

Webアプリケーション学習用ライブラリ Sabaphy - インターネット書店サンプル -

Sabaphy 0.5.3 向け 佐藤英人 (東京国際大学)

1. はじめに

Sabaphyは、Webアプリケーション開発の学習用に開発されたライブラリです。MVCパターンに沿ったWebアプリケーション開発に利用することができます。

言語はPHP5を使用しています。(1) 簡単なことは簡単に実現できる、(2) 段階的に学習していくことができる、(3) できるだけ世の中の標準的な概念に準拠する、(4) ライブラリ自身も単純で容易に読み解くことができる、といった点を特に重視して作られています。

Sabaphyの利用には、(1) 永続化機能のみを利用する方法、(2) アプリケーションフレームワークを利用する方法、(3) アプリケーションフレームワークとDIコンテナを利用する方法の3種類があります。ここでは2番目の方法を使い、インターネット書店のサンプルを例として説明します。3つの利用方法の概要とこれらの関係については、別稿の「サンプルでみるSabaphyの概要」をご覧ください。

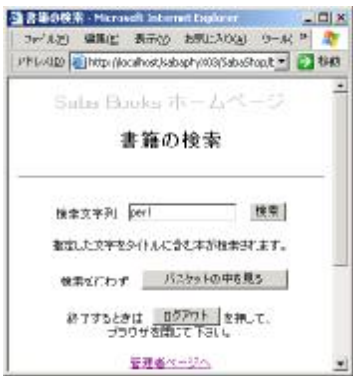
なお、Sabaphyでは、オブジェクト永続化にはJavaの世界のHibernateやEJB3のEntityManagerインタフェースを借用し、セッション管理の実装にはPHPが標準でもつセッション管理機能を利用しています。

2. Sabaphyアプリケーションの仕組み

Sabaphyは種々のWebアプリケーション開発に利用できますが、イメージをつかんでもらうために1つ例を挙げます。図1は後ほど詳しく説明するサンプルの最初の3画面です。


図1 Sabaphyアプリケーションの画面例

(1) 検索指示画面



「Perl」と入力して「検索」ボタンを押す


(2) 検索結果画面



書籍番号	ISBN	タイトル	出版社	著者	価格	数量
11	4797316849	実践PerlDBE	ソフトモジックパブリッシング	木田佳典	2800	<input type="checkbox"/>
12	4873110603	プログラミングPerl volume 1	オライリー・ジャパン	ラリーウォール	5000	<input type="checkbox"/>
13	4894216284	MySQL & Perl Webアプリケーション開発	オライリー・ジャパン	社、ル、ア、ボ、ワ	4700	<input type="checkbox"/>
14	4873110971	プログラミングPerl volume 2	オライリー・ジャパン	ラリーウォール	4700	<input type="checkbox"/>
15	4894216304	オブジェクト指向Perl マスターコース	オライリー・ジャパン	社、ル、ア、ボ、ワ	5200	<input type="checkbox"/>

2件チェックして、「バスケットに入れる」ボタンを押す

(3) バスケット画面



書籍番号	ISBN	タイトル	出版社	著者	価格	数量	選択
12	4873110603	プログラミングPerl volume 1	オライリー・ジャパン	ラリーウォール	5000	1	<input type="checkbox"/>
14	4873110171	プログラミングPerl volume 2	オライリー・ジャパン	ラリーウォール	4700	1	<input type="checkbox"/>

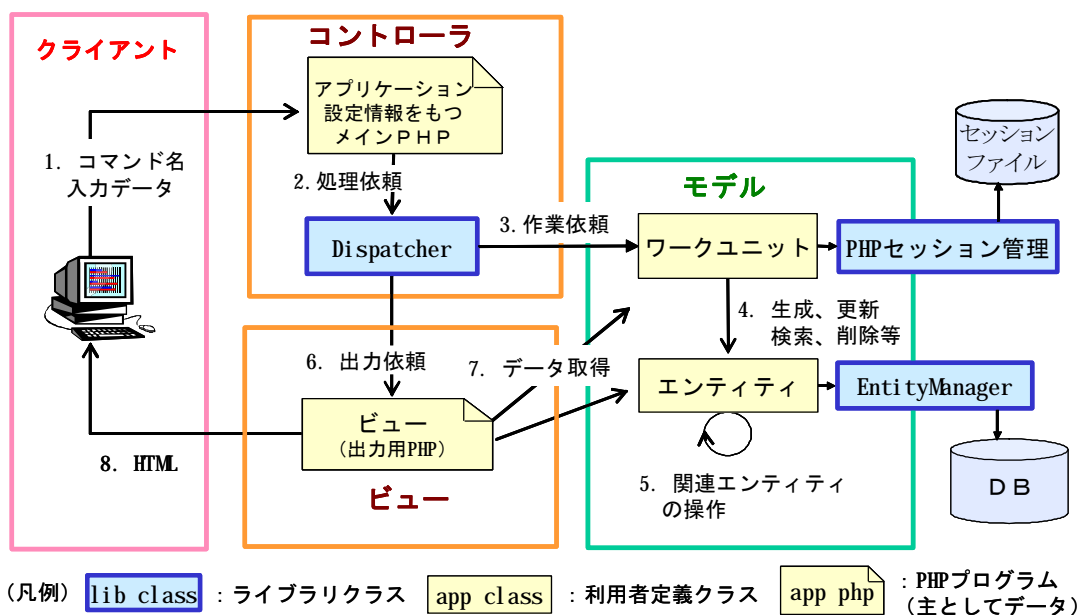
これはインターネット書店の例で、まず書籍の検索を指示するための画面が表示され

ます。そこに検索文字列を入力し、「検索」ボタンを押すと、検索結果が表示されます。ここで右端の「選択」欄のチェックボックスにチェックを入れて「バスケットに入れる」ボタンを押すと、これまでに選択された本の一覧が表示されます。

【備考】ここで使用している例題は、Sabaphyの前身であるPerl版のSabaps用に作られたDBを利用しています。PHPなのにPerl 関連書籍検索になってしまっています。済みません。

この例を利用しながらSabaphyアプリケーションの構造とその動作を説明しましょう。図2はSabaphyを利用したアプリケーションの構造を表しています。Webアプリケーションの世界のMVCパターンをご存じの方は、この図をみただけでSabaphyが何をどのようにやるかおおよそ見当がつくと思います。ここでは予備知識のない人を前提にやや丁寧に説明します。

