Java 后端工程师面试

Java 核心 (20 点)

1. 面向对象编程 (OOP) 原则：封装、继承、多态、抽象。
2. 泛型：类型参数、有界类型、通配符泛型。
3. 多线程：线程创建、生命周期、线程间通信。
4. JVM 内存管理：堆、栈、垃圾回收算法。
5. 异常处理：受检异常与非受检异常、try-catch-finally、多重捕获。
6. 序列化：Serializable 接口、自定义序列化。
7. 集合框架：List、Set、Map、Queue 接口及其实现。
8. Lambda 表达式与函数式接口：Predicate、Consumer、Supplier、Function。
9. Stream API：中间操作与终端操作、并行流。
10. 反射 API：运行时访问类、方法、字段，注解处理。
11. Java IO vs NIO：文件处理、基于通道的 I/O、非阻塞 I/O。
12. 日期与时间 API：LocalDate、LocalDateTime、Duration。
13. 网络编程：Socket 编程、URL 连接、HTTP 客户端。
14. Java 安全：加密、数字签名、安全编码实践。
15. Java 模块系统：JPMS（Java 平台模块系统）与模块化。
16. 枚举：枚举的使用、ordinal 值、自定义方法。
17. 注解：内置注解、自定义注解、注解处理。
18. 并发工具：CountDownLatch、CyclicBarrier、Semaphore。
19. 内存泄漏：原因、检测与预防。
20. 性能调优：JVM 参数、性能分析工具、内存优化。

Spring 生态系统 (20 点)

21. Spring IoC 容器：依赖注入、Bean 生命周期、作用域。
22. Spring Boot 自动配置：Spring Boot 如何自动配置 Bean。
23. Spring Data JPA：Repository 模式、CRUD 操作、查询方法。
24. Spring Security：认证、授权、保护 REST API。
25. Spring MVC：控制器方法、请求映射、视图解析。
26. Spring Cloud：服务发现（Eureka）、负载均衡（Ribbon）。
27. Spring AOP：面向切面编程、横切关注点、通知类型。
28. Spring Boot Actuator：监控端点、健康检查、指标收集。
29. Spring Profiles：环境特定配置、Profile 激活。
30. Spring Boot Starter 依赖：使用 Starter 简化依赖管理。
31. Spring Integration：系统集成、消息传递、适配器。

32. Spring Batch：批处理、任务调度、步骤实现。
33. Spring 缓存：缓存策略、注解、缓存管理器。
34. Spring WebFlux：响应式编程、非阻塞 I/O。
35. Spring Cloud Config：微服务的集中配置管理。
36. Spring Cloud Gateway：API 网关模式、路由、过滤。
37. Spring Boot 测试：使用 @SpringBootTest、MockMvc、TestRestClient。
38. Spring Data REST：将 Repository 暴露为 REST 服务。
39. Spring Cloud Stream：与消息代理（如 RabbitMQ、Kafka）集成。
40. Spring Cloud Sleuth：微服务中的分布式追踪与日志。

微服务架构 (20 点)

41. 服务发现：Eureka、Consul、Zookeeper 的工作原理。
42. API 网关：模式、路由、安全性。
43. 熔断器：使用 Hystrix、Resilience4j 实现容错。
44. 事件驱动架构：事件溯源、消息代理、事件处理器。
45. RESTful API 设计：HATEOAS、无状态设计、REST 约束。
46. GraphQL：实现 GraphQL API、模式定义、解析器。
47. 微服务通信：同步与异步通信。
48. Saga 模式：跨服务的分布式事务管理。
49. 健康检查：实现存活与就绪探针。
50. 契约优先开发：使用 Swagger 定义 API 契约。
51. API 版本控制：RESTful API 的版本控制策略。
52. 限流：实现限流以防止滥用。
53. 熔断器模式：实现回退与重试机制。
54. 微服务部署：使用 Docker、Kubernetes、云平台。
55. 服务网格：Istio、Linkerd 及其优势。
56. 事件协作：Saga 与 Choreography 模式。
57. 微服务安全：OAuth2、JWT、API 网关。
58. 监控与追踪：Prometheus、Grafana、Jaeger。
59. 微服务测试：集成测试、契约测试、端到端测试。
60. 每个服务的数据库：微服务中的数据管理与一致性。

数据库与缓存 (20 点)

61. SQL 连接：内连接、外连接、左连接、右连接、交叉连接。
62. ACID 属性：事务的原子性、一致性、隔离性、持久性。
63. NoSQL 数据库：文档存储、键值存储、图数据库。
64. Redis 缓存：内存数据存储、数据结构、持久化选项。

