オブジェクト指向プログラミングによる wiki 作成について

溪 等

目　次

——オブジェクト指向プログラミングによる wiki 作成について——

1　はじめに

私は卒業制作でオブジェクト指向を用いた wiki を、PHP によって作成した。この wiki ではデータベースを用いた記事・コメントの投稿、デザインの設定等が可能である。

福田ゼミで必修となるプログラミング演習・UNIX 演習において、ごくわずかだがオブジェクト指向プログラミングを扱う部分があった。しかしそれは非常に基礎的な内容であり、物足りなさを感じたので私は卒業制作という形で、言語は授業やゼミでも扱う機会の多い PHP を用いて、オブジェクト指向によってプログラムを作成することにした。オブジェクト指向を用いてプログラムを作成するために、まず自分自身がその内容を理解せねばならないため、オブジェクト指向を学ぶために利用できそうな資料を探すことにした。

(1) オブジェクト指向の授業内での扱い

最初にプログラミング演習・UNIX 演習内で扱ったオブジェクト指向に関して触れておく。授業の電子テキスト内 [(1)]では以下の 3 つの節に分けて解説が行われた。

1. クラスとオブジェクト
2. クラスの定義の仕方
3. HTML のクラス化

「1. クラスとオブジェクト」では “オブジェクト指向”、“クラス” の考え方の説明が行われる。「2. クラス定義の仕方」では、実際に簡単なクラスを作成し、その内部でクラス変数や関数の解説が行われたのち、オブジェクトの使用方法が解説される。「3.HTML のクラス化」では HTML のタグを関数として定義し、それを用いて簡単なプログラムを

実行する方法が解説されている。先にも述べたがここで扱われる内容は非常に基礎的な内容であるため、メソッド間での値の受け渡し等もなく、ある程度の規模のあるプログラムを作成するにはこの内容だけでは理解が不十分である。

(2) 書籍、Web サイトに見る PHP オブジェクト指向

授業で扱った内容だけではオブジェクト指向は到底理解することができないため、書籍を探してみることにした。しかし PHP の基礎を扱う書籍ではオブジェクト指向に関する記載は乏しく、基礎的な内容を繰り返すばかりで実用性に欠けており、あまり参考にはならなかった。一方でオブジェクト指向に特化した書籍も多少刊行されてはいるものの、フレームワークの利用を前提としているなどオブジェクト指向そのものを一から学ぶために利用するにはやや問題があるように思われた。ただし、今回私はオブジェクト指向そのものの概念を学ぶために、PHP に特化しない『オブジェクト指向でなぜつくるのか』という書籍を参考とした。

Web サイトに関しては「Objective-PHP.NET」[(2)]が PHP におけるオブジェクト指向を基礎的な内容から発展的な内容まで非常に詳しく書かれており、たいへん参考になった。その他 Web サイトに関しては授業内容・書籍と同様に、オブジェクト指向には深く触れていないものがほとんどであった。

2　制作の目的と計画

(1) 何を作成するか

オブジェクト指向の基礎を学んだところで、次は実際に何を作成するかを検討した。簡単なプログラムを作成した上でそのことに関する教材を作成するという考えもあったが、そうするとまた内容的に昨年度以前の電子テキストと似た内容のものを作成することになる。

そこで今回はプログラムを作るのみで、それに関する細かい解説を行う教材は一切作成しないことにした。これは PHP をある程度習得した福田ゼミ生が、当論文と制作物のソースコードを眺めることで、自らの力で「オブジェクト指向」を習得することが理想であり、実際に動くもの、つまり作成した wiki そのものを教材として活用することを目指すものである。

何を作るかに関してだが、「Objective-PHP.NET」にオブジェクト指向を用いた掲示板の作成を行う項目がある。そこでその発展系として wiki を作成することにした。wiki の概念に関しては各授業等でも扱われる機会が多く、また私生活においても触れる機会が多いため、どういった機能を持ち、それがどう扱われているかが学習する側にとって理解しやすいと考えたからである。また wiki のデータをデータベースに登録させることで、データベースの読み書きのコードが必要になるなど、応用のきく内容だと考えたからである。

(2) wiki の仕様

福田ゼミの基本的なテーマは「人の役に立つものを作る」である。今回オブジェクト指向を用いて作る wiki に関しても同様であり、wiki そのものを利用する立場、学習教材として利用する立場の双方から物事を

検討しなければならない。いくら学習教材という面に重点を置いても、使い勝手の悪いプログラムを作成していてはプログラムを作成する意味がまったくなく、人の役に立つということから逸脱するためである。今回は基本的に以下の機能を持たせることにした。

・記事の閲覧
・記事の投稿
・記事編集・削除
・コメントの投稿・返信
・ページ全体の配色などの諸設定

wiki のスタイルとしては図 1 のように左右に 2 つのサイドメニュー、中央部にメインとなるコンテンツを表示する部分を持つ形とする。左サイドメニューはあらかじめ設定しておいてユーザーによる書き換えは行われないものとし、右サイドメニューは記事データの中から任意の 1 件を選択して表示させる形を基本とする [(3)]。

ユーザーが書き換え可能な設定データはデータベースに登録し、そのデータを更新していくという形にしている。始めは設定データ保存用の.dat ファイルを作成してデータを保存していたが、管理・処理のしやすさを考慮した結果、データベースを用いることにした。

また、一般ユーザーが利用するページとは別に管理用のページを作成した。そのページについては以後、管理設定画面と呼ぶことにする。

3　制作物

(1) データベース

このプログラムの記事データ、コメントデータ、CSS を除いた諸設定データの更新は MySQL を利用したデータベースを使用する。wiki_oop

というデータベース内に kiji、comment、setting の 3 つのテーブルを作成する。

kiji テーブルには id，hiduke，title，honbun の 4 つフィールドを作成する。hiduke フィールドには最終更新日を登録していく。データベース作成時、自動的に初期設定でのトップページと右サイドメニューで表示させるための 2 件のレコードが挿入されるようになっている。hiduke フィールドには unix_timestamp 関数 (4)を利用することでデータベース作成時の時刻が自動的に挿入されるようにしている。

comment テーブルには id，kiji_id，hiduke，name，com_honbun，reply_id の 6 つのフィールドを作成する。kiji_id は各コメントがどの記事に対して投稿されたコメントであるかを判定するものである。name には投稿者の名前を入れる。reply_id はコメント間での返信機能を利用した場合の返信元を登録するフィールドである。このテーブルにもデータベース作成時に自動的に 1 件のレコードが挿入される。この時の hiduke フィールドへの処理は kiji フィールドの際と同様である。

setting テーブルには SettnigKey，SettingValue の 2 フィールドを作成する。このテーブルは wiki の設定を保存するために利用し、SettingKey には項目名を、SettingValue にはその値を登録していく。wiki で利用するタイトル，フッター，トップページに表示する記事の ID，新着記事・コメントの表示件数，右側サイドメニューに表示する記事の ID をここで設定している。

(2) ファイル構成

このプログラムはデータベース作成に関わる.SQL ファイル 1 種類（wiki.sql)、クラスを定義したファイル 6 種類（下記参照)、

実際にブラウザで呼び出されるファイル 14 種類、JavaScript ファイル (javascript.js)、CSS ファイル 2 種類（StyleSheet.css, copy_StyleSheet.css）に分けられる。wiki.sql ではデータベースの登録、テーブル・レコードの追加が行われ、javascript.js では入力された値のチェック等に用いる関数がまとめられている。CSS に関しては初期状態では 2 ファイルとも中身は同じだが、ユーザーによるスタイルの変更が行われた場合、変更されたデータは StyleSheet.css に書き込まれる。copy_StyleSheet.css はスタイルを初期設定に戻す場合に用いられるファイルである。

