MV*

LoftSchool
от мыслителя к создателю

Содержание

1

Что такое паттерны

Паттерн - это проверенный временем способ решать архитектурные задачи.

Грубо говоря: есть какая-то задача, которая встает перед разработчиками чаще остальных. В таких случаях, разработчики собираются и придумывают универсальный вариант ее решения. Вот решение и называется паттерном. Почему именно паттерном? Паттерн, с английского - шаблон. То есть разработчики придумывают шаблон решения конкретной задачи.

Почему шаблон? Потому что паттерны - это абстрактные решения.
Вы не встретите паттерны типа “паттерн создания социальной сети” или “паттерн создания интернет-магазина”. Да, паттерны придумывают чтобы решать конкретные, но абстрактные задачи.

Например: есть сайт с множеством страниц. Необходимо сделать так, чтобы можно было в любой момент изменить оформление страницы. При этом не важно какой будет контент страницы. При этом не важно сколько будет страниц. При этом не важно как часто и на каких мы захотим менять это оформление.

Заметьте, задача уже становится абстрактной.

Если решать эту задачу как НЕ абстрактную, то вы, скорее всего, сделаете много страниц с разным оформлением и по мере надобности будете редактировать содержимое страниц, чтобы изменить оформление. Или второй вариант: сделаете большой if или switch в точке входа в ваше приложение и, по мере надобности, будете изменять условия или кейсы внутри него. Про такой подход обычно говорят: привязка к реализации.
Это когда разработчик разрабатывает что-то и не думает о том, что в любой момент, требования к этой системе могут измениться или потребуется масштабировать (расширить) саму системы. Такой подход обычно не приводит ни к чему хорошему. Систему, построенную таким образом, почти невозможно расширять и качественно поддерживать.

Чаще всего, для построения грамотной и расширяемой архитектуры, необходимо: абстрагироваться от конкретных условий задачи; создать абстрактную систему, которую можно расширять. И только потом уже строить на этой абстрактной системе конкретную реализацию.

И именно при построении таких абстрактных систем и пригождаются паттерны - проверенные временем способы решать конкретные, но абстрактные задачи.

Задача, которая была описана выше носит структурный характер.
Нам нужно построить структуру системы таким образом, чтобы было не важно: какой контент сейчас находится на странице в данный момент, сколько таких страниц, сколько шаблонов и т.д. Более того, необходимо иметь возможность просто и быстро расширять количество шаблонов.
Вот это и есть абстрактная задача, никак не связанная с конкретной реализацией.

А вот та же задача, но еще более абстрактно: нам нужно построить систему, которая может иметь любое количество отображений чего бы то ни было. Абстракция автоматически подразумевает простое и быстрое расширение системы, в случае необходимости.

Если подвести итог, то нам нужно решить задачу по разделению самих данных и их визуального представления.

Среди множества паттернов проектирования, есть подгруппа структурных паттернов, которая носит название MV-паттерны.

Так как паттерны - вещь абстрактная, то всё, что вы здесь увидите, носит абстрактный характер. Примеры и разбор кода смотрите в вебинаре.

2

MV-паттерны

MV(model-view)-паттерны - это набор паттернов, призванных решать задачи разделения самих данных и отображения этих данных.

Такие паттерны можно встретить и в таком написании MV*

Что означает, что M и V - это обязательные компоненты, в вместо звездочки может стоять еще какой-то компонент. Сейчас разберемся что это значит.

Мы уже знаем, что MV- паттерны служат для разделения данных и отображения этих данных. Так вот:

M(model, модель) - это сами данные
V(view, вид) - это представление этих данных.
А вот каким образом M и V будут взаимодействовать друг с другом - определяет третий компонент.

Мы рассмотрим 3 таких паттерна:

