Pesquisa Sobre Modelos de Gerenciamento de Memória

Aluna: Larissa da Silva Matos
Matrícula: 494223

Escolha duas linguagens além da sua linguagem de estudo e tente comparar como funciona o processo de alocação e desalocação de memória.

As duas linguagens de programação escolhidas para essa pesquisa são: Java e JavaScript. A motivação para tal escolha foi a pouca experiência em ambas e a semelhança nos nomes, que pode causar confusão.

JAVA:

De acordo com Schildt (2019, p. 12), a linguagem de programação orientada a objetos Java é considerada moderna e robusta, uma das razões para isso é sua maneira de lidar com erros de gerenciamento de memória. Pois, diferentemente de outras linguagens onde é preciso alocar e desalocar toda a memória, Em Java, a desalocação de memória é realizada de forma automática pelo *garbage collector*, que é responsável por identificar e remover objetos que não estão mais em uso, baseando-se em um contador de referências. Ainda no escopo desse processo automático, a linguagem elimina virtualmente alguns problemas que envolvem gerenciamento de memória, como aqueles causados por programadores que esquecem de liberar a memória que foi previamente alocada ou quando tentam liberar memória que ainda está sendo utilizada.

Para criar um novo objeto, usa-se o operador new, que aloca memória dinamicamente para um objeto durante o tempo de execução; essa é a maneira mais comum e adequada de criar um novo objeto. Entretanto, existem outras maneiras de criar objetos em Java que são consideradas menos eficientes e podem deixar o programa mais lento.

Um deles é o uso do método Class.newInstance(), que cria um novo objeto da classe representada pelo objeto Class, essa forma de criação é chamada de “instanciar objetos através de reflexão”, mas seu uso é obsoleto. Outra maneira é utilizando Clone(), que é um método que cria uma cópia exata ao objeto escolhido, podendo ser utilizado na forma de *shallow copy* onde não é feita a cópia dos objetos referenciados pelo objeto original e *deep copy,* que faz essa função, ou seja, todos os níveis do objeto são copiados. Por fim, existe a deserialização, que lê arquivos recebidos principalmente por rede (em bytes) através da classe ObjectInputStream(), e cria um objeto Java a partir desses dados, mas nesse caso ainda se usa o new.

Todavia, não é possível desalocar manualmente ou forçar o *garbage collector* a entrar em ação, mas é possível fazer a solicitação do mesmo usando System.gc() ao JVM (Java

Virtual Machine). Contudo, é uma boa prática utilizar o método Close() para liberar recursos externos, como arquivos, sockets, conexões de banco de dados, entre outros.

JAVASCRIPT:

O JavaScript assemelha-se ao Java em relação ao gerenciamento de memória, pois também possui um *garbage collector* para liberar memória que não está mais sendo utilizada. Contudo vale lembrar que, isso não significa que o programador não precisa tomar cuidados com a memória na hora de programar.

De forma geral, o *garbage collector* funciona com base na contagem de referência em segundo plano e é a única maneira para desalocar memória. Entretanto, existem outras maneiras para tentar fazer a desalocação acontecer: utilizando o método delete que pode remover propriedades de objetos e objetos em si; e fazendo com que os ponteiros apontem para *null*. Contudo, nenhuma dessas duas últimas formas garante que a memória realmente será desalocada.

Para criar um objeto em JavaScript existe mais de uma forma. A mais comum é definir o tipo de objeto utilizando um construtor, e em seguida criar uma instância do objeto com new. Outra forma é usar o método Object.create() que cria um objeto a partir do protótipo de um objeto já existente. Além desses, é possível simplesmente criar uma lista separada por vírgulas de zero ou mais pares de nomes de propriedades e valores associados a um objeto envolvidos por chaves.

De acordo com a documentação do JavaScript na MDN (Mozilla Developer Network) [1], o ciclo de vida da memória é basicamente sempre o mesmo:

- Alocar a memória desejada;
- Usar a memória alocada (*read, write*);
- Liberar a memória que não será mais utilizada.

Dentre todas essas partes, a primeira e a última geralmente são implícitas em linguagem de alto nível, como JavaScript.

Como dito anteriormente, o sistema automático de desalocação de memória não abstém o programador de se preocupar com memória, por exemplo, em casos de referências circulares é muito comum ter vazamento de memória e isso prejudica o código. É importante ressaltar que esse tipo de situação pode ocorrer em outras linguagens de alto nível também, como Java.

REFERÊNCIAS

[1]MOZILLA. Template literals. MDN Web Docs, 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Acesso em: 22 abr. 2023.

[2]SCHILDT, Herbert. Java: The Complete Reference. 11th ed. New York: McGraw Hill Education, 2018.

[3]W3SCHOOLS. Create an Object Without Using New Operator in Java. In: W3Schools Blog. Disponível em: https://www.w3schools.blog/create-an-object-without-using-new-operator-in-java. Acesso em: 22 abr. 2023.