Springとは

springの概要

㈱トラストサービス　2006/04/21

WEBの歴史（軽量コンテナの登場まで）

- 当初、HTML(静的コンテンツ)

  ↓（静的コンテンツだけでは業務として利用するには機能不足）

- CGI（Common Gertway Interface）・・動的コンテンツ

- ↓（問題点：処理要求のたびにプログラムが起動する・セッション管理がない・ページ生成とビジネスロジックを分離するが困難）

- JSP/Servlet の登場(マルチスレッドで実行、WEBコンテナは開発者に意識させることなく

  ↓ セッション管理を行ってくれる。またJSPがページ生成を行いビジネスロジックをServletで行えるといったアーキテクチャ上の魅力)

- EJBの登場(分散処理と分散トランザクションの融合コンポーネント)

  ・プログラマが書かなければならなかった定型的な処理の多くの部分をEJBコンテナが担う

  →・リモート(ローカル）メソッドだけの記述

  ・トランザクショ管理・インスタンスプール・オブジェクトの永続化管理

  しかし、EJBはそもそも分散用コンポーネントのためリモートアクセス（電話回線などを通じて、ネットワークやコンピュータに外から接続すること ）しかない。ところがWEBアプリケーションでは分散処理などはめったに行われないため、ローカルアクセスが必要となる。またEJBコンテナに依存しているEJBはTESTがしにくい問題点が指摘され始めた。（J2EEは重量級であることから軽量コンテナが考え始められた）

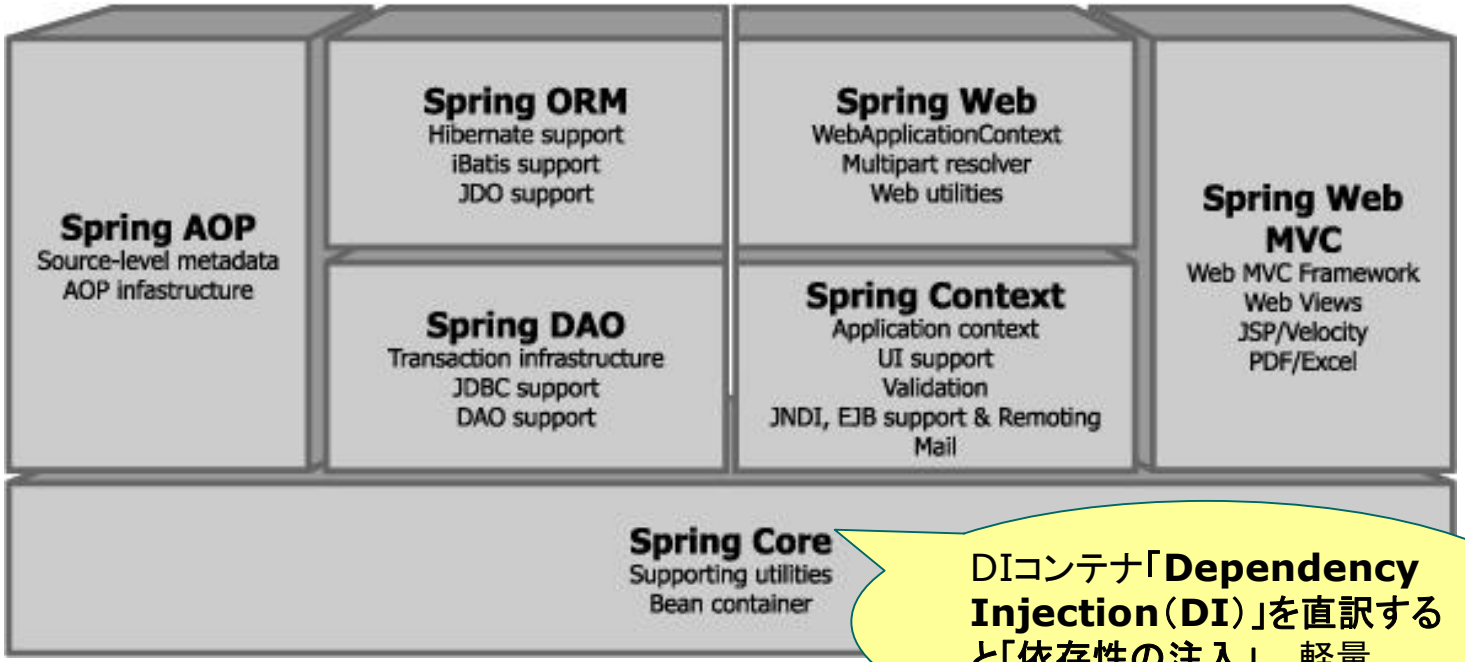
  ↓

  DIコンテナの登場となる(例：Spring・PicoContainer・Seaser・HiveMindなど)

