Singletonパターン～いくつ作るかを制限する

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

生成＋オブジェクト

はじめに

ここではSingletonパターンについて説明します。

singletonとは「一枚札」「一つずつ起こるもの」といった意味を持つ単語です。

Singletonパターンは、生成するオブジェクトの数を1つに制限するためのパターンです。では、なぜオブジェクトの数を制限する必要があるのでしょうか？

早速、見ていきましょう。

たとえば

クラスのインスタンスはnew演算子を使って生成されます。たとえば、5回new演算子を使った場合、5つのインスタンスが生成されます。当然、1000回実行すると1000個のインスタンスが生成されます。

しかし、インスタンスを生成するという処理は、コストがかかる処理です。オブジェクトの使いまわしをしないで毎回newするのは、大きなコストがかかってしまうことを意味します。

また、「どうしてもインスタンスを1つしか生成したくない」といった場面も出てきます。たとえば、システムの設定を表現するクラスや、システム全体で共用するデータプーリングクラスなどです。

この場合、プログラミングする際に注意深くnew演算子を使うことで、1つしかインスタンスを生成させないようにすることもできます。しかし、それは「保証」されたものではありません。当然、何らかのミスや、それを知らない開発者が後からどんどんnewしていってしまうことも考えられます。

開発者が意識しなくても、あるクラスのインスタンスが1つしか存在しないことを「保証する」ために使われるデザインパターン。それが**Singleton**パターンです。

Singletonパターンとは？

Singleton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。

> 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。

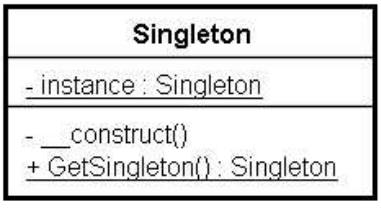
Singletonパターンは、オブジェクトの生成に関連するパターンです。

Singletonパターンを適用すると、インスタンスが1つしか生成されないことが保証されます。このため、開発者は「一度しかnewしてはならない」といった余計な事を考えずに済むようになります。

なお、PHP5の場合、オブジェクトを複製するための**clone**キーワードが用意されています。Singletonパターンを使う場合、このcloneキーワードへの対策が必要となります。具体的には、PHP5から追加された**__clone**メソッドをオーバーライドし、例外を発生、もしくは強制終了させます。

Singletonパターンの構造

Singletonパターンのクラス図と構成要素は、次のとおりです。



Singletonパターンの構成要素

Singletonクラス

Singletonパターンには、Singletonクラスしかありません。このクラスのコンストラクタは**private**になっています。このため、他のクラスから直接Singletonインスタンスを生成する事はできません。その代わり、ただ1つのインスタンスを返す**static**メソッドが用意されます。また、自分自身のインスタンス（**Singleton**インスタンス）を保持するための**static**変数を内部に持っています。

Singletonパターンのメリット

Singleton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。

インスタンスへのアクセスを制御する

Singletonパターンは、自分自身のインスタンスを内部に保持しています。また、そのインスタンスへのアクセス手段も限られているため、他のクラスからインスタンスへアクセスするための方法は、必然的に決まります。このため、クライアントからインスタンスへのアクセスを制御することができます。

インスタンスの数を変えることができる

これまでの説明では同時に存在できるインスタンスは1つでしたが、インスタンスの数を2つ以上に変えることもできます。この場合、GetInstanceメソッド(サンプルではgetInstanceメソッド)内の処理を変更するだけで済みます。

Singletonパターンの適用例

Singletonパターンの適用例を見てみましょう。

ここでは、Singletonパターンを適用したオブジェクトの動作や特徴を確認するためのサンプルを用意しました。

まずは、Singletonパターンを適用したSingletonSampleクラスから見ていきましょう。

SingletonSampleクラスは内部にIDを保持するだけの簡単なクラスです。

●SingletonSampleクラス(SingletonSample.class.php)

```php
<?php
class SingletonSample {

    /**
     * メンバー変数
     */
    private $id;

    /**
     * 唯一のインスタンスを保持する変数
     */
    private static $instance;

    /**
     * コンストラクタ
     * IDとして、生成日時のハッシュ値を作成
     */
    private function __construct() {
        $this->id = md5(date('r') . mt_rand());
    }

    /**
     * 唯一のインスタンスを返すためのメソッド
     * @return SingletonSampleインスタンス
     */
    public static function getInstance() {
        if (!isset(self::$instance)) {
            self::$instance = new SingletonSample();
            echo 'a SingletonSample instance was created !';
        }

        return self::$instance;
    }

    /**
     * IDを返す
     * @return インスタンスのID
     */
    public function getID() {
        return $this->id;
    }

    /**
     * このインスタンスの複製を許可しないようにする
     * @throws RuntimeException
     */
    public final function __clone() {
        throw new RuntimeException ('Clone is not allowed against ' . get_class($this));
    }
}
?>
```

このクラスで注目するのはコンストラクタ(__constructメソッド)とgetInstanceメソッドです。

SingletonSampleクラスはコンストラクタはprivateとして宣言されていますので、SingletonSampleクラス以外からインスタンス化することはできません。その代わり、getInstanceメソッドでSingletonSampleクラスをインスタンス化しています。これにより、SingletonSampleクラスをインスタンス化できるのを、SingletonSampleクラスのgetInstanceメソッドだけに限定できます。

また、SingletonSampleクラスはstatic変数$instanceを用意していますが、getInstanceメソッドではこの変数に生成した自分自身のインスタンスを保存しています。static変数は、1回目に呼び出された時のみ初期化され、その後スコープが移行しても値が初期化されず保持されるという特徴を持っています。その結果、getInstanceメソッドからは、最初にインスタンス化されたSingletonSampleオブジェクトがそのまま返されることになります。

