OpenVMS 上での Java™ アプリケーションの性能の最適化

ユーザのためのヒント集

2004 年 10 月

このドキュメントでは，*Java™ Platform* SDK (Software Development Kit) for the OpenVMS Alpha Operating System の性能を最適化する方法について説明します。

オペレーティング・システム HP OpenVMS Alpha バージョン 7.2-1 以降

ソフトウェア・バージョン *Java™ Platform* SDK (Software Development Kit) v 1.3.1-1 for the OpenVMS Alpha Operating System (またはそれ以降)

日本ヒューレット・パッカード株式会社

原典：Optimizing Java Technplogy Software Performance on HP OpenVMS - TIPS AND TRICKS FOR USERS

目次

本書の目的

本書の目的は，OpenVMS Alpha 上で SDK および SDK ベースのアプリケーションの性能を最適化するための実用的なヒントとガイドラインを説明することです。

本書の内容は，OpenVMS エンジニアの経験をまとめたもので，SDK の実践的な経験を元にしています。

本書を利用することで，OpenVMS SDK 環境とアプリケーションを体系的に最適化できるようになります。

注意事項

本書は 2003 年に執筆されたドキュメントを翻訳したものです。その後，新しい SDK およびオペレーティング・システムがリリースされていますが，本書の内容は執筆時点のままとします。

最新の Java SDK は下記の Web サイトからダウンロードできます。

http://h18012.www1.hp.com/java/download/index.html

新しいバージョンの SDK をご利用の際には，各バージョンの SDK に含まれているリリース・ノートに記載されている情報もご確認ください。

マーク

設定関連

SDK **アプリケーションの開発**

SDK **アプリケーションの実行**

SDK **アプリケーションのトラブルシューティング**

1. SDK のチェックリスト

最初に SDK のハードウェア要件とソフトウェア要件を満たしていることを確認し，次にアプリケーションに関する質問を見ていき，各自の状況に当てはまるものがないか確認します。

ハードウェア *どのようなシステムを使っていますか?*	**性能**		
	最適	**平均的**	**低い**
? CPU	500MHz以上	500MHz	500MHz 以下
? **システム・メモリ**	1GB 以上	512MB	最低 256MB

ソフトウェア

どのバージョンのソフトウェアを使っていますか?

? OpenVMS Alpha

✓ OpenVMS の最新版は 7.3-1 です。バージョン 7.2-1 以降を使用する必要があります。

? *Java™ Platform* SDK for OpenVMS Alpha Operating System

✓ *Java™ Platform* SDK for OpenVMS Alpha Operating System の最新版は，SDK (Software Development Kit) v 1.4.1 です。
バージョン 1.3.1-1 以降を使用する必要があります。

Java **ソフトウェアのダウンロード・ページから入手できます。**
http://h18012.www1.hp.com/java/download/index.html

? C **ランタイム・ライブラリ**

✓ C RTL 用の ECO の最新版を適用してください。

(ソフトウェア・パッチ (ECO) のダウンロード・ページで，`DEC-AXPVMS-VMS<Version>_ACRTL` のバージョン番号が最も大きなものを探します。)

2002 年 6 月の時点では，利用できる最新の ACRTL イメージの名前は，以下のようになっています。

OpenVMS 7.2-1 用
`DEC-AXPVMS-VMS721_ACRTL-V0400--4`
OpenVMS 7.2-1H1 用
`DEC-AXPVMS-VMS721H1_ACRTL-V0300--4`
OpenVMS 7.2-2 用
`DEC-AXPVMS-VMS722_ACRTL-V0100--4`
OpenVMS 7.3 用
`DEC-AXPVMS-VMS73_ACRTL-V0200--4`

ソフトウェア・パッチ (ECO) **ダウンロード・ページから入手できます。**
http://ftp.support.compaq.com/patches/.new/openvms.shtml

? TCP/IP **サービス**

✓ 5.0A 以降を使用し必須パッチを適用している必要があります。OpenVMS 用の最新版は 5.3 ECO 1 です。

TCP/IP Services for OpenVMS **のページに情報があります。**
http://h71000.www7.hp.com/network/tcpip.html

? **オペレーティング・システムの** ECO

✓ SDK のインストールと実行に必要です。

SDK v1.3.1-3 **または** 1.4.1 **の** Patch Installation **のページを参照してください。**
http://h18012.www1.hp.com/java/download/ovms/1.3.1/sdk1.3.1_patches.html
http://h18012.www1.hp.com/java/download/ovms/1.4.1/sdk1.4.1_patches.html

SDK アプリケーションの開発

開発中の SDK アプリケーションは以下の項目に該当しますか?

