

[www.qconferences.com](http://www.qconferences.com)

[www.qconbeijing.com](http://www.qconbeijing.com)

[www.qconshanghai.com](http://www.qconshanghai.com)

# QCon

伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

London • Beijing • Tokyo • New York • Sao Paulo • Shanghai • San Francisco

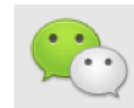
## QCon全球软件开发大会

International Software Development Conference

# InfoQ<sup>ueue</sup>



@InfoQ



infoqchina

软件  
正在改变世界!



# Java Virtual Machine Virtualization

## - building scalable JVM for Cloud

## About me

- 9 years working in Java
- Recent work focus:
  - ✓ OpenJDK (HotSpot) optimization in **alipay**
- Past lives
  - ✓ Java Virtual Machine (IBM' J9) improvements for 'cloud'
  - ✓ Java security development (Expeditior, kernel of Lotus notes)
- My contact information
  - mail: [sanhong.lsh@alibaba-inc.com](mailto:sanhong.lsh@alibaba-inc.com)
  - weibo: [sanhong\\_li](#)

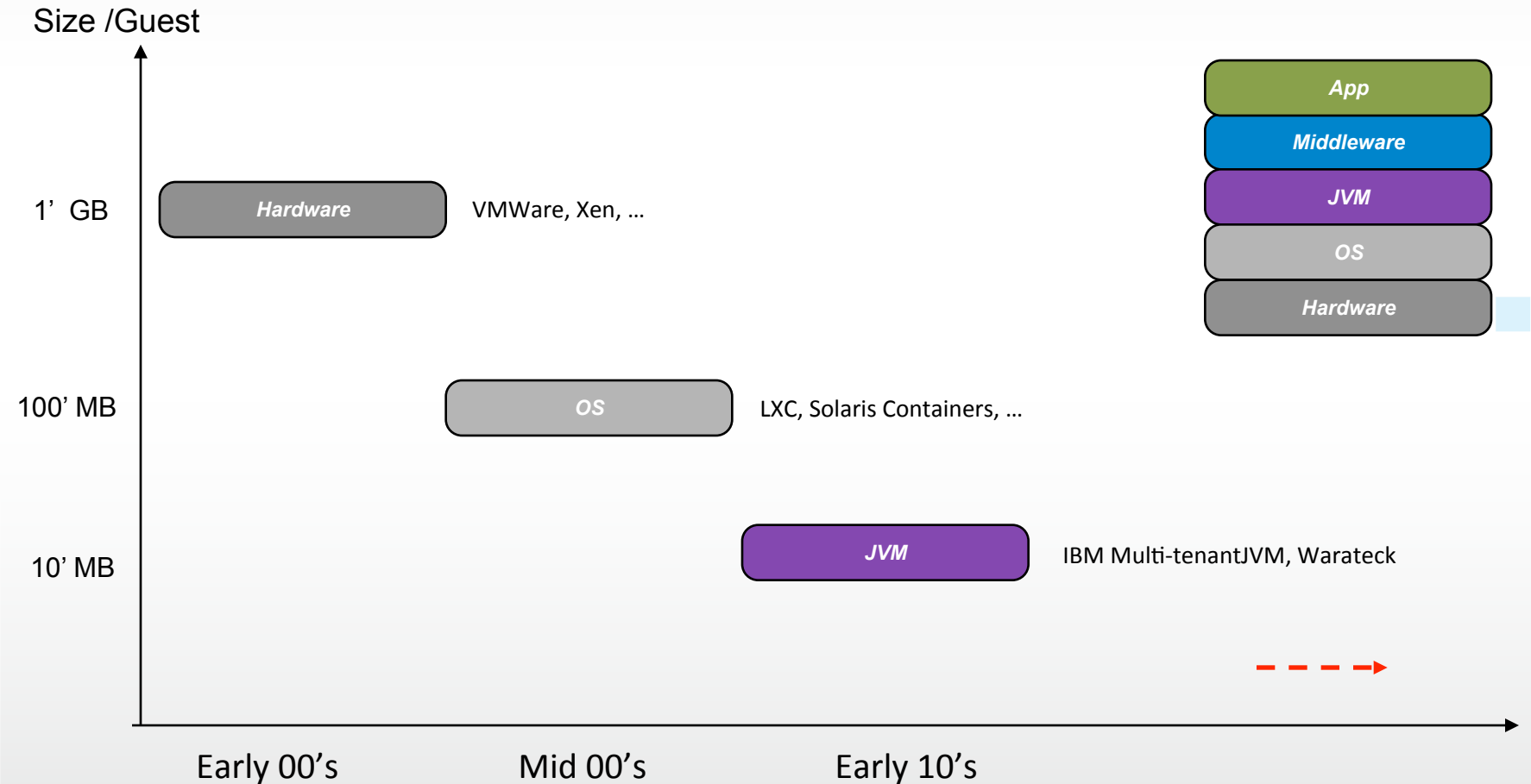
## The goal of talk

- By the end of this session, you should be able to:
  - Understand why the JVM virtualization is essential for cloud
  - Gain insight into the challenges of virtualizing JVM and as well as possible solutions
  - Discover the features of IBM Multi-tenant JVM for running multiple Java applications in shared JVM

