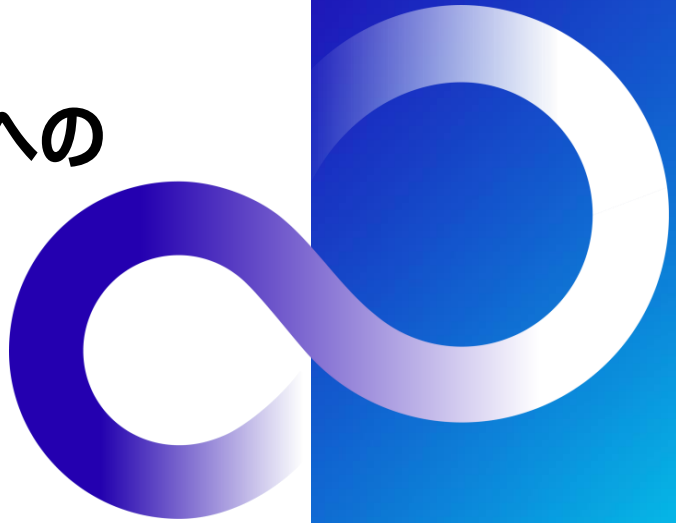
Spring BootからJakarta EEへの移行ガイド



富士通株式会社

2026年4月

目次

背景

- Spring BootはVMWare社が運営していたオープンソースコミュニティが公開し、機能拡張を続けてきた優れたJavaのフレームワークです。
- しかし、Broadcom社によりVMWare社の買収が2023年に完了したことで、Spring Bootコミュニティの運営やフレームワークの開発が、今後どのような方針で進められていくのか注目が集まっています。
- 本資料はSpring Boot利用者に向けて、Spring Bootの移行先にJakarta EEを選定する場合の移行方法を紹介する資料です。
- Jakarta EEは非営利団体であるEclipse Foundationが公開しており、富士通、IBM、Oracleなどの各製品ベンダーが参加して仕様策定などの技術発展に貢献する特定企業に依存しない標準技術です。
- 富士通も、INTERSTAGEの出荷開始から20年以上に渡り、Javaの実行環境製品を提供しています。現在は以下の製品を提供し、Jakarta EE仕様を実現するGlassFish機能を提供しています。
  - Interstage Application Server
  - Interstage Business Application Server
  - Enterprise Application Platform
- 本資料では富士通がJavaの実行環境製品を長年提供して培ったJakarta EEの技術と、Spring Bootの技術を比較して移行のポイントを解説します。

対象

FUJITSU

- 対象読者は以下です。
  - Java SEの基礎を理解している
  - Spring Bootで運用可能なアプリケーション開発経験者
  - Spring Bootを利用してシステム構築と運用をしたことがある技術者

- 対象バージョンは以下とします。

	Spring Boot【移行元】	Jakarta EE【移行先】
バージョン	Spring Boot 3.5	Jakarta EE 10

※本資料で解説する基本的な考え方（Uber JAR と配備モデルの違い、仕様と実装の関係、製品依存点の整理など）は、Spring Boot 4.0 においても有効です。

※移行元をSpring Boot 4.0とした場合には、移行先はJakarta EE 11に対応している必要があります。

1. はじめに

参考URL

【Spring Boot】
https://spring.pleiades.io/projects

【Jakarta EE】
https://jakarta.ee/

Spring Bootとは

FUJITSU

- Spring Bootとはエンタープライズ向けの機能を備えるJavaフレームワークです。
- Rod Johnson氏が書籍でSpring Frameworkの原型を2003年に発表し、Spring Bootの基盤となるSpring Frameworkの初版(1.0)を2004年に公開しました。
  - 「Spring と Spring Framework の歴史」より

> *Spring* は、初期の *J2EE [Wikipedia]* 仕様の複雑さに対応するために *2003* 年に誕生しました。*Java EE* とその現代の後継である *Jakarta EE* は *Spring* と競合していると考える人もいますが、実際にはそれらは補完的です。

- 元々はJ2EE(現在のJakarta EE)の煩わしさを解消する軽量なフレームワークとして公開されましたが、機能拡張を続けたことでSpring Frameworkも環境構築が複雑なフレームワークとなっていました。
- このため、環境設定が簡単な新たなフレームワークとして2014年にSpring Boot 1.0が公開されました。
  - 「Spring Boot 1.0 GA Released」より

> *Why containerless? Today's PaaS environments provide much of the management, scale out, and reliability features already, so we focus on making spring boot an **ultralight container**, great for application or service deployment in the cloud.*

- 現在ではSpringを用いたJavaアプリケーションを新規に開発する場合、Spring Bootを利用するのが一般的です。

Jakarta EEとは

FUJITSU

- エンタープライズ目的で標準的なJava機能を集約し、1999年にSun Microsystems社が初版のJava™ 2 Platform Enterprise Edition Specification, v1.2(J2EE 1.2)を公開しました。
  版数を重ねるごとに名称が J2EE → Java EE → Jakarta EE と変化しています。
  - 「Java™ 2 Platform Enterprise Edition Specification, v1.2」より

> *The Java™ 2 Platform, Enterprise Edition (J2EE) reduces the cost and complexity of developing these multi-tier services, resulting in services that can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures.*

- J2EEは規約と呼ばれるドキュメントにアプリケーションインタフェース仕様や、規約に対応した製品やOSSを提供する製品ベンダ(規約上の「Product vendor」)の役割が説明されています。
- J2EE1.2公開当時は、難解な仕様(特にEJB)が含まれており、その反発で各種フレームワークの登場を誘発しました。フレームワーク提供者はJ2EEのことを”heavy container”と呼び、フレームワークを”light container”(軽量コンテナ)と呼び、この頃に登場した代表的なフレームワークがSpring Frameworkです。
- このため、2006年「Java EE 5」で、ease of developmentのコンセプトで、SpringのDI(Dependency Injection)などの考え方を取り入れる方針に大転換しました。
- 現在全ての規約を含む Jakarta EE Platform の他に、Web アプリケーションに必要な機能セットの Web Profile、小規模なアプリケーションに必要な機能セットの Core Profile が用意されています。

移行について

FUJITSU

- 本資料における”移行”とは
  - Spring Bootアプリケーション（JAR）をJakarta EE標準API中心のアプリケーション（WAR）に作り変えてJakarta EE製品上で運用することを想定します。

- 過去の経緯からJakarta EEは「開発が難しい」「heavy container」という悪い印象を持っている方が多いかもしれませんが、前述したJava EE 5での方針転換以降、開発は容易となり、軽量化が図られています。
  - 【参考】「Java™ Platform, Enterprise Edition (Java EE) Specification, v5」より

> *The focus of Java EE 5 is **ease of development**. To simplify the development process for programmers just starting with Java EE, or developing small to medium applications, we've made extensive use of Java language annotations that were introduced by J2SE 5.0.*

移行のメリット

- Spring BootからJakarta EEに移行するメリットは以下です。
  - Jakarta EE製品提供元のサポートを受けられる
    - 製品利用時にトラブルが発生した場合、トラブル解消に向けたサポートを受けることができます。
    - 影響が大きい製品障害が検出された場合、製品提供元から修正パッチが提供されます。
  - 特定企業の影響を受け難い
    - Jakarta EEは非営利団体のEclipse Foundationが管理し、各種企業が活動に参加して仕様策定などを進めているため、例えば特定企業が買収されても活動が停止することがありません。
    - 特定企業の判断で機能拡張されず、各種企業システムでも有効と判断される機能のみ慎重に機能拡張が行われますが、一度公開した機能は比較的長く提供される傾向があります。

