Globalcode®
The Developers Company

Open-source Education

Java Performance & Tuning

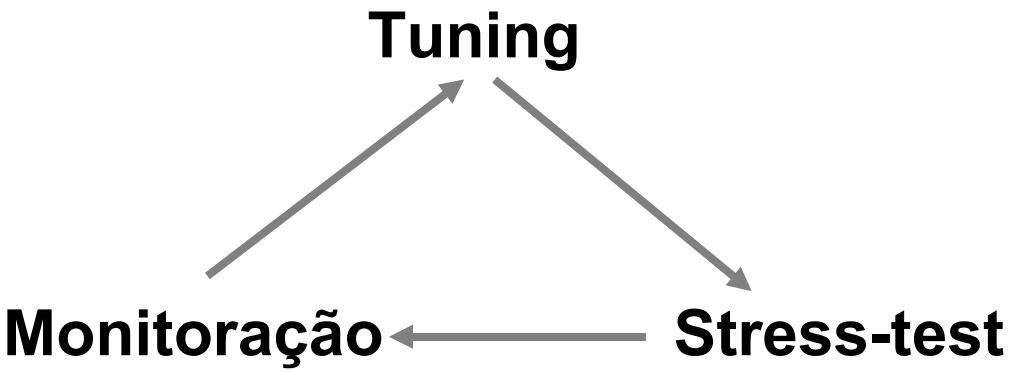
Iniciativa Globalcode

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

Introdução

- O objetivo deste mini-curso é apresentar a visão de performance e tuning através do seguinte ciclo:



Introdução

- No início deste mini-curso:

```
>java StressMemory
```

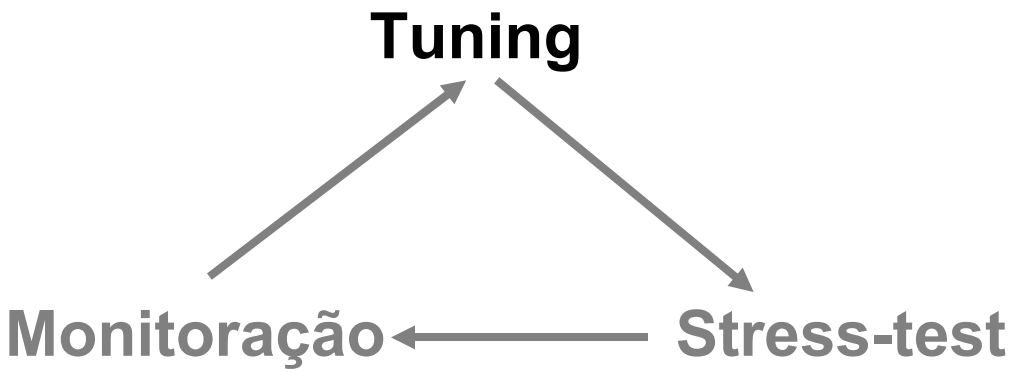
- No término...

```
>java -Dcom.sun.management.jmxremote
-verbose:gc -server -Xms50m -Xmx500m
-XX:+PrintGCDetails StressMemory
```

Tuning

Tuning = Ajuste fino

- JVM: memória, algorítimo de GC e parâmetros específicos.
- Web & Application Server: time-outs, threads, pooling.



Tuning

- Assunto extenso, que envolve diversas áreas;

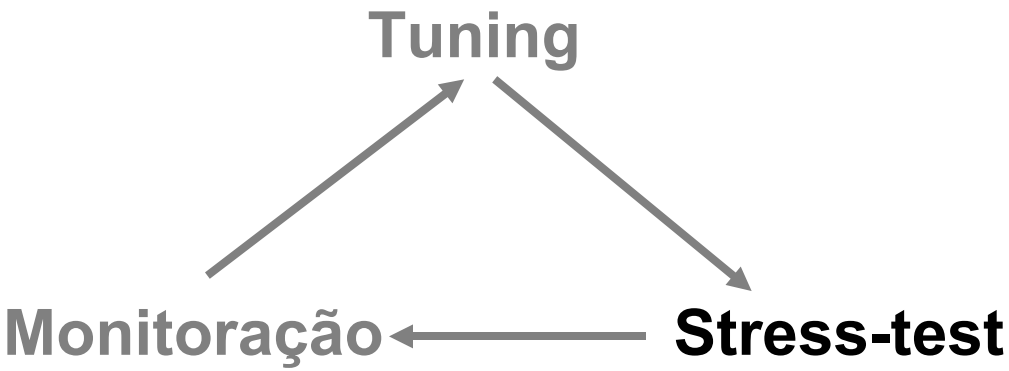
- Podemos encontrar gargalos e conseqüente necessidade de tuning nos seguintes recursos:

  - ✓ Sistema Operacional
  - ✓ Hardware do servidor
  - ✓ Rede
  - ✓ Máquina virtual
  - ✓ Application e Web Server
  - ✓ Database Server
  - ✓ Aplicativo Corporativo

Stress-testing

Stress-test = Simular

- Scripts que simulam o uso de uma demanda elevada;
- Ex. Ferramenta: Apache JMeter;



Monitoração

Show me the numbers...

- A monitoração fornece o estado de saúde de um servidor com um sistema, frente a uma determinada demanda.

- Através da monitoração identificamos gargalos e necessidade de otimizar / ajustar;

- Várias técnicas:
  - SNMP
  - Linhas de comando
  - Ferramentas específicas.

DEMO

Stress-testing, monitoração e tuning

DEMO

Código Java para stress:

```
public static void main(String args[]) throws Exception{
   if(args.length==0) {
     System.out.println("Uso: java StressMemory <numero
  objetos>");
     return;
   }
   int tamanho = Integer.parseInt(args[0]);
   ArrayList<String> v = new ArrayList<String>();
   System.out.println("Pressione enter tecla para começar...");
   System.in.read();
   for(int a=0;a<tamanho;a++) {
     double number = Math.random()*20000;
     v.add(new String("1234567890" + number));
     if(a%1000==0) System.out.println(a);
   }
 }
```

DEMO

Monitoração

- Utilizaremos o monitor do Windows / Linux na máquina do palestrante;
- A máquina virtual será monitorada através da ferramenta jconsole;
- Este recurso está na versão 1.5;

DEMO

Tuning

- Apesar de estarmos utilizando a máquina virtual 1.5, o parâmetro que será ajustado funciona igualmente na vm 1.4 e 1.3;
- O uso de vm 1.5 é em função dos recursos de monitoração;

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

JVM Tuning

- Diversas características podem ser ajustadas na Java Virtual Machine:
  - Memória máxima utilizada por objetos;
  - Distribuição da memória em diferentes áreas;
  - Perfil de cliente ou servidor;
  - Estratégia para limpar a memória;
- A máquina virtual provida no JDK é chamada de HotSpot;
- Duas formas: HotSpot Client ou HotSpot Server

JVM Tuning

- HotSpot client: é o comportamento padrão da máquina virtual*, reduz o tempo de startup e consumo de memória;
- HotSport server: ideal para aplicativos server-side onde performance é crítico, mas não tempo de startup e memória;
- De forma em geral HotSpot client vai privilegiar bastante GUI's;

```
java -client <Aplicativo> = HotSpot Client
java -server <Aplicativo> = HotSpot Server
```

JVM Tuning

- Parâmetros da JVM são divididos entre os padronizados (presentes em todas VM's) e os estendidos (não garantidos em todas VM's e continuidade).

- Para conhecer parâmetros de configuração da VM digite "java";

- Para conhecer parâmetros estendidos digite "java –X";

JVM Tuning

- Parâmetros simples e interessantes:

Apresenta log de execução de Garbage Collector

```
java -verbose:gc StressMemory
java -XX:+PrintGCDetails StressMemory
```

Apresenta log de carregamento de classes

```
java -verbose:class StressMemory
```

Configura caminho de pesquisa de classes

```
java -cp c:\ StressMemory
java -classpath c:\ StressMemory
```

JVM Tuning

- Ajustando o tamanho da memória heap:

Configura o mínimo de memória heap

```
java -Xms10m StressMemory
```

Configura o máximo de memória heap

```
java -Xmx64m StressMemory
```

Configura mínimo e máximo

```
java -Xms1g -Xmx3g StressMemory
```

JVM Tuning

Coletor de lixo de memória / Garbage Collection:

