Integration_types

Интеграция представляет собой единый процесс, объединяющий технологии и системы в единую последовательную цепочку и в результате преобразует форматы данных между системами.

Интеграция информационных систем — это процесс установки связей между информационными системами для получения единого информационного пространства, организации поддержки сквозных бизнес-процессов.

Без эффективного взаимодействия бизнес-процессы и данные не могут быть должным образом интегрированы.

Если компоненты разных слоев архитектуры информационных систем располагаются на различных устройствах, они должны взаимодействовать между собой по сети. Сетевое взаимодействие между разными системами лежит в основе интеграции, независимо от способа реализации этого обмена данными:

- ***файловый обмен***, т.е. импорт в систему А данных из системы В, упакованных в файле нужного формата (XML, JSON, XPDL, CSV и пр.);
- ***непосредственное обращение*** приложения системы А к базе данных системы В;
- ***через API*** – интерфейс прикладного программирования, который позволяет вызвать приложению системы А вызвать некоторые методы системы Б. Сегодня это реализуется с помощью протокола ***SOAP*** (Simple Object Access Protocol), который работает с данными в виде XML-файлов с четко определенной структурой, т.е. схемой (XSD, XML Schema Defeinition), системы удаленного вызова процедур от Google под названием ***gRPC*** (google Remote Procedure Calls), интеграционного API в стиле ***REST***, чаще всего работающий с данными в человекочитаемом формате JSON, или ***GraphQL*** — средой выполнения и языком запросов для взаимодействия клиента и сервера. Интеграцию информационных систем по API можно рассматривать как взаимодействие типа «запрос-ответ» к одной или нескольким конечным точкам (*endpoint*), которые реализуют возможности доступа к данным и манипулирование ими по заранее определенному шаблону запроса и ответа.

- ***через посредника (брокера)***, который выполняет роль временного хранилища данных в виде сообщений от продюсера – системы, которая отправляет данные, потребителю – системе, которая их получает или считывает самостоятельно, в зависимости от средства реализации такой интеграции. Такой способ интеграции ИС чаще всего используется для обмена данными между множеством сервисов, например, в рамках организации единой корпоративной шины данных для взаимодействия в режиме реального времени, платформах интернета вещей и прочих подобных случаев с большим количеством передаваемых событий, источников и потребителей данных.

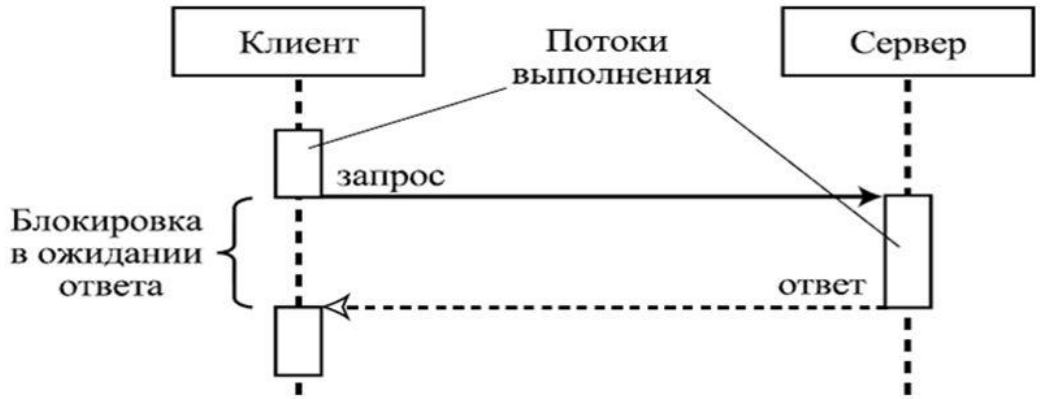
В соответствии с конкретными потребностями связи между приложениями интеграция может быть синхронной, асинхронной или их комбинацией.

