システムソフトウェアと
オペレーティング・システム　89－１２
システム評価　2－１２
（２００２．２．１５）

# リアルタイムオペレーティングシステム Easel における JavaVM の設計と実装

奥山 玄[†]　瀧本 栄二[†]　芝 公仁[†]　大久保 英嗣[††]

[†]立命館大学大学院理工学研究科　[††]立命館大学理工学部

現在，我々は，リアルタイムオペレーティングシステム Easel 上で JavaVM の設計と実装を行っている．リアルタイム環境における Java の問題点として，クラスのダイナミックローディングによりオーバヘッドが発生すること，ガーベジコレクション (以下，GC) が必要であること，また JavaVM に時間制約を保証する機能がないことなどが挙げられる．我々は，これらの問題を解決するために，JavaVM のリアルタイム拡張を行い，Easel 上で評価を行った．本論文では，特にガーベジコレクションに焦点を当て，JavaVM のリアルタイム性について述べる．

キーワード: リアルタイムオペレーティングシステム, JavaVM, ガーベジコレクション

# Java Virtual Machine on Easel Real-Time Operating System

Gen Okuyama[†]　Eiji Takimoto[†]　Masahito Shiba[†]　Eiji Okubo[††]

[†]Graduate School of Science and Engineering, Ritsumeikan University
[††]Faculty of Science and Engineering, Ritsumeikan University

We have been developing a Java VM (Virtual Machine) on Easel real-time operating system. Generally speaking, to use Java in real-time systems is improper from the following reasons: overhead of class loading, necessity of garbage collection, and lack of function for timing guarantee. To solve these problems, we have been developing the Java VM, which is extented for real-time systems, on Easel real-time operating system.

**Key words:** real-time operating system, JavaVM, garbage collection

# 1　はじめに

近年，Java を組み込みシステムに適用しようとする動きが活発になってきている．Java は，マルチプラットホーム，ソフトウェア生産性の向上などの利点を持ち，今後もさまざまな分野で用いられると予想される．しかし，リアルタイム環境において Java を適用するには，いくつかの問題点が挙げられる．すなわち，クラスのダイナミックローディングによりオーバヘッドが発生すること，ガーベジコレクション (以下，GC) が必要であること，また JavaVM に時間制約を保証する機能がないことなどである．これらのことから，Java はリアルタイム性に欠けているといわれている．我々は，この問題点を考慮したリアルタイム JavaVM(以下，RT-JavaVM) の設計を行った．RT-JavaVM は，以下の機能を提供することにより，Java アプリケーションのリアルタイム応答性を向上させる．

- クラスロード
  アプリケーション実行前に必要なクラスをロードすることにより，プログラム実行中にクラスのロードにかかる時間を削減する．
- リアルタイム GC(以下，RT-GC)
  GC プリエンプション，および GC の優先度の動的な変更により，Java アプリケーションのみならずシステム全体のリアルタイム性を考慮する．また，RT-JavaVM は，GC の起動条件，周期などを設定するためのクラスライブラリを提供する．
- スレッド管理
  OS のシステムコールを用いるためのクラスライブラリを提供する．ユーザは，これを用いて Java スレッドを生成する．また，Java スレッドを OS のネイティブスレッドと 1 対 1 にマッピングすることにより，OS は，Java スレッドを意識することなくスケジューリングを行うことが可能となる．

我々は，RT-JavaVM を現在開発を進めているリアルタイムオペレーティングシステム Easel[1] へ実装し，評価を行った．本研究で提案する手法を用いることにより，リアルタイム環境において Java を適用した際のシステム全体のパフォーマンスの向上を得ることができた．

本稿では，2 章で Easel における RT-JavaVM の構成，3 章でスレッド管理，4 章で RT-GC の優先度，5 章でリアルタイム性の評価について述べる．また，6 章で関連研究について述べ，最後に 7 章でまとめと今後の予定について述べる．