- O coletor default deve atender a grande parte das aplicações, principalmente aplicações client;
- A situação que você é realmente obrigado a ajustar o coletor é em aplicativos server-side que rodam com grande quantidade de threads e memória;
- 90% das situações e 99% das corporativas, o código não é alterado em função da forma de usar a memória;

JVM Tuning

Coletor de lixo de memória / Garbage Collection:

- Portanto o Garbage Collection é algo encapsulado do desenvolvedor.
- Você só precisará conhecer mais se o GC se tornar um gargalo no seu aplicativo.
- Objetos vivem conforme sua proposta de uso;

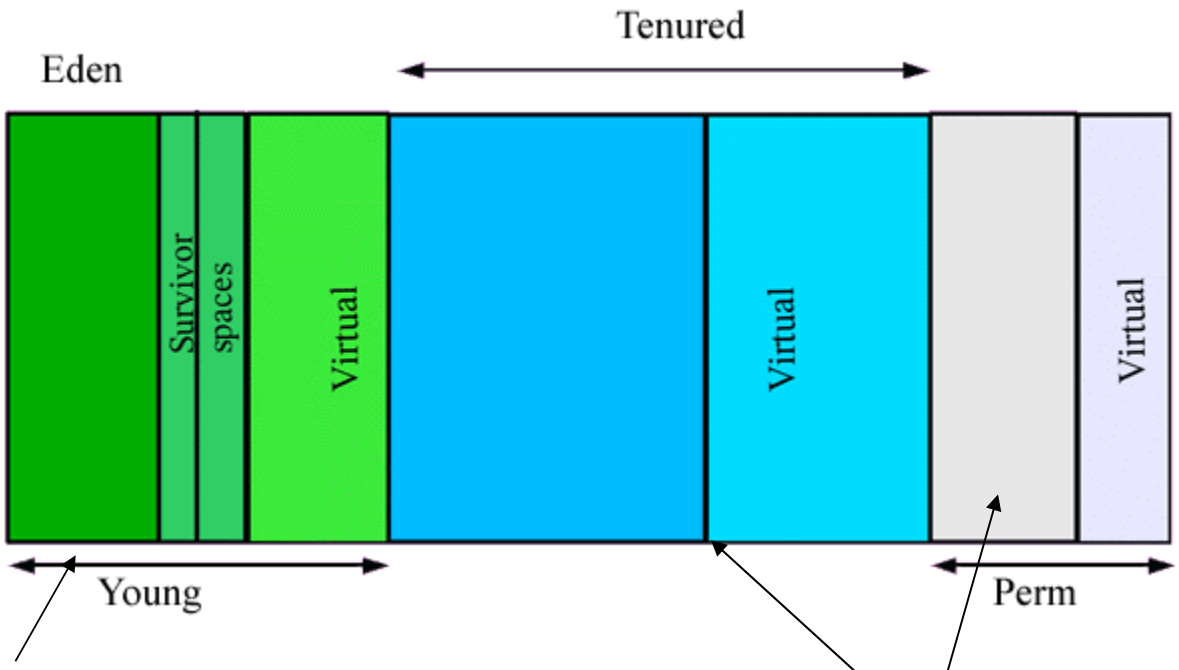
JVM Tuning

Áreas de memória

- Um objeto pode viver eternamente, médio prazo ou curto prazo;
- A JVM gerencia objetos por tempo de vida do objeto;
- Cada área de memória terá uma estratégia diferente de limpeza;

JVM Tuning

Áreas de memória



Muitas coletas, rápidas

Poucas coletas, lentas.

JVM Tuning

Áreas de memória - configurações

- Diversos parâmetros podem configurar as áreas de memória, alterando o comportamento do sistema:

```
-XX:MinHeapFreeRatio
-XX:MaxHeapFreeRatio
-Xms
-Xmx
-XX:NewRatio
-XX:NewSize
-XX:MaxNewSize
-XX:SurvivorRatio
```

JVM Tuning

Tipos de coletores

- Além do tamanho das áreas de memória, também podemos configurar o tipo / estratégia do coletor de lixo de memória;

- `Throughput: vai utilizar múltiplas threads para coletar Young / Eden Space.`

- `Concurrent low pause: múltiplas thread para young e tenured space.`

JVM Tuning

Tipos de coletores

- Incremental: faz diversas pequenas coletas de objetos velhos nas coletas dos objetos novos.