# Agenda

1. Why we want JVM Virtualization
2. Technical challenges for virtualizing JVM
  - Data Isolation
  - Neighbourhood Watch: prevent DoS
  - Java Class Library: multi-tenancy support
3. Basic Introduction to IBM Multi-tenant JVM
4. Q&A

# Virtualization history



# Virtualize JVM for cloud

- Consolidation
  - Automatic de-duplication (ability to share Java artifacts)
  - Reduce overhead at JVM level
  - Deployment
  - Resource usage
  - Startup
- Share
  - 'bursty' workloads give an opportunity to share through careful choreography
- Isolation
  - Protect application in their own sandbox



# Data isolation

- In Java, you can't
  - forge the data reference
  - do the unsafe casting
  - jump to arbitrary code location
- Only the data exchange mechanism is through **static field** of class!
- An Example:

```
3 public class Foo {  
4     private static int globalCount = 0;  
5  
6     public static void increase() {  
7         globalCount++;  
8     }  
9  
10    public int getGlobalCount() {  
11        return globalCount;  
12    }  
13 }
```

# ClassLoader based Isolation

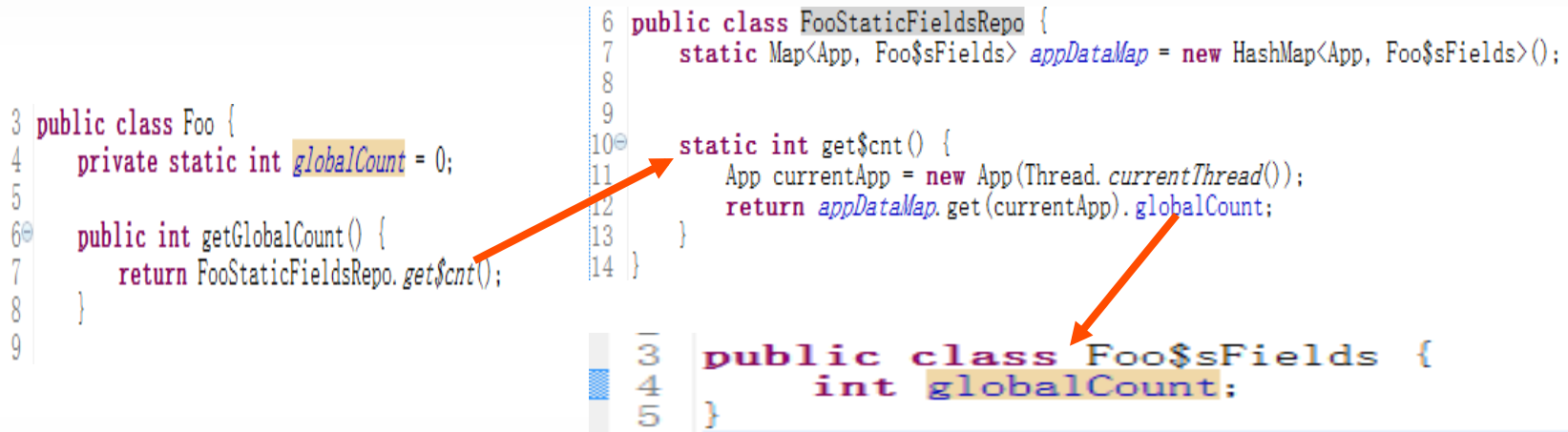
```
24  
25  
26 URLClassLoader urlLoader = new URLClassLoader(urls);  
27 Class<?> fooClass = urlLoader.loadClass("virtualization.Foo");  
28  
29
```

