Java による Web アプリケーション 開発フレームワークについて

SI 事業部 東京センター 開発部
青　江　　崇

1. は じ め に

近年，インターネットの爆発的普及に伴い，Web アプリケーションの構築が盛んに行われるようになってきた。最近ではインターネットだけでなく，社内の基幹システムまでもが Web アプリケーションに置き換えられようとしている。Web アプリケーションの構築がこれ程までに盛んになってきた理由としては，クライアントとして Web ブラウザを使用するため，クライアントアプリケーションを配布する必要がない，システムが多層構造をとることにより，スケーラビリティを高めることができること，等が挙げられる。

この Web アプリケーション開発において，現在最もよく利用されている言語が Java である。Java には J2EE（Java2 Platform Enterprise Edition）と呼ばれる，サーバーサイドアプリケーション開発を行うための仕様群があり，他の言語による開発と比較して，サーバーサイドアプリケーションを構築するための強力な機能を提供してくれることが最も大きな理由であろう。

しかしながら J2EE を用いた場合においても，Web アプリケーションの構築は決して容易ではない。しかも Web アプリケーション開発案件は多くの場合，大変な短納期である。このため最近になって Web アプリケーションを開発するためのフレームワークが注目されるようになってきた。

当社 SI 事業部では，これまでの Java による Web アプリケーション開発によって蓄積された社内ライブラリをもとに独自のフレームワークを開発し，業務への適用を行った。本稿では簡単ながらこの Web アプリケーション開発フレームワークについて紹介を行い，Web アプリケーションの開発に対してどのようなメリットがあるかについて述べていきたい。

2. フレームワークについて

2.1 フレームワークとは何か

フレームワークとは，共通の目的を持つアプリケーションに対し，共通する構造や機能を提供する半完成のアプリケーションのことである。こう説明するとライブラリと同じように思われる方がいるかもしれない。しかしながら，ライブラリはあくまで部品の形で提供されており，メインプログラムは開発者が開発することになる。これに対し，フレームワークはアプリケーションのメインプログラムがフレームワークによって提供されている。開発者は開発しようとするアプリケーション固有の処理を実装した部品を作り，フレームワークがその部品を呼び出すことになる。つま

りライブラリを用いた開発とは視点が逆なのである。図 1 にフレームワークを用いた開発のイメージを示す。

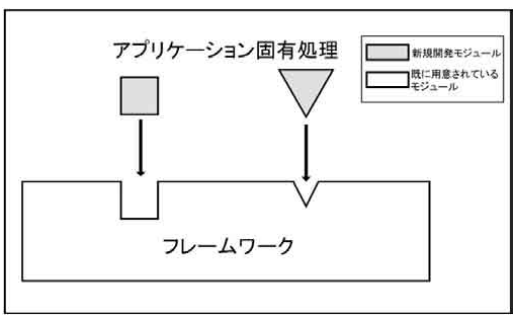

図 1 フレームワークを用いた開発のイメージ

2.2 Web アプリケーション開発におけるフレームワークの重要性

では，なぜ Web アプリケーションの開発において，フレームワークを用いた開発が注目されているのであろうか。理由を簡単に述べてしまうと，Web アプリケーションの開発は「非常に手間がかかる」からである。以下に Web アプリケーションの開発において，開発者が作成しなければならないものを列挙してみよう。

- ユーザに対するフロントエンドとしての画面。
- ビジネスロジック（トランザクション処理）。
- クライアント（画面）からの要求とビジネスロジックのマッピング。
- クライアントからの入力パラメータのチェック。
- ビジネスロジックの処理結果に応じた画面遷移の制御。
- DB アクセス処理（コネクションの確立，トランザクション管理）。
- アプリケーション内のエラーをハンドリングする仕組み。

基本的に Web アプリケーションの開発においては，これらをすべて一から「手作り」しなければならない。この中で，クライアントからの要求とビジネスロジックのマッピング，DB アクセス処理やエラーハンドリングなどといったものはシステムレベルのサービスであり，アプリケーション開発の本質と関係がないのである。また，クライアントから送信されるパラメータのチェックや画面遷移処理は毎回必ず発生する作業であるが，意外に工数を取られる作業である。

よって，上に挙げたような Web アプリケーション開発における煩雑な作業を吸収し，ビジネスロジックの開発に集中できるようなフレームワークが求められているわけである。

