DRLVM Development Tasks

This page contains a list of development tasks for the Harmony Virtual Machine (DRLVM) which are not JIT or GC specific.

The list of tasks is being developed now, so it will be extended in nearest future. Also the tasks will be reprioritized according to current needs of Apache Harmony project. There is also a list of Known Non Bug Issues And Limitations - most of issues are also a subject for resolution, so the development task may be taken from that list.

If you are not sure what you can start with then refer toDynamic Runtime Layer Virtual Machine Developer's Guide to get the basic concepts and terms, components and modules description.

Kernel Classes

This section contains a list of isolated development tasks, which do not require advanced knowledge of VM and could be a nice start for newbies to get acquainted with the code. All items are targeted for better code sharing.

1. Improve/re-implement a Java5.0 generic signatures parser in DRLVM

Problem: (HARMONY-2052). The current implementation is messy and half-baked, one needs to invent more shaped and modular API to the module, develop a simple parser/lexer to avoid antlr dependency and move this functionality to classlib (luni ?), so other VMs could reuse it for 1.5 support.

Task: Improve/re-implement a Java5.0 generic signatures parser in DRLVM.

2. Unify implementation of j.l.Runtime/Process

Problem: (HARMONY-2051) The classlib provides a neat portlib-based reference implementation [*], which should be reused. These two implementations are roughly identical, so one needs to make more scrupulous comparison and squeeze some features/fixes of [**], which may be missing in [*], then employ [*] in `j.l.Runtime` of DRLVM and drop [**].

Task: Eliminate duplicate implementation of j.l.Runtime.Process in kernel classes of DRLVM

[*] working_classlib\modules\luni\src\main\java\org\apache\harmony\luni\internal\process
* + working_classlib\modules\luni\src\main\native\luni\shared\process.c
[**] working_vm\vm\vmcore\src\kernel_classes\javasrc\java\lang\Runtime.java + working_vm\vm\vmcore\src\kernel_classes\native\Runtime_[lnx|win].cpp

3. Move port_user_timezone() from DRLVM to classlib

Problem: (HARMONY-2053) There is no reason to burden VM with initializing "user.timezone" property value during VM startup for Classlib's j.u. TimeZone class. This action should be done by hyluni natives during `JNI_OnLoad`, therefore it is suggested to move `port_user_timezone()` function from DRLVM to classlib (luni/port), and fix DRLVM & hyluni accordingly.

Task: Move port_user_timezone() from DRLVM to classlib.

JNI

4. Develop JNI Weak References

Problem: The Weak References implementation is required by Java Native Interface Specification. Ok, it just needs to be implemented.

Task: Develop JNI Weak References.

Launcher

5. Main Thread stack size control

Problem: (HARMONY-4733) Java Thread can be created with any stack size (limited by maximum), still the main Thread is provided by the underlying operation system, so its stack size can't be changed, so this stack can be simply overflowed – it is a problem. The solution is to keep the main system thread unused and create the java main thread by VM means.

Task: Reimplement main thread processing in launcher.

Application Binary Interface Glue

6. Unify JIT/VM ABI glue (replace 5 approaches with a single approach)

Problem: Currently there are five approaches to generating ASM that translates from JIT ABI to C/C++ ABI. This glue also deals with exception throwing and live reference enumeration. The five approaches are:

- Microsoft ASM
- Linux/gcc ASM
- JIT back-end (hardcoding what gets emitted by using the JIT emitter)
- LIL
- vmmagic (currently allows type-unsafe memory read/writes from Java)

The above five approaches times two architectures (x86 and x86_64) means we have 10 distinct, hard to maintain kinds of asm in the code base. The goal is to reduce this to the bare minimum – one for x86 and one for x86_64. First, determine if we can use a combination of vmmagic and JIT intrinsics to accomplish the task. Second, if the above is doable, cleanup the old ASM code.

Task: Replace 5 approaches to JIT/VM ABI glue with a single approach.

Finalization System

7. Improve Finalization Work Balance Mechanism (FWBM)

Problem: FWBM was changed with a move to new Garbage Collector (GCv5). This resulted in FWBM becomes not well adjusted, the finalization system behavior differs from RI one which may be critical for stability and performance of resource critical applications. One needs to evaluate FWBM in DRLVM and RI, identify issues (e.g. one may relate to dead lock between finalizer thread and another thread using same monitor) and get them resolved.

Task: Evaluate compatibility with RI and make required improvements.

Monitor support

8. Release monitor in unwinded frame

Problem: There are three cases when monitor must be released:

- when exception handling
- when the owning monitor thread dies in unusual place (say due to `exit` call)
- when JVMTI `PopFrame` is called

The support for all three cases should be implemented. Also support for first two cases is performance critical (must be implemented in optimal way).

Task: Implement monitor releasing for frame unwinding.

Crash Handler

9. Implement Ctrl-C and Ctrl-\ correct processing

Problem: (HARMONY-5390) The handlers for Ctrl-C and Ctrl-\ (SIGINT and SIGQUIT) are incorrect. This leads to crash or hang-up in these handlers on Linux. The handlers need to be carefully rewritten with proper synchronization and correct access to internal VM data.

