Ingénieur Backend Java - Entretien

Java Core (20 points)

1. Compréhension des principes de la POO : Encapsulation, héritage, polymorphisme, abstraction.
2. Génériques en Java : Utilisation des paramètres de type, des types bornés et des génériques sauvages.
3. Multithreading en Java : Création de threads, cycle de vie des threads et communication inter-threads.
4. Gestion de la mémoire JVM : Heap, Stack, espaces PermGen/Survivor, algorithmes de collecte des ordures.
5. Gestion des exceptions : Exceptions vérifiées et non vérifiées, blocs try-catch, finally et multi-catch.
6. Sérialisation en Java : Interface Serializable, sérialisation personnalisée avec writeObject et readObject.
7. Framework des collections Java : Interfaces List, Set, Map, Queue et leurs implémentations.
8. Expressions lambda et interfaces fonctionnelles : Utilisation des prédicats, consommateurs, fournisseurs et fonctions.
9. API Stream : Opérations intermédiaires et terminales, flux parallèles et pipeline de flux.
10. API Reflection : Accès aux classes, méthodes et champs en temps d'exécution, traitement des annotations.
11. Java IO vs NIO : Différences dans la gestion des fichiers, I/O basé sur les canaux et I/O non bloquant.
12. API Date et Heure Java : Travail avec LocalDate, LocalDateTime et Duration.
13. Réseautage Java : Programmation de sockets, connexions URL et clients HTTP.
14. Sécurité Java : Cryptographie, signatures numériques et pratiques de codage sécurisé.
15. Modules Java : Compréhension du JPMS (Java Platform Module System) et de la modularité.
16. Énumérations Java : Utilisation des énumérations, valeurs ordinales et méthodes personnalisées dans les énumérations.
17. Annotations Java : Annotations intégrées, annotations personnalisées et traitement des annotations.
18. Utilitaires de concourrence Java : CountDownLatch, CyclicBarrier, Semaphore et Exchanger.
19. Fuites de mémoire en Java : Causes, détection et stratégies de prévention.
20. Optimisation des performances Java : Options JVM, outils de profilage et techniques d'optimisation de la mémoire.

Écosystème Spring (20 points)

21. Conteneur IoC Spring : Injection de dépendances, cycle de vie des beans et portée.

22. Auto-configuration Spring Boot : Comment Spring Boot configure automatiquement les beans.
23. Spring Data JPA : Modèles de dépôt, opérations CRUD et méthodes de requête.
24. Spring Security : Authentification, autorisation et sécurisation des API REST.
25. Spring MVC : Méthodes de contrôleur, mappage des requêtes et résolution des vues.
26. Spring Cloud : Découverte de services avec Eureka, équilibrage de charge avec Ribbon.
27. Spring AOP : Programmation orientée aspect, préoccupations transversales et types de conseils.
28. Spring Boot Actuator : Points de terminaison de surveillance, vérifications de santé et collecte de métriques.
29. Profils Spring : Configurations spécifiques à l'environnement et activation des profils.
30. Dépendances Spring Boot Starter : Utilisation des starters pour simplifier la gestion des dépendances.
31. Spring Integration : Intégration de différents systèmes, messagerie et adaptateurs.
32. Spring Batch : Traitement par lots, planification des tâches et implémentations des étapes.
33. Spring Cache : Stratégies de mise en cache, annotations et gestionnaires de cache.
34. Spring WebFlux : Programmation réactive, I/O non bloquant et frameworks WebFlux.
35. Spring Cloud Config : Gestion centralisée de la configuration pour les microservices.
36. Spring Cloud Gateway : Modèles de passerelle API, routage et filtrage.
37. Tests Spring Boot : Utilisation de @SpringBootTest, MockMvc et TestRestClient.
38. Spring Data REST : Exposition des dépôts en tant que services RESTful.
39. Spring Cloud Stream : Intégration avec des courtiers de messages comme RabbitMQ et Kafka.
40. Spring Cloud Sleuth : Tracage distribué et journalisation dans les microservices.

