Xhwlay Tutorial

Version 1.1

author: msakamoto-sf@users.sourceforge.net

目次

2007/10/09 : Version1.0 created
2008/02/14 : Version1.1 created

0. 前書き

このチュートリアルは３つのパートで出来ています。

Part1: ログイン・ログアウトを実現する簡単なページフローを作成します。

Part2: 簡単なウィザード形式のページフローを作成します。

Part3: "Page-action mapping"型のページフローを作成します。また、ステートレスにする方法を紹介します。

このチュートリアルで示されているソースコードは、全て Xhwlay のリリースアーカイブの "sample" ディレクトリの中に含まれています。Part1 を開始する前に、以下のセクションを読んでチュートリアル用の環境を整えましょう。

0.A. Xhwlay のバージョン、およびブラウザの注意点

開発者(=msakamoto-sf)のミスにより、Xhwlay-0.9.0 の sample コードには Internet Explorer 上では正常に動作しない、"<button>"タグを含んでいました。0.9.0 の sample を動かす時は他のブラウザを使用して下さい。

0.9.1 以降ではこの問題は修正され、本ドキュメントも更新されています。(Part2 参照)

0.B. 対象読者とチュートリアルの準備

このチュートリアルでは、読者は Apache と PHP を用いた Web アプリケーション作成を有る程度独力でできるレベルを想定しています。

もし読者である貴方が、昨日今日始めたばかりで Apache の設定や php.ini の設定を自信を持って行えないようであれば、まずは基礎固めをしてから再度このチュートリアルを読んでください。

$ チュートリアル用の VirtualHost 設定

このチュートリアルでは作成する PHP スクリプトを "http://xhwlay-tutorial/" という URL からアクセスできるようにしています。Apache の VirtualHost 設定を以下のような形で追加します。

```
NameVirtualHost *:80
<VirtualHost *:80>
ServerName xhwlay-tutorial
DocumentRoot /your/xhwlay/tutorial/path
</VirtualHost>
```

※すでにお使いのサーバーでは PHP と Apache が適切に設定され、動いていることを前提とします。

Apache の VirtualHost 設定の詳細については次の URL を参照してください。
http://httpd.apache.org/docs/2.0/vhosts/

$ クライアント PC の host ファイル設定

host ファイルに "xhwlay-tutorial" のホスト設定を追加します。

```
192.168.0.1 xhwlay-tutorial   # 192.168.0.1 is your server's ip address.
```

host ファイルの主な置き場所は以下の通りです。

- Windows XP :
C:\WINDOWS\system32\drivers\etc\host

- UNIX, Linux :
/etc/host

自分で設定できない場合は、サーバの管理者に問い合わせてみてください。

$ Apache の設定を操作できない場合

以下のチュートリアルで "http://xhwlay-tutorial/" となっているところを、適宜読者の環境の URL に読み替えてください。

Apache の設定が完了したら、設定したチュートリアル用のディレクトリに適当な HTML 或いは PHP ファイルを作成し、ブラウザ上から見えることを確認して下さい。

1. Part 1

このパートではログイン・ログアウト機能を備えたページフローを作成することで、Xhwlay の基本的な使い方を見ていきます。完全なソースコードは、Xhwlay リリースアーカイブの"sample"ディレクトリの中に含まれています。

（注意：完全なソースコードは、パート 2 のウィザードアプリケーション(main.php)との連携コードが含まれて居ます）

1.A. ページフローの設計

最初に、これから作成するアプリケーションの**ページフロー**を作ります。

これまでの"Page-action mapping"型の PHP フレームワークでは、最初に考えるのはどの PHP がどのアクションを実行するか、だったと思います。

Xhwlay などのイベント駆動指向のステートフルなアプリケーションでは、最初にどの**ページ(State)**があり、どういう**イベント**によりどう遷移していくかを定義した**「ページフロー」**を考えます。Xhwlay における"ページ"は、ステートフルアプリケーションにおける"状態(State)"とほぼ同義です。

$ ページ(Page)

ログイン・ログアウトアプリケーションですので、以下の二つのページが考えられます。

- **"login" ページ** : "username"と"password"入力と submit ボタンを表示するページ

- **"logout" ページ** : ログアウトメッセージを表示するページで、ユーザーが"logout"をリクエストしたときに表示されるページ

"login"した後はどこへ遷移すればよいでしょうか？今回は"ログイン中"と表示するページを一枚、用意します。

- **"main"ページ** : "ログイン中"であることを表示するページ

次に、開始と終了ページを定義します。

Xhwlay では一つのページフローは一冊の本に喩えることができます。一つの本には、開始ページが一つ、そして終了ページが複数存在します。

ここでは"login"ページを開始ページとし、"logout"ページを終了ページとします。

```
 "login" page (start) ----> "main" page ----> "logout" page (end)
```

$ イベント(Event)

次に、ページを遷移させるための「**イベント**」を定義します。イベントとは、ページ遷移のトリガーとなるユーザーからのリクエストのことです。今回のアプリケーションでは、"login"イベントと"logout"イベントの二つがあることが明らかです。

- **login イベント** : "login"ページで login イベントが発生すると、username と password をチェックし、チェック OK であれば"main"ページに遷移します。"onLogin" イベントと名づけます。

- **logout イベント** : "main"ページで logout イベントが発生すると、"logout"ページへ遷移します。"onLogout" イベントと名づけます。

```
              "onLogin" event      "onLogout" event
                     |                  |
                     |                  |
                     V                  V
 "login" page (start) ---> "main" page ----> "logout" page (end)
```

$ ガード(Guard)

"onLogin"イベントでは、リクエストされた username と password が正しいことをチェックする必要があります。もし正しければ、アプリケーションは"login"ページから"main"ページへ遷移します。もし正しくなければ、"main"ページへは遷移せず、"login"ページへロールバックします。

Xhwlay や他の有限状態遷移マシンのコンセプトでは、ページ遷移の是非を決定するこのような機能を**"ガード(Guard)"**と呼びます。ガードはいわば、ページ遷移における「門番(Gate Keeper)」です。

今回は"onLogin"イベントで "validateLogin"というガードを定義します。

```
"login" page (start)
 ↓
("validateLogin" guard) ← "onLogin" event
 ↓
```

```
"main" page
  ↓← "onLogout" event
"logout" page (end)
```

これで、ログイン・ログアウトアプリケーションのページフローを作成できました。続いて、これを動かすための Xhwlay アプリケーションのスクリプトをざっと書いてみます。

1.B. Xhwlay アプリケーションの骨組み

では、Xhwlay アプリケーションのアウトラインを組み立ててみましょう。

最初に以下のようなスケルトンスクリプトを作ります。"http://xhwlay-tutorial/"でアクセスできるディレクトリに、"login.php"として保存してください。

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');


// {{{ requires
require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Serialize.php');
require_once('Xhwlay/Renderer/Include.php');
// }}}
// {{{ Bookmark Container and Page Flow (Story) Configurations
$bookmarkContainerParams = array(
    "dataDir" => $__base_dir.'/datas',
    "gc_probability" => 1,
    "gc_divisor" => 1,
    "gc_maxlifetime" => 30,
);
$configP = array(
    // Yet empty array.
```

```
    );

// }}}
// {{{ main scripts

$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);

