DOI: 10.58168/MIST2025_345-349
УДК 004.9

МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

О.В. Оксюта[1], В.А. Иванников[1], А.В. Толкачев[1]
[1]ФГБОУ ВО «Воронежский государственный лесотехнический университет имени Г.Ф. Морозова»

Аннотация. Работа посвящена описанию межпроцессного взаимодействия в микросервисной архитектуре. Представленная информация может быть использована для построения высоконагруженных, распределенных сервисов. Статья не предлагает исчерпывающий ответ на вопрос, как необходимо выстраивать межпроцессное взаимодействие, а лишь описывает возможные варианты построения архитектуры.

Ключевые слова: агрегат, брокеры сообщений, REST, микросервисная архитектура.

INTERPROCESS COMMUNICATION IN A MICRO-SERVICE ARCHITECTURE

O.V. Oksuyta[1], V.A. Ivannikov[1], A.V. Tolkachev[1]

[1]Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. The paper is devoted to the description of interprocess communication in a micro-service architecture. The information provided can be used to build highly loaded, distributed services. The article does not offer an exhaustive answer to the question of how to build interprocess communication, but only describes possible options for building an architecture.

Keywords: aggregate, message brokers, REST, micro service architecture.

Хорошей практикой является выстраивание всей архитектуры целевой системы вокруг бизнес-объектов. Можно определить набор ключевых бизнес-объектов, называемых агрегатами, и реализовывать бизнес-логику вокруг этих целевых объектов.

Доменные события – события, которые генерируются агрегатом при его создании, обновлении или удалении. Сообщения могут быть типа события и типа

команда/ответ. Сообщение типа событие передается с помощью брокеров сообщений (Kafka, RabbitMQ, JMS…). Сообщений типа команда/ответ обычно передается в виде HTTP-запросов применяя концепцию REST. Между сервисами лучше взаимодействовать через обмен сообщениями, а между приложениями – через REST.

При проектировании сервисов лучше всего определить набор операций, которые будут выполняться сервисом. Таким образом изначально будет ясна зона ответственности сервисов. Если будет очевидно, что у одного сервиса слишком много зон ответственности – это знак, что следовало бы декомпозировать его на несколько сервисов. Таким образом, описание сервиса должно начинаться с его API, так повысится шанс, что сервис будет удовлетворять потребностям клиентов.

Семантическое версионирование

Программное обеспечение постоянно нуждается в доработках и изменении кодовой базы. Это связано с внесением новой функциональности в приложение, выполнением требований информационной безопасности или банально с исправлением дефектов. В связи с этим есть необходимость версионировать изменения кода в целях обратной совместимости. Существует 3 типа изменений: major, minor, patch

MAJOR – при внесении в API несовместимого изменения;

MINOR — при изменении API с сохранением обратной совместимости;

PATCH — при исправлении ошибки с сохранением обратной совместимости.

Major-версию можно сделать частью url. Например, чтобы обратиться к версии 1.x сервиса, клиент выполнит такой запрос:

GET /orders/xyz HTTP/1.1

Accept: application/vnd.example.resource+json; version=l

Этот запрос говорит сервису о том, что клиент ожидает получить ответ версии 1.x.

Если сервис задействует обмен сообщениями, то можете включать номер версии в публикуемые сообщения.

REST

Ресурс – ключевая концепция REST. Он представляет отдельный бизнес-объект, для взаимодействия с которым нужно выполнять HTTP-запросы. Существуют даже отдельные концепции, описывающие работу с REST, например, HATEAOS.

Основной его смысл в том, что представление ресурса, возвращаемое GET-запросом, содержит ссылки для выполнения действий с этим ресурсом.

Например, клиент может отменить заказ с помощью ссылки в представлении, полученном в результате GET-запроса, который этот заказ извлек. В число преимуществ HATEOAS входит то, что вам больше не нужно встраивать URL-адреса в клиентский код.

Обнаружение сервисов

IP-адреса сервисов динамические. Приложение должно автоматически обнаруживать сервисы.

Ключевой компонент обнаружения – реестр сервисов: бд с информацией о том, где находятся экземпляры сервисов. Когда экземпляры сервисов запускаются и останавливаются, механизм обнаружения обновляет реестр.

Когда клиент обращается к сервису, механизм обнаружения получает список его доступных экземпляров, запрашивая реестр, и направляет запрос одному из них. Для вызова сервиса клиент сначала обращается к реестру, чтобы получить список его доступных, а затем шлет запрос одному из них.

Одно из преимуществ обнаружения на уровне приложения(самообнаружение) – возможность разворачивания на разных платформах.