## Problem

- Duplicated copies of *Foo.class*
- Duplicated JIT efforts
- Can not handle the case where the static field of system class is shared, e.g System.out

# Bytecode transformation (aka. BCI)

```
3 public class Foo {  
4     private static int globalCount = 0;  
5  
6     public int getGlobalCount() {  
7         return FooStaticFieldsRepo.get$cnt();  
8     }  
9  
6 public class FooStaticFieldsRepo {  
7     static Map<App, Foo$sFields> appDataMap = new HashMap<App, Foo$sFields>();  
8  
9  
10    static int get$cnt() {  
11        App currentApp = new App(Thread.currentThread());  
12        return appDataMap.get(currentApp).globalCount;  
13    }  
14 }  
  
3 public class Foo$sFields {  
4     int globalCount;  
5 }
```



- Actually need additional check in reflection system and load-time to prevent from accessing generated classes
- Problem
  - Correctness/security issue from handcrafted bytecode/JNI
  - Performance overhead incurred by generated classes

# Modify Bytecode Implementation in Runtime

## Bytecode interpreter (pseudo –code)

```
while (true) {  
    switch (*pc++) {  
        case putstatic:  
            //set per application  
            ...  
            break;  
        case getstatic:  
            //get per application  
            ...  
            break;  
        ...  
    }  
}
```

- Cool! It is a clean way for isolation
- But, need care a couple of more things:
  - Class initialization, <clinit> handling
  - Just in time (JIT) support

# <clinit> gets called per app

**getstatic** does 2 things (*similar logic to putstatic*)

1. Triggers class initialization on first contact
2. Resolves a name (**in**) to a real storage location and reads from it

```
55  * @since   JDK1.0
56  */
57  public final class System {
58
59      /* First thing--register the natives */
60      private static native void registerNatives();
61      static {
62          registerNatives();
63      }
64
65      /** Don't let anyone instantiate this class */
66      private System() {
67      }
68
69      /**
70       * The "standard" input stream. This stream is already
71       * open and ready to supply input data. Typically this stream
72       * corresponds to keyboard input or another input source specified by
73       * the host environment or user.
74       */
75      public final static InputStream in = nullInputStream();
76
77  }
78
79  public abstract class HelloWorld {
80
81      public static void main(String[] args) throws IOException {
82          System.in.read();
83      }
84  }
```

## <clinit> gets called per app

By JVM spec, the <clinit> happens when

- The invocation of a constructor on a new instance of the class
- The invocation of a method declared by the class (not inherited from a superclass)
- The use or assignment of a field declared by a class (not inherited from a superclass), except for fields that are both static and final, and are initialized by a compile-time constant expression.