$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);

echo $runner->run();
?>
```

細かいところを見ていきましょう。

requirements:

- Xhwlay/Runner.php : ページフローの実行エンジン本体

- Xhwlay/Bookmark/FileStoreContainer.php : Bookmark のファイルストア用エンジン (Bookmark については後述)

- Xhwlay/Config/PHPArray.php : ページフローを PHP の連想配列で定義するためのクラス

- Xhwlay/Renderer/Include.php : ファイルを include するだけの単純なレンダリングエンジン (後述)

settings:

- $bookmarkContainerParams : Bookmark コンテナの設定パラメータ(後述)

- $configP : ページフロー定義(後で中身を埋めます)

main scripts:

```
$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);
```

レンダラとコンフィギュレーションのインスタンスを作成します。今の時点では、コンフィギュレーションは空のページフロー設定で初期化されます。

```
$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);
```

Xhwlay_Runner のインスタンスを作成します。続いて、Bookmark コンテナの実装クラス名とそのパラメータを設定します(後述)。その後、レンダラとコンフィギュレーションのインスタンスを設定します。

```
echo $runner->run();
```

ページフローを実行し、出力データを受け取り echo して出力します。この時点では唯のスケルトンで、何も意味のある処理は行いません。

$ ページフローの埋め込み

では、ページフローをスクリプトに埋め込んでみましょう。

```
$configP = array(
    "story" => array(
        "name" => "Login Example",
        "bookmark" => "on",
        ),
    "page" => array(
        "logout" => array(
            "bookmark" => "last", // indicate this page is end page.
            ),
        "main" => array(
            "event" => array(
                // Acceptable Event list.
                // key is event name, and value is guard name.
                // If guard name is null, guard is not defined and not invoked.
                "onLogout" => null,
```

```
                ),
            ),
        // "*" means start page, equals, "login" page.
        "*" => array(
            "event" => array(
                "onLogin" => "validateLogin",
                ),
            ),
        ),
    "event" => array(
        "onLogout" => array(
            "transit" => array(
                // key is string which returned by event handler,
                // value is page name which defined above "page" configuration.
                "success" => "logout",
                ),
            ),
        "onLogin" => array(
            "transit" => array(
                "success" => "main",
                ),
            ),
        ),
    "guard" => array(
        "validateLogin" => array(
            ),
        ),
    );
```

Xhwlay のデフォルトのコンフィギュレーションエンジンは、PHP の連想配列を使います。もちろん、デフォルトを拡張して yaml や xml などの好みのフォーマットでページフローを定義できるようにすることも可能です。

これでページフローも定義できたように見えます。しかし、まだ実際の処理(Action)を定義していません。

$ Page Action のスケルトン

では、ページ、イベント、ガードの各 Action を定義していきましょう。

最初に"login"ページの Action を作ってみます。

```
function demo_page_login(&$runner, $page, &$bookmark, $params) {
}
```

中身は後で埋めていきますので、今はスケルトンにしておきます。作成した Action を実際のページ定義に結び付けるには、"login"ページの定義に"user_function"エントリを追加します。

```
"*" => array(
    "user_function" => "demo_page_login", // This line is added.
    "event" => array(
        "onLogin" => "validateLogin",
        ),
    ),
```

同様に"main"と"logout"ページのアクションを定義します。

```
function demo_page_main(&$runner, $page, &$bookmark, $params) {
}
function demo_page_logout(&$runner, $page, &$bookmark, $params)
}
```

ではここで、ページ Action に渡される引数を簡単に説明します。

- **&$runner** : Xhwlay_Runner インスタンスへの参照です。ここから、レンダラやコンフィギュレーションのインスタンスを取得できます。

- **$page** : 現在のページ名が文字列で渡されます。

- **&$bookmark** : 現在の Bookmark オブジェクトへの参照です。後述するように、このオブジェクトを使ってアプリケーション固有のデータをストアすることが出来ます。

- **$params** : mixed な配列です。例えば、"*" ページ Action は次のような形の配列で$params が設定されます。

```
array(
    "user_function" => "demo_page_login",
    "event" => array( "onLogin" => "validateLogin" )
)
```

これを使って、アプリケーション固有の設定をページ Action に渡すことも可能です。

$ Event Action のスケルトン

次にイベントの Action を定義してみます。もちろん、こちらもまだスケルトンです。

```
function demo_event_onLogin(&$runner, $event, &$bookmark, $params) {
    return "success";
}
function demo_event_onLogout(&$runner, $event, &$bookmark, $params) {
    return "success";
}
```

戻り値はページフローで定義された"transit"名になります。もし戻り値がページフローで定義されていない transit 名の場合は、Xhwlay はページ遷移を行いません。

では、イベント Action をページフローの定義に追加してみます。

```
"event" => array(
    "onLogout" => array(
        "user_function" => "demo_event_onLogout",
        ,,,
        ),
    "onLogin" => array(
        "user_function" => "demo_event_onLogin",
        ...
        ),
    ),
```

イベント Action に渡される引数を簡単に説明します。

- **&$runner** : Xhwlay_Runner インスタンスへの参照です。ここから、レンダラやコンフィギュレーションのインスタンスを取得できます。

- **$event** : 現在のイベント名が文字列で渡されます。

- **&$bookmark** : 現在の Bookmark オブジェクトへの参照です。後述するように、このオブジェクトを使ってアプリケーション固有のデータをストアすることが出来ます。

- **$params** : mixed な配列です。例えば、"onLogout" イベント Action は次のような形の配列

で$params が設定されます。

```
array(
   "user_function" => "demo_event_onLogout",
   "transit" => array( "success" => "logout" )
)
```

これを使って、アプリケーション固有の設定をイベント Action に渡すことも可能です。

$ Guard Action のスケルトン

続いてガード Action をスケルトンで定義します。

```
function demo_guard_validateLogin(&$runner, $event, &$bookmark, $params) {
    return true;
}
```

ガード Action が true を返すと、Xhwlay はイベント Action を実行します。もし false を返せば、イベント Action は実行されず、ページ遷移も発生しません。

では、ガード Action をページフローの定義に追加してみます。

```
"guard" => array(
    "validateLogin" => array(
        "user_function" => "demo_guard_validateLogin"
        ),
    ),
```

ガード Action に渡される引数を簡単に説明します。

- **&$runner** : Xhwlay_Runner インスタンスへの参照です。ここから、レンダラやコンフィギュレーションのインスタンスを取得できます。

- **$event** : 現在のイベント名が文字列で渡されます。

- **&$bookmark** : 現在の Bookmark オブジェクトへの参照です。後述するように、このオブジェクトを使ってアプリケーション固有のデータをストアすることが出来ます。

- **$params** : mixed な配列です。例えば、今回のガード Action は次のような形の配列で $params が設定されます。

```
array(
   "user_function" => "demo_guard_validateLogin"
```

```
)
```

これを使って、アプリケーション固有の設定をガードActionに渡すことも可能です。

$ "View Name"とスケルトンテンプレートの準備

これでActionは準備できました。次は、ブラウザに表示するHTMLを生成する**"View"**を準備します。**Xhwlayではページ Actionの戻り値を"View Name"と呼んでいます。**

"View Name"はレンダリングエンジンに渡されます。

"View Name"はレンダリングエンジンにとって、一種のリソース識別子として扱われます。リソースは、出力データを作成するための材料です。

"View Name"は例えばHTMLテンプレートファイルかもしれませんし、SkinやThemeを実装しているテーブルの主キーかもしれませんし、あるいは他の何かかもしれません。
"View Name"が何を表すのかは、レンダリングクラスの実装に依存します。

(注：Xhwlayが提供しているいくつかのレンダラでは、"View Name"は無視されます。"Serialize", "VarDump"レンダラが該当します。)

このチュートリアルでは、Xhwlay_Renderer_Includeをレンダラとして使用します。このクラスは、"View Name"で指定された HTML/PHPテキストファイルをincludeします。では、各"View Name"と対応する実際のファイルを用意してみましょう。

- "login" page : View Name は "templates/login_login.html" とします。
- "main" page : View Name は "templates/login_main.html" とします。
- "logout" page : View Name は "templates/login_logout.html" とします。

$$ ページActionの編集

ページActionに "return <View Name>;"の一行を追加します。Xhwlay_Renderer_Includeクラスでは、View Nameはアプリケーションのスクリプトに対する相対パスで指定します。

今回は"templates"ディレクトリを作成し、相対パスは"templates/..."となります。