- Aggressive Heap: inspeciona os recursos do servidor e configura o que for melhor para "long-running & memory allocation-intensive" jobs.

JVM Tuning

Outros parâmetros

- Mais algumas dúzias de parâmetros podem ser alterados para pequenos ajustes, consulte a documentação do fabricante da sua máquina virtual.

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Tuning de Application Server;
6. Pooling de Threads;
7. Profiling de código Java com NetBeans;
8. Stress-testing com JMeter;

Instrumentação

- JMX – Java Management Extension é um padrão para gerenciar recursos como aplicativos, devices e serviços;
- Diversas classes de controle e serviços da máquina virtual 1.5 adotaram o padrão JMX;
- Um serviço gerenciado é chamado de Managed Bean ou MBean;
- Um servidor é um conjunto de MBean's, chamado de MBean Server;

Instrumentação

- Podemos monitorar um MBean server através de conectores;
- Para monitorar um o MBean server da VM, devemos configurar o seguinte parâmetro:

```
java -Dcom.sun.management.jmxremote
     -jar SwingSet2.jar
```

- Neste caso ligaremos o aplicativo SwingSet2.jar com recursos de monitoração de VM;

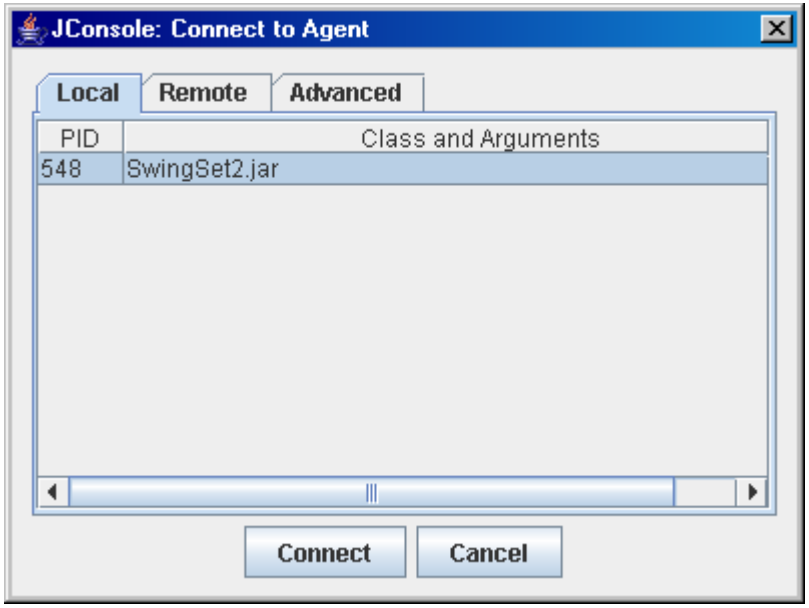
Instrumentação

- Agora ligaremos a ferramenta de monitoração embutida no JDK 5, digite jconsole:

```
c:\>jconsole  ou  [root@java root]jconsole
```

