

```

    "status": "UP",
    "components": {
      "camel": {
        "status": "UP",
        "details": {
          "context": "camel-springboot-app",
          "status": "Started",
          "uptime": "5 minutes"
        }
      }
    }
  }
}

```

✓ Camelの状態も `/actuator/health` に統合されており、ルートの死活監視にも活用可能です。

## ● 6.4 BacklogTracer／Tracerの使い方と変更点

Camelには、ルート実行のトレース機能として `Tracer` および `BacklogTracer` が用意されています。

**BacklogTracer**: ルートの「最後のメッセージ」を記録

```

Java

BacklogTracer tracer = camelContext.getExtension(BacklogTracer.class);

tracer.setEnabled(true);

tracer.setBacklogSize(100);

tracer.dumpTracedMessages("myRouteId").forEach(System.out::println);

```

## Tracer: 全メッセージの逐次トレース(パフォーマンスへの影響大)

```
Java
Tracer tracer = new Tracer();
tracer.setLogName("my.camel.trace");
tracer.setLogLevel("INFO");
camelContext.addInterceptStrategy(tracer);
```

### 注意点(Camel 4.10での変更)

- `camelContext.getBacklogTracer()` は非推奨
- `getExtension()` で明示的にトレース機能を取得する必要あり
- パフォーマンスへの影響があるため、本番環境では原則無効化推奨

### Spring Bootでの有効化例(application.properties)

```
Unset
camel.springboot.backlogTracerEnabled=true
camel.springboot.backlogTracerBacklogSize=200
```

✅ トレースはユニットテストやPoC検証時に有効。運用ではActuatorやログでの代替監視を推奨。

この章では、Camel for Spring Bootにおけるテスト実装、ログ管理、ヘルスチェック、ルートトレースの手法を整理しました。次章では、アプリケーションのビルド、実行、運用監視に関する具体的な手順と推奨設定を解説します。

## 第7章:運用・監視・ビルド・デプロイ

本章では、Camel for Spring Boot アプリケーションの運用・監視・デプロイに関する基本的な方法を解説します。従来のKarafとは異なり、Spring Bootでは fat JAR を用いた単一プロセスの起動や Actuator ベースの監視が主流となります。また、外部構成管理やJMX、Hawtioの活用についても触れます。

### ● 7.1 Fat JARの作成と実行方法

Camel for Spring Boot アプリケーションは、すべての依存を含む **Fat JAR**(実行可能**JAR**) としてビルド・配布されます。

#### Fat JAR作成手順(Maven)

1. `pom.xml` に Spring Boot Plugin を追加済みであることを確認:

```
XML
<build>

  <plugins>

    <plugin>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-maven-plugin</artifactId>

    </plugin>

  </plugins>

</build>
```

2. 以下のコマンドでビルド:

```
Shell
mvn clean package
```

3. 作成されたJARファイルを実行:

Shell

```
java -jar target/camel-springboot-app-1.0.0.jar
```

✓ `application.properties` で `server.port` を変更すれば任意のポートで起動可能です。

## ● 7.2 Mavenビルドと依存関係管理

Camel 4.10では、コンポーネントベースでの依存管理が徹底されています。使用するコンポーネントを明示的に `pom.xml` に定義する必要があります。

依存の定義例 (**HTTP, CXF, XML DSL**)

XML

```
<dependencies>
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-spring-xml-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-http-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-cxf-starter</artifactId>
  </dependency>
</dependencies>
```

✓ Karafの `features.xml` による依存管理とは異なり、ビルド時にすべての依存を解決するのがSpring Boot方式です。

## ● 7.3 外部プロパティ管理 (ConfigMap非使用前提)

Spring Bootでは、環境ごとに設定ファイルを分けることができます。外部構成管理にKubernetesのConfigMapなどを使わない場合でも、以下のように柔軟に管理できます。

プロファイルごとのプロパティファイル

```
Unset
# application-dev.properties
server.port=8081
camel.springboot.name=camel-app-dev
```

```
Unset
# application-prod.properties
server.port=8080
camel.springboot.name=camel-app-prod
```

実行時にプロファイル指定

```
Shell
java -jar target/app.jar --spring.profiles.active=dev
```

コマンドラインからの上書きも可能

```
Shell
java -jar app.jar --server.port=9090
```

✓ OSGi ConfigAdminや `.cfg` ファイルのような構成は不要。単一ファイルかつ明快な優先順位制御 (コマンドライン > プロファイル > デフォルト) で管理できます。

## ● 7.4 JMX監視とHawtIO (Spring Bootでの使い方)

KarafではJMXおよびKaraf Console経由でのルート監視が一般的でしたが、Spring BootでもJMX監視と **HawtIO** を組み合わせて利用できます。

**JMX有効化**

`application.properties` に以下を追加:

```
Unset
spring.jmx.enabled=true

camel.springboot.jmxEnabled=true
```