クラスを定義したファイル

- cols.php（連想配列で用いるための定数定義）
- database.php（データベース接続・停止、クエリの実行などを行うクラスを持つ）
- disp.php（データの表示を行うクラスを持つ）
- editData.php（データベースへの登録などを行うクラスを持つ）
- htmltag.php（HTML タグを集めたクラスを持つ）
- system.php（データの編集を行うクラスを持つ）

(3) クラス

今回、下記の 6 つのクラスを作成した。

- database クラス
- base クラス
- cols クラス
- system クラス
- disp クラス

・edit クラス

(4) database クラス

このクラスはデータベースへの接続・切断とクエリを実行する際に用いる。

定数として今回の wiki で利用するデータベース名 'wiki_oop' を定義しておく。connect() メソッドはデータベース接続用、close() メソッドはデータベースの停止を行うメソッドである。各ページの最上部で connect() メソッド、最下部で close() メソッドを実行している。

query() メソッドはデータベースへの書き込みに用いるクエリの実行を行うメソッドである。引数に SQL 文を想定しており、クエリ実行の結果 (5)を戻り値としている。

現在はデータベースにはパスワードを設定せず、その項目は空欄となっている。仮にデータベース接続にパスワードを設定した場合でも、全ページで共通の処理となっているため、connect() メソッドの mysql_connect() 関数に引数を加える (6)だけで、容易に対応できる。

(5) base クラス

このクラスでは HTML のタグを設定している。基本的にはタグごとに 1 つのメソッドを設け、各メソッドでは結果をリターンするようにしている。

開始タグと終了タグの間に他のタグを挟むものに関しては、開始タグと終了タグを分けて作成した。今回は form，ul，table，tr，div の 5 種類である。名称に関しては、それぞれ開始部分に '要素名 Start'、終了部分に '要素名 End' という名前を付与した。例えば form の場合であれ

ば ‘formStart()’ と ‘formEnd()’ である。

(i) クラス、ID の付与

HTML タグでも p，td，div 要素に関しては id 属性や class 属性などを付けられる専用のメソッドを作成している。例えば p 要素においては通常の ～ で文字列を挟む p() メソッドと、 ～ で文字列を挟む p_class() メソッドの 2 種である。

(6) cols クラス

(i) メソッド間のデータの受け渡し

メソッド間でデータの受け渡しを行う場合、必要な数だけ引数を設定すればよいが、戻り値は 1 つしか使用できないため、配列という形をとらなければならない。単純に表示させるだけであれば値を受け渡すことなくメソッド内のその場でプリントして表示させればよい。しかし記事データを扱う場合など、取得したデータを更に加工して表示する時、特に getData() メソッドのように記事データの全件取得を行った場合は配列内に配列が入るという複雑な形になる。getData() メソッドであれば、ID，TITLE，HONBUN，HIDUKE の 4 つの項目が入った記事データの配列を戻り値としている。

(ii) 定数の使い方

この cols クラス内で定義されている定数は、表 1 のようにデータベースのそれぞれのフィールドに対応した 12 種類である。メソッド間のデータ受け渡しを行う際、それぞれの値の管理がしやすいようにこのクラス内の定数を表 2 のように配列のキーとして利用している。

実際に定数を利用する場合「クラス名::定数名」と記述する。この構造が理解できれていれば、データのやりとりに関しては問題ない。表 2

は記事の場合に関して書いているが、コメントやスタイルを含めた他の処理においても同様に配列内に配列を設定する方法を用いている。

（7）system クラス

wiki で利用するデータの呼び出し、編集等を行うためのクラスである。基本的にはこのクラス内のメソッドでデータを呼び出し、後述の disp クラス内のメソッドでそのデータを受け取って表示する形をとる。

（i）グローバル変数

このクラスのグローバル変数には$id，$header，$menu，$function_list，$file_name，$css_data の 6 種がある。

$id は記事を表示するための ID を受信して入れておく。この値は後述するコンストラクタで代入される。

$header，$menu はヘッダーと左メニューの設定に用いられる。それぞれ SetHeader() メソッドと SetMenu() メソッドから値を代入する。それぞれのメソッドは使用するヘッダーと左メニューを引数として扱う。引数となる文字列は、通常時は ‘NormalHeader’ と ‘NormalMenu’，管理設定画面では ‘SetupHeader’ と ‘SetupMenu’ である。送られてきた値は$header と$menu にセットされる。この値に合わせてヘッダーと左メニューを生成するときに呼び出すメソッドが変わってくる。

$function_list は右サイドメニューでメソッド一覧を表示させるときに使用するための変数である。この変数にはメソッド名の一覧が配列として代入される。

$file_name は現在接続しているページのアドレスが入っている。この値もコンストラクタによる。

$css_data は wiki で使用するスタイルシートのデータが配列として代

入される。この変数は管理設定画面での CSS データの編集を行う際に用いられる。

（ii）定数

グローバル変数の次に宣言されている定数は wiki で使用する CSS, JavaScript ファイル名の設定と、データ整形等で用いる正規表現のパターンを規定しておくものである。定数 StyleSheet では全ページ共通のスタイルを定めた StyleSheet.css、定数 JavaScript では各ページの入力値チェックなどで用いる JavaScript.js が定義されている。

（iii）コンストラクタ

PHP オブジェクト指向において、__construct() はコンストラクタと呼ばれ、新しいクラスが生成された際に自動的に実行されるメソッドである。今回は各ページ共通の処理となる、記事の表示で使用するための ID の受信と、現在接続しているページのアドレスの取得、各種設定データの受信をコンストラクタの内部で行っている。

ID の受信は複数のパターンが考えられ、まずアドレスの後ろに GET として送信されていないかを確認する。それがなかった場合には Form からの POST、それもなかった場合には直近のクエリで実行された ID を呼び出す。もちろん、最初にページにアクセスした場合など ID の受信がない場合も考えられるので、else は特に設定していない。

```
if(isset($_GET['id'])) {
    $id = $_GET['id'];
    $this->id = $id;
} else if (isset($_POST['id'])) {
    $id = $_POST['id'];
    $this->id = $id;
```

```
} else if ( mysql_insert_id() ) {
    $id = mysql_insert_id();
    $this->id = $id;
}
```

(iv) メソッド

このクラス内では記事やコメントを呼び出すために用いられるが、例えば記事を呼び出す場合でも 2 パターンがある。一覧表示で使用するために記事を全件取得する getData() メソッドと、記事の詳細表示に使用するために ID によって記事を取得する getDataId() メソッドである。

この 2 つのメソッドでは、それぞれ必要な SQL を定義し [7]、その SQL を引数として getKiji() メソッドを実行する。getKiji() メソッドでは引数として受け取った SQL を使ってクエリを実行し、記事データを配列化して返している。このようにして、なるべくメソッド内に重複が発生しないように心掛けている。なお、この system クラスは表示を行うものではない。配列化した記事データを返して、表示を行う disp クラスのそれぞれのメソッド [8]で更に表示に適した形に整形し表示する。

(8) disp クラス

このクラスは基本的にはデータの表示を行ったり、表示するための整形をしたりするメソッドの集まるクラスである。このクラスは system クラスを拡張したものでる。

すべてのページで共通して使用するメソッドとして、HTML 開始部分に利用する HtmlStart() メソッド、HTML 終了部分に利用する HtmlEnd() メソッド、ボックス構造の中心部分を定義する frame() メソッドなどがある。これらのメソッドによって各ページの HTML の骨

格部分とボックス構造が生成されていく。今回 wiki ではボックス構造を構成するために以下の通り 7 つの div 要素に ID を付与している。

・ wrapper：全体を囲むもの
・ header：ヘッダー部分
・ main：ヘッダー下部からフッター上部までを囲む
・ menu：左サイドメニュー
・ contents：メインコンテンツ表示部分
・ source：右サイドメニュー
・ footer：フッター

