Erciyes Üniversitesi

Fen Bilimleri Enstitüsü Dergisi

Cilt 34, Sayı 2 , 2018

Erciyes University

Journal of Institue Of Science and Technology

Volume 34, Issue 2, 2018

|  |
| --- |
|  |
| **The Effect of Data and Object Types on Java Virtual Machine**Abdullah Talha KabakusDepartment of Computer Engineering, Faculty of Engineering, Duzce University, DUZCE(Alınış / Received: 20.03.2018, Kabul / Accepted: 30.08.2018, Online Yayınlanma / Published Online: 31.08.2018) |
|  |
|  |  |
| **Keywords**Java, Java virtual machine, variable type,data storage | **Abstract:** How the data is stored in the memory becomes more critical when the size of data increases. The programming languages define data and object types that can be used while programming. Most programming languages provide more than one data and object type in order to let developers use more sensitive types which address their needs. Memory management is a key concept for the data-intensive systems. Also, the NoSQL databases, which are alternatives to relational database management systems, tend to store the data in memory and serve the data from memory. In this study, the effect of data and object types on Java Virtual Machine is evaluated in order to reveal its effect in terms of consumed memory on Java programming language. Experimental results reveal some key points for developers to use memory more efficiently. |
|  |  |
|  |  |
| **Veri ve Nesne Türlerinin Java Sanal Makinasına Olan Etkisi**  |
|  |
|  |
| **Anahtar Kelimeler**Java, Java sanal makinası, değişken tipi, veri depolama | **Öz:** Verinin nasıl depolandığı verinin boyutu arttıkça daha kritik hale gelmektedir. Programlama dilleri, program geliştirirken kullanılabilecek veri ve nesne tiplerinin tanımlamaktadırlar. Çoğu programlama dili birden çok veri ve nesne tipi sağlayarak geliştiricilerin ihtiyaçlarını karşılayacak daha hassas tipler sağlamaktadır. Bellek yönetimi veri-odaklı sistemler için anahtar bir kavramdır. Ayrıca ilişkisel veritabanı yönetim sistemlerine alternatif olan NoSQL veritabanları, verilerin bellekte depolanıp, bellekten servis edilmesine yönelik eğilim göstermektedirler. Bu çalışmada, veri ve nesne tiplerinin Java sanal makinasına olan etkisi değerlendirilerek Java programlama diline olan bellek tüketim etkisi ortaya çıkarılmıştır. Deneysel sonuçlar, belleğin daha efektif bir şekilde kullanılabilmesine yönelik geliştiricilere anahtar bilgiler sunmaktadır. |
|  |  |

**1. Introduction**

Memory management becomes more critical with the rise of NoSQL (Not-only-SQL) databases which do not only serve the data from memory, also store the data on the memory too. Usage of too much memory may lead to poor cache utilization, and eventually, need to use of disk instead of memory which is expensive in terms of the required time to access and manipulate data [1]. Java virtual machine (JVM), which is actually a stack machine that executes the compiled Java programs (also known as bytecodes), assures the security, integrity, and portability of Java applications [2], [3]. In order to free objects which are stored in memory and are no longer referenced by the program that is executed on JVM a process named garbage collector is used by Java. This process is automatically handled by JVM in order to relieve programmers of the burden of having to keep track of when to free allocated memory by detecting objects that are no longer required [2], [3]. Java provides various primitive types and in-built reference types to enrich the way of data definition. Despite that the same data can be defined in various ways, the performance in terms of consumed memory and provided methods vary through the selection of (1) the data type of elements of array/list, and (2) the object type of list. Therefore, in this study, how the data and object types of array/lists which are provided by Java programming language affect memory consumption is experimented in order to give an idea to programmers to be away from the memory overhead which is a common issue for the garbage collector [4], [5]. The less consumption of memory means the less requirement of garbage collection which is slow and expensive [6], and even sometimes it is the performance bottleneck [7] as historical data shows that garbage collection may even occupy 20% or more of an application’s total running time [8]. This paper deals with the way of efficient memory consumption in terms of data storage to be away from this slow and expensive garbage collection. The rest of the paper is structured as follows: Section 2 describes the experiments and the method used to evaluate them. Section 3 presents the experimental result and discussion. Finally, Section 4 concludes the paper with findings.

