Глава 7

Что такое шаблоны проектирования и зачем они нужны

Большинство задач, с которыми часто приходится сталкиваться программистам, уже давным-давно решены другими членами нашего сообщества. Шаблоны проектирования как раз и являются тем средством, с помощью которого люди могут поделиться друг с другом накопленным опытом. Как только шаблон становится всеобщим достоянием, он обогащает наш язык и позволяет легко поделиться с другими новыми идеями проектирования и их результатами. С помощью шаблонов проектирования просто выделяют общие задачи, определяют проверенные решения и описывают вероятные результаты. Во многих книгах и статьях описываются особенности конкретных языков программирования, имеющиеся функции, классы и методы. А в каталогах шаблонов, наоборот, упор сделан на том, как перейти в ваших проектах от этих основ ("что именно") к пониманию задач и возможных решений ("почему" и "как").

В этой главе мы познакомимся с шаблонами проектирования и рассмотрим некоторые причины их популярности.

Будут рассмотрены следующие темы.

- *Основы шаблонов*. Что такое шаблоны проектирования?
- *Структура шаблона*. Основные элементы шаблона проектирования.
- *Преимущества шаблонов*. Почему шаблоны стоят того, чтобы потратить на них время?

Что такое шаблоны проектирования

В мире программного обеспечения, шаблон — это реальное проявление генетической памяти организации.

— Гради Буч (Grady Booch), из книги *Core J2EE Patterns*

Шаблон — это решение задачи в некотором контексте.

— "Банда четырех" (The Gang of Four),
из книги *Design Patterns: Elements of Reusable Object-Oriented Software*

Как следует из приведенных выше цитат, шаблон проектирования — это задача, взятая из практики передовых программистов, решение которой проанализировано и объяснено.

Задачи имеют свойство повторяться, и веб-программистам приходится решать их снова и снова. Как обработать входящий запрос? Как преобразовать данные в команды для нашей системы? Как ввести данные? Как представить результаты? Со временем мы находим более или менее изящные ответы на эти вопросы и создаем неформальный набор методов, которые затем снова и снова используем в своих проектах. Эти методы — и есть шаблоны проектирования.

С помощью шаблонов проектирования описываются и формализуются типовые задачи и их решения. В результате опыт, который нарабатывается с большим трудом, становится доступным широкому сообществу программистов. Шаблоны должны быть построены, главным образом, по "восходящему", а не "нисходящему" принципу. Их корень — в практике, а не в теории. Но это совсем не означает, что в шаблонах проектирования отсутствует элемент теории (как мы увидим в следующей главе). Шаблоны основаны на реальных методах, используемых реальными программистами. Знаменитый приверженец шаблонов Мартин Фаулер говорит, что он открывает шаблоны, а не создает их. Поэтому многие шаблоны будут вызывать у вас чувство "дежа вю" — ведь вы будете узнавать методы, которые используете сами.

Каталог шаблонов — это не книга кулинарных рецептов. Рецептам можно следовать буквально, а код можно скопировать и вставить в проект с незначительными изменениями. Вам не всегда нужно даже понимать весь код, используемый в этом "рецепте". Шаблоны проектирования описывают *подходы* к решению конкретных задач. Детали реализации могут существенно меняться в зависимости от более широкого контекста. От этого контекста зависит выбор используемого языка программирования, природа приложения, размер проекта и специфика задачи.

Например, предположим, что в проекте требуется создать систему обработки шаблонов. На основании имени файла шаблона вы должны синтаксически проанализировать его содержимое и построить дерево объектов, представляющих найденные теги.

Сначала синтаксический анализатор сканирует текст на предмет поиска триггерных лексем (trigger tokens). Когда он находит соответствие, то передает лексему другому объекту-анализатору, который специализируется на чтении содержимого, расположенного внутри тегов. В результате данные шаблона продолжаются анализироваться до тех пор, пока не произойдет синтаксическая ошибка, будет достигнут их конец либо будет найдена другая триггерная лексема. В случае нахождения такой лексемы объект-анализатор также должен передать ее на обработку соответствующей программе — скорее всего, анализатору аргументов. Все вместе эти компоненты образуют то, что называется рекурсивным нисходящим синтаксическим анализатором.

Итак, вот наши участники: `MainParser`, `TagParser` и `ArgumentParser`. Мы также определили класс `ParserFactory`, который создает и возвращает эти объекты.