2. Spring BootとJakarta EEの相違点

Spring BootとJakarta EEの相違点

FUJITSU

■主な相違点は以下です。

	Spring Boot	Jakarta EE
コミュニティ運営	Broadcom社が運営	非営利団体の Eclipse Foundationが運営
仕様と実装	仕様と実装が1:1	仕様と実装が1:多
機能の組合せ	小さく始めて、必要なものだけを使用する スモールスタート	必要なものがすべて揃っている オールインワン
ビルド生成物	Uber JAR(Fat JAR)	Thin JAR
メインアプリケーションクラス	必要	不要
起動・停止操作	Java SEアプリと同様	各製品が提供するGUIやコマンドを利用

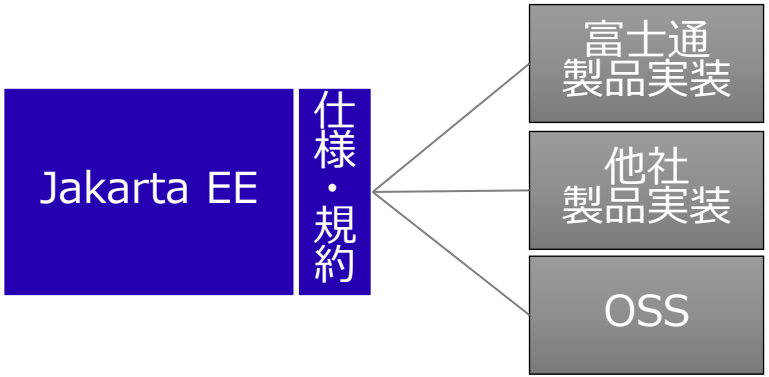
コミュニティ運営

- Spring Boot
  - Springを公開するSpringSource社を2009年にVMWare社が買収し、2023年にVMWare社をBroadcom社が買収したため、2025年9月現在はBroadcom社がSpring Bootを公開しています。
- Jakarta EE
  - Oracle社がJava EE 8を公開後に、「Java EE」の管理を非営利団体のEclipse Foundationに移管したことで、移管後の名前が「Jakarta EE」となりました。
  - 非営利団体のEclipse Foundationは、ベンダーニュートラル（特定の企業に依存しない中立な）なガバナンスモデルにより運営されており、富士通やIBMなどが活動に貢献しています。

仕様と実装

FUJITSU

- Spring Boot
  - 仕様も実装も全てBroadcom社が提供するため、仕様と実装は1:1です。
- Jakarta EE
  - 規約という仕様が書かれたドキュメントにJakarta EE規約に対応する製品やOSS(以降、Jakarta EE製品と呼ぶ)を提供する製品ベンダが担う役割とAPI仕様が記載されています。
  - Jakarta EE製品は規約に合わせて実装されるため、仕様(規約)と実装は1:多の関係です。
  - API仕様は規約で決められているため、開発したアプリケーションはどのJakarta EE製品上でも運用可能ですが、アプリケーションの運用方法やチューニング方法は製品により異なります。



機能の組合せ

FUJITSU

- Spring Boot
  - 使用する機能のみを選んで組合わせることができるため、使用する機能が少ない段階であれば組み合わせる機能を厳選すれば小さいリソースから始められます。
- Jakarta EE
  - Jakarta EE規約で示される一通りの機能を持つJakarta EE製品上でアプリケーションを運用するため、使用する機能が少ないアプリケーションでもある程度のリソースは必要ですが、利用する機能の構成を意識する必要がありません。

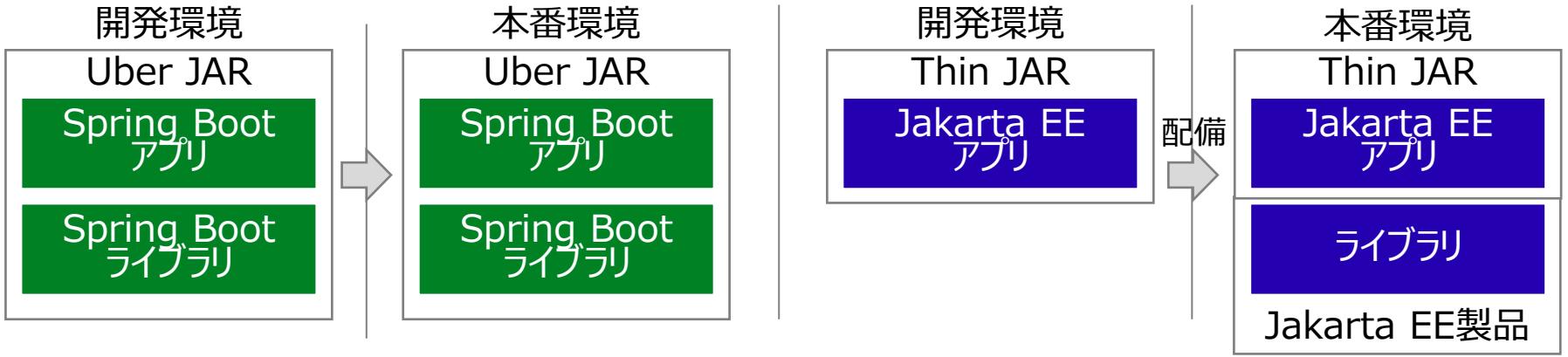
スモールスタート

オールインワン

ビルド生成物

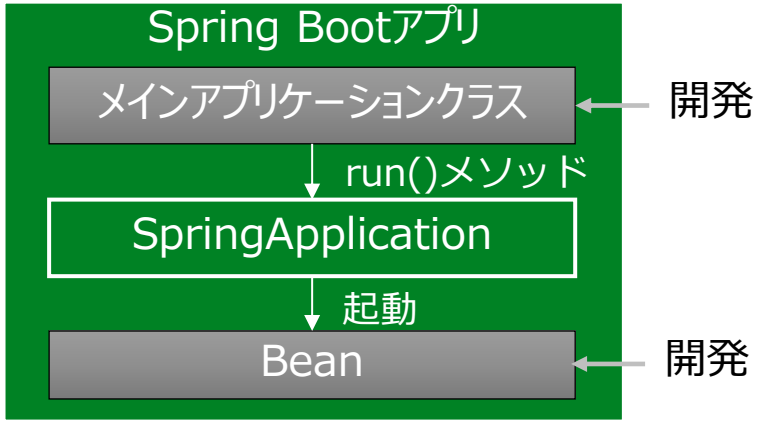
FUJITSU

- Spring Boot
  - ビルドした生成物は、アプリケーションだけでなくSpring Bootの実行に必要なライブラリ(クラスやjarファイル)もすべて含む一つの自己完結型の実行可能jarファイル(もしくはwarファイル)の”Uber JAR”(別名ではFat JAR)にパッケージングします。
- Jakarta EE
  - ビルドした生成物は、アプリケーションだけを含み、実行時のライブラリは含まない”Thin JAR”にパッケージングし、アプリケーション種別ごとに決められた拡張子(.warや.earなど)のアーカイブファイルとしてまとめた「配備モジュール」を作成します。