JMXを有効化すると、`jconsole` や `VisualVM` で CamelContext、ルート、プロセッサなどのメトリクスを確認可能になります。

### Hawtio導入(埋め込み)

#### 1. 依存追加:

```
XML
<dependency>
  <groupId>io.hawt</groupId>
  <artifactId>hawtio-springboot</artifactId>
  <version>4.2.0.redhat-00034</version>
</dependency>
```

#### 2. `application.properties` 設定例:

```
Unset
hawtio.authenticationEnabled=false

hawtio.proxyAllowed=true

management.endpoints.web.exposure.include=*
```

#### 3. アクセス方法:

```
Unset
http://localhost:8080/hawtio
```



ログイン不要でCamelルートの状態や履歴、メッセージのトレースなどが可視化できます。

✅ Karafと同様の視認性をHawtioで確保可能。軽量なUIツールとしてローカル環境・検証環境に最適です。

---

この章では、Camel for Spring Bootアプリケーションのビルド方法 (fat JAR)、構成管理、JMX・Hawtioを活用した監視・運用管理の方法について解説しました。

次章では、Blueprint DSLからSpring XML DSLへの実践的な移行演習に進みます。

## 第8章:移行演習

本章では、Karaf環境で作成されたBlueprint DSLベースのCamelルートを、Camel 4.10+ Spring Boot 3.x環境で動作する**Spring XML DSL**へ変換する演習を通じて、実践的な移行スキルを養います。特に、`<cxfr:cxfrEndpoint>`を含むSOAPルートや、例外処理付きのルート構造再現を中心に解説します。

### ● 8.1 Blueprintルート → Spring XML DSLへの変換演習

#### Blueprint DSL での例 (before)

```
XML
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:camel="http://camel.apache.org/schema/blueprint">

  <camel:camelContext id="ctx">
    <camel:route id="sample-route">
      <camel:from uri="direct:start"/>
      <camel:setBody>
        <camel:constant>Hello Blueprint</camel:constant>
      </camel:setBody>
      <camel:to uri="log:result"/>
    </camel:route>
  </camel:camelContext>
</blueprint>
```

#### Spring XML DSL での例 (after)

```
XML
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         https://www.springframework.org/schema/beans/spring-beans.xsd
         http://camel.apache.org/schema/spring
         https://camel.apache.org/schema/spring/camel-spring.xsd">

  <camel:camelContext id="ctx">
    <camel:route id="sample-route">
      <camel:from uri="direct:start"/>
      <camel:setBody>
        <camel:constant>Hello Spring DSL</camel:constant>
      </camel:setBody>
    </camel:route>
  </camel:camelContext>
</beans>
```

```

        </camel:setBody>
        <camel:to uri="log:result"/>
    </camel:route>
</camel:camelContext>
</beans>

```

## 変換ポイント

要素	Blueprint	Spring XML DSL
名前空間	schema/blueprint	schema/spring
camelContext配置	<blueprint>タグ配下	<beans>タグ配下
DI (Bean定義)	OSGi <reference> 等	Spring <bean>

✓ Blueprint特有の構文(<reference>, <property-placeholder>)は削除し、Spring標準構文で置き換える必要があります。

## ● 8.2 cxf: コンポーネントを含むSOAPルートの移行

KarafでCXFを使っていたルートも、Camel 4.10 + Spring Boot環境では `camel-cxf-starter` により同様に実現可能です。

### Blueprint DSL の例 (before)

XML

```

<camel:route id="soapRoute">

    <camel:from uri="cxf:/hello?serviceClass=com.example.HelloService"/>

    <camel:to uri="bean:helloProcessor"/>

</camel:route>

```

## Spring XML DSL の例(after)

XML

```
<camelContext xmlns="http://camel.apache.org/schema/spring">

  <route id="soapRoute">

    <from uri="cxf:/hello?serviceClass=com.example.HelloService"/>

    <to uri="bean:helloProcessor"/>

  </route>

</camelContext>
```

## 注意点

- Blueprint用の `<cxf:cxfEndpoint>` 定義は不要。Spring DSLではURI直書きが可能です。
- 必要に応じて、JAX-WSアノテーションを付与した `HelloService` インターフェースを `serviceClass` に指定します。
- `camel-cxf-starter` 依存を `pom.xml` に必ず追加する必要があります。

XML

```
<dependency>

  <groupId>org.apache.camel.springboot</groupId>

  <artifactId>camel-cxf-starter</artifactId>

</dependency>
```

✅ Spring Boot + Camel 4 でも `cxf:` は引き続き利用可能で、Blueprintからの移行が容易です。

## ● 8.3 ルーティングとException処理の構造再現演習

Camel ルートの移行では、ルーティングだけでなく `onException` による例外処理構造も重要です。

### Blueprint DSL の例(before)

```
XML
<onException>
  <exception>java.lang.Exception</exception>
  <handled>
    <constant>true</constant>
  </handled>
  <to uri="log:exception"/>
</onException>

<route id="exception-test">
  <from uri="direct:start"/>
  <to uri="bean:faultyProcessor"/>
  <to uri="log:success"/>
</route>
```

