Архітектурні стилі

Зміст

Вступ

На цей час існують узагальнені шаблони і принципи, які використовуються при проектуванні застосування сьогодні.
Їх називають архітектурними стилями чи парадигмами.

До них відносяться такі шаблони як:
клієнт/сервер,
багаторівнева архітектура,
компонентна архітектура,
архітектура, основою якої є шина повідомлень, та
сервісно-орієнтована архітектура.

Важливо розуміти, що стилі описують різні аспекти застосувань. Деякі архітектурні стилі описують схеми розгортання, дякі питання структури та дизайну, інші аспекти зв'язку.

Таким чином, типове застосування, як правило, використовує поєднання декількох стилів

Що таке архітектурний стиль

Архітектурний стиль, іноді називаємий архітектурним шаблоном, - це набір принципів, високорівнева схема, яка забезпечує абстрактну інфраструктуру для сімейства систем.

Архітектурний стиль покращує секціонування і сприяє повторному використанню дизайну завдяки забезпеченню рішень проблем, які часто зустрічаються.

Що таке архітектурний стиль

Гарлан (Garlan) і Шоу (Shaw) визначають архітектурний стиль як:
«... Сімейство систем з точки зору схеми організації структури. Точніше кажучи, архітектурний стиль визначає набір компонентів і з'єднань, які можуть використовуватися в примірниках цього стилю, а також ряд обмежень по їх можливим поєднанням. Сюди можуть відноситися топологічні обмеження на архітектурні рішення (наприклад, не використовувати цикли). Опис стилю також може включати і інші обмеження, такі як, скажімо, необхідність обробки семантики виконання.»

Що таке архітектурний стиль

Найголовніша перевага розуміння архітектурних стилів - спільна мова. Вони дають можливість вести діалог, не торкаючись технологій, тобто обговорювати схеми і принципи, не вдаючись у деталі.

Архітектурні стилі можна організувати по їх фокусу:

Категорія	Архітектурні стилі
зв'язок	Сервісно-орієнтована архітектура (SOA), шина повідомлень
розгортання	Клієнт / сервер, N-рівнева, 3-рівнева
предметна галузь	Дизайн на основі предметної області (Domain Driven Design)
структура	Компонентна, об'єктно-орієнтована, багаторівнева архітектура

Огляд основних архітектурних стилів

Клієнт / сервер
Система поділяється на два застосування, де клієнт виконує запити до сервера. У багатьох випадках в ролі сервера виступає база даних, а логіка застосування представлена процедурами зберігання.

Компонентна архітектура
Дизайн застосування розкладається на функціональні або логічні компоненти з можливістю повторного їх використання, які мають та надають ретельно опрацьовані інтерфейси зв'язку.

Дизайн на основі предметної галузі
Об'єктно-орієнтований архітектурний стиль, орієнтований на моделювання сфери ділової активності і визначає бізнес-об'єкти на підставі сутностей цієї сфери.

Багатошарова архітектура
Функціональні області застосування поділяються на багатошарові групи (рівні).

Огляд основних архітектурних стилів

Шина повідомлень
Архітектура, заснована на шині повідомлень (message bus), має на увазі наявність загального комунікаційного каналу, використовуючи який компоненти обмінюються інформацією. Компонент поміщає повідомлення в комунікаційний канал, після цього повідомлення розсилається всім зацікавленим компонентам. Дана архітектура спрямована на рішення комунікаційних завдань.

N-рівнева / 3-рівнева (вузлова)
Функціональність виділяється в окремі сегменти, багато в чому аналогічно багатошаровому стилю, але в даному випадку сегменти фізично розташовуються на різних комп'ютерах.

Об'єктно-орієнтована
Парадигма проектування, заснована на розподілі відповідальності застосування або системи між окремими самостійними об'єктами, які придатні для повторного використання, і які містять дані і поведінку.

Сервісно-орінеєнтована архітектура (SOA)
Описує додатки, що надають і споживають функціональність у вигляді сервісів за допомогою контрактів і повідомлень.

Поєднання архітектурних стилів

Архітектура програмної системи практично ніколи не обмежена лише одним архітектурним стилем, часто вона є поєднанням архітектурних стилів, що утворюють повну систему.

Наприклад, може існувати SOA-дизайн, що складається із сервісів, при розробці яких використовувалася багатошарова архітектура і об'єктно-орієнтований архітектурний стиль.

Поєднання архітектурних стилів також корисно при побудові Інтернет Веб-застосувань, де можна досягти ефективного розподілу функціональності за рахунок використання багатошарового архітектурного стилю.

Таким чином можна відокремити логіку уявлення від бізнес-логіки і логіки доступу до даних.
Вимоги безпеки організації можуть обумовлювати або 3-рівневе (вузлове) розгортання програми, або розгортання з більш ніж трьома рівнями (вузлами). Рівень представлення може розгортатися в прикордонній мережі, розташованої між внутрішньою мережею організації та зовнішньою мережею.

Поєднання архітектурних стилів

Створюючи настільне застосування, можна реалізувати клієнта, який буде відправляти запити до програми на сервері.

В цьому випадку розгортання клієнта і сервера можна виконати за допомогою архітектурного стилю клієнт / сервер і використовувати компонентну архітектуру для подальшого розкладання дизайну на незалежні компоненти, що надають відповідні інтерфейси.

Застосування об'єктно-орієнтованого підходу до цих компонентів підвищить можливості повторного використання, тестування і гнучкість

Поєднання архітектурних стилів

На вибір архітектурних стилів впливає безліч факторів:
здатність організації до:
проектування;
реалізації,
можливості і
досвід розробників, а також
обмеження:
інфраструктури і
організації.

