ГЛАВА 1

Введение

В современном мире постоянно работающих приложений и программных интерфейсов (API) к ним предъявляются такие требования, которые пару десятилетий назад предъявлялись только к небольшому количеству наиболее важных систем. Аналогичным образом наличие возможности быстрого, «вирусного» роста популярности сервиса означает, что любое приложение должно создаваться с расчетом на почти мгновенное масштабирование в ответ на увеличивающийся пользовательский спрос. Эти ограничения и требования означают, что почти каждое разрабатываемое приложение, будь то мобильная клиентская программа или сервис обработки платежей, должно быть распределенной системой.

Но строить распределенные системы непросто. Как правило, это единичные заказные системы. В этом смысле разработка распределенных систем поразительно похожа на разработку программного обеспечения (ПО) в тот период, когда еще не существовало современных объектно-ориентированных языков программирования. К счастью, как и в случае с созданием объектно-ориентированных языков, имел место технический прогресс, существенно снизивший трудоемкость построения распределенных систем. В данном случае он связан с растущей популярностью контейнеров и инструментов оркестрирования. Подобно объектам в объектно-ориентированном программировании, контейнеризированные строительные блоки — основа разработки повторно используемых компонентов и паттернов проектирования. Они существенно упрощают создание надежных распределенных систем и делают его более доступным для начинающих разработчиков. Далее будет кратко описана история разработок, приведших к современному состоянию отрасли.

Краткая история разработки систем

Первое время вычислительные машины создавались для какой-то одной цели: расчета артиллерийских таблиц, прогнозирования приливов и отливов, взлома шифров и других точных, сложных, но рутинных математических вычислений. Спустя годы специализированные машины превратились в программируемые компьютеры общего назначения. Те со временем перешли от выполнения одной программы на одной машине к параллельному выполнению многих программ на одной машине с помощью операционных систем с разделением времени. Эти машины все еще были отделены друг от друга.

Постепенно компьютеры стали объединяться в сети, в результате чего появились клиент-серверные архитектуры. Относительно маломощные компьютеры на рабочих местах получили доступ к вычислительным ресурсам мощных мейнфреймов, находящихся в других помещениях или даже зданиях. И хотя такой вид клиент-серверного программирования был несколько сложнее написания программы для одного компьютера, он все еще был относительно прост для понимания. Клиент (-ы) делал (-и) запросы, а сервер (-ы) их обслуживал (-и).

Рост Интернета и появление в начале 2000-х годов крупных центров обработки данных (ЦОД), состоящих из тысяч относительно недорогих массово производимых компьютеров, которые объединялись в сеть, привели к широкому распространению распределенных систем. В отличие от клиент-серверных архитектур распределенные приложения состоят либо из нескольких разных приложений, либо из нескольких копий одного приложения, работающих на разных компьютерах. Взаимодействуя, они реализуют некоторый сервис, например веб-поисковик или систему розничных продаж.

В силу своего распределенного характера такие системы при грамотной их структуризации более надежны по определению. А при грамотно спроектированной архитектуре системы масштабируемой становится и ее команда разработчиков. К сожалению, за эти преимущества приходится платить. Распределенные системы существенно сложнее в проектировании, построении и отладке. При построении надежной распределенной системы к инженерно-техническим

навыкам специалистов предъявляются существенно более высокие требования, чем при построении локальных приложений. Так или иначе, потребность в надежных распределенных системах продолжает расти. Следовательно, возникает необходимость в соответствующих инструментах, паттернах и практиках их построения.

К счастью, современные технологии упрощают разработку распределенных систем. В последние годы контейнеры, их образы и оркестраторы стали популярными в силу того, что являются неотъемлемыми составными частями надежных распределенных систем. Взяв за основу контейнеры и оркестраторы контейнеров, мы можем создать набор повторно используемых компонентов и паттернов проектирования. Такие паттерны и компоненты составляют инструментарий, необходимый для разработки более эффективных надежных систем.

Краткая история паттернов проектирования в разработке ПО

Это не первый случай, когда подобная трансформация происходит в индустрии разработки ПО. Чтобы лучше понять, как паттерны, практики и повторно используемые компоненты изменили разработку систем, имеет смысл взглянуть на то, как подобные трансформации происходили в прошлом.

Формализация алгоритмического программирования

Люди писали программы задолго до опубликования Дональдом Кнутом сборника «Искусство программирования»[1] в 1962 году. Тем не менее это событие стало важной вехой в развитии информатики. В частности, описанные в книгах Кнута алгоритмы не ориентированы на какой-либо компьютер, а предназначены для обучения читателя алгоритмическому мышлению. Эти алгоритмы могут быть адаптированы к конкретной компьютерной архитектуре или к конкретной задаче, решаемой читателем. Такая формализация была

[1] *Кнут Д.* Искусство программирования. В 4 т. — М., 2019.