```
.../login.php
       templates/
                 xxxx.html
```

では、次のように"return"文を各ページ Action に埋め込みます。

```
function demo_page_login(&$runner, $page, &$bookmark, $params) {
    return "templates/login_login.html";
}
function demo_page_main(&$runner, $page, &$bookmark, $params) {
    return "templates/login_main.html";
}
function demo_page_logout(&$runner, $page, &$bookmark, $params) {
    return "templates/login_logout.html";
}
```

$$ テンプレート HTML ファイルの準備

スクリプトと同じディレクトリで"templates"ディレクトリを作成し、3 つの HTML ファイルを作成します。完全な HTML ファイルとディレクトリ群は、Xhwlay アーカイブ中の"sample"ディレクトリを参照して下さい。

これで、最低限度のページフローの準備が整いました。ブラウザを起動し、以下の URL にアクセスしてみてください。

http://xhwlay-tutorial/login.php

"templates/login_login.html"が表示されるはずです。

ようやく Xhwlay の実行が確認できましたが、各 Action の実装は依然として不完全です。また、まだイベントを発生させることも出来ません。次のセクションでは、イベントを発生させるための処理を実装していきます。

1.C. Xhwlay アプリケーションの構築

最初に、「どうやって**"Bookmark Container ID"**をページ間で共有するか？」を決定する必要があります。その前に、Xhwlay の"Bookmark"と"Bookmark Container"の概念を簡単にまとめます。

$ ″Bookmark″

"Bookmark" は、まさしく現実世界の「しおり」(Bookmark)に喩えることができます。一つの Bookmark は一つの本と関連付けられます。Xhwlay ではこの本のことを **"Story"** と呼びます。"Story"はページフローと同義です。一つの Bookmark は、今自分がどこにいるのかをあらわすページ名を一つだけ保持します。オプションとして、一つの Bookmark にはアプリケーション固有のデータを書き込むことができます。

Xhwlay では**"Bookmark Container ID"**という識別子により、複数の Bookmark を識別しています。

$ ″Bookmark Container″と″Bookmark Container Id (BCID)″

最初に Xhwlay アプリケーションにアクセスするとき、Xhwlay は新しい"Bookmark Container ID(BCID)"を一つ、発行します。一つの BCID は一つの**"Bookmark Container"**と関連付けられます。一つの"Bookmark Container"は複数の Bookmark を保持し、管理します。

ここで Bookmark はページフローと結びついていることを思い出してください。

一つの BCID は、複数のページフローと関連付けされています。つまり、**一つの BCID により複数のページフローを同時に動かすことが出来るのです。**ただし重要な注意点があります。一つの BCID で、同じページフローを複数関連付けることは出来ません。BCID に紐付く Bookmark は、全て異なるページフローのものでなければなりません。

まとめて見ましょう。

・一つの Bookmark は、現在位置を示すページ名と、自由書き込みの出来るデータ領域を持ちます。

・一つの Bookmark Container は複数の Bookmark を保持します。（ただし、Bookmark は全て異なるページフローで無ければなりません。)

・一つの BCID は、一つの Bookmark Container に紐付きます。

では、どうやって BCID を保持し、また Xhwlay に BCID を知らせるのでしょうか？また、イベントが発生したことを Xhwlay に知らせるにはどうすればよいのでしょうか？

$ ″″HOOK″と″setup″, ″terminate″フック

「あなたのお好きなように」は Xhwlay の基本コンセプトの一つです。 Web アプリケーションを構築するための多くの機能が、Xhwlay にはありません。それは、まさにそれら

こそ、開発者が好きなように作るべきものだと考えているからです。

BCID をどうやって保持し、Xhwlay に伝えるか？イベントの発生をどうやって Xhwlay に伝えるか？Xhwlay はその基本コンセプトに従い、これらの疑問に答えるための典型的な実装は一切提供していません。なぜか？

Xhwlay と Xhwlay 開発者が、これらの実装方法のバリエーションを全て完全に想像することは不可能だからです。

URL クエリからくるのでしょうか？POST パラメータでしょうか？XML-RPC のどこかからでしょうか？あるいは、Xhwlay が想像も付かないようなフォーマットでくるのでしょうか？

Xhwlay の開発者は、具体的な実装は提供しないことにしました。というよりは、上記理由により提供できなかったのです。その代わり、ユーザーがこれらの機能を好きなように実装するための**"HOOK"**ポイントを提供することにしました。

この"HOOK"ポイントは **"setup"**と**"terminate"**フックと呼ばれます。

"HOOK"(他のライブラリやフレームワークでは"Plugin"などとも呼ばれる概念に相当)については、他のフレームワークに触ったことのある人であればそのコンセプトはご存知のことと思います。「百聞は一見にしかず」、Xhwlay の提供する HOOK 機能の使い方を見ていきましょう。login.php を下記のように修正します。

```
....
$config =& new Xhwlay_Config_PHPArray($configP);
// append start
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");

$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");
// append end
$runner =& new Xhwlay_Runner();
...
// add this function
function demo_hook_setup_session($hook, &$runner) {
    session_start(); // starts the session.

    $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
    $event = isset($_REQUEST['_event_']) ? $_REQUEST['_event_'] : "";
```

```
    Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
    Xhwlay_Var::set(XHWLAY_VAR_KEY_EVENT, $event);
}
function demo_hook_terminate($hook, &$runner) {
    $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
    $sid = session_id();
    if (!empty($sid)) {
        $_SESSION['bcid'] = $bcid;
    }
}
```

(Xhwlay_Var は static な getter/setter を備えたクラスで、Xhwlay の内部ではグローバル変数のストックとして使われています。)

Xhwlay_Hook::getInstance() により、引数で指定された HOOK のインスタンスを取得できます。HOOK インスタンスは、pushCallback() された callback のスタックを保持しています。今回は$h1 と$h2 にそれぞれ setup と terminate フックのインスタンスを受け取り、pushCallback()で関数名をスタックに push しています。

"setup"フックは、Xhwlay 内部のメイン処理が走る前に必ず実行される HOOK ポイントです。Xhwlay のメイン処理が開始されると、Xhwlay は BCID やイベント名（あれば）を Xhwlay_Var より取り出します。よって、Xhwlay にイベント発生や BCID を伝えるには、"setup"フック内で Xhwlay_Var にしかるべき値をセットすればよいことになります。

"terminate"フックは、Xhwlay 内部のメイン処理が終わった後に必ず実行される HOOK ポイントです。この時点で、Xhwlay_Var の XHWLAY_VAR_KEY_BCID に BCID がセットされています。よって、アプリ側で BCID を確実に取り出すには、"terminate"フックを利用できます。

上記の実装では、まず terminate フックで BCID を$_SESSION に保存しています。これは毎回行われ、また、毎回行われても特に支障の無い処理です。$_SESSION に保存されている BCID を取り出し、Xhwlay_Var にセットする処理は setup フックで実装されています。また、"setup"フックでは GET または POST のリクエスト値から、"_event_"パラメータを取得し、その値を発生したイベント名として Xhwlay_Var にセットし、Xhwlay に伝えています。

では、テンプレート HTML を少し修正し、現在の BCID を表示できるようにしてみましょう。とりあえず、下記のようにテンプレート HTML を直してみてください。