3. Java による Web アプリケーション開発フレームワーク

ここでは当社にて開発した Java による Web アプリケーション開発フレームワークについて解説する。まず，フレームワークを開発するにあたって，以下に示す事柄を満たすことを目標とした。

- DB 接続処理やトランザクション管理，エラーハンドリング等のシステムレベルのサービスをフレームワーク側で実装しておき，アプリ開発者にこれらを意識させないようにする。
- 画面遷移やクライアントからの要求のビジネスロジックコンポーネントへの転送といった，Web アプリケーションでは手続きが複雑になる処理を出来るだけ隠蔽し，ロジック開発に集中できるようにする。
- ビジネスロジック開発についても，テンプレー

トを用意して，開発者間でのコード設計のばらつきを減らすようにする。

3.1 基本となる考え方

MVC モデルを基本として設計を行った。MVC モデルとは，アプリケーション構成を，ビジネスロジック実行モジュールである Model，プレゼンテーションロジック実行モジュールである View，処理の流れを制御するモジュールである Controller に分割して組み立てて行く設計手法である。図 2 に MVC モデルでアプリケーションを構築した際の処理の流れについて説明している。

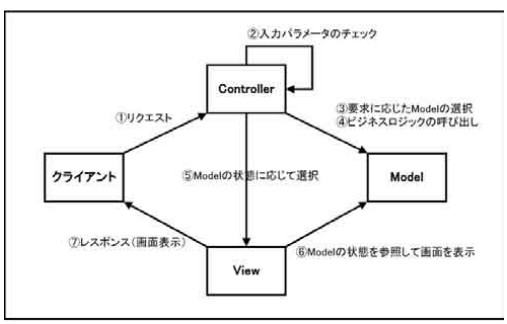

図 2 MVC モデル

理想は Controller，Model，View の結びつきをなるべく疎にして，それぞれに開発者を分担させることができるようにすることである。

3.2 全体の構成

今回作成したフレームワークは大きく分けて Web 層とデータアクセス層に分けることができる。それぞれの役割は以下のようになる。

（Web 層）

・クライアントから送信されてくる HTTP リクエストの処理を担当する。

・クライアントからのサービス要求を適切なビジネスロジックへ渡す。

・遷移先の画面を決定，ディスパッチする。

（データアクセス層）

・データベースアクセス処理を一元管理する。

・ビジネスロジックがデータアクセスする際に必要なコネクションの取得，トランザクション管理を行う。

・容易にデータアクセスオブジェクトを作成できるようなフレームワークを提供する。

以下，それぞれの層について解説していく。

3.3 Web 層

Web 層フレームワークの処理の流れは図 3 のようになっている。網掛けのないクラスはフレームワークで提供されているクラス，網掛けされているクラスはフレームワークが提供するクラスを継承して，実装者が実装するクラス，黒塗りのクラスは実装者が開発するクラスである。

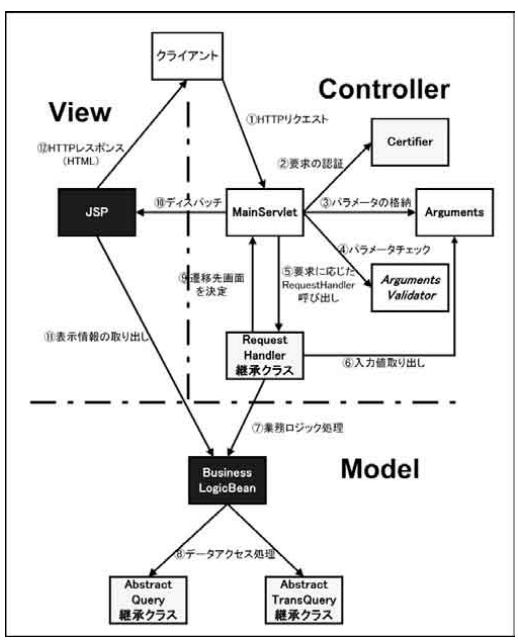

図 3 Web 層フレームワークの処理の流れ

以下，それぞれのモジュールについて解説していく。

(Main Servlet)

クライアントからの HTTP リクエストを集中して受け付けるクラスで，アプリケーション全体のコントローラの役割を持ち，Web 層フレームワークの中心となっている。リクエストパラメータの処理，認証，クライアントから要求された「処理」に応じた Request Handler（後述）実装クラスの検索と処理呼び出し及びその処理結果に応じた JSP へのディスパッチを行う。

