Ссылка на статью:
// Ученые записки УлГУ. Сер. Математика и информационные технологии. УлГУ. Электрон. журн. 2021, № 2, с. 90-96.

Поступила: 01.12.2021
Окончательный вариант: 10.12.2021

УДК 004.42

Взаимодействие паттернов проектирования для эффективной web-разработки

Шмаков С. Э.*[*], *Щенникова Е. В.*, *Определенцева А. Е.

[*]m.shmakov.sergei@yandex.ru
МГУ им. Н.П. Огарева, Саранск, Россия

В статье исследуются паттерны(шаблоны) проектирования на языке JavaScript и их комбинации для повышения эффективности web-разработки. Разработаны примеры реализации шаблонов Стратегия, Event Emitter и Декоратор. Выполнено комбинирование рассмотренных шаблонов в один составной, обладающий свойствами трех паттернов. Проанализированы возможности использования разработанного составного шаблона проектирования.

Ключевые слова: *JavaScript, front-end, web-разработка, шаблоны, паттерны, программирование, Стратегия, Декоратор, Event Emitter.*

Введение

В настоящее время совершенствование web-технологий является активно развивающимся направлением, и в рамках этого направления появляются различные подходы к разработке программного кода и структурирования данных. Одним из таких подходов является использование паттернов (шаблонов) проектирования.

Как известно, под паттерном (шаблоном) проектирования в разработке программного обеспечения понимается повторяемая архитектурная конструкция, которая применяется для решения задачи проектирования в рамках некоторого часто возникающего контекста. Большую популярность паттерны проектирования получили в 1994 г. после выхода книги «Design Patterns: Elements of Reusable Object-Oriented Software» Э. Гамма, Р. Хелмома, Р. Джонсона и Дж. Влиссидеса [2]. Авторы описали простые и изящные решения типичных задач, возникающих в объектно-ориентированном программировании. В книге изложены принципы использования паттернов, а также их первоначальный каталог и классификация.

Важное значение при разработке web-приложений имеет выбор языка программирования. Особенности использования паттернов при программировании на языке JavaScript подробно рассмотрел ведущий специалист компании «Yahoo!» С. Стефанов [4]. Интересны в

указанной книге и примеры антишаблонов, т.е. приемов программирования, которые следует избегать.

Теоретические и практические аспекты паттерного программирования рассмотрены в [3, 4, 5, 8] и в других работах. В настоящее время паттерны часто используются вместе, взаимодействуют и комбинируются в реализациях проектировочных решений [6].

Цель данной работы состоит в том, чтобы проанализировать задачи, с которыми сталкивается web-разработчик, и разработать новые эффективные решения с использованием составных паттернов проектирования. В данной работе с использованием языка программирования JavaScript разработаны примеры реализации шаблонов «Стратегия», «EventEmitter» и «Декоратор». Выполнено комбинирование рассмотренных шаблонов в один составной. Охарактеризованы возможности использования рассмотренного составного шаблона проектирования.

1. Реализация паттерна «Стратегия»

Структурный паттерн «Стратегия» [3] является одним из самых популярных и часто используемых шаблонов проектирования в web-разработке. Суть данного шаблона заключается в том, чтобы заранее указать последовательность вызовов программных функций, которые будут запущены только при выполнении условий, установленных разработчиком.

Приведем пример SEO-оптимизации. Как известно, SEO (Search Engine Optimization) – это комплекс мер по улучшению сайта для его ранжирования в поисковых системах. При оптимизации web-приложения могут отслеживаться нажатия пользователя по ссылкам [7]. Это требуется для определения предпочтений, касающихся интерфейса и продукции. Такие технологии обычно используются при разработке программного обеспечения для интернет-магазинов. Для того чтобы отследить переход пользователя по ссылке, необходимо отправить данные на сервис аналитики. На рис. 1 приведен пример реализации и использования сервиса аналитики.

```
class AnalyticService {
  push(event) {
    return fetch(“/analytic_url”, { method: “POST”, body: event })
  }
}

const handleLinkClick = () => {
  AnalyticService.push(“awasome_event”);
}

<a href=”/awesome_page” onClick={handleLinkClick} />
```

Рис. 1. Пример реализации и использования сервиса аналитики.