? fork/exec によるサブプロセスを使用する

✓ 「親子プロセスの最適化」の「メールボックス値を増やす」を参照してください。

? 子プロセスからのデータを処理する

✓ 「親子プロセスの最適化」のUNIXパイプのエミュレートを参照してください。

SDK アプリケーションの実行

現在実行している SDK アプリケーションは以下の項目に該当しますか?

? メモリ量が異なるさまざまなシステムで実行する

✓ 「メモリの管理」の「ヒープ値の動的な設定」を参照してください。

? Fast VM での動作が「クラシック VM」よりも遅い

✓ 「メモリの管理」の「Fast VMのメモリ使用量」を参照してください。

? 大きなオブジェクトをすぐに再利用する

✓ 「性能に関するその他のヒント」の「ガベージ・コレクション時間の短縮」を参照してください。

? ODS-5 ボリューム上でファイルを排他的に使用する

✓ 「メモリの管理」の「ファイル名変換 (マッピング) のオーバヘッドを削減する」を参照してください。

? 小規模なクライアント・サイド・アプリケーションである

✓ 「メモリの管理」の「クライアント・サイド・アプリケーションの最適化」を参照してください。

アプリケーションのトラブルシューティング

現在起こっている問題は，以下のどの説明に該当しますか?

? 論理名変換の比率が高い

✓ 「性能に関するその他のヒント」の「非ユーザ・モードの削減」を参照してください。

? 「Exec モード」の比率が高い

✓ 「性能に関するその他のヒント」の「非ユーザ・モードの削減」を参照してください。

? J2EE サーバでページングやページ・フォルトが多発する

✓ 「性能に関するその他のヒント」の「ヒープ値を増やす」を参照してください。

? 監視対象ファイルの変更を常にポーリングしている

✓ 「性能に関するその他のヒント」の「キャッシュ間隔の拡大」を参照してください。

? ファイル名の過剰なマッピング

✓ 「性能に関するその他のヒント」の「ファイル名マッピングの削減」を参照してください。

? 過剰な stat() 呼び出し

✓ 「性能に関するその他のヒント」の「2 度目のstat() 呼び出しの回避」を参照してください。

? 過剰なファイル共用

✓ 「メモリの管理」の「ヒープ値の動的な設定」を参照してください。

? 大規模なディレクトリの読み込み

✓ 「性能に関するその他のヒント」の「バージョン数の制限」を参照してください。

2. メモリの管理

利用できるシステム・メモリを管理することは，SDK アプリケーションの性能を最大限に向上させる上で，最も有効な手段です。ここでは，Alpha システムでのメモリ割り当てを最適化する方法を説明します。最初のポイントは，十分な物理メモリを搭載することです。Java アーキテクチャ，特に JVM (Java Virtual Machine)では，システム・リソースに高い負荷がかかるためです。

適切なプロセス・クォータの設定

JVM は，UNIX システムで最適な性能が得られるように設計されています。UNIX では，デフォルトで各プロセスに大きなクォータが割り当てられています。 しかし OpenVMS では，デフォルトでは各プロセスに小さいクォータが割り当てられ，システム上で多数のプロセスが共存できるようになっています。OpenVMS 上で最高の SDK 性能を得るには，プロセス・クォータを一般的な UNIX システムと同じにします。多くの場合，これらは最低限必要なクォータ設定になっています。

✓推奨内容
以下の数値は，デフォルトの UNIX のプロセス・クォータに近い値です。

UAF FILLM	4096
CHANNELCNT	4096
WSDEF	2048
WSQUOTA	4096
WSEXTENT AND WSMAX	16384
PGFLQUO	2097152
BYTLM	400000
BIOLM	150
DIOLM	150
TQELM	100

CHANNELCNT の算出

CHANNELCNT は，UAF クォータではなく，SYSGEN パラメータです。したがって，変更内容を有効にするためにはリブートが必要です。SDK プロセスは，UAF クォータ FILLM と SYSGEN パラメータ CHANNELCNT の小さい方の値に従うことに注意してください。

PGFLQUO の算出

PGFLQUO の適切な値は，ヒープ・サイズに 2 を掛けた値です。たとえば，128MB (2*128*1024*1024)/512 = 524288 となります。
RTE（Runtime Environment）を使用する場合には，PGFLQUO の推奨する最小値は 96MB です。
PGFLQUO パラメータを増やす場合には，新しい PGFLQUO パラメータに合わせて，システムのページ・ファイル・サイズも大きくする必要があります。

