Java 后端工程师面试

Java 核心 (20 分)

1. 面向对象编程（OOP）原则：封装、继承、多态、抽象。
2. Java 泛型：使用类型参数、有界类型和通配符泛型。
3. Java 多线程：创建线程、线程生命周期和线程间通信。
4. JVM 内存管理：堆、栈、PermGen/Survivor 空间、垃圾回收算法。
5. 异常处理：检查和非检查异常、try-catch 块、finally 和多重捕获。
6. Java 序列化：Serializable 接口、自定义序列化使用 writeObject 和 readObject。
7. Java 集合框架：List、Set、Map、Queue 接口及其实现。
8. Lambda 表达式和函数式接口：使用谓词、消费者、供应者和函数。
9. 流 API：中间和终端操作、并行流和流管道。
10. 反射 API：在运行时访问类、方法和字段、注解处理。
11. Java IO vs NIO：文件处理的差异、基于通道的 I/O 和非阻塞 I/O。
12. Java 日期和时间 API：使用 LocalDate、LocalDateTime 和 Duration。
13. Java 网络编程：套接字编程、URL 连接和 HTTP 客户端。
14. Java 安全性：加密、数字签名和安全编码实践。
15. Java 模块：理解 JPMS（Java 平台模块系统）和模块化。
16. Java 枚举：使用枚举、序数值和自定义枚举方法。
17. Java 注解：内置注解、自定义注解和注解处理。
18. Java 并发工具：CountDownLatch、CyclicBarrier、Semaphore 和 Exchanger。
19. Java 内存泄漏：原因、检测和预防策略。
20. Java 性能调优：JVM 选项、分析工具和内存优化技术。

Spring 生态系统 (20 分)

21. Spring IoC 容器：依赖注入、bean 生命周期和作用域。
22. Spring Boot 自动配置：Spring Boot 如何自动配置 bean。
23. Spring Data JPA：仓库模式、CRUD 操作和查询方法。
24. Spring Security：身份验证、授权和保护 REST API。

25. Spring MVC：控制器方法、请求映射和视图解析。
26. Spring Cloud：使用 Eureka 的服务发现、使用 Ribbon 的负载均衡。
27. Spring AOP：面向切面编程、交叉关注点和建议类型。
28. Spring Boot Actuator：监控端点、健康检查和指标收集。
29. Spring 配置文件：环境特定配置和配置文件激活。
30. Spring Boot Starter 依赖：使用启动器简化依赖管理。
31. Spring 集成：集成不同系统、消息传递和适配器。
32. Spring Batch：批处理、作业调度和步骤实现。
33. Spring 缓存：缓存策略、注解和缓存管理器。
34. Spring WebFlux：响应式编程、非阻塞 I/O 和 WebFlux 框架。
35. Spring Cloud Config：微服务的集中化配置管理。
36. Spring Cloud Gateway：API 网关模式、路由和过滤。
37. Spring Boot 测试：使用 @SpringBootTest、MockMvc 和 TestRestClient。
38. Spring Data REST：将仓库公开为 RESTful 服务。
39. Spring Cloud Stream：与消息代理（如 RabbitMQ 和 Kafka）集成。
40. Spring Cloud Sleuth：微服务中的分布式追踪和日志记录。

微服务架构 (20 分)

41. 服务发现：Eureka、Consul 和 Zookeeper 的工作原理。
42. API 网关：网关模式、路由和安全性。
43. 断路器：使用 Hystrix、Resilience4j 实现弹性。
44. 事件驱动架构：事件源、消息代理和事件处理程序。
45. RESTful API 设计：HATEOAS、无状态设计和 REST 约束。
46. GraphQL：实现 GraphQL API、模式定义和解析器。
47. 微服务通信：同步 vs 异步通信。
48. Saga 模式：跨服务管理分布式事务。
49. 健康检查：实现活跃性和就绪性探针。
50. 契约优先开发：使用 Swagger 进行 API 契约。
51. API 版本控制：RESTful API 版本控制策略。

52. 速率限制：实现速率限制以防止滥用。
53. 断路器模式：实现回退和重试。
54. 微服务部署：使用 Docker、Kubernetes 和云平台。
55. 服务网格：了解 Istio、Linkerd 及其好处。
56. 事件协作：Saga vs 编排模式。
57. 微服务安全：OAuth2、JWT 和 API 网关。
58. 监控和追踪：Prometheus、Grafana 和 Jaeger 等工具。
59. 微服务测试：集成测试、契约测试和端到端测试。
60. 每个服务的数据库：微服务中的数据管理和一致性。