Task: Redevelop Ctrl-C and Ctrl-\ handler.

Utilities

10. Implement VM specific utilities library and remove C++ exception support

Problem: Currently STL is used by set of DRLVM component, also it was practiced to develop own simple utilities specific to particular components (often produces the code / bugs duplication). One needs to identify the requirements for DRL-specific utilities and approach the specific library for shared use (thus replace STL and inherent buggy utilities).

Code for handling C++ exceptions creates additional overhead and since we don't use C++ exceptions, VM could be compiled without its support. To make it possible it is necessary not to use MS.NET STL classes because they require comparator to generate code with support for C++ exceptions.

Task: Design and implement the VM specific utilities library.

Debugging / Profiling support

11. Complete implementation of j.l.instrument

Problem: The implementation of j.l.instrument contains a set of stabs instead of code being directly connected to Virtual Machine internals. This may partially caused by missed functionality in DRLVM. So one need to identify DRLVM requirements for j.l.instrument support, develop the missed functionality to provide 100% support of j.l.instrument.

Task: Develop unit test for j.l.instrument, identify the missed functionality and get it developed.

12. Implement RedefineClasses JVMTI feature for DRLVM

Problem: The RedefineClasses is the major JVMTI feature which is not implemented yet. Also this feature affects the desing of major DRLVM components so it is just complex development task.

Task: Develop JVMTI RedefineClasses support.

Stack suppport

13. Desing/implement VM stack iteration interface

Problem: There are three interfaces to Stack Iteration – one interfaces for JVMTI support, one – for Stack Integrator and final one – for Stack Trace… There are also functions to work with Stack that do not belong to any of these 3 interfaces (for exception support, crash handler support, security and so on). Such interfaces / code mess is buggy. One needs to propose the flexible unified interface for work with Stack, get it implemented and transit existing VM modules to use it.

Task: Clean up VM stack iteration interface used by all of its components.

Overall VM

14. Use correct version of abort/exit

Problem: VM should not use abort/exit to harm the process it is running in. When VM is run from within a browser or any other process it cannot shut down the process, it needs to use abort and exit function pointers that are passed to it in JNI_CreateJavaVM.

Task: Patch all VM code to use the correct version of abort/exit functions.

15. Change static arrays to the dynamically allocated ones

Problem: This task addresses VM Extensibility - getting rid (where possible and reasonable) of hardcoded values for arrays length to keep data for/in different VM components, document all remained restrictions.

Task: Change static arrays to the dynamically allocated ones.

16. Add I18N support for exception messages

Problem: Right now there is no support to localize the message given in objects thrown by DRLVM. Like if Class Support fails to load a class, the NoClass DefFoundError can't have a message on Russian. The I18N support for this case should be design and implemented, all of messages in DRLVM should be internationalized.

Task: Add I18N support for exception messages in DRLVM.

17. Add Java6 support in DRLVM

Problem: Java6 support is incompleted in DRLVM.

Task: Java6 support is incompleted in DRLVM. This includes the following (with priorities):

1. Reflection – changes are trivial, could be done quickly. Will be required for correct Java 6 classlib, so P1. 2. Management (ThreadMXBean) – according to Andrey Yakushev's estimation it is 1-2 WW. P3 3. JVMTI
    - ForceEarlyReturn – seems to be the most useful new feature for debugging, so P2
    - AddToSystemClassLoaderSearch event – might be used in debugging as well, P3
    - GetOwnedMonitorStackDepthInfo – might be used in debugging for analyzing dead locks, P3
    - RetranformClasses functionality – needed for instrumentation support, depends on implemented RedefineClasses from Java 5. P4
    - Enhanced heap iteration API – according to Chris no one uses even the first version of heap iteration API yes, so P4
    - Implement dynamic attach of debugger/profiler. Sun doesn't open the protocol for it and spec is not clear. P4
    - ResourceExhausted event – P5

    4. Instrumentation – seems to depend on JVMTI implementation of RetransformClasses and other functionality for class redefinition, also Java code should be written in classlib. I don't know applications that use this functionality, so P4

18. Implement HYTHR_0.2 interface in DRLVM's HyThread

Problem: If HYTHR_0.2 interface is implemented in both DRLVM and J9 VM then one has the following benefits:

- copying DRLVM's HyThread library from bin/default/ to bin/ will be not needed anymore
- 'hy.no.thr' mode of Classlib build will become default, so classlib's HyThread will disapper; both VMs will have their own HyThreads in bin/default/
- the above will simplify switching between DRLVM and J9

Task: Implement HYTHR_0.2 interface in DRLVM's HyThread to support HARMONY-3090 changes (affects modularity)

The task from old list which required clarification

*** Garbage Collection**

Task: Implement SoftReference semantics: use heap usage information to make a decision on resetting soft references. Currently soft reference act exactly like weak references and are reset immediately after the target object became weakly reachable.

*** MMTk Integration**

MMTk_Development_Tasks