Но, конечно, все идет не так гладко, как хотелось бы, и позже, на одном из совещаний, вы узнаете, что в шаблонах нужно поддерживать несколько синтаксисов. И теперь вам нужно создать параллельный набор объектов-анализаторов в соответствии с синтаксисом: `OtherTagParser`, `OtherArgumentParser` и т. д.

Вам поставлена такая задача: нужно генерировать различные наборы объектов в зависимости от конкретной ситуации, и эти наборы объектов должны быть более или менее "прозрачными" для других компонентов системы. Случилось так, что "Банда четырех" в своей книге определила следующую задачу для шаблона Abstract

Factory (Абстрактная фабрика): “Предусмотреть интерфейс для создания семейств связанных или зависимых объектов без указания их конкретных классов”.

Этот шаблон нам прекрасно подходит! По сути нашей задачи мы как раз должны определить и очертить рамки использования данного шаблона. Но решение задачи не имеет ничего общего с операциями вырезания и вставки, как вы увидите в главе 9, где я буду рассказывать о шаблоне Abstract Factory.

Наименование шаблона уже само по себе очень ценно; таким образом создается что-то вроде общего словаря, который годами накапливается и используется в среде профессионалов. Такие условные обозначения помогают в совместных разработках, когда оцениваются и тестируются альтернативные подходы и их различные результаты. Например, при обсуждении семейств альтернативных синтаксических анализаторов вы можете просто сказать коллегам, что система создает каждый набор с помощью шаблона Abstract Factory. Они покивают с умным видом; кто-то сразу поймет, о чем идет речь, а кто-то отметит про себя, что нужно будет узнать об этом позже. Но суть в том, что у этого набора идей и различных результатов есть абстрактный описатель, который способствует созданию более кратких обозначений. Я проиллюстрирую это далее в главе.

И наконец, согласно международному законодательству, некорректно писать о шаблонах, не процитировав Кристофера Александера (Christopher Alexander), профессора архитектуры, работы которого оказали огромное влияние на первых сторонников объектно-ориентированных шаблонов. Вот что он пишет в книге *A Pattern Language* (Oxford University Press, 1977).

> *Каждый шаблон описывает задачу, которая возникает снова и снова, а затем описывает суть решения данной задачи, так что вы можете использовать это решение миллион раз, каждый раз делая это по-разному.*

Важно, что это определение (которое относится к архитектурным задачам и решениям) начинается с формулировки задачи и ее более широкого контекста и движется к решению. В последние годы звучала критика, что шаблонами проектирования злоупотребляют, особенно неопытные программисты. Причина в том, что решения применялись там, где не было сформулировано соответствующей задачи и контекста. Шаблоны — это несколько больше, чем конкретная организация классов и объектов, совместно работающих определенным образом. Шаблоны создаются для определения условий, в которых должны применяться решения, и для обсуждения результатов этих решений.

В данной книге основной упор будет сделан на особенно влиятельном направлении в сфере шаблонов: форме, описанной в книге Эриха Гаммы (Erich Gamma), Ричарда Хелма (Richard Helm), Ральфа Джонсона (Ralph Johnson) и Джона Влиссидеса (John Vlissides) *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1995). Она посвящена использованию шаблонов при разработке объектно-ориентированного программного обеспечения, и в ней описываются некоторые классические шаблоны, которые присутствуют в большинстве современных объектно-ориентированных проектов.

Книга “Банды четырех” очень важна не только потому, что в ней описываются основные шаблоны, но и потому, что в ней рассматриваются принципы проектирования, которые положены в основу этих шаблонов. Некоторые из этих принципов мы рассмотрим в следующей главе.

На заметку! Шаблоны, описанные "Бандой четырех", а также в данной книге, — это настоящие примеры языка шаблонов, т.е. каталога задач и решений, объединенных вместе таким образом, чтобы они дополняли друг друга и формировали взаимозависимое целое. Существуют языки шаблонов для других сфер задач, таких как визуальное проектирование и менеджмент проектов (и, конечно, архитектура). Но когда в этой книге я обсуждаю шаблоны проектирования, имею в виду задачи и решения в области разработки объектно-ориентированного программного обеспечения.

Обзор шаблонов проектирования

По сути, шаблон проектирования состоит из четырех частей: имени, формулировки задачи, описания решения и результатов.

Имя

Выбор имени для шаблона очень важен. Имена обогащают язык программистов; несколько коротких слов могут служить для обозначения довольно сложных задач и решений. Имена должны сочетать в себе краткость и описательность. "Банда четырех" утверждает: "Поиск хороших имен был одной из самых трудных задач при разработке нашего каталога".