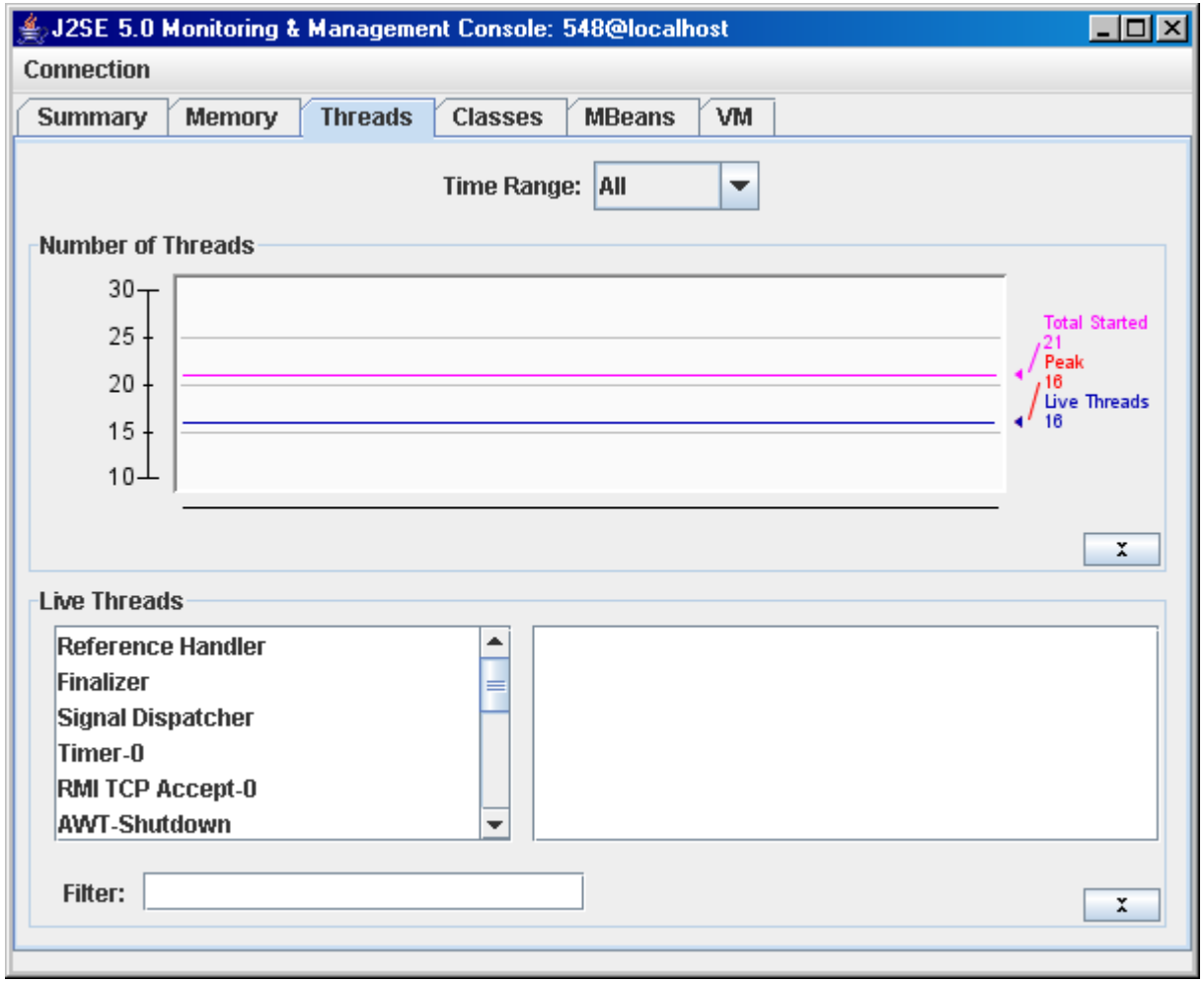
Instrumentação

- A seguinte janela deve aparecer, apresentando uma lista das VM's que podem ser monitoradas:



Instrumentação

- Memória, Thread, classes e seus próprios MBeans podem ser monitorados



Instrumentação

- Você pode criar objetos para embutir neste sistema de monitoração com baixo custo de desenvolvimento;
- Basicamente terá que desenvolver uma interface MBean e efetuar o registro do objeto no MBean Server;
- Maiores informações:

  JMX Tutorial
  http://java.sun.com/j2se/1.5.0/docs/guide/jmx/tutorial/tutorialTOC.html