# Denial-of-service attacks prevention

- Data isolation can't prevent DoS, we need approach for resource control, including:
  - CPU
  - Heap
  - Network IO
  - Disk IO

# CPU management in OS

- Control Group directly provided by Linux kernel
  - Allocate resources for aggregated processes
    - Partition all Processes (&children) into groups
    - Organize groups in hierarchies
    - Associate groups with particular resource
    - Manage resource among groups
  - Adopted by Docker, LXC
- WLM on AIX

```
#!/etc/cgconfig.conf

mount {
    cpu = /mnt/cgroups/cpu;
    cpuacct = /mnt/cgroups/cpuacct;
    net_cls = /mnt/cgroups/net_cls;
}

group . {
    perm {
```



# CPU management in container

- [Linux-Vserver](#) is operating system-level virtualization technology
- Basic idea of CPU Isolation
  - Token Bucket Scheme on top of process scheduler
    - Assign token bucket per container
    - Charge tokens from container at every timer tick
      - The process of container will be removed from run-queue when it is running out of tokens

# CPU management in JVM

- Challenges in Java World
  - JVM doesn't have the ability to schedule the thread which in turn depends on OS
  - The underlying CPU capacity change, which often happens in virtualization environment, is a black box to JVM
- Related research paper
  - [A Portable CPU-Management Framework for Java](#)
    - The rationale:
      - The CPU consumption of thread is expressed by executed JVM bytecode
    - The problem
      - Hard to evaluate the real cost made by bytecode
      - JIT and JNI limitation

# Two possible approaches

1. Delegate the management to OS
  - How to glue JVM to OS?
2. Manage by JVM self
  - CPU time calculation per thread
  - Constraint algorithm
  - Thread scheduling

# Resource reclamation

How to reclaim the resources when the application died in shared JVM? The challenges include:

- Terminate the threads
  - ✓ No safe way to kill a thread, `Thread.terminate` is not encouraged
- Close any opened IO(File/Socket) handle
  - ✓ OS manages these handles per process, not per thread

# Multi-tenant aware class library

Relatively easy to achieve...

- System properties isolation
- The semantic of `System.exit(code)`
  - ✓ The shutdown of application is different from JVM
- Create file in relative path
- java shutdown hook
- ..., etc.

```
10 public static void main(String[] args) throws IOException {  
11  
12     String strHello = "Hello from application:" + System.getProperty("appProperty");  
13  
14     byte[] outBytes = strHello.getBytes();  
15     OutputStream out = new FileOutputStream("./output.txt");  
16     out.write(outBytes, 0, outBytes.length);  
17  
18     out.close();  
19     System.exit(0);  
20 }
```

# Precedents

- Sun/Oracle MVM & JSR 121 - Application Isolation API Specification



ORACLE

- Is a multi-tenant JRE
  - Allows multiple Java applications to run in the same JVM
  - Provides fine isolation among applications by isolating all static fields, as opposed by using different classloaders
  - Enables sharing of class bytecode and meta-data

- Google App Engine & JSR 284 - Resource Consumption Management API



Google™

- Is a multi-tenant middleware service
  - Allows multiple Servlet applications deployed into the engine (and scaled to multiple nodes on demand)
  - Controls resource consumption explicitly
    - CPU
    - Bandwidth
  - Limits Java SE API access
- Provides a “namespace” based multi-tenant programming model for hosted applications

# Waratek Cloud JVM

- **About**

- Based in Dublin, Ireland
- web: <http://www.waratek.com/>

- **In a Nutshell**

- *Java-compatible Virtual Machine (JVM) based on Oracle HotSpot*
- *Run many Java applications encapsulated in isolated domains on a single CloudVM host for higher utilization, without risking critical processes being starved of compute or memory resources*

- **Approach to Virtualization**

- Waratek JVM is a meta-circular interpreter, written in Java
- Each application runs in a **Java Virtual Container** or **JVC**
  - Each JVC contains a fully isolated virtual image of a shared JVM system (instead of ISA) for each guestLogically isolated view of the Java type

# IBM Multi-tenant JVM & Introduction

- Allow for collocation of multiple Java applications in a single instance of JVM
  - *Isolate application from one another and each "thinks" it has the whole VM all to itself.*
  - *Share metadata aggressively and transparently, such as:*
    - *bytecodes of methods*
    - *GC*
    - *JIT*