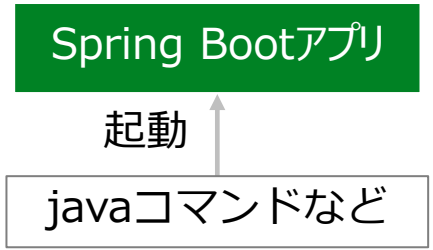
メインアプリケーションクラス

- Spring Boot
  - Java SEと同様に起動時に最初に呼び出されるメインアプリケーションクラスを用意し、Spring Bootをブートストラップするspringapplicationクラスのrun()メソッドを呼び出します。(Eclipse STSなどのIDEツールでアプリを開発している場合、自動生成されるため、意識されていないかもしれません。)
- Jakarta EE
  - メインアプリケーションを用意する必要はありません。Jakarta EE製品の実装をブートストラップするメインアプリケーションクラスはJakarta EE製品内で自動的に実行されます。



起動・停止操作

- Spring Boot
  - Java SEと同様に、javaコマンド(もしくは、Mavenなど)などでアプリを起動します。
- Jakarta EE
  - Jakarta EE製品が提供するGUIやコマンドで、アプリの起動・停止を要求します。
  - Jakarta EE製品は、単にアプリを起動するだけではなく、Java VMを多重で起動したり、異常終了したJava VMを再起動するなど、信頼性を向上する機能を提供しているのが一般的です。



3. 移行方法

移行作業について

FUJITSU

- 本章では、Spring BootからJakarta EEへの移行において、基本的な作業内容を紹介します。
  - アプリケーション実装の修正（Spring Boot依存からJakarta EE標準APIへ）
  - ビルド設定の変更とビルド生成物（war）
  - 実行環境の準備および起動手順の変更

- 本章で扱わない機能差分や個別機能に関する対応は、Spring Bootのリファレンスをもとに整理した【付録】機能の対比を参考に、必要に応じて実施してください。

アプリケーション実装の修正

FUJITSU

- アプリケーションの実装は、Spring Bootに依存した実装をJakarta EE標準APIに変更する必要があります。

- WebアプリケーションのGET処理の書換例

helloController.java

```
import org.springframework.stereotype.Controller;
…

@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }

  …
}
```

移行

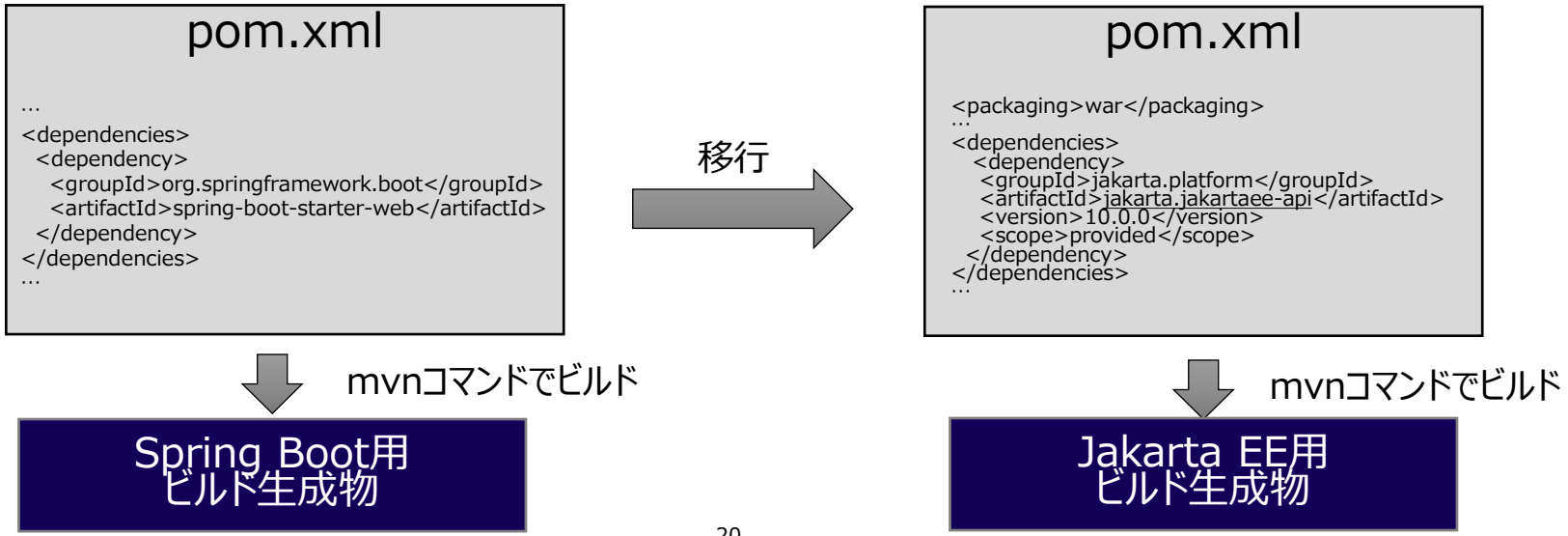
helloServlet.java

```
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
…

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws IOException  {
        response.getOutputStream().print("Hello World");
  }

  …
}
```

ビルド設定の変更とビルド生成物

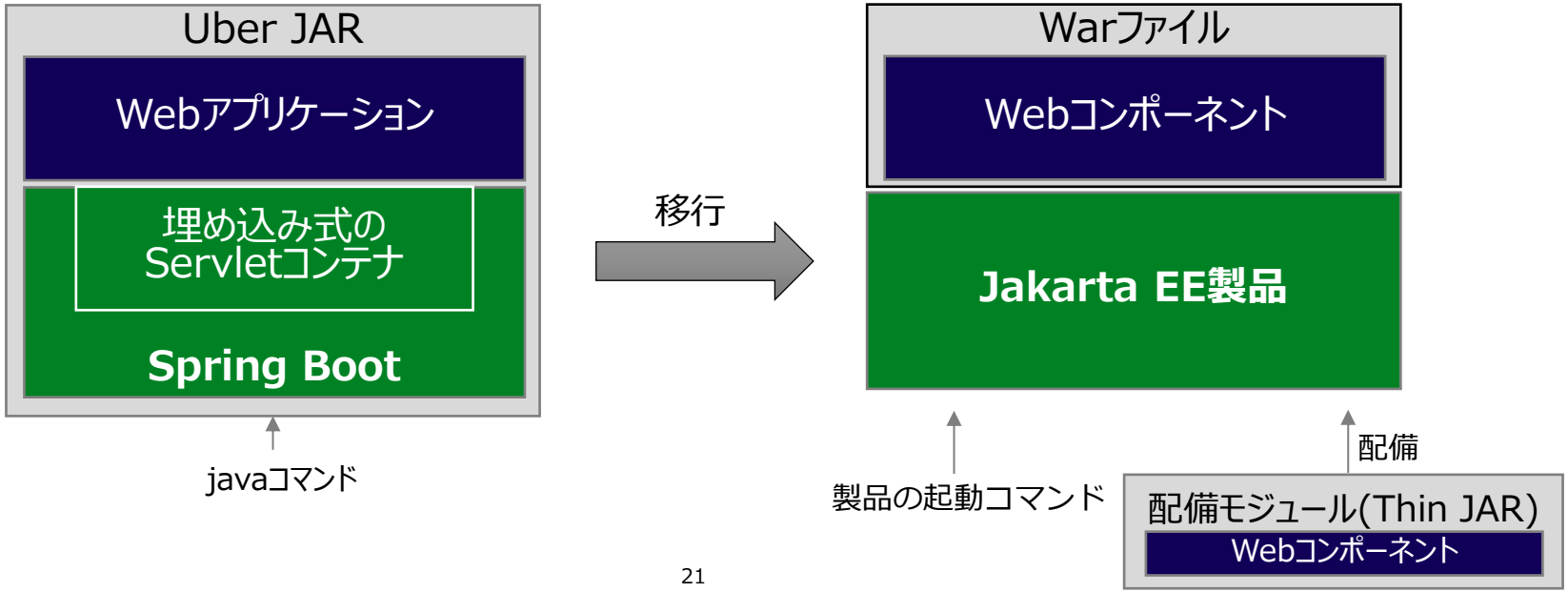
- Jakarta EE製品に配備するビルド生成物はビルド設定を書換た上で生成する必要があります。
  - Spring用の依存関係記述子を、Jakarta EE用の依存関係記述子に変更する
  - パッケージ形式はwarを指定する