В данном коде (рис. 1) метод *AnalyticService.push* является асинхронным, поэтому код прервется после перехода на страницу *«/awesome_page»*. Чтобы заставить этот код работать, необходимо использовать паттерн «Стратегия» и переходить по ссылке только после отправки данных на сервис аналитики (рис. 2).

```
class AnalyticService {
  push(event, callback) {
    return fetch(“/analytic_url”, { method: “POST”, body: event }).then(callback)
  }
}

const redirect = () => window.open(”/awesome_page”)

const handleLinkClick = () => {
  AnalyticService.push(“awasome_event”, redirect);
}

<button onClick={handleLinkClick} />
```

Рис. 2. Пример реализация сервиса аналитики с использованием паттерна «Стратегия».

В данной реализации (рис. 2) метод push имеет второй аргумент *callback*, принимающий функцию обратного вызова перехода по ссылке. Эта функция запустится после того, как отправится запрос на сервис аналитики.

2. Реализация паттерна «Event Emitter»

Еще одним важным паттерном программирования является поведенческий паттерн «Event Emitter». Благодаря этому шаблону можно создать и подписаться на какое-либо событие, которое можно вызвать в любой части кода. Данный шаблон часто используется в web-разработке, например для того, чтобы выйти из учетной записи пользователя на сайте и при этом оповестить все страницы о том, что пользователь разлогинен и не имеет доступа к ресурсу.

В реализации, проиллюстрированной на рис. 3, был создан класс EventEmitter с тремя публичными методами *subscribe, unsubscribe, emit*. Для того чтобы подписаться, требуется создать экземпляр данного класса и вызвать у него метод subscribe, где первым аргументом является название события, а вторым – функция *callback*, которая будет вызвана при триггере этого события.

В реализации шаблона «EventEmitter» также используется паттерн «Стратегия» посредством задействования аргумента *callback* при подписке на какое-либо событие. Таким образом, описана процедура создания одного паттерна на основе другого. Следует отметить, что эта процедура в полной мере иллюстрирует сущность паттернов.

```
class EventEmitter {
  constructor() {
    this.events = {};
  }

  subscribe(eventName, callback) {
    !this.events[eventName] && (this.events[eventName] = []);
    this.events[eventName].push(callback);
  }

  unsubscribe(eventName, callback) {
    this.events[eventName] = this.events[eventName].filter(eventCallback => callback
!== eventCallback);
  }

  emit(eventName, args) {
    const event = this.events[eventName];
    event && event.forEach(callback => callback.call(null, args));
  }
}
```

Рис. 3. Реализация паттерна «Event Emitter».

3. Реализация паттерна «Декоратор»

«Декоратор» является структурным паттерном. Данный шаблон предоставляет возможность добавить новые свойства и функциональность объектам, классам и функциям. Важной особенностью шаблона «Декоратор» является возможность его использования для определения желаемого поведения объектов и программных функций [3].

В примере, представленном на рис. 4, создан паттерн «activeDecorator», который в зависимости от статуса пользователя добавляет новое поле *isActive*.

```
const user1 = { name: “User1”, status: “removed” }
const user2 = { name: “User2”, status: “active” }

const activeDecorator = (user) => {
  return {…user, isActive: user.status === “active”}
}

const users = [user1, user2]

const activeUsers = users.map(activeDecorator).filter(user => user.isActive)
```

Рис. 4. Реализация паттерна «Декоратор».

В данной реализации представлен пример добавления нового свойства массиву пользователей при помощи экземпляра паттерна «Декоратор» (*activeDecorator*) и метода массива *map*. В свою очередь применение метода *activeDecorator* в данной реализации является использованием шаблона «Стратегия».

4. Комбинирование шаблонов

Нередко разработчики сталкиваются с задачами, связанными с учетной записью пользователя и доступом к приложению. Одна из таких задач – выход из учетной записи. Как было указано, для решения такой задачи подходит паттерн «Event Emitter» (раздел 2). Однако с его помощью нельзя вызвать событие, на которое подписка была осуществлена на другой вкладке браузера.

Для того, чтобы избежать повторов и не тратить усилия на создание однотипных паттернов, целесообразно воспользоваться программным пакетом Storage-Emitter из репозитория **npm**. Шаблон типа «Event Emitter» работает с локальным хранилищем браузера и способен вызывать событие во всех вкладках, однако вызвать событие в текущей кладке он не может.

