// JavaJub

Собес в VK

Группа балансеров · One-cloud

Java Middle

Вопросы, задачи и подготовка
к live-coding и техническому интервью

Стек

Java 21 · Spring Boot 3 · Kafka · PostgreSQL · Kubernetes · gRPC · HAProxy · Docker

// содержание

Содержание

01 Про VK и группу балансеров

VK — одна из крупнейших IT-компаний России (15 000+ сотрудников, 200+ продуктов, 95% аудитории рунета). Центр Технологий VK отвечает за IT-инфраструктуру всех продуктов: ВКонтакте, ОК, Дзен, Mail.ru, RuStore, VK Play.

One-cloud — технологический фундамент VK: единая среда запуска приложений, хранилищ, баз данных и сервисов. Группа балансеров занимается управлением и администрированием L3- и L4-балансеров облака. Через них идёт ВЕСЬ сервисный и пользовательский трафик — это tier 1 уровень стабильности.

Чем занимается группа балансеров

- Разработка Java-бэкенда инфраструктурного сервиса для регулирования трафика в дата-центрах
- Написание Java-клиентов для взаимодействия с другими инфраструктурными сервисами
- Создание API с авторизацией для взаимодействия пользователей через UI
- Управление L3- и L4-балансеров: чтобы другие разработчики не думали о тонкостях балансировки
- Поддержание высочайшего уровня стабильности (tier 1) — любой сбой = падение всех продуктов VK

Как устроен процесс найма

По отзывам кандидатов (DreamJob, Habr 879068, Habr 1006022): VK проводит единое техническое интервью для всех Java-вакансий, после которого распределяет по командам. Процесс похож на Яндекс, но обычно быстрее.

Этап	Формат	Длительность
HR-скрининг	Zoom/Telegram, мотивация, ЗП, стек	20–30 мин
Техническое интервью	Теория + live-coding + SQL	60–90 мин
Алгоритмическая секция	Live-coding, 2–3 задачи	45–60 мин
Архитектурная секция	System Design, Miro + Zoom	45–60 мин
Финал с командой	Знакомство с тимлидом и командой	30–60 мин

ФИШКА · Единое интервью

В VK техническое интервью общее для всех Java-вакансий. HR предлагает 2–3 команды на выбор, но тех-собес один. Это значит: готовиться нужно к общей Java-базе, а не к специфике балансеров. Специфика обсуждается на финале с командой.

ВНИМАНИЕ · Что бросается в глаза

> По отзывам на Habr: VK спрашивает алгоритмы и структуры данных даже у Middle. Бинарный поиск, хэш-таблицы, деревья, очереди с приоритетом — будьте готовы. На этапе SQL всё обычно проходит хорошо, а вот алгоритмы — главный фильтр.

02 Стек по вакансии

Группа балансеров — часть One-cloud, разработка ведётся на современном стеке без устаревшего кода.

Основной стек

- Java 21 — последняя LTS-версия, Virtual Threads, Records, Pattern Matching
- Spring / Spring Boot 3 — основной фреймворк
- Kafka — брокер сообщений для межсервисного взаимодействия
- PostgreSQL — основная СУБД
- Kubernetes — оркестрация контейнеров (One-cloud построен поверх K8s)
- Docker / Docker Compose — контейнеризация
- gRPC — внутренний RPC между инфраструктурными сервисами
- Prometheus + Grafana — мониторинг и метрики
- HAProxy / Nginx / Envoy — L3/L4/L7 балансеры (предметная область команды)

Будет плюсом

- Понимание сетевых протоколов: TCP/IP, UDP, HTTP/2, VRRP
- Опыт с L3/L4/L7 балансировкой (HAProxy, Nginx, Envoy, IPVS)
- Опыт работы с OpenStack, облачными платформами
- Ant, Gradle — системы сборки (указаны в вакансии)
- Навыки автоматизации тестирования
- Опыт разработки и интеграции сложных внутренних продуктов

> **СОВЕТ · Про Java 21**
>
> VK использует Java 21 — это важно. Готовьтесь к вопросам про Virtual Threads (Project Loom), Records, Sealed Classes, Pattern Matching for switch. Это не «будет плюсом», а рабочий стек.

03 Java Core — что точно спросят

equals/hashCode, HashMap, Stream API, многопоточность — must-have. VK спрашивает глубже среднего: устройство JVM, GC, ссылочные типы.

