1. -XX:+UseZGC  
  
Catégorie : Garbage Collector  
  
Description : Le Z Garbage Collector (ZGC) est optimisé pour une ultra-faible latence, maintenant   
les pauses de collecte sous 10 ms, même avec de grandes tailles de mémoire (jusqu’à plusieurs   
téraoctets).  
  
Justification : Pour une instance de GraphHopper qui gère de nombreuses requêtes de routage et de grandes cartes (par exemple, des pays entiers), le ZGC minimise la latence et permet une expérience utilisateur fluide en éliminant les longues pauses de collecte de mémoire. Ceci est particulièrement utile pour des applications nécessitant des temps de réponse en temps réel sur de grands ensembles de données.  
  
2. -Xmx4G  
  
Catégorie : Taille du monceau  
  
Description : Définit la taille maximale de la mémoire du monceau que la JVM peut utiliser. Cela   
garantit que l'application dispose de suffisamment de mémoire pour stocker de grands ensembles de   
données graphiques et traiter les requêtes de routage.  
  
Justification : Comme GraphHopper fonctionne avec de grands ensembles de données (par exemple, des   
cartes de pays ou de continents), spécifier une taille de tas généreuse aide à éviter les problèmes   
de mémoire lors du chargement des graphes et des calculs d'itinéraires. Par exemple, -Xmx4G alloue   
jusqu'à 4 Go de mémoire pour le monceau, ce qui peut prendre en charge des cartes plus grandes et   
plus de requêtes de routage en parallèle.  
  
3. -XX:+UseStringCache  
  
Catégorie : Gestion des strings  
  
Description : Active un mécanisme de mise en cache pour les chaînes courtes, en se concentrant   
particulièrement sur la réutilisation des chaînes de petite taille fréquemment utilisées dans   
l’application. Ce cache peut réduire l’utilisation de la mémoire en évitant des allocations   
redondantes de chaînes.  
  
Justification : GraphHopper traite souvent des chaînes courtes et répétées, comme des noms de rues,   
des identifiants de lieux, et des marqueurs d’itinéraires. Activer -XX:+UseStringCache réduit la   
surcharge de mémoire en réutilisant les instances de ces chaînes courtes. Ce flag aide à optimiser   
l’utilisation de la mémoire lors de la gestion de grands ensembles de données, ce qui est   
particulièrement utile dans les applications où des chaînes courtes apparaissent fréquemment,   
améliorant ainsi l’efficacité mémoire et les performances en réduisant la pression sur le   
garbage collector.  
  
4. -XX:+UseCompressedOops  
  
Catégorie : Compression des pointeurs d'objets ordinaires  
  
Description : Active la compression des pointeurs d'objets ordinaires (OOPs), ce qui réduit   
l'utilisation de la mémoire en compressant les pointeurs 64 bits en 32 bits, utile pour les tas de   
taille modérée.  
  
Justification : Avec GraphHopper, de nombreux objets (par exemple, des nœuds, des arêtes) occupent   
de la mémoire, et -XX:+UseCompressedOops permet une utilisation efficace de la mémoire en   
compressant les références. Ce flag aide à gérer l’empreinte mémoire de GraphHopper sans   
compromettre la capacité du tas, ce qui le rend particulièrement utile pour les applications   
intensives en graphes sur des JVM 64 bits.  
  
5. -Dfile.encoding=UTF-8  
  
Description : Définit l'encodage de fichier par défaut sur UTF-8, garantissant que la JVM lit et   
écrit les fichiers en utilisant cet encodage.  
  
Justification : GraphHopper gère divers types de données géographiques, qui peuvent inclure des   
caractères non ASCII (par exemple, des noms de rues dans différentes langues). Ce flag assure une   
gestion correcte de ces données, évitant ainsi d'éventuels problèmes d'encodage avec des cartes ou   
des noms de lieux internationalisés.