ТЕХНОЛОГИИ КОМПЬЮТЕРНЫХ СИСТЕМ И СЕТЕЙ

УДК 004.624

УНИВЕРСАЛЬНЫЙ БИНАРНЫЙ ФОРМАТ ДЛЯ БЫСТРОЙ ПЕРЕДАЧИ ДАННЫХ ПО СЕТИ

Лезин И.А.
Самарский национальный исследовательский университет имени академика С.П. Королева, Самара, РФ
E-mail: lezin.ia@ssau.ru

В условиях современных подходов к разработке приложений все чаще приходится работать с распределенными системами обработки данных, например, в микросервисной архитектуре. Здесь актуальным становится вопрос быстрого обмена данными между частями одного приложения. В статье рассматривается новый формат двоичного кодирования данных не фиксированной заранее структуры с поддержкой кросс-платформенного подхода и возможностью реализации операций сериализации и десериализации на разных языках с использованием универсального способа представления данных. Степень эффективности новых алгоритмов определяется сравнением с результатами работы других широко применяемых на данный момент форматов и фреймворков. Проведенный анализ показывает, что с учетом накладываемых решаемой задачей ограничений но-вый формат представления данных обеспечивает более высокую скорость обмена за счет оптимизации алгоритмов сериализации-десериализации.

Ключевые слова: *сериализация, десериализация, бинарный формат, обмен данными, кроссплатформенный обмен*

Введение

Сериализация и десериализация данных – это два связанных процесса преобразования данных (объекта) для их представления в виде последовательности битов, с возможностью передачи по сетевым каналам или хранения в буферах памяти и файлах, и последующего восстановления исходной структуры без потерь, как в этом же приложении, так и в каком-либо другом. Считывание результирующей последовательности битов в соответствии с форматом сериализации должно приводить к созданию абсолютно точной копии исходного объекта с учетом накладываемых ограничений, поскольку комплексные вычислительные системы не являются гомогенными и могут физически не позволять оперировать идентичными типами данных или объектов в разных частях. Процесс создания двоичной копии и восстановления исходного объекта не является простым и может сопровождаться наложением различных ограничений на способы и области его применения. При разработке распределенных приложений сериализация и десериализация являются неотъемлемой частью процесса программирования при передаче данных по сети, сохранении объектов в базу данных, либо при кэшировании. Например, если мы рассматриваем передачу собранной телеметрии через сеть, то некоторый набор данных сериализуется на стороне источника в виде формата JSON или бинарного потока, затем происходит передача данных по сети и их восстановление в такую же, либо максимально подобную структуру на стороне приемника для дальнейшего использования.

Для сериализации и десериализации данных на разных языках программирования было придумано множество форматов и разработан целый ряд реализующих их фреймворков, поскольку данный аспект необходимо учитывать при работе распределенных систем для реализации обмена информацией между разными компонентами программного обеспечения.

Общие преимущества и недостатки сериализации и десериализации

Технология преобразования форматов данных с помощью сериализации и десериализации обладает рядом преимуществ, которые обуславливают ее использование программистами при работе с данными [1].

Сериализация предоставляет возможность передавать произвольные структуры данных в удобном формате по сети, без чего невозможно обойтись при реализации клиент-серверного взаимодействия и при обмене данными в распределенных системах.

Состояние объектов в сериализованном виде можно сохранять в файлах, базах данных или системах кэширования, если требуется зафиксировать некоторый срез данных, либо текущее состояние программы, чтобы возобновить работу позже.

Даже если программы или сервисы написаны на разных языках программирования, им необходимо взаимодействовать между собой. Для упрощения интеграции разнородных систем можно сериализовать данные в общие форматы. При этом зачастую бинарные форматы, которые используются в фреймворках для сериализации,

являются более компактными, чем текстовые, что ускоряет передачу данных по сети.

Сериализацию можно использовать для обеспечения совместимости между разными версиями программ, если поддерживается версионирование данных через сохранение возможности десериализовать старые данные в предыдущих форматах в новых версиях программного обеспечения. Также для повышения гибкости и создания возможности расширения форматов некоторые фреймворки позволяют работать с метаданными и сохранять информацию о пользовательских типах данных в процессе сериализации.

Но, как и любой другой инструмент, помимо преимуществ, сериализация обладает и некоторыми недостатками.