важна не только потому, что предоставляла разработчикам общий инструментарий для написания программ, но и потому, что демонстрировала существование универсальных идей, которые можно применять в разнообразных контекстах. Понимание алгоритмов имеет ценность само по себе, безотносительно к какой-либо решаемой с их помощью задаче.

Паттерны в объектно-ориентированном программировании

Если появление книг Кнута стало важной вехой в теории компьютерного программирования, то алгоритмы — ключевой составляющей его развития. Однако по мере роста сложности компьютерных программ и увеличения численного состава разрабатывающих их команд с единиц до сотен и тысяч стало ясно, что языков процедурного программирования и алгоритмов уже недостаточно для решения насущных задач. Эти изменения привели к появлению и развитию объектно-ориентированных языков программирования, которые уравняли в правах с алгоритмами данные, повторное использование и расширяемость.

В ответ на эти изменения изменениям подверглись также паттерны и практики программирования. В начале и середине 1990-х годов произошел взрывной рост количества книг об объектно-ориентированном программировании. Наиболее известна из них книга «банды четырех» «Паттерны объектно-ориентированного проектирования»[1]. *Паттерны проектирования* привнесли в работу программистов инфраструктуру и общий язык. В этой книге описывается набор интерфейсных паттернов, которые можно использовать в различных контекстах. Благодаря развитию объектно-ориентированного программирования в целом и интерфейсов в частности появилась возможность реализовать такие паттерны в виде повторно используемых библиотек. Эти библиотеки можно написать единожды и затем многократно использовать, экономя тем самым время и повышая надежность.

---

[1] *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Паттерны объектно-ориентированного проектирования. — СПб.: Питер.

Расцвет программного обеспечения с открытым исходным кодом

Делиться исходным кодом с другими разработчиками было принято со времен появления вычислительной техники. Формальные организации в поддержку свободного программного обеспечения существовали с середины 1980-х годов. Но именно в конце 1990-х — начале 2000-х резко возросло количество разработчиков и потребителей программного обеспечения с открытым исходным кодом. Хотя движение open source лишь относительно связано с разработкой паттернов проектирования распределенных систем, его заслуга состоит в том, что именно сообщества open source показали миру: создание программного обеспечения в целом и распределенных систем в частности — труд коллективный. Здесь важно отметить, что все технологии контейнеров, лежащие в основе паттернов проектирования, описанных в этой книге, разрабатывались и выпускались именно как программное обеспечение с открытым исходным кодом. Ценность паттернов для документирования и усовершенствования практик разработки распределенных программных систем становится наиболее очевидной именно с точки зрения коллективной разработки.

Что такое паттерн проектирования распределенной системы? Написано множество инструкций по установке конкретных распределенных систем (например, баз данных NoSQL). Кроме того, у каждого набора систем (например, стека MEAN1) есть свои правила установки. Но, когда я говорю о паттернах, я имею в виду обобщенные схемы организации распределенных систем, не зависящие от конкретных технологий или приложений. Цель паттерна — предоставить общие предложения по архитектуре системы, задать ее ориентировочную структуру. Надеюсь, что эти паттерны направят ход ваших мыслей в верную сторону и окажутся применимы в широком спектре приложений и программных сред.

Ценность паттернов, практик и компонентов

Прежде чем тратить ценное время на чтение книги о наборе паттернов, которые, с моих слов, усовершенствуют ваши подходы к работе, научат вас новым приемам разработки и — давайте посмотрим правде в глаза — изменят вашу жизнь, имеет смысл спросить: «А зачем?»

Что такого есть в паттернах и методиках разработки, что может поменять подход к проектированию и компоновке программного обеспечения? В этом разделе я объясню, почему считаю их важными. Надеюсь, мои доводы убедят вас прочесть книгу полностью.

Стоя на плечах гигантов[1]

Начнем с того, что паттерны проектирования распределенных систем позволяют, образно говоря, стоять на плечах гигантов. Задачи, которые мы решаем, или системы, которые мы создаем, нечасто становятся действительно уникальными. В конечном счете собранные воедино компоненты и общая бизнес-модель, которую позволяет организовать разрабатываемое программное обеспечение, являются чем-то новым. Но то, как система построена, и те проблемы, с которыми она сталкивается в своем стремлении быть надежной и масштабируемой, отнюдь не новы.

В этом, стало быть, состоит первая ценность паттернов: они позволяют учиться на чужих ошибках. Возможно, вы никогда раньше не разрабатывали распределенные системы. Возможно, вы никогда раньше не разрабатывали определенный вид распределенных систем. Вместо того чтобы надеяться на опыт вашего коллеги в этой области или учиться на ошибках, которые совершали другие, вы можете обратиться за помощью к паттернам.

