Web-разработка на PHP-технологиях

Курс лекций и семинаров для студентов,
желающих научиться основам Web-разработки на PHP

Осень-Зима 2014

Лекция №8

Паттерны ООП

Автор: Дмитрий Левин, Senior PHP Developer, Sibers

Sibers®

Паттерны ООП

Порождающие

Abstract Factory (Абстрактная фабрика)
Factory Method (Фабричный метод)
Singleton (Одиночка)
Prototype (Прототип)
Builder (Строитель)

Структурные

Adapter (Адаптер)
Bridge (Мост)
Composite (Компоновщик)
Decorator (Декоратор)
Facade (Фасад)
Flyweight (Приспособленец)
Proxy (Прокси)

Паттерны поведения

Chain of Responsibility (Цепочка обязанностей)
Command (Команда)
Interpreter (Интерпретатор)
Iterator (Итератор)
Mediator (Посредник)
Memento (Хранитель)
Observer (Наблюдатель)
State (Состояние)
Strategy (Стратегия)
Template Method (Шаблонный метод)
Visitor (Посетитель)

Паттерны ООП: **Singletone**

Шаблон проектирования Синглтон применяется в тех ситуациях, когда нужно получить единственный экземпляр класса.

Синглтон является одним из четырех ("Gang of Four") основных создающих шаблонов.

Синглтон обычно применяется в классах для баз данных, логирования, контроллеров (Front Controller) и объектах, определяющих запросы и ответы(Request и Response).

Паттерны ООП: **Singletone**

```php
class ExampleSingletone
{
    protected static $instance;

    protected function __construct(){}

    protected function __clone(){}

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new ExampleSingletone();
        }
        return self::$instance;
    }

}

$singleTone = ExampleSingletone::getInstance(); // object(ExampleSingletone)#1 (0) { }
$singleTone = new ExampleSingletone(); // Call to protected ExampleSingletone::__construct()
```

Sibers

Паттерны ООП: **Factory**

Шаблон Фабрика позволяет создавать объекты во время работы программы. Этот шаблон называется Фабрикой потому, что он отвечает за производство объекта. Параметризированная фабрика принимает в качестве аргумента имя класса, объект которого создается.

```php
class ExampleFactory
{
    // Параметризированный фабричный метод
    public static function factory($type)
    {
        if (include_once 'Drivers/' . $type . '.php') {
            $classname = 'Driver_' . $type;
            return new $classname;
        } else {
            throw new Exception('Драйвер не найден');
        }
    }
}
```

Sibers®

Паттерны ООП: **Factory**

```
class FormManagerFactory{
    public static function getForm($name){
        switch ($name){
            case "RegForm":
                return new RegForm();
                break;
            case "AuthForm":
                return new AuthForm();
                break;
            case "PostForm":
                return new PostForm();
                break;
        }
    }
}
```

Паттерны ООП: **Factory**

```php
abstract class Form{
    abstract public function generateForm();
}

class RegForm extends Form{
    public function generateForm() {
        echo "Generate registration form!";
    }
}

class AuthForm extends Form{
    public function generateForm() {
        echo "Generate authentication form!";
    }
}

class PostForm extends Form{
    public function generateForm() {
        echo "Generate posting form!";
    }
}
```

Паттерны ООП: **Factory**

```
echo FormManagerFactory::getForm('AuthForm')->generateForm();
echo FormManagerFactory::getForm('PostForm')->generateForm();

# Generate authentication form!
# Generate posting form!
```

Паттерны ООП: **Abstract Factory**

Абстрактная фабрика — Abstract Factory

Фабрика должна возвращать объекты. Но так как она абстрактная, то изначально нужно описать абстрактный класс фабрики, и абстрактные объекты которые она будет производить.

Далее мы описываем реализации классов фабрик на основе абстрактной фабрики и реализации класса объектов которые она производит.

Тем самым у нас есть различные варианты фабрик и объектов с одинаковыми интерфейсами, что может быть удобно.