# 2　Easel における RT-JavaVM の構成

Easel における RT-JavaVM の実装モデルを図 1 に示す．図 1 の RT Process は，Easel のネイティブプロセスであり，マルチメディアアプリケーションを想定している．Easel のネイティブプロセスは，複数個存在する．クラスライブラリは，JDK 相当のクラスに加え，マルチメディアアプリケーションを操作可能とするためのクラスライブラリを提供する．

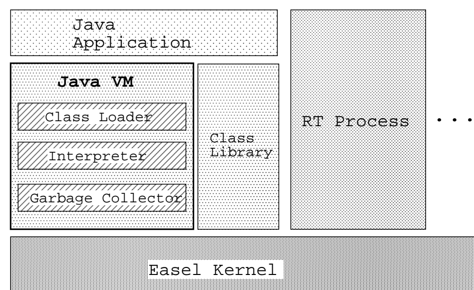


図 1　Easel における RT-Java VM の実装モデル

RT-JavaVM は，Easel 上で 1 つのプロセスとして動作する．RT-JavaVM は，インタプリタ，クラスローダ，ガーベジコレクタから構成される．以下の節では，これら各機能について述べる．

## 2.1　クラスローダ

クラスローダは，クラスをロードし，その中で使用しているクラスやそのメソッドの参照情報を整理する機構である．Java では，プログラムの実行に必要最小限のクラスを必要にあわせて動的にロードし実行する [2][3]．その際，JavaVM は，プログラムの実行を中断し，クラスのロードに専念する．また，クラスのロードにかかる時間は予測が困難であるため，アプリケーションの最悪実行時間を予測することは困難となる．

RT-JavaVMの内部構成を図2に示す．図2中の数字は，以下の数字と対応している．

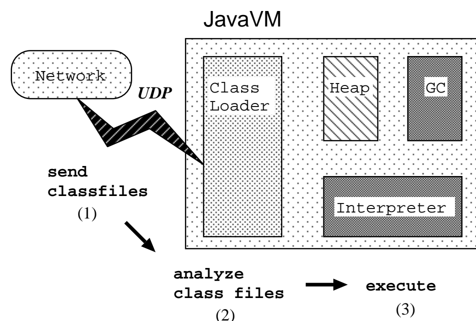


図2　RT-JavaVMの内部構成

(1) UDPを用いてEaselへクラスファイルを送信する．

(2) クラスファイルをクラスローダで解析(クラス解決を含む)する．

(3) インタプリタで実行する．

RT-JavaVMでは，上記の問題を解決するために，アプリケーション実行前に，(1)，(2)の処理を行う．これにより，アプリケーション実行時の，クラスロードにかかる予測が困難な時間を削減することが可能である．このことから，Javaアプリケーションの最悪実行時間を予測することが容易となる．

### 2.2　インタプリタ

JavaVMは，クラスローダによって読込まれたクラスをメモリに展開し，そのクラスのメソッドの中に埋め込まれているインストラクションコード(バイトコード)を実行する．RT-JavaVMでは，インタプリタ方式を用いてメソッドを実行する．

### 2.3　GC

GCは，不要となったオブジェクト(ガーベジ)を回収し，これを解放するための機構である．従来のJavaVMでは，GCの実行中はプリエンプト禁止としている．また，ヒープの状態によりGCの実行時間は変化するため，その時間を予測することは困難である．このことから，リアルタイム応答性を保証することが困難となる．以下では，RT-GCの特徴について述べる．

#### 2.3.1　プリエンプト可能なGC

我々は，GCをスレッドとして実現し，プリエンプト可能なRT-GCの設計，実装を行った．RT-GCのアルゴリズムとして，マークアンドスイープ方式[4][5]を採用した．この方式は，参照されているオブジェクトにマークを付けるマークフェーズ，マークされていないオブジェクトを解放するスイープフェーズという2つのフェーズから構成される．

