Отчет по Лабораторной работе #3

Описание эксперимента:

1. Сравниваются 2 сериализатора: Jackson и Protobuf.
2. Оборудование:
    a. Процессор Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz   2.50 GHz
    b. Оперативная память 16,0 ГБ
3. Операционная система: Windows 10
4. Версия Java:
    a. openjdk 16.0.2 2021-07-20
    b. OpenJDK Runtime Environment (build 16.0.2+7-67)
    c. OpenJDK 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)

Результат запуска бенчмарка:

```
Benchmark                                          Mode  Cnt    Score     Error  Units
SerializationBenchmark.testDeserializeFromJson     avgt    5  941,409 ? 371,922  ns/op
SerializationBenchmark.testDeserializeInProtoFormat  avgt    5  201,081 ?  13,449  ns/op
SerializationBenchmark.testSerializeInProtoFormat  avgt    5   83,128 ?  12,192  ns/op
SerializationBenchmark.testSerializeToJson         avgt    5  485,056 ? 101,951  ns/op
```

Выводы:

- Сериализатор Protobuf оказался быстрее, чем Jackson, при сериализации и десериализации.
- Сериализатор Protobuf имеет смысл использовать, когда предъявляются высокие требования к времени сериализации / десериализации, либо когда есть жесткие ограничения по пропускной способности сети. Стоит заметить, что объект, сериализованный в формате Protobuf, занимает меньше места, чем объект, сериализованный в формате JSON.
- Сериализатор Jackson имеет смысл использовать, когда некритична разница в доли миллисекунды на сериализацию / десериализацию одного объекта. Также формат JSON является человекочитаемым и его проще использовать для взаимодействия с приложением и для отладки.

Дополнительные замечания:

1. Высокую скорость работы Protobuf можно объяснить тем, что для каждого поля объекта хранится размер его данных, что позволяет быстро десериализовывать объекты. В то же время для парсинга Json нужно искать соответствия открывающих / закрывающих кавычек и скобок.
2. Можно ускорить любой из сериализаторов, если использовать генерацию байткода для доступа к полям DTO вместо рефлексии.