Spring概要

J2EEアプリケーション開発支援するフレームワークが揃っている

- Spring Framework (単に Spring と呼ばれることが多いです) は Rod Johnson 氏の著書 "Expert One-on-One J2EE Design and Development" の中で使用されたコードを元にしたオープンソースの Java/J2EE アプリケーションフレームワークです。Spring は DI コンテナと呼ばれる Bean コンテナをコアモジュールとし、下図のように 7つのモジュールからなります。



DIコンテナ「**Dependency Injection（DI）**」を直訳すると「**依存性の注入**」。軽量(Lightweight)なコンテナ

モジュールの機能

唯一のインスタンスであることを保証できる。
インスタンスの生成数をコントロールできる。
機能拡張、汎用化ができる。(実装例参照)

- Core モジュール
  - Core モジュールは Spring の根幹をなすモジュールであり、オブジェクトの生成・登録、オブジェクト間の関連づけを行う Bean コンテナの機能を提供します。このモジュールにより、必要なオブジェクトの生成および初期化を一元化できるだけでなく、プログラムから Singleton の実装とオブジェクト間の依存関係の実装を排除することができます。
- Context モジュール
  - JNDI のような方法で JavaBeans にアクセスする方法を提供します。Context モジュールの1つである org.springframework.context は Core モジュールである org.springframework.beans パッケージからその機能を継承していますが、サーブレットコンテナによるリソースバンドルやイベント伝播等のテキストメッセージングのサポートが追加されています。
- DAO モジュール
  - JDBC 抽象化レイヤを提供します。このモジュールを使用することで開発者が Connection オブジェクトの取得・破棄、Statement オブジェクトの生成等を行う必要がなくなります。
- ORM モジュール
  - JDO, Hibernate, iBatis 等の object/relational mapping API を統合するレイヤを提供します。ORM モジュールを使用することで O/R マッパを Spring の他の機能と組み合わせて使用できるようになります。

モジュールの機能

「AOP（アスペクト指向プログラミング）」は
とは作り上げたオブジェクト群に対して「**機能を挿入する（織り込む）**」ことができるアーキテクチャである。つまり、作成したオブジェクト群に対して「アトヅケ」で機能や処理を挿入することができる仕組み。