このように 1 つのクラスでリクエストを集中的に受け付ける設計（J2EE パターンでは Flont Controller パターンと呼ばれている）にしているため，認証やパラメータの処理，画面遷移処理のようなリクエストのたびに毎回行われる処理をアプリケーション開発者が実装しなくても済むようにしている。

(Certifier)

認証処理を実装するスケルトンクラス。認証処理の内容はアプリケーション要件によって異なるので，アプリケーション開発者が処理を実装する。

(Arguments)

Http Servlet Request にセットされたフォームパラメータを Hashtable 形式で格納するクラス。Http Servlet Request よりも使いやすくしている。パラメータのセットは Main Servlet が行うため，アプリケーション開発者がセット処理を実装する必要はない。

(Request Handler)

クライアントから呼び出される「処理（ログイン処理，検索処理，会員登録処理など）」に相当するクラスである。このクラスの役割は，相当する「処理」に必要とされるビジネスロジックの呼び出しと，その結果に応じた遷移先画面の決定である。Main Servlet がアプリケーション全体の Controller ならば，この Request Handler はビジネスロジック単位の Controller と言える。

アプリケーション開発者は実装したい「処理」ごと（通常はボタン押下処理と 1 対 1 になるであろう）に，この Request Handler を継承したクラスを作成して，ビジネスロジックを呼び出す処理を実装することになる。なお，クライアントのボタン押下処理と Request Handler の結びつけには URL のサーブレットマッピングを利用する。マッピング情報は設定ファイルに定義しておく。

(Business Logic Bean)

ビジネスロジックをカプセル化した JavaBeans である。MVC モデルの中の Model に相当する。基本的にアプリケーション仕様に応じて設計するものであるので，フレームワークの提供するものではない。ただし，後述する JSP が利用するので Java Beans の仕様を満たす必要はある。

(JSP)

いわゆる JSP である。Business Logic Bean を参照し，処理結果をクライアントに表示する。どの JSP に遷移させるのかを決定するのは Request Handler で，実際のディスパッチ処理を行うのは Main Servlet である。

(Abstract Query)

検索系のデータアクセスオブジェクト。Business Logic Bean はこのオブジェクトを通して，データベースの検索処理を行う。詳細はデータアクセス層の節で解説する。

（Abstract Trans Query）

更新処理系のデータアクセスオブジェクト。Business Logic Bean はこのオブジェクトを通して，データベースの更新処理を行う。詳細はデータアクセス層の節で解説する。

さらに，現在開発中の段階であるが，次のようなモジュールを提供する予定である。

（Arguments Validator）

入力パラメータのチェック処理を一括して行うクラスである。入力されたパラメータを格納する Arguments クラスを解析し，入力値の型，文字列長についてチェックを行う。チェック内容については CSV 形式の設定ファイルに定義する（フォーム名に対して定義する）。こうすることによって入力チェック処理をアプリケーション開発者がコーディングしなくて済むようにすることを目指している。

3.4 データアクセス層

データアクセス層のフレームワークはデータベースのコネクションプール管理機能を提供するフレームワークとデータベースへのアクセス機能を提供するフレームワークに分けることができる。図 4 にコネクションプール管理機能を提供するフレームワークの，コネクションプール取得までの処理の流れを，図 5, 6 にデータベースへのアクセス機能を提供するフレームワークの，データアクセス処理の流れを検索処理と更新処理に分けて示す。

