РЕАЛИЗАЦИЯ ШАБЛОНА ПРОЕКТИРОВАНИЯ «НАБЛЮДАТЕЛЬ» НА PHP

Козырев Егор Алексеевич

студент обучающейся по направлению подготовки 02.03.02 Фундаментальная информатика и информационные технологии 4 курс, федеральное государственное автономное образовательное учреждение высшего образования «Белгородский государственный национальный исследовательский университет»
308015, РФ, г. Белгород, ул. Победы, 85
E-mail: kozyrev_e@bsu.edu.ru

Бурданова Екатерина Васильевна

канд. техн. наук, федеральное государственное автономное образовательное учреждение высшего образования «Белгородский государственный национальный исследовательский университет»
308015, РФ, г. Белгород, ул. Победы, 85
E-mail: burdanova@bsu.edu.ru

THE IMPLEMENTATION OF DESIGN PATTERN "THE OBSERVER" IN PHP

Yegor Kozyrev

student learning in the direction of training 02.03.02 Fundamental Informatics and information technologies 4 course, Federal state Autonomous educational institution of higher professional education "Belgorod state national research University»
308015, Russia, Belgorod, Pobeda St., 85

Ekaterina Burdanova

cand. Techn. of Sciences, Federal state Autonomous educational institution of higher professional education "Belgorod state national research University»
308015, Russia, Belgorod, Pobeda St., 85

АННОТАЦИЯ

В статье описывается способ решение не нетривиальных задач, когда по причине присутствия в системе большого количество объектов, находящихся в согласованном состоянии, «неосторожная» работа с ними может привести к нарушению целостности данных и корректности работы системы целиком. Во избежание подобного рода ситуаций стало необходимо отслеживать все важные манипуляции, способные спровоцировать нежелательные последствия. Автор пришел к выводу, что доработка стандартных методов, в данном случи, путем редактирования фалов системы, дает плохой результат, т.к. нарушает саму концепцию ядра, которое должно быть гибким и предоставляло бы инструменты для реализации различного нестандартного функционала. Показано, что для решения поставленной задачи лучше всего подходит шаблон проектирования «Наблюдатель», принцип работы которого заключается в механизме асинхронного оповещения экземпляров одного класса об изменении их состояния других объектов, таким образом наблюдая за ними. В статье подробно описывается пример реализации шаблона проектирования «Наблюдатель». Приводится описание созданных классов и методов, показаны листинги. Приводимая в статье реализация шаблона проектирования «Наблюдатель» хороша своей простотой и универсальностью. Это решение легко интегрируется в любой проект и не зависит от его структуры, функционала и прочих особенностей. Оно позволяет создавать неограниченное количество ранее непредусмотренных обработчиков, не затрагивая уже готовые компоненты системы.

ABSTRACT

The article describes a method of solving non-trivial tasks, when due to the presence in the system of a large number of objects in a consistent state, "careless" work with them can lead to violation of data integrity and correctness of the system as a whole. In order to avoid such situations, it has become necessary to monitor all important manipulations that can provoke undesirable consequences. The author came to the conclusion that the revision of standard methods, in this case, by editing the files of the system, gives a bad result, because. it violates the very concept of the kernel, which should be flexible and provide tools for the implementation of various non-standard functionality. It is shown that the design pattern "Observer" is best suited for solving the problem, the principle of which is the mechanism of asynchronous notification of instances of one class about changes in their state of other objects, thus observing them. The article describes in detail an example of the implementation of the design template "Observer". The description of the created classes and methods is given, the listings are shown. The implementation of the "Observer" design pattern given in the article is good

Библиографическое описание: Козырев Е.А., Бурданова Е.В. Реализация шаблона проектирования «Наблюдатель» на PHP // Universum: Технические науки: электрон. научн. журн. 2018. № 6(51). URL: http:///7universum.com/ru/tech/archive/item/6083

for its simplicity and versatility. This solution is easily integrated into any project and does not depend on its structure, functionality and other features. It allows you to create an unlimited number of previously unintended handlers without affecting the ready-made components of the system.

