Альтернативные виды связывания модулей ПО

JSP & Servlets
14 уровень, 9 лекция

ОТКРЫТА

Замена прямых зависимостей на обмен сообщениями

Иногда модулю нужно всего лишь известить других о том, что в нем произошли какие-то события/изменения и ему не важно, что с этой информацией будет происходить потом.

В этом случае модулям вовсе нет необходимости “знать друг о друге”, то есть содержать прямые ссылки и взаимодействовать непосредственно, а достаточно всего лишь обмениваться сообщениями (messages) или событиями (events).

Иногда кажется, что связь модулей через обмен сообщениями является гораздо более слабой, чем прямая зависимость. Действительно, ведь методы не вызываются, информации о классах нет. Но это не более чем иллюзия.

Вместо имен методов логика начинает привязываться к типам сообщений, их параметрам и передаваемым данным. Связность таких модулей размазывается.

Раньше было как: вызываем методы — есть связность, не вызываем методы — нет связности. А теперь представь, что модуль А стал посылать немного другие данные в своих сообщениях. И при этом все зависимые от этих сообщений модули будут работать неправильно.

Допустим, раньше, при добавлении нового пользователя модуль авторизации слал сообщение **USER_ADDED, а после апдейта он стал слать это сообщение при попытке регистрации и дополнительно в параметрах указывать успешная регистрация или нет.**

Поэтому очень важно реализовывать механизм сообщений очень грамотно. Для этого есть различные шаблоны.

Наблюдатель (Observer). Применяется в случае зависимости «один-ко-многим», когда множество модулей зависят от состояния одного — основного. Использует механизм рассылки, который заключается в том, что основной модуль просто осуществляет рассылку одинаковых сообщений всем своим подписчикам, а модули, заинтересованные в этой информации, реализуют интерфейс “подписчика” и подписываются на рассылку.

Этот подход находит широкое применение в системах с пользовательским интерфейсом, позволяя ядру приложения (модели) оставаться независимым и при этом информировать связанные с ним интерфейсы о том, что произошли какие-то изменения и нужно обновиться.

Тут формат сообщений стандартизируется на уровне операционной системы, разработчики которой должны позаботиться об обратной совместимости и хорошей документации.

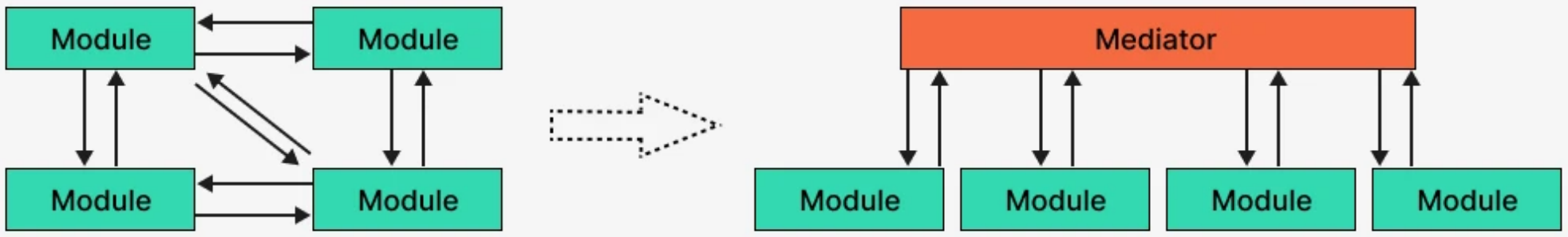
Организация взаимодействия посредством рассылки сообщений имеет дополнительный “бонус” — необязательность существования “подписчиков” на “опубликованные” (то есть рассылаемые) сообщения. Качественно спроектированная подобная система допускает добавление/удаление модулей в любое время.

Шина обмена сообщениями

Можно организовать обмен сообщениями и по-другому использовать для этого паттерн **Посредник (Mediator)**.

Он применяется, когда между модулями имеется зависимость “многие ко многим”. Медиатор выступает в качестве посредника в общении между модулями, действуя как центр связи и избавляет модули от необходимости явно ссылаться друг на друга.