65. Memcached vs Redis：缓存解决方案对比。
66. 数据库分片：水平分区与负载均衡。
67. ORM 框架：Hibernate、MyBatis、JPA 规范。
68. JDBC 连接池：DataSource 实现与连接生命周期。
69. 全文搜索：在 Elasticsearch 中实现搜索。
70. 时间序列数据库：InfluxDB、OpenTSDB 用于时间数据。
71. 事务隔离级别：读未提交、读已提交、可重复读、串行化。
72. 索引策略：B 树、哈希索引、复合索引。
73. 数据库复制：主从复制、主主复制。
74. 数据库备份与恢复：数据保护策略。
75. 数据库性能分析：SQL Profiler、慢查询日志。
76. NoSQL 一致性模型：最终一致性、CAP 定理。
77. 数据库迁移：使用 Flyway、Liquibase 进行模式变更。
78. 缓存策略：Cache-aside、Read-through、Write-through。
79. 缓存失效：管理缓存过期与失效机制。
80. 数据库连接池：HikariCP、Tomcat JDBC 池配置。

并发与多线程 (20 点)

81. 线程生命周期：新建、可运行、运行中、阻塞、等待、终止。
82. 同步机制：锁、同步块、内置锁。
83. 可重入锁：相比同步块的优势、公平性、超时。
84. Executor 框架：ThreadPoolExecutor、ExecutorService、线程池配置。
85. Callable vs Runnable：区别与使用场景。
86. Java 内存模型：可见性、happens-before 关系、内存一致性。
87. volatile 关键字：确保变量修改的可见性。
88. 死锁预防：避免与检测死锁。
89. 异步编程：使用 CompletableFuture 实现非阻塞操作。
90. ScheduledExecutorService：以固定速率与延迟调度任务。
91. 线程池：固定、缓存、调度线程池。
92. 锁分段：通过分段锁减少锁争用。
93. 读写锁：允许多个读或单个写。
94. 等待与通知机制：使用 wait/notify 进行线程间通信。
95. 线程中断：处理中断与设计可中断任务。
96. 线程安全类：实现线程安全的单例模式。
97. 并发工具：CountDownLatch、CyclicBarrier、Semaphore。
98. Java 8+ 并发特性：并行流、Fork/Join 框架。

99. 多核编程：并行处理的挑战与解决方案。
100. 线程转储与分析：通过线程转储识别问题。

Web 服务器与负载均衡 (20 点)

101. Apache Tomcat 配置：设置连接器、context.xml、server.xml。
102. Nginx 反向代理：配置 proxy_pass、上游服务器、负载均衡。
103. HAProxy 高可用：设置故障转移与会话持久化。
104. Web 服务器安全：SSL/TLS 配置、安全头、防火墙规则。
105. 负载均衡算法：轮询、最少连接、IP 哈希。
106. 服务器端缓存：使用 Varnish、Redis 或内存缓存。
107. 监控工具：使用 Prometheus、Grafana、New Relic 进行服务器监控。
108. 生产环境日志：使用 ELK 栈或 Graylog 进行集中日志管理。
109. 水平与垂直扩展：理解权衡与使用场景。
110. Web 服务器性能调优：调整工作线程、连接超时、缓冲区。
111. 反向代理缓存：配置缓存头与过期时间。
112. Web 服务器负载测试：使用 Apache JMeter、Gatling 进行性能测试。
113. SSL 卸载：在负载均衡器处理 SSL/TLS 终止。
114. Web 服务器加固：安全最佳实践与漏洞评估。
115. 动态与静态内容服务：优化服务器配置。
116. Web 服务器集群：设置集群以实现高可用性。
117. Web 服务器认证：实现基本、摘要、OAuth 认证。
118. Web 服务器日志格式：常见日志格式与解析工具。
119. Web 服务器资源限制：配置连接、请求、带宽限制。
120. Web 服务器备份与恢复：灾难恢复策略。

CI/CD 与 DevOps (20 点)

121. Jenkins Pipeline 即代码：编写 Jenkinsfile 实现 CI/CD 流水线。
122. Docker 容器化：编写 Dockerfile、多阶段构建、容器编排。
123. Kubernetes 编排：部署、服务、Pod、扩展策略。
124. GitOps 原则：使用 Git 进行基础设施与配置管理。
125. Maven 与 Gradle 构建工具：依赖管理、插件、构建生命周期。
126. 单元与集成测试：使用 JUnit、Mockito、TestNG 编写测试。
127. 代码覆盖率工具：使用 Jacoco 测量代码覆盖率。
128. 静态代码分析：使用 SonarQube 进行代码质量检查。
129. 基础设施即代码 (IaC)：使用 Terraform、CloudFormation 进行基础设施配置。
130. 蓝绿部署：在部署期间最小化停机时间。
131. 金丝雀部署：逐步推出新功能。

