Антипаттерны

JSP & Servlets
17 уровень, 3 лекция

ОТКРЫТА

Знакомство с анти-паттернами

Анти-паттерны — полная противоположность паттернам. Напомним, паттерны проектирования — это примеры практик хорошего программирования, то есть шаблоны решения определённых задач. А вот анти-паттерны — их полная противоположность, то есть шаблоны ошибок, которые совершаются при решении различных задач.

Частью практик хорошего программирования является именно избежание анти-паттернов. Не надо думать, что это такая непонятная теоретическая фигня — это конкретные проблемы, с которыми сталкивался практически каждый разработчик. Кто осведомлен, тот и вооружён!

Рассмотрим несколько анти-паттернов, распространённых среди новичков:

- Магические числа и строки
- Класс бога
- Преждевременная оптимизация
- Изобретение велосипеда
- Изобретение одноколесного велосипеда

Магические числа и строки

Магическое число — константа, использованная в коде для чего-либо (чаще всего — идентификации данных), само число которой не несёт никакого смысла без соответствующего комментария. Числа не несут абсолютно никакой семантики.

Когда в коде вашего проекта начинаются появляться числа, значение которых не является очевидным, — это очень плохо. Программист, который не является автором такого кода, с трудностями сможет объяснить, как это работает. Со временем и автор кода с магическими числами не сможет объяснить его.

Числа затрудняют понимание кода и его рефакторинг. Главные причины этой ошибки — спешка при разработке и отсутствие практики программирования. Этот анти-паттерн надо пресекать на корню, оговаривая использование числовых констант перед началом разработки.

Для решения такой проблемы нужно создать переменную, имя которой объясняет назначение числовой константы, и присвоить ей требуемое значение.

Класс бога

Божественный объект — анти-паттерн, который довольно часто встречается у разработчиков ООП. Такой объект берет на себя слишком много функций и/или хранит в себе практически все данные. В итоге мы имеем непереносимый код, в котором, к тому же, сложно разобраться.

К тому же подобный код довольно сложно поддерживать, учитывая, что вся система зависит практически только от него. Причины этой ошибки: некомпетентность разработчика, взятие одним разработчиком большой части работы (особенно, когда объем работы превышает уровень опыта этого разработчика).

Бороться с таким подходом надо, разбивая задачи на подзадачи, которыми смогут заниматься разные разработчики.

Преждевременная оптимизация

Преждевременная оптимизация — это оптимизация, которую выполняют до того, как у программиста есть вся информация, необходимая для принятия взвешенных решений по поводу того, где и как нужно её проводить.

На практике сложно предсказать, где встретится узкое место. Попытки навести оптимизацию до получения эмпирических результатов приведут к усложнению кода и появлению ошибок, а пользы не принесут.

Как избежать? Сначала пиши чистый, читаемый, работающий код, используя известные и проверенные алгоритмы и инструменты. При необходимости используй инструменты для профилирования для поиска узких мест. Полагайся на измерения, а не на догадки и предположения.

Примеры и признаки

Кэширование до того, как провели профилирование. Использование сложных и недоказанных эвристических правил вместо математически верных алгоритмов. Выбор новых, непротестированных фреймворков, которые могут повести себя плохо под нагрузкой.

В чём сложность

Непросто определить, когда оптимизация будет преждевременной. Важно заранее оставлять место для роста. Нужно выбирать решения и платформы, которые позволят легко оптимизировать и расти. Также иногда преждевременную оптимизацию используют в качестве оправдания за плохой код. Например, берут алгоритм O(n2) только из-за того, что алгоритм был бы O(n) сложнее.

Изобретение велосипеда

Смысл этого анти-паттерна в том, что программист разрабатывает собственное решение задачи, для которой уже существуют решения, и зачастую куда более удачные.

Разработчик считает себя умнее, поэтому для каждой задачи пытается придумать собственное решение, несмотря на опыт его предшественников. Чаще всего это приводит только к потере времени и понижению эффективности работы программиста. Ведь решение скорее всего окажется неоптимальным, если вообще будет найдено.

Конечно, нельзя полностью отбрасывать возможность самостоятельного решения, так как это прямой дорогой приведет к программированию копипастом. Разработчик должен ориентироваться в задачах, которые могут предстать перед ним, чтобы грамотно их решить, используя при этом готовые решение или изобретая собственные.

Очень часто причиной этого анти-паттерна является банальная нехватка времени. А время — это деньги.

Изобретение велосипеда с квадратными колесами

Этот анти-паттерн очень тесно связан с простым изобретением велосипеда — это создание собственного плохого решения, когда существует более удачное решение.

Этот анти-паттерн вдвойне забирает время: сначала время тратится на изобретение и реализацию собственного решения, а потом — на его рефакторинг или замену.

Программист должен знать о существовании различных решений для определённых кругов задач, ориентироваться в их преимуществах и недостатках.

Все проблемы, с которыми ты столкнешься как программист, можно поделить на две части:

- эту проблему умные люди решили 30 лет назад
- эту проблему умные люди решили 50 лет назад

Большинство проблем в программировании были **успешно решены еще до твоего рождения**. Не нужно ничего изобретать — просто изучай опыт других людей (для этого и пишут книги).

В 2022 году мы можем отпраздновать такие дни рождения:

- Языки программирования
  - Языку C исполнилось 50 лет (1972)
  - Языку Java исполнилось 27 лет (1995)

    - Языку Python исполнился 31 год (1991)
- Связь
    - Интернету исполнилось 39 лет (1983)
    - Мобильному телефону исполнилось 49 лет (1973)
    - Первую СМС отправили 30 лет назад (1992)
- Паттерны
    - Паттерну MVC исполнилось 44 года (1978)
    - SQL придумали 48 лет назад (1974)
    - Java Beans придумали 26 лет назад (1996)
- Библиотеки
    - Hibernate придумали 21 год назад (2001)
    - Spring придумали 20 лет назад (2002)
    - Tomcat выпустили 23 года назад (1999)
- Операционные системы
    - Unix выпустили в 51 год назад (1971)
    - Windows увидела свет 37 лет назад (1985)
    - Mac OS выпустили 21 год назад (2001)

И все эти вещи были не просто так придуманы, они были разработаны как решения проблем, которые были очень распространены и актуальны в то время.

‹ Предыдущая лекция

Следующая лекция ›