# ТЕХНОЛОГИИ МЕЖСЕРВИСНОЙ СВЯЗИ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ ПРОГРАММНЫХ СИСТЕМ

Синкевич М.Э.
Научный руководитель — проф. Лесная Н.С.
Харьковский национальный университет радиоэлектроники
(61166, Харьков, пр. Науки 14, каф. Программной инженерии,
тел.(057) 702-13-06)
e-mail: maksym.synkevych@nure.ua, тел. 095-579-43-02

The given work considers the interservice communication methods used in the microservice architecture of software systems. Since the connection between microservices should be efficient and reliable, with a large number of small services interacting to perform a single transaction, this can be a challenge, therefore, it is necessary to identify ways to achieve better efficiency. Consider the main approaches to the process organizing of interaction and identify a compromise between asynchronous and synchronous messaging which significantly affect the work of the software.

Микросервисная архитектура структурирует приложение как набор сервисов, обладающих следующими свойствами: легкие в поддержке и тестировании, слабо сцепленные, независимо разворачиваемые [1]. В классических вариантах сервисно-ориентированной архитектуры модули сами по себе могут быть достаточно сложными программными системами, а взаимодействие между ними зачастую возлагается на стандартизированные протоколы (такие, как SOAP, XML-RPC), которые обычно тяжеловесны. В микросервисной же архитектуре приложения выстраиваются из компонентов, которые выполняют относительно простые функции и взаимодействуют с использованием экономичных сетевых коммуникационных протоколов (например, в подходе REST с использованием JSON, Protocol Buffers, Thrift). За счёт повышения гранулярности модулей архитектура нацелена на увеличение связности и на уменьшение степени сцепления, что разрешает проще добавлять и изменять функции в системе. Несмотря на преимущества архитектурного стиля он не лишен недостатков. Среди основных проблем при внедрении можно отметить: если в модулях, выполняющих несколько функций, взаимодействие локально, то микросервисная архитектура накладывает требование атомизации модулей и взаимодействия их в сети; отсутствие стандартизации и необходимость согласования форматов обмена между сервисами; баланс нагрузки и отказоустойчивость. Для них существуют решения, используя которые можно пойти на те или иные уступки. Существует два основных шаблона обмена сообщениями, которые микросервисы могут использовать для связи с другими микросервисами. Синхронное общение — сервис вызывает API, предоставляемый другим сервисом, используя протокол, такой как HTTP или gRPC. Этот вариант является синхронной схемой обмена сообщениями, потому что

вызывающая сторона ожидает ответа от получателя. Асинхронное общение — сервис отправляет сообщение без ожидания ответа, и один или несколько сервисов обрабатывают сообщение асинхронно.

Важно различать асинхронный ввод-вывод и асинхронный протокол. Первый означает, что вызывающий поток не блокируется, второй — что отправитель не дожидается ответа от получателя.

Существуют компромиссы для обоих шаблонов. Запрос и ответ - это хорошо понятная парадигма, поэтому разработка API может показаться более естественной, чем разработка системы обмена сообщениями. Однако асинхронный обмен сообщениями имеет некоторые преимущества, которые могут быть полезны в микросервисной архитектуре: отправителю сообщения не нужно знать о потребителе; используя модель издатель-подписчик, множество потребителей могут подписаться на получение событий; если потребитель отказывает, отправитель по прежнему может отправлять сообщения, они будут получены, когда потребитель восстановится; чувствительность — вышестоящий сервис может ответить быстрее если он не дожидается нижестоящих вервисов; выравнивание нагрузки — очередь сообщений может выступать в качестве буфера для выравнивания рабочей назрузки. Однако существуют также некоторые проблемы с эффективным использованием асинхронного обмена сообщениями: связь с инфраструктурой обмена сообщениями — использование конкретной инфраструктуры для обмена сообщениями может вызвать тесную связь с ней, в результате чего будет сложнее при необходимести переключиться на другую; сквозная задержка может становиться высокой при заполнении очередей; при высокой пропускной способности денежные затраты на инфраструктуру для системы обмена сообщениями могут стать значительными; сложность — обработка асинхронных сообщений является нетривиальной задачей накладывающей свои ограчинения; если сообщения требуют семантики очереди, то она может стать узким местом в системе.

Принято считать, что построение микросервисной архитектуры основано на тех же принципах, что и REST сервис с JSON форматом взаимодействия. Это самый распространенный метод, но, как можно заметить, он не единственный. Комбинируя между собой подходы используя для определенных операций сихронные вызовы, для других — асинхронные можно достичь наиболее эффективного взаимодействия между сервисами.

Список источников:

1. Newman S. Building Microservices / Newman S. — O'Reilly Media, 2015 p 304.