Mediator ～すべては相談役が知っている

絶版に伴い、校正前の原稿テキストを公開したものです。基本的に原稿そのままをHTML形式に変換したものですので、誤字/脱字、説明不足の箇所もあるかも知れませんがご了承ください。初出：「PHPによるデザインパターン入門」（下岡秀幸/畑勝也/道端良 著，秀和システム，ISBN4-7980-1516-4，2006年11月23日発売）

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Mediatorパターンについて説明します。

「mediator」とは「仲介者」「調停者」という意味です。

Mediatorパターンは、クラスどうしの複雑なやりとりを一極集中するためのパターンです。

では、早速見ていきましょう。

たとえば

突然ですが、みなさんは無線LANをご存じでしょうか？職場や学校、ご自宅でも使っている型は多いと思います。最近のノートパソコンはまず間違いなく無線LANが使えるようになっていますね。

それでは、無線LANの接続形態にはどの様なものがあるでしょうか？それは、アドホックモードとインフラストラクチャモードです。前者はアクセスポイントを経由しないで、パソコンどうしでデータ通信をおこなうのに対し、後者はアクセスポイントを経由して他のパソコンとデータ通信をおこないます。

では、話を戻しましょう。オブジェクト指向プログラミングでは、お互いにやりとりをおこなうオブジェクトの集まりとしてプログラミングをおこないます。この「やりとり」を実現する最も簡単な方法は何でしょうか？そう、他のオブジェクトのメソッドを直接呼び出すことです。

しかし、オブジェクトどうしがお互いにやりとりをしてしまっては都合が悪い場合があります。特にオブジェクトの数が増えた場合、オブジェクトどうしのやりとりがあちらこちらで発生することになります。その結果、オブジェクトどうしの関係が分かりにくくなります。また、やりとりのための処理はそれぞれのオブジェクトに存在していますので、非常にメンテナンスしづらい状況に陥ってしまいます。

これはまさにアドホックモードと同じ状態です。

ここで、オブジェクトどうしのやりとりを仲介してくれる「仲介者」がいた場合はどうでしょうか？つまり、オブジェクトどうしのやりとりをインフラストラクチャモードにするということです。

この場合、他のオブジェクトとのやりとりは必ず仲介者を通じておこなうことになり、どのオブジェクトとやりとりをおこなうかは、仲介者が面倒を見るようにします。こうすると、オブジェクトは「どのオブジェクトのどのメソッドを呼び出す」といった難しいことを考える必要はなく、仲介者にお願いするだけで他のオブジェクトとやりとりができます。また、オブジェクトどうしの具体的なやりとりの手順が仲介者に集中しますので、管理も容易になります。

Mediatorパターンは、このような場面で適用されるパターンです。

Mediatorパターンとは？

Mediatorパターンの目的は、GoF本では次のように定義されています。

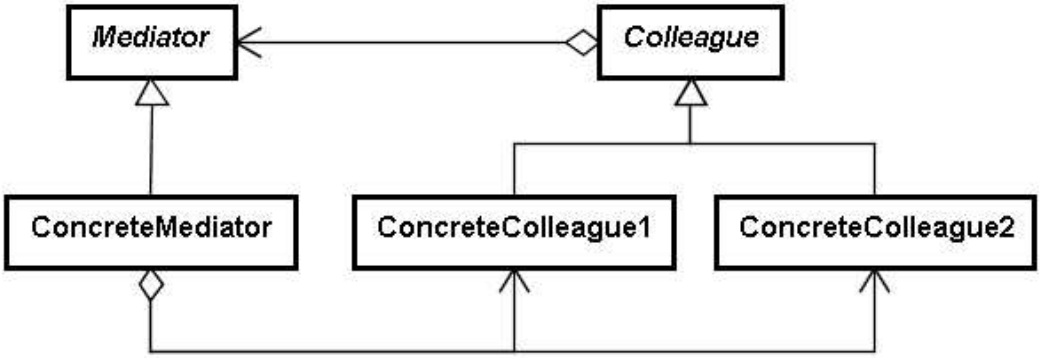
> オブジェクト群の相互作用をカプセル化するオブジェクトを定義する。
> Mediatorパターンは、オブジェクト同士がお互いを明示的に参照し合うことが
> ないようにして、結合度を低めることを促進する。
> それにより、オブジェクトの相互作用を独立に変えることができるようになる。