```
templates/login_login.html
...
<ul>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
</ul>
...
```

さて、どうやって BCID を HTML に埋め込めばよいでしょうか？
Xhwlay_Renderer_Include(および他の)レンダラは、View に変数を埋め込むための setter メソッドを提供しています。

```
Xhwlay_Renderer_Include::set("name", $var);
```

これは static メソッドではありませんので、かならずレンダラのインスタンスを取得して呼び出す必要があります。Xhwlay_Renderer_Include では、set した値は $GLOBALS['template']["name"] としてテンプレート HTML 内で参照することが可能です。

では、どこで set()メソッドを呼ぶべきでしょうか？つまり、どこでレンダラのインスタンスを取得できるのでしょうか？イベント、ページ、ガード Action、つまり Xhwlay_Renderer へのインスタンスを取得できるところであれば可能です。レンダラのインスタンスは、Xhwlay_Runner のインスタンス(&$runner)を用いて次のように取得できます。

```
$renderer =& $runner->getRenderer();
```

今回のチュートリアルでは、これをページ Action 内で使用してみます。demo_page_login() を次のように編集します。

```
function demo_page_login(&$runner, &$bookmark, $event, $params) {
    $renderer = $runner->getRenderer();
    $renderer->set('BCID', Xhwlay_Var::get(BCID));
    ...
}
```

これで、HTML テンプレート内で $GLOBALS['template']['bcid'] により BCID を表示できるようになりました。Xhwlay_Renderer_Include レンダリングエンジンでは、set した値は HTML テンプレート内で$GLOBALS['template']により参照することが可能です。今回の HTML テンプレートとページ Action の変更を、他の二つの HTML テンプレートとページ Action にも適用しておきましょう。

1.D. ログイン・ログアウトとその他の機能の実装

最後に、いよいよ実際のログイン・ログアウトの機構とその他おまけ機能を実装してみます。

$ ログイン・ログアウト

最初にログイン・ログアウトを作りこみましょう。

$$ "demo_guard_validateLogin" ガード Action の実装

最初に "demo_guard_validateLogin" ガード Action の中身を埋めます。今まではスケルトンコードでしたが、実際の認証処理を書いて見ます。以下に "demo_guard_validateLogin()" の完全なソースコードを示します。

```
function demo_guard_validateLogin(&$runner, $event, &$bookmark, $params) {
    $_user_name = @$_REQUEST['user_name'];
    $_password = @$_REQUEST['password'];
    if (empty($_user_name) || empty($_password)) {
        return false;
    }
    // Store into Bookmark user data area.
    $bookmark->set("user_name", $_user_name);
    $bookmark->set("password", $_password);

    $sid_old = session_id(); // save old sid.
    session_regenerate_id(); // generate new sid.
    $sid_new = session_id(); // save it.
    session_id($sid_old);    // now, set current as saved old sid.
    session_destroy();       // destroy current (equals old sid).
    session_id($sid_new);    // re-set current as new sid.
    session_start();         // re-start session.

    return true;
```

```
}
```

細かく見ていきましょう。最初に、$_REQUEST からユーザー名とパスワードを取得しています。もしユーザー名またはパスワードが empty()なら、false を返します。ガード Action で false が返されるため、onLogin イベントは実行されず、依然として"login"ページのままです。実際の認証では、この条件判断の中身が DB アクセスや LDAP アクセスなどの処理になるはずです。

認証が通った後は、一旦ユーザー名とパスワードを Bookmark に保存しています。これは"onLogin"イベント内で参照されたのち、クリアされます。また、SessionID 固定化攻撃を避けるため、セッション ID をリセットする処理を入れています。

それが終われば true を返します。ガード Action が true を返すと、Xhwlay は続けて"onLogin"イベント Action を実行します。

$$ ″demo_event_onLogin″ イベント Action の実装

では、"demo_event_onLogin"イベント Action の実際の処理を作りこんで見ます。以下に完全な"demo_event_onLogin"イベント Action のソースコードを示します。

```
function demo_event_onLogin(&$runner, $event, &$bookmark, $params) {
    // These are stored in demo_guard_validateLogin().
    $user_name = $bookmark->get("user_name");
    $password =$bookmark->get("password");

    // demo codes.
    $user_id = md5( $user_name . $password );
    $bookmark->remove("user_name");
    $bookmark->remove("password");
    // stores user_id into a session variable.
    $_SESSION['user_id'] = $user_id;
    // demo counter.
    $_SESSION['count'] = 0;

    return "success";
}
```

最初に、Bookmark からユーザー名とパスワードを取り出しています。その二つの文字列

をつなげた値の MD5 チェックサム値を、仮のユーザー ID としてセッションに登録しています。使い終わったユーザー名とパスワードは、Bookmark から削除しています。

また、$_SESSION['count']という値を 0 クリアしています。これは後ほどのおまけ機能で使います。

$$ "demo_event_onLogout" イベント Action の実装

最後に、"onLogout"イベント Action の中身を作ります。以下が完全なソースコードになります。

```
function demo_event_onLogout(&$runner, $event, &$bookmark, $params) {
    session_destroy();
    return "success";
}
```

ここでは単純にセッション変数を破棄しています。これにより$_SESSION['user_id']が破棄され、ログアウト状態になります。

$ おまけ機能の実装

最後に、おまけ機能を実装してみます。

$$ demo_page_main()

"main"ページ Actin では、セッション変数を用いた簡単なリロードカウンタを表示させて見ます。以下に "demo_page_main" ページ Action の完全なコードを示します。

```
function demo_page_main(&$runner, $page, &$bookmark, $params) {
    $user_id = $_SESSION['user_id'];
    // count up demo.
    $count = $_SESSION['count'];
    $count++;
    $_SESSION['count'] = $count;

    $renderer =& $runner->getRenderer();
```

```
    $renderer->set('page', 'main');
    $renderer->set('user_id', $user_id);
    $renderer->set('count', $count);
    $renderer->set('bcid', $bookmark->getContainerId());
    return "templates/login_main.html";
}
```

$_SESSION から"user_id"を受け取り、レンダラにセットしています。また、$_SESSION から"count"を受け取り、インクリメントして$_SESSION を更新した後、レンダラにセットしています。

$$ templates/login_main.html

最後に、main ページにセッションのカウンタ値を表示させて見ましょう。以下に templates/login_main.html の完全なコードを示します。

```
<html>
<body>
<ul>
<li>Page Position : <?php echo $GLOBALS['template']['page']; ?></li>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
<li>Session Count : <?php echo $GLOBALS['template']['count']; ?></li>
<li>User ID : <?php echo $GLOBALS['template']['user_id']; ?></li>
</ul>

<hr>
<h3><a href="main.php">Go to Wizard Example.</a></h3>
<hr>
<form action="" method="POST">
<button type="submit" name="_event_" value="onLogout">Logout</button>
</form>
<a href="?_event_=onLogout">Logout</a>