**2. Material and Method**

It is possible to use various data types to define the same variable on memory as the primitive data types that Java programming language provides are listed in Table 1. Each data type has its own characteristic in terms of how much memory it consumes, what kind of data it represents. Even though the data is stored in the persistent storage as the relational database management systems do, the way it is stored is still critical since it is being served from memory. When the size of data gets bigger, it becomes more critical to consume as less memory as possible. Two experiments are evaluated to reveal the effect of data and object types on memory in terms of consumed memory: (1) The effect of data type, and (2) the effect of the object type.

**Table 1**. The primitive data types that are defined by Java programming language

|  |  |  |
| --- | --- | --- |
| Data Type | Size in Bytes | Value Range |
| int | 4 | [-2,147,483,648, 2,147,483, 647] |
| long | 8 | [-9,223,372,036,854,775,808, 9,223,372,036,854,775,807] |
| float | 4 | approximately ±3.40282347E+38F |
| boolean | Not precisely defined[[1]](#footnote-1) | true or false |
| short | 2 | [-32,768, 32,767] |
| double | 8 | approximately ±1.79769313486231570E+308 |
| byte | 1 | [-128, 127] |
| char | 2 | [0, 65,536] |

Each primitive data type has an object equivalent in Java which extends the base java.lang.Object[[2]](#footnote-2) class which is the base class of all classes in Java.

**2.1. Experiment #1 – The effect of data type**

500 thousand (500K), 5 million (5M), and 10 million (10M) variables are stored respectively in the arrays of primitive and reference types of each data type in order to reveal how the size and type of data affect the memory consumption. The memory consumption is calculated using the methods provided by java.lang.Runtime[[3]](#footnote-3) class. The consumed memory is calculated just before and after storing the data. The difference is set as the size of the consumed memory to store the data. The experimental result is listed in Table 2.

**Table 2**. The effect of data type on memory consumption

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Consumed Memory for 500K items (MB) | Consumed Memory for 5M items (MB) | Consumed Memory for 10M items (MB) |
| int[] | 2 | 19 | 38 |
| Integer[] | 10 | 93 | 190 |
| long[] | 4 | 39 | 77 |
| Long[] | 13 | 134 | 265 |
| float[] | 3 | 19 | 38 |
| Float[] | 10 | 94 | 190 |
| boolean[] | 0 | 4 | 10 |
| Boolean[] | 2 | 23 | 38 |
| short[] | 1 | 9 | 19 |
| Short[] | 2 | 19 | 38 |
| double[] | 3 | 38 | 76 |
| Double[] | 12 | 131 | 266 |
| byte[] | 0 | 8 | 9 |
| Byte[] | 2 | 19 | 38 |
| char[] | 1 | 9 | 19 |
| Character[] | 2 | 19 | 41 |

**2.2. Experiment #2 – The effect of list type on memory**

When the type of data is defined, then it is critical to determine which list type is used to store the data. The object type lists that are evaluated in this study are LinkedList and ArrayList. LinkedList is implemented as a doubly-linked list [9], [10]. Alongside these object type lists, the array of the object is also included in experiments in order to compare memory consumptions of different list implementations as the experimental result is listed in Table 3. The method which is used to calculate the consumed memory for the experiment #1 is also used for the experiment #2 in order to ensure consistency.