Mediatorパターンは、関連しあうクラス群のやりとりを1つのオブジェクトに集約させ、クラス群のやりとりの仲介者とします。こうすると、クラスどうしのやりとりを一極集中させる、つまり仲介者とクラスの関係を一対多にします。それぞれのクラスは、仲介者のクラスだけを知っていれば、他のクラスを知らなくてもお互いやりとりをおこなうことができます。このため、知らなければならないクラスの数が減ることになり、結果としてクラスどうしの関連を弱くできます。

また、それぞれのクラスに「どのタイミングでどのクラスのどのメソッドを呼び出す」といった具体的なやりとりの内容を記述する必要がなくなりますので、修正をおこなう場合に大きなメリットとなります。

Mediatorパターンの構造

Mediatorパターンのクラス図と構成要素は、次のようになります。



Mediatorクラス

仲介者のクラスです。Colleagueクラスからの通知を受け取るメソッドを定義します。

ConcreteMediatorクラス

Mediatorクラスのサブクラスで、Mediatorクラスで定義されたメソッドを実装します。・Colleagueクラス

Mediatorクラスに通知をおこなうクラスです。「colleague」とは「同僚」「仲間」という意味です。Mediatorクラスからの通知を受け取るメソッドを定義する場合もあります。

ConcreteColleagueクラス

Colleagueクラスのサブクラスです。内部にMediatorオブジェクトを保持しており、このMediatorオブジェクトを通じて他のConcreteColleagueクラスとやりとりをおこないます。

Mediatorパターンのメリット

Mediator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。

クラスどうしのやりとりを一極集中する

Mediatorパターンを適用すると、クラスどうしのやりとりはMediatorオブジェクトを通じておこなわれるようになります。これにより、やりとりの具体的な方法をMediatorクラスに持たせることができます。

見方を変えると、それぞれのColleagueオブジェクトどうしがどういった関係なのかが集中する事になり、不明瞭になりがちなColleagueオブジェクトどうしの関係を明確にする助けにもなります。

Colleagueオブジェクトの結びつきを緩くする

Mediatorクラスを通じて他のColleagueオブジェクトにアクセスすることで、Colleagueオブジェクトどうしの結びつきが緩くなり、MediatorクラスとColleagueクラスの関係は一対多になります。

Mediatorパターンの適用例

Mediatorパターンの適用例を見てみましょう。

ここでは、複数のユーザがチャットを使ってやりとりをしている模様を表すシンプルなアプリケーションを用意しました。

まずは、ユーザを表すクラスです。このクラスは、ColleagueクラスとConcreteColleagueクラスの両方に相当します。

●Userクラス(User.class.php)

```
<?php
require_once 'Chatroom.class.php';
?>
<?php
class User {
    private $chatroom;
    private $name;
    public function __construct($name) {
        $this->name = $name;
    }
    public function getName() {
        return $this->name;
    }
    public function setChatroom(Chatroom $value) {
         $this->chatroom = $value;
    }
    public function getChatroom() {
        return $this->chatroom;
    }
    public function sendMessage($to, $message) {
        $this->chatroom->sendMessage($this->name, $to, $message);
```

```
    }
    public function receiveMessage($from, $message) {
        printf('<font color="008800">%sさんから%sさんへ</font>： %s<hr>', $from, $this->getName(), $message);
    }
}
?>
```

特徴的なのは、Userクラスの中にチャットする場を表すChatroomクラスのインスタンスを保持していることです。sendMessageメソッドで他のユーザにメッセージを送信しますが、この時直接そのユーザオブジェクトにアクセスするのではなく、このChatroomオブジェクトに送信しています。

次は、チャットする場であるChatroomクラスです。このクラスは、MediatorクラスとConcreteMediatorクラスに相当しています。

Chatroomクラスでは、ユーザーを表すUserオブジェクトを保持し、入室しているユーザを管理しています。

●Chatroomクラス（Chatroom.class.php）

```
<?php
require_once 'User.class.php';
?>
<?php
class Chatroom {
    private $users = array();
    public function login(User $user) {
        $user->setChatroom($this);
        if (!array_key_exists($user->getName(), $this->users)) {
            $this->users[$user->getName()] = $user;
            printf('<font color="#0000dd">%sさんが入室しました</font><hr>', $user->getName());
        }
    }
    public function sendMessage($from, $to, $message) {
        if (array_key_exists($to, $this->users)) {
            $this->users[$to]->receiveMessage($from, $message);
        } else {
            printf('<font color="#dd0000">%sさんは入室していないようです</font><hr>', $to);
        }
    }
}
?>
```