132. CI 流水线中的自动化测试：将测试集成到构建阶段。
133. 环境管理：使用 Ansible、Chef 或 Puppet 进行配置管理。
134. CI/CD 最佳实践：持续集成、持续部署、持续交付。
135. 回滚策略：在部署失败时实现自动回滚。
136. 安全扫描：在流水线中集成 SAST、DAST 等安全检查。
137. 微服务的 CI/CD 流水线：管理多个服务的流水线。
138. 监控 CI/CD 流水线：对流水线失败与性能问题发出警报。
139. DevOps 工具生态系统：理解 Docker、Kubernetes、Jenkins、Ansible 等工具。
140. 云原生应用的 CI/CD：在云平台上部署应用。

设计模式与最佳实践 (20 点)

141. 单例模式：实现线程安全的单例。
142. 工厂模式：在不指定具体类的情况下创建对象。
143. 策略模式：封装算法并在它们之间切换。
144. SOLID 原则：理解并应用单一职责、开闭原则、里氏替换、接口隔离、依赖反转。
145. 依赖注入：减少耦合，提高代码可维护性。
146. 事件溯源模式：通过存储事件重建应用状态。
147. CQRS 架构：分离命令与查询职责。
148. 可扩展性设计：使用水平扩展、分片、负载均衡。
149. 代码重构技术：提取方法、重命名变量、简化条件。
150. 代码整洁实践：编写可读、可维护、自文档化的代码。
151. 测试驱动开发 (TDD)：在实现之前编写测试。
152. 代码版本控制：使用 Git 分支策略，如 GitFlow、Trunk-Based Development。
153. 可维护性设计：使用模块化设计、关注点分离。
154. 避免反模式：上帝类、面条代码、紧耦合。
155. 安全性设计：实现最小权限、纵深防御。
156. 性能设计：优化算法、减少 I/O 操作。
157. 可靠性设计：实现冗余、容错、错误处理。
158. 可扩展性设计：使用插件、扩展、开放 API。
159. 可用性设计：确保 API 直观且文档完善。
160. 可测试性设计：编写易于测试与模拟的代码。

安全 (20 点)

161. OAuth2 与 JWT：实现基于令牌的认证。
162. 基于角色的访问控制 (RBAC)：为用户分配角色与权限。
163. 安全头：实现内容安全策略、X-Frame-Options。
164. SQL 注入预防：使用预处理语句与参数化查询。

165. 跨站脚本 (XSS) 防护：对输入与输出进行消毒。
166. 加密与解密：使用 AES、RSA 进行数据保护。
167. 安全编码实践：避免常见漏洞，如缓冲区溢出。
168. 实现审计跟踪：记录用户操作与系统事件。
169. 处理敏感数据：使用哈希算法安全存储密码。
170. 合规性：遵守 GDPR、PCI-DSS 等数据保护法规。
171. 实现双因素认证 (2FA)：增加额外的安全层。
172. 安全测试：渗透测试、漏洞评估。
173. 安全通信协议：实现 SSL/TLS 进行数据加密。
174. 安全会话管理：管理会话令牌与超时。
175. 实现 Web 应用防火墙 (WAF)：防御常见攻击。
176. 安全监控与警报：使用 SIEM 进行威胁检测。
177. 微服务安全：保护服务间通信。
178. 实现 CAPTCHA 防止机器人攻击：防止自动化攻击。
179. CI/CD 流水线中的安全：在构建期间扫描漏洞。
180. 安全设计：从开发初期就考虑安全性。

性能调优与优化 (20 点)

181. Java 应用性能分析：使用 JProfiler、VisualVM 进行性能分析。
182. 垃圾回收调优：调整 GC 参数以优化性能。
183. 数据库查询优化：索引、查询重写、使用执行计划。
184. 缓存策略：使用分布式缓存、缓存失效机制。
185. 负载测试与压力测试：识别性能瓶颈。
186. 优化 RESTful API：减少响应时间、最小化数据传输。
187. 减少网络延迟：使用 CDN、优化 API 调用。
188. 连接池大小：确定数据库与连接的最佳池大小。
189. 监控与警报设置：使用 Prometheus、Grafana 进行实时监控。
190. 识别与解决瓶颈：分析 CPU、内存、I/O 使用情况。
191. 优化 Java 堆设置：为不同环境设置合适的堆大小。
192. 减少垃圾回收停顿：使用 G1GC、ZGC 实现低延迟应用。
193. 优化磁盘 I/O：使用 SSD、RAID 配置、文件系统优化。
194. 缓存 vs 存储：决定何时缓存数据或存储到数据库。
195. 优化日志：减少日志开销、管理日志量。
196. 优化并发访问：高效使用锁、减少争用。
197. 内存使用分析：识别内存泄漏、优化对象分配。
198. 优化线程池大小：在太少与太多线程之间找到平衡。

199. 优化数据结构：为特定用例选择合适的数据结构。
200. 性能指标与 KPI：定义并跟踪应用的关键性能指标。