Классификация методов сериализации данных

Студент: Варин Д. В., ИУ7-76Б
Научный руководитель: Кузнецов Д. А.

Москва, 2022 г.

Цель и задачи работы

Цель работы: проанализировать существующие методы сериализации данных

Задачи работы:

1. Провести обзор предметной области.
2. Описать термины предметной области.
3. Описать анализируемые форматы сериализации данных.
4. Выявить критерии сравнения.
5. Сравнить форматы.

Предметная область

RPC - это процесс, при котором программа на одной машине вызывает выполнение процедуры на другой.

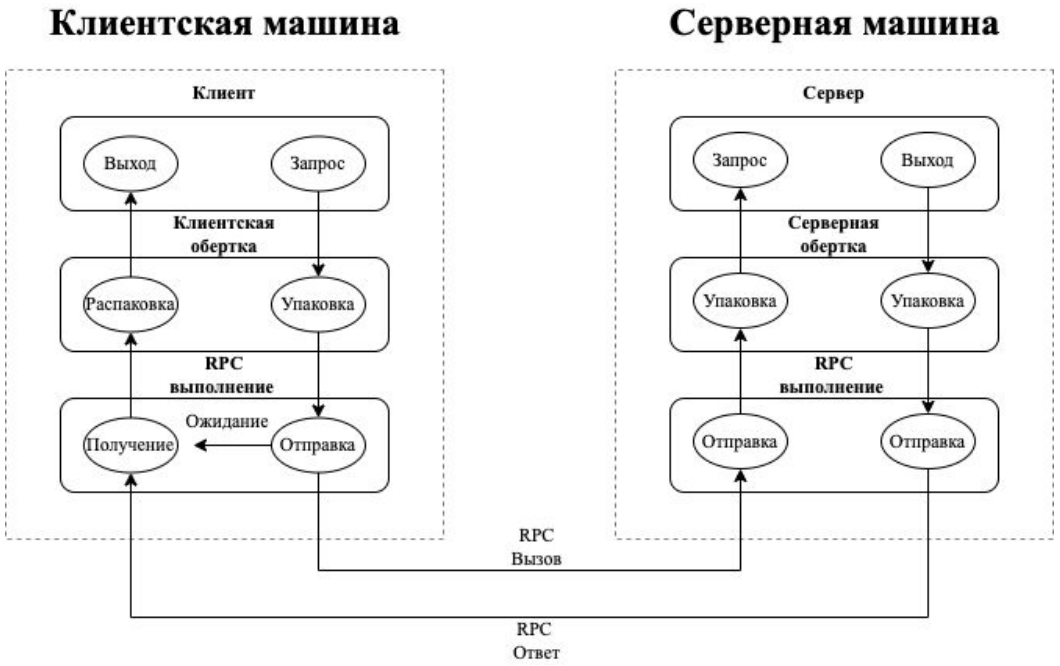
Обычно, процесс включает в себя два компонента:

— сетевой протокол для обмена данными по сети (транспорт);
— язык сериализации.

Реализации RPC в качестве транспорта используют TCP, UDP или HTTP.
В качестве формата данных используют форматы, основанные на JSON или XML.

RPC вызов

1. Клиент вызывает процедуру-обертку.
2. Обертка упаковывает параметры, копируя их в сообщение и передает их на сервер.
3. На сервере сообщение распаковывается и вызывается процедура.
4. Когда процедура завершается, она возвращается к обертке, которая упаковывает возвращаемые значения в сообщение.
5. Транспортный уровень отправляет сообщение с результатом обратно клиенту.
6. Клиент распаковывает сообщение, и возвращает результат.



Форматы сериализации

В RPC используется множество форматов сериализации данных.

Самые популярные построены на базе JSON.

В работе сравниваются:

- **JSON** - сериализатор, основанный на JSON по описанной в RFC-7159 спецификации;
- **JSON+GZIP** - сериализатор JSON с сжатием GZIP;
- **JSON+ZSTD** - сериализатор JSON с сжатием Zstandard;
- **CBOR** - бинарный сериализатор на основе JSON;
- **BSON** - сериализатор на основе BSON (Binary JavaScript Object Notation);
- **MessagePack** - бинарный сериализатор данных на основе JSON;
- **Protobuf** - бинарный сериализатор данных от Google.

Критерии сравнения

Для сравнения форматов следует выделить критерии.

Такими являются:

— максимальная вложенность;
— нумерация полей;
— эффективность компрессии данных (размер сериализованных данных);
— время, затрачиваемое на сериализацию;
— возможность работы из различных языков (Go, Python);
— поддержка версионности/эволюционирования структуры данных.

