Семинар 4. Событийно-ориентированные архитектуры

Принципы построения высоконагруженных систем

Георгий Семенов

Институт прикладных компьютерных наук
Университет ИТМО

осень 2025

Содержание

Оставшиеся активности на курсе

- Защита домашнего задания 4 - сегодня после семинара, в понедельник и вторник
- Ликвидация долгов: до $\approx$ 20 января, что нужно сделать – обсуждается индивидуально

Вспоминаем лекцию I

- Synchronous / Asynchronous
- Thread-Per-Request vs. Event Loop
- Очереди сообщений
  - Гарантии доставки: at-most-once, at-least-once, exactly-once
  - Dead Letter Queue (чтобы не останавливать конвейер)
  - Transactional Outbox Pattern (чтобы делать композитные штуки без потери данных)
- Kafka: быстро пишем в топик и читаем по указателю потребителями из партиций
- RabbitMQ: пишем в exchange, маршрутизируем по routing key, читаем из queue (DIRECT = exact match, FANOUT = broadcast, TOPIC = regex)

Вспоминаем лекцию II

- Теория массового обслуживания
  - Нужен карман запаса для толерантности к bursts
  - Формула Кингмана: среднее время ожидания в очереди заданий
- Event-Driven Architecture
  - Хореография – сервисы реагируют на события децентрализованно
  - Оркестрация – сервисы реагируют в порядке обхода координатором

Содержание

EDA: основные понятия

- Событие – иммутабельное сообщение, описывающее факт произошедшего действия
- Сервисы принимают/порождают сообщения в асинхронном режиме
  - Передаем ID событий – event notification (обратный callback)
  - Передаем событие целиком – event-carried state transfer (безопасность, нагрузка данными)

Форматы данных для представления событий

- Сообщение – это объектная сущность (например, JSON, XML, Protobuf, Avro)
- Совместимость сообщений
  - Backward - новые сервисы не ломаются от старых сообщений
  - Forward - старые сервисы могут видеть новые поля
- Schema Registry – централизованное хранилище схем сообщений

Эволюция схемы сообщения на примере protobuf

Version 1:

```
message Order {
    int32 order_id = 1;
    optional string customer_name = 2;
    optional double amount = 3;
    optional string status = 4;
}
```