Ключевые слова: «ядро» (системная часть), файлы системы, класс, метод, обработчик событий.
Keywords: "core" (system part), system files, class, method, event handler.

При работе над проектами часто приходится решать нетривиальные задачи, когда по причине присутствия в системе большого количество объектов, находящихся в согласованном состоянии, «неосторожная» работа с ними может привести к нарушению целостности данных и корректности работы системы целиком. Во избежание подобного рода ситуаций стало необходимо отслеживать все важные манипуляции, способные спровоцировать нежелательные последствия.

Сам проект состоит из двух частей: «ядро» (системная часть) и прочие компоненты (проектная часть). Некоторые компоненты обеих частей связаны между собой, однако это требует наличие ряда нестандартных обработчиков, гарантирующих согласованность, которые не предусмотрены методами системных объектов. Первым рассматриваемым решением данной проблемы [1-3] стала доработка стандартных методов, путем редактирования фалов системы. Однако такой подход нарушает саму концепцию ядра, которое должно быть гибким и предоставляло бы инструменты для реализации различного нестандартного функционала. К тому же обновления ядра может удалить внесенные изменения в системные файлы, что может существенно нарушить работу системы. В итоге была поставлена задача создания решения, которое должно было быть «гибким», универсальным и оградило бы от нежелательного вмешательства в файлы системы.

Для окончательного решения поставленной задачи лучше всего подходит шаблон проектирования «Наблюдатель», принцип работы которого заключается в механизме асинхронного оповещения экземпляров одного класса об изменении их состояния других объектов, таким образом наблюдая за ними. Именно такой подход может гарантировать условное разделение системных и проектных объектов и в тоже время выступить в роли «моста» между ними, для обеспечения согласованности в работе. Вариантов реализации данного шаблона в интернете не мало, однако в этой статье будет представлен очень простой и эффективный вариант решения данной задачи.

В данной реализации массив обработчиков представлен в виде стека, то есть вызов обработчиков для одного и того же события происходит по очереди их добавления, удаляется последний.

В классе наблюдателя должны быть реализованы три обязательных метода:

- добавление обработчика для события;
- удаление обработчика для события;
- вызов обработчиков для события.

Данные методы описаны в интерфейсе, исходный код которого показан ниже:

```
interfaceiObserver
{static public function attach($event, $handler);
    static public function detach($event);
    static public function notify($args, $event);}
```

Далее был реализован класс [3,4] наблюдателя, имплементирующий данный интерфейс. Единственным свойством данного класса является массив (стек) наблюдателей:

```
staticprivate $observers = [];
```

Метод добавления наблюдателя довольно прост. Он принимает два параметра: название события, для которого устанавливается обработчик, и безымянная функция, вызывающаяся при срабатывании события:

```
static public function attach($event, $handler)
{self::$observers[$event][] = $handler;}
```

Следующий метод отвечает за удаление последнего добавленного обработчика для определенного события, так как возможен сценарий, когда обработчик необходим лишь один раз в определенном месте. В качестве аргументов метод принимает лишь название события, для которого необходимо удалить последний добавленный обработчик.

```
static public function detach($event)
 {foreach (self::$observers as $name => $observers)
if($name == $event)
array_pop(self::$observers[$name]);}
```

Третий и последний метод отвечает за оповещение других объектов о срабатывании события. В качестве параметров принимает массив аргументов для безымянной функции (обработчика), переданной в метод attach и название произошедшего события.

```
static public function notify($args, $event)
 {foreach (self::$observers as $name => $observers)
if($name == $event)
foreach ($observers as $name => $function)
{$function($args); } }
```

В качестве примера был создан классCar и реализованодва метода –drive и stop. Метод drive [4] отвечает за движение машины, однако до начала движения и после иногда необходимо выполнить некоторые действия. Аналогично метод stoротвечает за остановку машины. Реализация данных методов представлена ниже.

```
class Car{
public function __construct(){}
public function drive(){
 Observer::notify(array(&$this), "beforeDrive");
echo "<b>Машина в движении</b><br>";
Observer::notify(array(&$this), "afterDrive"); }
public function stop()
{ Observer::attach(array(&$this), "beforeStop");
    echo "<b>Машина остановилась</b><br>";
    Observer::attach(array(&$this), "afterStop");} }
```

Теперь необходимо создать несколько наблюдателей.