(i) HtmlStart() メソッド

HtmlStart メソッドでは HTML の<!DOCTYPE html>から<body>までを定めている。ページごとにタイトルは異なるため、<title>～</title>で囲まれるタイトル部分に表示する文字を引数として設定しており、その文字列がページのタイトルとして使用されるようになっている。

このメソッドでは最初に PHP のタイムゾーンの設定を行っている。タイムゾーンの変更はこのプログラムでは想定しておらず、東京をデフォルトで設定してある。またこのメソッドでは使用するスタイルシートと JavaScript ファイルの呼び出しも行っている。この wiki における JavaScript の役割については後述する。

(ii) HtmlEnd() メソッド

HtmlEnd() メソッドは表示画面での中央部に当たるメインコンテンツ（#contents，下記 ボックス構造 参照）終了部分から始まる。右サイドメニューを表示したのちフッターの表示を行い、body，html を終了させる。

```
</div>
<div id="source">
eot;
$this->setSideMenu_r();
print <<< eot
</div><!-- end of #source-->
</div><!-- end of #main-->
<div id="footer">
{$this->footer()}
</div><!-- end of #footer-->
</div><!-- end of wrapper-->
</body>
</html>
```

（iii）frame() メソッド

frame() メソッドは div 要素の#wrapper 開始部分から#contents 開始部分までを定義する。

```
print <<< eot
<div id="wrapper">
<div id="header">
{$this->{"$this->header"}()}
</div>
<div id="main">
<div id="menu">
{$this->{"$this->menu"}()}
```

```
</div>
<div id="contents">
eot;
```

ここで表示されるサイドメニューだが、後述する「その他設定」での新着記事表示件数が 1 以上に設定されている場合には記事の新規投稿，新着記事，新着コメント，検索メニュー，管理設定画面へのリンクの 5 項目を表示する。新着記事の表示件数が 0 件に設定されている場合は、新着記事，新着コメントの表示は行われない。

管理設定画面での左サイドメニューは通常表示へ戻るリンクと設定画面トップ，記事全件一覧表示，コメント全件一覧表示，その他設定，スタイルシートの設定それぞれのページへのリンクを表示する。

(9) edit クラス

このクラスは disp クラスを拡張して作られ、データの登録・更新などが行われるメソッドを集めたクラスである。ユーザーが Form で送信したデータを受信するメソッドなどはこのクラスにまとめられている。

この wiki では基本的にはデータベースを用いてデータ管理を行っているため、ほとんどのメソッドではデータを受信してそのデータをもとに SQL 文を生成し、その SQL 文を引数として database クラス内の query() メソッドによってクエリを実行し、データの登録・更新を行っている。

ただし、スタイルシートの更新のみはデータベースで行うことができないため、直接 CSS ファイルに書き込んでいる [(9)]。

（10）JavaScript の役割

この wiki では入力内容のチェックにはできる限り JavaScript を使用するようにしている。入力内容のチェックは入力必須項目が空欄である場合や入力内容に不正な値がある場合など、誤った値がデータベース等へ登録されることを防ぐためのものである。これらの機能は PHP のみを用いても実装可能ではあるが、その場合どうしてもページ遷移を伴うチェックとなってしまい、使い勝手の悪いものとなってしまう。そのため今回はページに留まったままでアラート表示などが可能な JavaScript を使用している。

チェック内容に関しては利用項目ごとに関数にまとめ、送信ボタン等を押した際に実行される仕組みである。実行された関数では問題がないと判断された場合のみデータ送信の処理を行うようにするため、チェックを行う箇所での送信ボタンの type 属性は button としている。

（11）特記すべき項目

（i）記事の並び替え

記事の閲覧に際して、図 2 のような記事の一覧表示をする list.php（管理設定画面では SetupList.php・図 3 ）と、記事の詳細を表示する shosai.php（管理設定画面では SetupShosai.php）を作成した。一覧表示には ichiran() メソッドを利用している。このメソッド内では sort() メソッドと getData() メソッドを主として利用している。sort() メソッドでは並び替えの基準となるセレクトボックスを生成している。並び替え項目とそれぞれ値は以下の通りである。（項目：値）

- 日付昇順：kiji.hiduke
- 日付降順：kiji.hiduke desc

・タイトル昇順：kiji.title

・タイトル降順：kiji.title desc

・ID 昇順：kiji.id

・ID 降順：kiji.id desc

option 要素では SQL での order by 以下の記述がそのまま値となっている。この値を引数として getData() メソッドを利用する[10]ことで、指定した並び方でレコードが呼び出され、並び替えが実行される。設定管理画面での一覧表示は基本的には通常画面のものと同様であるが、トップページに登録されている記事が「☆」印をつけて表示されるようにしてある。

(ii) 記事の投稿

記事の投稿と編集というのは一見全く別の処理を行っているように思えるが、データを入力させ登録するという点では同じである。処理が共通化できると保守管理面でも都合が良いと考え、これらの動作を一つにまとめる方法を考えた。しかし新規登録と更新とでは SQL に差があるため、完全に同じ処理とするのは難しい。結果として考えたのが全て更新の処理にまとめるということで、新規投稿を行う場合は図 4 のように先にタイトルをユーザーに入力してもらい、タイトルのみをまずデータベースに登録する。その時点では他フィールドは空である。その後記事編集画面へ移動し、そこで図 5 のように先ほど登録したレコードの編集を行うという形である。

この一連の動作は edit.php 内の edit() メソッドを呼び出して行われる。データベースに登録した直後に同一ページ内で再度呼び出すに当たって当該記事の ID をどのように取得するかという問題があったが、MySQL で直近のクエリで生成された ID を得るという my_insert_id()

関数を利用することで簡単に解決された。この my_insert_id() 関数はコンストラクタ内で実行され、グローバル変数$id に代入されている。

記事の新規登録・編集後に送信ボタンをクリックすると JavaScript の title_check() 関数が呼び出される。これは入力された値に問題がないかを判定するためのもので、タイトルが入力されていない場合と、すでにそのタイトルが使用されている場合はその旨をアラート表示し、データを送信できないように設定している。タイトルが重複していると右サイドメニューに表示するデータを選択する際に不具合が生じることから追加した機能である。

(iii) コメントの返信機能

コメントには input 要素を使って返信ボタンを作成している。このボタンがクリックされると JavaScript の SetReply() 関数が実行される。この関数はそのコメントの ID とその投稿者のデータを引数としている。SetReply() 関数内では送られてきたデータを基に textarea の値として、図 6 のように “>>(name) さん” という表示が追加され、input 要素の reply_id には設定が hidden となっているため表示は行われないが、返信元のコメントの ID が値として付与される。もし返信機能を利用しなかった場合は、reply_id には 0 が入る。

(iv) 表示項目に関する設定

表示項目の設定としてタイトルの変更、フッターの変更、サイドメニューの新着記事・新着コメント表示件数の変更、右サイドメニューに表示する記事の選択が可能である。図 7 がその設定画面である。これらのデータの変更に関しては setupList() メソッドの中で、表示するデータが扱われる。タイトルとフッターに関してはテキストボックスとして直接文字を打ち込むのみである。新着記事・コメントの表示件数に関し

ては input 要素の type 属性を number[11]としている。

右サイドメニューに表示する記事の選択はセレクトボックスから記事タイトルを選択して決定する。先に述べた通り、記事登録時に JavaScript でタイトルの重複が発生しないようにしたのはこのためである。右サイドメニューに関しては記事データの他に各ページで使用されるメソッド一覧を表示することも可能である。