Если часть сервисов развернута в k8s, а часть в Legacy – то обнаружение с использованием Eureka охвактывает обе среды, тогда как k8s охватывает только k8s.

Обычно предпочтительно задействовать механизмы обнаружения, предоставляемые инфраструктурой развертывания (Docker, k8s).

Платформа развертывания выдает каждому сервису DNS-имя, виртуальный IP-Адрес и привязанное к нему доменное имя.

Клиент делает запрос к DNSимени/VIP, а платформа развертывания автоматически направляет его к одному из доступных экземпляров сервиса.

В итоге регистрация и обнаружение сервисов, а также маршрутизация запросов выполняются самой платформой.

Платформа развертывания включает в себя реестр с информацией об IP-адресах развернутых сервисов.

Этот подход имеет 2 шаблона:

* сторонняя регистрация

сервис не прописывает себя в реестре сам, за него это делает компонент регистратор, который обычно является частью платформы развертывания и отвечает за регистрацию;

* обнаружение на стороне сервера

вместо того чтобы обращаться к реестру, клиент делает запрос к DNS-имени, которое направляет его к маршрутизатору запросов, а тот в свою очередь идет в реестр и балансирует нагрузку.

Взаимодействие с помощью асинхронного обмена сообщениями

В асинхронной модели обмена сообщениями без брокера, клиент, ожидающий ответа, не блокируется, а ждет...))))

Сообщение состоит из заголовка и тела. Заголовки – метаданные, которые описывают отправляемую информацию. Там содержится идентификатор сообщения и reply-адрес. Обмен сообщениями по своей природе асинхронен, но клиент может блокироваться, ожидая ответа.

При реализации асинхронных запросов-сообщений в заголовок сообщения кладется имя канала-ответа. Сервер записывает в этот канал свой ответ, содержащий идентификатор соответствия(correlationId) с тем же значением, что и id-запроса.

пример заголовков:

* msgId
* ReplyChannel

Список литературы

1. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. — 3-е издание. Пер. с англ. — М. : Издательский дом «Вильяме», 2003. — 1440 с.

2. Курипта, О. В. Архитектурное решение проектирования сервисов пространственно-временной навигации в образовательных учреждениях / О. В. Курипта, О. В. Минакова, И. В. Поцебнева // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 65-72.

3. Мартин, Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин. — СПб.: Питер, 2013. – 464 с.

4. Минакова, О. В. Повышение эффективности работы в проектах open sourse на основе архитектурного анализа (на примере проекта Сахана) / О. В. Минакова, И. В. Поцебнева, П. Ю. Гусев // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 84-92.

5. Ричардсон, К. Микросервисы. Паттерны микросервисной архитектуры / К. Ричардсон. – Майнинг, 2019. – 554 с.

6. Сидоров, А. А. Проектирование информационных систем / А. А. Сидоров. – СПб.: Издательство, 2019. – 318 с.

7. Сумин, В. И. Разработка сетевой модели целевых установок сложных организационных систем специального назначения / В. И. Сумин, А. С. Кравченко, С. В. Родин // Моделирование систем и процессов. – 2024. – Т. 17, № 3. – С. 79-87.

References

1. Connolly, T. Databases. Design, implementation and maintenance. Theory and practice. / T. Connolly, K. Begg — 3rd edition. Translated from English — M. : Publishing house "Williams", 2003. — 1440 p.

2. Kuripta, O. V. Architectural solution for designing space-time navigation services in educational institutions / O. V. Kuripta, O. V. Minakova, I. V. Potsebneva // Modeling systems and processes. – 2024. – Vol. 17, No. 1. – pp. 65-72.

3. Martin, R. Clean code: creation, analysis and refactoring / R. Martin. — St. Petersburg: Peter, 2013. — 464 p.

4. Minakova, O. V. Improving the efficiency of work in open sourse projects based on architectural analysis (using the example of the Sakhan project) / O. V. Minakova, I. V. Potsebneva, P. Yu. Gusev // Modeling of systems and processes. – 2024. – Vol. 17, No. 1. – pp. 84-92.

5. Richardson, K. Microservices. Patterns of microservice architecture / K. Richardson. – Mining, 2019. — 554 p.

6. Sidorov, A. A. Designing information systems / A. A. Sidorov. — St. Petersburg: Publishing House, 2019. — 318 p.

7. Sumin, V. I. Development of a network model of target installations of complex organizational systems for special purposes / V. I. Sumin, A. S. Kravchenko, S. V. Rodin // Modeling of systems and processes. – 2024. – Vol. 17, No. 3. – pp. 79-87.