More info, pls. access [Multi-tenant JVM public community](#)

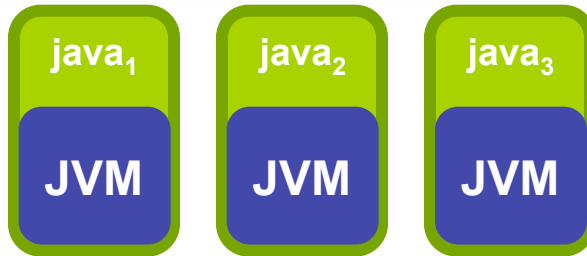


# Getting started

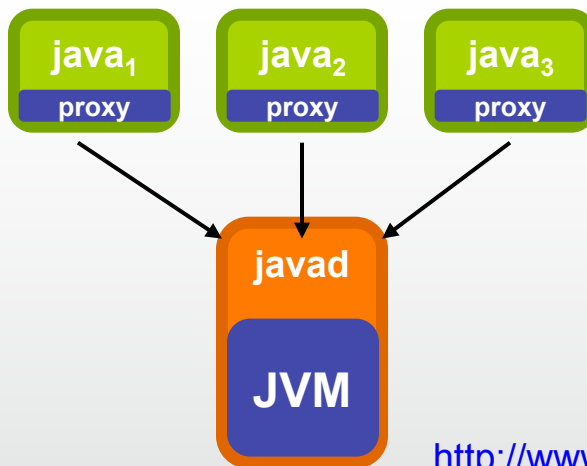
- **Multitenancy is an opt-in feature** of IBM Java 7.1
  - Just add the **-Xmt** command-line option to opt-in
  - Enables a model very similar to JSR-121: Isolates but doesn't require any new API
- **Daemon startup and communications is handled automatically** by the 'java' launcher
  - One daemon per user to keep permissions aligned between launcher & daemon
  - Launcher: daemon rendezvous accomplished using advertisement files
- **Standard in / out / error streams are connected to daemon**
  - e.g. `System.out.println()` in the daemon works as expected
  - Javad will multicast messages like dump events to all connected tenants
- **Most standard JVM options are used as-is**
  - `-classpath` / `-jar`/`-cp` entries
  - `-Dname=value` system properties
- **Select JVM options are mapped to tenant-specific values**
  - `-Xmx` applies to the tenant being launched
  - See documentation for details
- **Daemon-wide options are stored in `JAVA_HOME/bin/javad.options` file**

# How does it work

- A standard **java** invocation creates a dedicated (non-shared) JVM in each process



- IBM's Multitenant JVM puts a lightweight 'proxy' JVM in each **java** invocation. The 'proxy' knows how to communicate with a shared JVM daemon called **javad**.

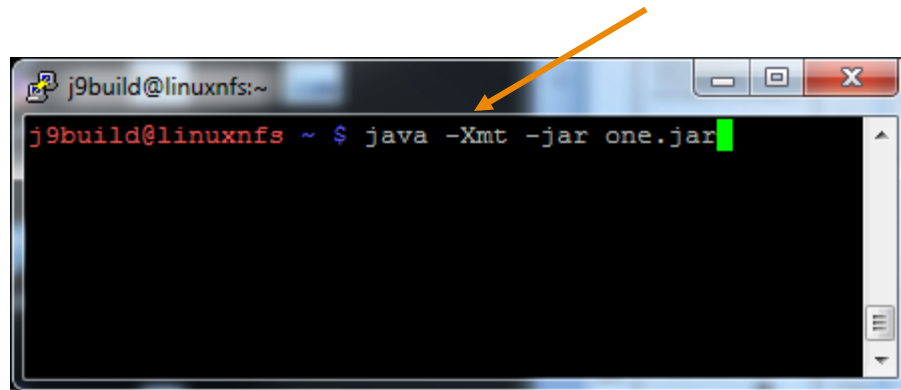


- **javad** is launched and shuts down automatically
- no changes required to the application
- **javad** process is where aggressive sharing of runtime