この setupList() メソッドにおいても、更新ボタンのクリックで入力値チェックのための、JavaScript の setting_check() 関数が呼び出される。タイトル・フッターに関しては入力されているかどうかのチェックに留まるが、新着記事表示件数の各項目に対しては 0 以上の数値が入力されているかのチェックが行われる。問題のなかったものに関しては edit クラスの RemoveData() メソッドでデータの登録が行われる。

(v) CSS の設定

この wiki ではスタイルに関する情報はすべて StyleSheet.css に記録される。制作段階におけるスタイル設定画面（SetupCss.php）では StyleSheet.css の中身を呼び出し、全てをそのまま textarea に入れて表示をしていたため、スタイルの書き換えとしては実用的なものではかった。そこで書き換え項目ごとに input 要素で値を表示し、スタイルの変更が簡単にできるものを目指した。結果として完成したのが図 8 のようにテキストボックスを利用したものである。なお今回ユーザーがページ上で書き換えができるのは色に関するデータのみとしている。

このページを表示させるためには setCssData() メソッドによって StyleSheet.css の内容を取得するところから始まる。ファイルから一行ずつ読み込む中であらかじめオブジェクト定数として定義してある正規表現の検索パターンを用いて取得したデータからセレクタのみを抜き出

す。そうして取得したセレクタ名をキー、そのセレクタに対応するプロパティ・値を要素とした配列 $css_array を生成する。生成された配列はグローバル変数$css_data に代入しておく。ここからは背景色書き換えのための入力欄を作成するという具体的な手順で解説していく。

まずユーザーの書き換え対象となるプロパティをグローバル定数で定義する。この定数は背景色の場合 “`/background-color[ ]?:[#a-z0-9 ]*/`” とするなど、CSS の中からそのプロパティとその値を探すのに用いるため、その項目ひとかたまりを表すことのできる形にしておく (12)。

続いて書き換え対象をどのセレクタに対して行うかを決めておく必要がある。これは項目ごとに専用のメソッドを一つ作成する。setbg-color_key() メソッド、あるいは setcolor_key() メソッドの中身を見れば分かるが、セレクタ名をキー、そのセレクタの意味を説明する日本文を値とした配列を作成している。ここで作った値（日本語の説明文）は実際に表示した時にユーザーに分かりやすいものとするために利用する。

ここまでで作成してきた CSS 全データを持つ配列 (13)、書き換え対象のセレクタを定めた配列、書き換え対象のプロパティを定義した定数の 3 つを使用して実際に表示される部分を作成していく。ここからは表示用の EditCss() メソッドの内部の記述である。

書き換え対象のセレクタを入れた配列のキーを用いて css_data から該当するセレクタのプロパティと値を取り出す。取り出したそのデータと書き換え対象のプロパティを定めた定数の 2 つを引数として find_css() メソッドを実行すると、該当プロパティの現在の値が戻り値としてセットされる。その時のセレクタ名をキー、find_css() メソッドからの戻り値を値として配列を作成する。これを foreach の中で、値を input 要素で表示させればよい。

下記が実際に EditCss() メソッド内で背景色に関してこの動作を実行している部分である。

```
$css_array = $this->css_data;
$bg = $this->setbgcolor_key();
$css_bgcolor = system::CSS_BGCOLOR;
       // "/background-color[ ]?:[#a-z0-9 ]*/"のこと
foreach ( $bg as $key => $val) {
    $bgcolor[$key] = $this->find_css($css_array[$key], $css_bgcolor);
}
```

$css_array には CSS 全データが配列として収められている。また、$bg の中身は setbgcolor_key() メソッドによって使用するセレクタと説明が、それぞれ配列として収められており、下記表の通りである。

$bg の中身（一部抜粋）	
キー	値
body	背景
div#wrapper	サイドメニュー
div#menu li a:hover	メニューにマウスを載せたとき

これによって定義される$bgcolor という配列の中身は以下のようになる。

$bgcolor の中身（一部抜粋）	
キー	値
body	#eef3ff
div#wrapper	#dfefff
div#menu li a:hover	#ffc9d5

この 2 つの配列を用いて下記記述より、ユーザーによる入力画面を作成する。

```
<tr>
    <td>$bg[$selecter]</td>
    <td><input type='text' name='bgcolor[$selecter]' value='$value'></td>
    <td class='mihon'></td>
</tr>
```

実際にはテーブルとして表示するので<tr>～</tr>や<td>～</td>の記述が見られる。

CSS における色の指定は 16 進数 RGB 各 2 桁ずつの計 6 桁で行われることが多い。しかし設定自体は 16 進数 RGB 各 1 桁ずつ計 3 桁での指定や、10 進数 RGB 各 3 桁ずつをコンマで区切った計 9 文字、具体的な色の名前の指定なども利用できる。また 10 進数を利用すると透明度の設定も可能となる。

データ送信段階で入力値に異常がないかチェックを行う都合で、使用できる値に制限を設けなければならない。そのため今回は一般的に利用されることの多い 16 進数と、あまり RGB 値の知識がなくとも設定できる具体的な色名の直接指定が可能な形としている。直接色を指定す

る場合は非常に多くの数があるため、今回は基本 16 色 [14]のみを指定可能とした。CSS の設定は基本的に 16 進数が用いられることが多いため、10 進数に関しては今回は非対応とした。

スタイル設定画面下部の適用ボタンをクリックすると JavaScript の css_check() 関数が呼び出される。css_check 内では入力されている値に不正な値がないかを全てのテキストボックスに対して調べる。このチェックでは JavaScript 上の配列 css に登録された値以外をエラーとする設定にしている。規定された値以外の入力値が発見された場合は“不正な値が入力されました” とアラート表示させた上で図 9 のようにテキストボックスに背景色を付けることで、エラー箇所が分かりやすくなるよう工夫している。エラーが発見されず無事に送信された値は UpdateCss() メソッドで新しく受信した値を含んだ全 CSS データを変数に入れ、StyleSheet.css ファイルに上書きする。

なお入力値のチェックさえクリアできればあとは機械的に CSS が更新されるため、チェックを行うために使用される JavaScript の css_check() 関数の配列 css に値を追加することで、色の直接指定などで使用可能となる値は簡単に追加することができる。

4　評価

(1) アンケート

ある程度形となったものを福田ゼミ 3 回生 8 名に wiki を利用する立場として制作物を使用してもらい、アンケートを実施した。全員からシンプルで見やすく、操作もしやすいという意見を得られた。また、すごく完成度が高いという評価を得ることもできた。しかし問題点・改善すべき点もいくつか指摘された。

(i) 指摘された問題点

指摘された主な問題点は以下の 5 点である。

1. テキストエリアの大きさが不適切
2. スタイル設定のクリアができない
3. 新着記事の件数表示がおかしい
4. リンクなどにデザインがほしい
5. 検索結果にも色付けがほしい

(ii) 改善を行った点

テキストエリアの大きさに関しては、ブラウザのサイズを変更するとデザインが崩れるという指摘が 4 名からあったので、StyleSheet.css に ‘textarea　width：95%; ’ という記述を追加することによって解決した。

スタイル設定のクリアに関しては全データのコピーを取っておき、内容を全て置き換えることでひとまず可能となった。つまり copy_StyleSheet.css で初期設定時の CSS データをそのまま保持させておき、スタイル設定の画面で「初期設定」ボタンが押された時に StyleSheet.css の中身を copy_StyleSheet.css で置き換える、という処理を追加することで解決した。

新着記事の表示件数は恐らく左サイドメニューの表示に関わるものと思われる。元々トップページに表示する新着件数が 5 件と設定されていれば、左サイドメニューには実際に存在するレコード件数に関わらず「新着記事 5 件」「新着コメント 5 件」と表示されていた。これを表示を行う前にレコードの件数を調べ、新着記事表示件数で設定された値と比較し、小さい方の値で表示させることで解決した。