図2 Sabaphyアプリケーションの構造



利用者が検索指示画面上で検索文字列「perl」を入力して「検索」ボタンを押すと、この入力データはサーバ上のメインPHPに伝えられます。メインPHPはアプリケーションのメインプログラム（フロントコントローラ）です。メインPHPは、アプリケーション設定情報をディスパッチャ（Dispatcher）に与え、処理の実行を依頼します

ディスパッチャは、入力コマンドに応じて処理を振り分けるコンポーネントです。いまの場合、押されたボタンの名前である「検索」がコマンド名で、「検索文字列=perl」が入力データです。ディスパッチャはこのコマンド名に対応するコマンド情報（どのようなアクションを実行するかを指定したアプリケーション設定情報）を見つけ、その情報に従ってワークユニット（WorkUnit）のインスタンスを生成（あるいは再現）し、これにコマンド情報で指定されたアクションの実行を依頼します。

このワークユニットは、一定時間内に行われるあるひとまとまりのビジネス処理ロジックを扱うオブジェクトです。Sabaphyのワークユニットは、この時間がただだか1セッションに限られています。これはEJBの世界のStateful SessionBeanに相当します。

ワークユニットはいくつかのメソッドをもっています。これらはアクションメソッドと呼ばれ、ディスパッチャから依頼されたアクションを実行するものです。そこではディスパッチャから渡された入力データをもとにエンティティの生成、更新、検索、削除等を行い、結果を出力や次のアクセス時に利用できるようにキャッシュします。

エンティティはデータベースに永続化されるオブジェクトです。そのインスタンスはDBテーブルの1行に対応しています。その生成、更新、削除結果はDBに伝えられ、次回以降の接続時にもこの変更結果が反映されます。この永続化サービスを提供するコンポーネントが**EntityManager**です。**Sabaphy**の場合、データベースとのやりとりはユーザプログラムから隠蔽されています。**Sabaphy**利用者は、アプリケーションプログラムを書く際に、SQL文（検索条件文を除く）を書く必要はありません。これはEJBの**CMF (Container Managed Persistence) EntityBean**に相当しています。

いまの書籍検索の例では、ワークユニットとして**OrderWorkUnit**クラス、エンティティとして**Book**クラスを使用します。これらはアプリケーション開発者が用意したものです。**OrderWorkUnit**は**doSearch()**というアクションメソッドをもっており、これをディスパッチャが呼び出します。このメソッドの中で**Book**クラスのインスタンスでタイトルに「perl」という文字列を含むものを検索し、その配列を自身の属性として保持します。

次に、ディスパッチャは出力用に用意された**PHP**を起動します。**Java**の世界の**JSP**に相当します。出力用**PHP**はワークユニットのデータを参照してプレースホルダに値を埋め込み、**HTML**を完成させ、クライアントに送り返します。こうして、今の場合、クライアントのブラウザ上に図1の2番目の画面である検索結果画面が表示されます。

以上がサンプルアプリケーションの入口部分の説明です。後は同様の処理の繰り返しです。呼び出されるワークユニットのアクションメソッドと出力用**PHP**が変わるだけです。

コントローラ（メイン**PHP**と**Dispatcher**）、モデル（ワークユニットとエンティティ）、ビュー（出力用**PHP**）の3者の関係は、**Struts**や**Mjavi**等で**MVC**パターンと呼んでいるものとほぼ同じです。つまり、**Sabaphy**のオブジェクト永続化とセッション管理はEJBに、フレームワークの構造は**Struts**や**Mjavi**に類似しています。これらの学習向けに細部を簡略化したものが**Sabaphy**であるということが出来ます。

3. サンプルプログラムSabaShopの概要

本節では**Sabaphy**に添付されているサンプルプログラムに沿って、アプリケーションの構成要素を説明します。ここではフレームワークの使い方の流れを理解していただくことに主眼をおいています。このため、コードの細部の説明は省略している部分が多いです。**Sabaphy**の詳細については「**Sabaphy**ユーザマニュアル」をご覧ください。

(1) 画面遷移とコマンド情報

サンプルプログラム**SabaShop**は、2節でその一部を紹介したインターネット書店の注文受付プログラムです。図3はその全体の画面遷移をUMLの状態図で書き表したものです。

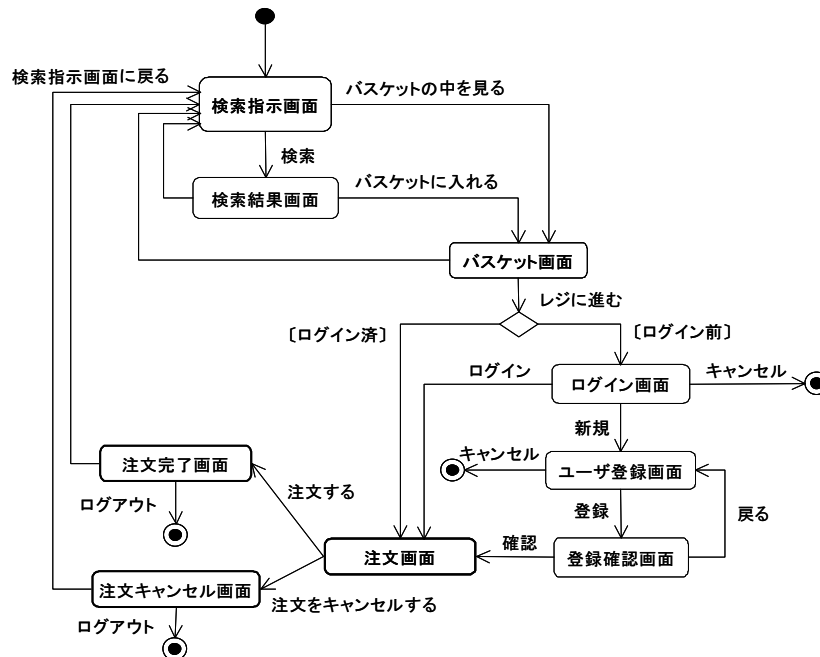
黒丸がアプリケーションの開始を、中黒2重丸が終了を表しています。角の丸い矩形が画面で、矢印は画面遷移の方向を示しています。矢印の上に書かれたラベルは遷移を引き起こすコマンド（イベント）です。先の図1は最初の3つの画面「検索指示画面」「検索結果画面」「バスケット画面」を表しています。見比べていただくと画面遷移図の意味が良く分かります。