Базовые вопросы

- JDK, JRE, JVM.
  → *JVM — виртуальная машина (исполняет байт-код). JRE = JVM + библиотеки. JDK = JRE + инструменты. С Java 9 JRE отдельно не поставляется. Java 21: Virtual Threads, Records, Sealed Classes, Pattern Matching.*
- Области памяти JVM.
  → *Heap (Eden/Survivor/Old), Stack (фреймы), Metaspace (метаданные классов, раньше PermGen), PC Register, Native Method Stack, CodeCache (JIT). GC работает только с Heap.*
- Контракт equals/hashCode.
  → *a.equals(b) → hashCode одинаков. Обратное необязательно. Переопределяешь один — переопределяй оба. Нарушение ломает HashMap/HashSet: объект «теряется» при изменении поля в hashCode.*
- Что сломается, если hashCode() — константа?
  → *Все в одном bucket. Java 8+: при 8 коллизиях И capacity ≥ 64 → red-black tree O(log n) вместо O(1).*
- String Pool и immutability.
  → *Pool в heap — уникальные литералы. new String() — вне пула. intern() добавляет. Immutability: безопасность, потокобезопасность, кэширование hashCode, пул.*
- Generics: type erasure, PECS.
  → *Type erasure: в рантайме нет типового параметра. PECS: Producer Extends (читать), Consumer Super (писать). Wildcard <?> — только чтение как Object.*
- WeakReference / SoftReference / PhantomReference.
  → *Weak — GC собирает при первой сборке (WeakHashMap). Soft — при нехватке памяти (кэши). Phantom — постфинализационная очистка. ReferenceQueue для уведомлений.*
- Функциональные интерфейсы.
  → *Один абстрактный метод. Function<T,R>, Predicate<T>, Consumer<T>, Supplier<T>, UnaryOperator<T>, BiFunction, Comparator, Runnable, Callable.*
- Stream API.
  → *Промежуточные (ленивые): filter, map, flatMap, sorted, distinct. Терминальные: collect, forEach, count, reduce, toList. Обработка вертикальная. Стрим одноразовый.*
- Virtual Threads (Java 21).
  → *Лёгкие потоки, управляемые JVM (не ОС). Thread.ofVirtual().start(). Миллионы потоков без проблем. НЕ подходят для CPU-bound задач (нет preemption). Идеальны для I/O-bound (сетевые запросы, БД). Executors.newVirtualThreadPerTaskExecutor().*
- Records (Java 16+).
  → *Иммутабельные data-классы: record Point(int x, int y) {}. Автоматически: конструктор, getters, equals, hashCode, toString. Нельзя наследовать (implicitly final). Можно реализовывать интерфейсы.*
- Sealed Classes (Java 17+).
  → *Ограничение иерархии наследования: sealed class Shape permits Circle, Square. Компилятор гарантирует exhaustive switch. Связка с Pattern Matching for switch (Java 21).*
- final: класс, метод, поле.

→ *Класс — нельзя наследовать. Метод — нельзя переопределить. Поле — нельзя переприсвоить (мутабельное содержимое можно менять). effectively final — для лямбд.*

▸ Задачи «Что выведет?»

Тест 1. Integer cache

```
Integer a = 127, b = 127;
System.out.println(a == b); // true

Integer c = 128, d = 128;
System.out.println(c == d); // false
```

IntegerCache: -128..127. Для 127 — один объект. Для 128 — два разных. Мораль: для объектов — equals().

Тест 2. String Pool

```
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");

System.out.println(s1 == s2);    // true  (оба из пула)
System.out.println(s1 == s3);    // false (s3 — новый объект)
System.out.println(s1.equals(s3)); // true  (содержимое одинаковое)
```

Тест 3. Stream — ленивость

```
List.of(1,2,3,4,5).stream()
  .peek(x -> System.out.print("A" + x + " "))
  .filter(x -> x % 2 == 0)
  .peek(x -> System.out.print("B" + x + " "))
  .toList();
// A1 A2 B2 A3 A4 B4 A5
```

Вертикальная обработка: каждый элемент проходит всю цепочку. Нечётные не проходят filter → B не печатается.

> **СОВЕТ · Про Java 21**
>
> VK использует Java 21. Готовьтесь к вопросам про Virtual Threads, Records, Sealed Classes, Pattern Matching for switch. Пример: «Чем Virtual Thread отличается от Platform Thread? Когда НЕ стоит использовать Virtual Threads?» — CPU-bound задачи, synchronized блоки (pinning).

04 Коллекции

HashMap — абсолютный чемпион вопросов. В VK спрашивают глубоко: treeify threshold, resize, null-ключи, ConcurrentHashMap внутренности.