Architecture des Microservices (20 points)

41. Découverte de services : Fonctionnement d'Eureka, Consul et Zookeeper.
42. Passerelle API : Modèles, routage et sécurité dans les passerelles API.
43. Disjoncteur : Mise en œuvre de la résilience avec Hystrix, Resilience4j.
44. Architecture orientée événements : Source d'événements, courtiers de messages et gestionnaires d'événements.
45. Conception d'API RESTful : HATEOAS, conception sans état et contraintes REST.
46. GraphQL : Mise en œuvre des API GraphQL, définitions de schéma et résolveurs.

47. Communication des microservices : Communication synchrone vs asynchrone.
48. Modèle Saga : Gestion des transactions distribuées entre services.
49. Vérifications de santé : Mise en œuvre des sondes de vivacité et de préparation.
50. Développement contract-first : Utilisation de Swagger pour les contrats API.
51. Versioning des API : Stratégies de versionnement des API RESTful.
52. Limitation des taux : Mise en œuvre des limites de taux pour prévenir l'abus.
53. Modèles de disjoncteur : Mise en œuvre des retours en arrière et des réessais.
54. Déploiement des microservices : Utilisation de Docker, Kubernetes et plateformes cloud.
55. Maillage de services : Compréhension d'Istio, Linkerd et leurs avantages.
56. Collaboration par événements : Modèles Saga vs Choreography.
57. Sécurité des microservices : OAuth2, JWT et passerelles API.
58. Surveillance et traçage : Outils comme Prometheus, Grafana et Jaeger.
59. Tests des microservices : Tests d'intégration, tests de contrat et tests de bout en bout.
60. Base de données par service : Gestion des données et cohérence dans les microservices.

Bases de données et mise en cache (20 points)

61. Jointures SQL : Jointures internes, externes, gauches, droites et croisées.
62. Propriétés ACID : Atomicité, cohérence, isolation, durabilité dans les transactions.
63. Bases de données NoSQL : Magasins de documents, magasins clé-valeur et bases de données graphiques.
64. Mise en cache Redis : Magasin de données en mémoire, structures de données et options de persistance.
65. Memcached vs Redis : Comparaison des solutions de mise en cache.
66. Partitionnement de la base de données : Partitionnement horizontal et équilibrage de charge.
67. Frameworks ORM : Hibernate, MyBatis et spécifications JPA.
68. Pooling de connexions JDBC : Implémentations DataSource et cycle de vie des connexions.
69. Recherche en texte intégral : Mise en œuvre de la recherche dans des bases de données comme Elasticsearch.
70. Bases de données de séries temporelles : InfluxDB, OpenTSDB pour les données basées sur le temps.
71. Niveaux d'isolation des transactions : Non engagé, engagé, lecture répétable, sérialisable.

72. Stratégies d’indexation : Index B-tree, index hash et index composites.
73. Réplication de base de données : Configurations maître-esclave, maître-maître.
74. Sauvegarde et récupération de base de données : Stratégies de protection des données.
75. Profilage de base de données : Outils comme SQL Profiler, journaux de requêtes lentes.
76. Modèles de cohérence NoSQL : Cohérence éventuelle, théorème CAP.
77. Migrations de base de données : Utilisation de Flyway, Liquibase pour les changements de schéma.
78. Stratégies de mise en cache : Modèles cache-aside, read-through, write-through.
79. Invalidation de cache : Gestion de l’expiration et de l’invalidation du cache.
80. Pooling de connexions de base de données : Configurations HikariCP, Tomcat JDBC pool.