Мартин Фаулер согласен с этим утверждением: "Имена шаблонов чрезвычайно важны, потому что одна из целей шаблонов — создать словарь, позволяющий разработчикам общаться более эффективно" (*Patterns of Enterprise Application Architecture*[1], Addison-Wesley Professional, 2002).

В книге *Patterns of Enterprise Application Architecture* Мартин Фаулер совершенствует шаблон доступа к базе данных, который я впервые встретил в книге Дипака Алура (Deepak Alur), Дэна Малкса (Dan Malks) и Джона Крупи (John Crupi) *Core J2EE Patterns* (Prentice Hall, 2003). Фаулер определяет два шаблона, которые описывают специализацию старого шаблона. Логика этого подхода полностью правильна. Один из новых шаблонов моделирует объекты предметной области, в то время как другой — таблицы базы данных (а в предыдущей работе это различие было нечетким). Было трудно заставить себя мыслить в категориях новых шаблонов. Я использовал имя первоначального шаблона при проектировании и в документации так долго, что оно прочно вошло в мой язык.

Формулировка задачи

Независимо от изящества решений (а некоторые из них действительно очень изящны), формулировка задачи и ее контекста — это основа шаблона. Сформулировать задачу гораздо труднее, чем применить какое-либо решение из каталога шаблонов. Это одна из причин того, что некоторые шаблоны могут применяться неправильно.

В шаблонах очень тщательно описываются условия задачи. Сначала кратко описывается сама задача, затем ее контекст, обычно приводится типичный пример и одна или несколько диаграмм. Анализируется специфика задачи, ее различные проявления. Также описываются все признаки, которые могут помочь при идентификации задачи.

[1] *Архитектура корпоративных программных приложений*, ИД "Вильямс", 2004.

Решение

Решение сначала кратко описывается вместе с задачей. Оно также описывается подробно, как правило, с использованием UML-класса и диаграмм взаимодействия. В шаблон обычно включается пример кода.

Но хотя код может присутствовать, в качестве решения никогда нельзя использовать метод "вырезать и вставить". Помните, что шаблон описывает подход к решению задачи, поскольку в реализации могут быть сотни нюансов. Представьте, что перед вами — инструкции о том, как сеять хлеб. Если вы просто слепо выполните все указания, то, скорее всего, будете голодать после сбора урожая. Гораздо полезнее будет подход, основанный на шаблоне, в котором описываются различные условия его применения. Основное решение задачи (заставить хлеб расти) всегда будет одним и тем же (посеять семена, поливать, собрать урожай), но реальные шаги, которые надо будет предпринять, зависят от всевозможных факторов, таких как тип почвы, местность, местные вредные насекомые и т.д.

Мартин Фаулер называет решения, описанные в шаблонах, "полусырыми", т.е. программист должен взять идею решения и закончить ее самостоятельно.

Результаты

Каждое решение по проектированию, которое вы принимаете, будет иметь широкий набор результатов. Конечно, в него должно быть включено удовлетворительное решение поставленной задачи. Решение, однажды примененное, может идеально подходить для работы с другими шаблонами. Но его нужно применять с особой осторожностью.

Формат "Банды четырех"

Сейчас, когда я пишу эти строки, передо мной на рабочем столе находятся пять каталогов шаблонов. Даже поверхностный взгляд на шаблоны в каждом каталоге говорит о том, что ни в одном из них не используется такая же структура, как в других. Одни более формализованы, другие очень детализированы (содержат множество подразделов), а третьи более беспорядочны.

Существует ряд четко определенных структур шаблонов, включая первоначальную форму, разработанную Кристофером Александером (александрийская форма), и описательный подход, который используют в Портлендском хранилище шаблонов (Portland Pattern Repository) (портлендская форма). Поскольку "Банда четырех" очень влиятельна и мы будем рассматривать многие из описанных ими шаблонов, давайте изучим несколько разделов, которые они включили в свои шаблоны.

- *Предназначение.* Краткая формулировка цели шаблона. Вы должны с первого взгляда понять суть шаблона.
- *Мотивация.* Задача описывается, как правило, для типичной ситуации. На конкретных примерах легче понять, что представляет собой шаблон.
- *Применимость.* Исследование различных ситуаций, в которых можно применить шаблон. В то время как в разделе мотивации описывается типичная задача, в данном разделе определяются конкретные ситуации и оцениваются преимущества решения в контексте каждой из них.
- *Структура/взаимодействие.* Эти разделы могут содержать UML-класс и диаграммы взаимодействия, описывающие отношения между классами и объекты в решении.