Архітектура клієнт / сервер

Клієнт / серверна архітектура описує розподілені системи, які складаються з окремих клієнта і сервера і мережі, що з'єднує їх.

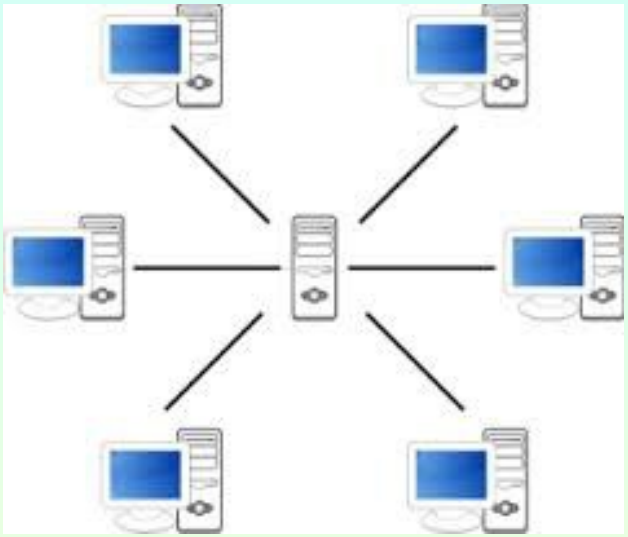
Найпростіша форма системи клієнт / сервер, що називається 2-рівневої архітектурою - це серверний додаток, до якого безпосередньо звертаються безліч клієнтів.

Історично архітектура клієнт / сервер являє собою настільне застосування з графічним UI, яке обмінюється даними з сервером бази даних, на якому у формі процедур розташовується основна частина бізнес-логіки, або з виділеним файловим сервером.
Архітектурний стиль клієнт / сервер описує відносини між клієнтом і одним або більше серверами, де клієнт ініціює один або більше запитів (можливо, з використанням графічного UI), очікує відповіді і обробляє їх при отриманні. Зазвичай сервер авторизує користувача і потім проводить обробку, необхідну для отримання результату.
Для зв'язку з клієнтом сервер може використовувати широкий діапазон протоколів і форматів даних.

Архітектура клієнт / сервер

На сьогоднішній день прикладами архітектурного стилю клієнт / сервер можуть служити:
- Веб-застосування, які працюють в Інтернеті або внутрішніх мережах організацій;
- настільні застосування, які здійснюють доступ до мережевих сервісів даних;
- застосування, що виконують доступ до віддалених сховищ даних (такі як програми читання електронної пошти, FTP-клієнти (FTP — File Transfer Protocol) і засоби доступу до баз даних);
- інструменти та утиліти для роботи з віддаленими системами (такі як засоби управління системою і засоби моніторингу мережі).

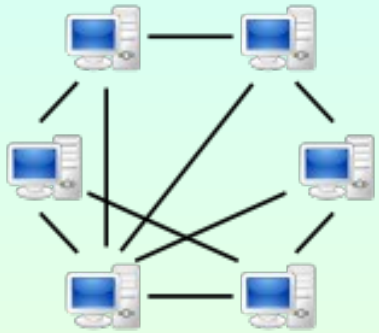
Архітектура клієнт / сервер



До інших різновидів стилю клієнт / сервер відносяться:
Системи клієнт-черга-клієнт.

Цей підхід дозволяє клієнтам обмінюватися даними з іншими клієнтами через чергу на сервері. Клієнти можуть читати дані з і відправляти дані на сервер, який виступає в ролі простої черги для зберігання даних. Завдяки цьому клієнти можуть розподіляти і синхронізувати файли і відомості. Іноді таку архітектуру називають пасивною чергою

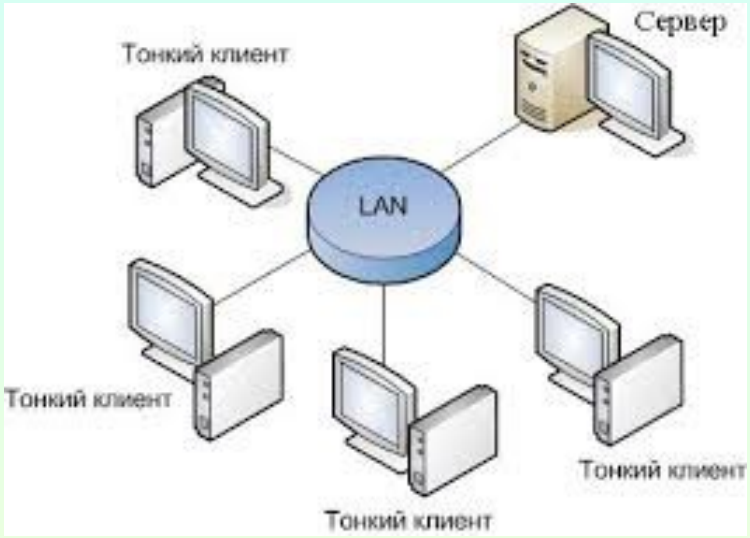
Архітектура клієнт / сервер



Одноранговi (Peer-to-Peer, P2P) застосування

Створені на базі клієнт-черга-клієнт, стиль P2P дозволяє клієнту і серверу обмінюватися ролями з метою розподілу і синхронізації файлів і даних між безліччю клієнтів. Ця схема розширює стиль клієнт / сервер, додаючи множинні відповіді на запити, спільно використовувані дані, виявлення ресурсів і стійкість при видаленні учасників мережі.

