GoFパターン

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)

1995年、Erich Gamma氏、Richard Helm,氏 Ralph Johnson氏、John Vlissides氏の4名(GoF:Gang of Four;4人のギャングたち)により、ある一冊の書籍が世に送り出されました。通称「GoF本」と呼ばれる「Design Patterns : Elements of Reusable Object Oriented Software」(邦題:「オブジェクト指向における再利用のためのデザインパターン」/ソフトバンクパブリッシング/1999年)です。聞いたことのある方も多いかと思います。

この本では、23個のパターンが紹介されています。また、それぞれに名前が付けられており、「カタログ」として整理されています。また、単に「デザインパターン」という場合、この23個のパターンを指す場合もあり、非常に有名なパターンとして広く知られています。

GoF本では、23個のデザインパターンの目的を、次の3つのカテゴリに分類しています。

- オブジェクトの生成に関するパターン
- プログラムの構造に関するパターン
- オブジェクトの振る舞いに関するパターン

表にGoF本で名前を与えられた23個のパターンの名前とその目的をまとめます。

パターン名	目的
オブジェクトの生成に関するパターン	
Abstract Factory	互いに関連したり依存し合うオブジェクト群を、その具象クラスを明確にせず生成するためのインタフェースを提供する。
Builder	複合オブジェクトについて、その作成過程を表現形式に依存しないものにすることにより、同じ作成過程で異なる表現形式のオブジェクトを生成できるようにする。
Factory Method	オブジェクトを生成するときのインタフェースだけを規定して、実際にどのクラスをインスタンス化するかはサブクラスが決めるようにする。Factory Methodパターンは、インスタンス化をサブクラスに任せる。
Prototype	生成すべきオブジェクトの種類を原型となるインスタンスを使って明確にし、それをコピーすることで新たなオブジェクトの生成を行う。
Singleton	あるクラスに対してインスタンスが1つしか存在しないことを保証し、それにアクセスするためのグローバルな方法を提供する。
プログラムの構造に関するパターン	
Adapter	あるクラスのインタフェースを、クライアントが求める他のインタフェースへ変換する。Adapterパターンは、インタフェースに互換性のないクラス同士を組み合わせることができるようにする。
Bridge	抽出されたクラスと実装を分離して、それらを独立に変更できるようにする。
Composite	部分ー全体階層を表現するために、オブジェクトを木構造に組み立てる。Compositeパターンにより、クライアントは、個々のオブジェクトとオブジェクトを合成したものを一様に扱うことができるようになる。
Decorator	オブジェクトに責任を動的に追加する。Decoratorパターンは、サブクラス化よりも柔軟な機能拡張方法を提供する。
Facade	サブシステム内に存在する複数のインタフェースに1つの統一インタフェースを与える。Facadeパターンはサブシステムの利用を容易にするための高レベルインタフェースを定義する。
Flyweight	多数の細かいオブジェクトを効率よくサポートするために共有を利用する。
Proxy	あるオブジェクトへのアクセスを制御するために、そのオブジェクトの代理、または入れ物を提供する。
オブジェクトの振る舞いに関	

するパターン

Chain of Responsibility	1つ以上のオブジェクトに要求を処理する機会を与えることにより、要求を送信するオブジェクトと受信するオブジェクトの結合を避ける。受信する複数のオブジェクトをチェーン状につなぎ、あるオブジェクトがその要求を処理するまで、そのチェーンに沿って要求を渡していく。
Command	要求をオブジェクトとしてカプセル化することによって、異なる要求や、要求からなるキューやログにより、クライアントをパラメータ化する。また、取り消し可能なオペレーションをサポートする。
Interpreter	言語に対して、文法表現と、それを使用して文を解釈するインタプリタを一緒に定義する。
Iterator	集約オブジェクトが基にある内部表現を公開せずに、その要素に順にアクセスする方法を提供する。
Mediator	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に明示的に参照しあ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。
Memento	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。
Observer	あるオブジェクトが状態を変えたときに、それに依存するすべてのオブジェクトに自動的にそのことが知らされ、また、それらが更新されるように、オブジェクト間に一対多の依存関係を定義する。
State	オブジェクトの内部状態が変化したときに、オブジェクトが振る舞いを変えるようにする。クラス内では振る舞いの変化を記述せず、状態を表すオブジェクトを導入することでこれを実現する。
Strategy	アルゴリズムの集合を定義し、各アルゴリズムをカプセル化して、それらを変換可能にする。Strategyパターンを利用することで、アルゴリズムを、それを利用するクライアントからは独立に変更することができるようになる。
Template Method	1つのオペレーションにアルゴリズムのスケルトンを定義しておき、その中のいくつかのステップについては、サブクラスでの定義に任せる事にする。Template Methodパターンでは、アルゴリズムの構造を変えずに、アルゴリズム中のあるステップをサブクラスで定義する。
Visitor	あるオブジェクトを構造上の要素で実行されるオペレーションを表現する。Visitorパターンにより、オペレーションを加えるオブジェクトのクラスに変更を加えずに、新しいオペレーションを定義することができるようになる。

また、GoF本では、パターンを効率よく利用できるよう、利用の範囲でも分類しています。これは、それぞれのデザインパターンを、「クラス」と「オブジェクト」のどちらに適用するかを示すためです。

表は、利用範囲で分類した23個のパターンを示しています。

	生成	構造	振る舞い
クラス	Factory Method	Adapter	Interpreter、Template
オブジェクト	Abstract Factory、Builder、Prototype、Singleton	Adapter、Bridge、Composite、Decorator、Facade、Flyweight、Proxy	Chain of Responsibility、Command、Iterator、Mediator、Observer、State、Strategy、Visitor

「クラス」に分類されるパターンは、親クラスとサブクラスの関係を利用して、対象とする問題を解決するパターンになります。当然、この関係は継承を使用することになりますので、コーディングをおこなった時点でクラスどうしの関係が決まります。このような関係を「静的」と言います。

●クラスどうしの静的な関係

```
<?php
class SomeClass
{
        :
}

/**
 * この時点でクラスどうしの関係が決まる
 */
class AnotherClass extends SomeClass
{
        :
}
```

一方、「オブジェクト」に分類されるパターンの場合、オブジェクトどうしの関係を利用して、対象とする問題を解決するパターンになります。この場合、その関係を実行時に決定することができます。つまり、オブジェクトを生成する、具体的に

はnew演算子を使ってインスタンスを生成したり、アクセサメソッド（setXXXという名前のメソッド）を使ってオブジェクトを他のオブジェクトに挿入するタイミングなどでオブジェクトどうしの関係が決まります。このような関係を「動的」と言います。

●オブジェクトどうしの動的な関係(1)

```
<?php
        :
    /**
     * このタイミングでオブジェクトどうしの関係が決まる
     */
    $object = new SomeClass();
        :
```

●オブジェクトどうしの動的な関係(2)

```
<?php
        :
    $object = new SomeClass();
    $another_object = new AnotherClass();

    /**
     * このタイミングでオブジェクトどうしの関係が決まる
     */
    $object->setObject($another_object);
        :
```

また、表をよく見ると、Adapterパターンがクラス、オブジェクトのいずれの範囲にも分類されていることに気づくかと思います。実際に、Adapterパターンの実装方法には、静的・動的の2パターンが存在します。これについては、第3章で見ていきます。