Сравнительная таблица постоянных характеристик

Формат	Бинарный	Макс. вложенность	Нумерация полей	Версионность	Языки
BSON	Да	65535	Нет	Да	Go,Python
CBOR	Да	Размер стека	Нет	Да	Go,Python
JSON	Нет	10000	Нет	Да	Go,Python
MessagePack	Да	Размер стека	Нет	Да	Go,Python
Protobuf	Да	100*	Да	Нет	Go,Python

Исследование непостоянных характеристик

Технические характеристики:

— Процессор: Apple M1 Pro.
— Память: 32 Гб.
— Операционная система: macOS Monterey 12.4.

Измеряемые характеристики:

— Название формата.
— **NS/op** - среднее время выполнения каждого вызова функции в наносекундах (чем меньше, тем лучше);
— **MB/s** - пропускная способность в мегабайтах (скорость обработки, чем больше, тем лучше);
— **B/op** - это число байт, выделяемых за операцию (чем меньше, тем лучше).

Сравнение непостоянных параметров

Golang

Строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	1 947 565	551.08	1 109 587
JSON+GZIP	7 505 140	6.56	1 228 276
JSON+ZSTD	2 072 994	24.83	1 178 614
CBOR	3 773 309	209.15	376 789
BSON	6 719 164	182.56	2 599 949
MessagePack	2 334 210	363.70	2 097 938
Protobuf	962 667	508.97	491 522
Десериализация			
JSON	15 610 498	110.25	4 385 190
JSON+GZIP	16 417 442	3.45	1 001 751
JSON+ZSTD	15 325 713	3.65	1 220 576
CBOR	112 696	15097.51	1 705 460
BSON	10 002 288	122.65	4 582 385
MessagePack	4 367 620	194.38	730 293
Protobuf	1 963 471	249.54	2 525 047

По времени обработки и пропускной способности лучше всего справляется Protobuf, по количеству выделяемой памяти - CBOR.

Числа и строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	199 762	461.77	125 136
JSON+GZIP	1 199 464	76.90	130 031
JSON+ZSTD	247 166	373.21	126 482
CBOR	78 878	1008.31	98 403
BSON	218 095	515.15	119 181
MessagePack	102 063	1092.18	261 500
Protobuf	103 150	666.70	73 728
Десериализация			
JSON	686 517	146.93	217 785
JSON+GZIP	1 011 915	23.04	165 771
JSON+ZSTD	780 738	29.78	393 457
CBOR	10 321	9773.44	108 017
BSON	476 252	235.91	184 688
MessagePack	93 540	1191.69	48
Protobuf	58 568	1174.20	152 241

По всем показателям лучше всего себя показал CBOR.

Сравнение непостоянных параметров

Python3

Строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	3 194	73.85	428 131
JSON+GZIP	439 313	0.1	875 000
JSON+ZSTD	9 603	6.1	1 137 784 615
CBOR	120 484	0.35	472 527
BSON	13 260 222	2.1	4 978 888 889
MessagePack	13 373	18.2	179 253 012
Protobuf	59 520	0.78	367 717
Десериализация			
JSON	171 805	0.12	15 625
JSON+GZIP	154 063	0.49	527 972
JSON+ZSTD	66 668	2.12	436 615
CBOR	574 250	17.3	743 483 333
BSON	15 681 333	1.1	7 436 666 667
MessagePack	207 438	0.37	701 786
Protobuf	650	22.3	12 631

Protobuf показал себя лучше всего в скорости обработки и требуемой памяти, в пропускной способности первое место занял JSON.

Числа и строки

Формат	Ns/op	MB/s	B/op
Сериализация			
JSON	12	26,56	569
JSON+GZIP	10 538	0,02	24 312
JSON+ZSTD	229	1,68	1 183
CBOR	2 913	0,01	181
BSON	7 351	0,05	38 274
MessagePack	4 547	1,33	24 312
Protobuf	104	2,27	1 859
Десериализация			
JSON	97	1,84	1 400
JSON+GZIP	373	0,32	771
JSON+ZSTD	118	3,36	1 221
CBOR	7 202	0,07	7 967
BSON	2 548	0,01	2 951
MessagePack	18 814	0,05	7 792
Protobuf	6	70,65	397

По всем параметрам лидерство занимает Protobuf.

Выводы

В ходе выполнения данной работы были выполнены следующие задачи

1. Проведен обзор предметной области и описаны ее термины.
2. Классифицированы форматы сериализации данных.
3. Выявлены критерии сравнения.
4. Проведено сравнение форматов.

Все поставленные задачи были решены. Цель данной работы была достигнута.

Исследование показало, что для наборов данных, состоящих из чисел и строк, лучше всего с сериализацией справляется protobuf. Но наличие схемы усложняет работу с ним.
В качестве альтернативы следует использовать CBOR, который чуть менее производительный, но более легкий в разработке формат сериализации.