Паттерны ООП: **Abstract Factory**

```
abstract class abstract_machine_factory {
    function generate_sedan() {
        return new machine('sedan');
    }

    function generate_universal() {
        return new machine('universal');
    }

    function generate_hatchback() {
        return new machine('hatchback');
    }
}
```

Паттерны ООП: **Abstract Factory**

```php
abstract class abstract_machine {
    protected $_param;

    function __construct($type = 'sedan') {
        $this->_param = array();
        $this->_param['type'] = $type;
    }

    function run() {
        echo 'brrrrrrrrrrrrr ...';
    }

    function get_all_param() {
        return $this->_param;
    }
}
```

Паттерны ООП: **Abstract Factory**

```
class ford_factory extends abstract_machine_factory {
    function generate_sedan() {
        return new ford_machine('sedan', 'ford');
    }

    function generate_universal() {
        return new ford_machine('universal');
    }

    function generate_hatchback() {
        return new ford_machine('hatchback');
    }
}
```

Паттерны ООП: **Abstract Factory**

```php
class ford_machine extends abstract_machine {
    function __construct($type = 'sedan') {
        parent::__construct($type);

        $this->_param['firm'] = 'ford';
    }

    function run() {
        echo 'rrrrrrrrrrrrrrrr ...';
    }
}
```

Паттерны ООП: **Abstract Factory**

```
$ford_factory = new ford_factory();
var_dump( $ford_machine = $ford_factory->generate_sedan() ) ;
//object(ford_machine)#2 (1) {
//["_param":protected]=> array(2) {
//["type"]=> string(5) "sedan" ["firm"]=> string(4) "ford"
//} }
```

Паттерны ООП: **Proxy**

Прокси (Proxy, Заместитель) относиться к классу структурных паттернов. Является суррогатом другого объекта и контролирует доступ к нему.

Наиболее частым применением паттерна прокси является ленивая загрузка (lazy load). «Тяжёлые» объекты не всегда разумно загружать в момент инициализации.

Более правильным решением будет загрузить его по первому требованию. Давайте рассмотрим это на примере некой обертки для удалённых (расположенных удалённо) файлов. Информацию о самом файле мы будем хранить в базе данных, а сам файл где-то на другом сервере.

```php
<?php
class RemoteFile
{
    protected $_fileId = 0;
    protected $_filepath = "";
    protected $_filesize = 0;
    protected $_filename = "";
    protected $_filedata = null;

    public function loadById($fileId)
    {
        $this -> _fileId = $fileId;
        $this -> _loadFromDatabase($fileId);
        $this -> _filedata = file_get_contents($this -> _filepath);
    }

    public function _loadFromDatabase($fileId)
    {
        $fileinfo = DbAdapter::loadFileInfo($fileId);
        $this -> _filepath = $fileinfo['path'];
        $this -> _filesize = $fileinfo['size'];
        $this -> _filename = $fileinfo['name'];
    }
    /**
     * @return int
     */
    public function getFileSize()
    {
        return $this -> _filesize;
    }
}
```

```php
$file = new RemoteFile();
$file->loadById(1);
var_dump( $file -> getFileSize() );
```

Паттерны ООП: **Proxy**

```php
class RemoteFileProxy extends RemoteFile
{
    /**
     * Load file by file id
     *
     * @param int $fileId
     */
    public function loadById($fileId)
    {
        // Мы загружаем информацию только из БД, а сам файл не грузим
        $this -> _loadFromDatabase($fileId);
    }
    /**
     * @return string
     */
    public function getFileContents()
    {
        if (null === $this -> _filedata) {
            $this -> _filedata = file_get_contents($this -> _filepath);
        }
        return $this -> _filedata;
    }
}
```

Паттерны ООП: **Observer**

Наблюдатель (Observer) — поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents), «издатель-подписчик» (Publisher-Subscriber).

Создает механизм у класса, который позволяет получать оповещения от других классов об изменении их состояния, тем самым наблюдая за ними.

Шаблон состоит из трёх основных компонентов – слушателя событий, подписчика на события и объекта события.

Заключение:

В восьмой лекции мы определили три большие категории паттернов ООП и рассмотрели подробно некоторые паттерны из каждой категории

В следующей лекции:

PHP, AJAX, JSON