GoF本における分類

振る舞い＋オブジェクト

はじめに

ここではMementoパターンについて説明します。

「memento」とは聞き慣れない単語と思いますが、「記念品」「形見」「記憶」「思い出」といった意味があります。Mementoパターンは「Snapshotパターン」とも呼ばれますが、これなら分かりやすいでしょうか。Mementoパターンとは、ある時点のオブジェクトの記憶を保存し、あとで思い出せるようにするパターンです。

では、早速Mementoパターンを見ていきましょう。

たとえば

表計算ソフトやワードプロセッサ、テキストエディタなどのデスクトップアプリケーションを考えてみましょう。

ほとんどのアプリケーションでアンドゥ（Undo：やり直し）がサポートされています。アンドゥとは「ある状態を保存しておいて、その状態に戻せるようにしておく」機能です。また、リドゥ（Redo：やり直しのやり直し）をサポートしたアプリケーションも多いですね。リドゥも広い意味でアンドゥと同じと考えて良いと思います。なぜなら、リドゥは「アンドゥする前の状態を保存しておいて、その状態に戻せるようにしておく」機能だからです。

それでは、アンドゥやリドゥという機能はどういう風に実現しているのでしょうか？

オブジェクト思考的に考えてみると、「ある状態」というのはオブジェクトとして表せそうですね。そうすると、ある状態のオブジェクトをそのままどこかに保存しておき、必要なときにそのオブジェクトに置き変えてしまえばアンドゥが実現できそうです。

ここで言っている「状態」とはオブジェクト内部の状態、つまりオブジェクト内部に保持している値のことです。ですので、アンドゥを実現するために保存しておく必要があるのは、内部に保持している値そのもの、ということで良さそうです。ただし、アンドゥして期待した状態に復元させるには、保存した値を厳重に管理しなければならないでしょう。

いかがでしょうか？何となくイメージできましたか？

Mementoパターンは、これまで説明してきた内容そのままに、アンドゥを実現するパターンなのです。

Mementoパターンとは？

Mementoパターンはオブジェクトの振る舞いに注目したパターンで、オブジェクトのスナップショットを採ることを目的としています。

GoF本では、Mementoパターンの目的は次のように定義されています。

> カプセル化を破壊せずに、オブジェクトの内部状態を捉えて外面化しておき、オブジェクトを後にこの状態に戻すことができるようにする。

Mementoパターンでは、ある時点のオブジェクトの状態を別のオブジェクトとして保存しておき、状態が変化した場合でも、その時の状態に戻すことを可能にするパターンです

Mementoパターンは、あるオブジェクトの記憶を保存するための専用のクラスと、その保存を管理するクラスから構成されます。

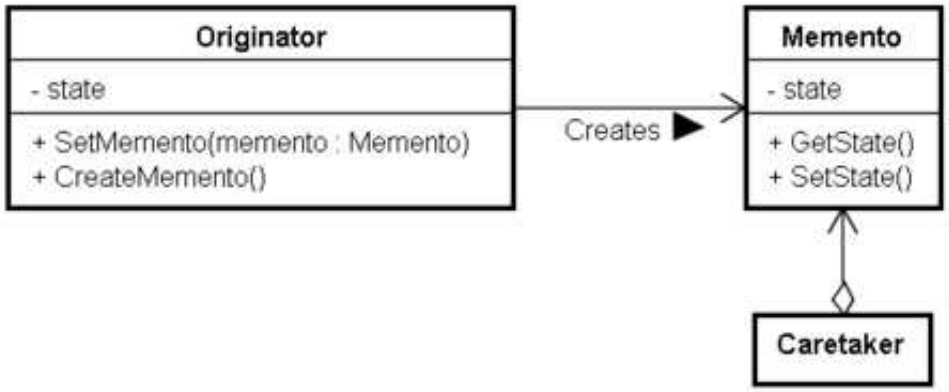
あるオブジェクトが記憶を保存する場合は、保存用のクラスをインスタンス化し、自分の記憶を詰め込みます。保存する記憶は、必要となる一部だけで構いませんが、他のクラスから記憶を書き換えられないよう注意する必要があります。そして、管理用のクラスに渡し、後で取り出せるよう管理してもらいます。逆に、記憶を戻したいときは、管理クラスに渡した記憶オブジェクトを取得し、自分自身に渡して記憶を復元します。管理クラスは、その記憶オブジェクトを操作したり、中を覗いたりしません。あくまで渡された記憶オブジェクトを管理するだけの役目を持ちます。