### Spring XML DSL の例(after)

```
XML
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <onException>
    <exception>java.lang.Exception</exception>
    <handled>
      <constant>true</constant>
    </handled>
    <to uri="log:exception"/>
  </onException>

  <route id="exception-test">
    <from uri="direct:start"/>
    <to uri="bean:faultyProcessor"/>
    <to uri="log:success"/>
  </route>
</camelContext>
```

ポイント

- `onException` の構文・機能は Blueprint とほぼ同一。
- Camel 4 では `DefaultErrorHandler` の構成やリトライ設定も併用可能。
- Bean の DI やメソッド参照は Spring 標準構文に従う(例: `<bean id="faultyProcessor" class="..." />`)

---

この章では、Blueprint DSL で作成されたルートを Spring XML DSL に移行する実践演習を通じて、構文変換・依存関係・ルート設計の再現方法を習得しました。最終章では、Camel for Spring Boot を本番運用する際の設計上のベストプラクティスをまとめます。

## 第9章：まとめと今後の運用設計

本章では、Camel 4.10 + Spring Boot + Spring XML DSLを導入することで得られるメリットを再確認するとともに、Karaf環境からの移行を段階的に進めるための戦略、そして移行後の継続的な改善・運用を実現するための設計ポイントについてまとめます。

### ● 9.1 Camel for Spring Bootの導入効果

Camel for Spring Bootを採用することで、以下のような効果が期待できます：

項目	導入効果
開発効率	Spring Bootの自動構成・DIとの統合により、複雑な設定不要
テスト容易性	camel-test-spring-junit5によるルート単位の自動テストが容易 バグの削減とリリースサイクルの高速化」
ビルド・デプロイ	Fat JARによりOSGi構成不要、単一JARで配布・実行可能
構成管理	<code>application.properties</code> による一元的なプロパティ管理
運用・監視	Actuator, Hawtio, JMX連携による可視性と柔軟な監視体制
将来性	Camel 4.x系・Spring Boot 3.x系での標準技術スタックに準拠

✓ Blueprint DSLやKaraf固有技術への依存を排除することで、技術的負債を整理し、保守性・再利用性を高めることができます。

## ● 9.2 段階的移行戦略(モジュール単位移行、ゼロダウンタイム等)

KarafからSpring Bootへの全面移行は一括ではなく、段階的に進めることでリスクを最小化できます。

推奨される段階的移行アプローチ

1. 現行**Blueprint DSL**の棚卸し
  - 各バンドルの責務と依存関係を明確化
2. ルート単位での**Spring XML DSL**移行
  - 検証済みルートから順次Spring Boot化し、ユニットテストを追加
3. 段階的切り替え
  - ルート単位でRESTエンドポイントやQueueなどのI/F経由で段階的に新旧を併用
4. ゼロダウンタイム戦略
  - Blue/Green デプロイまたは Canary Release を活用し、切り替えを段階化
5. 運用チームへのドキュメント展開
  - KarafとSpring Bootの違いを整理した運用マニュアルの整備

✓ モジュール単位での移行を行い、逐次テストとリリースを繰り返すことで、システム全体への影響を抑えたスムーズな移行が可能です。

## ● 9.3 継続的改善に向けた運用設計のポイント

移行後も持続的な運用改善を実現するには、以下の運用設計ポイントを押さえることが重要です。

### 1. 監視と可視化の徹底

- Actuator, JMX, HawtIO による稼働状態の常時監視
- Camelのルート単位の稼働・失敗統計の収集
- ログ出力形式の標準化(JSON形式や構造化ログの導入)

### 2. 設定の一元管理

- `application.properties` による環境別設定切り替え



- 機密情報はSpringの外部プロファイルやSecretsVault等で安全に管理

### 3. CI/CDパイプラインとの統合

- Mavenビルド＋ユニットテスト自動化
- 静的解析 (SpotBugs, Checkstyle) やセキュリティ検査 (OWASP)
- JenkinsやGitHub Actionsとの統合でリリース効率化

### 4. Camelアップデートと互換性対応

- 定期的にCamelのマイナーバージョンを追従
- 移行後もコンポーネントの非推奨確認とドキュメント管理を継続

✔ 「移行して終わり」ではなく、移行後に安定した運用と将来の拡張性を担保する設計が、成功の鍵となります。

## 最終まとめ

本トレーニングでは、Apache Camel 4.10 と Spring Boot 3.x を用いたルート開発とKarafからの移行について、構成・実装・運用の各側面から実践的に学びました。Blueprint DSLからの脱却、モダンなビルド／デプロイの習得、そして運用改善を実現することで、今後の開発・保守における柔軟性と生産性を飛躍的に向上させることができます。

今後は、以下のステップでさらなる改善・活用を目指しましょう：

- 自動テスト・監視体制の整備による運用の高度化
- Apache CamelのEIPやカスタムプロセッサの活用範囲拡大
- チーム内でのSpring Boot＋Camelの開発標準整備とドキュメント化