数据库和缓存 (20 分)

61. SQL 连接：内连接、外连接、左连接、右连接和交叉连接。
62. ACID 属性：事务中的原子性、一致性、隔离性和持久性。
63. NoSQL 数据库：文档存储、键值存储和图数据库。
64. Redis 缓存：内存数据存储、数据结构和持久化选项。
65. Memcached vs Redis：比较缓存解决方案。
66. 数据库分片：水平分区和负载均衡。
67. ORM 框架：Hibernate、MyBatis 和 JPA 规范。
68. JDBC 连接池：DataSource 实现和连接生命周期。
69. 全文搜索：在 Elasticsearch 等数据库中实现搜索。
70. 时间序列数据库：InfluxDB、OpenTSDB 等时间序列数据。
71. 事务隔离级别：未提交读、已提交读、可重复读和可串行化。
72. 索引策略：B-tree、哈希索引和复合索引。
73. 数据库复制：主从、主主设置。
74. 数据库备份和恢复：数据保护策略。
75. 数据库分析：SQL Profiler、慢查询日志等工具。
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 进行模式更改。
78. 缓存策略：旁路缓存、读穿透、写穿透模式。

79. 缓存失效：管理缓存过期和失效。
80. 数据库连接池：HikariCP、Tomcat JDBC 池配置。

并发和多线程 (20 分)

81. 线程生命周期：新建、可运行、运行、阻塞、等待、终止。
82. 同步机制：锁、同步块和内在锁。
83. 可重入锁：与同步块相比的优势、公平性和超时。
84. 执行器框架：ThreadPoolExecutor、ExecutorService 和线程池配置。
85. Callable vs Runnable：差异和用例。
86. Java 内存模型：可见性、happens-before 关系和内存一致性。
87. volatile 关键字：确保变量变化在线程间的可见性。
88. 死锁预防：避免和检测死锁。
89. 异步编程：使用 CompletableFuture 进行非阻塞操作。
90. ScheduledExecutorService：按固定速率和延迟调度任务。
91. 线程池：固定、缓存和调度线程池。
92. 锁分片：使用分片锁减少锁争用。
93. 读写锁：允许多个读者或一个写者。
94. 等待和通知机制：使用 wait/notify 进行线程间通信。
95. 线程中断：处理中断和设计可中断任务。
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：并行处理的挑战和解决方案。
100. 线程转储和分析：使用线程转储识别问题。

Web 服务器和负载均衡 (20 分)

101. Apache Tomcat 配置：设置连接器、context.xml 和 server.xml。
102. Nginx 作为反向代理：配置 proxy_pass、上游服务器和负载均衡。
103. HAProxy 高可用性：设置故障转移和会话持久性。

104. Web 服务器安全：SSL/TLS 配置、安全头和防火墙规则。

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 和 New Relic 进行服务器监控。

108. 生产日志记录：使用 ELK 栈或 Graylog 进行集中化日志记录。

109. 水平 vs 垂直扩展：理解权衡和用例。

110. Web 服务器性能调优：调整工作线程、连接超时和缓冲区。

111. 反向代理缓存：配置缓存头和过期时间。

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 服务器硬化：安全最佳实践和漏洞评估。

115. 动态 vs 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、摘要和 OAuth 认证。

118. Web 服务器日志格式：常见日志格式和解析工具。

119. Web 服务器资源限制：配置连接、请求和带宽限制。

120. Web 服务器备份和恢复：灾难恢复策略。

CI/CD 和 DevOps (20 分)