Для восстановления данных при десериализации может потребоваться дополнительная информация о сложных типах данных, которые в таком случае при сериализации также необходимо сохранять в формируемый пакет, чтобы се-риализированные данные хранили сведения о своей исходной структуре.

Разработка алгоритмов механизмов сериализации и десериализации становится в ряде случаев сложной задачей, особенно при реализации механизма версионирования или обеспечении совместимости форматов между разными платформами. Если при изменении структуры данных между двумя версиями программного обеспечения требуется сохранение обратной совместимости, то используемый формат должен предусматривать возможность управления версионированием данных.

Сериализованные данные также могут быть объектом атак, например, для внедрения вредоносного кода, но эта проблема является общей при передаче данных по сети, а не исключительно проблемой сериализации. Тем не менее, при работе с восстановлением сериализованных данных необходимо обеспечивать должный уровень безопасности.

Обилие различных форматов представления данных в сериализованном виде предполагает у разных фреймворков разную степень экономичности с точки зрения занимаемого объема, и эффективности с точки зрения требуемого времени на сериализацию-десериализацию. Поэтому выбор формата следует выполнять исходя из имеющихся ограничений по скорости обработки или объему сериализованных данных.

Также данные могут содержать иерархически вложенные друг в друга объекты, либо иметь циклические зависимости, поэтому механизмы сериализации и десериализации могут столкнуться с трудностями при обработке подобных конструкций.

Наиболее популярные форматы сериализации

При практической разработке нового бинарного формата и новых алгоритмов преобразования данных все актуальные проблемы сериализации и десериализации рассматривались в рамках разработки микросервисного приложения на фреймворках Spring Boot [2], Flask [3] и платформе .NET [4], версий языков Java 17, Python 3.10 и C# 12.0 соответственно. Основными направлениями использования механизма сериализации являлась передача данных для задач ETL [5], в которых формат данных является произвольным и заранее неизвестен, но при этом сами данные не содержат кольцевых ссылочных зависимостей. Для структурного представления таких данных идеально подходят форматы JSON или XML, но остается вопрос эффективности при обмене данными между микросервисами по сети.

Как показано в исследовании современных текстовых форматов JSON, XML и YAML [6], наиболее популярным форматом в настоящее время является JSON. Он и был выбран для сравнения.

Все рассмотренные фреймворки для сериализации приведены в таблице 1.

Поскольку универсальных форматов для передачи данных по сети не так много, а основным языком разработки приложения является Java, и большая часть обмена происходит между Java-сервисами, то были рассмотрены не только универсальные форматы, но и Java-специфичные [7; 8; 9].

Как видно из проведенного анализа, полностью подходят по всем ограничениям форматы JSON, BSON, Hessian и MessagePack, поскольку они работают с данными произвольной структуры и поддерживаются множеством языков программирования.

Форматы Protobuf, Thrift и Avro требуют предварительной фиксации и описания формата передаваемых данных.

Форматы JDK Serializable, FST, Kryo и One Nio подходят только для решений на Java.

Новый двоичный формат

Для обеспечения вариативности, чтобы не ограничиваться исключительно двумя форматами при выборе, был разработан новый универсальный двоичный формат представления данных,

Таблица 1. Фреймворки для сериализации

Формат	Универсальность платформы	Универсальность структуры данных
JSON	Да	Да
JDK Serializable	Java	Да
FST	Java	Да
Kryo	Java	Да
Protobuf	Да	Жесткая
Thrift	Да	Жесткая
Avro	Да	Жесткая
Hessian	Да	Да
One Nio	Java	Да
MessagePack	Да	Да
BSON	Да	Да

который можно использовать для сериализации, передачи по сети и десериализации.

Основная идея заключается в построении JSON-подобной последовательности байт с поддержкой коллекций и объектов в формате «ключ-значение». Предусмотрена поддержка основных примитивных типов данных фиксированной длины: логические данные, 1-байтовое целое, 2-байтовое целое, 4-байтовое целое, 4-байтовое с плавающей запятой и 8-байтовое с плавающей запятой. А также три типа данных произвольной длины: строка, массив и коллекция «ключ-значение». Хранение имен классов не предусмотрено для обеспечения совместимости между платформами на разных языках. Это одно из ограничений формата, которое приходится принимать для поддержки кросс-платформенности, но в контексте решаемых задач с учетом априорной неопределенности форматов ограничение накладывается не фреймворком, а самим условием задачи.