Mementoパターンは、状態を生成するクラスとその履歴の管理をするクラスを分離するパターンとも言え

ますね。

Mementoパターンの構造

Mementoパターンのクラス図と構成要素は、次のとおりです。



- Mementoクラス

Originatorオブジェクトの記憶を保持する記憶用クラスです。Originatorオブジェクトの内部状態を保持します。厳密には、Originator以外のオブジェクトによってアクセスさせないようにする必要があります。理想的には、Mementoオブジェクトはそのオブジェクトを生成したOriginatorからのみアクセスできるようにすべきです。つまり、Caretakerオブジェクトなどの他のオブジェクトによって、内部状態を変化させないようにしなければなりません。保存した記憶の内容が変わってしまうからです。しかし残念ながら、PHP5には「特定のクラスだけにアクセスを許可する」といった制御機構はありません。これを回避するには、本来のMementoパターンの構造を若干変更する必要があります。

- Originatorクラス

オブジェクトの内部状態を保存される側のクラスです。内部状態をMementoオブジェクトに詰め込み、それを返すメソッドを持ちます。

- Caretakerクラス

Mementoオブジェクトを管理するクラスです。

Mementoパターンの適用例

Mementoパターンの適用例として、1行コメントを入力する小さなメモ帳を作ってみましょう。

このメモ帳はコメントを登録できるだけでなく、ある時点の状態を保存しておくことができます。また、その状態に戻すこともできる、というものです。

まずは、DataSnapshotクラスから見ていきましょう。DataSnapshotクラスはMementoクラスに相当するクラスで、「記憶」であるひとつひとつのコメントを保存します。コンストラクタとコメントを取り出すgetCommentメソッドがprotectedとして定義されていますが、これについては次のDataクラスで説明しています。

その他は、特に難しいところはないと思います。

DataSnapshotクラス（DataSnapshot.class.php）

```
<?php
/**
 * Mementoに相当する
 */
class DataSnapshot
{

    private $comment;

    protected function __construct($comment)
    {
        $this->comment = $comment;
```

```
    }

    protected function getComment()
    {
        return $this->comment;
    }
}
```

次に、メモ帳のデータを表すDataクラスです。ここではDataSnapshotクラスのサブクラスとして定義しています。また、finalクラスとなっていることにも気づいたでしょうか？

これは構成要素のところでも説明しましたが、本来Mementoオブジェクトはそのオブジェクトを生成したOriginatorからのみアクセスできるようにする必要があります。さもないと、「記憶」であるDataSnapshotオブジェクトの内容が他のクラスから変更されてしまう可能性があるためです。

これを回避するためには、PHP5では継承とprotectedメソッドを組み合わせる必要があります。これにより、Mementoオブジェクトであるdata Snapshotオブジェクトのメソッドには、Dataクラス以外からアクセスすることはできなくなり、記憶の保全が保証されます。その代わり、クラスが大きくなってしまいますので注意が必要です。