В результате взаимодействие модулей друг с другом (“все со всеми”) заменяется взаимодействием модулей лишь с посредником (“один со всеми”). Говорят, что посредник инкапсулирует взаимодействие между множеством модулей.



Это так называемый **умный посредник**. Именно там чаще всего разработчики начинают добавлять свои костыли, чем влияют на поведение отдельных модулей через включение/выключение получения определенных сообщений.

Типичный пример из жизни — контроль трафика в аэропорту. Все сообщения, исходящие от самолетов, поступают в башню управления диспетчеру вместо того, чтобы пересылаться между самолетами напрямую. А диспетчер уже принимает решения о том, какие самолеты могут взлетать или садиться, и, в свою очередь, отправляет самолетам соответствующие сообщения.

Важно! Модули могут пересылать друг другу не только простые сообщения, но и объекты-команды. Такое взаимодействие описывается шаблоном Команда (**Command**). Суть заключается в инкапсулировании запроса на выполнение определенного действия в виде отдельного объекта.

Фактически этот объект содержит один единственный метод `execute()`, что позволяет затем передавать это действие другим модулям на выполнение в качестве параметра и вообще производить с объектом-командой любые операции, какие могут быть произведены над обычными объектами.

Закон Деметры (law of Demeter)

Закон Деметры запрещает использование неявных зависимостей: "Объект A не должен иметь возможность получить непосредственный доступ к объекту C, если у объекта A есть доступ к объекту B и у объекта B есть доступ к объекту C".

Это означает, что все зависимости в коде должны быть “явными” — классы/модули могут использовать в работе только “свои зависимости” и не должны лезть через них к другим. Хороший пример – это трехуровневая архитектура. Уровень интерфейса должен работать с уровнем логики, но не должен напрямую взаимодействовать с уровнем базы данных.

Кратко этот принцип формулируют еще таким образом: "Взаимодействуй только с непосредственными друзьями, а не с друзьями друзей". Тем самым достигается меньшая связанность кода, а также большая наглядность и прозрачность его дизайна.

Закон Деметры реализует уже упоминавшийся “принцип минимального знания”, являющийся основой слабой связанности и заключающийся в том, что объект/модуль должен знать как можно меньше деталей о структуре и свойствах других объектов/модулей и вообще чего угодно, **включая собственные компоненты**.

Аналогия из жизни: если ты хочешь, чтобы собака побежала, глупо командовать ее лапами, лучше отдать команду собаке, а она уже разберется со своими лапами сама.

Композиция вместо наследования

Это очень большая и интересная тема и она достойна как минимум отдельной лекции. На эту тему в интернете было сломано немало копий пока не был достигнут консенсус — наследование используем по минимуму, композицию — по максимуму.

Все дело в том, что наследование дает фактически самую сильную связь между классами, поэтому его следует избегать. Эта тема хорошо раскрыта в статье Герба Саттера — “Предпочитайте композицию наследованию”.

Когда ты начнешь изучать паттерны проектирования, то столкнешься с целой кучей паттернов, которые управляют созданием объекта или его внутренним устройством. Кстати, могу посоветовать в данном контексте обратить внимание на шаблон **Делегат (Delegation/Delegate)** и, пришедший из игр, шаблон **Компонент (Component)**.

Детальнее о паттернах мы поговорим немного позднее.

‹ Предыдущая лекция

Следующая лекция ›

+18

Комментарии (2) популярные новые старые

JavaCoder

Введите текст комментария

Руслан Шмаков Уровень 82 EXPERT 21 октября 2022, 12:08

Материал лекции показался сложным и непонятным, но на все вопросы я ответил правильно)

Ответить +1

Andrey Panchenko Моет полы в **Яндекс** 20 сентября 2022, 11:20

Насчёт композиции, советую.

Ответить +4

ОБУЧЕНИЕ

Курсы программирования

Курс Java

Помощь по задачам

Подписки

Задачи-игры

СООБЩЕСТВО

Пользователи

Статьи

Форум

Чат

Истории успеха

Активности

КОМПАНИЯ

О нас

Контакты

Отзывы

FAQ

Поддержка

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store