АРХИТЕКТУРА МОДУЛЬНОЙ СИСТЕМЫ УДАЛЁННОГО ВЫЗОВА МЕТОДОВ ДЛЯ СОВРЕМЕННЫХ ПЛАТФОРМ ПРОГРАММИРОВАНИЯ

Киселёв А. И.
Кафедра программного обеспечения информационных технологий,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: and.kis135@gmail.com

В статье представлено исследование разработанной архитектуры объектно-ориентированной системы удалённого вызова процедур (удалённого вызова методов) в случае, когда обе стороны используют общую платформу с поддержкой механизма рефлексии. Проведен анализ достоинств и недостатков предложенного решения.

Введение

Неизменно растущее количество многопользовательских приложений и систем на основе микросервисной архитектуры создаёт высокий спрос на решения в области межпроцессорной (межсетевой) коммуникации. В настоящее время существует широкий выбор технологий и стилей межпроцессорного взаимодействия: REST через HTTP (запрос-ответ), брокеры сообщений (событийный подход), базы данных (общие данные) [1].

Одним из способов организации межсетевого взаимодействия является удалённый вызов процедур (RPC). RPC позволяет изменить состояние удалённого процесса путём вызова процедуры в его виртуальном адресном пространстве. Концепция RPC широко применяется, когда требуется взаимодействие между компонентами, которые могут быть как удалёнными, так и локальными относительно друг друга, так как это минимизирует накладные расходы на вызов при отсутствии межпроцессорной коммуникации. В общем случае возможность локальной работы подразумевает реализацию компонентов на одной и той же платформе программирования. В случае объектно-ориентированной платформы используется термин «удалённый вызов метода» (RMI) экземпляра класса.

В данной работе представлено описание разработанной архитектуре системы вызова удалённых методов для общей платформы с поддержкой механизма рефлексии (такие современные платформы программирования, как .NET, Java и др.).

I. Описание архитектуры

Разработанная архитектура была спроектирована с учётом следующих нефункциональных требований.

1. Наличие строгой схемы;
2. Независимость от канала передачи данных;
3. Независимость от алгоритма сериализации;
4. Минимальные накладные расходы при отсутствии межпроцессорного взаимодействия.

Для удовлетворения четвёртого требования можно ввести соглашение, что вызов метода объекта, который может привести к межпроцессорному взаимодействию, должен осуществляться через интерфейсную переменную. Если объект представлен локально, его метод будет вызван соответствующим для платформы способом (например, через таблицу виртуальных методов). В случае, когда требуется вызвать метод удалённого объекта, интерфейсная переменная должна содержать прокси-объект, производящий межпроцессорный вызов. Таким образом, выбор между использованием локального экземпляра или объекта-прокси может осуществляться, например, прямо в корне композиции внедрения зависимостей [2].

Отдельно стоит отметить возможность создания прокси-объектов для удалённого вызова через соответствующий механизм платформы, например, DispatchProxy для .NET или InvocationHandler в Java. Широкое распространение подобных механизмов на платформах программирования обусловлено, в том числе, их использованием в аспектно-ориентированном программировании [3].

Наличие строгой схемы позволяет выявлять нарушения контракта межпроцессорного взаимодействия, вызванные несогласованным изменением, добавлением или удалением методов одной из сторон. Использование интерфейсных методов приводит к появлению строгой схемы, где схемой является представление интерфейса в системе типов платформы программирования, что обеспечивает выполнение первого требования.

Кроме того, появляется возможность валидации совместимости удалённых компонентов, например, путём проверки совпадения результатов хэш-функции для упорядоченных уникальных идентификаторов артефактов платформы, содержащих определения используемых интерфейсов.

Выполнение второго требования можно обеспечить путём введения абстракции пакета. Реализация пакета является внешней по отношению к системе удалённого вызова методов и определя-

ется потребностями канала передачи данных. В данном случае пользователь может определить логику маршрутизации произвольной сложности, что обеспечивает такие возможности как, например, работу поверх существующего канала, мультиплексирование и мультикастинг, а также повышает тестируемость получаемого решения.