バスケット画面以降の処理の主な流れは以下ようになります。バスケット画面で「レジに進む」ボタンが押されたとき、利用者のセッションがログイン済みか否かによって分岐します。この分岐を表しているのが図の菱形です。

ログイン済みのときは直ちに注文画面を表示します。ログイン前ならログイン画面を表示します。既にユーザ名、パスワードをもっている人はここでログインし、注文画面に進みます。そうでない人は「新規」ボタンを押してユーザ登録を行います。ユーザ登録内容を「確認」すると注文画面が表示されます。ここで「注文する」を選ぶと注文完了画面が表示され、「ログアウト」するか、更に別の注文をするために「検索指示画面に戻る」かを選びます。ログアウトすればセッションのログイン済み情報が

消されます。検索指示画面に戻るを選べば、ログインしたまま、はじめの画面に戻ります。

図3 SabaShopの画面遷移



以上がこのサンプルの全体像です。これは現実のインターネット書店を模したものです。検索のやり方や配送方法・支払い方法などのオプションが省略されていますが、処理の流れは現実のものと同様です。

図4 コマンド情報一覧

	受取コマンド	ユーザクラス	アクション	ビュー	送出コマンド
検索・注文	1. null		-	search.html	(検索) (バスケットの中を見る)
	2. 検索		OrderWorkUnit::doSearch()	foundBooks.html	(バスケットに入れる) (検索指示画面に戻る)
	3. バスケットの中を見る		OrderWorkUnit	basket.html	(レジに進む) (basket- 検索指示画面に戻る)
	4. バスケットに入れる		OrderWorkUnit::addSelection()	-	バスケットの中を見る
	5. 検索指示画面に戻る		-	search.html	(検索) (バスケットの中を見る)
	6. basket- 検索指示画面に戻る		OrderWorkUnit::changeSelection()	-	検索指示画面に戻る
	7. レジに進む		OrderWorkUnit::changeSelection(). LoginWorkUnit::checkLogin()	-	logged=>showOrderPage not-logged=>showLoginPage
	8. showOrderPage	Customer	OrderWorkUnit::prepareToOrder()	order.html	(注文する) (注文をキャンセルする)
	9. 注文する	Customer	OrderWorkUnit::fixOrder()	orderCompleted.html	(ログアウト) (検索指示画面に戻る)
	10. 注文をキャンセルする	Customer	OrderWorkUnit::cancelOrder()	cancelCompleted.html	(ログアウト) (検索指示画面に戻る)
ログイン・ログアウト	11. showLoginPage		-	login.html	(ログイン) (キャンセル) (新規)
	12. ログイン		LoginWorkUnit::doLogin()	-	showOrderPage
	13. キャンセル		-	-	検索指示画面に戻る
	14. 新規		-	inputCustomer.html	(登録) (キャンセル)
	15. 登録		NewCustomerWorkUnit::doInput()	confirmCustomer.html	(確認) (戻る)
	16. 確認		NewCustomerWorkUnit::doSave()	-	showOrderPage
	17. ログアウト		LoginWorkUnit::doLogout()	-	null

- 備考:
1. nullは初期の起動コマンドである。
 2. 送出コマンド欄でshowLoginPageなどカッコ付きでないコマンドはプログラム内で発行されるコマンドである。
 3. その他のカッコでくられた送出コマンドは、ブラウザ上でユーザがボタンを押すことで発行される。
 4. [戻る]はJavaScriptで処理され、サーバプログラムには影響しない。
 5. ログインユーザがユーザクラス欄のクラスまたはそのサブクラスのインスタンスでなければエラーとなる。

図4は図3の画面遷移図をもとに作られたコマンド情報の一覧です。この各行はコマ

ンドを受け取ったとき、どのワークユニットのどのアクションメソッドを実行するか、そしてその結果をどのビュー（出力用PHP）を使って表示するかを1行で表しています。

1行目の受取コマンド**null**が開始時のコマンドです。コマンドが指定されていないとき、このコマンドが実行されます。このときビュー欄に書かれている**search.html**という出力用PHPを使って次画面のHTMLを生成します。送出コマンド欄に書かれている「検索」と「バスケットの中を見る」はこのHTMLの表示画面上で選択できるコマンドです。

【備考】出力用PHPは動的に変化する出力内容の埋め込みにPHPプログラムを使います。しかし、その他の部分は普通のHTMLです。そこで、ここでは拡張子を「.html」で表記しています。なお、実際のファイルでは拡張子は「.html.php」としています。

2行目は「検索」ボタンが押されたときの処理です。アクション欄に書かれているように、**OrderWorkUnit**を生成するか再現し、**doSearch()**というアクションメソッドを実行します。その後、ビュー欄に書かれている**foundBooks.html**という出力用PHPを使って次画面のHTMLを生成します。

3行目は「バスケットの中を見る」ボタンが押されたときの処理です。アクション欄には、ワークユニットクラス**OrderWorkUnit**の指定だけでアクションメソッドが書かれていません。このときはワークユニットの生成または再現のみ行い、何もせずにビューの表示を行います。ビューでワークユニットのデータを利用するとき、例外的に、このような書き方をします。

4行目以降も同様に読んでいくことができます。7行目はアクション欄にアクションメソッドが2つ記述されています。この2つのメソッドの実行を連続して行う指定です。また、ビューの指定がなく、送出コマンド欄に、「**logged=>showOrderPage, not-logged=>showLoginPage**」と書かれています。これはアクションメソッド**checkLogin()**の戻り値による分岐で、戻り値が**"logged"**なら**showOrderPage**コマンドを、**"not-logged"**なら**showLoginPage**コマンドを発行することを表しています。

図3の画面遷移図はアプリケーションの状態遷移を表していますので、図4のコマンド情報一覧は状態遷移図に対応するデシジョンテーブルであるということが出来ます。このコマンド情報一覧がSabaphyフレームワークの核心です。ここでアプリケーション開発者が用意したワークユニットのアクションメソッド、出力用PHPといったコンポーネントが画面遷移に結びつけられ、アプリケーションが完成します。

