СОВРЕМЕННЫЕ ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ В РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ REACT

Прохоренко Артём Сергеевич
магистрант, Белорусский государственный университет информатики и радиоэлектроники,
Беларусь, г. Минск
E-mail: acprohor@gmail.com

Якимович Алексей Владимирович
магистрант, Белорусский государственный университет информатики и радиоэлектроники,
Беларусь, г. Минск
E-mail: yaki.alex123@gmail.com

Ладных Елена Александровна
студент, Белорусский государственный университет культуры и искусств,
Беларусь, г. Минск
E-mail: ladnyh.lena@mail.ru

Возможно, во время решения той или иной задачи при создании приложения или программы вы могли понять, что уже делали это ранее. Прежде всего, опытному разработчику понятно, что не нужно решать каждую новую задачу с нуля, можно использовать найденные решения повторно. Дизайн должен, с одной стороны, соответствовать решаемой задаче, с другой - быть общим, чтобы удалось учесть все требования, которые могут возникнуть в будущем. Должна быть достаточная гибкость. Чтобы добавление новой функциональности в проект не было серьёзной проблемой.

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Вместо того чтобы решать каждую новую задачу с нуля можно повторно воспользоваться теми решениями, которые оказались удачными в прошлом. Отыскав хорошее решение один раз, вы можете прибегать к нему снова и снова. Хотелось бы также избежать вовсе или, по крайней мере, свести к минимуму необходимость перепроектирования.

Паттерны проектирования упрощают повторное использование удачных проектных и архитектурных решений.

По словам Кристофера Александра, «любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип ее решения, причем таким образом, что это решение можно потом использовать миллион раз, ничего не изобретая заново». Хотя автор имел в виду паттерны, возникающие при проектировании зданий и городов, но его слова верны и в отношении паттернов проектирования.

Паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна.

В общем случае паттерн состоит из четырех основных элементов:

1. Имя. Оно должно чётко описывать проблему проектирования и её решение. По имени паттерна должно сразу быть понятно в каком случае необходимо использовать шаблон проектирования.

2. Задача. Более подробное описание проблемы, которое решает паттерн и когда его следует применять.

3. Решение. Описание элементов дизайна, отношений между ними, функций каждого элемента. Имеется в виду абстрактный дизайн так как шаблоны применяются в самых разных ситуациях. Даётся абстрактное описание задачи проектирования и того, как она может быть решена с помощью некоего весьма обобщенного сочетания элементов.

4. Результаты. Этот пункт часто опускается, однако он помогает точнее понять результат, который вы получите при внедрении паттерна и оценить влияние на степень гибкости, расширяемости и переносимости системы.

Помимо того, что вы получаете возможность использовать решение какой-то проблемы повторно, также у вас появляется возможность добавлять, изменять и удалять функциональность. В некоторых случаях без использования паттернов проектирования это может вызвать серьёзные трудности. Если, например, связи между компонентами системы сложно переплетены, то добавить новый компонент будет непросто. Некоторые паттерны проектирования позволяют создать такую систему, где расширение функционала не будет проблемой. Также в системе часто необходимо создавать ограничения взаимодействий компонентов друг с другом или, другими словами, инкапсулировать данные. Благодаря правильному использованию паттернов можно создать систему с правильным взаимодействием компонентов внутри.

Классификация паттернов.

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы.

Самые низкоуровневые и простые паттерны — *идиомы*. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные — *архитектурные паттерны*, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

Кроме того, паттерны отличаются и предназначением. Выделяют три основные группы паттернов:

Порождающие паттерны – позволяют гибко создавать объекты, не внося в систему лишние зависимости. Для порождающих паттернов актуальны две темы. Во-первых, эти паттерны инкапсулируют знания о конкретных классах, которые применяются в системе. Во-вторых, скрывают детали того, как эти классы создаются и стыкуются. Следовательно, порождающие паттерны обеспечивают большую гибкость при решении вопроса о том, что создается, кто это создает, как и когда.

Структурные паттерны – позволяют создавать необходимые связи между объектами системы. В них рассматривается вопрос о том, как из этих объектов формируется более крупная структура. Структурные паттерны на уровне класса используют наследование для создания системы из интерфейсов и реализаций. Один из примеров в объектно-ориентированном программировании это множественное наследование, в результате которого получается класс, который обладает всеми свойствами своих родителей. этот паттерн полезен, когда нужно организовать совместную работу нескольких независимо разработанных библиотек.

Также структурные паттерны позволяют компоновать объекты для получения новой функциональности. Таким образом достигается некоторая гибкость, выраженная возможностью изменить композицию объектов во время выполнения, что невозможно для статической системы.

Поведенческие паттерны – эти паттерны связаны с алгоритмами и распределением обязанностей между компонентами системы. Эти паттерны решают задачи эффективного и безопасного взаимодействия между объектами программы.

Паттерны проектирования в веб-разработке.

В программировании паттерны проектирования появились относительно недавно. За создание общепринятых шаблонов и описания их работы серьёзно взялись только в начале

2000-х. Разумеется и до этого разработчики использовали свои собственные наработки с предыдущих проектов, однако общепринятых шаблонов не существовало.

Если говорить о технологии React примерно четыре года назад ещё не было устоявшихся практик, изучая которые и следуя которым можно было бы улучшить качество своих проектов.