Архітектура клієнт / сервер



Сервери застосувань.

Спеціалізований архітектурний стиль, при якому застосування і сервіси розміщуються і виконуються на сервері, і тонкий клієнт виконує доступ до них через браузер або спеціальне встановлене на клієнті ПЗ.

Прикладом є клієнт, який працює з додатком, що виконується на сервері, через таке середовище як Terminal Services (Служби терміналів)

Архітектура клієнт / сервер

Основні переваги архітектурного стилю клієнт / сервер:

Велика безпека.
Всі дані зберігаються на сервері, який зазвичай забезпечує більший контроль безпеки, ніж клієнтські комп'ютери.

Централізований доступ до даних.
Оскільки дані зберігаються тільки на сервері, адміністрування доступу до даних набагато простіше, ніж в будь-яких інших архітектурних стилях.

Простота обслуговування.
Ролі та відповідальність обчислювальної системи розподілені між кількома серверами, спілкуються один з одним по мережі. Завдяки цьому клієнт гарантовано залишається необізнаним і не схильним до впливу подій, що відбуваються з сервером (ремонт, оновлення або переміщення).

Архітектура клієнт / сервер

Можливості (варіанти) застосування архітектури клієнт / сервер:
- якщо створене застосування повинно розміщуватись на сервері і не повинно підтримувати безліч клієнтів;
- якщо створюються Веб-застосування, доступ до яких здійснюється через Веб-браузер;
- якщо реалізуються бізнес-процеси, які будуть використовуватися в рамках організації; або
- якщо створюються сервіси для використання іншими додатками.

Архітектура клієнт / сервер

Архітектурний стиль клієнт / сервер, як і багато стилів мережевих застосувань, також підходить, якщо необхідно:
- централізувати сховище даних,
- функції резервного копіювання та управління, або
- якщо застосування, що розробляється має підтримувати різні типи клієнтів і різні пристрої.

Архітектура клієнт / сервер

Традиційна 2-рівнева архітектура клієнт / сервер має основний недолік:
- тенденція тісного зв'язування даних і бізнес-логіки застосування на сервері, що може мати негативний вплив на розширюваність і масштабованість системи, і залежність від центрального сервера, що негативно позначається на надійності системи.

Для вирішення цих проблем архітектурний стиль клієнт / сервер був розвинений в більш універсальний 3-рівневий (або N-рівневий) (вузловий), в якому усунуті деякі недоліки, властиві 2-рівневої архітектурі клієнт / сервер, і забезпечуються додаткові переваги.

Компонентная архитектура

Основна увага в цьому випадку приділяється розкладанню дизайну на окремі функціональні або логічні компоненти, що надають чітко визначені інтерфейси, що містять методи, події і властивості.

В даному випадку забезпечується більш високий рівень абстракції, ніж при об'єктно-орієнтованій розробці.

Компонентная архитектура

Основний принцип компонентного стилю - застосування компонентів, що володіють такими якостями:

Придатність для повторного використання.
Як правило, компоненти проектуються із забезпеченням можливості їх повторного використання в різних сценаріях різних додатків. Однак деякі компоненти створюються спеціально для конкретного завдання.

Заміщуваність.
Компоненти можуть без проблем замінюватися іншими подібними компонентами. Незалежність від контексту.

Компоненти проектуються для роботи в різних середовищах і контекстах.
Спеціальні відомості, такі як дані про стан, повинні не включатися або вилучатися компонентом, а передаватися в нього.

Можливість розширення.
Компонент може розширювати існуючі компоненти для забезпечення нової поведінки.

Компонентная архитектура

Інкапсуляція.
Компоненти надають інтерфейси, що дозволяє стороні, що викликає використовувати їх функціональність, не розкриваючи при цьому деталі внутрішніх процесів або внутрішні змінні або стан.

Незалежність.
Компоненти проектуються з мінімальними залежностями від інших компонентів. Таким чином, компоненти можуть бути розгорнуті в будь-якому зручному середовищі без впливу на інші компоненти або системи.

Компонентная архитектура

Зазвичай в застосуваннях використовуються:
- компоненти інтерфейсу користувача (їх часто називають елементами управління), такі як таблиці і кнопки, а також допоміжні або службові компоненти, що надають певний набір функцій, які використовуються в інших компонентах.

- ресурсомісткі компоненти, доступ до яких здійснюється нечасто, і активація яких виконується «точно вчасно» (just-in-time, JIT) (зазвичай використовується в сценаріях з віддаленими або розподіленими компонентами);

- компоненти з чергою, виклики методів які можуть виконуватися асинхронно за рахунок застосування черги повідомлень, для зберігання і пересилки.

Компонентная архитектура

Компоненти залежать від механізмів платформи, що забезпечує середу виконання.
Часто ці механізми називають просто компонентної архітектурою.

Прикладами такої архітектури можуть служити:
- Об'єктна модель програмних компонентів (component object model, COM) і
- Об'єктна модель розподілених програмних компонентів (distributed component object model, DCOM) в Windows,
- Загальна архітектура брокера об'єктних запитів (Common Object Request Broker Architecture, CORBA) і
- Серверні компоненти Java (Enterprise JavaBeans, EJB) на інших платформах.

Використовувана компонентна архітектура описує механізми розміщення компонентів і їх інтерфейсів, надсилання повідомлень або команд між

Компонентная архитектура

Однак термін компонент часто використовується в більш загальному сенсі як складова частина, елемент або складова.