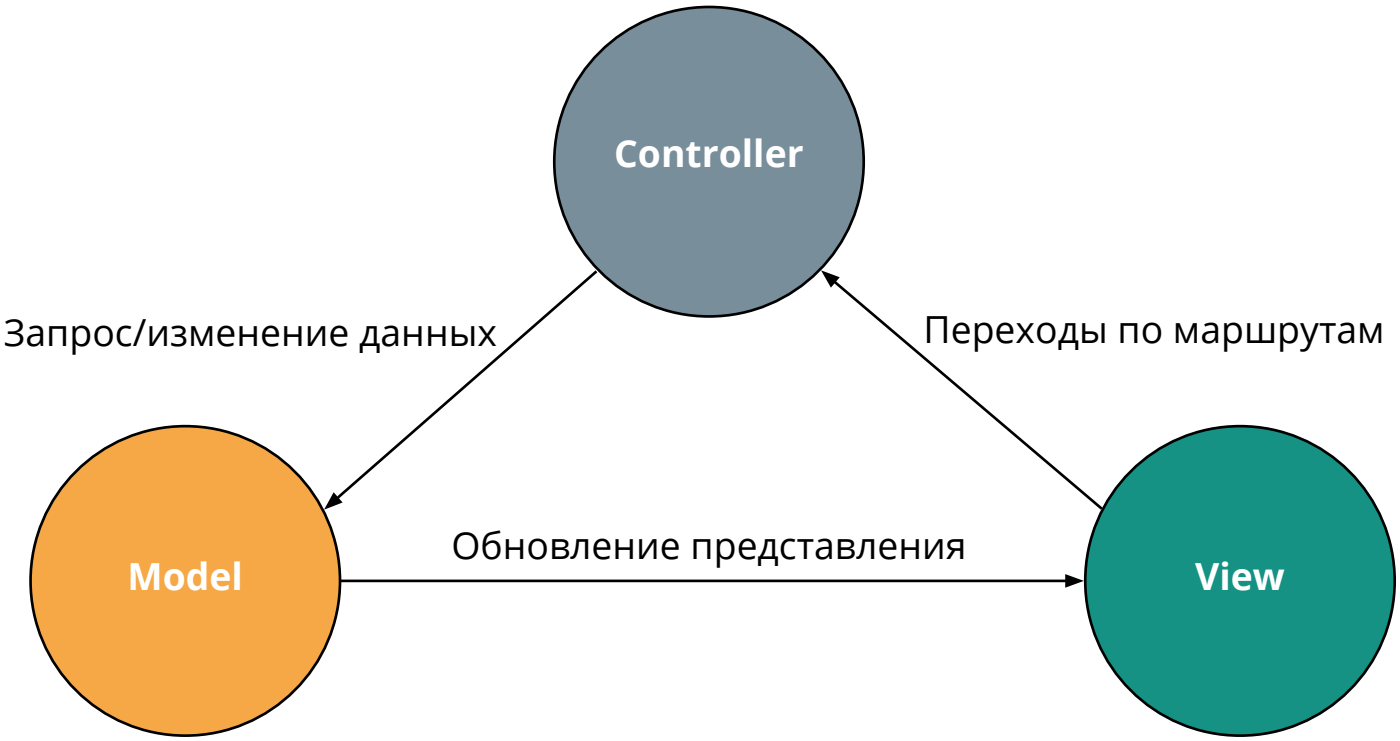
MVC - model-view-controller
MVP - model-view-presenter
MVVM - model-view-viewModel

MVC

Model-View-Controller

Как видно из названия, связующим звеном между моделью и видом, является контроллер.

Контроллер - это нечто, через что проходят все данные между видом и моделью. То есть вид и модель напрямую не взаимодействуют между собой. Только через контроллер.



Это упрощенное изображение взаимодействия всех трех частей в модели

Чаще всего, контроллер представлен в виде маршрутизатора, через который проходят все запросы пользователя отобразить ту или иную информацию. Этот пример будет рассмотрен на вебинаре.

Пока обратите внимание на то, что в **MVC**, контроллер не может получать никаких данных от вида, то есть связь типа контроллер-вид - односторонняя.

А вот более детальное изображения **MVC**. Это переложение **MVC** на то, как пользователь работает со своим браузером.

Т.к. паттерны - вещь абстрактная, попытайтесь переложить идеологию **MVC** на какой-нибудь другой пример (придумайте).