また、ユーザどうしのメッセージの流れを管理しています。このアプリケーションでは、受け取ったメッセージを単純にユーザオブジェクトのreceiveMessageメソッドに渡しています。

次はクライアント側のコードです。

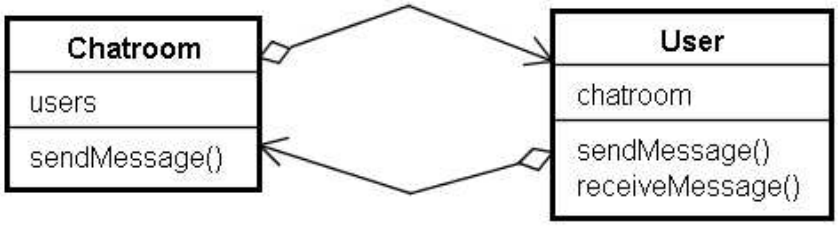
まず、Chatroomオブジェクトを生成し、それぞれのユーザを入室させています。そして、ユーザーどうしでメッセージのやりとりをおこなっています。

●クライアント側コード（mediator_client.php）

```
<?php
require_once 'Chatroom.class.php';
require_once 'User.class.php';
?>
<?php
    $chatroom = new Chatroom();

    $sasaki = new User('佐々木');
    $suzuki = new User('鈴木');
    $yoshida = new User('吉田');
    $kawamura = new User('川村');
    $tajima = new User('田島');

    $chatroom->login($sasaki);
    $chatroom->login($suzuki);
    $chatroom->login($yoshida);
    $chatroom->login($kawamura);
    $chatroom->login($tajima);

    $sasaki->sendMessage('鈴木', '来週の予定は？') ;
    $suzuki->sendMessage('川村', '秘密です') ;
    $yoshida->sendMessage('萩原', '元気ですか？') ;
    $tajima->sendMessage('佐々木', 'お邪魔してます') ;
    $kawamura->sendMessage('吉田', '私事で恐縮ですが・・・') ;
```

```
?>
```

最後にサンプルコードのクラス図を示しておきます。



Mediatorパターンのオブジェクト指向的要素

Mediatorパターンはオブジェクトどうしのやりとりを「カプセル化」するパターンです。

冒頭にも出てきましたが、オブジェクト指向プログラミングでは、お互いにやりとりをおこなうオブジェクトの集まりとしてプログラミングをおこないます。このやりとりしあうオブジェクトの数が少ない場合、その複雑さは問題にならないかも知れません。しかし、オブジェクトの数が少し増えただけでも、お互いの関連の数はあっという間に増えてしまいます。

Mediatorパターンでは、これを「ColleagueオブジェクトはMediatorオブジェクトとやりとりをする」という非常にシンプルな処理に置き換えて、Colleagueオブジェクトどうしの具体的なやりとりやColleagueクラスどうしの関連を隠蔽しています。

関連するパターン

Facadeパターン

Facadeパターンもクラスの集まりを対象にしたパターンです。

Mediatorパターンはクラスどうしのやりとり自体を抽出しますが、Facadeパターンはサブシステムを利用するためのAPIを抽出するパターンです。また、Mediatorパターンは構成しているオブジェクトと双方向のやりとりをおこなう場合がありますが、Facadeパターンではサブシステムに対して要求を送信する一方向のやりとりになります。

Observerパターン

Mediatorパターンに似たパターンとしてObserverパターンがあります。MediatorパターンはColleagueオブジェクトどうしのやりとりの仲介者を用意します。一方のObserverパターンでは、Observerクラス自身が処理をおこないます。

また、Observerパターンを適用することでColleagueオブジェクトとMediatorオブジェクトを通信させることができます。

まとめ

ここでは複雑なクラス間のやりとりを一元管理するMediatorパターンについて見てきました。