</body>
```

```
</html>
```

$ アクセステスト

これでログイン・ログアウトを動かす準備は整いました。http://xhwlay-tutorial/login.php にアクセスしてみてください。

ページフローが開始され、スタートポイントである"login"ページが表示されているはずです。ユーザー名とパスワードに何か適当な文字列を設定し、"Login"ブタンをクリックします。すると、"login"イベントが発生し、まず "demo_guard_validateLogin" ガード Action が呼ばれ、続いて "demo_event_onLogin" イベントアクションが実行されます。ユーザー名とパスワードから MD5 チェックサム文字列が生成され user_id として保持され、レンダラに設定され、main.html が表示されます。

ここでブラウザの「更新」ブタンをクリックしてみます。「再度 POST しますがよろしいですか？」といった内容のメッセージボックスが出た場合は、"OK"をクリックします。すると、相変わらず(当然といえば当然ですが)"main"ページにいて、main.html が表示されています。セッションカウンタが、リロードするたびにカウントアップされていくのが確認できると思います。

今度は"logout"ボタンをクリックしてみます。"logout"の HTML 画面が表示されます。その後リロードすると、"login"ページが表示されます。ログアウトされたとき、終端ページに達したので Bookmark が破壊されます。ここでリロードすると、Xhwlay は新しい Bookmark を作り、開始ページの Action を実行するためです。

1.E. 次のチュートリアル

以上で Xhwlay アプリケーションの作り方の基礎を習得できました。続いて、より実際的であろうと思われる、5 つのページを持つ簡単なウィザード形式のアプリケーションを作成してみます。

2. Part 2

このパートではXhwlayを用いた簡単なウィザード形式のアプリケーションを作成してみます。完全なソースコードは、Xhwlayリリースアーカイブの"sample"ディレクトリの中に含まれています。

2.A. ページフローの設計

このチュートリアルでは、5つのページをもつ簡単なウィザード形式のアプリケーションを作成してみます。ウィザードは典型的な申し込みフォームを提供します。名前、メールアドレス、郵便番号、住所、電話番号、年齢、趣味をユーザーは入力します。これらの入力フォームは複数のページに分割されます。このアプリケーションは、ユーザーの入力した全データを表示する確認ページと、アプリケーションとして申し込み完了後に「何か」した後の画面（例えばメール送信やDB登録がありうるでしょう。今回はそこまでは実装しません）の2つのページを含みます。

ざっくりと次のようにページを用意することにしましょう。

- page0 : 最初の画面で、名前とメールアドレスの入力フォームを表示します。

- page1 : 郵便番号、住所、電話番号の入力フォームを表示します。

- page2 : 年齢と趣味の入力フォームを表示します。

- page3 : ユーザーの入力した全データの確認ページです。

- page4 : 終了ページです。「何か」し終わったあとの表示画面になります。

page 0 は、page 1 へ進むためのボタンを一つ持ちます。page 1 - 3 は、「前へ」と「次へ」の2つのボタンを持ちます。page 4 は最初のページ、page 0 へ進むボタンを一つ持ちます。

続いて、このページフローがどのようなイベントを必要としているのか考えます。結果から言うと、全てのイベントは一つのイベントに集約できます。

page0 → 1, page 1 → 2, page 2 → 3 へ進むイベントは、全て、リクエストにある入力フォームの値をブックマークに保存してtransitしているだけになります。page3 → page4 ですが、今回はデモ用のサンプルですので、（メール送信やDB登録などの)具体的な「アクション」を実装する必要は有りません。そのおかげで、他のイベントをそのまま流用することが出来ます。

今回はガードは実装しません。Xhwlayを用いたウィザードアプリケーションのエッセン

スとしては不要だからです。

では、View 名を決めてしまいましょう。

- page0 : templates/main_page0.html
- page1 : templates/main_page1.html
- page2 : templates/main_page2.html
- page3 : templates/main_page3.html
- page4 : templates/main_page4.html

HTML の内容は part1 のものと似ていますが、若干の差異があります。差異については後続のセクションで解説します。とはいえ、全ての詳細を載せることはこのセクションの本質とは関係ないため、今回はこれらの HTML の説明はスキップさせてください。Xhwlay のアーカイブに含まれている実際のコードを参照してください。

2.B. Xhwlay アプリケーションの骨組み

これまでのことを踏まえて、下記のようなコードを組んでみます。 : main.php

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');

require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Include.php');

$bookmarkContainerParams = array(
    "dataDir" => $__base_dir.'/datas',
    "gc_probability" => 1,
    "gc_divisor" => 1,
    "gc_maxlifetime" => 30,
);
```

```
$configP = array(
    "story" => array(
        "name" => "Wizard Example",
        "bookmark" => "on",
        ),
    "page" => array(
        "page4" => array(
            "user_function" => "demo_page_page4",
            "bookmark" => "last",
            ),
        "page3" => array(
            "user_function" => "demo_page_page3",
            "event" => array(
                "onSubmitPage4" => null,
                "onBacktoPage2" => null,
                ),
            ),
        "page2" => array(
            "user_function" => "demo_page_page2",
            "event" => array(
                "onSubmitPage3" => null,
                "onBacktoPage1" => null,
                ),
            ),
        "page1" => array(
            "user_function" => "demo_page_page1",
            "event" => array(
                "onSubmitPage2" => null,
                "onBacktoPage0" => null,
                ),
            ),
        "*" => array(
            "user_function" => "demo_page_page0",
```

```
            "event" => array(
                "onSubmitPage1" => null,
                ),
            ),
        ),
    "event" => array(
        // go to next page
        "onSubmitPage1" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page1")),
        "onSubmitPage2" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page2")),
        "onSubmitPage3" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page3")),
        "onSubmitPage4" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page4")),

        // back to previous page
        "onBacktoPage2" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page2")),
        "onBacktoPage1" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "page1")),
        "onBacktoPage0" => array(
            "user_function" => "demo_event_onSubmit",
            "transit" => array("success" => "*")),
        ),
    );

$renderer =& new Xhwlay_Renderer_Include();
```

```
$config =& new Xhwlay_Config_PHPArray($configP);

// setup "setup" hooks (executed before Xhwlay)
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");

// setup "terminate" hooks (executed after Xhwlay)
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");

$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);

echo $runner->run();

function demo_hook_setup_session($hook, &$runner) {
    session_start();

    $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
    $event = isset($_REQUEST['_event_']) ? $_REQUEST['_event_'] : "";

    Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
    Xhwlay_Var::set(XHWLAY_VAR_KEY_EVENT, $event);
}

function demo_hook_terminate($hook, &$runner) {
    $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
    $sid = session_id();
    if (!empty($sid)) {
        $_SESSION['bcid'] = $bcid;
    }
```

```
}

function demo_page_page4(&$runner, $page, &$bookmark, $params) {
    return "templates/main_page4.html";
}

function demo_page_page3(&$runner, $page, &$bookmark, $params) {
    return "templates/main_page3.html";
}

function demo_page_page2(&$runner, $page, &$bookmark, $params) {
    return "templates/main_page2.html";
}

function demo_page_page1(&$runner, $page, &$bookmark, $params) {
    return "templates/main_page1.html";
}

function demo_page_page0(&$runner, $page, &$bookmark, $params) {
    return "templates/main_page0.html";
}

function demo_event_onSubmit(&$runner, $event, &$bookmark, $params)
{
    $vars = array(
        "name", "email", // input at "*"
        "zip", "address", "telephone", // input at "page1"
        "age", "hobby", // input at "page2"
        );

    foreach ($vars as $_k) {
        if (isset($_REQUEST[$_k])) {
            $bookmark->set($_k, $_REQUEST[$_k]);
        }
```

```
    }
    return "success";
}
?>
```

コードの大部分は part1 の login.php からの流用ですが、demo_hook_setup_session()関数が少し変更されています。まずイベントアクションについて細かく見た後、demo_hook_setup_session()関数の変更箇所とそれに連動したテンプレート HTML の変更箇所について見てみます。