Чтобы создать инструмент, который способен вызвать событие и в текущей, и в соседней вкладке, следует применить паттерн «Декоратор» (раздел 3) для паттерна типа «Event Emitter», который рассмотрен в разделе 2. На рис. 5 представлен пример реализации комбинирования паттернов.

```
import storageEmitter from “storage-emitter”;

const eventEmitter = new EventEmitter();

eventEmitter.emit = (event) => {
  eventEmitter.emit(event);
  storageEmitter.emit(event);
}

eventEmitter.subscribe = (event, callback) => {
  eventEmitter.subscribe(event, callback);
  storageEmitter.subscribe(event, callback);
}
```

Рис. 5. Реализация комбинирования паттернов.

В реализации, проиллюстрированной на рис 5, методы *emit* и *subscribe* были переопределены и дополнены аналогичными методами из Storage-Emitter. Такое переопределе-

ние является шаблоном «Декоратор». Третьим используемым шаблоном в данной комбинации является паттерн «Стратегия» (раздел 1). Применение данного шаблона заключается в передаче функции обратного вызова в метод *subscribe*.

Заключение

В данной работе были созданы реализации отдельных шаблонов проектирования и их комбинаций. Каждый пример реализации был создан при помощи предыдущего описанного шаблона. Таким образом, в последнем разделе приведено описание паттерна, который является объединением всех трех описанных в статье шаблонов: «Стратегия», «Декоратор» и «Event Emitter».

Такие составные паттерны значительно повышают эффективность практической разработки. Во-первых, улучшается «читабельность» кода, т. е. код становится понятным не только автору, но и другому специалисту, который будет работать с кодом, написанным с использованием паттернов. Во-вторых, объекты, классы и функции с использованием паттернов становятся открытыми для расширения и закрытыми для модификации, что является применением принципа Открытости-Закрытости парадигмы объектно-ориентированного программирования [1].

Ни одно крупное web-приложение не обходится без задач, где можно использовать данные конструкции. Представленные паттерны часто используются в web-разработке. Они являются одними из основных шаблонов, которые следуют изучить начинающему разработчику, для поддержания понятного, чистого, расширяемого наследуемого кода. Комбинация паттернов, представленная в третьем разделе, является готовым решением для многих задач, в которых требуется вызвать или отследить какое-либо событие во всех открытых вкладках web-приложения (например, выход из учетной записи).

Приведенный в настоящей статье результат разработки составного паттерна представляет как теоретический, так и прикладной интерес в web-проектировании. С учетом того, что известные программные пакеты с открытым исходным кодом подобным функционалом не обладают, составные паттерны рассмотренного типа могут найти применение при решении различных задач web-проектирования.

Список литературы

1. Вейсфельд М. *Объектно-ориентированное мышление.* СПб.: Питер, 2014. 304 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. *Паттерны объектно-ориентированного проектирования.* СПб: Питер, 2020. 448 с.
3. Крайнова Е. А. Теоретические аспекты паттерного программирования // *Вестник Волжского университета им. В. Н. Татищева*. 2013, № 2 (21), с. 82–90.
4. Майоров А. ES декораторы: разбираемся в деталях //*Системный администратор*. 2016, №5 (162), с. 56–61.
5. Стефанов С. *Javascript. Шаблоны*. СПб.: Изд-во Символ-Плюс, 2011. 272 с.

6. Фримен Э., Робсон Э., Сьерра К., Бейтс Б. *Head First. Паттерны проектирования*. Обновленное юбилейное издание. СПб.: Изд-во Питер, 2018. 656 с.
7. Эрик Э., Спенсер С., Стриккиола Д., Фишкин Р. SEO. *Искусство раскрутки сайтов.* СПб.: БХВ-Петербург, 2014. 688 с.
8. Юнкин И.В., Титов Д.А. Применение структурных шаблонов во встроенных системах //*Наука, образование, бизнес.* Материалы научно-практической конференции ученых, преподавателей, аспирантов, студентов, специалистов промышленности и связи, посвященной Дню радио. Омск: Омский государственный технический университет, 2016. С. 235–239.

Interaction of design patterns for effective web development

Shmakov, S. E.**, *Shchennikova, E. V., Opredelentseva, A. E.

*m.shmakov.sergei@yandex.ru

Ogarev Mordovia State University, Saransk, Russia

The paper examines design patterns (patterns) in the JavaScript language and their combinations to improve the efficiency of web development. Examples of implementation of the Strategy, Decorator, and Event Emitter templates are considered. An example of combining the previously considered templates into one composite one, which has the properties of three patterns, has been developed. The possibilities of using the developed composite template are analyzed.

Keywords: *JavaScript, front-end, web development, templates, patterns, programming, Strategy, Decorator, Event Emitter.*