Рассмотрим пример передачи данных следующей структуры:

[{“Value”:1234}].

После выполнения сериализации с использованием нового формата получается следующая последовательность байт:

[**3**, **1**, **13**, 5, 86, 97, 108, 117,
101, **9**, 4, -46, 0, 0, **2**, **4**].

Первый байт с кодом 3 означает начало коллекции. Затем идет байт с кодом 1, означающий начало объекта в формате «ключ-значение». Далее код 13 означает начало строки, используемой как имя поля текущего объекта. Байт с кодом 5 означает длину текущей строки, а следующие 5 байт содержат коды символов строки. После этого идет байт с кодом 9, это код целого 4 –байтового числа. Следующие 4 байта содержат код этого числа в «swapped» формате. Затем следует байт с кодом 2 – конец текущего объекта. И последний байт с кодом 4 – конец текущей коллекции.

При кодировании используется следующий набор из 15 служебных кодов:

1 – начало объекта;
2 – конец объекта;
3 – начало коллекции;
4 – конец коллекции;
5 – логическая 1;
6 – логический 0;
7 – целое со знаком (1 байт);
8 – целое со знаком (2 байта);
9 – целое со знаком (4 байта);
10 – целое со знаком (8 байт);
11 – число с плавающей запятой (4 байта);
12 – число с плавающей запятой (8 байт);
13 – строка в универсальном формате;
14 – строка Java в формате LATIN1;
15 – строка Java в формате UTF16.

Для записи служебных кодов используются 4 бита, это означает, что в сжатом формате есть возможность записывать два кода в 1 байт. Тогда приведенный выше пример можно представить в виде следующего сериализованного кода:

[**19**, **157**, 5, 86, 97, 108, 117, 101, 4, -46, 0, 0, **66**].

Здесь первый байт содержит коды начала коллекции и начала объекта в младшей и старшей тетрадах соответственно. Второй байт содержит коды начала строки и 4-байтового целого. Подобный подход не вносит неопределенности, поскольку после чтения кода строки из младшей тетрады 2-го байта при десериализации идет чтение длины строки из следующего байта, а затем чтение указанного числа байт для формирования строки. После этого идет анализ старшей тетрады 2-го байта, из которой десериализатор понимает,

что нужно прочитать 4 следующих байта для восстановления целого числа со знаком. Последний байт содержит коды конца объекта и конца коллекции.

В итоге количество байт, используемых под служебные коды, сокращается вдвое, а общая длина сериализованного объекта уменьшается с 17 до 14 байт.

Сравнение скорости работы

Для сравнения эффективности работы рассмотренных фреймворков с новым предлагаемым подходом были выбраны все фреймворки с возможностью реализации на Java и не требующие предварительной фиксации структуры передаваемых данных на этапе разработки. Алгоритмы сериализации и десериализации для нового фреймворка также были реализованы на языке Java.

Объектом сериализации являлся экземпляр класса Map с 1000000 пар «ключ-значение». Для ключей использовались сгенерированные строки длиной от 15 до 20 символов, а значения генерировались случайным образом в виде примитивов boolean, int и double.

Результаты замеров приведены в таблице 2.

Сравнение результатов проводилось с разбиением на две категории: универсальные форматы и Java-форматы.

В число универсальных форматов вошли JSON, Hessian и новый формат с универсальной кодировкой строк. Как видно из таблицы, фреймворк Jackson проигрывает конкурентам с огромным отрывом, что легко объясняется его человеко-читаемым способом представления данных в сериализованном виде. При этом между фреймворками Hessian и предложенным форматом кодировки развернулась борьба. Hessian чуть более эффективен, если сравнивать объемы данных в сериализованном виде, но в 2 раза уступает по скорости при сериализации и на 20% с восстановлением данных из двоичного формата.

Если же рассматривать Java-фреймворки, то здесь основное соперничество развернулось между новым способом сериализации и фреймворком Kryo. Фреймворк Kryo чуть более эффективен по объему занимаемой памяти, обладает такой же скоростью сериализации, но в 1,5 раза медленнее оказывается при восстановлении, если сравнивать его новым фреймворком в режиме работы с сериализацией строк в Java-формате. Новый фреймворк в универсальном формате представления строк обладает примерно схожими характеристиками с фреймворком Kryo, но при этом является универсальным.

