Java Performansı

Online Seminer

Akın Kaldıroğlu

- Java gerçekten yavaş mı? Yoksa developerlar mı yavaş?
- Dil mi yavaştır yoksa uygulama mı yavaştır?
- C/C++ kadar hızlı çalışan Java kodu yazılabilir mi?
- Yüksek performanslı Java kodu yazmak için nelere dikkat etmek gereklidir?
- Java ile geliştirilen uygulamalarda sıklıkla görülen performans problemleri ve sebepleri nelerdir?
- Java ile geliştirilen uygulamalarda sıklıkla görülen performans problemlerinden nasıl kaçınılır ve nasıl çözülür?

Küçük Ama Önemli Bir Konu

- Bu dosya ve beraberindeki tüm, dosya, kod, vb. eğitim malzemelerinin tüm hakları **Selsoft Yazılım, Danışmanlık, Eğitim ve Tic. Ltd. Şti**.'ne aittir.
- Bu eğitim malzemelerini kişisel bilgilenme ve gelişiminiz amacıyla kullanabilirsiniz ve isteyenleri **http://www.selsoft.academy** adresine yönlendirip, bu malzemelerin en güncel hallerini almalarını sağlayabilirsiniz.
- Yukarıda bahsedilen amaç dışında, bu eğitim malzemelerinin, ticari olsun/olmasın herhangi bir şekilde, toplu bir eğitim faaliyetinde kullanılması, bu amaca yönelik olsun/olmasın basılması, dağıtılması, gerçek ya da sanal/Internet ortamlarında yayınlanması yasaktır. Böyle bir ihtiyaç halinde lütfen benimle, **akin.kaldiroglu@selsoft.academy** adresinden iletişime geçin.
- Bu ve benzeri eğitim malzemelerine katkıda bulunmak ya da düzeltme ve eleştirilerinizi bana iletmek isterseniz çok sevinirim.
- Bol Java'lı günler dilerim.

Java Yavaş mı?

- Java, sıklıkla “yavaş” olarak nitelenen bir dil.

- Bunu iddia edenlerin temelde iyi dayanağı var:
  - Java native çalışmadığı için yavaş olması da zaten normaldir, beklenendir.
    - Bu yüzden Java hiç bir zaman C/C++’ın hızına erişemez.

  - Biz Java ile uygulama geliştirdik ve yavaş çalıştı/çalışıyor.

Tanımlar

Yavaş ya da Hızlı Olmak Nedir?

- Uygulamalarda farklı şeylerin performanslarından bahsedebiliriz:
  - Ayağa kalkma zamanı (startup time),
  - Harcanan bellek (memory footprint),
  - Cevap verme süresi (response time, responsiveness)
  - İş üretme (throughput),
  - Ölçeklenirlik (scalability)

Yavaş ya da Hızlı Olan Nedir?

- Hangisinin yavaş ya da hızlı olması, bir uygulamanın yavaş ya da hızlı olmasını daha çok belirler?
  - Dil
  - Yazılımın mimarisi
  - Üzerinde çalışılan donanım

- Yazılım mimarisi, uygulamaların performanslarını belirlemede en fazla etkiye sahip olandır.
- Bu amaçla facebook gibi yüksek ölçeklenirlik gerektiren uygulamaların mimarilerini inceleyebilirsiniz.
- Ayrıca http://www.javaturk.org/tag/java-performans/ adresindeki yazıma bakabilirsiniz.

Performans Çalışmaları - I

- Performans ile ilgili çalışmalar genelde 3 başlık altında ele alınır:
  - **Monitoring** (gözlemleme): Sistemin davranışını, ona müdahele etmeden ve yük getirmeyecek şekilde, gözlemlemektir.
    - Sistemin tamamı için kullanılır.
    - Daha çok önleyici amaçlar için kullanılır.
  - **Profiling** (tarama): Az ya da çok müdahele ederek ve sistemin davranışı ile ilgili veri toplamaktır.
    - Muhtemelen sıkıntılı olduğu düşünülen taraflar için yapılır.
    - Problem olduğunda başvurulur.
    - Sampling (örnekleme) şeklinde daha az müdahele eden şekli de vardır.

Performans Çalışmaları - II

- **Tuning** (ince ayar): Performansı iyileştirmek amacıyla, kodda, ortamda vs. yapılan ince ayar çalışmalarıdır.
  - Genelde sistemin bütününden ziyade belli alanlara odaklıdır:
    - Uygulamanın veri tabanı iletişimi ve sorguların (SQL) iyileştirilmesi,
    - Kullanıcı arayüzlerinin inceltilmesi,
    - Sistemin ayağa kalkarken yaptığı işlerin azaltılması,
    - Sistemin XML işlemesinin iyileştirilmesi,

Tuning - I

- Tuning farklı yerlere ve şekillerde uygulanabilir:
  - **Bazen sadece algoritmiktir**: Kullanılan algoritma iyileştirilir ya da değiştirilir.
  - **Bazen belli kod parçalarına (aspect) uygulanır**: Oturuma konan nesnelerin azaltılması (örneğin *HttpSession* temizlemesi), veri tabanı iletişimi için *Statement* yerine *PreparedStatement* kullanılması, loglama, sorgulama (querying) ya da transaction performansının iyileştirilmesi.