Для обеспечения независимости пакета от системы удалённого вызова методов требуется механизм взаимодействия между ними. Учитывая характер использования, можно ввести зависимость от реализации интерфейса, представляющего фабрику для создания функций записи аргументов вызываемого метода в пакет, а также функций чтения аргументов из него [4]. Методы интерфейса могут принимать тип аргумента удалённо вызываемого метода в качестве входного параметра. Создаваемые функции инкапсулируют логику сериализации, что позволяет выполнить третье требование.

Независимость алгоритма сериализации предоставляет возможность гибко выбирать между границами применения и производительностью, например, поддерживая сериализацию только примитивных типов данных или типов с глубокой вложенностью и базовых коллекций платформы. Также интерфейс может содержать метод для создания пакета, что позволяет реализовать такие оптимизации, как, например, использование пула объектов.

Одним из ограничений описанной архитектуры является то, что удалённый вызов метода осуществляется через сам метод интерфейса, что означает необходимость представления всей конфигурации вызова в сигнатуре метода и его аргументах. Однако системы типов современных платформ программирования предоставляют широкий набор инструментов для решения подобных задач. Таким образом, пакет, содержащий информацию об удалённом вызове, может быть сконфигурирован следующими способами:

- Название метода;
- Атрибут (аннотация) метода;
- Атрибут (аннотация) параметра;
- Тип параметра.
- Возвращаемое значение.

Определение поведения удалённого вызова метода на основе его названия возможно, но не рекомендуется, так как поддержка работы со схемой именования отсутствует у большинства сред разработки.

Использование атрибута метода подходит для конфигурирования удалённого вызова, например, путём передачи его в метод создания пакета.

Атрибут параметра, как и его тип, могут предоставлять информацию об особом значении параметра на этапе создания пакета. Вместо того чтобы быть сериализованным вместе с остальными аргументами, значение может интерпретироваться как, например, опция, определяющая гарантии доставки при использовании сетевого протокола, работающего поверх UDP. Такая гибкость достигается за счёт того, что определение пакета и логика работы с ним остаются внешними по отношению к системе удалённых вызовов.

Возвращаемое значение удалённо вызываемого метода может иметь особое значение, требующее специальной поддержки со стороны системы удалённых вызовов. Например, возвращаемое значение типа результата, параметризованного типом данных, может указывать на блокирующий вызов (поток ожидает завершения удалённого вызова и получения результата), тогда как тип void – на неблокирующий вызов. Также в качестве типа результата можно использовать тип с поддержкой асинхронного шаблона выполнения платформы [5].

Множество реализаций RPC скрывают от конечного пользователя факт наличия межпроцессорного вызова. Однако со временем такой подход был признан неудачным, поскольку для межпроцессорных вызовов возможны ошибки, нехарактерные для локальных вызовов [1]. Учитывая описанные способы определения методов интерфейса, разработанной архитектуре скорее характерно сокрытие наличия локального выполнения.

II. Заключение

Разработана архитектура системы удалённого вызова методов для случаев, когда имеется общая платформа с поддержкой метапрограммирования. Достоинствами данного решения являются высокая модульность и строгая схема, обеспечиваемая системой типов платформы программирования. К недостаткам можно отнести ограничения по платформам для реализации, а также необходимость реализации абстракции пакета конечным потребителем решения.

Список литературы

1. Ньюмен, С. Создание микросервисов / С. Ньюмен // Издательство: Питер, 2024. – 624 с.
2. Марк, С. Внедрение зависимостей на платформе .NET / С. Марк // Издательство: Питер, 2021. – 608 с.
3. Мартин, Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин // Издательство: Питер, 2021. – 464 с.
4. Паттерны объектно-ориентированного проектирования / Э. Гамма [и др.]. – СПб.: Питер, 2021. – 448 с.
5. Документация по программированию .NET [Электронный ресурс] / Асинхронный шаблон на основе задач. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap – Дата доступа: 18.10.2024.