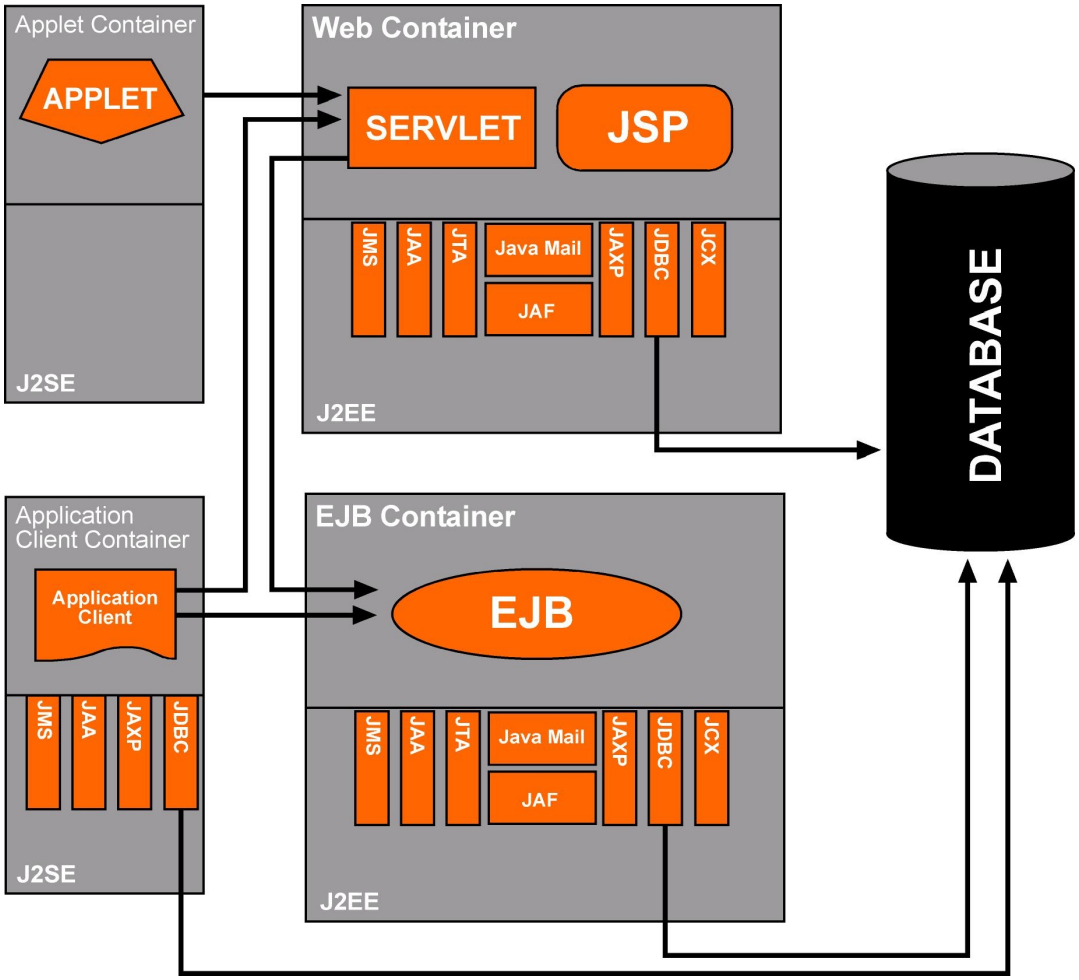
Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

Java EE

- A maior parte das implementações práticas de uma VM no mercado corporativo são os servidor EE;
- Portanto otimizar um aplicativo, muitas vezes significa otimizar um Servidor EE;
- Como o padrão Java Enterprise Edition oferece diversas opções de arquitetura, o tuning pode variar conforme a escolha da arquitetura;

Java EE


Applet Container
APPLET
J2SE
Web Container
SERVLET
JSP
JMS
JAA
JTA
Java Mail
JAF
JAXP
JDBC
JCX
J2EE
DATABASE
Application Client Container
Application Client
JMS
JAA
JAXP
JDBC
J2SE
EJB Container
EJB
JMS
JAA
JTA
Java Mail
JAF
JAXP
JDBC
JCX
J2EE

Java EE

- Arquiteturas Java EE comuns com EJB:

  ✓ 2 servidores físicos:
  1. HTTP + Web Container EE + EJB Container EE
  2. Database

  ✓ 3 servidores físicos:
  1. HTTP + Web Container EE
  2. EJB Container EE
  3. Database

Java EE

- Arquiteturas Java EE comuns com EJB:

  ✓ 4 servidores físicos:
  1. HTTP
  2. Web Container EE
  3. EJB Container EE
  4. Database

Java EE

- Arquiteturas Java EE comuns sem EJB:

  ✓ 2 servidores físicos:
  1. HTTP + Web Container EE
  2. Database

  ✓ 3 servidores físicos:
  1. HTTP
  2. Web Container EE
  3. Database

Java EE

- Todas as arquiteturas citadas podem trabalhar com servidores em clustering em qualquer camada;
- Cada Web Container e EJB Container pode trabalhar com uma ou mais máquinas virtuais;
- Cada máquina virtual de cada container, de cada aplicativo, pode utilizar ajustes de memória específicos para o contexto em questão;

Java EE

- Portanto este é o primeiro tuning que podemos fazer com Enterprise Edition: modelar a quantidade de camadas físicas, lógicas e a quantidade de máquinas virtuais em cada uma delas.