Microsoft .NET Framework надає підтримку для побудови застосувань з використанням такого компонентного підходу наприклад, бізнес-компоненти і компоненти даних, які зазвичай є класами, скомпільованими в збірки .NET Framework. Вони виконуються під управлінням середовища виконання .NET Framework. Кожна збірка може включати більше одного подібного компонента.

Компонентная архитектура

Основні переваги компонентного архітектурного стилю:
Простота розгортання.
Існуючі версії компонентів можуть замінюватися новими сумісними версіями, не впливаючи на інші компоненти або систему в цілому.

Менша вартість.
Використання компонентів сторонніх виробників дозволяє розподіляти витрати на розробку та обслуговування.

Простота розробки.
Для забезпечення заданої функціональності компоненти реалізують широко відомі інтерфейси, що дозволяє вести розробку без впливу на інші частини системи.

Компонентная архитектура

Можливість повторного використання.
Застосування компонентів, що багаторазово можуть використовуватись означає можливість розподілу витрат на розробку та обслуговування між декількома додатками або системами.

Спрощення з технічної точки зору.
Компоненти спрощують систему через використання контейнера компонентів і його сервісів. Як приклади сервісів, що надаються контейнером, можна привести активацію компонентів, управління життєвим циклом, організацію черги викликів методів, обробку подій і транзакції.

Компонентная архитектура

Для управління залежностями між компонентами, підтримки слабкого зв'язування і повторного використання можуть використовуватися шаблони проектування, такі як Dependency Injection (Впровадження залежностей) або Service Locator (Локатор сервісу).

Можливості застосування компонентої архітектури:
- якщо вже маєте в своєму розпорядженні відповідні компоненти або можете отримати їх від сторонніх виробників;
- якщо передбачається, що створюване застосування буде переважно виконувати процедурні функції, можливо, з невеликою кількістю даних, що вводяться або взагалі без таких; або
- якщо хочете мати можливість поєднувати компоненти, написані на різних мовах програмування;
- для створення архітектури, що організовує підключення або є архітектурою, що складається з частин, і яка дозволяє без проблем замінювати і оновлювати окремі компоненти.

Проектування на основі предметної галузі

Проектування на основі предметної галузі (Domain Driven Design, DDD) - це об'єктно-орієнтований підхід до проектування, який базується на предметній галузі, її елементах, поведінці і відносинах між ними.
Метою є створення програмної системи, яка є реалізацією предметної галузі і лежить в основі, шляхом визначення моделі предметної галузі, вираженої мовою фахівців в цій галузі.
Модель предметної галузі може розглядатися як каркас, на підставі якого будуть реалізовуватися рішення.

Для застосування DDD необхідно чітко розуміти предметну галузь, яку передбачається моделювати, або мати можливості для оволодіння такими знаннями.
При створенні моделі предметної галузі група розробки нерідко працює в співпраці з фахівцями в даній галузі.
Архітектори, розробники і фахівці в даній галузі мають різну підготовкою і в багатьох ситуаціях використовуватимуть різні мови для опису своїх цілей, бажань і вимог. Однак в рамках DDD вся група домовляється використовувати тільки одну мову, орієнтовану на предметну галузь і виключає всі технічні жаргонізми.

Проектування на основі предметної галузі

Досить часто в групах виникають проблеми з обміном інформацією як через нерозуміння мови предметної галузі, так і через невизначеність мови самої по собі.

Процес DDD має на меті не тільки реалізацію використовуваної мови, але також поліпшення та уточнення мови предметної галузі. Це, в свою чергу, позитивно відбивається на створюваному ПЗ, оскільки модель є прямою проекцією мови предметної галузі.
Щоб зробити модель суворою і корисною мовною конструкцією, як правило, доводиться інтенсивно використовувати ізоляцію і інкапсуляцію в рамках моделі предметної галузі. Це може зумовити відносну дорожнечу системи, заснованої на DDD.
Незважаючи на те, що DDD забезпечує масу переваг з технічної точки зору, таких як зручність в обслуговуванні, ця схема повинна застосовуватися лише для складних предметних галузей, для яких процеси моделювання і лінгвістичного аналізу забезпечують безумовні переваги при обміні складною для розуміння інформацією і формулюванні загального бачення предметної галузі.

Проектування на основі предметної галузі

Основними перевагами стилю DDD є:
- Обмін інформацією.
Всі учасники групи розробки можуть використовувати модель предметної галузі і описувані нею сутності для передачі відомостей і вимог предметної галузі за допомогою спільної мови предметної галузі, не вдаючись до технічного жаргону.

- Можливість розширення.
Модель предметної галузі часто є модульною і гнучкою, що спрощує оновлення і розширення при зміні умов і вимог.

- Зручність тестування.
Об'єкти моделі предметної галузі характеризуються слабким зчепленням і високою зв'язністю, що полегшує їх тестування.

Проектування на основі предметної галузі

Можливості застосування DDD при роботі зі складною предметною областю:
- якщо хочете поліпшити процеси обміну інформацією та її розуміння групою розробки, або
- якщо необхідно представити проект програми спільною мовою, зрозумілою всім зацікавленим сторонам.
- для сценаріїв обробки великих обсягів складних даних підприємства, з якими складно впоратися, використовуючи інші методики.

Багатошарова архітектура

Багатошарова архітектура забезпечує угруповання пов'язаної функціональності застосування в різних шарах, що вибудовується вертикально, поверх один одного.

Функціональність кожного шару об'єднана спільною роллю або відповідальністю.

Шари слабо пов'язані, і між ними здійснюється явний обмін даними.

Правильне поділ застосування на шари допомагає підтримувати суворий поділ функціональності, що в свою чергу, забезпечує гнучкість, а також зручність і простоту обслуговування.