- ビルドシステムとしてMavenを利用する時の書換イメージ



実行環境の準備および起動手順の変更

- Spring BootのシステムをJakarta EEに移行する場合、アプリケーションコンポーネントを運用するJakarta EE製品をOS上にインストールし、ビルド生成物を配備できる状態にします。
- 移行したビルド生成物をJakarta EE製品に配備後、起動操作でアプリケーションを起動します。



【付録】機能の対比

資料の構成について

FUJITSU

- 以降の説明ではSpring Bootのリファレンスガイドで言及されている機能説明に対して、Jakarta EEの同等機能の説明の有無と同等機能がある場合の説明の参照先の対比表を示し、移行方法を示します。
  - https://spring.pleiades.io/spring-boot/3.5/reference/index.html
- 移行先のJakarta EEは、全サブ規約を含むJakarta EE Platformを対象とします。
  - https://jakarta.ee/specifications/platform/10/jakarta-platform-spec-10.0.html
- Jakarta EE規約に同等の説明があれば、規約の記載箇所を記載します。記載が無いものは、以下の不要/製品依存/無し/同一仕様である旨を記載します。

記載	意味
不要	機能の特徴上で提供不要の機能です
製品依存	Jakarta EE製品が機能を提供します
無し	同等機能はありません
同一仕様	Java基本機能の説明のため同一の機能を提供します

- 製品依存の仕様は、富士通製品で採用するオープンソースのGlassFishのドキュメント記載箇所を参考に示します。

FUJITSU

1. ビルドシステム

対比表

■Spring Bootリファレンスガイドの「ビルドシステム」に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
Spring Boot での開発	ビルドシステム	依存関係管理	jakarta.platform:jakarta.jakartaee-api:jar:10.0.0	
		Maven	Maven ※GradleがMaven Centralをリポジトリとして参照するため、Gradleに関する説明はありませんがビルドは可能です。	
		Gradle		
		Ant		
		スターター	不要	

Spring Bootの場合

FUJITSU

- 各機能のライブラリを組み合わせて利用するため、各種機能に必要なSpringライブラリとサードパーティライブラリとして以下のページで依存関係のリストを提供しています。
  - https://spring.pleiades.io/spring-boot/3.5/appendix/dependency-versions/coordinates.html
- アプリケーションをGradleやMavenのビルドシステムでビルドする場合、アプリケーションで使用する機能の依存関係をpom.xml (Maven)やbuild.gradle (Gradle)に定義する必要があります。
- 使用する機能の依存関係をすべて定義するのは管理が大変のため、特定機能を利用する時によく利用される依存関係記述子のセットであるスターターが用意されています。
  - 以下はWebアプリケーションをビルドする時にspring-boot-starter-webのスターターを指定した例です。

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

- 機能を利用する上で必要なものはすべてビルド生成物に含まれるため、アプリケーションを運用する環境にビルド生成物を転送し、javaコマンドもしくはMavenやGradleを利用してアプリケーションを実行します。

Jakarta EEの場合

- Spring Bootのように機能を組み合わせて利用するのではなく、以下に記載されているサブ規約の機能をすべて利用できます。(製品やOSSで一部機能を制限としている場合があります。)
  - https://jakarta.ee/specifications/platform/10/jakarta-platform-spec-10.0.html#a3252
- アプリケーションをMavenでビルドする際には、全機能のAPIが含まれる「jakarta.platform:jakarta.jakartaee-api:jar:10.0.0」を指定してビルドします。
- 機能の依存関係を意識する必要がないため、Spring Bootのスターターのような仕組みはありません。
  - 以下はpom.xml(Maven)に「jakarta.platform:jakarta.jakartaee-api:jar:10.0.0」を指定した例です。

```
<dependencies>
        <dependency>
                <groupId>jakarta.platform</groupId>
                <artifactId>jakarta.jakartaee-api</artifactId>
                <version>10.0.0</version>
                <scope>provided</scope>
        </dependency>
</dependencies>
```

- 機能を利用する上で必要なものはビルド生成物（配備モジュール）に含まれず、この配備モジュールをJakarta EE製品がセットアップされた環境に「配備（デプロイ）」という操作を実行して初めて動かすことができます。配備方法はJakarta EE製品で異なるため、各製品のマニュアルを参照する必要があります。

2. 基本構成

対比表

■Spring Bootリファレンスガイドのビルドシステム以外の「Spring Bootでの開発」に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
Spring Boot での開発	コードの構造化	「デフォルト」パッケージの使用	同一仕様(Java SEの仕様)	
		メインアプリケーションクラスの特定	不要（Jakarta EE製品がアプリケーションの起動と実行を管理）	
	構成クラス		アプリケーションアセンブル（モジュールの作成）	
	@SpringBootApplication アノテーションの使用		不要	
	アプリケーションを実行する		製品依存(製品の起動機能でアプリケーションを起動する)	Deployment Planning Guide
	開発者ツール		製品依存(製品の配備機能で動的に配備されたアプリケーションをリロードする機能を提供する)	Application Development Guide
	本番用にアプリケーションをパッケージ化する		製品依存(製品の起動・停止・監視などの管理機能を利用する)	Administration Guide

Spring Bootの場合

FUJITSU

- Spring BootはServletコンテナ上で動作するサーバーサイドのJavaアプリケーションを開発効率化するフレームワークとライブラリ群です。
- 利用するにはJava SE環境とServletコンテナを用意する必要がありますが、埋め込み式のServletコンテナを複数内包しており、デフォルトでは埋め込み式のTomcatが動作します。
- Java SEのアプリケーションのようにmainメソッドを定義したメインアプリケーションクラスを用意し、@Configuration、 @EnableAutoConfiguration、@ComponentScanの3つのアノテーションの機能を持つ@SpringBootApplicationのアノテーションを定義するのが一般的です。
- メインアプリケーションクラスのmainメソッドでは、org.springframework.boot.SpringApplicationのrunメソッドを、以下のようにrunメソッドの引数にメインアプリケーションクラスを指定して実行します。

```
package sample;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

【参考】Spring Bootでjar形式を利用する場合

FUJITSU

- 開発したアプリケーションは通常のJavaアプリケーションと同様にjarファイルとしてパッケージングします。
- jarファイルは以下の構成とします。BOOT-INFにネストされたクラスファイルやjarファイルはSpring Bootのエントリポイントとして使用される特別なブートストラップクラスからロードされます。

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-BOOT-INF
    +-classes
    |  +-mycompany
    |     +-project
    |        +-YourClasses.class
    +-lib
       +-dependency1.jar
       +-dependency2.jar
```