$ ウィザードアプリケーションのサンプルのイベントアクション

今回のページフローでは、全てのイベントアクションは**"demo_event_onSubmit"**を共用しています。この中では想定される全ての入力フォーム名を"$vars"という配列で保持しており、$_REQUEST 変数に対してループ処理し、値があれば取り出し、ブックマークに保存しています。

またこのイベントは"success"のみを返します。実際の transit 先は、ページフローで設定されます。例えば、page1 は"onSubmitPage2"と"onBacktoPage0"のイベントを受け付けます：

```
"page1" => array(
    "user_function" => "demo_page_page1",
    "event" => array(
        "onSubmitPage2" => null,
        "onBacktoPage0" => null,
        ),
    ),
```

"onSubmitPage2"イベントの定義を見てみると、"success"は"page2"へマッピングされます。

```
"onSubmitPage2" => array(
    "user_function" => "demo_event_onSubmit",
    "transit" => array("success" => "page2")),
```

"onBacktoPage0"イベントであれば、"success"は page0 へマップされます。これにより、Xhwlay は「前へ」ボタンが押されたら page0 に transit します。

```
"onBacktoPage0" => array(
```

```
    "user_function" => "demo_event_onSubmit",
    "transit" => array("success" => "*")),
```

$ demo_hook_setup_session()関数とテンプレートHTMLの若干の差異について

Part 1 と Part 2 の demo_hook_setup_session()をもう一度見比べてみます。

Part 1(login.php)

```
function demo_hook_setup_session($hook, &$runner)
{
    ...
    // get Event from request parameters.
    $event = isset($_REQUEST['_event_']) ? $_REQUEST['_event_'] : "";
    ...
}
```

Part 2(main.php)

```
function demo_hook_setup_session($hook, &$runner)
{
    ...
    // get Event from request parameters.
    $event = '';
    foreach ($_REQUEST as $_k => $_v) {
        if (preg_match('/^_event_(¥w+)$/', $_k, $m)) {
            $event = $m[1];
        }
    }
    ...
}
```

なぜこのような変更が Part2 の main.php では必要なのでしょうか？それは、Part2 の主要画面では「前の画面へ」「次の画面へ」遷移する為に二つのボタンが必要になり、そのため二つ以上のボタン(<input type="submit">)の name 属性に"_event_[イベント名]"をセットし、リクエスト値を元に区別する必要があったからです。実際のテンプレートファイルでは次のようになっています。

templates/main_page1.html

```
...
<input type="submit" name="_event_onBacktoPage0" value="&lt;&lt; Page 0" />
<input type="submit" name="_event_onSubmitPage2" value="&gt;&gt; Page 2" />
...
```

これにより、リクエストのキー値をループさせ、"_event_[イベント名]"のパターンにマッチしたものがあればイベント名を取り出して内部にセットするようになります。

Xhwlay-0.9.0 の sample と本ドキュメントでは"<button>"タグを用いていました。しかしこれは開発者(=msakamoto-sf)のミスです。msakamoto-sf は複数の"<button>"タグが Internet Explorer 上で正しく動かないという事実について無知でした。これにより 0.9.0 の sample コードは Internet Explorer 上では正しく動作しません。

この問題は上記変更により、Xhwlay-0.9.1 以降で修正されています。

2.C. Xhwlay アプリケーションの構築

では、ソースコードの残りの部分を実装しましょう。

$ ページアクションの実装

このウィザードの例では、ページアクションは簡単です。単にブックマークに保存されているユーザー入力データを、レンダラにセットするだけです。たとえば page0 のページアクションである demo_page_page0()は次のようになります：

```
function demo_page_page0(&$runner, $page, &$bookmark, $params) {
    $renderer =& $runner->getRenderer();
    $renderer->set('f_name', $bookmark->get('name'));
    $renderer->set('f_email', $bookmark->get('email'));
    return "templates/main_page0.html";
}
```

他のページ(page1 - 3)のアクションもこれと同じようになります。実際のコードは Xhwlay アーカイブに含まれているソースコードを参照してください。

$ 再POST確認メッセージBOXの対処

これで一通りのWizardとして動かせるようになりました。しかし、弄っている内にたまたま"更新"ボタンをクリックすると、いわゆる"再POST確認メッセージBOX"が表示されることでしょう。ここではLocationヘッダーをブラウザに送出することにより、このメッセージボックスを表示させなくしてみます。

demo_event_onSubmit() メソッドを次のように編集してみます。

```
function demo_event_onSubmit(&$runner, $event, &$bookmark, $params) {
    $vars = array(
        ...
        );
    foreach ($vars as $_k) {
        ...
    }
    // append start
    header("Location: http://xhwlay-tutorial/main.php");
    $runner->wipeout();
    // append end
    return "success";
}
```

wipeout()メソッドは、ページアクションおよびViewのレンダリングを強制的にスキップさせます。Xhwlay自身はHTTPのボディエンティティを出力しなくなります。これにより、ブラウザはLocationヘッダによりリダイレクトされ、main.phpへGETメソッドでアクセスします。

動かしてみましょう。

- page 0 → page1 : いいですね。

- page 1 → page2 : オッケー。

- page 2 → page3 : オッケー。

- page 3 → page0 : ?? page0が表示されてしまいます。page4はどこへ行ってしまったのでしょうか？

page4はStoryの終了ページであることを思い出してください。つまり、"onSubmitPage4"イベント発生によりブックマークはクリアされてしまいます。ここでLocationヘッダーが

ブラウザに送信され、ブラウザがもう一度リクエストを投げます。すると、新しいブックマークが作成され、そのため、page0 が表示されるという仕掛けです。page4 を表示するにはどうすればよいでしょうか？

いくつか解決策はあると思いますが、今回は page3 の時だけ Location ヘッダーを送出しないようにしてみます。

```
"onSubmitPage4" => array(
    "user_function" => "demo_event_onSubmit",
    "transit" => array("success" => "page4")),
→
"onSubmitPage4" => array(
    "user_function" => "demo_event_onSubmit",
    "send_location_header" => false,
    "transit" => array("success" => "page4")),
```

demo_event_onSubmit() を次のように編集します。

```
header("Location: http://xhwlay-tutorial/main.php");
$runner->wipeout();
→
if (!isset($params['send_location_header']) ||
    $params['send_location_header'] == true) {
    header("Location: http://xhwlay-tutorial/main.php");
    $runner->wipeout();
}
```

"onSubmitPage4"が実行されると、$params 引数にはページフローで設定したとおり"send_location_header"エントリが false 値でもって設定されています。このため、上記 if 文の条件は成立しなくなるため、header()と wipeout()はコールされず、page4 が表示されます。他のイベントが発生した場合は、$params には"sent_location_header"エントリは設定されないため、if 文の条件式が成立し、header()と wipeout()が呼ばれ GET メソッドリクエストが発生します。

2.D. おまけ：login/logout との組み合わせ

では、おまけとして login.php と main.php を組み合わせてみましょう。main.php に login/logout 機能をつけたし、もし$_SESSION の user_id が未設定であれば login.php へリダ

イレクトするようにしてみます。

以下の"demo_hook_setup_auth()"関数を main.php に追加します：

```
function demo_hook_setup_auth($hook, &$runner) {
    if (!isset($_SESSION['user_id'])) {
        // If not logined yet, send "Location" header and ...
        header("Location: http://xhwlay-tutorial/login.php");
        // and, restrain continuous page action invoking, terminate.
        $runner->wipeout();
    }
}
```