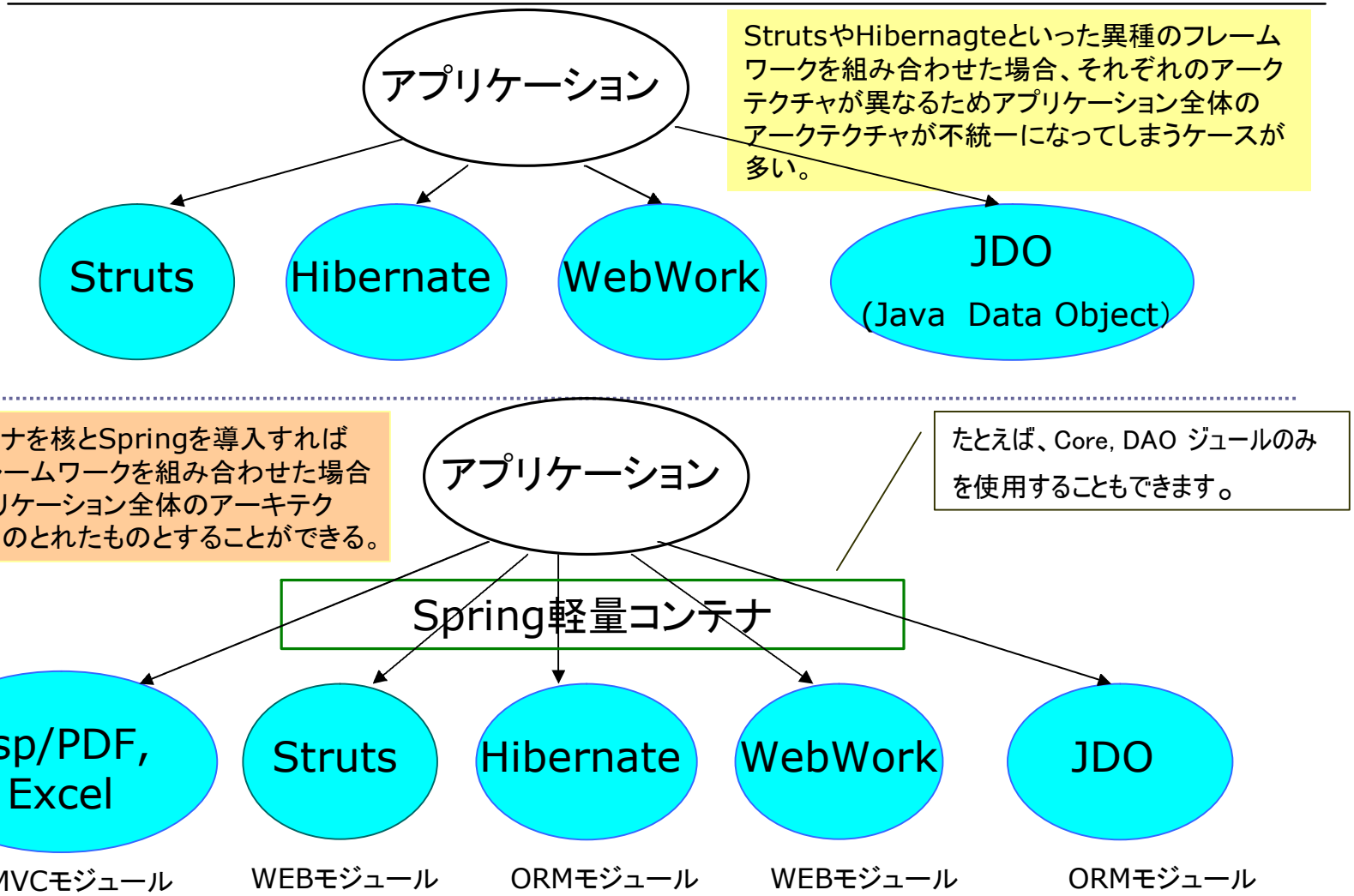
- AOP モジュール
  - AOP Alliance と互換性のあるアスペクト指向プログラミングのフレームワークを提供します。AOP モジュールを使用することで、たとえば session のチェック等のオブジェクトの責務外のロジックを明示的に記述することなく、宣言的に追加することができます。
- Web モジュール
  - 基本的な Web 指向の統合機能を提供します。Spring を WebWork や Struts と一緒に使用する場合は、このモジュールが統合を行います。
- Web MVC モジュール
  - Web アプリケーション用の Model-View-Controller フレームワークを提供します。Spring の MVC モジュールの特徴としてユーザが入力したデータを JavaBeans として取得できることや、入力されたデータの検証処理が MVC モジュールから切り離されていることが挙げられます。

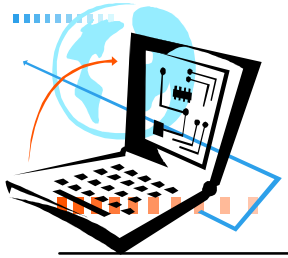
WebWorkは**Struts**と似たような役割を持つWeb**フレームワーク**で、今までのServet/**JSP**とはまったく違ったPull HMVCというコンセプトを持っています。
ServletAPIを一切使わないなど、今までのWeb**システム**の**常識**を越えたシンプルさと柔軟性があり、**アジャイル**にWeb**システム**を構築することができます。しかし2006/3/24 Apache Struts Action 2.0へ移行するようです。