Tuning - II

- **Bazen teknolojiye uygulanır**: JSF 1.2'yi JSF 2 ile değiştirmek gibi.
- **Bazen yazılımın mimarisine uygulanır**: Cache kullanımına geçilmesi, nesne havuzu (object pool) kullanılması ya da yazılımın topolojisinde değişiklik yapılması gibi.
- **Bazan veriye uygulanır**: Veri tabanındaki verinin denormalize edilmesi ya da veri tabanlarının sağladığı optimizasyon imkanlarının kullanılması.

Tuning - III

- **Bazan donanıma uygulanır**: Daha güçlü donanıma geçilmesi gibi.

- **Bazan JVM'e uygulanır**: Java'ya özel olarak JVM'in davranışını iyileştirmek amacıyla parametreler kullanılır. Bu parametrelerin yüksek performans sağlaması amacıyla optimum noktaya getirilmesidir.

Ne Zaman Tuning?

- Tuning yapmadan önce, neyin problemli olduğu ve ne şekilde iyileştirilmesi gerektiği kesinleştirilmelidir:

Premature optimization is the root of all evil – D. Knuth

- Ayrıca atılan taş, ürkütülen kurbağaya değmelidir.

Java Performansı

Java Yavaş mıdır?

- Java dil olarak yavaş değildir. Çünkü,
  - Java hemen hemen native çalışır,
  - Java'nın yorumlanan (interpreted) yapısı, pek çok çalışma zamanı (run-time) iyileştirmesine imkan verir.

- Algoritmik performansı göz önüne alırsak, Java'nın C/C++'tan daha yavaş olduğunu söylemek zordur.

  - Bu anlamda Java'nın daha yavaş ya da daha hızlı olduğu durumsaldır.
    - Kullanılan algoritmaya da bağlıdır.

Örnekler

- Aynı algoritmayı C++ ve Java'da tamamen aynı şekilde yazarak, algoritmik açıdan bu iki dilin performansları karşılaştırılabilir.
- Bu türden karşılaştırmalar uygulama performansı ile ilgili bir şey söylemedikleri gibi sadece algoritmik dil performansı hakkında fikir vermeye yöneliktir.
- İki örnek:
  - Asal sayıları bulmak için Sieve of Atkin, http://www.sanfoundry.com/java-program-sieve-of-atkin-algorithm/
  - Monte carlo simulasyonu ile PI'ye yakınsamak.

MonteCarloPI.cpp

```
#include <iostream>
#include <stdlib.h>
#include <ctime>

namespace MonteCarloPI {
    using namespace std;
    int n;
    int main() {
        cout.precision(10);
        int dotsInCircle = 0;
        cout << "Number of points: " << endl;
        cin >> n;

        clock_t start = clock();

        for (int i = 0; i < n; i++) {
            double x = static_cast<double> (rand()) / static_cast<double> (RAND_MAX);
            double y = static_cast<double> (rand()) / static_cast<double> (RAND_MAX);
            double distance = x * x + y * y;
            if (distance <= 1.0)
                dotsInCircle++;
        }

        clock_t end = clock();
        int time = (int) (1000 * (end - start) / CLOCKS_PER_SEC);
        double myPi = (double) 4 * dotsInCircle / n;
        cout << "My PI is: " << myPi << endl;
        cout << "Time is: " << time << " ms." << endl;
        return 0;
    }
}
```

MonteCarloPI.java

```
package org.javaturk.performance.algorithm.pi;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import org.apache.commons.math.random.BitsStreamGenerator;
import org.apache.commons.math.random.MersenneTwister;

/**
 * Monte Carlo simulation to get closer to PI. It uses MersenneTwister to  produce random numbers.
 * The implementation of this algorithm comes with Apache's commons Math library.
 */
public class MonteCarloPI2 {
    private static long n;
    private static BitsStreamGenerator randomData = new MersenneTwister();

    public static void main(String[] args) {
        System.out.println("*** MonteCarloPI2 ***");
        System.out.print("Number of points: ");
        Scanner in = new Scanner(System.in);
        n = in.nextLong();
        calculatePI();
    }

    public static void calculatePI() {
        int dotsInCircle = 0;
        double start = System.currentTimeMillis();
        for (int i = 0; i < n; i++) {
            double x = randomData.nextDouble();
            double y = randomData.nextDouble();
            double distance = x * x + y * y;
            if (distance <= 1)
                dotsInCircle++;
        }
        double finish = System.currentTimeMillis();
        double seconds = (finish-start);
        double myPI = (double) 4*dotsInCircle/n;
        System.out.println("My PI is: " + myPI + " and Java's PI is: " + Math.PI);
        System.out.println("Time is: " + seconds);
    }
}
```