これは HOOK です。どこで呼んで欲しいかというと、やはりセッションが開始された後でしょう。つまり、"setup"フックに追加すればよいということになります。

```
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");
$h1->pushCallback("demo_hook_setup_auth"); // add this line
```

これで、ログイン・ログアウト機能を実装できました。

2.E. 次のチュートリアル

これまでのチュートリアルの内容で、自分自身の Xhwlay アプリケーションを構築できると思います。しかし、説明していない Tips や Xhwlay の機能もまだまだあります。例えばアクションとしてクラスのメソッドを呼ぶにはどうすればよいでしょうか？また、ストーリー設定の"bookmark"エントリに"false"を設定すると何が起こるでしょうか？

Part3 ではこれらのトピックを踏まえつつ、Xhwlay が"action-page mapping"という旧来のページコントローラとしても使えることを示すデモンストレーションを紹介したいと思います。

3. Part 3

このパートではXhwlayの残された機能について解説します。

Xhwlayはイベントドリブン指向のステートフルページフローだけでなく、古典的な"action-page mapping"ページコントローラの機能も提供しています。ただし、Xhwlayのメインターゲットは空くまでもイベントドリブンなページフローアプリケーションの構築であることは忘れないでいて下さい。このチュートリアルで解説されている"Action-page mapping"機能は、Xhwlayにとってはあくまでもオプショナルなものです。完全なソースコード一式は、Xhwlayアーカイブの"sample"ディレクトリを参照してください。

3.A. 典型的な″Action-page mapping″アプリケーションのメカニズム

最初にPHPにおける典型的な "Action-page mapping" アプリケーションのメカニズムについてまとめます。

本質は簡潔で、

1. 一つのアプリケーションは一つのエントリポイントとなるPHPファイルを有します。(index.phpなど)

2. ユーザーは"action"パラメータを付けて上記エントリポイントをリクエストします。

3. リクエストされたエントリポイントは制御ロジックを実行し、"action"に関連付けられた実際のクラスの"mapping"を取得します。

4. 制御層は実際のクラスをロードし、制御層により決定されたメソッド名を実行します。

5. 続いて制御層は"View"コンポーネントを実行します。(プレーンなHTML/PHPファイルや、Smartyなどのテンプレートエンジン)

6. ユーザーは制御層よりレスポンスを取得します。

"Action-page mapping"のメインとなる概念は、"action"と実際のクラスを紐付ける"mapping"にあります。どのアクションが実行されるのかは、ユーザーのリクエストに依存します。このため、このメカニズムは"ステートレス"とも呼ばれます。

Xhwlayはこの点についてユニークな機能を提供しています。"Bookmark"です。Xhwlayは"今どのページに居るのか？"をBookmarkに保存し、ロードできます。Xhwlayにより、開発者は本来ステートレスであるところの"Action-page mapping"方式を"ステートフル"にすることが可能となります。もちろん、後述するようにXhwlayで"ステートレス"なアプリケーションを構築することも可能です。

3.B. ページフローの設計

ページフローについてはチュートリアル 1/2 と同じような方法で設計できます。いくつかの相違点があります：

・"Event"アクションについては考慮する必要はありません。

・"Guard"アクションの代わりに"Barrier"アクションを定義します。

・現在ページから遷移可能なページを制限できます。

では、今回のチュートリアルのページフローを具体的に設計していきましょう。まず、次のような 3 つのページを作ってみました。

- default : 初期ページです。このページからは、page1 へしか遷移できません。

- page1 : このページからは、page0 と page2 にのみ遷移できます。page2 へ遷移するときに"Barrier"アクションが実行されます。"barrier"というリクエストパラメータの値が"pass"であることをチェックします。

- page2 : このページからは、page1 または pge3 にのみ遷移できます。

- page3 : 終了ページです。

デモを簡単にするため、今回はそれぞれのページアクションでは特別な処理は行わせません。各ページアクションは現在のページ名をレンダラにセットするだけとします。

また、それぞれのページのテンプレートファイルも、現在のページ名とその他いくつかの情報を表示するだけの 1 つのみを共用することとします。

では、View 名を決定します。

- 全ページ共通 : templates/pages_default.html

ごめんなさい、今回はこれらの HTML の説明はスキップさせてください。HTML の内容は part1 のものと似ており、また、その詳細を載せることはこのセクションの本質とは関係ないからです。Xhwlay のアーカイブに含まれている実際のコードを参照してください。

3.C. Xhwlay アプリケーションの骨組み

では、"action-page mapping"ページフローはどのように定義すればよいのでしょうか？以下に、今回のデモ用のコードを示します。 : pages.php.

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');

require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Include.php');

$bookmarkContainerParams = array(
    "dataDir" => $__base_dir.'/datas',
    "gc_probability" => 1,
    "gc_divisor" => 1,
    "gc_maxlifetime" => 30,
);

$configP = array(
    "story" => array(
        "name" => "Page Oriented Example",
        "bookmark" => "on",
        ),
    "page" => array(
        "page3" => array(
           "user_function" => "demo_page_userfunc",
            "bookmark" => "last",
            ),
        "page2" => array(
           "user_function" => "demo_page_userfunc",
            "next" => array(
                "page3" => null,
                "page1" => null,
                ),
```

```
            ),
        "page1" => array(
           "user_function" => "demo_page_userfunc",
            "next" => array(
                "page2" => "barrier_sample",
                "page0" => null,
                ),
            ),
        "*" => array(
           "user_function" => "demo_page_userfunc",
            "next" => array(
                "page1" => null,
                ),
            ),
        ),
    "barrier" => array(
        "barrier_sample" => array(
            "user_function" => "demo_barrier",
            ),
        ),
    );

$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);

// setup "setup" hooks (executed before Xhwlay)
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");

// setup "terminate" hooks (executed after Xhwlay)
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");

$runner =& new Xhwlay_Runner();
```

```
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);

echo $runner->run();

function demo_hook_setup_session($hook, &$runner) {
    session_start();

    // get BCID from session variables.
    $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
    // get Page from request parameters.
    $page = isset($_REQUEST['_page_']) ? $_REQUEST['_page_'] : "";

    Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
    Xhwlay_Var::set(XHWLAY_VAR_KEY_PAGE, $page);
}

function demo_hook_terminate($hook, &$runner) {
    $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
    $sid = session_id();
    if (!empty($sid)) {
        $_SESSION['bcid'] = $bcid;
    }
}

function demo_page_userfunc(&$runner, $page, &$bookmark, $params) {
    $renderer =& $runner->getRenderer();
    $renderer->set('page', $page);
    $config =& $runner->getConfig();
    $bm = $config->needsBookmark() ? "on" : "off";
    $renderer->set('bookmark', $bm);
    if ($config->needsBookmark()) {
```

```
        $renderer->set('bcid', $bookmark->getContainerId());
    } else {
        $renderer->set('bcid', '(none)');
    }

    return "templates/pages_default.html";
}

/**
 * Barrier Example
 */
function demo_barrier(&$runner, $current, $next, &$bookmark, $params) {
    return isset($_REQUEST['barrier']) && $_REQUEST['barrier'] == "pass";
}
?>
```

"demo_hook_terminate" はチュートリアル 1/2 と同じです。"demo_hook_setup_session" は違います。チュートリアル 1/2 では、"demo_hook_setup_session" は "_event_" 変数を操作していましたが、このチュートリアルでは"_page_"変数になっています。

"demo_page_userfunc" は、ブックマークの基本情報をレンダラにセットしています。

template/pages_default.html の中身は以下のようになります :