**Table 3**. The effect of data type on memory consumption

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | List Type | Consumed Memory for 500K items (MB) | Consumed Memory for 5M items (MB) | Consumed Memory for 10M items (MB) |
| Integer | LinkedList | 12 | 116 | 228 |
| ArrayList | 2 | 19 | 38 |
| Array | 10 | 93 | 190 |
| Double | LinkedList | 23 | 227 | 456 |
| ArrayList | 2 | 19 | 38 |
| Array | 12 | 131 | 266 |
| Float | LinkedList | 19 | 190 | 382 |
| ArrayList | 2 | 19 | 38 |
| Array | 10 | 94 | 190 |
| Long | LinkedList | 12 | 116 | 228 |
| ArrayList | 2 | 19 | 38 |
| Array | 13 | 134 | 265 |
| Short, Character, Byte, Boolean | LinkedList | 19 | 190 | 228 |
| ArrayList | 2 | 19 | 38 |
| Array | 2 | 20 | 38 |

**3. Results and Discussion**

As the experimental result is listed in Table 2, the reference type of arrays consumes much more than the primitive type arrays. The main reason behind this result is that reference type variables are complex in terms of memory allocation and management compared to primitive type variables since they are extended from Object and when they are stored in a variable or passed to a method, a reference is used to access them [11]. According to the result of experiment #2 which is listed in Table 3, while ArrayList consumes the least memory, LinkedList consumes the memory most. Since the LinkedList is the double-linked implementation of the list interface and provides reserve iterator, it is more complex compared to ArrayList. The advantage of LinkedList over ArrayList is that the insertion and deletion are less expensive in terms of data manipulation. In order to avoid memory overhead, the experimental results clearly indicate that programmers should prefer (1) the primitive types over the reference types, and (2) the array over the object type lists. When these are not possible due to the necessity of the reference and object type lists, the ArrayList should be preferred instead of LinkedList unless clear need for data insertion and deletion. Another finding is that the object types Short, Character, Byte, and Boolean consume the same amount of memory when they are used within arrays and lists.

**4. Conclusion**

The type of data and the way it is stored in memory are critical for the data-intensive systems as they determine the memory overhead and the processing load. Both various data and list types are interchangeable for the Java programming language. For this reason, the performance of both data and list types in terms of consumed memory are evaluated using two experiments. The experimental result reveals some key points for the programmers in order to avoid memory overhead and need of garbage collection which is slow and expensive. As a future work, the reasons behind these findings may be explained after examining the way Java virtual machine handles data types and reference type lists.

**Declaration of Conflicting Interests**

The author declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author received no financial support for the research, authorship, and/or publication of this article.

**References**

1. Sestoft, P. 2005. Java performance Reducing time and space consumption. Royal Veterinary and Agricultural University and IT University of Copenhagen.
2. Cramer, T., Friedman, R., Miller, T., Seberger, D., Wilson, R., Wolczko, M. 1997. Compiling Java just in time. IEEE Micro, 17(3), 36-43.
3. Venners, B. 1996. Java’s garbage-collected heap. JavaWorld.
4. Ritzau, T, Fritzson, P. 2002. Decreasing memory overhead in hard real-time garbage collection. Second International Conference (EMSOFT 2002), October 7-9, Grenoble, France, 213-226.
5. Prechelt, L. 2000. An empirical comparison of seven programming languages. IEEE Computer, 33(10), 23-29.
6. Appel, A. W. 1987. Garbage collection can be faster than stack allocation. Information Processing Letters, 25(4): 275-279.
7. Zhang, X., Seltzer, M. 2001. HBench: JGC-An Application-Specific Benchmark Suite for Evaluating JVM Garbage Collector Performance. 6th USENIX Conference on Object-Oriented Technologies and Systems, January 29 – February 2, San Antonio, Texas, USA.
8. Larose, M., Feeley, M. 1998. A Compacting Incremental Collector and its Performance in a Production Quality Compiler. 1st International Symposium on Memory Management, October 17-19, Vancouver, BC, Canada, 1-9.
9. González P. A. 2000. Applying knowledge modelling and case-based reasoning to software reuse. IEE Proceedings – Software, 147(5), 169-177.
10. Mittal, B. 2015. 5 Difference between ArrayList and LinkedList in Java with Example. <http://javahungry.blogspot.com/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html> (Access Date: 09.04.2018).
11. Flanagan, D. 2002. Java In A Nutshell. 2nd edition. O’Reilly Media. California, 945s.
1. boolean represents a bit of information, its size is not something that is precisely defined. [↑](#footnote-ref-1)
2. https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html [↑](#footnote-ref-2)
3. https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html [↑](#footnote-ref-3)