(2) コントローラ

1) アプリケーション設定とメインプログラム

Sabaphyではアプリケーション設定情報を、メインプログラムであるPHPファイル（メインPHP）に書くようにしています。この手のフレームワークでは、設定ファイルを別に用意するのが普通ですが、これまでの経験では、そのようにすると初心者にはプログラムの処理の流れを追うことが難しくなります。PHPのようなインタプリタ言語では、プログラムでも設定ファイルでもメンテナンス上の手間は変わりません。そこで、ここでは設定情報をプログラムとして書く方法を採用しています。

以下のリスト1がSabaShopのメインプログラムです。これはSabaShopフォルダの中の**bookshop.php**というファイルに書かれています。

<リスト1> メインプログラム (bookshop.php)

```

1) <?php
2) //////////////////////////////////////
3) // bookshop.php : Simplified WebShop Sample
4) //////////////////////////////////////

```

```

5) // (1) クラスオートローダの設定
6) require_once '../sabaphy/AutoLoader.class.php';
7) AutoLoader::setup("model");
8)
9) // (2) セッションの開始と初期設定
10) session_start();
11) $eh = new ExceptionHandler();
12) $em = new EntityManager('db/mydb');
13)
14) // (3) ディスパッチャの生成
15) $d = new Dispatcher($eh, $em);
16)
17) // (4) コマンド情報の設定
18) $d->addCommand(' null', array(
19)     'view' => 'view/search.html.php' ));
20) $d->addCommand(' 検索', array(
21)     'action' => 'OrderWorkUnit::doSearch',
22)     'view' => 'view/foundBooks.html.php' ));
    -- (略) --
34) $d->addCommand(' レジに進む', array(
35)     'action' => array(
36)         'OrderWorkUnit::changeSelection', 'LoginWorkUnit::checkLogin'),
37)     'forward' => array(
38)         'logged' => 'showOrderPage', 'not-logged' => 'showLoginPage' ));
39) $d->addCommand(' showOrderPage', array('userClass' => 'Customer',
40)     'action' => 'OrderWorkUnit::prepareToOrder',
41)     'view' => 'view/order.html.php' ));
    -- (略) --
64) $d->addCommand(' ログアウト', array(
65)     'action' => 'LoginWorkUnit::doLogout',
66)     'forward' => ' null' ));
67)
68) // (5) コマンドの実行
69) $d->execute();
70)
71) ?>

```

メインプログラムはブラウザから呼ばれると立ち上がるものです。これはリスト1にあるように、5つの部分から構成されています。なお、同じような記述が続くところは、スペースを節約するために「-- (略) --」と書いて詳細を省略していません。

2) クラスオートローダの設定

メインプログラムの先頭で使用するプログラムファイルを指定します。ここではこれを簡略化するために、**Sabaphy**組み込みの**AutoLoader**を使用します。6行目がその読み込み、7行目がその初期設定です。

AutoLoaderはクラスが使用されたとき、対応するクラスファイルを指定された検索パスから検索して読み込むものです。デフォルトでは**Sabaphy**のライブラリとメインプログラムが属するフォルダが検索パスに組み込まれています。

7行目の**AutoLoader::setup("model")**は、**model**フォルダを検索パスに追加する指定です。この**model**フォルダに、このアプリケーションで使用するクラスが格納さ

れています。

3)セッションの開始と初期設定

10行目はセッション開始または再現する指示です。ここの`start_session()`はPHPの標準関数です。11行目は例外処理を扱う`ExceptionHandler`のインスタンスの生成です。これを使うことで、PHP標準のエラーハンドラを置き換え、例外を捕捉して、詳しいエラー情報が表示されるようになります。これを後でディスパッチャに組み込みます。

12行目は永続化サービスを提供する`EntityManager`のインスタンスの生成です。引数で与えている`'db/mydb'`は、DBファイルへのパスです。`EntityManager`は、このように一度生成すれば、以後グローバル関数`getEntityManager()`でこのインスタンスを取得できます。`EntityManager`の使用例は後で出てきます。

【備考】Sabaphyの`EntityManager`は、Javaの世界のEJB3.0の`EntityManager`やHibernateの`session`とよく似たインタフェースで永続オブジェクトの生成、更新、削除、検索のサービスを提供します。この例ではDBMSとしてSQLiteを使用していますが、MySQLやPDOも使うことができます。MySQLやPDOを使うときは、インスタンス生成時に与える引数が異なってきますが、それ以外の利用方法はまったく同じです。詳しくは「ユーザマニュアル」を参照してください。

4)ディスパッチャの生成

Sabaphyアプリケーションの全体をコントロールするのは、2節で説明したように、ディスパッチャです。そこで実行に先立ってディスパッチャ (`Dispatcher`) のインスタンスを生成します。それが15行目です。その後、このディスパッチャオブジェクトに設定情報を伝える形で、アプリケーションの設定を行います。

ディスパッチャの生成時に引数として、`ExceptionHandler`と`EntityManager`のインスタンスを与えます。`ExceptionHandler`は11行目で生成したものです。これを組み込むことで、例外発生時にJavaと同様のスタックトレース情報が表示されるようになります。`EntityManager`は12行目で生成したものです。これを組み込むことで、トランザクションモードでコマンドの実行が行えます。

【備考】このオブジェクト生成時に組み込むコンポーネントを指定することを `Constructor Injection` といいます。ここではDIコンテナをいりませんが、DI (Dependency Injection: 依存性注入) 手法をここで使っています。これにより、別の`ExceptionHandler`や`EntityManager`に差し替えることができます。

5)コマンド情報の設定

18～26行はコマンド情報の設定です。ここで図4のコマンド情報一覧のデータをディスパッチャに与えていきます。コマンド情報の記述には、PHPの連想配列 (`array(キー=>値, ...)`) を利用しています。18～19行は図4の1行目に対応しており、`null` (コマンドの指定がない) とき、ビューファイル `view/search.html.php` を使ってHTMLを生成するということを指定しています。

20～22行は図4の2行目に対応しており、`OrderWorkUnit`がセッションに登録されていなければ新規に生成し、登録されていればそれを再現し、アクションとして`doSearch()`メソッドを呼び出し、最後にビューファイル `view/foundBooks.html.php` を使ってHTMLを生成するということを指定しています。

64～66行は図4の最後の行に対応しています。ここで`LoginWorkUnit`の`doLogout()`メソッドを呼び出します。その後の `'forward'=>'null'` はnullイベントの発行を意味しています。これで再び検索画面が表示されます。