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

Threads

- Grande parte dos servidores Java EE permitem um ajuste na quantidade de threads em cada camada;
- Podemos configurar o número de threads para servlets, EJBs session bean, Entity Bean, processos agendados e muito mais;
- O número de threads deve crescer conforme o número de processadores;
- Cada servidor oferecerá um mecanismo diferente para tal configuração.

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

Profer

Se o código for o problema?

- Em diversas ocasiões encontramos os servidores com seus parâmetros supostamente ajustados e o problema de performance e gargalo persiste;
- Nestas ocasiões precisamos recorrer à ferramentas que analisam o tempo de execução do código;
- São chamadas de Profiler;

Profiler

Se o código for o problema?

- O NetBeans disponibiliza um profiler open-source e gratuito;
- Através dele podemos analisar o tempo e quantidade de vezes que métodos são executados em uma VM;

Agenda

1. Introdução;
2. Tuning de Java Virtual Machine;
3. Monitoração e instrumentação de JVM;
4. Arquiteturas Java Enterprise Edition;
5. Pooling de Threads;
6. Profiling de código Java com NetBeans;
7. Stress-testing com JMeter;

Fundamentos Stress-test

- Podemos dividir em quatro partes:

1. Metodologia;
2. Planejamento;
3. Execução & Monitoração;
4. Conclusões e plano de capacidade;

Fundamentos Stress-test

- O que devemos evitar?

  - ✓ Testar o “mais usado”;
  - ✓ Confundir requests simultâneos com usuários simultâneos;
  - ✓ Não planejar;

Fundamentos Stress-test

- Duas visões diferentes de stress-test

  - ✓ Quantos usuários suporta a solução?
  - ✓ Minha solução suporta X usuários?

Analise de requisitos

- Durante o período da analise devemos entender o ambiente e documentar as seguintes questões:

  - ✓ O que estressar?
  - ✓ Qual o objetivo do stress-test?
  - ✓ O que é um usuário?

Analise de requisitos

- Como?

  - ✓ Questionário perguntas diversas;
  - ✓ Distribuição e análise de incidência de use-cases;
  - ✓ Demo

Ferramenta JMeter

- Principais características:

  - ✓ Open-source
  - ✓ Simples, funcional e completo
  - ✓ Independente de plataforma (Java puro)
  - ✓ Pode-se usar para testar softwares Web em geral (Micro$oft, PHP, J2EE, Perl etc.)
  - ✓ Outros testes: FTP, SOAP, JDBC, LDAP e Java Classes

Ferramenta JMeter

- Básico
  - ✓ Ligando o JMeter
  - ✓ Documentação do JMeter
  - ✓ Extensão do JMeter

Demo

Diversas funcionalidades do JMeter.

Conclusões

Java oferece diversas oportunidades de tuning:

Memória
Threads
Objetos de negócio com Java EE (EJB)
Objetos Web com Java EE (Servlets e objetos públicos)
Conexões com banco de dados

Conclusões

Java EE oferece diversas oportunidades de tuning de arquitetura:

2, 3 ou 4 camadas com EJB's
2 ou 3 camadas sem EJB's
Com ou sem clustering
Com múltiplas máquinas virtuais
Com diferentes configurações

Conclusões

Java oferece ferramentas para monitoração e stress-testing:

jconsole
Apache JMeter

Conclusões

Por esses e outros motivos Java é a plataforma mais utilizada no mundo para aplicativos na baixa plataforma.

Perguntas & Respostas

Q&A

vinicius@globalcode.com.br

Globalcode
The Developers Company

Globalcode®
The Developers Company

Open-source Education

Java Performance & Tunning

Iniciativa Globalcode