その他、スナップショットを生成するtakeSnapshotメソッドやスナップショットから状態を復元するrestoreSnapshotメソッドも用意されています。

Dataクラス(Data.class.php)

```
<?php
require_once 'DataSnapshot.class.php';

/**
 * Originatorに相当する
 */
final class Data extends DataSnapshot
{
    private $comment;

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        $this->comment = array();
    }

    /**
     * Mementoを生成する
     */
    public function takeSnapshot()
    {
        return new DataSnapshot($this->comment);
    }

    /**
     * Mementoから復元する
     */
    public function restoreSnapshot(DataSnapshot $snapshot)
    {
        $this->comment = $snapshot->getComment();
    }

    public function addComment($comment)
    {
        $this->comment[] = $comment;
    }

    public function getComment()
    {
        return $this->comment;
    }
}
```

次はCaretakerクラスに相当するDataCaretakerクラスです。このサンプルではスナップショットの保存

先としてPHPのセッション機能を利用していますが、セッションの開始やセッションとのデータのやりとりをおこなっています。

DataCaretakerクラス(DataCaretaker.class.php)

```
<?php
/**
 * Caretakerに相当する
 */
class DataCaretaker
{

    public function __construct()
    {
        if (!isset($_SESSION)) {
            session_start();
        }
    }

    public function setSnapshot($snapshot)
    {
        $this->snapshot = $snapshot;
        $_SESSION['snapshot'] = $this->snapshot;
    }

    public function getSnapshot()
    {
        return (isset($_SESSION['snapshot']) ? $_SESSION['snapshot'] : null);
    }
}
```

最後にクライアント側のコードです。コメントの一覧と入力用HTMLフォームの表示を表示します。

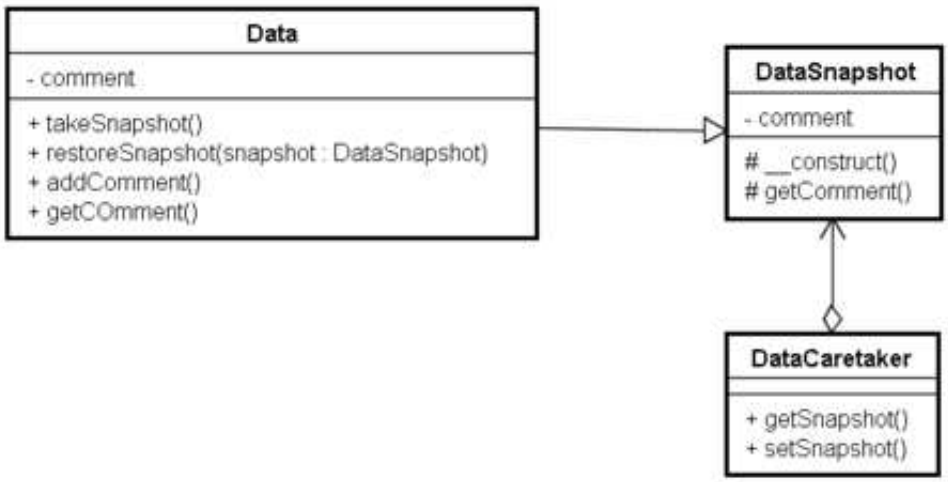
コメントの状態のスナップショットを取る場合は、Dataオブジェクトからスナップショットを作成し、DataCaretakerオブジェクトに渡して保存してもらっています。また、状態を復元する際は、逆にDataCaretakerオブジェクトから「記憶」を取り出し、Dataオブジェクトに渡しています。