本項の「ヒープ値の動的な設定」も参照してください

Fast VM のメモリ使用量

Fast VM は，大容量のメモリを搭載したシステム向けに最適化されています。つまり，いくつかのトレードオフが判断され，メモリ使用量よりも速度が優先されています。その結果，Fast VM では，同じアプリケーションでもより多くの物理メモリと仮想メモリを使用します。従来の SDK JIT (「クラシック VM」)より 50 パーセントも多くなることがあります。その結果，システムが正しくチューニングされていなかったり，システムの物理メモリが不足していると，過度のページングが発生し，性能が低下することになります。

「性能に関するその他のヒント」の「Fast VM の設定」も参照してください

✓推奨内容
クラシック VM と比べて，Fast VM でのアプリケーションの動作が遅いと感じる場合は，以下の対策を実行する必要があります。

- -Xmxおよび -Xmsの値について， Fast VMがデフォルトで選択する値ではなく，明示的に値を指定してみます。

本項の「 ヒープ値の動的な設定」も参照してください

- 利用可能な物理メモリの量を基に，プロセス・クォータを増やします。

本項の「適切なプロセス・クォータの設定」も参照してください

- システムで過剰なページ・フォルトが発生していないか調べます。
- システムの未使用ページ・リストを監視して，物理メモリを追加する必要があるかどうかを判断します。

経験則
物理メモリが不足している場合は，最大ヒープ (-mx<n>m) 値を小さくすることで改善されます。これによりガベージ・コレクションが増え，ヒープ・サイズが大きいことによる過剰なページ・フォルトが発生しなくなります。

次の場所にある『OpenVMS Performance Management』マニュアルを参照してください
http://h71000.www7.hp.com/doc/72final/6491/6491PRO.HTML

ヒープ値の動的な設定 (または静的な設定)

メモリ割り当てプール (SDK ヒープ) のデフォルト設定として固定値を使用するのではなく，実行環境に応じて，Fast VM が動的に初期ヒープサイズ (-Xms) と最大ヒープサイズ (-Xmx) を決めるように指定することができます。

Fast VM では，以下の計算式を使用します。

max_memory = プロセスが利用できる物理メモリと総メモリの小さい方*
デフォルトの初期ヒープサイズ= *max_memoryの10%*
デフォルトの最大ヒープサイズ = *max_memoryの 60%*

* 物理メモリ・サイズと PGFLQUO (ここでは WSMAX は計算に入れません)。

このようにして，Fast VMでは利用できるメモリ量に基づいてヒープ・サイズを調整します。これにより，-Xmxおよび-Xmsに固定値を指定するよりも，一般には良い結果となります。特に，メモリ量が異なるさまざまなシステムで実行するアプリケーションで有効です。

✓推奨内容
状況によっては，Fast VMがデフォルト値を決めるのではなく，-Xmxおよび-Xmsに対して独自に適切な値を指定した方が良い結果が得られる場合もあります。指定する値を決めるには，コマンド行オプション -verbose:gcを使って，アプリケーションのヒープの動作状況を監視します。

短期間に多数のガベージ・コレクションが発生しているようであれば，過度のページ・フォルトが発生しないぎりぎりまでヒープ・サイズを増やします。

注意
最大メモリ・サイズは，利用できる物理メモリ・サイズより絶対に大きくしないでください。大きくすると，確実に性能に悪影響が出ます。

クライアント・サイド・アプリケーションの最適化

Fast VM は，サーバ上で動作する，大規模で長時間実行するようなプログラム向けに最適化されています。Fast VM をワークステーション上でクライアント・サイド・アプリケーション用に使用する場合は，必要なリソースの量を減らすことができます。クライアント向けの構成では，Fast VM のメモリ専有量が大幅に減ります。

✓推奨内容

`-client` スイッチを指定して Fast VM を実行します。これは，次のようにスイッチを指定するのと同じです。

```
java -Xmx64m -Xglobal128m -Xgc:compacting
```

`-client` スイッチを構成する個々の設定を置き換えてもかまいません。たとえば，`java -client -Xmx256m` と指定すると，最大ヒープ・サイズが `256MB`，最大グローバル・リージョン・サイズが `128MB` で，コンパクティング・コレクタを使用するように Fast VM が初期化されます。

J2EE アプリケーション・サーバによるページングの削減

BEA WebLogic ServerなどのJ2EE アプリケーション・サーバが稼働している場合は，ページングを減らすことで，性能を向上させることができます。ページングは通常のシステム動作で発生しますが，J2EE アプリケーション・サーバが動作している最中のページングは性能低下をもたらすことがあります。