(iii) 課題

指摘された問題点の中でも改善できていない点がある。リンクにもデザインがほしいという点に関しては検討は重ねたのだが、スタイルシートによるこれ以上の設定は困難と考え、リンク部分を画像化するなどの措置を考えていたが、現在のところまだ実装できていない。

検索結果への色付けに関してはメソッド詳細表示と同様の処理を行えば可能だと考えたが、正規表現のパターンが一定ではないため、処理を模索しているところである。

またアンケートとは別に、3 カラムを表示する今回のレイアウトにおける各カラムの背景色の設定方法についてもまだ模索段階である。というのも 3 つのカラムの中で一番高さのあるものにあわせて他のカラムも高さを設定する必要があるレイアウトなのだが、カラムの高さがページとその時の状況によって変わってくるため、それにあわせた処理が必要となるのである。現時点では一見解決しているかのように思われるが、カラム部分の div 要素に対して 'overflow:hidden;' と設定しているため、ID へのリンクを設定できないという難点がある。左メニューの新着コメントの項目からのリンクが、詳細表示の先頭に飛ぶのみでコメント欄に行かないのはこのためである。

(2) 自己評価

今回、オブジェクト指向で wiki を作成した。機能に関してはアンケートからもわかるが、利用しやすい画面が作れたことをはじめとしてある程度満足はしているが、オブジェクト指向そのものの体をなしているか少し疑問の残るところである。

「人の役に立つものを作る」ことに関して、wiki 利用者の立場として

はそれなりの配慮のできたものが作成できたと思う。しかしこの wiki を学習教材として見るとなると問題点は多々ある。たとえば現在すべてのアクセス修飾子が public となっていること、メソッドの内部が多少重複していたりするなどスマートさに欠ける部分があり、改善すべき点は多い。制作にあたってきちんとした設計図を作成しなかったのがまず大きな失敗であったと考える。オブジェクト指向プログラミングにおいて重要な汎用性の高さ、たびたび再利用のできるクラスやメソッド作りをするにはまずきちんとした構想と設計図が必要であったと、一番重要なことを今になって感じる。

しかし少なくとも利用者としては扱いやすい wiki が出来たということがアンケートから見えてきたこともあり、目標の半分は達成されたことには満足している。今後この wiki や論文を見て、オブジェクト指向に少しでも興味を持つ学生が現れれば幸いである。

注

（1）http://fukuda.tibetan-studies.net/otani/ 内の 2013 年度配付資料「15．関数を定義する」と「16．HTML を関数化する」が該当する

（2）http://www.objective-php.net/

（3）記事表示かメソッド一覧を選択

（4）1970 年 1 月 1 日 0 時 0 分 0 秒からの経過秒数を返す

（5）mysql_query() の結果。データもしくは FALSE が入る

（6）引数は mysql_connect(MySQL サーバー名, ユーザー名，パスワード)

（7）全件の場合：‘select * from kiji order by $field;’ $field は引数。並び替え順序が送られる。

ID による場合：‘select * from kiji where id = $id;’ $id は引数。記事の ID が送られる。

（8）ichiran() メソッド，shosai() メソッドなど

（9）fopen() と fwrite() を用いる

（10）引数を$order とすると SQL は ‘select * from kiji order by $order’ となる

（11）HTML5 で追加された。数値の入力欄を生成する。

（12）抜き出した値は具体的には “background-color：#ff4500;” のようになる。

（13）グローバル変数$css_data

（14）基本 16 色：Red, Maroon, Fuchsia, Purple, Lime, Green, Yellow, Olive, Blue, Navy, Aqua, Teal, White, Silver, Gray, Black

文献表

平澤章

2011　『オブジェクト指向でなぜつくるのか 第２版』日経 BP 社

高橋良明

2005　『正規表現実践のツボ』九天社

オブジェクト指向プログラミングによる wiki 作成について

これまでのスケジュールとクラス再編成案

人文情報学科　情報デザインコース
1148055　溪　等

目次

制作におけるスケジュール

今回の wiki を作成するにあたっての大まかな流れをここに示す。

制作開始

制作を開始したのは 6 月のはじめである。6 月の間に各ページ共通となる部分を作成している。また、データベース間でのデータの受け渡しに関してもこの時期に検討している。

7 月に入ってコンストラクタの概念を導入している。その後少しずつ細かい部分に肉付けする作業を進めている。

後期授業期間

後期開始段階でまだ実装されていなかった機能が以下のとおりである。

☆ 管理設定画面
- まだタイトル・ヘッダーの入れ替えしかできなかった。
- 新着記事表示件数や右サイドメニューの編集

☆ トップページの処理
- トップページでブログのように一覧表示を行っていた
- wiki らしいデザインとするためにトップページには任意の記事を表示する仕様に変更。

☆ サイドメニュー
- 新着記事・コメントの表示がまだ行われていなかったので、新着記事を追加した。

☆ 一覧表示におけるソート
- 並び替え機能を追加するために getData()メソッドに引数を追加した。

☆ 検索
- その後単語間をスペースで区切り、and 検索に対応させた。

10 月に入って諸設定データをデータベースに登録する仕様に変更した。これは論文にも記載した通り、呼び出し・編集・更新が容易にできるようにするためである。この処理方法の変更によってファイルの読み書きは CSS データのみが対象となった。

その後正規表現を学習してメソッド一覧と CSS の編集を行えるようにしている。これは、ここまで機能重視で作制してきた wiki に、実用性を求めるようになった結果である。ここから「人の役に立つ」デザインで制作していく。

根本的なデザインの変更

教材作成を行わないという決定をした後期になってから、大幅にデザインと機能を変更

した。元々は左側にサイドメニュー、右側にコンテンツ本体の表示を行う 2 カラム型であった。しかし、この wiki そのものを教材として使用してもらうにあたり、何かオブジェクト指向を学べる仕様に変更する必要が生じた。その際に追加した機能がメソッド一覧であり、このメソッド一覧を右サイドメニューへ配置したことから 3 カラム型となった。

disp クラスの HtmlEnd()メソッドが、HTML 終了部分を名乗りながら右サイドメニューの表示を行っているなど、不自然な点が発生したのはこのためである。この段階ではすでに大まかな形として出来上がっており、新たに frame()メソッド以下、HtmlEnd()メソッド以上の位置で処理を行うメソッドを追加することは困難であったため、やむを得ず追加したものである。

アンケートとその結果を受けて

12 月上旬に 3 回生に試用してもらい、アンケートを取った。アンケート結果に関しては主論文に記載した通りだが、テキストボックスのサイズの問題やスタイルのクリアの問題、リンクの間違い、検索結果の色付けなど、多々指摘があった。基本的な部分に関してはおおよそクリアしたが、それでも時間が足りなかった感は否めない。制作物に関する構想が甘く、形を組み上げていくのに時間がかかったためである。

制作過程における反省点

制作物そのものに関する構想も甘かったが、制作のスケジュールをきちんと考えなかったのも問題であったと考える。副査の上田先生の指摘の通り、マイルストーン的な節目をあらかじめ設定しておいた上で制作を行う必要があった。次年度以降の後輩たちには是非そこに気を配って制作をしてもらいたいと思う。

クラス再編成案

今回オブジェクト指向プログラミングによって制作した wiki のクラスやメソッドを再編成するならば、どのような構造にするかという考えをまとめてみた。データベースそのもののフィールド名や、ページの持つ機能はそのままに、クラスやメソッドのみを再編成する。