<http://www.ibm.com/developerworks/library/j-multitenant-java/>

# Launch your application

- Opt-in to multitenancy by adding **-Xmt**

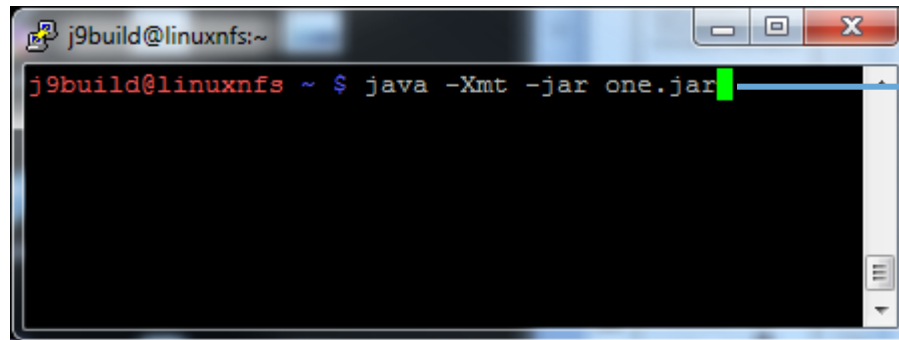


A terminal window titled 'j9build@linuxnfs:~' showing the command `java -Xmt -jar one.jar` being entered. An orange arrow points from the text '-Xmt' in the list above to the '-Xmt' in the command. The terminal has a black background with white text. The window title bar shows standard Linux window controls (minimize, maximize, close).

```
j9build@linuxnfs ~ $ java -Xmt -jar one.jar
```

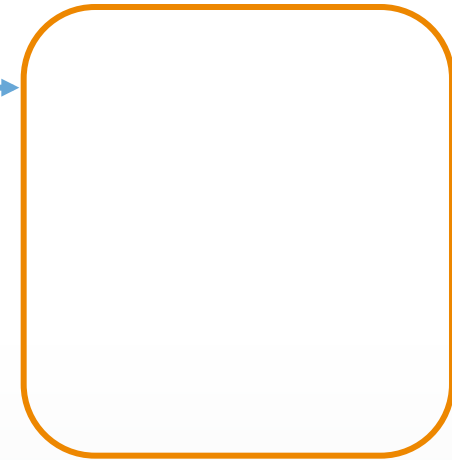
# Register with **javad** daemon

- JVM will locate/start daemon automatically



A terminal window with a title bar showing 'j9build@linuxnfs:~'. The prompt is 'j9build@linuxnfs ~ \$' and the command entered is 'java -Xmt -jar one.jar'. A green cursor is at the end of the command. A blue arrow points from the end of the command to a large orange rounded rectangle.