- *Реализация.* В данном разделе рассматриваются детали решения. В нем исследуются любые вопросы, которые могут возникнуть во время применения данного метода, и предоставляются советы по его применению.
- *Пример кода.* Я всегда сразу перехожу к этому разделу. Я считаю, что простой пример кода помогает разобраться в шаблоне. Этот пример часто сведен до минимальной основы с целью демонстрации самой сути решения. Пример может быть написан на любом объектно-ориентированном языке. Конечно, в данной книге всегда будет использоваться PHP.
- *Примеры применения.* Реальные системы, в которых встречается шаблон (задача, контекст и решение). Некоторые люди считают, что, для того чтобы шаблон был настоящий, он должен присутствовать, по меньшей мере, в трех широко известных контекстах. Это иногда называется "правилом трех".
- *Родственные шаблоны.* Некоторые шаблоны влекут за собой другие. Применяя одно решение, вы можете создать контекст, в котором станет полезным другое. Эти связи и исследуются в данном разделе. Здесь также могут обсуждаться шаблоны, у которых есть что-то общее в задаче и решении, а также любые "предшественники": шаблоны, определенные где-то еще, на основе которых строится текущий шаблон.

Зачем используются шаблоны проектирования

Так в чем заключаются преимущества шаблонов? Учитывая, что шаблон — это поставленная задача и описанное решение, ответ, казалось бы, очевиден. Шаблоны помогают решать распространенные задачи. Но, конечно, шаблон — это нечто большее.

Шаблоны определяют задачи

Сколько раз вы доходили до некоторого этапа в проекте и обнаруживали, что дальше идти некуда? Вполне вероятно, вам нужно вернуться немного назад, прежде чем начать снова.

Определяя распространенные задачи, шаблоны помогают улучшить проект. И иногда первый шаг к решению — это осознание того, что есть проблема.

Шаблоны определяют решения

Определив и осознав проблему (и убедившись, что это именно та проблема), с помощью шаблона вы получаете доступ к решению, а также к анализу результатов его использования. Хотя шаблон не избавляет вас от необходимости рассмотреть последствия выбранного решения, вы, по крайней мере, будете уверены, что используете проверенный метод.

Шаблоны не зависят от языка программирования

Шаблоны определяют объекты и решения с помощью "объектно-ориентированных" терминов. Это означает, что многие шаблоны одинаково применимы ко многим языкам программирования. Когда я впервые начал использовать шаблоны, изучал примеры кода на C++ и Smalltalk и писал свои решения на Java. На другие языки шаблоны переносятся с некоторыми изменениями в их реализации или в получаемых результатах, но они всегда остаются правомерными. В любом случае,

шаблоны помогают при переходе от одного языка к другому. Точно так же приложение, построенное на четких принципах объектно-ориентированного проектирования, легко перенести с одного языка на другой (хотя при этом всегда остаются проблемы, которые нужно решать).

Шаблоны определяют словарь

Предоставляя разработчикам имена методов, шаблоны обогащают процесс общения и передачи информации. Давайте представим совещание, посвященное вопросам проектирования. Я уже описал мое решение с помощью шаблона Abstract Factory и теперь должен изложить стратегию обработки данных, которые собирает система. Я рассказываю о своих планах Бобу.

Я: Я собираюсь использовать шаблон Composite.

Боб: Мне кажется, ты не продумал это как следует.

Что ж, Боб не согласен со мной. Он никогда со мной не согласен. Но он знает, о чем я говорю и почему моя идея плоха. А теперь проиграем эту сцену еще раз без использования словаря.

Я: Я собираюсь использовать три объекта, в которых используется один и тот же тип данных. В интерфейсе типа будут определены методы для добавления дочерних объектов собственного типа. Таким образом, мы можем построить сложную комбинацию реализованных объектов во время выполнения программы.

Боб: Да?

Шаблоны, или методики, которые в них описаны, имеют тенденцию к взаимодействию. Так шаблон Composite взаимодействует с шаблоном Visitor.

Я: А затем можно использовать шаблон Visitors для получения итоговых данных.

Боб: Ты упустил суть.

Не будем обращать внимание на Боба. Я не буду долго и мучительно описывать версию этого разговора без использования терминов шаблонов. В главе 10 я расскажу о шаблоне Composite, а в главе 11 — о шаблоне Visitor.

Суть в том, что эти методики можно использовать и без языка шаблонов. Сами методики всегда появляются *до* того, как им будет назначено имя и они будут организованы в виде шаблона. Если шаблона нет, его могут разработать. А любой инструмент, который используется достаточно часто, в конце концов, получит имя.