Concurrence et multithreading (20 points)

81. Cycle de vie des threads : Nouveau, exécutable, en cours d’exécution, bloqué, en attente, terminé.
82. Mécanismes de synchronisation : Verrous, blocs synchronisés et verrous intrinsèques.
83. Verrous réentrants : Avantages par rapport aux blocs synchronisés, équité et délais d’attente.
84. Framework d’exécution : ThreadPoolExecutor, ExecutorService et configurations de pool de threads.
85. Callable vs Runnable : Différences et cas d’utilisation.
86. Modèle de mémoire Java : Visibilité, relations happens-before et cohérence de la mémoire.
87. Mot-clé volatile : Assurer la visibilité des modifications de variables entre threads.
88. Prévention des interblocages : Éviter et détecter les interblocages.
89. Programmation asynchrone : Utilisation de CompletableFuture pour les opérations non bloquantes.
90. ScheduledExecutorService : Planification des tâches avec des taux fixes et des délais.
91. Pools de threads : Pools fixes, mis en cache et planifiés.
92. Rayures de verrou : Réduction de la contention des verrous avec des verrous rayés.
93. Verrous de lecture-écriture : Autorisation de plusieurs lecteurs ou d’un seul écrivain.
94. Mécanismes wait et notify : Communication inter-threads utilisant wait/notify.
95. Interruption des threads : Gestion des interruptions et conception des tâches interruptibles.
96. Classes thread-safe : Mise en œuvre des modèles de singleton thread-safe.
97. Utilitaires de concourrence : CountDownLatch, CyclicBarrier, Semaphore.
98. Fonctionnalités de concourrence Java 8+ : Flux parallèles, framework fork-join.

99. Programmation multicœur : Défis et solutions pour le traitement parallèle.
100. Dumps de threads et analyse : Identification des problèmes avec les dumps de threads.

Serveurs web et équilibrage de charge (20 points)

101. Configuration Apache Tomcat : Configuration des connecteurs, context.xml et server.xml.
102. Nginx en tant que proxy inverse : Configuration de proxy_pass, serveurs amont et équilibrage de charge.
103. HAProxy pour la haute disponibilité : Configuration de la bascule et de la persistance de session.
104. Sécurité du serveur web : Configurations SSL/TLS, en-têtes de sécurité et règles de pare-feu.
105. Algorithmes d'équilibrage de charge : Round Robin, Least Connections, IP Hash.
106. Mise en cache côté serveur : Utilisation de Varnish, Redis ou caches en mémoire.
107. Outils de surveillance : Utilisation de Prometheus, Grafana et New Relic pour la surveillance du serveur.
108. Journalisation en production : Journalisation centralisée avec la pile ELK ou Graylog.
109. Mise à l'échelle horizontale vs verticale : Compréhension des compromis et des cas d'utilisation.
110. Optimisation des performances du serveur web : Ajustement des threads de travail, des délais de connexion et des tampons.
111. Mise en cache du proxy inverse : Configuration des en-têtes de cache et d'expiration.
112. Tests de charge du serveur web : Outils comme Apache JMeter, Gatling pour les tests de performance.
113. Déchargement SSL : Gestion de la terminaison SSL/TLS au niveau de l'équilibreur de charge.
114. Durcissement du serveur web : Meilleures pratiques de sécurité et évaluations des vulnérabilités.
115. Service de contenu dynamique vs statique : Optimisation des configurations de serveur.
116. Clustering de serveur web : Configuration de clusters pour une haute disponibilité.
117. Authentification du serveur web : Mise en œuvre de l'authentification de base, de la digestion et OAuth.
118. Formats de journalisation du serveur web : Formats de journalisation courants et outils d'analyse.
119. Limites de ressources du serveur web : Configuration des limites sur les connexions, les requêtes et la bande passante.
120. Sauvegarde et récupération du serveur web : Stratégies de récupération après sinistre.