Изучение паттернов проектирования распределенных систем — то же самое, что изучение любых других передовых практик программирования. Оно ускоряет разработку программного обеспечения, не требуя наличия непосредственного опыта разработки систем, исправления ошибок и набивания собственных шишек.

Общий язык обсуждения подходов к разработке

Возможность учиться, а также лучше и быстрее понимать устройство распределенных систем — лишь часть преимуществ от наличия общего набора паттернов проектирования. Паттерны ценны даже

[1] Отсылка к высказыванию, приписываемому Ньютону: «Если я видел дальше других, то потому, что стоял на плечах гигантов».

для опытных разработчиков распределенных систем, уже хорошо их понимающих. Паттерны предоставляют общий словарь, позволяющий нам быстро понимать друг друга. Понимание — основа обмена знаниями и дальнейшего обучения.

Для того чтобы было понятнее, представим, что мы оба используем один и тот же инструмент для постройки дома. Я называю его «сепулька», а вы называете его «бутявка». Как долго мы будем спорить о преимуществах «сепулек» над «бутявками» или пытаться объяснить различие в их свойствах, пока не придем к тому, что это одно и то же? Только когда мы поймем, что «сепульки» и «бутявки» — одно и то же, мы сможем учиться на опыте друг друга.

При отсутствии общего словаря много времени тратится либо на споры в поисках «насильственного согласия», либо на объяснение понятий, которые остальные понимают, но называют по-другому. Следовательно, ценность паттернов состоит еще и в том, что они обеспечивают наличие общего набора понятий и их определений, позволяющего не тратить время на дискуссии об именах, а перейти к обсуждению деталей реализации основных идей.

За то короткое время, что я работал над технологией контейнеров, я убедился в этом. На тот момент идея *контейнеров-прицепов* (*sidecar*; будут описаны в главе 3) прочно укрепилась в сообществе «контейнерщиков». Благодаря этому не было необходимости тратить время на разъяснение того, что значит быть контейнером-прицепом, а вместо этого можно было перейти к обсуждению того, как использовать этот паттерн для решения конкретной задачи. «А вот если мы здесь используем паттерн Sidecar...» — «Ага, кажется, я знаю, какой контейнер отлично подойдет для этой задачи». Этот пример подводит нас к третьему ценному аспекту паттернов проектирования — возможности создания повторно используемых компонентов.

Общие повторно используемые компоненты

Паттерны позволяют учиться на чужом опыте и предоставляют общий язык для обсуждения тонкостей построения систем, но, помимо этого, они дают программисту еще один инструмент — возможность выявлять общие компоненты, которые достаточно реализовать однократно.

Если бы мы писали весь необходимый программный код самостоятельно, то никогда бы ничего не доделали. Более того, у нас едва бы получалось начать. Каждая созданная или создаваемая на сегодня система является результатом тысяч, а то и сотен тысяч человеколет работы. Код операционных систем, драйверов принтеров, распределенных баз данных, исполнительных сред контейнеров и их оркестраторов — все, что мы сегодня строим, создается на основе совместно используемых библиотек и повторно используемых компонентов.

Паттерны — основа формирования и развития таких компонентов. Формализация алгоритмов привела к созданию повторно используемых реализаций сортировки и других канонических алгоритмов. Благодаря выявлению интерфейсных паттернов проектирования появился целый ряд объектно-ориентированных библиотек, их реализующих.

Выявление базовых паттернов проектирования распределенных систем позволяет создавать разделяемые общие компоненты таких систем. Реализация этих паттернов в виде контейнеров с HTTP-интерфейсом дает возможность использовать их в различных языках программирования. И конечно же, разработка, ориентированная на построение повторно применяемых компонентов, позволяет улучшать качество каждого из них, поскольку в используемом многими людьми коде более высока вероятность обнаружения ошибок и недостатков.

Недавняя серия атак на цепочки поставок программного обеспечения показала, что управление зависимостями и обеспечение их безопасности являются важнейшей частью защиты наших приложений. В контексте безопасности цепочек поставок ПО эти совместно используемые компоненты приобретают еще большее значение. Каждая библиотека или приложение, которые мы используем, создают больше зависимостей — и, следовательно, больше рисков. Опора на единую общую реализацию ключевой идеи сокращает общий объем программного обеспечения, от которого должен зависеть мир, а сосредоточение внимания на нескольких зависимостях значительно повышает их защищенность от атак по цепочке поставок ПО.

Резюме

Распределенные системы обеспечивают более высокий уровень надежности, гибкости и масштабируемости, ожидаемый от современных компьютерных программ. Проектирование распределенных систем пока остается «черной магией» для посвященных, а не наукой, доступной непрофессионалу. Выявление общих паттернов и практик упорядочило и усовершенствовало подходы к алгоритмическому и объектно-ориентированному программированию. Эта книга призвана сделать то же для распределенных систем. Поехали!