Столь существенная разница по части скорости работы нового фреймворка в режимах универсальной кодировки и Java-кодировки объясняется тем, что Java-формат предусматривает работу с данными напрямую в памяти через Unsafe и методы рефлексии. Разумеется, это приводит к ускорению работы алгоритмов, поскольку минует этапы преобразования форматов представления строковых данных.

Заключение

Как показало сравнение с наиболее популярными на данный момент фреймворками, новый бинарный формат представления данных и алгоритмы его сериализации и десериализации в условиях наложенных задачей ограничений показывают более высокую эффективность обмена, как при сравнении с универсальными фреймворками, так и ориентированными на платформу Java.

Стоит заметить, что общий объем исходного кода составляет чуть более 300 строк, при этом включать в сборку внешние библиотеки не требуется, что позволяет в дальнейшем как расширять, так и оптимизировать существующие алгоритмы,

Таблица 2. Сравнение скорости работы

Фреймворк	Сериализация, мс	Десериализация, мс	Объем, байт
JSON Jackson	173	545	21161004
JDK Serialization	427	259	209935090
FST	159	100	15960511
Kryo	80	55	13690063
One Nio	791	71	16293146
Hessian	109	88	13831124
MessagePack	180	365	14021928
BSON	131	131	14187753
Новый (универсальные строки)	55	71	14093287 (15093288)
Новый (строки Java)	56	55	

подстраивая их под текущие условия очередной задачи. Например, при передаче телеметрии количество используемых форматов данных сокращается вдвое.

Универсальность формата была подтверждена через организацию обмена сообщениями по соединениям на основе сокетов [10] с сервисами, написанными на языках Python и C#, для интеграции модулей в распределенной системе.

Стоит отметить, что различие подходов к представлению данных на разных языках программирования привносит дополнительную сложность в разработку универсального формата двоичного представления сериализованных данных, который подходил бы для обработки на всех используемых платформах.

Литература

1. Кряхтунов Г.М., Боронников А.С. Подходы к созданию проприетарного формата представления данных // International Journal of Open Information Technologies. 2024. Vol. 12, no. 5. P. 141–150.
2. Turnquist G. Learning Spring Boot 3.0. Third Edition. Birmingham: Packt Publishing Ltd, 2022. 270 p.
3. Dwyer G. Flask by Example. Birmingham: Packt Publishing Ltd, 2016. 276 p.
4. Price M.J. C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals – Eighth Edition. Birmingham: Packt Publishing Ltd, 2023. 828 p.
5. Арьков В.Ю. Бизнес-аналитика. Извлечение, преобразование и загрузка данных. Ridero, 2020. 128 с.
6. Канаев К.А., Фалеева Е.В., Пономарчук Ю.В. Сравнительный анализ форматов обмена данными, используемых в приложениях с клиент-серверной архитектурой // Фундаментальные исследования. 2015. № 2-25. С. 5569–5572.
7. An introduction and comparison of several common java serialization frameworks. URL: https://www.alibabacloud.com/blog/an-introduction-and-comparison-of-several-common-java-serialization-frameworks_597900 (дата обращения: 15.06.2024).
8. Java-сериализация: максимум скорости без жесткой структуры данных. URL: https://habr.com/ru/companies/sberbank/articles/488612/ (дата обращения 15.06.2024).
9. Optimizing data serialization: faster alternatives to JSON. URL: https://medium.com/@shipshoper986/optimizing-data-serialization-faster-alternatives-to-json-a3685d21008/ (дата обращения: 05.07.2024).
10. Дубаков А.А. Сетевое программирование: учебное пособие. СПб.: НИУ ИТМО, 2013. 248 с

Получено 20.06.2024

Лезин Илья Александрович, к.т.н., заведующий кафедрой информационных систем и технологий Самарского национального исследовательского университета имени академика С.П. Королева. 443086, Российская Федерация, г. Самара, Московское шоссе, 34. Тел. +7 902 320-92-99. E-mail: lezin.ia@ssau.ru

UNIVERSAL BINARY FORMAT FOR FAST DATA TRANSMISSION OVER THE NETWORK

Lyozin I.A.
Samara National Research University, Samara, Russian Federation
E-mail: lezin.ia@ssau.ru

In the context of modern approaches to application development, it is increasingly necessary to use distributed data processing systems, for example, in a microservice architecture. Here, the issue of fast data exchange between parts of the same application becomes relevant. The article discusses a new format for binary data encoding of the non-fixed, supported by the cross-platform approach and with the ability to implement serialization and deserialization operations in different languages using a universal data representation method. The efficiency degree of the new algorithms is deter-mined by comparing them with the results for another formats and frameworks that are widely used at the moment. The analysis shows that, taking into account the limitations imposed by the problem being solved, the new data presentation format provides a higher exchange rate due to optimization of serialization-deserialization algorithms.