Первое, что необходимо перед началом движения – пристегнуть ремень безопасности и завести двигатель. Перед остановкой необходимо снизить скорость, а в конце заглушить двигатель. Однако глушить двигатель необходимо в самом конце, поэтому данный обработчик будет объявлен перед окончанием поездки. Обработчики событий будут вызываться в порядке их добавления в «стек» [5,6].

Пример создания обработчиков и их работы продемонстрирован ниже.

```
Observer::attach("beforeDrive", function($args){
echo "Пристегнуть ремень безопасности<br>";};
Observer::attach("beforeDrive", function($args){
echo "Завести двигатель<br>";};
Observer::attach("beforeStop", function($args){
echo "Снизить скорость<br>";};
$Car->drive();
//Поскольку нет необходимости пристегивать ремень и заводить двигатель
//после каждой остановки эти наблюдатели временно удаляются
Observer::detach("beforeDrive");
Observer::detach("beforeDrive");
$Car->stop();
echo "<br>";
$Car->drive();
echo "<br>";
Observer::attach("afterStop", function($args){
echo "Заглушить двигатель<br>";};
$Car->stop();
```

Результат работы данного кода можно увидеть на рисунке 1. Для наглядности текст из методов объекта машины выделен жирным шрифтом.

Пристегнуть ремень безопасности
Завести двигатель
Машина в движении

Снизить скорость
Машина остановилась

Машина в движении

Снизить скорость
Машина остановилась
Заглушить двигатель

Рисунок 1. Пример работы наблюдателей

Однако данная реализация шаблона позволяет также вносить изменения в свойства самого объекта. К примеру, при ремонте машины возрастает её цена. Для этого необходимо добавить ещё одно свойство priceи метод renovation:

```
public function renovation($renovationPrice){
    echo "<b>Ремонтмашины</b><br>";
    Observer::notify(array(&$this, $renovationPrice), "afterRenovation");}
```

Создание наблюдателя, который прибавлял бы к стоимости автомобиля цену за ремонт:

```
Observer::attach("afterRenovation", function($args){
    $Car = $args[0];
    $renovationPrice = $args[1];
    $Car->price += $renovationPrice; };
```

И пример обработчика, в котором задается изначальная цена автомобиля, после чего вызывается метод renovation. Результат выполнения данного кода продемонстрирован на рисунке 2.

```
$Car->price = 2000;
echo "Изначальная стоимость машины: " . $Car->price . "<br>";
$Car->renovation(300);
echo "Стоимость машины после ремонта: " . $Car->price . "<br>";
```

Изначальная стоимость машины: 2000
Ремонт машины
Стоимость машины после ремонта: 2300

Рисунок 2. Пример изменения наблюдателем свойств объекта

Данная реализация шаблона проектирования «Наблюдатель» хороша своей простотой и универсальностью. Это решение легко интегрируется в любой проект и не зависит от его структуры, функционала и прочих особенностей. Оно позволяет создавать неограниченное количество ранее непредусмотренных обработчиков, не затрагивая уже готовые компоненты системы.

Список литературы:

1. Котеров Д.В. PHP 7 / Д.В. Котеров, И.В. Смидянов. – Санкт Петербург: БХВ-Петербург 2017. –265 с.
2. Кузнецов М. В. Объектно-ориентированное программирование на PHP. – БХВ-Петербург, 2012.
3. Кэй М. XSLT. Справочник программиста / М. Кэй. – Москва: Символ-Плюс 2002. –1013 с.
4. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство первое издание / Маклафлин Б. – Санкт Петербург 2004.
5. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS. 2-е изд //СПб.: Питер. – 2013. – 2013.
6. Симонова О. Н., Лясин Д. Н. Шаблон проектирования MVC как эффективное средство построения архитектуры программной системы //Современные наукоемкие технологии. – 2014. – №. 5-2. – С. 96-97.