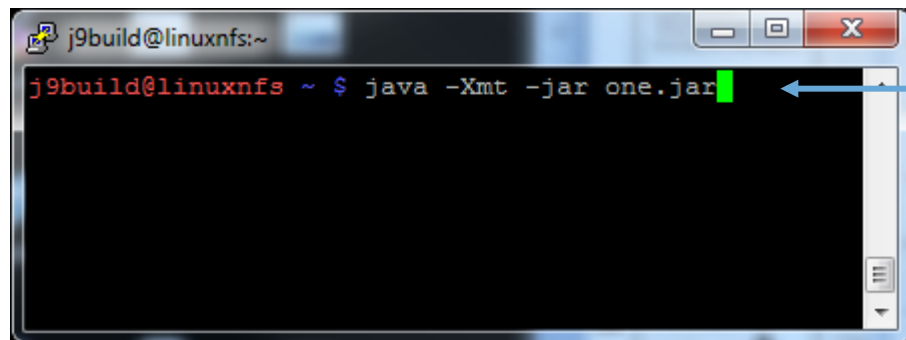
locate



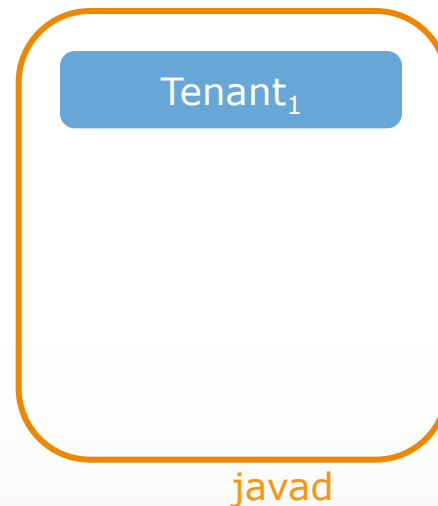
javad

# Create a new tenant

- New tenant created inside the **javad** daemon

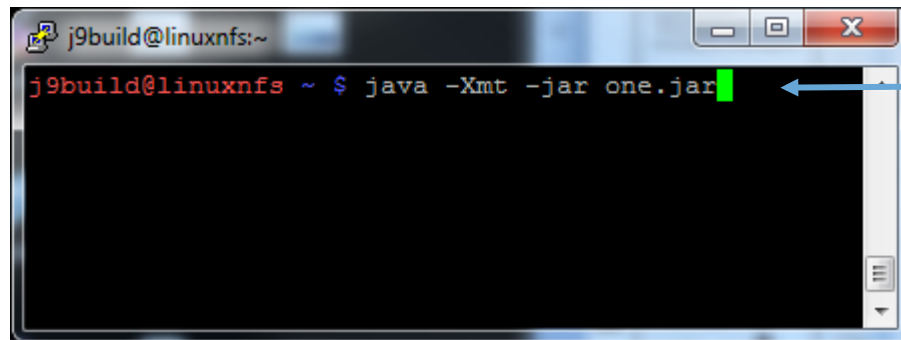


A terminal window with a title bar showing 'j9build@linuxnfs:~'. The command prompt is 'j9build@linuxnfs ~ \$' followed by the command 'java -Xmt -jar one.jar' with a green cursor at the end.

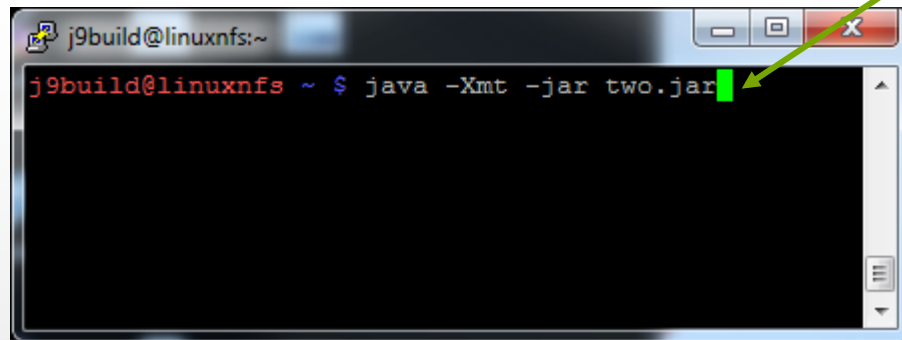


# Create a second tenant

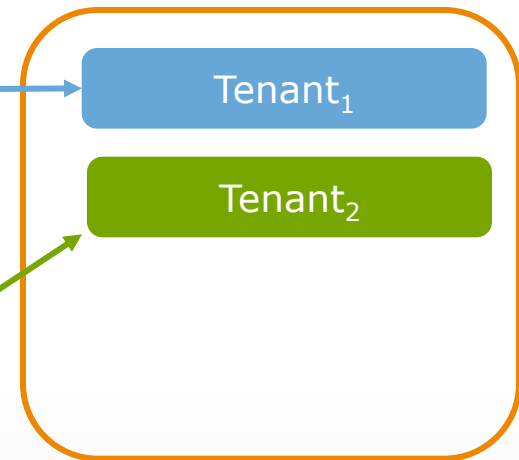
- New tenant created inside the **javad** daemon



```
j9build@linuxnfs:~  
j9build@linuxnfs ~ $ java -Xmt -jar one.jar
```



```
j9build@linuxnfs:~  
j9build@linuxnfs ~ $ java -Xmt -jar two.jar
```



javad



One copy of common code  
lives in the javad process.