SieveOfAtkin.cpp

```
#include "SieveOfAtkin.h"
#include <iostream>
#include <cmath>
#include <vector>
#include <ctime>

using namespace SieveOfAtkin;
using namespace std;

int main() {
    int n;
    cout << "Enter the number up to which
             prime numbers are counted: " << endl;
    cin >> n;
    clock_t start = clock();
    int numberOfPrimes = countPrimesUsingSieveOfAtkins(n);
    clock_t end = clock();
    int time = (int) (1000 * ((float) end - start) / CLOCKS_PER_SEC);

    cout << "Number of primes up to " << n << " : "
         << numberOfPrimes << endl;
    cout << "Time: " << time << " ms." << endl;
    cout << endl;
    return 0;
}
```

```
namespace SieveOfAtkin {
    using namespace std;

    int countPrimesUsingSieveOfAtkins(int n) {
        bool* is_prime = new bool[n + 1];
        is_prime[2] = true;
        is_prime[3] = true;

        for (int i = 5; i <= n; i++)
            is_prime[i] = false;

        int limit = ceil(sqrt(n));

        for (int x = 1; x <= limit; x++) {
            for (int y = 1; y <= limit; y++) {
                int num = (4 * x * x + y * y);
                if (num <= n && (num % 12 == 1 || num % 12 == 5))
                    is_prime[num] = true;

                num = (3 * x * x + y * y);
                if (num <= n && (num % 12 == 7))
                    is_prime[num] = true;

                num = 3 * x * x - y * y;
                if ((x > y) && (num <= n) && (num % 12 == 11))
                    is_prime[num] = true;
            }
        }

        for (int i = 5; i <= limit; i++)
            if (is_prime[i])
                for (int j = i * i; j <= n; j += i)
                    is_prime[j] = false;

        int numberOfPrimes = 2;
        for (int i = 5; i <= n; i++)
            if (is_prime[i])
                numberOfPrimes++;
        return numberOfPrimes;
    }
}
```

SieveOfAtkin.java

```
package org.javaturk.performance.algorithm.prime;
import java.util.Scanner;

public class SieveOfAtkin {
    public static void main(String[] args) {
        System.out.println("Please enter a
                  number to which to list prime numbers:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        long start = System.currentTimeMillis();
        int numberOfPrimes = countPrimesAsUsingSieveOfAtkins(n);
        long end = System.currentTimeMillis();
        long miliSeconds = end - start;
        System.out.println("There are " + numberOfPrimes +
                           " prime numbers up to " + n + ".");
        System.out.println("Time to find them: " +
                            miliSeconds + " ms.");
    }

    private static int countPrimesAsUsingSieveOfAtkins(int n) {
        boolean[] prime = new boolean[n + 1];
        prime[2] = true;    prime[3] = true;
        int limit = (int) Math.ceil(Math.sqrt(n));

        for (int x = 1; x <= limit; x++) {
            for (int y = 1; y <= limit; y++) {
                int num = 4 * x * x + y * y;
                if (num <= n && (num % 12 == 1 || num % 12 == 5))
                    prime[num] = true;
                num = 3 * x * x + y * y;
                if (num <= n && num % 12 == 7)
                    prime[num] = true;
                num = 3 * x * x - y * y;
                if ((x > y) && (num <= n) && (num % 12 == 11))
                    prime[num] = true;
            }
        }

        for (int i = 5; i <= limit; i++)
            if (prime[i])
                for (int j = i * i; j <= n; j += i)
                    prime[j] = false;

        int numberOfPrimes = 2;
        for (int i = 5; i <= n; i++)
            if (prime[i])
                numberOfPrimes++;

        return numberOfPrimes;
    }
}
```

Sonuçlar

- Farklı *n* girdisi için *mili saniye* cinsinden Java ve C++ programlarının çalışma süreleri şunlardır:

Program	10^6	10^7	10^8	10^9	2*10^9
MonteCarloPI.java	38	296	1,637	24,329	50,486
MonteCarloPI.cpp	21	214	2,404	33,898	81,899

Program	10^6	10^7	10^8	10^9	2*10^9
SieveOfAtkin.java	16	84	1,025	18,507	47,752
SieveOfAtkin.cpp	18	218	2,461	34,871	82,059

- Bu sonuçlar 16 GB RAM ve 8 çekirdekli i7 CPU'ya sahip, üzerinde El Capitan OS çalışan MacBook Pro üzerinde elde edilmiştir.
- Java için JDK 1.8.0_25 JVM, C++ için kullanılmıştır. Çalışma zamanı için tamamen varsayılan yapılar kullanılmış, hiç bir parametre verilmemiştir.
- Farklı iki Windows 10 makinada da benzer sonuçlar elde edilmiştir.