クラスの現状

オブジェクト指向プログラミングを用いた wiki を再構築するにあたって、まず先に作成したプログラムを再確認してみる。今回作成したクラスは下記の 6 つである。

- base クラス
- cols クラス
- database クラス
- system クラス
- disp クラス
- edit クラス

基本的な処理としては database クラスでデータベースに接続し、system クラスでデータベース内のデータを抜き出して整形、それを disp クラスに渡して表示を行う。データの中身を編集する場合は disp クラスからデータを飛ばし、edit クラスで受信して必要な整形を行った上で登録を行う。処理を行う部分に関してはきちんとクラスごとに分化されているが、表示に関するメソッドは呼び出し先に関わらず、すべて disp メソッドに含まれるなど、クラスが非常に肥大化している。そのためこれを細分化し、より効率の良い形にすることを目指す。

表示クラスの編成例

表示を 3 カラム（左メニュー・コンテンツ・右メニュー）で行うために、ボックス構造を行わなければならないが、ボックス構造は全ページにおいて共通である。またボックス構造以外に、タイトルやフッターなど諸設定を行うための各データの受信も全ページ共通であり、かつ必須の処理となる。このような処理を持つクラスをまず一つ作成する。そのクラスを継承して、各処理に必要なメソッドなどを追加していくことにする。すべての大元となるクラス名を今回は wiki クラスとする。

クラスの継承を行うにあたって、まずどのような機能が今回作成した wiki に実装されているかを考える。共通の処理に関しては継承元となるクラスに持たせているので考えないとすると、コンテンツ部分の処理に応じてクラス分けが必要であると考えられる。記事を表示する場合で考えると補助的に必要となる処理を除いて大まかに以下のようになる。なお括弧内はクラス名である。

○ 記事表示画面
  - 一覧表示（kiji_ichiran）
  - 詳細表示（kiji_shosai）
    - ✧ 通常の詳細表示（normal_kiji_shosai）
    - ✧ トップページ用の詳細表示（index_kiji_shosai）

○ 編集画面
  - 新規投稿（new_edit）
  - 通常編集（normal_edit）

○ 検索結果表示画面（search）
  - 検索結果詳細表示（search_shosai）

これらの処理を大元となる wiki クラスから継承したクラスに追加していく。通常のユーザーが利用する場面と管理設定画面で利用する場面のそれぞれ専用クラスを wiki の次位に継承する。クラス名はそれぞれ user クラスと setting クラスとする。

具体例として、ユーザー画面での通常の詳細表示を行う normal_kiji_shosai クラスに関しては、wiki→user→kiji_shosai→normal_kiji_shosai という順に継承を行うためソースコードは以下のようになる。

```
class wiki {
        // 各ページ共通となる処理
        // ボックス構造の生成
}
class user extends wiki {
        // 諸設定データの受信・定数化
}
class kiji_shosai extends user {
        // 記事の詳細表示を行う上で必要な処理
        // ID による記事データの受信など
}
class normal_kiji_shosai extends kiji_shosai {
        // 通常画面での記事の詳細表示を行う上で必要な処理
        // コメントデータの受信など
        // print
}
```

もちろん wiki クラスにおいて、各ページ共通となる諸設定データの受信・定数化はコン

ストラクタを利用して行う。

設定画面でのクラスに関しては以下のようなクラス編成となる。括弧内はクラス名である。

○ 設定画面

- 管理設定画面トップ（ setting_top)
- 記事一覧ページ（setting_kiji_ichiran）
- 記事詳細ページ（setting_kiji_shosai）
- コメント一覧ページ（setting_comment_ichiran）
- コメント詳細ページ（setting_comment_shosai）
- その他設定（setting_other）
- CSS 設定（setting_style）

記事一覧ページである setting_ichiran クラスに関しては wiki → setting → setting_ichiran の順に継承を行うため、下記のようなソースコードになる。

```
class setting extends wiki {
        // 管理設定画面での諸設定データの受信・定数化
}
class setting_ichiran extends setting {
        // 記事データの受信など必要な処理
        // print
}
```

読み込みに関するクラスの編成例

データの読み込みを行うメソッドは大まかに、全ページで共通の処理を行うものと、使用用途ごとに全く別の処理を行うものの 2 つに分類される。前者のようなメソッドの場合は、上のソースコード例にも書いたように wiki クラスの内部で定義してしまって問題ないと思われる。後者の使用用途ごとに特有の処理を行うものに関しても、一度しか使用しないものに関しては表示を行うためのメソッド内で記述すると無駄のないものが作成できると考えられる。

ここで考えなければならないのは、2 つ目に分類した使用用途ごとに別の処理を行うメソッドの中で、再利用が可能と思われるメソッドである。例えば現在のところでいうと、getDataId()メソッドなどが該当する。このメソッドは id をもとに記事データを受信するために利用している。実際に使用されているのは記事の詳細表示（通常のもの、トップページ用のものいずれも）や記事の編集（新規投稿を除く）などである。専用のクラスを作って使用する場面でそれを継承すれば処理が楽なのだが、PHP では 2 つのクラスから継承をすることができないため、それは不可能である。そのため、専用のクラスを生成してインスタン

ス化して呼び出すのが有用かと思われるが、特にインスタンス化する必要もないように思われる。そのため静的メンバとして、インスタンス化せずに呼び出すことにする。

記事データを呼び出すための getKijiData クラスを生成し、その中に記事の受信に関するメソッドをまとめるようにする。具体例として記事の全データの呼び出し（getData()メソッド）と id に応じた記事の受信（getDataId()メソッド）の場合は次のようになる。

```
class getKijiData {
        public static function getData($order)
        {
                $sql = “select * from kiji $order; ”;
                $data = mysql_query($sql);
                while ( $record = mysql_fetch_array($data)) {
                        $kiji[cols::ID] = $record[“id”];
                        $kiji[cols::TITLE] = $record[“title”];
                        $kiji[cols::HONBUN] = $record[“honbun”];
                        $kiji[cols::HIDUKE] = $record[“hiduke”];
                        $kiji_data[] = $kiji;
                }
                return $kiji_data;
        }
        public static function getDataId($id)
        {
                $sql = “select * from kiji where id = $id; “;
                $data = mysql_query($sql);
                if ( $record = mysql_fetch_array($data) ) {
                        $kiji[cols::ID] = $record[“id”];
                        $kiji[cols::TITLE] = $record[“title”];
                        $kiji[cols::HONBUN] = $record[“honbun”];
                        $kiji[cols::HIDUKE] = $record[“hiduke”];
                }
                return $kiji;
        }
}
```

getData()メソッドを呼び出すときは「getKijiData::getData(“”);」で呼び出される。引数としてクエリ実行時の抽出順序を指定できる。例のように「getKijiData::getData(“”)」とした

場合、引数は特に指定されていないので、抽出順序は指定しないことになる。現状のgetData()メソッドではorder by以下が引数となっているため、抽出順序を問わない場合も何かしらのフィールド名を指定しなければならなかったため、今回は多少手間ではあるがorder by という文字列も引数として設定する必要がある処理としている。例えば引数として「order by hiduke desc」を持たせることで、実行するSQLを”select * from kiji order by hiduke desc;”とすることができる。また、getDataId()メソッドを利用する場合は引数としてidを持たせるが、idはwikiクラス内にグローバル変数を持たせ、それを継承した各クラスから行うので問題ないと思われる。

しかし2つのメソッド内で実行される、mysql_query()関数やmysql_fetch_array()関数などで実行されるMySQL関連の関数はPHP5.5.0以降非推奨とされている。そのため今後変更することを検討すると、データベース関連の関数を実行する場合は専用のメソッドを一つ設け、将来的にMySQLiやPDO_MySQLへの変更が容易にできるようにした方がよいかもしれない。現在systemクラス内にあるquery()メソッドなどはそのためのメソッドであるが、再編成にあたってはこのメソッドは使用していない。