6)コマンドの実行

以上でアプリケーションの設定は終わりです。次いで69行目で、コマンド情報をセットしたディスパッチャに実行を指示します。ここでこのPHPが呼び出されたコ

マンドに応じて、コマンド情報が選択され、そこに記載されたワークユニットのアクションメソッドが起動されます。

ワークユニットが指定されていないとき、あるいはワークユニットの処理が終わったとき、その後に記載されたviewの指定に従ってHTMLの生成が行われます。生成するHTMLが指定されていないときは、フォワードが指定されていますので、そこで指定されたコマンドを実行します。この実行が正常に終われば、ディスパッチャはDBにコミットを発行して、DBへの変更を確定します。

このディスパッチャによって行われる処理がSabaphyのアプリケーションフレームワークの実装です。

(3) ワークユニット

1) サンプルで使用するワークユニット

ワークユニットはコマンド情報に従って起動されるアクションメソッドをもつオブジェクトです。通常はひとまとまりの作業毎にクラスを用意します。このサンプルでは本の検索から注文の確定までを扱うOrderWorkUnit、ログイン処理を行うLoginWorkUnit、顧客の登録を行うNewCustomerWorkUnitの3つのワークユニットを使用しています。

これらのうち、OrderWorkUnitとNewCustomerWorkUnitは作業データを属性値としてもちます。これらの属性値はセッションが維持されている間利用可能です。これらのワークユニットはEJBのStateful SessionBeanに相当します。LoginWorkUnitは作業データをもたないその場限りの作業を記述するワークユニットです。これはEJBのStateless SessionBeanに相当します。

本節ではOrderWorkUnitを例として、これまで主に説明してきた図1の3画面に係るアクションを取り上げ、ワークユニットの機能を解説します。なおOrderWorkUnitクラスはSabaShop/modelフォルダの中のOrderWorkUnit.class.phpファイルで定義されています。

2) ワークユニットクラス

Sabaphyではワークユニットも普通のオブジェクトとして定義します。その生成、再現は、上で述べたコマンド情報に従ってフレームワークが行います。削除はセッションが切断されたときにPHPが自動的に実行します。以下のリスト2はOrderWorkUnitのクラス定義です。クラス定義ファイルはSabaShop/modelフォルダに格納されています。

<リスト2> ワークユニットクラスの例 (model/OrderWorkUnit.class.php)

```

1) <?php
2) //////////////////////////////////////
3) // OrderWorkUnit.class.php : 注文の処理
4) //////////////////////////////////////
5) class OrderWorkUnit {
6)     public $basket = null;
7)     public $order = null;
8)     public $foundBooks = array();
9)
10)    function OrderWorkUnit() {
11)        $this->basket = new Basket();
12)    }
13)    //////////////////////////////////////
14)    function doSearch($commandInfo) {
15)        -- (略) --
26)    }
```

```

27)     function addSelection($commandInfo) {
        -- (略) --
33)     }
34)     function changeSelection($commandInfo) {
        -- (略) --
44)     }
45)     function prepareToOrder($commandInfo) {
        -- (略) --
50)     }
51)     function fixOrder($commandInfo) {
52)         $this->order->fixOrder();
53)         $this->order = null;
54)         $this->basket->clear();
55)     }
56)     function cancelOrder($commandInfo) {
57)         $this->order->delete();
58)         $this->order = null;
59)     }
60) }
61)
62) class exceptionOrderWorkUnit extends exception {
63)     public $options = array('trace'=>false);
64) }
65) ?>

```

6～8行が属性の宣言です。**\$basket**には、バスケットの内容を表すオブジェクトを格納します。これは、バスケットアイテム（注文された本と注文数量のペア）の配列をその属性値としてもつものです。**\$order**には、注文内容が決まったときに生成される注文オブジェクトを格納します。**\$foundBooks**には、検索で見つかった本の配列を格納します。

【備考】ここのバスケットは、最近はショッピングカートあるいはカートということが多いようです。どちらも同じで、購入予定のものを入れておく入れ物です。

10～12行は、このクラスのコンストラクタです。インスタンス生成時に呼びされます。**Basket**のインスタンスを生成して、属性**basket**にセットします。**PHP5**では、6行目で「**\$basket = new Basket()**」とすることは許されていません。そこでコンストラクタの中で初期化することになっています。

14～59行はアクションを記述するメソッドで、フレームワークから呼び出されます。これらについては後で説明します。62～64行はこのワークユニットで発行する例外（エラー情報をもつオブジェクト）のクラス定義です。エラー発生時にスタックトレースを表示しない例外オブジェクトとして定義しています。

3)アクションの呼び出しと実行

先のリスト1（メインプログラム）の30～32行のコマンド情報に書かれているように、クライアント画面で「検索」ボタンが押されると**OrderWorkUnit**の**doSearch()**アクションメソッドが呼び出されます。この呼び出しは、フレームワークの中で自動的に行われます。この自動的に行われる部分を書き出したものが下のリスト3です。

<リスト3> アクションの呼び出し

```

1) if (! isset($_SESSION['OrderWorkUnit'])) {
2)     $_SESSION['OrderWorkUnit'] = new OrderWorkUnit();
3) }

```

- ```

4) $workunit = $_SESSION['OrderWorkUnit'];
5) $workunit->doSearch($commandInfo);

```

Sabaphyでは、ワークユニットオブジェクトをPHPの標準グローバル変数である\$\_SESSIONにクラス名をキーとして格納します。これにより、セッションが維持されている間、同じインスタンスを使用することができます。1～4行がこのワークユニットオブジェクトの生成、再現の処理です。

次いで、5行目でこのワークユニットにdoSearch()を依頼します。引数はコマンド情報を属性値として格納するCommandInfoオブジェクトです。今の場合、「検索」コマンドに対応するコマンド情報で、リスト1の20～22行で与えた連想配列の各要素を属性値としてもつオブジェクトです。

上のリスト3の5行目が実行されると、OrderWorkUnitクラスのメソッドdoSearch()が実行されます。それはリスト2の14～26行のプログラムです。そこで省略されていたメソッド本体は以下のようになっています。