121. Jenkins 管道即代码：编写 Jenkinsfiles 进行 CI/CD 管道。

122. Docker 容器化：创建 Dockerfile、多阶段构建和容器编排。

123. Kubernetes 编排：部署、服务、Pod 和扩展策略。

124. GitOps 原则：使用 Git 进行基础设施和配置管理。

125. Maven 和 Gradle 构建工具：依赖管理、插件和构建生命周期。

126. 单元和集成测试：使用 JUnit、Mockito 和 TestNG 编写测试。

127. 代码覆盖工具：使用 Jacoco 衡量代码覆盖率。

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 进行基础设施供应。

130. 蓝绿部署：最小化部署期间的停机时间。

131. 金丝雀部署：逐步推出新功能。

132. 自动化测试在 CI 管道中：将测试与构建阶段集成。

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、持续部署和持续交付。

135. 回滚策略：在部署失败时实现自动回滚。

136. 安全扫描：在管道中集成安全检查，如 SAST、DAST。

137. CI/CD 管道的微服务：管理多个服务的管道。

138. 监控 CI/CD 管道：在管道失败和性能问题时发出警报。

139. DevOps 工具生态系统：了解 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程序。

设计模式和最佳实践 (20 分)

141. 单例模式：实现线程安全的单例。

142. 工厂模式：创建对象而不指定具体类。

143. 策略模式：封装算法并在它们之间切换。

144. SOLID 原则：理解和应用单一职责、开闭、里氏替换、接口隔离和依赖倒置。

145. 依赖注入：减少耦合并提高代码可维护性。

146. 事件源模式：存储事件以重建应用程序状态。

147. CQRS 架构：分离命令和查询职责。

148. 设计可扩展性：使用水平扩展、分片和负载均衡。

149. 代码重构技术：提取方法、重命名变量和简化条件语句。

150. 清晰代码实践：编写可读、可维护和自文档化的代码。

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、基于主干开发。

153. 设计可维护性：使用模块化设计、关注点分离。

154. 反模式：避免神类、意大利面条代码和紧耦合。

155. 设计安全性：实现最小权限、防御深度。

156. 设计性能：优化算法、减少 I/O 操作。

157. 设计可靠性：实现冗余、容错和错误处理。

158. 设计可扩展性：使用插件、扩展和开放 API。

159. 设计可用性：确保 API 直观且文档齐全。

160. 设计可测试性：编写易于测试和模拟的代码。

安全 (20 分)

161. OAuth2 和 JWT：实现基于令牌的身份验证。

162. 基于角色的访问控制（RBAC）：为用户分配角色和权限。

163. 安全头：实现内容安全策略、X-Frame-Options。

164. SQL 注入防护：使用准备语句和参数化查询。

165. 跨站脚本（XSS）保护：对输入和输出进行清理。

166. 加密和解密：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. 实现审计日志：记录用户操作和系统事件。

169. 处理敏感数据：使用哈希算法安全存储密码。

170. 遵守法规：GDPR、PCI-DSS 和数据保护法规。

171. 实现双因素认证（2FA）：添加额外的安全层。

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：管理会话令牌和超时。

175. 实现 Web 应用防火墙（WAF）：保护免受常见攻击。

176. 安全监控和警报：使用 SIEM 进行威胁检测。

177. 微服务中的安全最佳实践：保护服务间通信。

178. 实现 CAPTCHA 以防止机器人攻击：防止自动化攻击。

179. CI/CD 管道中的安全：在构建期间扫描漏洞。

180. 实现安全设计：从开发过程开始集成安全。

性能调优和优化 (20 分)

181. 分析 Java 应用程序：使用 JProfiler、VisualVM 进行性能分析。

182. 垃圾回收调优：调整 GC 参数以提高性能。

183. 数据库查询优化：索引、查询重写和使用解释计划。

184. 缓存策略：使用分布式缓存、缓存失效机制。

185. 负载测试和压力测试：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：确定数据库和连接的最佳池大小。

189. 监控和警报设置：使用 Prometheus、Grafana 进行实时监控。

190. 识别和解决瓶颈：分析 CPU、内存和 I/O 使用情况。

191. 优化 Java 堆设置：为不同环境设置适当的堆大小。

192. 减少垃圾回收暂停：使用 G1GC、ZGC 进行低延迟应用。

193. 优化磁盘 I/O：使用 SSD、RAID 配置和文件系统优化。

194. 缓存 vs 存储：决定何时缓存数据或将其存储在数据库中。

195. 优化日志记录：减少日志开销并管理日志量。

196. 优化并发访问：有效使用锁并最小化争用。

197. 分析内存使用：识别内存泄漏并优化对象分配。

198. 优化线程池大小：在过少和过多线程之间平衡。

199. 优化数据结构：为特定用例选择合适的数据结构。

200. 性能指标和 KPI：定义和跟踪应用程序的关键性能指标。