また，GCがプリエンプト可能であるだけでは不十分である．GCスレッドの実行中に他のスレッドにプリエンプトされた場合，オブジェクトのマーク状態に不整合が発生する可能性がある．そのため，RT-JavaVMでは，ガーベジコレクタだけでなくスレッドも自身が参照するオブジェクトにマークするという方法をとっている．これにより，RT-GCが中断している際にスレッドが新しいオブジェクトを参照しても，マーク状態に不整合が発生することはない．

#### 2.3.2　RT-GCのタイミング

Javaスレッドを実行する際，RT-GCスレッドは，常時固定優先度で実行するより，ヒープの状態に応じた優先度で実行することが望ましい．これは，ヒープに余裕がある状態でもGCが起動し，その処理に妨げられるスレッドが存在する可能性があるからである．

しかし，この方法は，Javaスレッドのみを考慮した場合は有効であるが，JavaスレッドとEaselのネイティブスレッドを共存させるには適していない．例えば，JavaスレッドよりEaselのネイティブスレッドを優先すべき際，ヒープ中にガーベジが多くの割合を占めている場合であっても，RT-GCスレッドを起動することは望ましくない．そのため，RT-GCスレッドを起動する際には，ヒープ中のガーベジの割合だけでなくシステム全体のリアルタイム性を考慮する必要がある．このことから，RT-GCスレッドの優先度は，オブジェクトを生成したJavaスレッド全体の優先度の範囲内に設定されることが望ましい．しかし，ヒープの状態とシステム全体の優先度を考慮し，適切な優先度やタ

イミングで RT-GC スレッドを動作させることは非常に困難である．

そこで，RT-JavaVM では，オブジェクトを生成した Java スレッドの優先度に基づき，RT-GC スレッドの優先度を動的に変更する手法をとっている．本手法の詳細は，4 章で述べる．

## 3 スレッド管理

RT-JavaVM は，Java プログラムから Easel のシステムコールを使用するためのクラスライブラリを提供する．ユーザは，これを用いて Java スレッドを生成する．また，OS は，Java スレッドを意識することなく，Easel のネイティブスレッドと同等に管理する．

Easel では，非リアルタイムスレッドに対してはラウンドロビンスケジューリング，リアルタイムスレッドに対しては優先度を考慮した EDF スケジューリングを採用している．スレッドは，優先度順にキューにつながれており，同一優先度においてはデッドライン順につながれている．スレッドが新しく生成される場合，create_thread() システムコールにより非リアルタイムスレッドとして生成される．生成されたスレッドは，必要に応じて setrtparam() システムコールにより優先度，デッドラインを設定し，自身をリアルタイムスレッドへ遷移させる．

RT-JavaVM では，リアルタイム Java スレッドを生成するために，RTThread クラスを提供する．Java スレッドクラスの概要を以下に示す．

- コンストラクタ
  public **RTThread()**
- メソッド
  public int **setrtparam(int priority, int deadline)**

Java スレッド生成のプログラミング例を以下に示す．また，その際の処理の流れを以下に示す．

```
public class JavaThread extends RTThread{

    public void run(){
        t1.setrtparam(10,10);
        スレッドの処理
    }
    public static void main(){
        RTThread t1 = new RTThread();
        main の処理
    }
}

public class RTThread{
    private int priority;
    private int deadline;

    RTThread(){
        create_thread();
    }
    private native int create_thread();

    int setrtparam(priority, deadline){
        this.priority = priority;
        this.deadline = deadline;

        setrtparam();
    }
    public native int
        setrtparam(priority, deadline);

}
```

(1) RTThread オブジェクト生成時には，コンストラクタ内で create_thread() を呼び出し，スレッド生成を要求する．

(2) スレッド生成のシステムコールを受けた Easel は，スレッドを生成する．このスレッドは，Java スレッドにマップされる．

(3) 生成された Java スレッドは，その本体である run メソッドを実行する．

(4) Java スレッドは，RTThread クラスのメソッドである setrtparam() により，優先度，デッドラインを設定する．

生成された Java スレッドは，図 3 に示すように，Easel のネイティブスレッドと 1 対 1 にマッピングされる．これにより，Easel のスケジューラは，Java スレッドを意識することなくスケジューリングを行うことが可能となり，Easel は，Java スレッドを自身のネイティブスレッドと同等に管理する．