Багатошарова архітектура описана як перевернута піраміда повторного використання, в якій кожен шар агрегує відповідальності і абстракції рівня, розташованого безпосередньо під ним.

При строгому поділі на шари компоненти одного шару можуть взаємодіяти тільки з компонентами того ж шару або компонентами шару, розташованого прямо під даним шаром.

Більш вільний поділ на шари дозволяє компонентам шару взаємодіяти з компонентами того ж шару і всіх нижчих шарів.

Багатошарова архітектура

Шари застосування можуть розміщуватися фізично на одному комп'ютері (на одному рівні) або бути розподілені по різних комп'ютерах (n-рівнів), і зв'язок між компонентами різних рівнів здійснюється через строго певні інтерфейси.
Наприклад, типове Веб-застосування складається з шару представлення (функціональність, пов'язана з UI), бізнес-шару (обробка бізнес-правил) і шару даних (функціональність, пов'язана з доступом до даних, часто практично повністю реалізована за допомогою високорівневих інфраструктур доступу до даних).

Терміни шар (layer) і рівень (tier) часто змішують. Тут шар позначає логічний поділ функціональності, а рівень фізичне розгортання на різних системах.

Багатошарова архітектура

Загальні принципи проектування з використанням багатошарової архітектури:
- Абстракція.
Багатошарова архітектура, що представляє систему як єдине ціле, забезпечуючи при цьому досить деталей для розуміння ролей і відповідальностей окремих шарів і відносин між ними.

- Інкапсуляція.
Під час проектування немає необхідності робити будь-які припущення про типи даних, методи і властивості або реалізації, оскільки всі ці деталі приховані в рамках шару.

- Чітко визначені функціональні шари.
Поділ функціональності між шарами дуже чітка. Верхні шари, такі як шар представлення, посилають команди нижнім шарам, таким як бізнес-шар і шар даних, і можуть реагувати на події, що виникають в цих шарах, забезпечуючи можливість передачі даних між шарами вгору і вниз.

Багатошарова архітектура

-Високе зв'язність (зчеплення). Чітко визначені межі відповідальності для кожного шару і гарантоване включення в шар тільки функціональності, безпосередньо пов'язаної з його завданнями, допоможе забезпечити максимальну зв'язність в рамках шару.

- Можливість повторного використання. Відсутність залежностей між нижніми і верхніми шарами забезпечує потенційну можливість їх повторного використання в інших сценаріях.

- Слабке зв'язування (залежність). Для забезпечення слабкого зв'язування між шарами зв'язок між ними грунтується на абстракції і події.

Багатошарова архітектура

Прикладами багатошарових застосувань:
- бізнес-застосування (line-of-business, LOB), такі як системи бухгалтерського обліку та управління замовниками;
- Веб-застосування та веб-сайти підприємств;
- настільні або смарт-клієнти підприємств з централізованими серверами застосувань для розміщення бізнес-логіки.

Багатошарова архітектура

Основні принципи шаблонів Separated Presentation:
- Поділ функціональності.
Шаблони Separated Presentation поділяють функціональність UI на ролі; наприклад, в шаблоні MVC є три ролі: Модель (Model), Представлення (View) і Контролер (Controller). Модель представляє дані (можливо, модель предметної області, яка включає бізнес-правила); Подання відповідає за зовнішній вигляд UI; Контролер обробляє запити, працює з моделлю і виконує інші операції.

- Повідомлення на основі подій.
Шаблон Observer (Спостерігач) зазвичай використовується для повідомлення Представлення про зміни даних, керованих Моделлю.

- Делегуюча обробка подій.
Контролер обробляє події, що формуються елементами управління UI в Представленні.

Багатошарова архітектура

Основними перевагами багатошарового архітектурного стилю і застосування шаблону Separated Presentation є:
- Абстракція.
Рівні забезпечують можливість внесення змін на абстрактному рівні. Рівень абстракції кожного шару може бути збільшений або зменшений.

- Ізоляція.
Оновлення технологій можуть бути ізольовані в окремих шарах, що допоможе скоротити ризик і мінімізувати вплив на всю систему.

- Керованість.
Поділ основних функцій допомагає ідентифікувати залежності і організувати код в секції, що підвищує керованість.

Багатошарова архітектура

-Продуктивність.
Розподіл шарів по декільком фізичним рівнями може поліпшити масштабованість, відмовостійкість і продуктивність.

- Можливість повторного використання.
Ролі підвищують можливість повторного використання. Наприклад, в MVC Контролер часто може використовуватися повторно з іншими сумісними Уявленнями для забезпечення характерного для ролі або налаштованого для користувача представлення одних і тих же даних і функціональності.

- Тестування.
Поліпшення тестованості є результатом наявності строго визначених інтерфейсів шарів, а також можливості перемикання між різними реалізаціями інтерфейсів шарів. Шаблони Separated Presentation дозволяють створювати під час тестування фіктивні об'єкти, що імітують поведінку реальних об'єктів, таких як Модель, Контролер або Подання.

Багатошарова архітектура

Можливості застосування багатошарової архітектури:
- якщо у вашому розпорядженні є вже готові рівні, які підходять до повторного використання в інших застосуваннях;
- якщо є застосування, що надають відповідні бізнес-процеси через інтерфейси сервісів; або
- якщо створюється складне застосування і попереднє проектування вимагає поділу, щоб групи могли зосередитися на різних ділянках функціональності;
- якщо програма має підтримувати різні типи клієнтів і різні пристрої, або якщо потрібно реалізувати складні бізнес-правила і процеси, які мають налаштовуватися.