CI/CD et DevOps (20 points)

121. Pipeline Jenkins en tant que code : Écriture de Jenkinsfiles pour les pipelines CI/CD.

122. Conteneurisation Docker : Création de Dockerfile, builds multi-étapes et orchestration de conteneurs.
123. Orchestration Kubernetes : Déploiements, services, pods et stratégies de mise à l'échelle.
124. Principes GitOps : Utilisation de Git pour la gestion de l'infrastructure et de la configuration.
125. Outils de build Maven et Gradle : Gestion des dépendances, plugins et cycle de vie de build.
126. Tests unitaires et d'intégration : Écriture de tests avec JUnit, Mockito et TestNG.
127. Outils de couverture de code : Utilisation de Jacoco pour mesurer la couverture de code.
128. Analyse statique du code : Outils comme SonarQube pour les vérifications de qualité de code.
129. Infrastructure as Code (IaC) : Utilisation de Terraform, CloudFormation pour le provisionnement de l'infrastructure.
130. Déploiements Blue/Green : Minimisation des temps d'arrêt pendant les déploiements.
131. Déploiements Canary : Déploiement progressif de nouvelles fonctionnalités.
132. Tests automatisés dans les pipelines CI : Intégration des tests avec les étapes de build.
133. Gestion des environnements : Utilisation d'Ansible, Chef ou Puppet pour la gestion de la configuration.
134. Meilleures pratiques CI/CD : Intégration continue, déploiement continu et livraison continue.
135. Stratégies de retour en arrière : Mise en œuvre de retours en arrière automatisés en cas d'échec de déploiement.
136. Scans de sécurité : Incorporation de vérifications de sécurité comme SAST, DAST dans les pipelines.
137. Pipelines CI/CD pour microservices : Gestion des pipelines pour plusieurs services.
138. Surveillance des pipelines CI/CD : Alertes sur les échecs de pipeline et les problèmes de performance.
139. Écosystème des outils DevOps : Compréhension des outils comme Docker, Kubernetes, Jenkins, Ansible.
140. CI/CD pour les applications cloud-native : Déploiement d'applications sur des plateformes cloud.

Modèles de conception et meilleures pratiques (20 points)

141. Modèle Singleton : Mise en œuvre de singletons thread-safe.
142. Modèle Factory : Création d'objets sans spécifier la classe exacte.
143. Modèle Strategy : Encapsulation des algorithmes et basculement entre eux.
144. Principes SOLID : Compréhension et application des principes de responsabilité unique, ouvert/fermé, substitution de Liskov, ségrégation de l'interface, inversion des dépendances.
145. Injection de dépendances : Réduction du couplage et augmentation de la maintenabilité du code.

146. Modèle de sourcing des événements : Stockage des événements pour reconstruire l'état de l'application.
147. Architecture CQRS : Séparation des responsabilités de commande et de requête.
148. Conception pour la scalabilité : Utilisation de la mise à l'échelle horizontale, du partitionnement et de l'équilibrage de charge.
149. Techniques de refactorisation du code : Extraction de méthodes, renommage de variables et simplification des conditionnels.
150. Pratiques de code propre : Écriture de code lisible, maintenable et auto-documenté.
151. Développement piloté par les tests (TDD) : Écriture de tests avant l'implémentation.
152. Versioning du code : Utilisation des stratégies de branchement Git comme GitFlow, Trunk-Based Development.
153. Conception pour la maintenabilité : Utilisation de la conception modulaire, séparation des préoccupations.
154. Anti-modèles à éviter : Classes Dieu, code spaghetti et couplage étroit.
155. Conception pour la sécurité : Mise en œuvre du principe du moindre privilège, défense en profondeur.
156. Conception pour les performances : Optimisation des algorithmes, réduction des opérations d'E/S.
157. Conception pour la fiabilité : Mise en œuvre de la redondance, de la tolérance aux pannes et de la gestion des erreurs.
158. Conception pour l'extensibilité : Utilisation de plugins, extensions et API ouvertes.
159. Conception pour l'ergonomie : Assurer que les API sont intuitives et bien documentées.
160. Conception pour la testabilité : Écriture de code facile à tester et à simuler.