## 4 RT-GC の優先度

2.3.2 節で述べたように，Java スレッドと Easel のネイティブスレッドを共存させる際は，システム全体の優先度を考慮し，RT-GC を動作させる必要がある．RT-JavaVM では，オブジェクトを生成

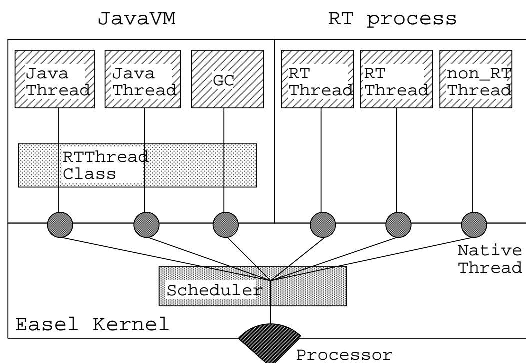


図 3　Easel のネイティブスレッドへのマッピング

した Java スレッドの優先度に基づき，RT-GC スレッドの優先度を動的に変更する手法をとっている．この手法を適用した際，RT-GC スレッドの優先度は以下のように変更される．

- オブジェクト生成の際，オブジェクトを生成した Java スレッドの優先度を保持していく．
- RT-GC スレッドは，起動時に，保持された値の平均値を自身の優先度とする．

この手法を用いることによって，RT-GC スレッドの優先度は，オブジェクトを生成した Java スレッドの優先度の範囲内に決定される．また，RT-GC スレッドは，起動するたびに異なる優先度で動作することになる．本手法は，システム全体の優先度を考慮し，また，動的なスレッドセットに対応した RT-GC スレッドの優先度の決定手法であるといえる．

### 4.1　RTGC クラス

RT-JavaVM では，RT-GC スレッドに関する設定を行うインタフェースとして RTGC クラスを提供する．RTGC クラスは以下のメソッドを提供する．

public int **setParameter(int period, int priority, int object_num)**

setParameter() は，RT-GC スレッドの周期，優先度，および規定オブジェクト数の設定を行う．規定オブジェクト数とは，オブジェクトがいくつ生成されたら RT-GC スレッドを起動するかという値である．ユーザがそれぞれの引数にマイナスの値を設定した場合，周期と規定オブジェクト数には RT-JavaVM が用意する初期値が設定され，優先度には前節の機構を用いて算出された値が設定される．このクラスを用いることにより，ユーザの意思に沿った GC の動作が可能となる．

### 4.2　動作例

RT-GC の動作例を図 4 に示す．Java Thread1 は優先度 10，Java Thread2 は優先度 20 である．また，Java Thread1，Java Thread2 は，共にオブジェクトを 10 個生成するスレッドである．また，setParameter() を用いて規定オブジェクト数を 10 と設定してあるとする．すなわち，RT-GC スレッドは，オブジェクトが 10 個生成されると起動される．また，図中の () 内の数字は生成したオブジェクトの数を示している．

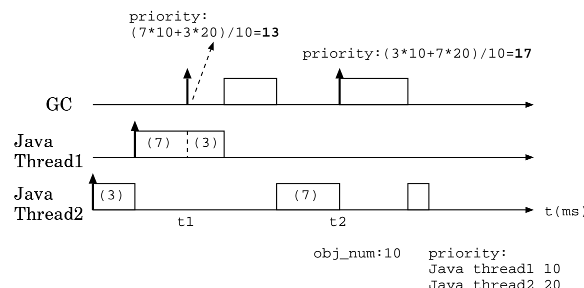


図 4　RT-GC の動作例

この例では，t1 で GC の優先度は 13 と設定され，t2 では 17 と設定される．このように，RT-GC スレッドの優先度はオブジェクトを生成した Java スレッド全体の範囲に設定され，RT-GC スレッドは起動するたびに異なる優先度で動作する．