При **синхронной связи** приложение-отправитель отправляет запрос вызова API в приложение-получатель и должно дождаться ответа, чтобы продолжить обработку. Этот режим обычно используется в сценариях, где запросы данных необходимо координировать последовательным образом.

Проблема синхронности в том, что удаленный сервис может отвечать не очень быстро даже при прохождении простой операции. Например, на время получения ответа может валять загруженность сетевой инфраструктуры. Блокированные ресурсы могут останавливать работу других экземпляров сервиса по обработке сообщений блокируя весь поток обработки. Таким образом если в момент обращения к внешнему сервису у нас есть незавершенная транзакция в базе данных, которая блокирует новые вызовы, мы можем получить каскадное распространение блокировок.

Синхронное взаимодействие

Синхронное взаимодействие

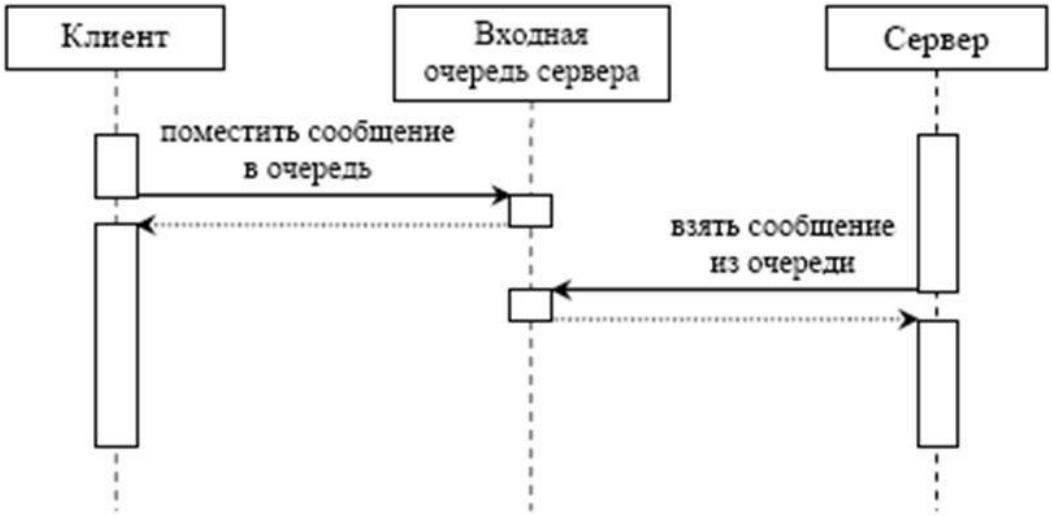


Синхронным (synchronous) называется такое взаимодействие между компонентами, при котором клиент, отослав запрос, **блокируется** и может продолжать работу только **после получения ответа от сервера**.
По этой причине такой вид взаимодействия называют иногда ***блокирующим (blocking)***.

При **асинхронной связи** приложение-отправитель отправляет сообщение приложению-получателю и продолжает локальную обработку до получения ответа. Другими словами, приложение-отправитель не зависит от приложения-получателя для завершения своей обработки. Если несколько приложений интегрированы таким образом, приложение-отправитель может завершить вызов API, даже если другие запросы вызова API еще не были обработаны. Взаимодействие происходит через очереди и таким образом ни один из участников не блокируется в ожидании. Обмен сообщениями происходит через очереди сообщений. Именно очереди обеспечивают асинхронную обработку данных. Они дают возможность поместить сообщение в очередь без обработки, позволяя системе обработать сообщение позднее, когда появится возможность.

Реализация асинхронного взаимодействия

Асинхронное взаимодействие



При разработке решений для интеграции приложений асинхронная связь имеет много преимуществ перед синхронной, особенно когда речь идет о взаимных вызовах между различными сервисами в архитектуре SOA и микросервисов. В режиме синхронной связи, когда приложение должно ждать ответа от нескольких других приложений одновременно, ситуация тайм-аута будет более распространенной. Это означает, что в режиме асинхронной связи нет необходимости ждать завершения вызова API и вызывать частую перегрузку системы, а доступность службы повысится. В режиме асинхронной связи подпроцессы также могут выполняться в любом заранее определенном порядке. Язык JavaScript является типичным представителем этого асинхронного режима связи: в JavaScript разработчики приложений могут легко взаимодействовать с другими приложениями в асинхронном режиме с помощью встроенных методов, таких как обратные вызовы и обещания.