<リスト4> アクションメソッドの例 (OrderWorkUnitのメソッドの1つ)

```

14) function doSearch($commandInfo) {
15) $searchStr = $_REQUEST['searchStr'];
16) if (extension_loaded("mbstring")) {
17) $searchStr = mb_convert_kana($searchStr, "as");
18) }
19) $searchStr = trim($searchStr);
20) if (! $searchStr) {
21) throw new exceptionOrderWorkUnit(
 "検索文字列が指定されていません。");
22) }
23) $this->foundBooks = getEntityManager()->query('Book',
24) "title like ?",
25) array("%$searchStr%"));
26) }

```

15行目で利用している\$\_REQUEST['searchStr']は、クライアントのブラウザから送られてきた入力データで、今の場合、検索用の文字列データです。15行目でそれを取りだし、\$searchStrという変数に代入します。今の例では「Perl」という文字列が\$searchStrに代入されます。16～18行では全角英数字を半角英数字に直しています。19～22行は入力値のチェックです。実質的なプログラムは23～25行です。

#### 4) 検索メソッドの利用

23～25行は、EntityManagerがもつ検索メソッドquery()を使って、Bookクラスから条件を満足するBookインスタンスを検索し、その配列を自身の属性foundBooksにセットしています。このquery()はEJB2.0で導入された問い合わせメソッドEJBQLと同様の性格のもので、以下の構文を持っています。

```
EntityManager->query(class, cond, array(arg1, arg2...));
```

ここでEntityManagerは、先にてきた永続化サービスを提供するオブジェクトです。Sabaphyでは上にあるようにグローバル関数getEntityManager()を使って、そのインスタンスを取得できます。

引数のclassは検索対象クラス、condは問い合わせ用の条件式 (SQLのwhere句に対応する)、arg1, arg2...は問い合わせ文中の変数「?」に代入される値です。

23～25行が実行されると、第2引数の条件式をもつSQL文が発行され、得られた行データに対応するBookクラスのインスタンスのリストが、ワークユニットのfoundBooksという属性 (配列変数) にセットされます。この属性値である配列は検

索結果を表示するビューで利用します。

## 5) セッション管理機能の利用

上の例は入力データを使って検索し、結果を出力に回すだけの単純なものでした。次のリスト5は選択された本をバスケットに追加していくもので、バスケット情報を保持するためにセッション管理機能が使われています。

このメソッドは、検索結果画面でクライアントが注文する本のチェックボックスをチェックして「バスケットに入れる」ボタンを押したときに起動されます。

<リスト5> セッション管理を利用するアクション (OrderWorkUnitのメソッド)

```

27) function addSelection($commandInfo) {
28) if (! isset($_REQUEST['selectedBookIds'])) return;
29) $newlySelectedBookIds = $_REQUEST['selectedBookIds'];
30) foreach ($newlySelectedBookIds as $bookId) {
31) $this->basket->addItem($bookId, 1);
32) }
33) }

```

29行目で、リクエストからチェックされた本のidの配列を取得します。30~32行でこのチェックされた本をバスケットに追加します。バスケットへの追加は、バスケットがもつメソッドaddItem()を利用しています。バスケットクラスについては、後述します。

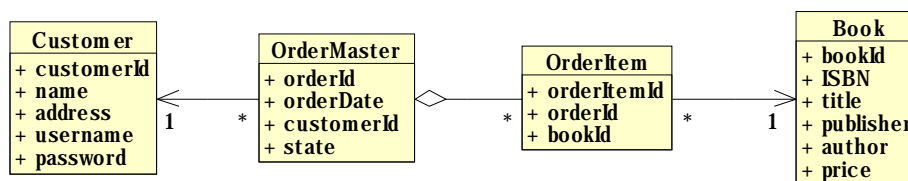
バスケット (\$this->basket) はこのワークユニットの属性です。SabaphyのワークユニットはEJBのStateful SessionBeanと同様にセッションの継続中その属性値を保持しますので、複数回のアクセスに分かれる場合でも、属性値があたかもメモリ上に常駐しているかのように扱うことができます。

## (4) エンティティ

### 1) サンプルで使用するDBテーブル

エンティティはデータベースに永続化されるオブジェクトで、データベーステーブルの1行に対応しています。このサンプルでは図5に示す4つのデータベーステーブルを利用しています。顧客 (Customer)、注文 (OrderMaster)、注文明細 (OrderItem)、本 (Book) の4つで、これらのテーブルは外部キーを使って図のような関連をもっています。

図5 サンプルで使用するDBテーブル



エンティティクラスは、これらのテーブル1つ1つに対応して定義されます。そこでは列 (属性) に関する情報とメソッドが定義されます。

### 2) 関連をもたないエンティティ

まず簡単な例からご紹介しましょう。リスト6はBookクラスの定義です。7~12行が属性の宣言です。永続化する属性は、ここにあるように、"public"として宣言します。

<リスト6> エンティティの例 (model/Book.class.php)

1) <?php

```

2) ///
3) // Book.class.php
4) // 書籍情報を収録するクラス
5) ///
6) class Book {
7) public $bookId;
8) public $ISBN;
9) public $title;
10) public $publisher;
11) public $author;
12) public $price;
13)
14) function keyName() { return 'bookId'; }
15)
16) // 生成と削除 ///////////////////////////////////
17) function init($ISBN, $title, $publisher, $author, $price) {
18) $this->ISBN = $ISBN;
19) $this->title = $title;
20) $this->publisher = $publisher;
21) $this->author = $author;
22) $this->price = $price;
23) getEntityManager()->save($this);
24) }
25) function delete() {
26) getEntityManager()->delete($this);
27) }
28) }
29) ?>

```

1 4行目の**keyName()**は主キー属性名を返すメソッドです。これはO/Rマッピング用の情報として使われます。

1 7～2 4行は、インスタンスの生成直後に呼び出される初期化メソッドです。初期データを属性にセットした後、**EntityManager**に依頼して、永続オブジェクトとして保存します。2 5～2 7行は、削除時の処理で、**EntityManager**に依頼して当該オブジェクトのDBデータを削除します。

#### 【備考】コンストラクタと初期化メソッド

リスト6の**init()**で行っているような初期化は、**Java**などではコンストラクタで行うのが普通です。**PHP5**では複数のコンストラクタの併存が許されません。そこで、引数をもつコンストラクタを定義する代わりに、**init()**というメソッドを使うようにしています。