データの登録・更新・削除を行うクラス

編集画面などから送られてきたデータを受信し、データベースへ登録・更新・削除するメソッドを持たせるクラスをdatabaseとする。databaseクラスから記事に関するもの、コメントに関するもの、諸設定に関するものの3種にそれぞれ継承する。SQLはそれぞれの処理ごとに異なってくるが、クエリの実行に関しては同じ処理を行うため、その部分は継承元となるdatabaseクラス内にメソッドとして定義しておく。

database クラス内の変数$sql と記事関係の処理を行う database_kiji クラスに関しては下記の記述となる。

```
class database {
	private $sql;
	public function query()
	{
		$sql = $this->sql;
		$data = query($sql);
		if ( ! $data ) {
			print “ERR：登録できませんでした。”;
		} else {
			print “登録完了。”;
		}
	}
```

```
}
class database_kiji extends database {
	public function kiji_jushin() {
		$data[cols::TITLE] = $_POST('title');
		$data[cols::HONBUN] = $_POST('honbun');
		$data[cols::HIDUKE] = time();
		return $data;
	}
	public function regist_sql() {
		$data = $this->kiji_jushin();
		$sql = " /*  ここに$data を用いて新規登録 SQL を作成  */ " ;
		$this->sql = $sql;
	}
	public function kiji_updata_sql() {
		$data = $this->kiji_jushin();
		$sql = " /*  ここに$data を用いて更新の SQL を作成  */ ";
		$this->sql = $sql;
	}
	public function kiji_delete_sql() {
		$id = $_POST['id'];
		$sql = "/*  ここに$data を用いて新規登録 SQL を作成  */ ";
		$this->sql = $sql;
	}
}
```

実際にインスタンス化し実行すると、下記のようなコードになる。

```
$regist = new database_kiji;
$regist = regist_sql();
$regist = query();
```

この 3 行の記述で、送られてきた記事のタイトル・本文を受信し、受信した時刻を取得して SQL を生成し、クエリを実行する。きちんと登録できた場合は「登録完了。」と表示され、万が一エラーした場合は「ERR：登録できませんでした。」と表示される。

　この database_kiji と同様の処理をコメントの場合と設定の場合においても行う。その際、クラス名はそれぞれ database_comment と database_setting とする。

データベースを用いないものの更新に関しては専用の updataCss クラスを作成し、その内部で完結する処理を行う。データベースを用いないものは今回 CSS の更新のみである。従来の edit クラスの中から CSS の更新に使用していたものをそのまま転用したものを updateCss クラスとする。

その他のクラス

提出時点でのデータでは 6 つのクラスを使用している。先にも述べたが、基本的な処理を行う上で使用しているのは database・system・disp・edit の 4 つのクラスである。ここでは残る cols クラスと base クラスに関しての処理を考える。

cols クラスに関してはデータベースからデータを取り出した際に項目番号を設定するための、補助的な役割を持つクラスである。このクラス内では定数を定義しているのみで、とくにメソッドなどは設定されていない。またいずれかのクラスを継承している、または継承されているクラスでもない、完全に独立した形となっている。今回のクラスの再編成にあたっても、このクラスに関しては用途的にも変更を行わないことが望ましいと考える。

base クラスは HTML タグをメソッドとして定義しているクラスである。現状では system クラスと disp クラスはこの base クラスを継承した形をとっており、system・disp クラス内のメソッドで表示に関する整形を行う際、HTML タグは base クラス内の HTML タグを規定したメソッドを呼び出して表示する形としている。そのため下記のようなコードとなる。

```
class base {
        // HTML タグの設定
        // p タグを設定する場合
        public function p($data)
        {
                return “<p>$data</p>¥n”;
        }
        //  （以下省略）
}
class system extends base {
        // 詳細表示の原型を設定するメソッドの場合
        public function shosai_disp()
        {
                //  （中略）
                $contentsData .= $this->p(nl2br($this->arrange($honbun)));
                //  （以下省略）
        }
```

```
}
```

このような HTML の p タグのようなタグを利用する場合に関しては、専用のメソッドを作り、それを呼び出して使用するメリットはほとんどないと考えられる。むしろコードが複雑化し、継承先のメソッド内での混乱を招く可能性すら考えられる。

しかし、今回の wiki の「その他設定」画面における input などの要素で、しかも下記のように書き換えが必要となった場合は有用かもしれない。wiki 制作中にあった実例を以下に挙げてみる。

```
class base {
	// type=”text”が予め規定された input タグを設定する場合
	public function input_text($name, $value, $size)
	{
		return “<input type=’text’ name=’$name’ value=’value’ size=’$size’>”;
	}
}
class system extends base {
	// 「その他設定」を表示するメソッド場合
	public function setupList()
	{
		// （中略）
		$this->input_text(“WikiTitle”, “$setting[WikiTitle]”, “80”)
		// （中略）
		$this->input_text(“footer”, “$setting[footer]”, “80”)
		// （以下省略）
	}
}
```

input_text()メソッドは input 要素に関する設定を行うものだが、type=”text”は決定事項としており、name・value・size の各属性に関する値を引数として設定している。それをインスタンス化する際、size 属性に関わる引数部分には 80 が与えられていた。size=”80”と指定されていたのである。初めは wiki のデザインに関しては全体を包括する div 要素を横幅を 1000px で固定していたため問題は発生しなかったが、固定値をやめ、ブラウザのサイズに合わせて横幅を変更する仕様に変更した際、size が固定値となっていてはデザインが崩れてしまうという問題が発生した。このため横幅をブラウザのサイズに合わせた相対的な値に変更する必要が生じた。この際に setupList()メソッドには手を加えず、input_text()メ

ソッドにクラスを設定して、スタイルシートからサイズを指定するように書き換えるだけで問題に対処することが出来た。具体的には、下記の通りである。

```
【input_text()メソッド】
public function input_text() {
        “<input type=’text’ name=’$name’ value=’$value’ class=’set’>”;
}

【スタイルシート】
input.class {
        width：98%;
}
```

input_text()メソッドの中身を“<input type=’text’ name=’$name’ value=’value’ class=’set’>”;とした上で、CSS 上で”input.class { width:98%; }”とする処理を追加した。

仮に setupList()メソッド内で使用している input 要素が直接その場で”<input type=”text” name=”WikiTitle” value=”$setting[WikiTitle]” size=”80” />”と記述されていた場合、書き換える対象が 2 か所になる。今回の場合は 2 か所なのでそれほどの手間とはならないかもしれないが、もし CSS 設定画面のように、input 要素が 10 個以上並んだ状態となっていると 1 つずつ書き換えることとなり、非常に面倒である。

ここで注意しておかなければならないのは、今回 input_text()メソッドは setupList()メソッドでしか使用されていなかったため、たまたまうまくいったという点である。もし仮にこれが別のメソッドでも使用されていた場合、input_text()メソッドを書き換えるとその別のメソッドの内部の input に関しても変更されてしまうのである。

これらの点を踏まえて再度検討すると以下のメリットとデメリットが見えてくる。

メリット

- 書き換えが発生した場合に書き換えが簡単にできる。

デメリット

- 単純な要素に関しては使用するとソースコードが複雑化する。
- 書き換えは簡単だが、複数メソッドで使用しているとむしろ混乱を引き起こす。

ここから考えられるのは、単純な要素に関しては表示するメソッド内で直接記述したほうがよいが、複雑な処理を行う可能性のある要素に関してはタグを使用用途に合わせて細かくメソッドとして定義しておくと後々書き換えが発生した場合に対処しやすい、ということである。今回の wiki の場合においては、基本的にはほぼすべてのタグはそのまま表示