Интеграции могут отличаться по структуре связи

Точка-точка

Взаимодействие систем по принципу точка-точка исторически появилось раньше остальных. Подразумевает этот принцип всего лишь ситуацию, когда каждая система попарно интегрируется с другими. А интегрируется с Б, Б с В и др., т.е. напрямую. Система А знает, как обратиться к системам Б и В, завязана на их интерфейсы, знает адрес сервера приложений.

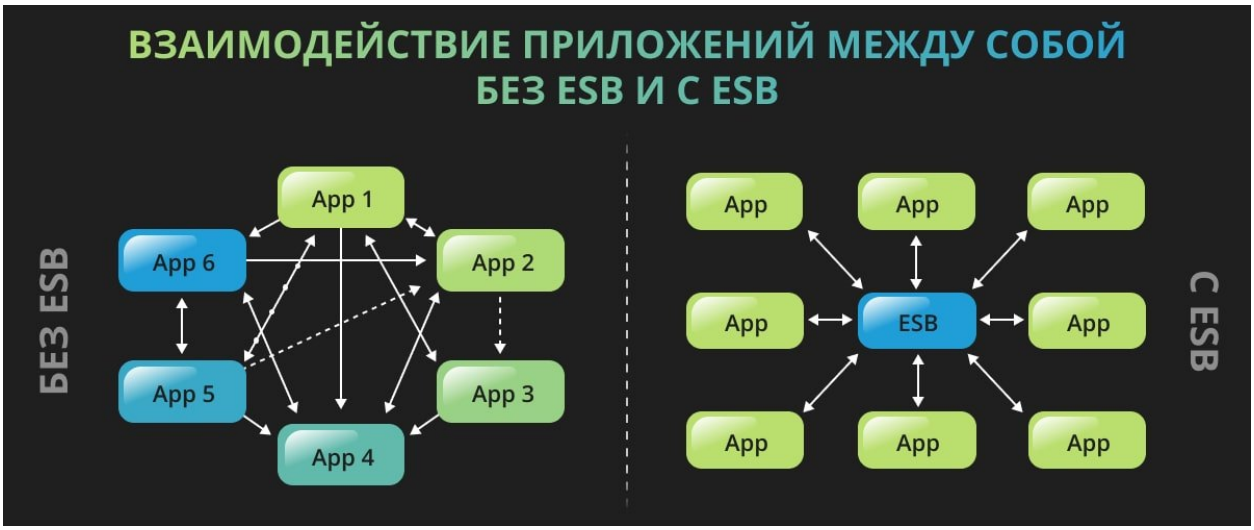
Звезда —

это когда мы добавляем, к примеру, шину (ESB). Какую-то прослойку, которая позволяет всем системам интегрироваться между собой. Это как единая точка. Шина — это тоже ПО. Звезда имеет много плюсов (например, если эта CRM система вам больше не нужна, меняем её на новую — всё не развалится). Но есть и минусы. Например, у вас есть единая точка отказа. Случись что-то с шиной — и не работает вообще всё (весь контур систем, которые интегрируются через эту шину).

Смешанная —

это когда у нас на проекте есть шина, общая точка, где системы объединяются. Но и есть попарно интегрированные системы. Так бывает! И это не всегда плохо.

Что такое шина данных? Шина данных (ESB) выступает прослойкой, связующей различные системы (ПО). Шина дает возможность службам, созданным в различных средах, легко и быстро взаимодействовать между собой. Также она служит для обмена данными с использованием различных протоколов и форматов, позволяя избежать доработок интегрируемых систем. Таким образом, — это промежуточное ПО, которое обеспечивает преобразование сообщений в нужный формат, контроль транзакций, маршрутизацию с учетом смысла, равномерное распределение нагрузки на сервисы и безопасность обмена данными.