Шаблоны проверяются и тестируются

Итак, если с помощью шаблонов, по сути, описываются лучшие методики решения задачи, то будет ли выбор имени самым важным элементом при создании каталогов шаблонов? В некотором смысле это так. Шаблоны представляют собой лучшие методики в сфере объектно-ориентированного программирования. Для некоторых очень опытных программистов это может показаться упражнением по перекомпоновке очевидных вещей. Но для всех остальных шаблоны — это доступ к задачам и решениям, которые в противном случае приходилось бы находить нелегким путем.

Шаблоны делают проект доступным. Каталоги шаблонов появляются для все большего количества специализированных областей, поэтому даже очень опытный специалист может извлечь из них пользу. Например, программист графического пользовательского интерфейса (GUI) может быстро получить доступ к задачам и

решениям при создании корпоративного приложения. Веб-программист сможет быстро наметить стратегию того, как избежать ошибок и подводных камней, которые могут возникнуть в проектах для PDA и мобильных телефонов.

Шаблоны предназначены для совместной работы

По своей природе шаблоны должны быть наиболее общими и компонуемыми. Это означает, что у вас должна быть возможность применить один шаблон и тем самым создать условия, подходящие для применения другого. Иначе говоря, используя шаблон, вы можете обнаружить, что для вас открываются другие двери.

Каталоги шаблонов обычно разрабатываются с расчетом на такого рода совместную работу, и возможность компоновки шаблонов всегда документируется в самом шаблоне.

Шаблоны способствуют хорошим проектам

В шаблонах проектирования демонстрируются и применяются принципы объектно-ориентированного проектирования. Поэтому изучение шаблонов проектирования может дать больше, чем конкретное решение в некотором контексте. В результате у вас может появиться новое видение того, как можно объединять объекты и классы для достижения поставленной цели.

PHP и шаблоны проектирования

В этой главе мало что относится конкретно к PHP, что в некоторой степени характерно для данной темы. Многие шаблоны применимы ко многим языкам программирования, в которых есть возможности работы с объектами, достаточно только решить некоторые вопросы реализации (если они вообще возникают).

Но, конечно, это не всегда так. Некоторые шаблоны корпоративных приложений прекрасно используются в тех языках, где прикладной процесс продолжает свою работу между запросами к серверу. PHP работает иначе. Выполнение нового сценария начинается для каждого запроса. Это означает, что с некоторыми шаблонами нужно обращаться более аккуратно. Например, для реализации шаблона Front Controller часто требуется довольно много времени. Хорошо, если инициализация имеет место один раз при запуске приложения, но гораздо хуже, если она происходит при каждом запросе. Это не значит, что нельзя использовать данный шаблон; я применял его в своей практике с очень хорошими результатами. Просто при обсуждении шаблона мы должны обязательно принять во внимание вопросы, связанные с PHP. PHP формирует контекст для всех шаблонов, которые изучаются в данной книге.

Выше в данном разделе я уже упоминал о языках программирования, в которых есть возможности работы с объектами. В PHP можно программировать, вообще не определяя никаких классов (хотя, учитывая продолжающееся развитие PEAR, вероятно, вы будете в некоторой степени оперировать объектами). Хотя в данной книге мы почти полностью концентрируемся на объектно-ориентированных решениях задач программирования, не считайте это залпом из всех орудий в войне между приверженцами разных стилей программирования. Шаблоны и PHP могут создать мощный союз, поэтому они и составляют основу данной книги. Тем не менее они могут прекрасно сосуществовать с другими, более традиционными, подходами. Убедительное доказательство тому — PEAR. В пакетах PEAR очень изящно используются шаблоны проектирования. Они склонны быть объектно-ориентиро-

ванными по своей природе. И это делает их более, а не менее, полезными в процедурных проектах. Поскольку пакеты PEAR автономны и их сложность скрыта за четко описанным интерфейсом, их легко включить в проект любого типа.

Резюме

В этой главе мы познакомились с шаблонами проектирования, рассмотрели их структуру (с помощью формы "Банды четырех") и изучили некоторые причины, по которым у вас может возникнуть необходимость использовать шаблоны проектирования в своих сценариях.

Важно помнить, что шаблоны проектирования — это не набор "готовых" решений, которые можно объединять, как компоненты, при построении проекта. Шаблоны — это предлагаемые подходы к решению распространенных задач. В этих решениях воплощены некоторые основные принципы проектирования. Именно их мы и будем изучать в следующей главе.