以下，図の中に登場したそれぞれのモジュールについて解説していく。

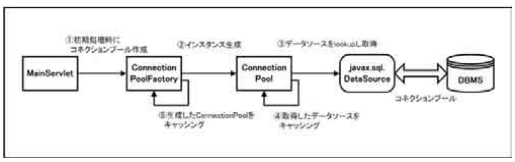

図 4　コネクションプール取得処理の流れ

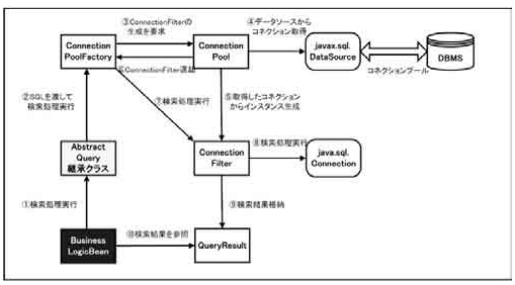

図 5　データアクセス処理（検索）の流れ

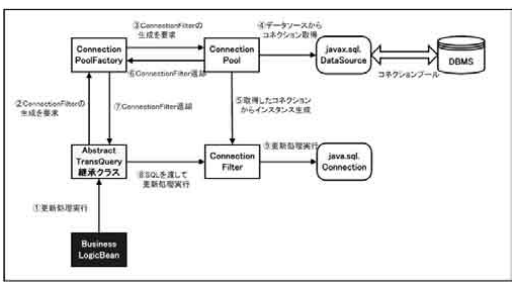

図 6　データアクセス処理（更新）の流れ

（Connection Pool Factory）

データアクセス層のフレームワークの中心となるクラスである。後述する Connection Pool クラスを管理し，データアクセス処理時にはコントローラの役割を果たす。

（Connection Pool）

JNDI ルックアップによって取得されたデータソース（javax. sql. Data Source クラス）を保持し，DB とのコネクションを管理するクラスである。データソースは Hashtable に格納しており，複数 DB にアクセスする必要があるケースにも対応している。

(Connection Filter)

java. sql. Connection クラスをラップし, 使いやすくしたクラスである。DB に対する 1 接続を確保し, DB に対して検索, 更新処理を行うためのインターフェースを提供する。

(Query Result)

DB の検索結果を格納するクラスである。項目名付きテーブルの構造をしており, Result Set と違って行番号を直接指定して値を取り出すことができる。

(Abstract Query)

検索処理をカプセル化したデータアクセスオブジェクトで, “Abstract ～” の名称が示すように抽象クラスである。コネクションを取得して検索処理を実行する処理を実装しており, SQL 文の取得についてのみサブクラスの定義に任せるようにしている。よってアプリケーション開発者は Abstract Query のサブクラスを作成し, SQL 文を実装するだけで検索処理を実行するデータアクセスオブジェクトを作ることができる。

(Abstract Trans Query)

更新処理をカプセル化したデータアクセスオブジェクト。コネクションを取得して, トランザクションを開始, コミット (例外発生時はロールバック) する処理を実装しており, 更新処理の内容はサブクラスの定義に任せるようにしている。アプリケーション開発者は Abstract Trans Query のサブクラスを作成し, 1 トランザクションで行いたい更新処理の内容を実装する。トランザクション管理やコネクションの解放といった処理は実装する必要がない。

4. フレームワークの実装例

この章では簡単な Web アプリケーションの構築を例に, フレームワークの実際の適用方法について解説したいと思う。

4.1 作成するアプリケーション

今回実装例として作成するアプリケーションは, 簡単なクイズアプリケーションである。まず画面遷移を図 7 に示す。最初に問い合わせ画面があり, 4 つの選択肢から 1 つを選んで「回答」ボタンを押す。選択肢が正解であれば次の画面へと進むことができ, 間違っていた場合は問い合わせ画面に戻る。このとき選択した答えが間違いであったことを知らせるメッセージを表示する。

次節よりこのアプリケーションを実装していく手順を示していく。

4.2 ビジネスロジックの実装

まずはビジネスロジックの実装から始める。実装コードをリスト 1 に示す。クラス名は “Apptec Quiz” とした。選択された答えが正解であるかを判定するメソッド mark() を実装する。また, プロパティとして正解, 不正解を表す “correct” プロパティを実装する。

ここで, このビジネスロジック実装クラスがクライアント非依存であることに注目してもらいたい。クライアントが Web ブラウザであることを意識せずに実装を行っている。