Sécurité (20 points)

161. OAuth2 et JWT : Mise en œuvre de l'authentification basée sur les jetons.
162. Contrôle d'accès basé sur les rôles (RBAC) : Attribution de rôles et de permissions aux utilisateurs.
163. En-têtes de sécurité : Mise en œuvre de la politique de sécurité de contenu, X-Frame-Options.
164. Prévention des injections SQL : Utilisation des instructions préparées et des requêtes paramétrées.
165. Protection contre les scripts intersites (XSS) : Sanitisation des entrées et sorties.
166. Chiffrement et déchiffrement : Utilisation d'AES, RSA pour la protection des données.
167. Pratiques de codage sécurisé : Éviter les vulnérabilités courantes comme les dépassements de tampon.

168. Mise en œuvre des journaux d'audit : Journalisation des actions des utilisateurs et des événements système.
169. Gestion des données sensibles : Stockage sécurisé des mots de passe avec des algorithmes de hachage.
170. Conformité aux réglementations : GDPR, PCI-DSS et lois de protection des données.
171. Mise en œuvre de l'authentification à deux facteurs (2FA) : Ajout d'une couche de sécurité supplémentaire.
172. Tests de sécurité : Tests d'intrusion, évaluations des vulnérabilités.
173. Protocoles de communication sécurisés : Mise en œuvre de SSL/TLS pour le chiffrement des données.
174. Gestion sécurisée des sessions : Gestion des jetons de session et des délais d'expiration.
175. Mise en œuvre des pare-feu d'applications web (WAF) : Protection contre les attaques courantes.
176. Surveillance et alertes de sécurité : Utilisation d'outils comme SIEM pour la détection des menaces.
177. Meilleures pratiques de sécurité dans les microservices : Sécurisation de la communication inter-services.
178. Mise en œuvre de CAPTCHA pour la protection contre les bots : Prévention des attaques automatisées.
179. Sécurité dans les pipelines CI/CD : Scans de vulnérabilités pendant les builds.
180. Mise en œuvre de la sécurité par conception : Incorporation de la sécurité dès le début du processus de développement.

Optimisation des performances et optimisation (20 points)

181. Profilage des applications Java : Utilisation d'outils comme JProfiler, VisualVM pour l'analyse des performances.
182. Optimisation de la collecte des ordures : Ajustement des paramètres de GC pour les performances.
183. Optimisation des requêtes de base de données : Indexation, réécriture des requêtes et utilisation des plans d'explication.
184. Stratégies de mise en cache : Utilisation de caches distribués, mécanismes d'invalidation du cache.
185. Tests de charge et de contrainte : Identification des goulots d'étranglement des performances.
186. Optimisation des API RESTful : Réduction des temps de réponse, minimisation du transfert de données.
187. Réduction de la latence réseau : Utilisation de CDN, optimisation des appels API.
188. Dimensionnement du pool de connexions : Détermination des tailles optimales de pool pour les bases de données et les connexions.

189. Configurations de surveillance et d'alerte : Utilisation de Prometheus, Grafana pour la surveillance en temps réel.
190. Identification et résolution des goulots d'étranglement : Profilage de l'utilisation du CPU, de la mémoire et de l'E/S.
191. Optimisation des paramètres de heap Java : Définition des tailles de heap appropriées pour différents environnements.
192. Réduction des pauses de collecte des ordures : Utilisation de G1GC, ZGC pour les applications à faible latence.
193. Optimisation de l'E/S disque : Utilisation de SSDs, configurations RAID et optimisations du système de fichiers.
194. Mise en cache vs stockage : Décision de mettre en cache les données ou de les stocker dans une base de données.
195. Optimisation de la journalisation : Réduction de la surcharge de journalisation et gestion des volumes de journaux.
196. Optimisation de l'accès concurrent : Utilisation efficace des verrous et minimisation de la contention.
197. Profilage de l'utilisation de la mémoire : Identification des fuites de mémoire et optimisation des allocations d'objets.
198. Optimisation de la taille des pools de threads : Équilibrage entre trop peu et trop de threads.
199. Optimisation des structures de données : Choix des structures de données appropriées pour des cas d'utilisation spécifiques.
200. Indicateurs clés de performance et KPI : Définition et suivi des indicateurs clés de performance pour les applications.