```
<html>
<body>
<h1>Page oriented flow example</h1>
<ul>
<li>Current Page : <?php echo $GLOBALS['template']['page']; ?></li>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
<li>Bookmark Mode : <?php echo $GLOBALS['template']['bookmark']; ?></li>
</ul>
<ul>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>">current</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page0">default</a></
li>
```

```
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page1">page
1</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=block">page 2 (Barrier Blocked ... finally, page
1)</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=pass">page 2 (Barrier Pass.)</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page3">page
3</a></li>
</ul>
</body>
</html>
```

いろいろクリックして遊べるようにしてみました。

$ "Barrier" アクション

"demo_barrier" は**"Barrier"**用のコールバック関数です。"Barrier"アクションは新しいページがリクエストされたときに実行されます。

例えば、現在ページが page1 だとしましょう。リクエストされた新しいページが、今と同じ page1 の場合は"Barrier"アクションは実行されません。単に page1 のページアクションが実行されます。リクエストされた新しいページが page2 のとき、Xhwlay は page1 と page2 の間で"Barrier"が定義されていないかチェックします。今回の例ですと"barrier_sample"が定義されていて、そのコールバック関数は"demo_barrier"と設定されています。

```
"page1" => array(
    "next" => array(
        "page2" => "barrier_sample",
        "page0" => null,
        ),
    ),
    ...
barrier" => array(
"barrier_sample" => array(
    "user_function" => "demo_barrier",
    ),
```

```
),
```

ここでXhwlayはバリアアクションを実行、つまり"demo_barrier"関数を呼びます。もし戻り値がtrueであれば、ブックマークの現在ページ名は"page2"へ更新され、page2のページアクションが実行されます。

このように、"Barrier"は"Guard"と良く似ています。

pages_default.htmlの中身をもう一度見てみましょう。次のようなリンクがあります：

```
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=block">page 2 (Barrier Blocked ... finally, page
1)</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=pass">page 2 (Barrier Pass.)</a></li>
```

現在ページがpage1の時に、*上の方のリンク*をクリックすると "barrier" リクエストパラメータは"pass"でない、つまり、"demo_barrier"が falseを返します。これにより、Xhwlayは page2に遷移せず、page1のページアクションを実行します。

*下部のリンク*をクリックすると、"barrier"リクエストパラメータには"pass"が入りますので、"demo_barrier"は trueを返します。これによりXhwlayは現在ページ名をpage2に更新し、page2のページアクションを実行します。

3.D. "Bookmark-OFF"モードのページフロー

イベントドリブン指向、および"action-page mapping"の両ページフローで、**"bookmark"の値を"off"にする**ことでわざとXhwlayの特徴である"stateful"機能を無効化することができます。

```
$configP = array(
    "story" => array(
        "name" => "...",
        "bookmark" => "off",
        ),
```

この設定が行われると、以下の機能が無効化されます。

・"Guard", "Event", "Barrier" アクション

・Bookmark, Bookmark Container の全機能

・BCID

また**各アクションの&$bookmark 引数には null** が設定されるようになります。この設定を**"Bookmark-OFF"モード**と呼びます。

"Bookmark-OFF"は Xhwlay を単なるステートレスな"action-page mapping"コントローラとして動作させます。どのページアクションが実行されるのかは、リクエスト中のパラメータに完全に依存するようになります。

3.E. 実行時にカスタムクラスをロードするには

これまでのサンプルでは、さまざまなアクションをユーザー関数として定義してきました。では、自作のクラスとそのメソッドを呼ぶにはどうすればよいでしょうか？そのやり方は次のようになります：

```
"page1" => array(
    "class" => "Tutorial_PageActions_Page1",
    "method" => "staticMethod",
```

"user_function", "class", "method"の全てが定義されていた場合は、"user_function"で定義されたユーザー関数が実行されます。

Xhwlay はまず、"user_function"の定義が有効であるかチェックします。有効であれば、"user_function"を実行します。無効な場合に、Xhwlay は"class"と"method"が有効であるかチェックし、有効な場合"class"クラスの"method"メソッドを static にコールします。（インスタンスは作られません）

それぞれのアクションごとに、予め require() を書いておく必要は有りません。

開発者定義の命名規約に基づいてクラスファイルを実行時にロードするための**"classload"フック**が用意されています。開発者自身の自動ロードをこのフックを用いて実装します。

以下にこの機能の実装例を示します：

ページフロー設定部分：

```
$configP = array(
    "story" => array(
        "name" => "Page Oriented Example",
```

```
        //"bookmark" => "on",
        "bookmark" => "off",
        ),
    "page" => array(
        "page3" => array(
            "class" => "InnerKlass",
            "method" => "staticMethod",
            "bookmark" => "last",
            ),
        "page2" => array(
            "class" => "Tutorial_PageActions_Page2",
            "method" => "staticMethod",
            "next" => array(
                "page3" => null,
                "page1" => null,
                ),
            ),
        "page1" => array(
            "class" => "Tutorial_PageActions_Page1",
            "method" => "staticMethod",
            "next" => array(
                "page2" => "barrier_sample",
                "page0" => null,
                ),
            ),
        "*" => array(
            "user_function" => "demo_page_userfunc",
            "next" => array(
                "page1" => null,
                ),
            ),
        ),
    "barrier" => array(
        "barrier_sample" => array(
```

```
            "user_function" => "demo_barrier",
            ),
        ),
    );
```

次のコードを挿入します：

```
class InnerKlass
{
    function staticMethod(&$runner, $page, &$bookmark, $params)
    {
        // lazy job :p
        return demo_page_userfunc($runner, $page, $bookmark, $params);
    }
}

function demo_hook_classload($hook, $params)
{
    $__basedir = dirname(__FILE__);
    if (!isset($params['class'])) {
        return;
    }
    $klass = $params['class'];
    // translate PEAR-like class name to actual file path
    $klass = strtr($klass, "_", "/");
    $file = $__basedir . "/classes/" . $klass . ".php";
    if (is_readable($file)) {
        require_once(realpath($file));
    }
}
```

続いて、"Tutorial/PageActions/Page1.php" と Page2.php を準備します。以下にこれらの簡単な実装を示します。

Tutorial/PageActions/Page1.php :

```
class Tutorial_PageActions_Page1
{
    function staticMethod(&$runner, $page, &$bookmark, $params)
    {
        // lazy job :p
        return demo_page_userfunc($runner, $page, $bookmark, $params);
    }
}
```

Tutorial/PageActions/Page2.php :

```
class Tutorial_PageActions_Page2
{
    function staticMethod(&$runner, $page, &$bookmark, $params)
    {
        // lazy job :p
        return demo_page_userfunc($runner, $page, $bookmark, $params);
    }
}
```

最後に、"demo_hook_classload()"を"classload"フックに追加します。次のように、二行を追加します。

```
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");
// add start
$h3 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_CLASSLOAD);
$h3->pushCallback("demo_hook_classload");
// add end

$runner =& new Xhwlay_Runner();
```

これにより、Xhwlay は Tutorial_PageActions_Page1/2 が未定義の場合、
demo_hook_classload()を実行し、クラスを定義している PHP ファイルのロードを行うようになります。

3.F. チュートリアルの終わりに

お疲れ様です。みなさんは今、自分自身の Xhwlay アプリケーションを組めるようになりました。

もしかしたら、Xhwlay_Hook や Xhwlay_Var などの、Xhwlay 独特の機能についてより詳しく知りたくなったかもしれません。

Xhwlay はコンパクトです。これら Xhwlay の内部機能についても、おそらく簡単に Hack できることでしょう。

この度は Xhwlay に目を向けていただきありがとう御座いました。