Keywords: *serialization, deserialization, binary format, data transmission, cross-platform exchange*

DOI: 10.18469/ikt.2024.22.1.09

Lyozin Ilya Alexandrovich, Samara National Research University, 34, Moskovskoye shosse, Samara, 443086, Russian Federation; Head of Information Systems and Technologies Department, PhD in Technical Science. Tel. +7 902 320-92-99. E-mail: lezin.ia@ssau.ru

References

1. Kryakhtunov G.M., Boronnikov A.S. Approaches to creating a proprietary data presentation format. *International Journal of Open Information Technologies*, 2024, vol. 12, no. 5, pp. 141–150. (In Russ.)
2. Turnquist G. *Learning Spring Boot 3.0. Third Edition*. Birmingham: Packt Publishing Ltd, 2022, 270 p.
3. Dwyer G. *Flask by Example*. Birmingham: Packt Publishing Ltd, 2016, 276 p.
4. Price M. *C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals – Eighth Edition*. Birmingham: Packt Publishing Ltd, 2023. 828 p.
5. Arkov V.Yu. *Business Analytics. Data Extraction, Transformation and Loading*. Ridero, 2020, 128 p. (In Russ.)
6. Kanaev K.A., Faleeva E.V., Ponomarchuk Yu.V. Comparative analysis of data exchange formats for applications with client-server architecture. *Fundamental'nye issledovaniya*, 2015, no. 2-25, pp. 5569–5572. (In Russ.)
7. An introduction and comparison of several common java serialization frameworks. URL: https://www.alibabacloud.com/blog/an-introduction-and-comparison-of-several-common-java-serialization-frameworks_597900 (accessed: 15.06.2024).
8. Java-serialization: maximum of speed without a fixed data structure. URL: https://habr.com/ru/companies/sberbank/articles/488612/ (accessed: 15.06.2024). (In Russ.)
9. Optimizing data serialization: faster alternatives to JSON. URL: https://medium.com/@shipshoper986/optimizing-data-serialization-faster-alternatives-to-json-a3685d21008/ (accessed: 05.07.2024).
10. Dubakov A.A. *Network Programming:* Textbook. Saint Petersburg: NIU ITMO, 2013, 248 p. (In Russ.)

Received 20.06.2024

НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 004.94

ОБЕСПЕЧЕНИЕ ПЕРСОНИФИЦИРОВАННОГО ПОДХОДА К ПАЦИЕНТУ НА БАЗЕ MHEALTH-ПРИЛОЖЕНИЯ

Диязитдинова А.Р., Загирова А.Р., Нужин В.И.
Поволжский государственный университет телекоммуникаций и информатики, Самара, РФ
E-mail: dijazitdinova@mail.ru

В работе представлено описание мобильного приложения mHealth-класса, с помощью которого пользователь может получить возможность расшифровки результатов общего анализа крови. Данный анализ, с точки зрения статистики, является одним из самых часто назначаемых и информативных, поскольку он позволяет врачу сделать первичные выводы о состоянии здоровья пациента. Общий анализ крови – это не один анализ, а совокупный комплекс показателей, каждый из которых в отдельности может сигнализировать о различных заболеваниях. Современные решения в области mHealth-здравоохранения содействуют получению пациентом своевременного и полноценного лечения. На рынке существует значительное число приложений для осуществления мониторинга здоровья (в том числе и анализа крови), однако они зачастую имеют ограниченную функциональность или предоставляют формальные и общеизвестные, а не персонифицированные сведения о состоянии здоровья. Инструментом обеспечения персонифицированного подхода к пользователю может выступать онтология. Онтология в разрабатываемом мобильном приложении рассматривается не просто как некоторое неформальное описание базы знаний, а как основа базы данных для упрощения процесса наполнения и поддержания онтологии в акту-