2 3行目が呼び出される時点では、主キー**bookId**の値は決まっていません。このときは、自動的に連番の主キー値が**DBMS**によって与えられます。

**Sabaphy**のエンティティクラスは、何も継承しない普通のクラスです。その生成、更新、削除を**EntityManager**に通知することで、永続オブジェクトにします。このやり方は、**Java**の世界の**Hibernate**が行い、**EJB3.0**でも採用される見込みのものです。

**Book**や**Customer**といった他のクラスへの参照をもたないエンティティの場合、リスト6のように属性、**keyName()**、**init()**、**delete()**を定義するだけで利用することができます。

### 3)関連オブジェクト

外部キーをもち、オブジェクト間の関連を記述するオブジェクトでは、関連を利用

した導出属性（その値が計算によって求まる属性、派生属性ともいう）を予めクラスに定義しておくことで、ワークユニットやビューでの操作が簡単になります。リスト7はそのような関連クラスである注文クラス（OrderMaster）のコードです。

<リスト7> 関連クラス（model/OrderMaster.class.php）

```
1) <?php
2) ///
3) // OrderMaster.class.php : 注文を収録するクラス
4) ///
5) class OrderMaster {
6) public $orderId;
7) public $orderDate;
8) public $username;
9) protected $customerRef;
10) protected $itemsRef;
11) public $state = '受注前';
12)
13) function keyName() { return 'orderId'; }
14)
15) /////// 生成と削除 ///////
16) function init($username, $basketItems) {
17) -- (略) --
33) }
34) function delete() {
35) foreach ($this->items() as $item) {
36) $item->delete();
37) }
38) getEntityManager()->delete($this);
39) }
40)
41) /////// アクション ///////
42) function fixOrder() {
43) $this->state = '受注済';
44) getEntityManager()->update($this);
45) }
46)
47) /////// 導出属性 ///////
48) function customer() {
49) if (!isset($this->customerRef)) {
50) $this->customerRef = getEntityManager()->
51) find('Customer', $this->username);
52) }
53) return $this->customerRef;
54) }
55) function items() {
56) if (!isset($this->itemsRef)) {
57) $this->itemsRef = getEntityManager()->query('OrderItem',
58) "orderId == ?",
59) array($this->orderId));
60) }
61) return $this->itemsRef;
62) }
63) function totalNetAmount() {
64) -- (略) --
```



```

7) public $basketItems = array();
8)
9) function clear() {
10) $this->basketItems = array();
11) }
12) function addItem($bookId, $quantity) {
13) -- (略) --
23) }
24) protected function findItem($bookId) {
25) -- (略) --
31) }
32) }
33) ?>

```

ここでは、7行目で**BasketItem**の配列をもつ属性**basketItems**を定義しています。属性の定義は、永続オブジェクトと同様です。永続オブジェクトと異なるのは、その生成、更新に**EntityManager**への通知を必要としないということだけです。

## 2) 一時使用オブジェクトとセッション

バスケットの内容は、長期に保存する必要はありませんが、注文が完了するまで、保持しておく必要があります。このオブジェクトの一時的な保持は、先に説明したワークユニットの機能によって実現されます。

先のリスト2を見直してください。**Basket**のインスタンスはリスト2の**OrderWorkUnit**の12行目で作られ、**basket**というワークユニットの属性にセットされています。先に述べたようにワークユニットの属性値はセッションが維持されている間、保持されるので、その要素である**Basket**のインスタンスも保持されます。**Basket**の属性値である配列に格納される**BasketItem**のインスタンスも保持されます。

このように**Sabaphy**では、一時使用オブジェクトをワークユニットの属性値あるいは属性値の一部に格納することで、セッション管理に気をつかうことなく、一時使用オブジェクトを利用することができます。

## (6) ビュー

### 1) ビューとPHP

ディスパッチャは、ワークユニットの処理が終わると、コマンド情報の指定に従って、ビューに**HTML**の生成を依頼します。**Sabaphy**ではビューの実装に**PHP**を使用します。**PHP**の言語仕様は**HTML**の中にプログラムを埋め込む**HTML**テンプレートですから、これが**PHP**本来の使い方であるといえます。

**PHP**には以下の2つの特殊タグが用意されています。

- 1) `<? ..... ?>` : プログラムタグ (.....のプログラムを実行する)
- 2) `<? = ..... ?>` : 値埋め込みタグ (.....の式を評価した結果を埋め込む)

プログラムタグを使えば、任意のプログラムを記述することができますが、ビューに書かれたプログラムは他の目的には使えません。共用利用できるアプリケーションロジックは、エンティティやワークユニットに記述し、ビューはそれらからデータをもらって出力するだけのプログラムにします。

### 2) サンプルで使用する出力用PHP

サンプルでは画面对応に9つのビューを記述する出力用**PHP**を使用しています。これらはいずれも**HTML**記述がほとんどで、最小限のプログラムが埋め込まれているだけです。そこで拡張子を「**.html.php**」にしています。ビュープログラムは**SabaShop/view**フォルダに格納されています。

下のリスト9はその1つで、「バスケットの内容」を表示する出力用PHPです。

<リスト9> ビューを記述するPHP (view/basket.html.php)

```

1) <html>
2) <head>
3) <title>バスケットの内容</title>
4) </head>
5) <body>
6) <h2 align="center">バスケットの内容</h2>
7) <form method="POST">
8) <table border="1" width="100%">
9) <p align="center">
10) <tr>
11) <td align="center">書籍番号</td>
12) <td align="center">ISBN</td>
13) <td align="center">タイトル</td>
14) <td align="center">出版社</td>
15) <td align="center">著者</td>
16) <td align="center">価格</td>
17) <td align="center">数量</td>
18) <td align="center">選択</td>
19) </tr>
20) <?
21) foreach ($workunit->basket->basketItems as $e) {
22) ?>
23) <tr>
24) <td><?= $e->content->bookId ?></td>
25) <td><?= $e->content->ISBN ?></td>
26) <td><?= $e->content->title ?></td>
27) <td><?= $e->content->publisher ?></td>
28) <td><?= $e->content->author ?></td>
29) <td align="right"><?= $e->content->price ?></td>
30) <td align="center">
31) <input type="text" name="quantity-<?=
32) <?= $e->content->bookId ?>"
33) value="<?= $e->quantity ?>" size=2>
34) </td>
35) <td align="center">
36) <input type="checkbox" name="selectedBookIds[]"
37) value="<?= $e->content->bookId ?>" checked>
38) </td>
39) </tr>
40) <? } ?>
41) </table>
42) <p align="center">
43) <input type="submit" name="_COMMAND" value="レジに進む">
44) <input type="submit" name="_COMMAND" value="検索指示画面に戻る">
45) </p>
46) <hr>
47) -- (略) --
48) </body>