Архітектура, основою якої є шина повідомлень

Заснована на шині повідомлень архітектура описує принцип використання програмної системи, яка може приймати і відправляти повідомлення по одному або більше каналах зв'язку, забезпечуючи, таким чином, з додатками можливість взаємодії без необхідності знання конкретних деталей один про одного.

Це стиль проектування, в якому взаємодії між засотуваннями здійснюються шляхом передачі (зазвичай асинхронної) повідомлень через загальну шину.

У типових реалізаціях архітектури, заснованої на шині повідомлень, використовується або маршрутизатор повідомлень, або шаблон Publish / Subscribe (Публікація / Підписка) і система обміну повідомленнями, така як Message Queuing (Черга повідомлень).

Архітектура, основою якої є шина повідомлень

Шина повідомлень забезпечує можливість обробляти:
- Засновану на повідомленнях взаємодію.
Вся взаємодія між застосуваннями ґрунтується на повідомленнях, які використовують відомі схеми.
- Складну логіку обробки.
Складні операції можуть виконуватися як частина багатокрокового процесу шляхом поєднання ряду менших операцій, кожна з яких підтримує певні завдання.
- Зміну логіки обробки.
Взаємодія з шиною реалізується за загальними схемами і з використанням звичайних команд, що забезпечує можливість вставки або видалення додатків на шині для зміни логіки, що використовується для обробки повідомлень.
- Інтеграцію з різними інфраструктурами.
Використання моделі зв'язку за допомогою повідомлень, заснованої на загальних стандартах, дозволяє взаємодіяти із застосуваннями, розробленими для різних інфраструктур, таких як Microsoft .NET і Java.

Шини повідомлень забезпечують підключення архітектури, яка дозволяє вводити застосування в процес або покращувати масштабованість, підключаючи до шини кілька примірників одного і того ж застосування.

Архітектура, основою якої є шина повідомлень

До різновидів шини повідомлень відносяться:
- Сервісна шина підприємства (Enterprise Service Bus, ESB).
ESB грунтується на шині повідомлень і використовує сервіси для обміну даними між шиною і компонентами, підключеними до шини. Зазвичай ESB забезпечує сервіси для перетворення одного формату в інший, забезпечуючи можливість зв'язку між клієнтами, які використовують несумісні формати повідомлень.

- Шина Інтернет-сервісів (Internet Service Bus, ISB).
Подібна сервісної шині підприємства, але застосування розміщуються не в мережі підприємства, а в хмарі. Основна ідея ISB - використання Уніфікованих ідентифікаторів ресурсів (Uniform Resource Identifiers, URIs) і політик, керуючих логікою маршрутизації через застосування і сервіси в хмарі.

Архітектура, основою якої є шина повідомлень

Основні переваги архітектури, заснованої на шині повідомлень:
- Можливість розширення.
Можливість додавати або видаляти програми з шини без впливу на існуючі програми.
- Невисока складність.
Додатки спрощуються, тому що кожному з них необхідно знати лише, як обмінюватися даними з шиною.
- Гнучкість.
Приведення набору додатків, що становлять складний процес, або схем зв'язку між додатками у відповідність до мінливих бізнес-вимог або вимог користувача просто шляхом внесення змін до конфігурації або параметрів, керуючих маршрутизацією.

Архітектура, основою якої є шина повідомлень

- Слабке зв'язування (залежність).
Крім інтерфейсу для зв'язку з шиною повідомлень, немає ніяких інших залежностей з самим застосуванням, що забезпечує можливість зміни, поновлення і заміни його іншим застосуванням, що надає такий же інтерфейс.
- Масштабованість.
Можливість підключення до шини безлічі екземплярів однієї програми для забезпечення одночасної обробки безлічі запитів.
- Простота застосування.
Незважаючи на те, що реалізація шини повідомлень ускладнює інфраструктуру, кожному застосуванню доводиться підтримувати лише одне підключення до шини повідомлень, а не безліч підключень до інших застосувань.

Архітектура, основою якої є шина повідомлень

Можливості застосування:
- якщо маєте існуючі програми, які виконують завдання шляхом взаємодії один з одним, або
- якщо хочете об'єднати безліч кроків в одну операцію.
- якщо необхідна взаємодія із зовнішніми застосуваннями або застосуваннями, розміщеними в різних середовищах.

N-рівнева / 3-рівнева архітектура (вузлова)

N-рівнева і 3-рівнева архітектура є стилями розгортання, що описують розподіл функціональності на сегменти, багато в чому аналогічно багатошарової архітектурі, але в даному випадку ці сегменти можуть фізично розміщуватися на різних комп'ютерах, їх називають рівнями.

Дані архітектурні стилі були створені на базі компонентно-орієнтованого підходу і, як правило, для зв'язку використовують методи платформи, а не повідомлення.

Характеристиками N-рівневої архітектури додатку є функціональна декомпозиція застосування, сервісні компоненти і їх розподілене розгортання, що забезпечує підвищену масштабованість, доступність, керованість і ефективність використання ресурсів.

Кожен рівень абсолютно незалежний від усіх інших, крім тих, з якими він безпосередньо контактує. N-ному рівню потрібно лише знати, як обробляти запит від n + 1 рівня, як передавати цей запит на n-1 рівень (якщо такий є), і як обробляти результати запиту. Для забезпечення кращої масштабованості зв'язок між рівнями зазвичай асинхронний.