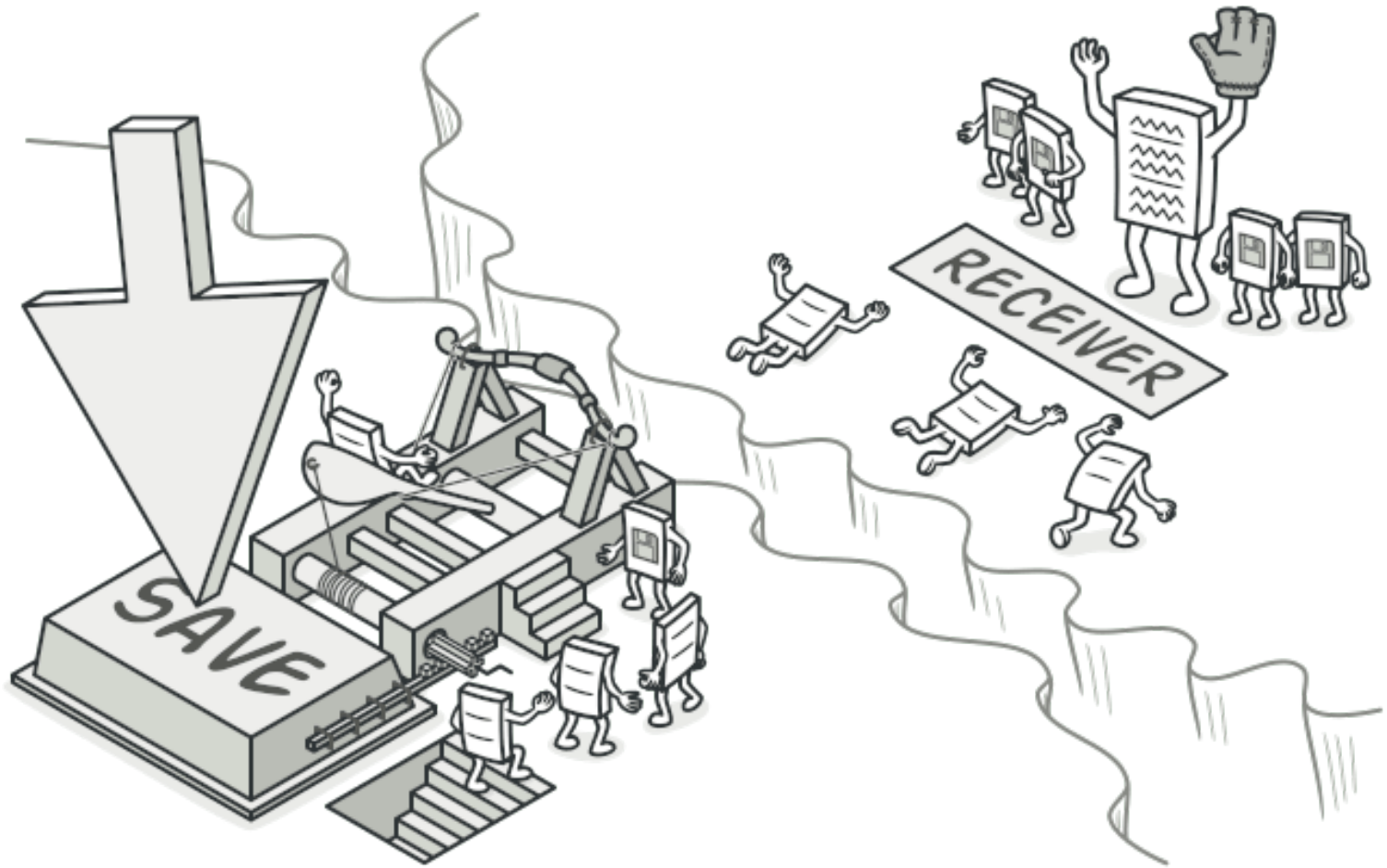
Эволюция схемы сообщения на примере protobuf

Version 2:

```
message Order {
    int32 order_id = 1;
    optional string customer_name = 2;
    optional double amount = 3 [deprecated=true];
    reserved 4;
    optional int64 timestamp = 5;
    optional string payment_method = 6;
}
```

Содержание

Команда - поведенческий паттерн ООП



Event Sourcing: основные понятия

- Источник истины – лог событий (Event Store), а не текущее состояние
- Каждое изменение состояния фиксируется как событие
- Состояние восстанавливается проигрыванием всех событий с начала
  - Полный аудит и история всех изменений
  - Временная машина – можем посмотреть состояние в любой момент
  - Отладка: повторяем последовательность событий
- Минусы: сложность, eventual consistency, размер хранилища (metadata overhead, retention, stabilization)
- Смотрим по ссылочке: microservices.io

CQRS: Command Query Responsibility Segregation

- Разделяем модели для чтения и записи
  - Write model (Command) – оптимизирован для изменений (в микросервисах)
  - Read model (Query) – оптимизирован для чтения (в специальном виде)
- Event Sourcing + CQRS работают в паре
  - События записываются в Event Store
  - Из событий строится Read Model (денормализованное представление)
  - Асинхронное обновление: eventual consistency
- Смотрим по ссылочке: microservices.io

SAGA: управление долгоживущими транзакциями

- Долгоживущая транзакция – несколько микросервисов, которые нужно скоординировать
- Два подхода:
  - Хореография (Choreography) – сервисы реагируют на события друг друга
  - Оркестрация (Orchestration) – есть координатор (Saga Orchestrator)
- Откат при ошибке: compensating transactions (выполняем обратные операции, aka undo-redo)
- Гарантия: каждый шаг либо успешен, либо откачен
- Посмотрим ссылочку здесь: microservices.io

Лирическое отступление: ETL (Extract, Transform, Load)