Most runtime structures  
are shared.

# Options for resource limit

- Throttling is controlled using a new **-Xlimit** command-line option
  - General form is: **-Xlimit:<resource\_name>=<min\_limit>-<max\_limit>**
  - <min\_limit>: Specifies the minimum amount of the resource that must be available for the tenant to start. This value is optional.
  - <max\_limit>: Specifies the maximum amount of the resource that the tenant is allowed to use.
- Examples:
  - Xlimit:cpu=10-30**
    - requires a 10% share of the processor to start and limits processor consumption to 30%.
  - Xlimit:threads=5-20**
    - requires a minimum reservation of five threads and an upper limit of 20
  - Xmx20m**
    - Limit heap consumption to 20 megabytes

# Sweet spot

- **How low can you go?**

- Simple ('Hello World') applications showing per-tenant sizes of ~170 KB of heap
- This equates to a **5-6x** more applications running on the same hardware

- **Performance(throughput)**

- Target is 10% overhead, still a work in progress

- **Second-run start-up times are significantly better**

- Faster because the JVM is already up and running

- **Application Sweet spot:**

- **One of:**
  - Relatively large class:heap ratio (JRuby and other JVM languages)
  - Require fast startup: run-and-done / batch
  - Workloads with varying busy:idle cycles – MT JDK is better at shifting resource between tenants
- **100% pure Java Code**



# Caveats & Limitations

## ■ Main Limitations of the MT Model

### JNI Natives

- The operating system allows the shared JVM process to load only one copy of a shared library. Only native libraries present on the bootclasspath of the JVM usable.

### JVMTI

- Because debugging and profiling activities impact all tenants that share the JVM daemon process, these features are not supported in the multitenant JVM process model. Note: we do have per-tenant -javaagent: support.

### GUI programs

- Libraries such as the Standard Widget Toolkit (SWT) are not supported in the multitenant JVM process model because the libraries maintain a global state in the native layer.

# Key Links and Contacts

- Download (IBM Java7.1)  
<https://www.ibm.com/developerworks/java/jdk/linux/download.html>
- Documentation (IBM Java7.1)  
[http://pic.dhe.ibm.com/infocenter/java7sdk/v7r0/index.jsp?topic=%2Fcom.ibm.java.aix.71.doc%2Fdiag%2Fpreface%2Fchanges\\_71%2Foverview\\_mt\\_evaluation.html](http://pic.dhe.ibm.com/infocenter/java7sdk/v7r0/index.jsp?topic=%2Fcom.ibm.java.aix.71.doc%2Fdiag%2Fpreface%2Fchanges_71%2Foverview_mt_evaluation.html)
- Public community  
<https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=aa61d6e5-75c1-4a61-b88c-6ec8d80c49c7>
- Contacts for feedback
  - Project lead: Michael Dawson ([michael\\_dawson@ca.ibm.com](mailto:michael_dawson@ca.ibm.com))
  - Manager of JTC Shanghai: Zhou Kai ([zhoukai@cn.ibm.com](mailto:zhoukai@cn.ibm.com))
  - Official weibo: @IBM\_JTC



# Q&A

reach to me:

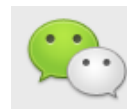
mail: [sanhong.lsh@alibaba-inc.com](mailto:sanhong.lsh@alibaba-inc.com)

weibo: [sanhong\\_li](#)

Brought by **InfoQ**



@InfoQ



infoqchina

软件  
正在改变世界!