```

55) </html>

20～22行と39行がプログラムの埋め込みです。埋め込まれたプログラムを抜き出すと以下の通りです。

<リスト10> ビューに埋め込まれたPHPプログラム

```

20) <?
21) foreach ($workunit->basket->basketItems as $e) {
22) ?>
 :
39) <? } ?>

```

Sabaphyのビューの中では、\$workunitという変数で直前に使われたワークユニットを参照することができます。今の場合、このワークユニットの属性basketにバスケットの内容がセットされています。21～39行のループで、ワークユニットのバスケットからバスケットアイテムを1件ずつ取り出し、処理を行います。

リスト9の24～29行では値埋め込みタグを使用しています。31～32行は数量の入力欄です。クライアントが数量を変更したときそれを捕捉できるように、「quantity-<本のid>」という名称を入力コントロールの名前にしています。35～36行は選択を示すチェックボックスです。ここではチェックボックスの名前に「selectedBookIds[]」という[]付の名前が使われています。これはPHPの標準機能で、複数の値を配列として取り出したいとき、このように名前に[]をつけます。

以上で、このサンプルプログラムの説明は終わりです。

## 4. サンプルの動かし方

### (1) 前準備

本節ではこのサンプルプログラムをWindows上で動かす場合を想定して説明します。Sabaphyアプリケーションを動かすためにはApacheとPHP5が必要です。これらはWebアプリケーションを扱う雑誌や入門書の付録のCDにしばしば収録されています。インターネットからダウンロードすることもできます。まず、これらを入手して、インストールして下さい。

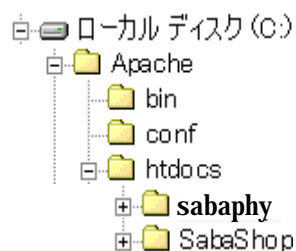
インターネットからの入手先は以下の通りです。

- 1) PHP5: <http://www.php.net/>
- 2) Apache: <http://www.apache.org/httpd>

### (2) Sabaphyライブラリとサンプルの配備

Sabaphyの配備は至って簡単です。Apacheでドキュメントホームとして指定しているフォルダにコピーするだけです。ApacheのデフォルトではApacheフォルダの中のhtdocsフォルダがそれです。この中にライブラリフォルダであるsabaphyとサンプルプログラムフォルダであるSabaShopをコピーして下さい。

図6 Sabaphyライブラリとサンプルの配備



### (3) サンプルの実行

まずApacheを起動して下さい。次いで、ブラウザを立ち上げて、そのアドレス欄に次のURLを入力して下さい。

<http://localhost/SabaShop/bookshop.php>

これで図1の検索指示画面が表示されます。後は画面の指示に従って動かして下さい。

## 5. おわりに

以上でSabaphyの一通りの紹介が終わりです。ここで図2の「Sabaphyアプリケーションの構造」を見直して下さい。そこでも述べたように、アプリケーション開発者が用意しなければならないものは、以下の4種のものです。

- 1) メインPHP (主として、EntityManagerやDispatcherに与えるアプリケーション設定情報)
- 2) ワークユニット (アクションメソッドの定義)
- 3) エンティティ (データ構造と導出属性・制約の定義など)
- 4) ビュー (出力用PHP)

これらのうち、メインPHPとビューはプログラムロジックをほとんど含んでいません。プログラムとして開発しなければならないものは、ワークユニットとエンティティだけといっても良いでしょう。

フレームワークやコンテナを利用することの利点は以下の3点です。これはSabaphyに限らず、フレームワークやコンテナ一般についていえることです。

- 1) 永続化、セッション管理などのシステムレベルのAPIの利用はフレームワークやコンテナに任せることができ、アプリケーション開発者は対象領域の問題に専念できる。
- 2) アプリケーションの構成要素の役割が明確で、見通しのよいアプリケーションを開発できる。個々の構成要素の再利用性も高くなる。
- 3) プログラムを必要とする部分とそうでない部分が明確に分かれるので、分業による開発が進めやすい。

このような利点がある反面、フレームワークやコンテナ固有の難しさがあるのも事実です。

- 1) フレームワーク利用の前提となっているオブジェクト指向が難しい。
- 2) アプリケーションの動作の基幹部分がフレームワークの中に隠蔽されているため、個々の構成要素の役割を理解することが難しい。
- 3) 将来のメンテナンスに不安があるオープンソースは使いたくないという心理的な抵抗感がある。

Sabaphyははじめから教育を念頭において開発されました。そのライブラリは、フレームワークを使わずに永続化・セッション管理の機能だけを使った開発や、オブジェクト指向を最小限に止めた開発なども可能なように設計されています。これらを利用して、段階的にフレームワーク概念を学習できる教材も用意しています。

これらは、プログラミングの初心者が最新のWebアプリケーションの諸概念を理解し、使いこなすことができるようにすることを目的とした教材です。現在、試験的にゼミの学生にこれで学んでもらっています。これが終わった後、EJBやStrutsの学習に進んでもらうことを予定しています。

また、ライブラリプログラムは、コンポーネントに慣れていない人でもソースコードが読みやすいように、敢えて細かくコンポーネントを分けずに構成にしています。このようなオープンソースを使ったシステム開発への抵抗感が少しでも少なくなることを期待しています。

最後にSabaphyを用いた実用システムの可能性に言及しておきましょう。Sabaphyは教

育を主目的にしていますが、本格的なRDBを併用すれば、特にアクセスが集中するようなサイトは別として、実用に耐えうるものだと思っています。ただわれわれは実務経験がありませんので、実用性について十分な評価はできません。今後、実務に携わられておられる方々のご意見・ご批判をいただきたいと思っています。

東京国際大学商学部情報システム学科 佐藤英人

e-mail: [sato@tiu.ac.jp](mailto:sato@tiu.ac.jp)

Sabaphyホームページ: <http://satolab.tiu.ac.jp/sabaphyHome/>