MVP

Model-View-Presenter

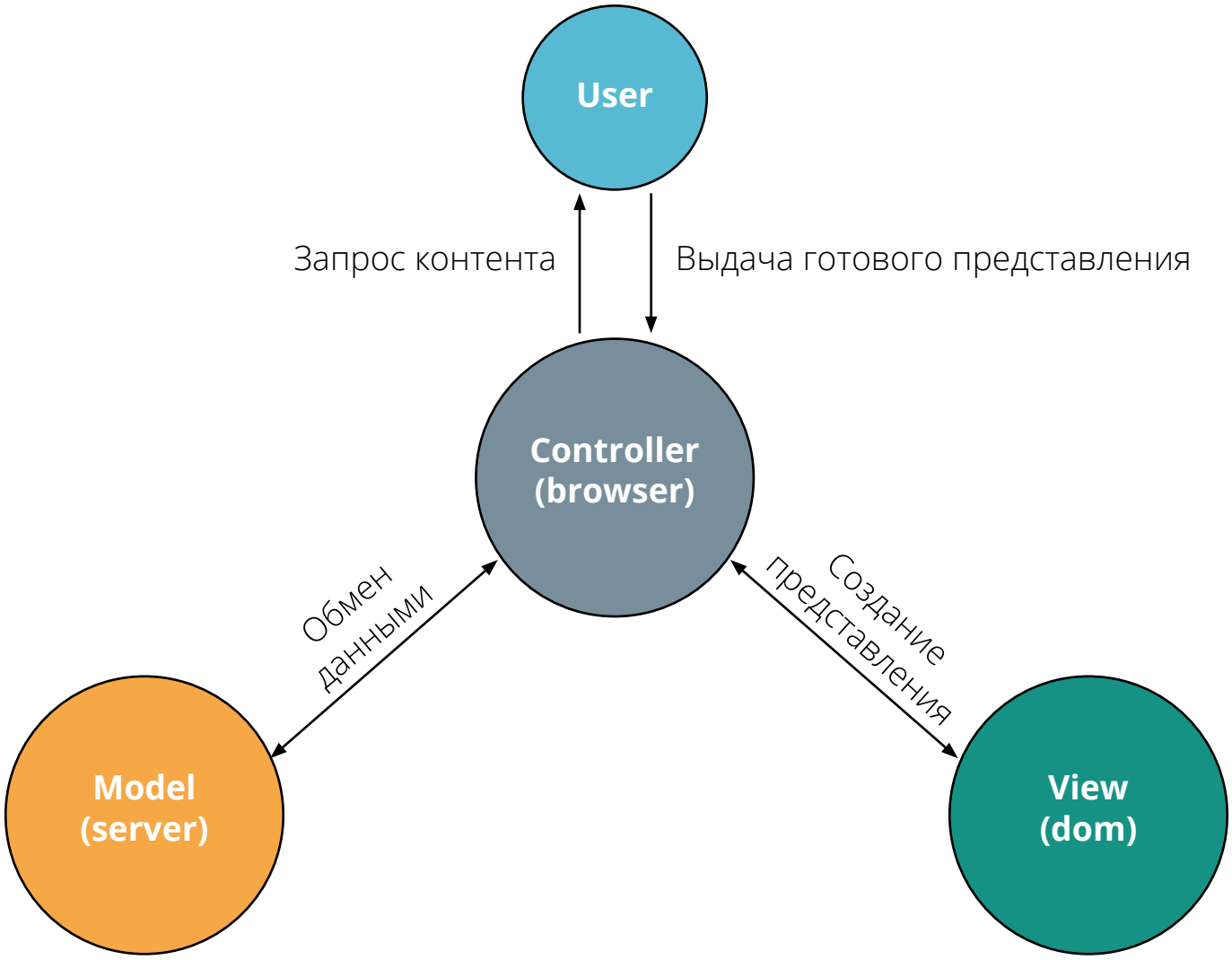
Опять же, как видно из названия, связующим звеном между моделью и видом является некий **Presenter**.

Presenter - похож на контроллер в том плане, что именно через него происходит взаимодействие между моделью и представлением.

Разница в том, что в **MVP**, между видом и **presenter** - двусторонняя связь. Как было сказано выше - в **MVC** между видом и контроллером - односторонняя связь.

Что значит двусторонняя связь? Это значит, что **presenter** подписывается на события от **view** и при поступлении этих событий, может менять содержимое **view**. Примером тому может служить DOM модель.

А вот схематическое определение MVP:


User
Запрос контента
Выдача готового представления
Controller
(browser)
Обмен
данными
Создание
представления
Model
(server)
View
(dom)

MVVM

Model-View-ViewModel

Как видно из названия, связующим звеном здесь является **ViewModel**.

ViewModel - это усовершенствованный presenter. Он имеет ту же схему работы, что и **MVP** - позволяет осуществить двусторонний обмен между с **view**. Но, плюс ко всему, позволяет реализовать **data-binding**.

Это не значит, что при помощи чистого **MVP** нельзя реализовать **data-binding.**

В этом вопросе очень легко запутаться. Мы предлагаем рассуждать так:

MVP - это двусторонний обмен между presenter и view, а MVVM - это тот же MVP, но с data-binding.

3

Data binding

Представьте: на странице есть текстовое поле, куда вам предложено ввести ваше имя.

По мере того, как вы вводите имя, где-то еще на странице выводится надпись: “Привет ____” и вместо прочерка - ваше имя. В то же время, имя в этом блоке обновляется динамически - по мере того, как вы вводите свое имя.

Это и есть ***data binding*** - то есть механизм связывания данных на странице. Блока вывода фразы “Привет ____” связан с данными, которые вы вводите в текстовое поле. По мере того, как вы вводите данные в поле, все зависимые от этих данных блоки обновляют свое содержимое.

Пример такого взаимодействия будет показан в вебинаре.