N-рівнева / 3-рівнева архітектура (вузлова)

N-рівнева архітектура зазвичай має не менше трьох окремих логічних частин, кожна з яких фізично розміщується на різних серверах.
Кожна частина відповідає за певну функціональність.
При використанні багатошарового підходу шар розгортається на рівні, якщо надається цим шаром функціональність використовується більш ніж одним сервісом або застосуванням рівня.
Прикладом N-рівневого / 3-рівневого архітектурного стилю може служити типове фінансове Веб-застосування з високими вимогами до безпеки.
Бізнес-шар в цьому випадку повинен бути розгорнутий за фаєрволом, через що доводиться розгортати шар представлення на окремому сервері в прикордонній мережі.

Інший приклад - типовий насичений клієнт, в якому шар представлення розгорнуто на клієнтських комп'ютерах, а бізнес-шар і шар доступу до даних розгорнуті на території одного чи більше серверних рівнях.

N-рівнева / 3-рівнева архітектура (вузлова)

Основними перевагами N-рівневого / 3-рівневого архітектурного стилю є:
- Зручність підтримки.
Рівні не залежать один від одного, що дозволяє виконувати оновлення або зміни, не впливаючи на додаток в цілому.
- Масштабованість.
Рівні організовуються на підставі розгортання шарів, тому масштабувати додаток досить просто.
- Гнучкість.
Управління та масштабування кожного рівня може виконуватися незалежно, що забезпечує підвищення гнучкості.
- Доступність.
Застосування можуть використовувати модульну архітектуру, яка дозволяє використовувати в системі легко масштабовані компоненти, що підвищує доступність.

N-рівнева / 3-рівнева архітектура (вузлова)

Можливості застосування N-рівневої або 3-рівневої архітектури:
- якщо вимоги з обробки рівнів застосувань відрізняються настільки сильно, що може виникнути перекіс у розподілі ресурсів, або істотно різняться вимоги з безпеки рівнів.
Наприклад, конфіденційні дані не повинні зберігатися на рівні представлення, але можуть розміщуватися на бізнес-рівні або рівні даних.

- якщо потрібно забезпечити можливість спільного використання бізнес-логіки різними застосуваннями і є достатня кількість обладнання для виділення необхідного числа серверів для кожного рівня.

N-рівнева / 3-рівнева архітектура (вузлова)

Використовуйте тільки три рівня, якщо створюєте додаток для внутрішньої мережі організації, де всі сервери будуть розташовуватися в закритій мережі; або

Інтернет-застосування, вимоги з безпеки якого не забороняють розгортання бізнес-логіки на Веб-сервері або сервері застосувань.

Розгляньте можливість застосування більш трьох рівнів, якщо відповідно до вимог з безпеки бізнес-логіка не може бути розгорнута в прикордонній мережі, або якщо застосування інтенсивно використовує ресурси, і для розвантаження сервера необхідно перенести цю функціональність на інший сервер.

Об'єктно-орієнтована архітектура

Об'єктно-орієнтована архітектура - це парадигма проектування, заснована на поділі відповідальностей на самостійні придатні для повторного використання об'єкти, кожен з яких містить дані і поведінку, що відносяться до цього об'єкта.

При об'єктно-орієнтованому проектуванні система розглядається не як набір підпрограм і процедурних команд, а як набори взаємодіючих об'єктів.

Об'єкти відокремлені, незалежні і слабо пов'язані; обмін даними між ними відбувається через інтерфейси шляхом виклику методів і властивостей інших об'єктів і відправки / прийому повідомлень.

Об'єктно-орієнтована архітектура

Основними принципами об'єктно-орієнтованого архітектурного стилю є:
- Абстракція.
Дозволяє перетворити складну операцію в узагальнення, що зберігає основні характеристики операції. Наприклад, абстрактний інтерфейс може бути широко відомим описом, що підтримує операції доступу до даних через використання простих методів, таких як Get (Отримати) і Update (Оновити). Інша форма абстракції - метадані, які використовуються для забезпечення зіставлення двох форматів структурованих даних.
- Композиція.
Об'єкти можуть бути утворені іншими об'єктами і за бажанням можуть приховувати ці внутрішні об'єкти від інших класів або надавати їх як прості інтерфейси.
- Спадкування.
Об'єкти можуть успадковуватися від інших об'єктів і використовувати функціональність базового об'єкта або перевизначати її для реалізації нової поведінки. Більш того, спадкування спрощує обслуговування і оновлення, оскільки зміни, що вносяться до базового об'єкту, автоматично поширюються на всі успадковані від нього об'єкти.

Об'єктно-орієнтована архітектура

- Інкапсуляція.
Об'єкти надають функціональність тільки через методи, властивості і події і приховують внутрішні деталі, такі як стан і змінні, від інших об'єктів. Це спрощує оновлення або заміну об'єктів і дозволяє виконувати ці операції без впливу на інші об'єкти і код, потрібно лише забезпечити сумісними інтерфейсами.
- Поліморфізм. Дозволяє перевизначати поведінку базового типу, що підтримує операції в застосуванні, шляхом реалізації нових типів, які є взаємозамінними для існуючого об'єкта.
- Відокремлення. Об'єкти можуть бути відокремлені від споживача шляхом визначення абстрактного інтерфейсу, реалізованого об'єктом і зрозумілого споживачеві. Це дозволяє забезпечувати альтернативні реалізації, не впливаючи на споживачів інтерфейсу.