1) spring boot loader classesにはSpring Bootのエントリポイントとして使用される特別なブートストラップクラスを格納します。

2) BOOT-INF/classesには開発したアプリケーションを格納します。

3) BOOT-INF/libにはSpring Bootを運用する際に依存するjarファイルを格納します。

4) META-INF/MANIFEST.MFには以下のようにMain-ClassにはSpring Bootのエントリポイントとして使用される特別なブートストラップクラスを指定し、Start-Classには開発したアプリケーションのメインクラスを指定します。

Main-Class: org.springframework.boot.loader.launch.JarLauncher
Start-Class: mycompany.project.YourClasses

【参考】Spring Bootでwar形式を利用する場合

FUJITSU

■warファイルとしてパッケージングすることも可能です。warファイルは以下の構成とします。

```
example.war
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-WEB-INF
   +-classes
   |  +-com
   |     +-mycompany
   |        +-project
   |           +-YourClasses.class
   +-lib
   |  +-dependency1.jar
   |  +-dependency2.jar
   +-lib-provided
      +-servlet-api.jar
      +-dependency3.jar
```

1) spring boot loader classesにはSpring Bootのエントリポイントとして使用される特別なブートストラップクラスを格納します。

2) WEB-INF/classesには開発したアプリケーションを格納します。

3) WEB-INF/libにはSpring Bootを運用する際に依存するjarファイルを格納します。

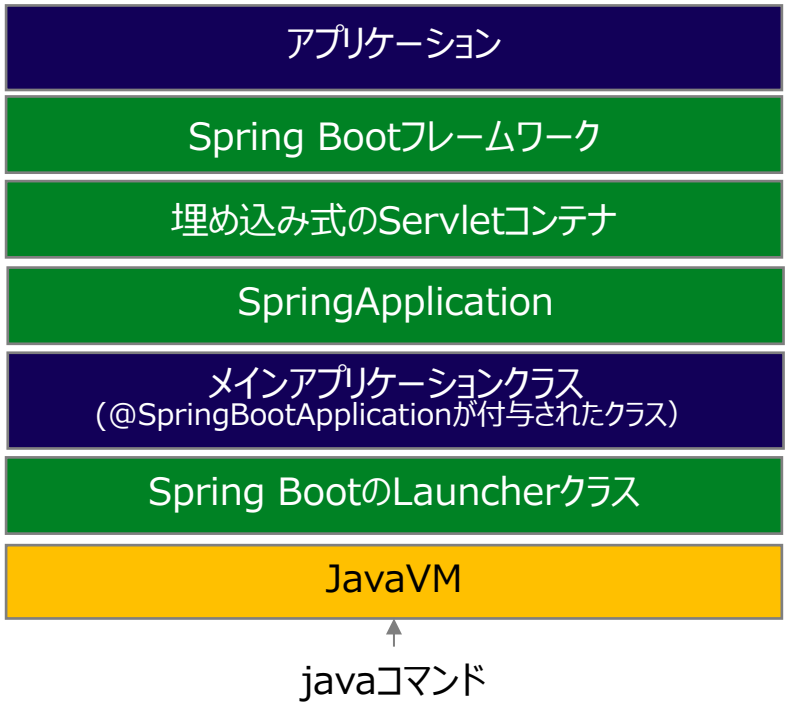
4) META-INF/MANIFEST.MFには以下のようにMain-ClassにはSpring Bootのエントリポイントとして使用される特別なブートストラップクラスを指定し、Start-Classには開発したアプリケーションのメインクラスを指定します。

Main-Class: org.springframework.boot.loader.launch.WarLauncher
Start-Class: mycompany.project.YourClasses

Spring Bootの場合

FUJITSU

■ 黄部分は事前に用意し、緑部分がSpring Bootが提供し、青部分はアプリケーション開発者が用意します。



Jakarta EEの場合

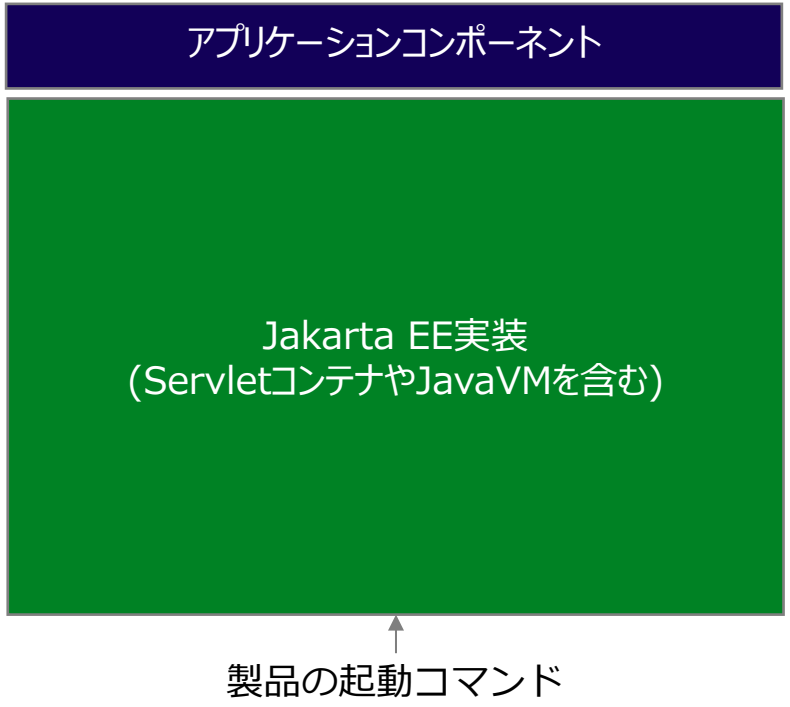
FUJITSU

- Jakarta EE製品は、Spring Bootと同様にサーバーサイドのJavaアプリケーションの開発を効率化するフレームワークとライブラリ群を提供します。
- Servletコンテナを含む各種アプリケーションを実行する実行環境を提供しており、Jakarta EEの実装と合わせてJava SEの環境も提供していることが一般的です。
- Java SEの場合と異なり、mainメソッドを定義したメインアプリケーションクラスを用意する必要はありません。
- アプリケーションのことをアプリケーションコンポーネントと呼び、アプリケーションコンポーネントをJARファイル形式の拡張子が異なる以下のファイルにパッケージしたものをモジュールと呼びます。 Jakarta EE製品が提供する起動機能(GUIやコマンド)により、配備されたアプリケーションコンポーネントを認識して起動します。

アプリケーションコンポーネントの種類	パッケージ形式の拡張子
Webコンポーネント	.war
リソースアダプタ	.rar
Jakarta Enterprise Beans コンポーネント	.jar(ejb-jarファイルと呼びます)
アプリケーションクライアント	.jar
コンポーネントを複数含めてパッケージングしたもの(Enterprise Archiveと呼ぶ)	.ear