4.3 Request Handler の実装

続いて, コントローラの役割を果たす Request Handler を実装する。実装コードをリスト 2 に示

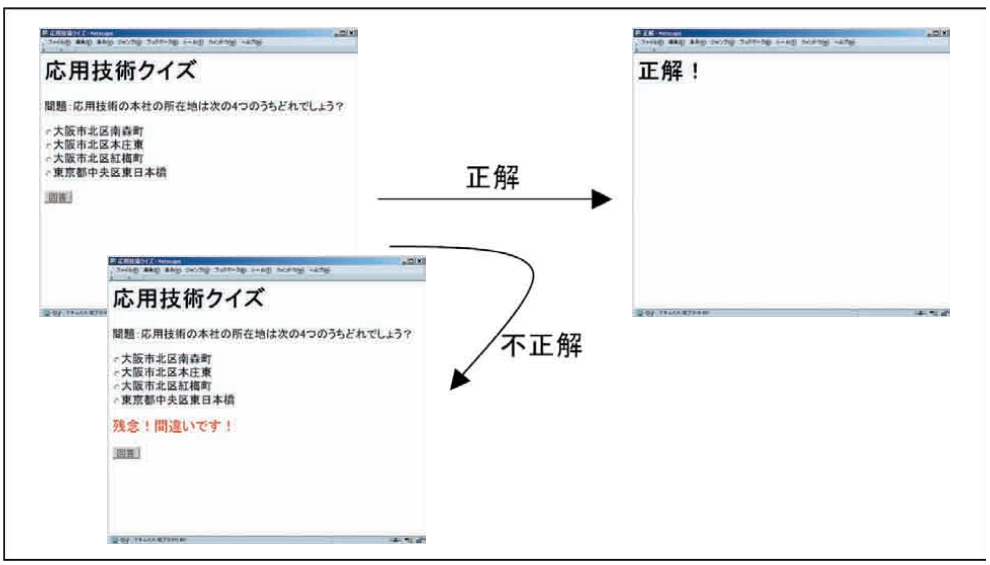

図７　サンプルアプリケーションの画面遷移

```
public class ApptecQuiz implements Serializable {
    private final String CORRECT_CHOICE = "0";
    private boolean correct = true;
    public boolean isCorrect() {
        return this.correct;
    }
    public void setCorrect(boolean correct) {
        this.correct = correct;
    }
    public ApptecQuiz() {
    }
    public void mark(String choice) {
        if (choice.equals(CORRECT_CHOICE)) {
            this.correct = true;
        } else {
            this.correct = false;
        }
    }
}
```

リスト１　Apptec Quiz. Java

す。Request Handler を継承し，handle() メソッドの中に処理を実装する。まずクライアントから送られてきたパラメータを Arguments から取り出し，先ほど実装した Apptec Quiz に渡して，ビジネスロジック処理を実行させる。その後 Apptec Quiz の状態（ここでは correct プロパティ）を判定して，遷移先を決定，return している。

ここで，遷移先は JSP の実 URL ではなく，論理名を返していることに注目してほしい。この論理名と実 URL のマップはリスト３のように設定

```
public class QuizHandler extends RequestHandler {
    public String handle(HttpServletRequestFilter request, Arguments args)
        throws OperationalException, SystemException {
        String selectedOption = args.get("choice");

        ApptecQuiz quiz = new ApptecQuiz();
        quiz.mark(selectedOption);
        request.setAttribute("apptecquiz", quiz);

        if (quiz.isCorrect()) {
            return "correct";
        } else {
            return "quiz";
        }
    }
}
```

リスト２　Quiz Handler. java

```
correct=/correct.jsp
quiz=/quiz.jsp
```

リスト３　JSP の論理名と実 URL のマッピング

ファイルに定義する。こうすることでプログラム中に JSP ファイルの実 URL をハードコーディングすることを防ぎ，アプリケーション構成の変更に対しても柔軟に対処できるようにしている。

4.4 JSP の実装

最後に JSP を実装する。問い合わせ画面の JSP（quiz. jsp）のソースはリスト４のようになっている。

ボタン押下に応じた Request Handler の呼び出しは，form タグの action 属性で行っている。action に記述されている URL の最後のパス“/quiz”が Handler の指定である。実は Web アプリケーションの配備ディスクリプタ“web. xml”にてリスト５のようなサーブレットマッピングを行っており，最後のパスを自由に指定できるようにしている。これを利用して，URL で起動 Handler を指定できるようにし，ボタン押下処理とビジネスロジックのマッピングを直観的で分かり易いようにしている。

実際の Handler と URL のマッピングはリスト６のように設定ファイルに定義している。こうすることで，もしボタンに対する処理（Request Handler）が変わったとしても設定ファイルの変更だけで済むようにしている。