AOP Alliance：アスペクト指向環境（AOE）における共通のアーキテクチャを策定する団体

アジャイル：「俊敏な」「すばやい」という意味の英単語で、IT業界では、経営環境の変化に迅速に対応できる柔軟な情報システムや、効率的なシステム開発手法などを指す。

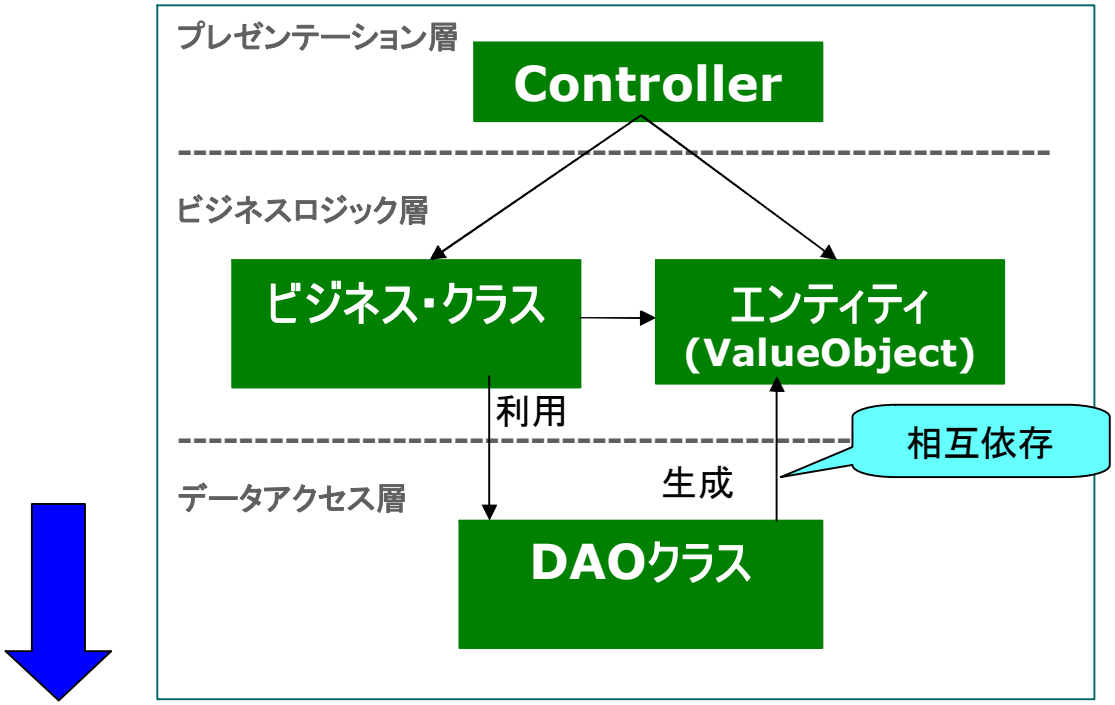
Springフレームワークのイメージ





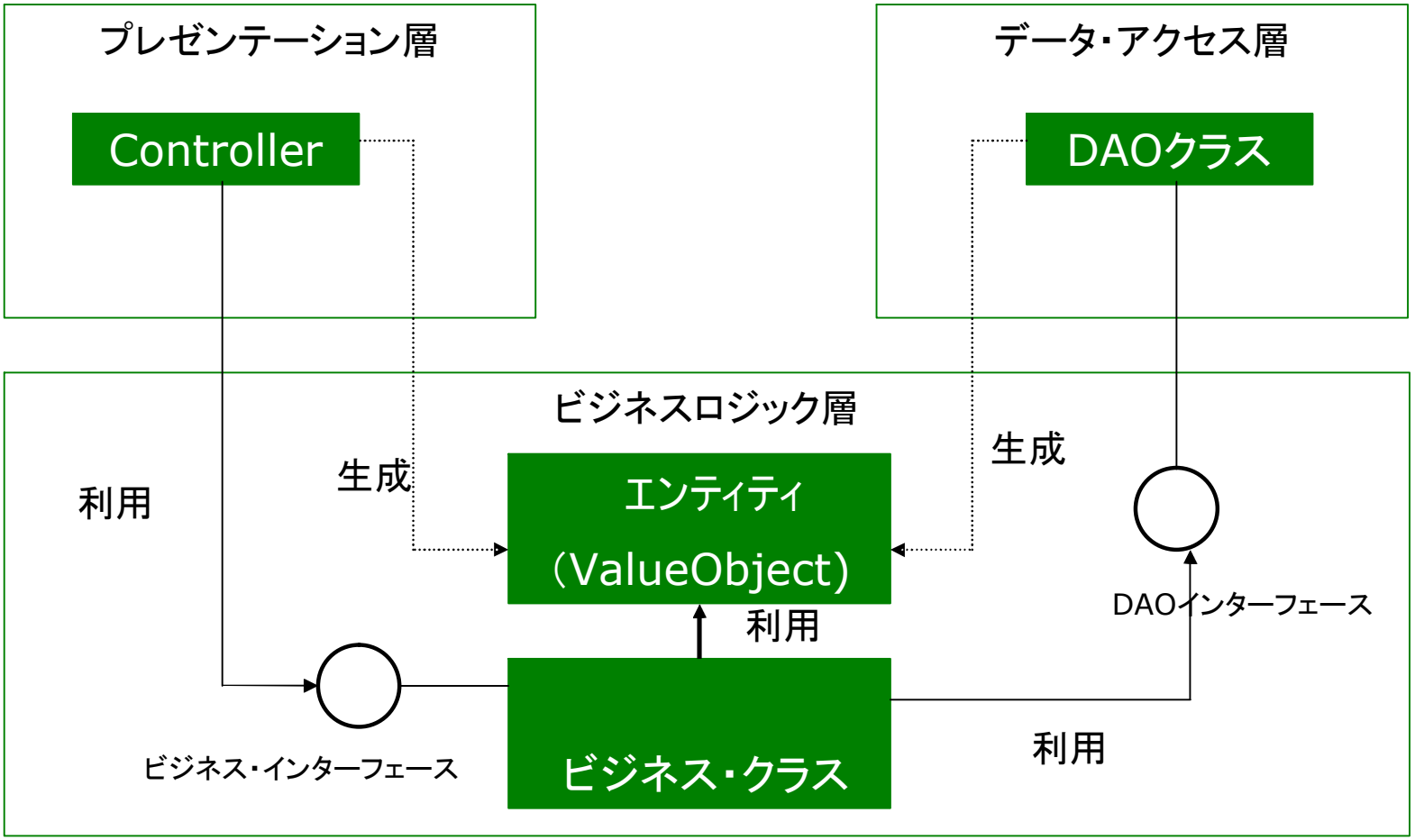
Spring導入

例えば、Springが提供するHibernateとの統合機能を利用して、ソースコード内からトランザクション制御に関するコードを排除し外部設定ファイルによってトランザクション管理を行う。→ ビジネスロジック層とデータアクセス層の依存関係がなくなり、安定かつ独立したビジネスロジック層ができる。



凹型レイヤ(インタフェースベース)

今までのMVCモデルに便利なフレームワークをいろいろ利用した際も下記のようにロジックを明確切り分けシステムを構築しコントロールするのがSpringである。



凹型レイヤー重要ポイント

- プレゼンテーション層を実現する技術を、StrutsからJSFやSwingに変更した場合や、DBアクセス層の部分をJDBC経由プログラムをO/Rマッピングフレームワークに利用するプログラムに変更した場合でもビジネスロジック部分に影響が及ばない点である。

■DIコンテナなし

アプリケーション君は手紙をだすために、はがきを用意しました。次に切手を用意して貼りました。次に住所を書いて本文を書いてちゃぶ台に起きました。

- 葉書　ハガキ ＝　new　葉書();//ハガキ用意
- 切手　きって　=new　切手();//切手を用意
- ハガキ.set切手(きって);//切手を貼る
- ハガキ.set宛先(“東京・・”);//宛先を書く

■DIコンテナあり

アプリケーション君が手紙を出そうとちゃぶ台を見ると、切手も張ってあって宛先の住所も自分の住所も、本分まで書いてあるはがきが用意してありました。(勝手にやってくれる？)