Jakarta EEの場合

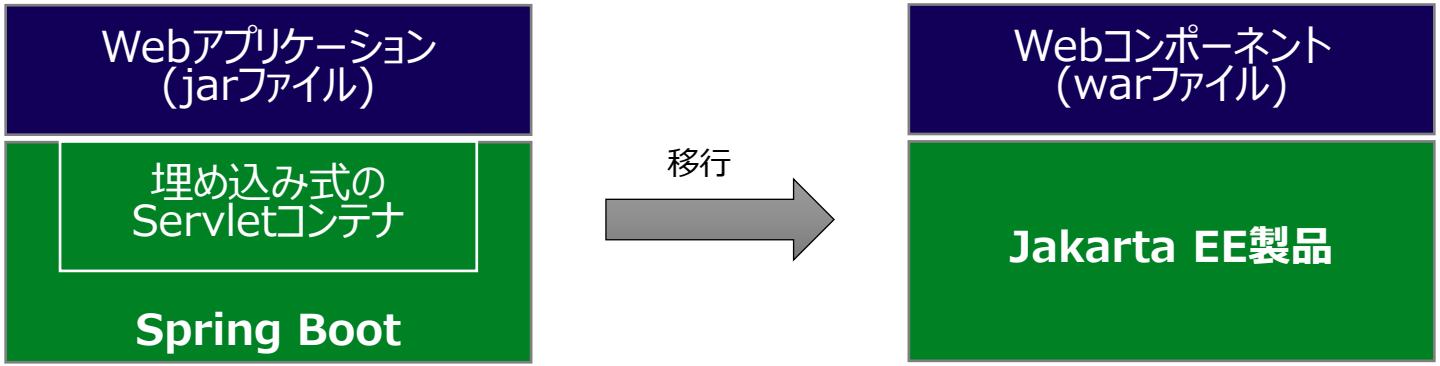
■緑部分はJakarta EE製品が提供し、青部分はアプリケーション開発者が用意します。



（参考）実行環境の移行について

FUJITSU

- 本資料の移行とは異なる、Spring Bootアプリケーションを Jakarta EE製品に配備可能なwarファイルとして作成し、実行環境を移行する方法がSpring Bootの以下のドキュメントには紹介されています。
  - 従来のデプロイ
    （https://spring.pleiades.io/spring-boot/3.5/how-to/deployment/traditional-deployment.html）



3. コア機能

対比表

FUJITSU

■Spring Bootリファレンスガイドの「コア機能」に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
コア機能	SpringApplication		製品依存	Administration Guide(9 Administering Life Cycle Modules)
	環境別の設定切り替え		環境別に有効となる設定は製品依存 環境別に有効となる設定を参照できるAPIは無し	Administration Guide(4 Administering the Virtual Machine for the Java Platform)
	プロファイル		Jakarta Contexts and Dependency Injection 4.0	
	ログ		製品依存(製品のログ機能を利用する。Java標準のロギングAPI（java.util.loggingなど）を利用可能。)	Administration Guide(7 Administering the Logging Service)
	国際化対応		製品依存	
	アスペクト指向プログラミング		Jakarta Interceptors 2.1	

対比表(つづき)

FUJITSU

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
コア機能	JSON		Jakarta JSON Processing 2.1	
			Jakarta JSON Binding 3.0	
	タスクの実行とスケジューリング		Jakarta Concurrency 3.0	
	開発時のサービス		製品依存	eclipse-ee4j/glassfish.docker
	独自の自動構成の作成		不要	
	Kotlin サポート		製品依存 ※Jakarta EEとしての公式サポートや専用機能は無し（Javaとして利用は可能）	無し
	SSL		製品依存	Security Guide(Administering JSSE Certificates)

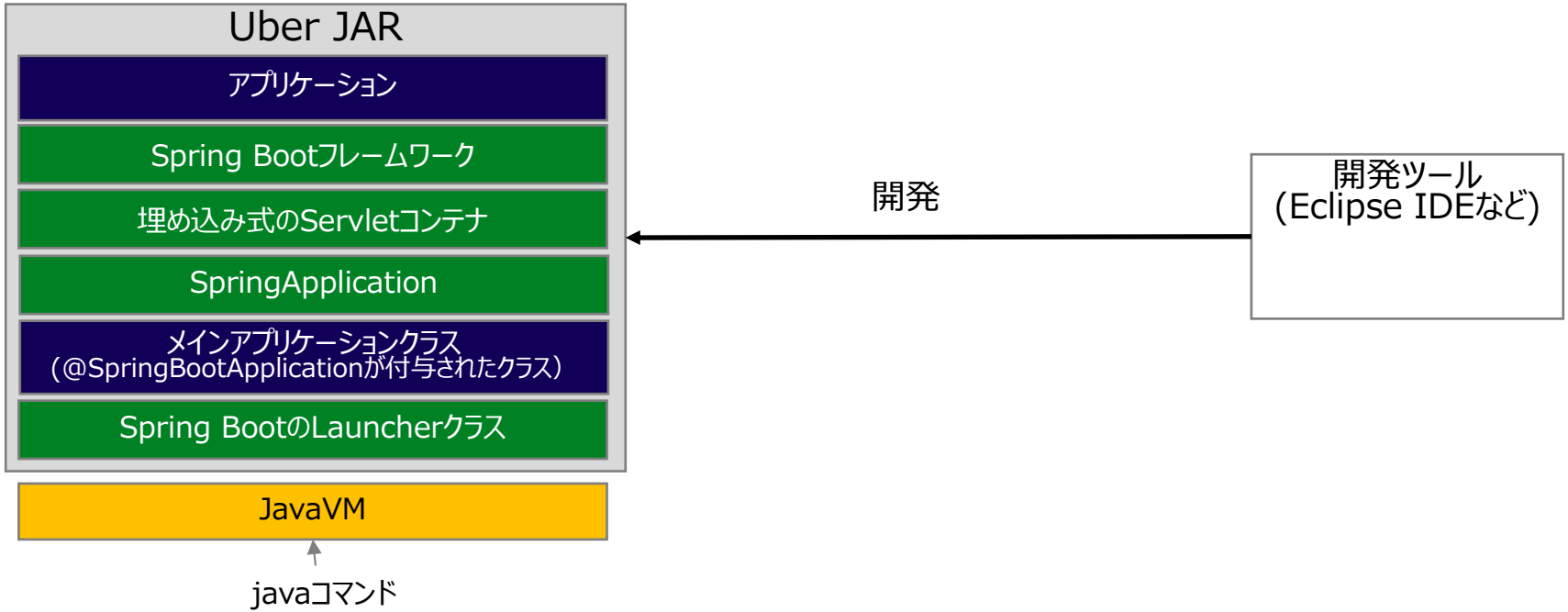
Spring Bootの場合

FUJITSU

- Spring Bootは、アプリケーションだけでなくSpring Bootの実行に必要なライブラリ(クラスやjarファイル)をすべて一つの自己完結型の実行可能jarファイル(もしくはwarファイル)にパッケージングして利用します。
- パッケージングした実行可能jarファイルを“Uber JAR” または “Fat JAR"と呼びます。
- 開発ツールなどで作成されたUber JARを通常のJavaアプリケーションと同様にjavaコマンド(もしくはMavenプラグインやGradleプラグイン)に指定して実行し、アプリケーションを起動します。
- 環境変数やシステムプロパティの設定などは通常のJava SEのアプリケーションと同様の方法で設定します。これらの環境別の設定をSpring Bootでは@Valueアノテーションを使用してBeanに注入するなど、アプリケーションで参照・更新できるSpring Boot独自の仕組みを提供しています。
- Spring Bootの動作をメインアプリケーションクラスのmainメソッドからSpring アプリケーションをブートストラップするSpringApplicationのrunメソッドを実行する際に、SpringApplicationに対してフレームワークの挙動を制御する各種設定が可能です。