する場所で記述したほうがよい。しかし input 要素に関しては各処理ごとに専用のメソッドを設けるべきだと私は考える。

例えば input 要素は記事編集画面のタイトル・その他設定・CSS の設定の各画面で用いられるが、それぞれ inputEdit()・inputSetting()・inputCSS()の各メソッドを作成すれば先に述べた機能を実装することが可能となる。

クラス一覧

予想されるクラスは以下の通りである。一段下がっているものに関しては、その上位のクラスを継承していることとする。また、各クラスの説明に書かれている文章に関しては基本的にそのクラスのメソッドの中身となる。

通常のクラス

- ○ wiki クラス
  - ➢ user クラス
    - ✧ kiji_ichiran クラス
    - ✧ kiji_shosai クラス
      - ● normal_kiji_shosai クラス
      - ● index_kiji_shosai クラス
    - ✧ new_edit クラス
    - ✧ normal_edit クラス
    - ✧ search クラス
      - ● search_shosai クラス
  - ➢ setting クラス
    - ✧ setting_top クラス
    - ✧ setting_kiji_ichiran クラス
    - ✧ setting_kiji_shosai クラス
    - ✧ setting_comment_ichiran クラス
    - ✧ setting_comment_shosai クラス
    - ✧ setting_other クラス
    - ✧ setting_style クラス
- ○ database クラス
  - ➢ database_kiji クラス
  - ➢ database_comment クラス
  - ➢ database_setting クラス
- ○ updateCss クラス

静的メンバを持つクラス

- ○ base クラス
- ○ cols クラス
- ○ edit クラス
- ○ getCommentData クラス
- ○ getKijiData クラス

wiki クラス

wiki 全体で共通の処理を規定するもの。メソッド一覧の生成やボックス構造の生成・データベースの定義・接続などを行う。ただし、ボックス構造の中で表示する内容に関してはページごとに異なる可能性があるので、ここでは定義を行わない。

user クラス

wiki クラスを継承しており、一般ユーザーが閲覧する際に使用するページ用に作成されるもの。一般ページで利用するタイトルやフッター、サイドメニューなどをデータベースから呼び出す。

kiji_ichiran クラス

user クラスを継承しており、通常画面での記事の一覧表示を行うためのクラス。記事データの取得にあたって、getKijiData クラスからデータを受け取る。その際インスタンス化は不必要。このクラスでは表示も行う。

kiji_shosai クラス

user クラスを継承している、記事を一件だけ選んで表示させるクラスに継承させるためのクラス。記事データの読み込みを getKijiData クラスいよって行う。表示はさらに継承された下位のクラスで行う。

normal_kiji_shosai クラス

kiji_shosai クラスを継承した、通常の記事詳細表示を行う際に用いるクラスである。記事に関するデータは kiji_shosai クラスで受信しているが、コメントデータはこのクラスで、getCommentData クラスより受け取る。

index_kiji_shosai クラス

kiji_shosai クラスを継承し、トップページ(index.php)で表示を行うためにしたもの。コメントの表示部分を持たないため、上の normal_kiji_shosai と区別される。

new_edit クラス

新規投稿を行うためのクラスで、wiki クラスを継承している。編集画面に関しては edit クラスを利用する。

normal_edit クラス

既に投稿されている記事を編集するためのクラスで wiki クラスを継承している。編集画面

に関しては edit クラスを利用し、編集元となる記事は getKijiData クラスから受信する。

search クラス

wiki クラスを継承して、検索結果の一覧表示を行う。送信された検索ワードをもとに SQL を生成し、クエリを実行、受け取ったデータをテーブルで表示する。検索ワードはメンバ変数に代入しておき、これを継承した search_shosai クラスでも利用できるようにする。

search_shosai クラス

search クラスを継承して検索結果の詳細表示を行う。検索結果一覧からユーザーが任意の記事を選択し、詳細表示させる。この際、id を利用して getKijiData クラスからデータを取得するが、取得した結果に対して search クラスで代入された検索ワードを照らし合わせ、該当箇所を span 要素で囲み、class 属性を割り振る。これによって、検索ワードへの色付けが可能となる。

setting クラス

wiki クラスを継承して作成する、管理設定画面の骨格となるもの。管理設定画面用の左サイドメニューの作成、諸設定データの受信などを行う。諸設定データなどはメンバ変数に代入し、継承先で使用できるようにしておく。

setting_top クラス

setting クラスを継承し、管理設定画面のトップページを作る。

setting_kiji_ichiran クラス

setting クラスを継承して、管理設定画面での記事の一覧表示を行うクラス。記事データに関しては getKijiData クラスを利用して受信し、結果をテーブルで表示する。

setting_kiji_shosai クラス

setting クラスを継承し、管理設定画面での記事詳細表示を行うためのクラス。詳細表示と言っても編集画面的なニュアンスも持ち、getKijiData から記事データを受信するとともに edit クラスから編集画面も呼び出しておく。

setting_comment_ichiran クラス

setting クラスを継承して、投稿されたコメントの一覧表示を行う。管理設定画面での記事一覧表示と同様に getCommentData クラスを利用してコメントの一覧を受信し、テーブル表示を行う。

setting_comment_shosai クラス

setting クラスを継承して、コメントの詳細表示を行う。コメントに関しては記事と違い、編集機能を持たせないため、getCommentData クラスからデータを受信して表示を行うのみである。

setting_other クラス

setting クラスを継承し、wiki タイトル・フッターに表示する文字列の規定などを行うためのクラス。編集する文字列やデータなどは setting クラスで既に受信済みである。

setting_style クラス

setting クラスを継承し、wiki のスタイル設定を行う画面を作るもの。スタイル設定で必要となる正規表現のパターンを定数として定義する他、正規表現による文字列の抽出、配列の定義など必要なメソッドを作成するのはこのクラス。

database クラス

database を実行してデータを登録する際に使用するクエリの継承元となるクラス。クエリの実行を行うメソッドと、SQL を格納するメンバ変数を持つ。

database_kiji クラス

database クラスを継承し、送られてきた記事データ・ID の受信と SQL の生成するクラス。

database_comment クラス

database クラスを継承し、送られてきたコメントデータ・ID の受信と SQL を生成するクラス。

database_setting クラス

database クラスを継承し、送られてきた諸設定データを受信して SQL を生成するクラス。

updateCss クラス

データベースを利用しないデータ保存を行う際に用いる fopen などを規定しているが、今回ファイルでデータを保存するのはスタイルシートのみなので、スタイルシートの更新に特化した形としている。

base クラス

HTML のタグを静的なメソッドとして持つ。タグに関しては単純なものはメソッド化しない。処理が複雑化することが見込まれる input 要素などに関してのみメソッド化することが理想である。使用する際もインスタンス化は不要。

cols クラス

データベースから値を取り出した際に項目番号を設定するために用いる補助的なクラス。定数のみでメソッドは持たない。定数に関しては今回使用するデータベースのすべての項目に関して規定されている。静的でインスタンス化不要。

edit クラス

各ページで利用する編集画面を生成するために利用する。記事の編集はタイトル・本文の入力欄を作るのに使用される。編集画面は先述の normal_edit クラスや new_edit クラス、setting_kiji_shosai クラスなどである。静的メソッドでインスタンス化は不要。

getCommentData クラス

コメントの呼び出しを行うための静的メソッドを集めたクラス。コメントも単純に全件呼び出す以外に、kiji_id に応じた呼び出し、comment テーブルの id フィールドによる呼び出し、件数のチェックなどが必要である。インスタンス化は不要。

getKijiData クラス

記事の呼び出しを行うための静的メソッドを集めたクラス。記事の全件取得、ID に応じた取得のほかに、件数のチェックなどの機能が必要である。インスタンス化は不要。