Паттерны проектирования: паттерн Цепочка Обязанностей в TypeScript

Добро пожаловать в серию «Паттерны проектирования в TypeScript», в которой будут представлен ряд полезных паттернов проектирования в веб-разработке с использованием TypeScript.

Вот ссылки на опубликованные статьи:

Паттерн Стратегия в TypeScript

Паттерн Цепочка Обязанностей в TypeScript

Паттерн Наблюдатель в TypeScript

Паттерн Паттернного метода в TypeScript

Паттерн Адаптер в TypeScript

Паттерн Фабричного метода в TypeScript

Паттерн Абстрактной фабрики в TypeScript

Паттерны проектирования очень важны для веб-разработчиков, которые освоив паттерны становятся способными улучшить качество написания кода. В этой статье я буду использовать TypeScript, чтобы рассказать о паттерне Цепочка Обязанностей.

Паттерн Цепочка Обязанностей — это способ избежать связи и взаимного влияния между отправителем и получателем запроса, предоставляя нескольким объектам возможность обрабатывать запрос. В паттерне Цепочка Обязанностей многочисленные объекты ссылаются последовательно друг на друга посредством ссылки от одного объекта к последующему объекту, чтобы сформировать цепочку. Запрос передаются по цепочке до тех пор, пока один из объектов в цепочке не осуществит обработку этого запроса.

Возьмем, к примеру, согласование отпуска в нашей компании: когда я прошу выходной, его нужно только утвердить тимлидом, и нет необходимости передавать его вышестоящему руководителю и директору. Разные должности в компании имеют разные обязанности и полномочия. Если звено в Цепочке Обязанностей не может обработать текущий запрос и в цепочке имеется следующее звено – то запрос будет перенаправлен на это последующее звено для обработки.

В процессе разработки программного обеспечения для Цепочки Обязанностей распространенным сценарием применения является мидлваре. Давайте посмотрим, как Цепочку Обязанностей используют для обработки запросов.

Чтобы лучше понять последующий код, необходимо сначала необходимо сначала внимательно рассмотреть диаграмму UML, описывающую этот код:

На приведенном выше рисунке мы определяем интерфейс Handler. Этот интерфейс в свою очередь определяет следующие два метода:

use(h: Handler): Handler => Используется для регистрации обработчика (мидлваре)

get(url: string, callback: (data: any) => void): void => Регистрирует обработчик запроса на получение

Интерфейс обработчика

Затем мы определим абстрактный класс AbstractHandler, который инкапсулирует логику обработки Цепочки Обязанностей. Другими словами, этот абстрактный класс соединяет обработчики, чтобы сформировать цепочку последовательных ссылок.

Абстрактный класс AbstractHandler

На основе абстрактного класса AbstractHandler мы определяем классы AuthMiddleware и LoggerMiddleware. Мидлваре AuthMiddleware используется для обработки аутентификации пользователей, а мидлваре LoggerMidddleware используется для вывода журналов запросов.

Класс AuthMiddleware

Класс LoggerMiddleware

Вместе с классами AuthMiddleware и LoggerMiddleware давайте определим класс Route для регистрации созданных мидлваре.

Класс Route

После определения класса Route мы можем использовать его для регистрации имеющихся мидлваре следующим образом:

Когда вы запустите приведенный выше код, вы получите соответствующий ему вывод, показанный ниже на рисунке:

Итак, подытожим сценарии использования паттерна Цепочка Обязанностей:

Когда системе необходимо отправить запрос одному из нескольких объектов без явного указания получателя.

Когда есть несколько объектов, которые могут обрабатывать запрос, и какой именно объект обрабатывает запрос, автоматически будет определятся только уже во время выполнения - клиенту нужно только отправить запрос в цепочку.

Если у вас есть какие-либо вопросы, пожалуйста, пишите мне. В дальнейшем я продолжу знакомить вас с другими паттернами, и если вам интересно, подпишитесь на меня в Medium или Twitter.