Yorum

- Bu sonuçlardan çıkabilecek tek şey olsa olsa "Java'nın, C++ kadar hızlı çalışabilir"dir.
  - Bu sonuçların uygulama performansını değil, algoritmik performansı gösterdiğini unutmayalım.

- Dolayısıyla "dil olarak Java yavaş" iddiası en azından bu iki algoritma açısından bakıldığında doğru değildir.
- Java'nın değerlerinin, n arttıkça C++'tan çok daha iyi hale gelmesi de bir tesadüf değildir.
  - Java'nın çalışma zamanı yapısı olan JIT'nin zamanla çok daha etkin çalıştığını göstermektedir.

Uygulama Performansı ve GC - I

- Önemli olan uygulama performansıdır.
- Java dilinde uygulama performansını etkileyen en temel faktör ise çoğu zaman Garbage Collector (GC) yani Çöp Toplayıcı'dır.
  - GC'nin görevi, nesneleri bellekte oluşturmak ve erişilemeyen nesneleri bellekten temizlemektir.
- Gelişi güzel yazılmış Java kodunda oluşturulan aşırı sayıdaki nesnenin GC tarafından toplanması, Java uygulamalarının performansı düşüren en temel etkendir.

Uygulama Performansı ve GC - II

- Dolayısıyla olabildiğince az nesne oluşturmak ya da oluşmasını sağlamak, GC'nin etkisini aza indirmenin en temel yoludur.

- Belleği etkin kullan, bu amaçla
  - Sadece gerektiği kadar nesne yaratmak ve
  - Nesneleri bellekte sadece ve sadece gerektiği kadar tutmak
  - Dolayısıyla nesneyi tekrar yaratmayı ya da getirmeyi, nesneyi bellekte tutmaya tercih etmek

olarak özetlenebilecek stratejiler, Java'nın uygulama performansını arttıracaktır.

En Basit En İyi Pratikler - I

1. String nesnelerin **new** ile değil de **""** ile oluştur.

2. String nesnelerini "**+**" ile değil *StringBuilder* (threadler varsa ya da *StringBuffer*) ile değiştir, ekleme vs. yap.

3. Primitiveleri wrapperlarına tercih et, wrapperları gerekmedikçe kullanma.

4. finalize metodundan uzak dur.

En Basit En İyi Pratikler - II

5. Veri yapılarının, torbaların ve algoritmaların karmaşıklıklarından haberdar ol. Örneğin ne zaman ArrayList ne zaman LinkedList kullanacağını bil.

6. Java torbalarının sık re-size etmelerine engel ol.

7. Nesneleri belekte uzun süre tutmak için concrete referans kullanma, soft ya da weak refereansları tercih et.

8. Eğer dosya işlemleri yapıyorsanız, özellikle sık okuma-yazma (mesela file copy gibi) muhakkak buffer kullan.

En Basit En İyi Pratikler - III

9. Thread ve lock mekanizmalarını etkin kullan.

Performans Problemlerinden Kaçınmak

Performans Problemlerinden Kaçınmak - I

- Java'da performans problemlerinden kaçınmak için dikkat edilmesi gereken şeyler şunlardır:
- Süreç seviyesinde:
  - Performans ihtiyaçlarını sorgulamak ve elde etmek.
  - Kurgulanan mimarinin istenilen performansı yüksek marjla sağlanıldığından, benchmarking ya da simulasyon vb. yöntemlerle emin olmak.
  - Birim testlerini yapmak ve yapmaya devam etmek,
  - Fonksiyonel testleri yapmak ve yapmaya devam etmek,
  - Performans testleri yapmak ve yapmaya devam etmek,
  - Canlıya geçiş öncesi monitoring, profiling ve tuning çalışmalarına kaynak ayırmak.

Performans Problemlerinden Kaçınmak - II

- Teknoloji seviyesinde:
  - Hakim olmadığınız teknolojiyi kullanmamak,
  - Kullanılan teknolojilerin en iyi ve en kötü senaryolarını (best and worst case scenarios) ve en iyi kullanımlarını ve kalıplarını (best practices & patterns) iyi bilmek,
- Java seviyesinde:
  - Java'yı iyi bilmek,
  - Java'nın bileşenlerinin en iyi kullanımlarını ve kalıplarını (best practices & patterns) iyi bilmek,
  - Java API'sini etkin kullanmak

Sorular

Dinlediğiniz için teşekkür ederim.

Bu sunuma **www.selsoft.academy**'den ve www.javaturk.org'dan ulaşabilirsiniz.