## 5　評価

前章の，RT-GC スレッドの優先度を動的に変更する手法を用い，RT-JavaVM を Easel に適用した際の評価を行った．評価環境としては，Pentium 200MHz を搭載した PC/AT 互換機を用いた．評価に用いたスレッドセットを表 1 に示す．Easel のネ

イティブスレッドと Java スレッドは共に 3 つずつ動作し，RT-GC スレッドは，RTGC クラスによって設定された obj_num，または自身の周期によって起動する．ここでは，RT-GC スレッドは，RTGC クラスによって obj_num が 100 と設定されたとする．すなわち，RT-GC スレッドは，オブジェクトが 100 個生成されたら起動する．RT-GC スレッドが動作しない状態での CPU 利用率は，90%である．なお，周期とデッドラインは同じとする．

表 1　スレッドセット

| スレッド | 優先度 | 周期 | 実行時間 |
|---|---|---|---|
| easel_1 | 10 | 100ms | 15ms |
| java_1 | 15 | 100ms | 15ms |
| easel_2 | 20 | 100ms | 15ms |
| java_2 | 25 | 100ms | 15ms |
| easel_3 | 30 | 100ms | 15ms |
| java_3 | 35 | 100ms | 15ms |
| gc | – | 1000ms | – |

この評価をするにあたって，以下の 2 つのケースを対象とした．

- ケース 1: RT-GC スレッドの優先度を最も高優先度の Java スレッドに合わせる，すなわち RT-GC スレッドの優先度を 15 として実行した場合
- ケース 2: RT-GC スレッドの優先度をオブジェクトを生成した Java スレッドの優先度にともない，動的に変更する手法を用いた場合

ケース 1 では，システム全体のデッドラインミス率は 13.7%であった．これに対し，ケース 2 では 10.3%であった．各スレッドのデッドラインミス率を表 2 に示す．

表 2　デッドラインミス率

| スレッド | ケース 1 | ケース 2 |
|---|---|---|
| easel_1 | 0% | 0% |
| java_1 | 0% | 0% |
| easel_2 | 7.2% | 0% |
| java_2 | 14.0% | 0% |
| easel_3 | 16.4% | 20.4% |
| java_3 | 17.2% | 20.8% |

ケース 1 では，easel_2 にデッドラインミスが発生した．これは，Java スレッドが生成したオブジェクトを回収する GC の処理のためである．このように，GC の処理によって中間優先度である easel_2 にデッドラインミスが発生することは好ましくない．これに対し，ケース 2 では，easel_3 と java_3 のデッドラインミス率は高くなっているが，easel_2 にデッドラインミスは発生しなかった．この結果より，GC は，オブジェクトを生成した Java スレッドの優先度に近い値で動作する必要があるといえる．

また，RTGC クラスを用いて obj_num を変更することで，RT-GC スレッドが起動するタイミングを変更した．システム全体のデッドラインミス率を図 5 に示す．

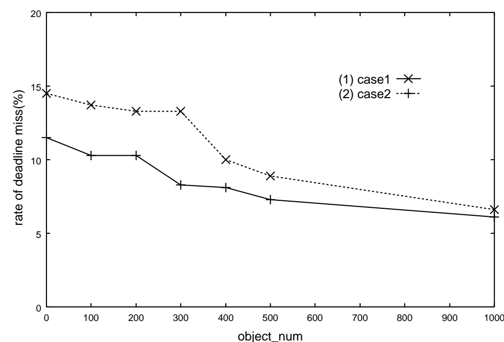


図 5　システム全体のデッドラインミス率

このように，いずれの場合も，ケース 2 の方がデッドラインミス率が低くなっていることがわかる．また，ケース 1，ケース 2 は，共に obj_num の値が高くなるにつれ，システム全体のデッドラインミス率が低くなった．これは，RT-GC スレッドが起動する回数が減り，RT-GC スレッドに処理を妨げられるスレッドが減ったためである．また，obj_num の値が 500 を越えると，デッドラインミス率にあまり変化が見られなかった．しかし，ケース 1 においては，obj_num を変更しても easel_2 のデッドラインミスを防ぐことができなかった．この結果からも，本手法の有用性がわかる．