クライアント側コード(memento_client.php)

```
<?php
require_once 'Data.class.php';
require_once 'DataCaretaker.class.php';

session_start();

$caretaker = new DataCaretaker();
$data = isset($_SESSION['data']) ? $_SESSION['data'] : new Data();

$mode = (isset($_POST['mode'])? $_POST['mode'] : '');

switch ($mode) {
    case 'add':
        /**
         * コメントをDataオブジェクトに登録する
         * 現時点のコメントはセッションに保存している事に注意
         */
        $data->addComment((isset($_POST['comment']) ? $_POST['comment'] : ''));
        break;
    case 'save':
        /**
         * データのスナップショットを取り、DataCaretakerに依頼して
         * 保存する
         */
        $caretaker->setSnapshot($data->takeSnapshot());
        echo '<font style="color: #dd0000;">データを保存しました。</font><br>';
        break;
    case 'restore':
        /**
         * DataCaretakerに依頼して保存したスナップショットを取得し、
         * データを復元する
         */
```

```
        $data->restoreSnapshot($caretaker->getSnapshot());
        echo '<font style="color: #00aa00;">データを復元しました。</font><br>';
        break;
    case 'clear':
        $data = new Data();
}

/**
 * 登録したコメントを表示する
 */
echo '今までのコメント';
if (!is_null($data)) {
    echo '<ol>';
    foreach ($data->getComment() as $comment) {
        echo '<li>'
        . htmlspecialchars($comment, ENT_QUOTES, mb_internal_encoding())
        . '</li>';
    }
    echo '</ol>';
}

/**
 * 次のアクセスで使うデータをセッションに保存
 */
$_SESSION['data'] = $data;
?>
<form action="" method="post">
コメント：<input type="text" name="comment"><br>
<input type="submit" name="mode" value="add">
<input type="submit" name="mode" value="save">
<input type="submit" name="mode" value="restore">
<input type="submit" name="mode" value="clear">
</form>
```

このサンプルコードのクラス図は、次のようになります。



Mementoパターンのメリット

Mementoパターンのメリットとしては、以下のものが挙げられます。

- オブジェクトの状態をある時点に戻すことができる

Mementoパターンを適用する目的でもあります。サンプルでは、1つ前の状態にしか戻せませんが、複数の記憶を管理することもできます。

- Originatorクラスを単純なものにする

Caretakerクラスに記憶を管理させることで、Originatorクラスを単純にすることができます。

適用例の実行結果

Mementoパターンを適用したサンプルの実行結果です。まずは、コメントを3つ追加し保存した状態です。

続けて、もう1つコメントを追加すると次のようになります。

ここで、「restore」ボタンをクリックし、保存した状態に戻すと次のようになります。

Mementoパターンのオブジェクト指向的要素

「オブジェクト指向的な要素」という視点から見ると、Mementoパターンはちょっと特殊な形をしています。ここでは、「クラスが負う責任」と「APIの公開」について見ていきましょう。

まずは、クラスが負う責任についてです。Mementoパターンでは、記憶オブジェクトを作るOriginatorクラスと、それを管理するCaretakerクラスが分かれています。一見、同じクラスで実装してしまっても問題がなさそうに見えます。

では、それぞれのクラスが負っている責任を見てみましょう。

Originatorクラスの責任は、Mementoクラスのインスタンスを生成することと、Mementoオブジェクトから自分の状態を元に戻すことです。

一方、Caretakerクラスの責任は、Mementoオブジェクトを管理することです。また、いつ記憶を取るか、いつ記憶を戻すかを決めます。

OriginatorクラスとCaretakerクラスの責任

クラス	責任
Originator	Mementoクラスのインスタンスを生成する。Mementoオブジェクトから自分の状態を元に戻す
Caretaker	Mementoオブジェクトを管理する。記憶を取るタイミングを管理する。

このように「クラスが負う責任」ごとに分けておくと、「記憶を複数保存したい」とか「記憶の保存方式を変更したい」といった場合、Caretakerクラスの変更だけで済むようになります。また、記憶として保存したい情報が増えた場合は、Caretakerクラスの週性は必要ありません。

次に「APIの公開」について見ていきましょう。

Mementoパターンでは、どのクラスが「記憶」にアクセスできるか、またそれをどのように保証するかが重要になります。本来のMementoパターンでは、Mementoクラスは2種類のアクセス制御を行います。1つは、インスタンス化や内部情報へのアクセスを許可しない「narrowインターフェース」、もうひとつは、それらを許可する「wideインターフェース」で、記憶を作るOriginatorクラスだけに公開し