なお、このサンプルでは、SingletonSampleクラスのメンバ変数として宣言していますが、getInstanceメソッドのローカル変数として宣言しても同様の効果を得られます。

●ローカル変数として宣言した場合のgetInstanceメソッド

```
<?php
                :
    /**
     * 唯一のインスタンスを返すためのメソッド
     * @return SingletonSampleインスタンス
     */
    public static function getInstance() {
        /**
         * 唯一のインスタンスを保持する変数
         */
        static $instance;

        if (!isset($instance)) {
            $instance = new SingletonSample();
            echo 'a SingletonSample instance was created !';
        }

        return $instance;
    }
```

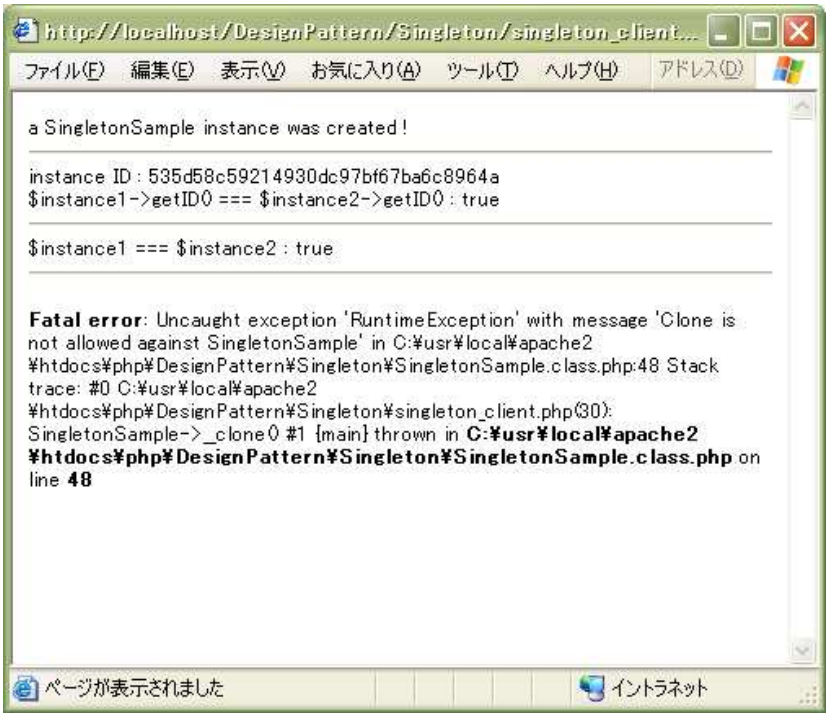
また、先の説明にも出てきましたが、いくらSingletonパターンを適用してもコピーされてしまっては意味がありません。そのため、**__clone**メソッドを呼び出された際に**Runtimeexception**を投げるようになっています。

このクラスを利用するクライアント側のコードは次のとおりです。

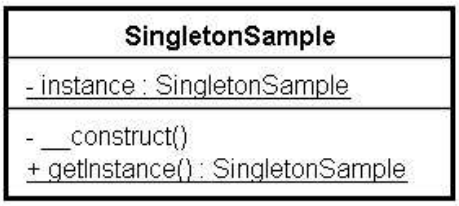
●クライアント側コード(singleton_client.class.php)

```
<?php
require_once 'SingletonSample.class.php';
?>
<?php
    /**
     * インスタンスを2つ取得する
     */
    $instance1 = SingletonSample::getInstance();
    sleep(1);
    $instance2 = SingletonSample::getInstance();

    echo '<hr>';

    /**
     * 2つのインスタンスが同一IDかどうかを確認する
     */
    echo 'instance ID : ' . $instance1->getID() . '<br>';
    echo '$instance1->getID() === $instance2->getID() : ' . ($instance1->getID() === $instance2->getID() ? 'true' : 'false');
    echo '<hr>';

    /**
     * 2つのインスタンスが同一かどうかを確認する
     */
    echo '$instance1 === $instance2 : ' . ($instance1 === $instance2 ? 'true' : 'false');
    echo '<hr>';

    /**
     * 複製できないことを確認する
     */
    $instance1_clone = clone $instance1;

?>
```

ここでやっていることは、SingletonSampleクラスのインスタンスを2つ用意し、生成されたIDの比較やオブジェクトとしての比較をおこなっています。また、cloneキーワードを使って複製を試みようとしています。この実行結果は次のようになります。



最後に、Prototypeパターンを適用したサンプルのクラス図を確認しておきましょう。



Singleton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、Singletonパターンはちょっと特殊な形をしています。「継承」も「ポリモーフィズム」も使っていません。あえて言えば、「カプセル化」を利用しているパターンです。

Singletonパターンでは、自分自身のインスタンスを内部に保持して管理しています。また、他のクラスからそのインスタンスへのアクセス方法も提供します。つまり、他のクラスからは公開されているアクセス方法でしかインスタンスを操作できなくなります。このため、Singletonクラスと他のクラスの独立性が高まることになります。この結果、Singletonクラス内部での仕様変更が他方に影響しなくなり、保守性や再利用性が高くなります。

関連するパターン

Abstract Factoryパターン、**Builderパターン**、**Factory Methodパターン**、**Prototypeパターン**、**Facadeパターン**

これらのパターンは、Singletonパターンと併用することで唯一のインスタンスとなるよう実装される場合が多いです。

まとめ

ここではインスタンスが唯一無二であることを保証するSingletonパターンを見てきました。GoFパターンの中で、唯一1つのクラスだけでパターンとなっていますね。