- Extract – извлекаем данные из источника
- Transform – преобразуем, очищаем, обогащаем
- Load – загружаем в хранилище
- Используется для интеграции разнородных систем
- Event-driven подход – streaming ETL (real-time вместо batch)
  - Apache Kafka + Spark Streaming, Flink, Kafka Streams
  - Меньше задержка, но сложнее в отладке

Содержание

Serverless: основные понятия

- Не о том, что нет серверов, а о том, что нет управления серверами
- Function as a Service (FaaS) – платим за время выполнения
- Холодный старт (cold start) – первый запрос медленнее
- Идеален для event-driven архитектур
- Примеры: AWS Lambda, Google Cloud Functions, Azure Functions
- Минусы: vendor lock-in, сложность с состоянием, стоимость при высокой нагрузке

Amazon Lambda

- Платим за вычисления (GB-seconds): (память $\times$ время выполнения)
- Триггеры: S3, API Gateway, SQS, SNS, DynamoDB Streams, Kinesis, EventBridge
- Максимум 15 минут на выполнение (для долгих задач – асинхронные вызовы)
- Инструменты: SAM (Serverless Application Model), Serverless Framework

Содержание

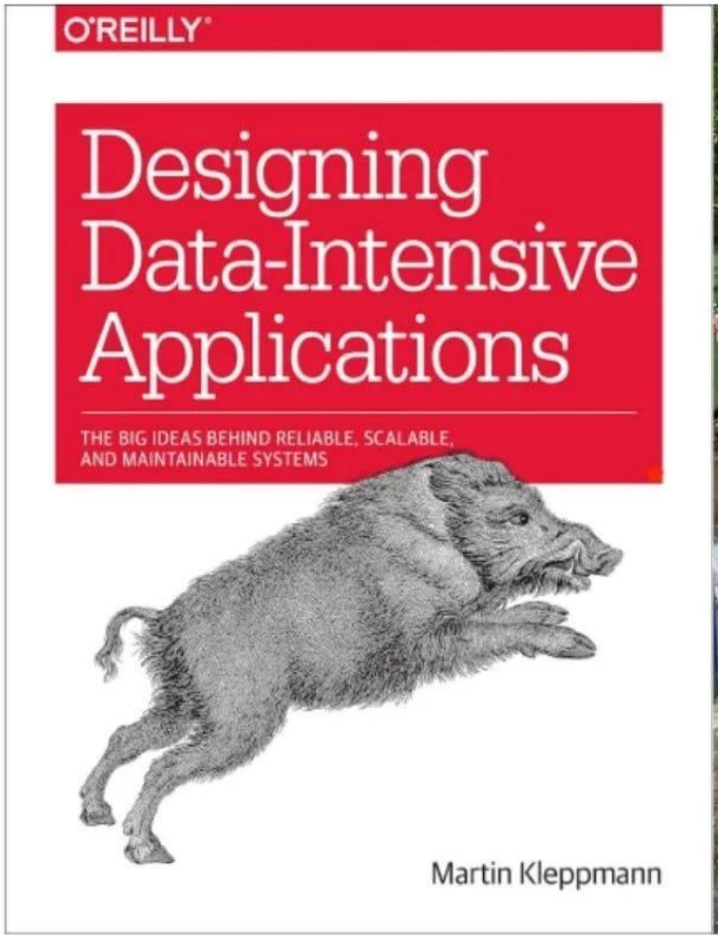
Демо: сервис обработки изображений

- NB! Пример сгенерирован AI.
- Папка **queues** $\rightarrow$ **./start.sh**

Содержание

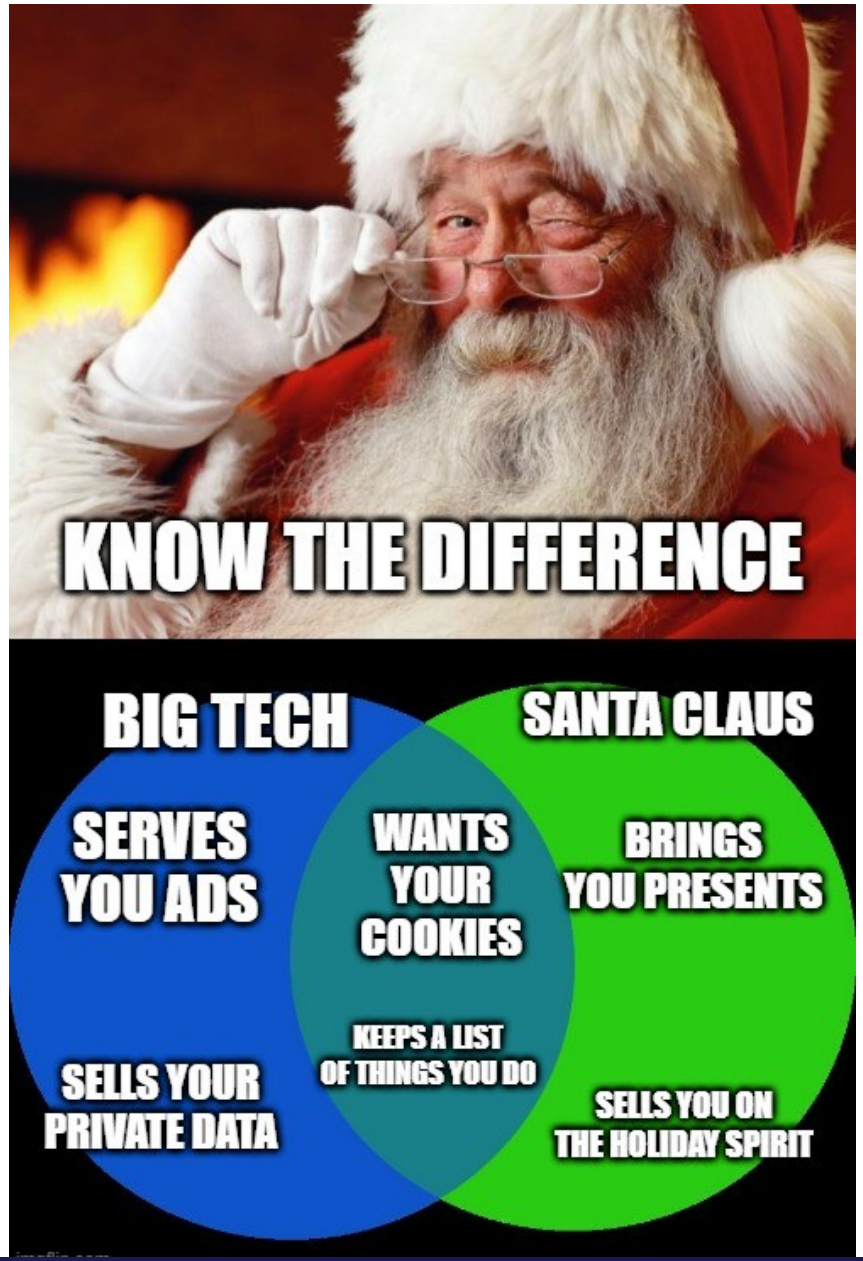
На кабана надейся, а сам...

interview preparation
vs actual work



Load balancing & Work-life balancing

Bigtech нужен и для благих дел



Спасибо!

Спасибо за терпение, работу и обратную связь!

Какие «академические» способы развития?

- Книги:
  - Release it! Проектирование и дизайн ПО для тех, кому не всё равно
  - Designing Data-Intensive Applications
  - Google SRE Books
  - Microservices Patterns
  - Распределенные данные
  - (*) Танненбаум. Операционные системы
  - (*) Банда четырех. Паттерны проектирования
- Школа анализа данных Яндекса: трек «Инфраструктура больших данных», поступление по альтернативному треку
- Андрей Суховицкий – лектор по проектированию и event sourcing
- Научные конференции (читать и обнаруживать направления): VLDB, SIGMOD, PODS, ICDT, Middleware, SOCC, EuroSys

Итоги

- Защищаем ДЗ-4, кому нужно – ликвидируем долги
- Отдыхаем и празднуем Новый год!