- HashMap — устройство.
  → *Node<K,V>[]. Размер — степень двойки (default 16). Индекс: (n-1) & hash(key). Коллизии — список. Java 8+: TREEIFY_THRESHOLD=8 И capacity ≥ 64 → red-black tree.*
- load factor и resize.
  → *0.75 по умолчанию. size >= capacity * loadFactor → resize (вдвое + перехеширование всех). Совет: initialCapacity = expectedSize / 0.75 + 1.*
- HashMap vs ConcurrentHashMap.
  → *HashMap: не потокобезопасен, null-ключ. ConcurrentHashMap: Java 8+ CAS + synchronized на головах бакетов. null запрещён. computeIfAbsent атомарен.*
- ArrayList vs LinkedList.
  → *ArrayList: O(1) доступ, CPU-кэш. LinkedList: O(1) вставка в начало. На практике ArrayList всегда лучше.*
- TreeMap vs LinkedHashMap.
  → *TreeMap: red-black tree, O(log n), отсортирован. LinkedHashMap: порядок вставки / accessOrder=true для LRU.*
- PriorityQueue.
  → *Min-heap по умолчанию. O(log n) offer/poll, O(1) peek. Для Top-K задач. Comparator для кастомного порядка.*
- Почему Tree индекс в БД, а не Hash?
  → *Хотя Hash = O(1), Tree (B-tree) поддерживает диапазонные запросы, сортировку, BETWEEN, LIKE 'abc%'. Hash — только точное совпадение.*

05 Многопоточность и JMM

В VK многопоточность критична — балансеры обрабатывают весь трафик. volatile vs synchronized спрашивают на каждом собесе. ThreadPoolExecutor, CAS, пулы потоков — обязательно.

- synchronized.
  → *Захват монитора. Instance → this. Static → Class. Reentrant (счётчик). Mutual exclusion + happens-before.*
- volatile.
  → *Видимость + запрет reordering. НЕ атомарность i++. Для атомарных — AtomicInteger/LongAdder.*
- happens-before.
  → *unlock → lock. volatile write → read. Thread.start() → первая инструкция. join(). final-поля после конструктора.*
- ConcurrentHashMap Java 8.
  → *CAS + synchronized на головах бакетов. Treeify при 8 коллизиях. null запрещён. computeIfAbsent атомарен.*
- ThreadPoolExecutor.
  → *corePoolSize, maxPoolSize, keepAliveTime, workQueue, threadFactory, rejectedExecutionHandler. newCachedThreadPool опасен — OOM.*
- Deadlock.
  → *Два+ потока ждут друг друга. Условия Коффмана. Решение: упорядочить захват мониторов, tryLock с таймаутом. Обнаружение: jstack.*
- CompletableFuture.
  → *thenApply (map), thenCompose (flatMap), thenCombine (join двух). exceptionally. Async-варианты в ForkJoinPool.*
- Virtual Threads и synchronized.
  → *Virtual Thread на synchronized → pinning (привязка к platform thread). Решение: ReentrantLock вместо synchronized. Это частый вопрос для Java 21.*

ЛОВУШКА · volatile counter++

volatile long counter; counter++ — НЕВЕРНО (три операции). Три корректных варианта: synchronized, AtomicLong.incrementAndGet(), LongAdder.increment(). LongAdder лучше при высоком contention.

06 Spring и Spring Boot

Spring Boot 3 — основной фреймворк VK для Java-сервисов. Спрашивают: прокси, жизненный цикл, автоконфигурация, транзакции.

- IoC и DI.
  → *IoC — контейнер управляет. DI — зависимости внедряются. Constructor injection лучший: final, явные зависимости, тестируемость.*
- Жизненный цикл бина.
  → *BeanDefinition → инстанцирование → DI → Aware → BPP.before → @PostConstruct → InitializingBean → init-method → BPP.after → ГОТОВ → @PreDestroy → destroy.*
- @Transactional.
  → *Прокси (JDK/CGLIB). Открывает транзакцию, commit/rollback. self-call минует прокси. Rollback на RuntimeException/Error; checked — нужен rollbackFor.*
- Propagation.
  → *REQUIRED (default), REQUIRES_NEW, NESTED, SUPPORTS, MANDATORY, NOT_SUPPORTED, NEVER.*
- @SpringBootApplication.
  → *@Configuration + @EnableAutoConfiguration + @ComponentScan.*
- Scope: prototype в singleton.
  → *Prototype-бин создастся один раз при инжекте. Решение: Provider<T>, ObjectFactory<T>, @Lookup.*
- Spring Boot 3 + Java 21.
  → *GraalVM native image, Virtual Threads support (spring.threads.virtual.enabled=true), Jakarta EE вместо javax.*

// JavaJub

Это только начало гайда.

VK · Middle

Полная версия — бесплатно в Telegram-канале:

- все вопросы с ответами по грейду
- задачи: live-coding, SQL, System Design — с разбором
- чек-лист «за 2 часа до собеса»

@java_jub

t.me/java_jub · подписка бесплатная