- Private 葉書　ハガキ;
- Public void set葉書(葉書　ハガキ);//これがあれば、宛先からなにから書いてある完璧なハガキをくれる！

Spring概要のまとめ

■DIコンテナを導入するには高いスキルが必要！

- DIコンテナはインターフェース経由で処理(ロジック)を行うことで設定に変更(仕様変更等)が起きた場合においてもコントローラ側はコード変更なしでOKであることを売りにしますが、重要なのは、インターフェースを定義するスキルです。
- 適切な範囲で処理を切り分け、インターフェース定義することは品質を保持しますが、そうでない場合かえって品質低下になりうるため、DIコンテナを導入する際はそういった役割等を把握し、理解した上で設計するスキルが必要です。

■DIコンテナを利用するには局所的に・・・・

- DIコンテナが便利とはいえ、どこでもDIコンテナ機能を利用するとこれも品質低下の恐れがあります。
- 特にAOPの利用には最新の注意が必要です。AOPは一見便利そうですが、一番の障害はJAVAコードからAOPの存在に気づきにくい点です。

Spring概要のまとめ

- 従来の開発のように一連の処理の流れでJAVAのコードを記述するのではなく、一連の処理の中に割り込んで固有の処理を行います。ここに一定のローカルルールをあらかじめ設定しておかないとどこでAOPを利用しているかわからなくなり重要なバグを残してしまう原因となります。
- またAOPの利用は比較的多くのメモリを消費するため、アクセス集が増えるとOutOfMemory等のハード的な障害もあるためコンシューマ向けのサイトやアクセス数が多いシステムなどは利用を最小限にしたほうがよいようです。
- この手の障害はかなり致命的のため、ユーザはDIコンテナ利用に消極的であることも念頭におき、提案していくことが必要と思われます。
- しかし、DIコンテナを使いこなせれば開発効率とシステム全体の品質もかなり有利になることも上げられ今後DIコンテナの利用は増えていくものと思われます。