また，リスト４の 13 行目では Apptec Quiz のステータスを参照して，不正解を知らせる文字列を表示するかどうかを判定していることが分かる。

4.5 実装例についてのまとめ

今回は非常に簡単なアプリケーションであったため，フレームワークを用いて実装したメリットはあまり感じられなかったかもしれない。しかしながら，ここまでの流れを見てきて，アプリケー

```
<%@ page contentType="text/html; charset=Shift_JIS"%>
<%@ page import="test.bean.*" %>
<jsp:useBean id="apptecquiz" class="test.bean.ApptecQuiz" scope="request" />
<html><body>
<h1>応用技術クイズ</h1>
<form action='<%= request.getContextPath() %>/handler/quiz' method='post'>
<p>問題：応用技術の本社の所在地は次の 4 つのうちどれでしょう？</p>
<input type='radio' name='choice' value='0'>大阪市北区南森町<br>
<input type='radio' name='choice' value='1'>大阪市北区本庄東<br>
<input type='radio' name='choice' value='2'>大阪市北区紅梅町<br>
<input type='radio' name='choice' value='3'>東京都中央区東日本橋<br>
<%
    if (!apptecquiz.isCorrect()) {
%>
<p>
<font color='red' size='+1'>残念！間違いです！</font>
</p>
<%
    }
%>
<p><input type='submit' value='回答'></p>
</form>
</body></html>
```

リスト 4 quiz.jsp

```
<web-app>
  <servlet>
    <servlet-name>main</servlet-name>
    <servlet-class>jp.co.apptec.web.MainServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>main</servlet-name>
    <url-pattern>/handler/*</url-pattern>
  </servlet-mapping>
</web-app>
```

リスト 5　web.xml でのサーブレットマッピング設定

```
/quiz=test.handler.QuizHandler
```

リスト 6　サーブレットパスと RequestHandler のマッピング

ションの開発の内容が以下のように「型ハメ」されていることに気づかれたと思う。

1) ビジネスロジックを実装した JavaBeans を作成する。
2) ボタン押下処理に対する RequestHandler を作成し，1) で作成したビジネスロジックを呼び出す処理を実装する。
3) JSP を作成し，1) で作成した Java Beans から値を参照して画面を表示するロジックを実装する。また，form に対する Request

Handler の呼び出しを実装する。

開発の順番はこうでなくても構わない。また，1), 2), 3) の開発を別の人に分担することも可能である。

開発の規模が大きくなってくると，このように作業内容が「型ハメ」されていることは作業効率の向上につながってくるはずである。

5. ま と め

本稿では当社 SI 事業部にて開発した Web アプリケーションフレームワークについて簡単ながら解説を行ってきた。以下に本フレームワークによる開発上のメリットについてまとめる。

- システムレベルのサービスを開発者が実装しなくて済む。
- Web アプリケーション開発おいて最も実装の手続きが複雑になる部分を隠蔽し，ビジネスロジックの実装に注力することができる。
- 開発手順を一定の型にはめることによって，作業効率を高めることができ，開発者間でのコードの品質のばらつきを抑えることができる。

本フレームワークを実際の業務開発に適用してみたところ，上記のメリットは概ね得られることができ，開発時の目標はおおよそ達成することができた。しかしながら，今回開発したフレームワークはまだまだ発展途上の段階である。より良いフレームワークとするために今後も更なる改善を行っていきたいと思っている。

最後に，今回開発したフレームワークは決して筆者が 1 人で開発したものではなく，これまでの応用技術の Web アプリケーション開発業務を通して積み重ねられてきた開発ノウハウや豊富なライブラリといった「財産」があったからこそ開発をすることが出来たこと強調しておきたい。特にデータアクセス層のフレームワークはほぼ完成状態であり，筆者はそれに僅かな変更を行っただけであった。ここに，このような貴重な財産を積み上げて頂いた応用技術の諸先輩方に感謝の意を示したい。

参 考 文 献

[1] ディーパック・アラー，ジョン・クルーピ，ダン・マークス著，ウルシステムズ株式会社監訳：J2EE パターン，p.166-178, ピアソン・エデュケーション (2002)

[2] 月刊ジャバワールド 2002 年 5 月号，p.52-86, IDG ジャパン (2002)

[3] 月刊 DB マガジン 2001 年 12 月号，p.75-88, 翔泳社 (2001)

[4] 深澤 良彰：Vol.10　オブジェクト指向再利用の真髄（オブジェクト工房 Web サイト内），http://www.itc.co.jp/kobo/manabu_fo10.html

[5] Inderjeet Singh, Beth Stearns, Mark Johnson, エンタープライズチーム共著：Java2 Platform Enterprise Edition アプリケーション設計ガイド 第 2 版 (J2EE1.3 対応)，http://sdc.sun.co.jp/NASApp/sdcpersonal/private/jdc/download/j2ee_blueprints/j2eebp_2e-ja.pdf