Интеграционное тестирование –

это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями. Именно поэтому оно также называется «I & T» (интеграция и тестирование), **«тестирование строк»** и иногда «тестирование потоков».

Ниже приведены различные стратегии, способы их выполнения и их ограничения, а также преимущества.

Подход Большого взрыва

Здесь все компоненты собираются вместе, а затем тестируются.

Преимущества:

- Удобно для небольших систем.

Недостатки:

- Сложно локализовать баги.
- Учитывая огромное количество интерфейсов, некоторые из них при тестировании можно запросто пропустить.

- Недостаток времени для группы тестирования, т.к тестирование интеграции может начаться только после того, как все модули спроектированы.
- Поскольку все модули тестируются одновременно, критические модули высокого риска не изолируются и тестируются в приоритетном порядке. Периферийные модули, которые имеют дело с пользовательскими интерфейсами, также не изолированы и не проверены на приоритет.

Инкрементальный подход

В данном подходе тестирование выполняется путем объединения двух или более логически связанных модулей. Затем добавляются другие связанные модули и проверяются на правильность функционирования. Процесс продолжается до тех пор, пока все модули не будут соединены и успешно протестированы.

Поэтапный подход, в свою очередь, осуществляется двумя разными методами:

- Снизу вверх
- Сверху вниз

Заглушка и драйвер

Инкрементальный подход осуществляется с помощью фиктивных программ, называемых заглушками и драйверами. Заглушки и драйверы не реализуют всю логику программного модуля, а только моделируют обмен данными с вызывающим модулем.

Заглушка: вызывается тестируемым модулем.

Драйвер: вызывает модуль для тестирования.

Интеграция «снизу вверх»

В восходящей стратегии каждый модуль на более низких уровнях тестируется с модулями более высоких уровней, пока не будут протестированы все модули. Требуется помощь драйверов для тестирования

Преимущества:

- Проще локализовать ошибки.
- Не тратится время на ожидание разработки всех модулей, в отличие от подхода Большого взрыва.

Недостатки:

- Критические модули (на верхнем уровне архитектуры программного обеспечения), которые контролируют поток приложения, тестируются последними и могут быть подвержены дефектам.
- Не возможно реализовать ранний прототип

Интеграция «сверху вниз»

При подходе «сверху вниз» тестирование, что логично, выполняется сверху вниз, следуя потоку управления программной системы. Используются заглушки для тестирования.

Преимущества:

- Проще локализовать баги.
- Возможность получить ранний прототип.
- Критические Модули тестируются на приоритет; основные недостатки дизайна могут быть найдены и исправлены в первую очередь.

Недостатки:

- Нужно много пней.
- Модули на более низком уровне тестируются неадекватно

Сэндвич (гибридная интеграция)

Эта стратегия представляет собой комбинацию подходов «сверху вниз» и «снизу вверх». Здесь верхнеуровневые модули тестируются с нижнеуровневыми, а нижнеуровневые модули интегрируются с верхнеуровневыми, соответственно, и тестируются. Эта стратегия использует и заглушки, и драйверы.

Лучшие практики / рекомендации по интеграционному тестированию

- Сначала определите интеграционную тестовую стратегию, которая не будет противоречить вашим принципам разработки, а затем подготовьте тестовые сценарии и, соответственно, протестируйте данные.
- Изучите архитектуру приложения и определите критические модули. Не забудьте проверить их на приоритет.

- Получите проекты интерфейсов от команды разработки и создайте контрольные примеры для проверки всех интерфейсов в деталях. Интерфейс к базе данных / внешнему оборудованию / программному обеспечению должен быть детально протестирован.
- После тестовых случаев именно тестовые данные играют решающую роль.
- Всегда имейте подготовленные данные перед выполнением. Не выбирайте тестовые данные во время выполнения тестовых случаев.