Об'єктно-орієнтована архітектура

Основні переваги ОО архітектури:
- Зрозумілість.
Забезпечується більш близька відповідність застосувань реальним об'єктам, що робить його більш зрозумілим.
- Можливість повторного використання.
Забезпечується можливість повторного використання через поліморфізм і абстракцію.
- Тестування. Забезпечується покращене тестування через інкапсуляцію.
- Можливість розширення.
Інкапсуляція, поліморфізм і абстракція гарантують, що зміни в представлення даних не вплинуть на інтерфейси, що надаються об'єктами, що могло б обмежити можливості зв'язку і взаємодії з іншими об'єктами.
- Висока зв'язність (зчеплення). Розміщуючи в об'єкті тільки функціонально близькі методи і функції і використовуючи для різних наборів функцій різні об'єкти, можна досягти високого рівня зв'язності.

Об'єктно-орієнтована архітектура

Можливості застосування об'єктно-орієнтованої архітектури:
- якщо хочете моделювати додаток на базі реальних об'єктів і дій або вже маєте в своєму розпорядженні відповідні об'єкти і класи, відповідні проектним і експлуатаційним вимогам.
- якщо необхідно інкапсулювати логіку і дані в компоненти, придатні для повторного використання, або якщо є складна бізнес-логіка, яка вимагає абстракції і динамічної поведінки.

Сервіс-орієнтована архітектура

Сервісно-орієнтована архітектура (Service-oriented architecture, SOA) забезпечує можливість надавати функціональність застосування у вигляді набору сервісів і створювати застосування, що використовують програмні сервіси.

Сервіси слабо пов'язані, тому що використовують засновані на стандартах інтерфейси, які можуть бути викликані, опубліковані і виявлені.

Основне завдання сервісів в SOA - надання схеми і взаємодії із застосуванням за допомогою повідомлень через інтерфейси, областю дії яких є застосування, а не компонент або об'єкт. Не слід розглядати SOA-сервіс як компонентний постачальник сервісів.

SOA-архітектура може забезпечити упаковку бізнес-процесів в сервіси, які підтримують можливість взаємодії і використовують для передачі інформації широкий діапазон протоколів і форматів даних. Клієнти та інші сервіси можуть виконувати доступ до локальних сервісів, що виконуються на тому ж рівні, або до віддалених сервісів по мережі.

Сервіс-орієнтована архітектура

Основними принципами архітектурного стилю SOA є:
- Сервіси автономні.
Обслуговування, розробка, розгортання і контроль версій кожного сервісу відбувається незалежно від інших.
- Сервіси можуть бути розподілені.
Сервіси можуть розміщуватися в будь-якому місці мережі, локально або віддалено, якщо мережа підтримує необхідні протоколи зв'язку.
- Сервіси слабо пов'язані.
Кожен сервіс абсолютно не залежить від інших і може бути замінений або оновлений без впливу на застосування, що його використовують, за умови надання сумісного інтерфейсу.

Сервіс-орієнтована архітектура

- Сервіси спільно використовують схему і контракт, але не клас.
При обміні даними сервіси спільно використовують контракти і схеми, але не внутрішні класи.
- Сумісність заснована на політиці.
Політика, в даному випадку, означає опис характеристик, таких як транспорт, протокол і безпеку.

Сервіс-орієнтована архітектура

Типові сервісно-орієнтовані застосування забезпечують спільне використання інформації, виконання багатоетапних процесів (системи резервування і онлайн-магазини), надання спеціальних галузевих даних або сервісів між організаціями та створення складених застосувань, які об'єднують дані з багатьох джерел.

Сервіс-орієнтована архітектура

Основними перевагами SOA-архітектури є:
- Узгодження предметних галузей.
Повторне використання загальних сервісів зі стандартними інтерфейсами розширює технологічні і бізнес-можливості, а також скорочує вартість.
- Абстракція.
Сервіси є автономними, доступ до них здійснюється за формальноим контрактом, що забезпечує слабке зв'язування і абстракцію.
- Можливість виявлення.
Сервіси можуть надавати описи, що дозволяє іншим програмам і сервісам виявляти їх і автоматично визначати інтерфейс.

Сервіс-орієнтована архітектура

- Можливість взаємодії.
Оскільки протоколи і формати даних грунтуються на галузевих стандартах, постачальник і споживач сервісу можуть створюватися і розвиватися на різних платформах.
- Раціоналізація.
Сервіси забезпечують певну функціональність, усуваючи необхідність її дублювання в додатках.

Сервіс-орієнтована архітектура

Можливості застосування SOA-підходу:
- якщо маєте доступ до сервісів, які бажаєте повторно використовувати;
- можете придбати відповідні сервіси у компанії, що надає послуги хостингу;
- хочете створити застосування, що поєднують різні сервіси в один UI; або
- створюєте додатки в моделі ПЗ + сервіси (Software plus Services, S + S), ПЗ як сервіс (Software as a Service, SaaS) або застосування для розміщення в хмарі.
- якщо потрібно підтримувати зв'язок за допомогою обміну повідомленнями між сегментами програми та надавати функціональність незалежно від платформи,
- якщо хочете використовувати інтегровані сервіси, такі як аутентифікація, або хочете надавати сервіси, видимі в каталогах, з можливістю використання клієнтами, які заздалегідь нічого не знають про інтерфейси.

Заключна частина

Перед проектуванням чергового застосування або сервісу вам необхідно визначитися з його архітектурним стилем, який повинен відповідати основним варіантам використання, обмеженням і нефункціональним вимогам.
У світовій практиці накопичилася непогана класифікація.

Під архітектурним стилем будемо розуміти деякий високорівневе уявлення про великі компоненти застосування або системи і взаємозв'язки між ними.

Мета ідентифікації архітектурного стилю - дати уявлення про загальну організацію та принципи роботи застосування, а також реалізацію основних функціональних вимог.

Архітектурні стилі

Дякую за увагу