本評価では，ヒープの状態を考慮していない．そのため，オブジェクトを生成することができない状況を防ぐために，ヒープの状態を考慮し，RT-GCスレッドを適切なタイミングで起動させる必要がある．

## 6 関連研究

リアルタイム JV-Lite[6] では，GC を周期スレッドとして実行している．また，GC の処理により不必要に CPU 負荷をかけ，システム全体のパフォーマンス低下を防ぐために，メモリの空き状況に応じて GC スレッドのタイムスライス時間，および周期を動的に変更している．しかし，これは Java スレッドのみを考慮した方法である．リアルタイム OS のネイティブスレッドを考慮した場合，本研究で提案した方法のように，システム全体の優先度を考慮する必要がある．

The Real-Time Specification for Java (RTSJ)[7] では，GC に関するクラスライブラリとして GarbageCollector クラスを提供している．ユーザは，このクラスを用いることにより，以下を得ることが可能である．

- ガーベジの最大回収率
- プリエンプション遅延
  Java スレッドは，GC をプリエンプトする際，安全な時点に到達するまで待たなければならない．プリエンプション遅延は，Java スレッドが待たなければならないおおよその最大時間を示す．

本研究では，RTGC クラスを提供することにより，RT-GC スレッドの周期，優先度，および RT-GC が起動する条件の設定が可能である．これにより，ユーザの意思に沿って RT-GC を動作させることが可能となる．

## 7 おわりに

本稿では，Easel における RT-JavaVM の構成，および RT-GC の評価について述べた．RT-JavaVM では，RTThread クラスを提供することにより，ユーザは，これを用いて Java スレッドを生成することが可能である．OS は，Java スレッドを意識することなくネイティブスレッドと同等に管理する．また，GC をスレッドとして実現し，プリエンプト可能な RT-GC を提供する．本研究で提案した，動的に RT-GC の優先度を変更する手法を用いることにより，システム全体の優先度を考慮した GC が可能となった．今後の予定としては，以下が挙げられる．

- デッドラインを考慮した RT-GC
- マルチメディアインタフェースのサポート
  MP3，MPEG 動画をサポート可能なインタフェースの設計を目標とする．

## 参考文献


[1] 谷出新: “マルチメディア処理に適したリアルタイムオペレーティングシステム Easel の設計と実装”，立命館大学大学院理工学研究科 修士論文 (2001).

[2] Jon Meyer，Troy Downing 共著，鷲見 豊 訳: “Java バーチャルマシン”，株式会社オライリー・ジャパン (1997).

[3] ティム・リンドホルム，フランク・イェリン 共著，野崎 祐子 訳: “The Java 仮想マシン仕様”，バリエ社 (1997).

[4] “GC FAQ – algorithms (Basic Algorithms)”，http://www.iecc.com/gclist/GC-algorithms.html#Jargon.

[5] ケン・アーノルド，ジェームズ・ゴスリン，デビッド・ホームズ共著，柴田 芳樹 訳: “プログラミング言語 Java 第 3 版”，株式会社ピアソン・エデュケーション (2001).

[6] “Java™ のリアルタイム拡張の動向”，http://www.jerty.com/40_press/99/CSPY-RTP99-3.html.

[7] ボレラ，ゴスリン，ブロスゴル，ディビル，フウ，ハーディン，ターンブル 共著，柴田 芳樹 監訳: “Java リアルタイム仕様”，株式会社ピアソン・エデュケーション (2000).