Паттерны проектирования: паттерн адаптера в TypeScript

Простое решение проблем несовместимости интерфейсов с помощью Паттерна Адаптер

Паттерны проектирования очень важны для веб-разработчиков, и мы можем писать лучший код, освоив их. В этой статье я буду использовать TypeScript, чтобы представить Паттерн Адаптер.

Для Интернета электронная почта является очень часто используемым сервисом. На платформе Node.js мы можем использовать модуль nodemailer и с помощью него легко реализовать функцию отправки электронных писем. После установки модуля nodemailer для отправки электронных писем достаточно выполнить следующий код:

Чтобы избежать привязки сервиса электронной почты к конкретному провайдеру услуг, перед разработкой почтового сервиса мы сначала определим интерфейс, связанный с почтовым провайдером:

С помощью этих интерфейсов мы можем легко создавать новые почтовые сервисы:

В обычной ситуации это решение не представляет большой проблемы в реализации. Но однажды вам может понадобится использовать стороннего поставщика облачных услуг электронной почты, например, sendgrid или mailersend и т.д. В их SDK вы найдете название метода, используемого для отправки почты - send. Пойдем дальше и определим интерфейс CloudEmailProvider:

Сравнивая его и ранее созданный интерфейс EmailProvider, вы обнаружите следующую проблему:

Очевидно, что использовать интерфейс EmailService для доступа к сторонним облачным службам электронной почты невозможно из-за несовпадения наименования методов. Для решения этой проблемы существует множество способов, но мы попробуем использовать Паттерн Адаптер для этой задачи.

Цель Паттерна Адаптер — это позволить двум объектам, которые не работают вместе из-за несоответствия интерфейсов, начать работать вместе. Это как клей, изменяющий свойства двух разные предметов, чтобы они могли работать вместе.

Паттерн Адаптер подразумевает следующие роли:

• Client(EmailService): это объект, который должен использовать целевой интерфейс;

• Target(EmailProvider): определяет интерфейс, ожидаемый Client;

• Адаптер(CloudEmailAdapter): адаптирует интерфейс Adaptee к интерфейсу Target;

• Adaptee(CloudEmailProvider): описывает интерфейс, который необходимо адаптировать.

После погружения в термины Паттерна Адаптер, давайте создадим класс CloudEmailAdapter:

В приведенном выше коде интерфейс EmailProvider и интерфейс CloudEmailProvider не совпадают по наименованию метода, мы вводим новый класс CloudEmailAdapter для решения данной проблемы совместимости.

Теперь давайте импортируем и воспользуемся модулем sendgrid в качестве примера для реализации SendgridEmailProvider:

Предупреждение: приведенный выше код предназначен только для демонстрационных целей и требует соответствующей корректировки при использовании в реальных проектах.

Теперь, когда определены классы SendgridEmailProvider и CloudEmailAdapter, давайте посмотрим, как их использовать вместе:

Итак, подытожим сценарии использования Паттерна Адаптер:

• Когда системе необходимо использовать существующий класс, а интерфейс этого класса не соответствует потребностям системы, то есть интерфейс несовместим с ней;

• Когда используется сервис, предоставляемый третьей стороной, но интерфейс этого сервиса отличается от интерфейсов, используемых в вашем коде.