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. Часто используется для создания веб-приложений. Также существует мобильный вариант библиотеки – *React-Native*. это платформа кроссплатформенных мобильных приложений также с открытым исходным кодом.

Шаблоны проектирования, которые возникли и развились в экосистеме *React* за время её существования, улучшают читабельность и чистоту кода, облегчают повторное использование компонентов. Сообществу *React* понадобилось некоторые время для того, чтобы выработать несколько идей, которые теперь стали популярными. Теперь, учитывая то, что число *React*-разработчиков постоянно растёт, то, что в развитие этого проекта вкладываются серьёзные силы, можно наблюдать эволюцию нескольких интересных шаблонов проектирования.

Передача свойств вниз по дереву компонентов (*passing down props*).

В результате роста и развития приложения на платформе *React* получается, что оно состоит из компонентов, которые являются контейнерами для других компонентов. В результате возникает необходимость передавать большие объемы свойств, предназначенных для потомков этих компонентов. Данный паттерн предоставляет возможность передачи данных от компонента родителя компонентам потомкам.

Деструктурирование свойств (*destructuring props*).

Со временем приложение меняется, то же самое происходит и с компонентами. Компонент, написанный долгое время назад, может использовать состояние, но в текущих условиях он может быть преобразован в компонент без состояния. Часто можно видеть и обратную ситуацию.

Если задействовать возможности деструктурирования свойств, можно использовать полезный приём, который часто применяется для того, чтобы, в долгосрочной перспективе, облегчить себе работу над проектом. Свойства можно деструктурировать для компонентов обоих типов.

Шаблон «провайдер»

Этот шаблон использует возможности *React*, которые появились сравнительно недавно. Как было сказано выше в *React* существует возможность передавать свойства потомкам компонента. Этот процесс усложняется при росте числа компонентов, которым необходимо передать некоторые свойства. В ситуации, когда существует необходимость передать свойства большому количеству компонентов полезно использовать *API React Context*. Эта возможность пригодится не в любой ситуации, но в некоторых ситуациях она значительно упрощает работу. Шаблон провайдер применён как в *API React Context*, так и в знакомом многим *React*-разработчикам *React Redux* и *Apollo*. В данном шаблоне компонент верхнего уровня называется провайдером. Он записывает в контекст какие-то свойства, которые компоненты потребители могут извлекать. Данные в таком случае получаются как бы глобально доступными.

Компоненты высшего порядка (*High Order Component,HOC*).

Компоненты высшего порядка предназначены для удовлетворения потребностей в повторном использовании функционала множеством компонентов. Компонент высшего порядка — это функция, которая принимает входной компонент и возвращает расширенную или изменённую версию этого компонента. На примере *Redux* компонент высшего порядка в функции *connect*, которая, принимая компонент, добавляет к нему некие свойства.

Однако у компонентов высшего порядка есть и недостатки. Каждый такой компонент приводит к созданию дополнительного компонента *React* в *DOM/vDOM*. Это, по мере роста приложения, может вести к потенциальным проблемам с производительностью.

Шаблон «render props» или «функция как потомок»

позволяет достичь того же самого, что достижимо с помощью компонентов высшего порядка. Эти шаблоны взаимозаменяемы. Сравнивая их, сложно отдать абсолютное предпочтение одному из них. Оба шаблона используются для того, чтобы сделать код чище и улучшить возможности его повторного использования. идея использования шаблона *render props* заключается в передаче управления вашей функцией рендеринга другому компоненту, который затем возвращает управление через свойство, являющееся функцией. Некоторые предпочитают применять для достижения того же эффекта динамические свойства, некоторые просто используют *this.props.children*.

шаблон может быть использован в ситуациях, где нужна некая подходящая для повторного использования логика внутри компонента, при этом данный компонент не планируется оборачивать в компонент высшего порядка.

Заключение.

Польза паттернов проектирования достаточно значительна. Шаблоны предлагают разработчику надёжные способы решения типичных задач, использование которых экономит время и улучшает код. Позволяют добиться гибкости и расширяемости проекта.

Общепринятых шаблонов применяемых в *React* пока не так много, однако область охвата технологии увеличивается, сама технология также совершенствуется в результате чего будут появляться новые шаблоны проектирования для решения типовых задач.

Список литературы:

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования./ Э. Гамма, Р. Хелм, Р.Джонсон, Дж. Влиссидес . — СПб: Питер, 2001. — 368 с.: ил. (Серия «Библиотека программиста»)
2. Фримен К., Фримен Э., Сьерра К., Бейтс Б. Паттерны проектирования/ Эрик Фримен, Элизабет Фримен, Кэтти Сьерра, Берт Бейтс. — СПб: Питер, 2019. —656с.
3. Патерны проектирования [Электронный ресурс]. —Режим доступа: www.url:https://refactoring.guru/ru/design-patterns. — 11.12.2019.
4. Moldovan A. Evolving Patterns in React [Электронный ресурс]/ Alex Moldovan. — Режим доступа: www.url: https://www.freecodecamp.org/news/evolving-patterns-in-react-116140e5fe8f/.—11.12.2019.
5. Поведенческие паттерны проектирования[Электронный ресурс]. —Режим доступа: www.url: https://refactoring.guru/ru/design-patterns/behavioral-patterns. — 11.12.2019.