УДК 004.75

БИБЛИОТЕКА НАДЁЖНЫХ ТРАНЗАКЦИОННЫХ УДАЛЁННЫХ ВЫЗОВОВ ПРОЦЕДУР НА ОСНОВЕ МОНАДИЧЕСКОГО ВВОДА-ВЫВОДА ДЛЯ JAVA 17

Мещеряков Л. В.[1]
Научный руководитель – Кудасов Н. Д.[1]

[1]Университет Иннополис

Введение. Управление распределёнными транзакциями в микросервисных системах на Java зачастую реализуется через Spring Transaction Management, или Java Transaction API (Java EE, Narayana, Atomikos), или фреймворки распределённых транзакций (Apache Seata). Однако эти системы имеют важный недостаток: контекст транзакции привязывается к потоку выполнения, из-за чего невозможно организовать транзакционные асинхронные операции без сложных обходных решений [1]. Существующие популярные RPC-фреймворки либо никак не реализуют транзакционную семантику (gRPC), либо делают это через интеграцию с перечисленными ранее фреймворками (Apache Dubbo и Apache Seata), сохраняя те же ограничения для асинхронных систем. Более того, в таких системах возникает риск появления дедлоков при циклических RPC-вызовах в рамках одной транзакции, поскольку если пул потоков ограничен, все они могут оказаться заняты ожиданием ответов. Помимо этого, контекст транзакции привязывается через ThreadLocal переменную самими фреймворками, из-за чего он не является строго типизированным. В результате этого разработчик не получает никаких гарантий на уровне типов, что нужный ему контекст доступен, что может привести к ошибкам при выполнении, которые нельзя выявить при компиляции.

Основная часть. В рамках этой работы была создана библиотека **io-rpc** [2], реализующая RPC со встроенной поддержкой распределённых транзакций и расширяющая библиотеку транзакционного монадического ввода-вывода для Java 17 [3]. Базовая библиотека представляет вычисления как ленивые значения с типом **IO**, который имеет параметры типов контекста выполнения, ошибки и результата. Для запуска вычислений нужно явно передавать контекст, в результате чего транзакция не привязывается к потоку. Более того, библиотека **IO** поддерживает стандартные асинхронные абстракции Java. К самим вычислениям в этой библиотеке можно привязать компенсирующие действия в соответствии с паттерном Saga [4] и транзакционные хуки, которые сохраняются при трансформации **IO**, поддерживают двухфазный коммит [5] и выполняются перед теми вычислениями, где они требуются, и в самом конце транзакции.

Библиотека **io-rpc** позволяет описать доступные операции удалённого сервиса через обычный интерфейс, у которого все методы должны возвращать результат типа **IO**. На стороне, которая отправляет запрос, библиотека автоматически создаст реализацию для данного интерфейса, которая будет перенаправлять вызов каждого метода на заранее настроенный сервер. В свою очередь, на стороне, которая принимает запрос, разработчик должен реализовать этот интерфейс и добавить его в конфигурацию RPC. Помимо этого, библиотека поддерживает двухфазный коммит, откаты по достижению таймаута и циклические вызовы между одними и теми же сервисами. Более того, библиотека не привязана к конкретному способу передачи данных между сервисами: эта логика вынесена в абстракцию, у которой есть встроенные реализации для HTTP и AMQP протоколов.

Выводы. В рамках этой работы была создана библиотека, которая устраняет ограничение существующих транзакционных фреймворков для Java в виде привязки контекста транзакции к одному потоку выполнения и обеспечивает типобезопасное управление распределёнными транзакциями с поддержкой двухфазного коммита без внешнего координатора. Функциональность библиотеки подтверждена модульными и интеграционными тестами.

Литература

1. Tuchin A. Mastering Spring: Synchronizing @Transactional and @Async Annotations With Various Propagation Strategies [Электронный ресурс]. – 2024. – URL: https://dzone.com/articles/mastering-spring-synchronizing-transactional-and-a (дата обращения: 26.02.2026)

2. Мещеряков Л. В. io-rpc – GitLab [Электронный ресурс]. – 2025. – URL: https://gitlab.com/worldm/functional/io-rpc (дата обращения: 26.02.2026)

3. Мещеряков Л. В. Type-Safe Transactional Monadic IO in Java 17 // Trends In Functional Programming. – 2026. — (на рассмотрении).

4. Garcia-Molina H., Salem K. Sagas // ACM SIGMOD Rec. – 1987. – V. 16. P. 249–259.

5. Skeen D., Stonebraker M. A Formal Model of Crash Recovery in a Distributed System // IEEE Trans. Softw. Eng. – 1983. – V. SE-9. P. 219–228.