Spring Bootの場合

■開発ツールでアプリケーションだけでなくSpring Bootの実行に必要なクラスやjarファイルなどをすべて含むUber JARを開発し、Uber JARを運用環境に転送してjavaコマンドでUber JARを実行します。



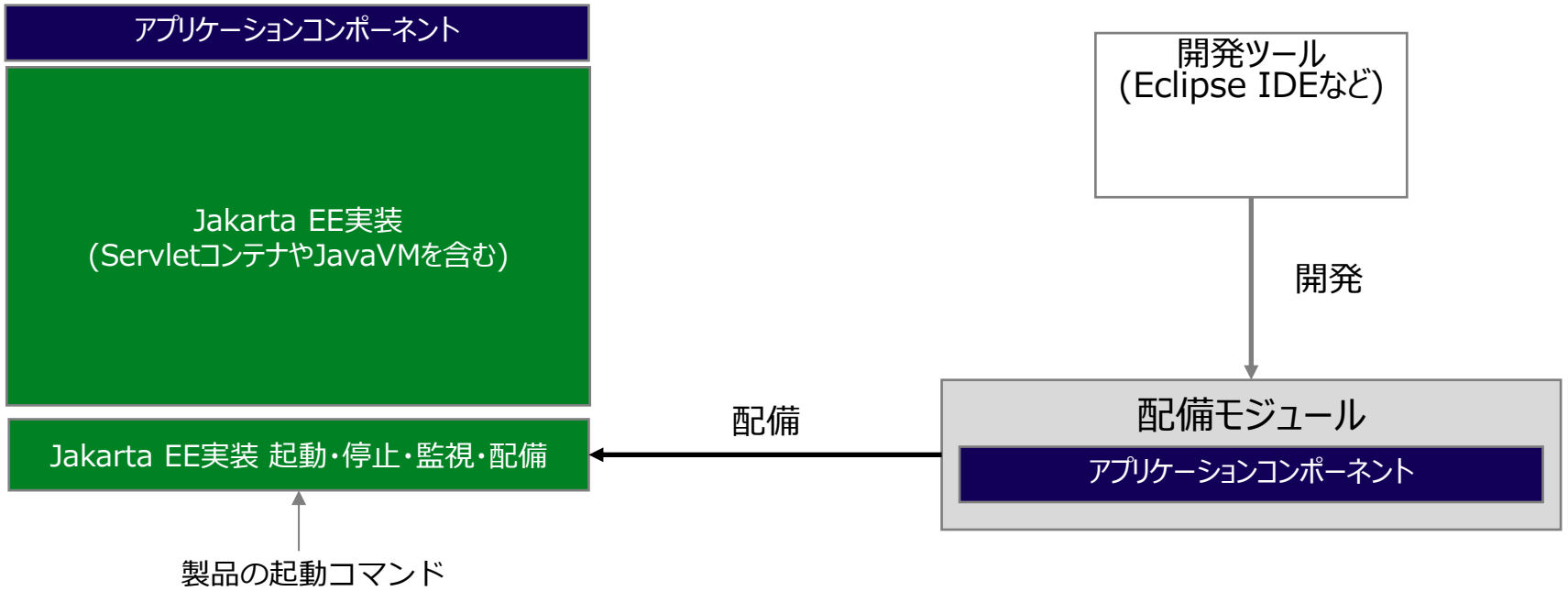
Jakarta EEの場合

FUJITSU

- Jakarta EEは、開発したアプリケーションコンポーネントをJARファイルフォーマットの、アプリケーション種別ごとに決められた拡張子(.warなど)のファイルにパッケージングします。
- パッケージングしたファイルを「配備モジュール」と呼びます。 “Uber JAR”や“Fat JAR”と異なり、配備モジュールは実行時のライブラリを含まない”Thin JAR”です。
- 開発ツールなどで開発した配備モジュールをJakarta EE製品に配備し、Jakarta EE製品が提供する起動・停止の操作を行う機能(GUIやCUI)でアプリケーションコンポーネントを起動します。
- 環境変数やシステムプロパティなどの環境別の設定は、Jakarta EE製品で設定方法が異なります。アプリケーション環境別の設定の自動読み込み、Beanへの自動注入、参照・更新できるAPIは標準で提供されていません。(Jakarta Configという規約で標準化の検討が進んでいます。)
- 起動・停止のライフサイクルに合わせて初期処理や終了処理を実行したい場合も、Jakarta EE製品の独自の方法(GlassFishのLife Cycle Moduleなど)が提供されていることが一般的です。

Jakarta EEの場合

■開発ツールでアプリケーションコンポーネントを含むモジュールを開発し、モジュールを運用環境に転送して使用するJakarta EE製品の配備操作でモジュールを配備し、起動操作によりアプリケーションを起動します。



4. Web

対比表

■Spring Bootリファレンスガイドの「Web」に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
Web	サーブレット Web アプリケーション		Jakarta Servlet 6.0	
			Jakarta Server Pages 3.1	
			Jakarta Server Faces 4.0	
			Jakarta Standard Tag Library 3.0	
			Jakarta Expression Language 5.0	
			Jakarta Managed Beans 2.0	
	リアクティブ Web アプリケーション		無し	
	グレースフルシャットダウン		製品依存(製品の停止機能として一般的に提供されます)	Reference Manual(--foceの説明)

対比表(つづき)

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
Web	Spring Security		Jakarta Servlet 6.0(13. Security)	
			Jakarta Authentication 3.0	
			Jakarta Authorization 2.1	
			Jakarta Security 3.0	
	Spring Session		製品依存(製品の可用性機能として一般的に提供されます)	High Availability Administration Guide
	Spring for GraphQL		無し	
	Spring HATEOAS		無し	

Spring Bootの場合

FUJITSU

- Webブラウザなどからhttp(s)要求を受けて、処理結果を返却するWebアプリケーションの場合、以下のように@Controllerアノテーションを付与したBeanを用意し、GETやPOSTなどのメソッドに合わせて@GetMappingや@PostMappingなどのアノテーションを付与したメソッドを用意します。

```
import org.springframework.stereotype.Controller;
…

@Controller
public class HelloController {

	@GetMapping("/hello")
	public String handle(Model model) {
		model.addAttribute("message", "Hello World!");
		return "index";
	}
}
```

- Session情報をRedisなどのデータストアに保持し、JavaVM異常終了時や縮退運用時でもSession情報を別JavaVMで利用可能とする機能が提供されています。
- 停止時に処理中の要求が完了するまで待機するグレースフルシャットダウン機能を提供しています。

Jakarta EEの場合

FUJITSU

- Webブラウザなどからhttp(s)要求を受けて、処理結果を返却するWebアプリケーションの場合、以下のようにHttpServletクラスを継承した@WebServletアノテーションを付与したクラスに、GETやPOSTなどのメソッドに合わせて用意されたdoGetやdoPostなどのメソッドをオーバーライドしたメソッドを用意します。

```
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;

…

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException  {
        response.getOutputStream().print("Hello World");
    }
}
```

- Session情報をデータストアに保持する機能はJakarta EE製品が提供しているのが一般的です。
- 停止時に処理中の要求が完了するまで待機するグレースフルシャットダウン機能も同様にJakarta EE製品が提供しているのが一般的です。

5. データ

対比表

■Spring Bootの「データ」の機能に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
データ	SQL データベース		Jakarta EE 10(2.6. Database)	
		DataSource を構成する	実装依存	Administration Guide(11 Administering Database Connectivity)
		JdbcTemplate の使用	無し	
		JdbcClient の使用	無し	
		JPA および Spring Data JPA	Jakarta Persistence 3.1	
		Spring Data JDBC	無し(Jakarta EE 11でJakarta Dataが提供されます)	
		H2 の Web コンソールを使用する	無し	
		jOOQ を使用する	無し	
		R2DBC を使用する	無し	
	NoSQL テクノロジーの使用		無し	

Spring Bootの場合

FUJITSU

- Spring Bootの場合、データベースにはJavaのjavax.sql.DataSourceを利用し、サポートするアルゴリズムで接続情報を使い回す接続プール(デフォルトはHikariCP)を利用します。
- アプリケーション開発時にアプリケーションが正しくデータベースにアクセスできるか確認するのに便利なインメモリ埋め込みデータベースとしてH2 、HSQL、Derbyを提供しています。インメモリ埋め込みデータベースは永続化などの信頼性に課題があるため、本番では外付けのデータベースの利用が推奨されています。
- データベースにアクセスする方法には、DataSourceのAPIを直接利用する以外に、JdbcTemplate、JdbcClient、JPA および Spring Data JPA、Spring Data JDBCなどの利用方法が存在します。
- 本資料ではJava標準のJDBC APIでデータベースにアクセスするDataSourceの利用を想定します。
- 以下の条件を満たす場合、 自動構成によって、デフォルトのDataSourceが作成されます。
  - application.properties または application.ymlに定義されたspring.datasource.*プロパティ
  - 接続プールを利用する関連ライブラリ (HikariCP 接続プールで JDBC を使用する場合、spring-boot-starter-jdbcで定義)を含む
  - 使用するJDBCドライバの関連ライブラリ(Oracleの場合、 com.oracle.database.jdbc)を含む

Spring Bootの場合

FUJITSU

- デフォルトのDataSourceを利用してデータベースにアクセスする場合の例です。
  - ソースコード

```
@Service
public class MyService {
    @Autowired private DataSource dataSource;
    Connection conn = null;
    public void doDatabaseOperation() {
        try {
            conn = dataSource.getConnection() ;
            // SQLの実行など
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

  - application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.datasource.username=root
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Jakarta EEの場合

FUJITSU

- Jakarta EEの場合、データベースにはJavaのjavax.sql.DataSourceを利用し、接続プールの実装はJakarta EE製品に任されています。
- アプリケーション開発時にアプリケーションが正しくデータベースにアクセスできるか確認するのに便利なインメモリ埋め込みデータベースをJakarta EE製品が独自に提供しているのが一般的でGlassFishではDerbyが提供されています。インメモリ埋め込みデータベースは永続化などの信頼性に課題があるため、本番では外付けのデータベースの利用が推奨されています。
- データベースにアクセスする方法には、DataSourceのAPIを直接利用する以外にJPAを利用する方法があります。
- 本資料ではJava標準のJDBC APIでデータベースにアクセスするDataSourceの利用を想定します。
- DataSourceを利用する方法は、Jakarta EE製品ごとに異なりますが、GlassFishの場合には以下の条件を満たすとDataSourceが利用できるようになります。
  - Jakarta EE製品が提供するコマンドやGUIを使ってJDBC接続プールとJDBCリソースを定義する
  - 使用するJDBCドライバの関連ライブラリ(Oracleの場合、 com.oracle.database.jdbc)をクラスパスに設定する

Jakarta EEの場合

FUJITSU

- DataSourceを利用してデータベースにアクセスする場合の例です。
  - ソースコード
    @Resourceのnameには、JDBCリソースの名前を指定します。

```
import java.sql.*;
…

@Resource(name = "jdbc/defaultCICSDataSource")
private DataSource dataSource;

public void doDatabaseOperation()
    try {
        Connection conn = null;
        conn = dataSource.getConnection();
        // SQLの実行など
    }catch (SQLException e){
        e.printStackTrace();
    }
}
```

FUJITSU

6. IOとメッセージング

対比表

■Spring Bootの「IO」「メッセージング」の機能に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
IO	キャッシング		無し	
	Hazelcast		無し	
	Quartz スケジューラー		無し	
	メール送信		Jakarta Mail 2.1	
			Jakarta Activation 2.1	
	検証		Jakarta Bean Validation 3.0	
	REST サービスの呼び出し		Jakarta RESTful Web Services 3.1	
	Web サービス		Jakarta RESTful Web Services 3.1	
	JTA による分散トランザクション		Jakarta Transactions 2.0	
メッセージング	JMS		Jakarta Messaging 3.1	
	AMQP		無し	
	Apache Kafka サポート		無し	
	Apache Pulsar サポート		無し	
	RSocket		無し	
	Spring Integration		Jakarta Connectors 2.1	
	WebSocket		Jakarta WebSocket 2.1	

説明

- メール送信など外部サービスと通信する各種IOのSpring Boot機能については、Jakarta EEの標準APIにも同等の機能が存在するかを確認し、同等機能を利用するようにアプリケーションを改修してください。

7. その他

対比表

■Spring Bootのその他機能に相当するJakarta EE規約の説明箇所を示します。

Spring Boot 3.5			Jakarta EE 10	
見出し1	見出し2	説明	規約の記載箇所、もしくは、不要/製品依存/無し/同一仕様	【参考】GlassFishの場合の、ドキュメント記載部分
テスト			製品依存	Application Development Guide(6. Debugging Applications)
Spring Boot アプリケーションのパッケージング			製品依存	Performance Tuning Guide
その他。リファレンスに記事が見当たらないが、Jakarta EE規約に記載があるサブ規約	Spring Batch		Jakarta Batch 2.1	
	無し		Jakarta Debugging Support for Other Languages 2.0	
	無し		Jakarta Enterprise Beans 4.0	

説明

- Spring BootにはテストフレームワークのJunitを利用するスターターなどが用意されていますが、Jakarta EEの規約にはテストに関する言及はありません。Junitなどフレームワークを利用する場合、必要なライブラリをクラスパスに設定するなどJavaアプリケーションと同等の設定を実施する必要があります。
- Spring Bootのマニュアルに記載されている、CDS、AOT、GraalVMやDockerなどのコンテナ対応についてもJakarta EE製品の対応状況を確認してください。

他社商標・免責事項

他社の商標について

FUJITSU

- Spring®、Spring Boot、Spring Framework は、Broadcom Inc.またはその子会社の商標または登録商標です。
- OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。
- Eclipse、Jakarta、GlassFishは、米国およびその他の国における Eclipse Foundation, Inc. の商標です。
- IBMは、世界の多くの国で登録された International Business Machines Corporation の商標です。
- Apache Maven、Mavenは、米国およびその他の国における Apache Software Foundation の商標です。
- その他記載されている製品名などの固有名詞は、各社の商標または登録商標です。

免責事項

FUJITSU

- 本資料は、公開されている情報をもとにSpring BootからJakarta EEへの移行に関する一般的な技術情報を提供することを目的としています。
- 記載されている製品仕様、機能、提供状況等は、予告なく変更される場合があります。
- 本資料の内容は特定の製品構成や利用環境での動作や適合性を保証するものではありません。
- 実際の設計・導入にあたっては、各製品の最新の公式ドキュメントをご確認ください。
- 本資料の利用にあたっては、本資料に記載の免責事項に加え、本資料を掲載するWebサイトに記載されている免責事項にも従うものとします。

Thank you