✓推奨内容

`WSMAX` や `WSEXTENT` の設定値が小さすぎると，ページ・フォルトが発生します。次の計算式を使って，`WSMAX(SYSGEN)`および `WSEXTENT(AUTHORIZE)`を正しく設定してください。

`(MaxJavaHeapSize + MaxJavaHeapSize * 20%)`をページ・サイズ`8KB`で割る

例
`MaxJavaHeapSize` が `–X-mx512m (512MB` メモリ`)`の場合，次のようになります。

```
(512MB + 512MB * 0.2)/8192B = 78643 (8KB ページの数)
```

注意
ヒープ・サイズは物理メモリより大きくしないでください。ヒープ用に十分な物理メモリが利用できないと，アプリケーションでページングが頻繁に発生します。ページングが発生すると，性能が大幅に低下します。

システムのページ・フォルト率を監視するには，`$ MONITOR SYSTEM` を使用します。

ファイル名変換 (マッピング) のオーバヘッドを削減する

RTEでは，OpenVMS上で直接表現できないファイル名を，表現できるファイル名に変換するようになっています。このファイル名マッピングでは，スタック上にいくつかの (大量になる可能性がある) バッファを必要とします。要求されるマッピングが増えると，必要なバッファの数も増えます。

(最新のC RTL ECOでは，OpenVMS AlphaのODS-5ボリュームで直接指定できるファイル名の種類が増えています。)

✓推奨内容

アプリケーションが ODS-5 ボリューム上でファイルを排他的に使用していて，システムに最新のC RTL ECOを適用している場合には，ファイル名マッピングのオーバヘッドを減らすことができます。ファイル名マッピングの機能を削除すると，割り当てる必要のある内部バッファの数が減り，アプリケーションのメモリ専有量が少なくなります。
ファイル名のマッピングが不要な場合には，JAVA$FILENAME_CONTROLSに 0 を指定して，スタック・サイズ値 (-Xss<number>) を小さくすることができます。これで，メモリ使用量が減りスレッドの作成が速くなります。

JAVA$FILENAME_CONTROLS の使用方法の詳細は，「性能に関するその他のヒント」の「ファイルのチューニング: ファイル名マッピングの削減」を参照してください

3. 性能に関するその他のヒント

ここでは，アプリケーションの性能に関するトラブルシューティングの際に一般的に有効な，その他のヒントを説明します。各項目を検討し，各自の状況に該当するかどうか判断してください。

一般的なチューニング

Fast VM の設定

✓推奨内容

最高の性能を得るためには，アプリケーションが「クラシック VM」を必要とする理由がないかぎり，Fast VM (仮想マシン) を使用すべきです。

OpenVMS Alpha 上の Fast VM は SDK 実行時の性能が最適化されています (Fast VM は，SDK v1.3.1-2 以降の SDK キットに含まれています)。

Fast VM はデフォルトの VM ではないため，Fast VM のための設定が必要になります。

Fast VM の設定方法

次のコマンドでパラメータ FAST を使用します。

```
$ @SYS$COMMON:[JAVA$131.COM]JAVA$131_SETUP FAST
```

`java -version` コマンドを入力すると，次のようなメッセージが表示されます。

```
$ java -version
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition
Fast VM (build 1.3.1-n ...)
```

ここで，*n* はインストールされている SDK バージョンのリリース番号を表し，`Fast VM` は仮想マシンを表します。

参考： クラシック VM に戻すには，次のコマンドを入力します。

```
$ @SYS$COMMON:[JAVA$131.COM]JAVA$131_SETUP
```

『SDK Release Notes』の「Using the Fast VM」を参照してください

http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#usingfastvm
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#usingfastvm

非ユーザ・モードの削減

マルチCPUシステム上で大量のリソースを必要とするアプリケーションが動作している場合には，*非ユーザ・モード*のCPU時間の量を減らしてみてください。以下の2つの場合が該当します。

システムで論理名変換の比率が高くなっている場合

(確認には `$ MONITOR IO` コマンドを実行)

✓推奨内容

`DECC$ENABLE_GETENV_CACHE`論理名を有効にします。この論理名により，C RTLは `getenv()`関数が返したエントリをキャッシュするようになるため，`sys$trnlnm()`呼び出しの回数が減り，`getenv()`が非ユーザ・モードを使わずにユーザ・モードのみで動作するようになります。

注意
論理名を変更した場合，有効にするためにはアプリケーションの再起動が必要となります。

システムで「Execモード」が多くなっている場合

Execモードとは，カーネル・モード，エグゼクティブ・モード，スーパバイザ・モードを合わせたものを意味します。これらのモードに対してシステムの統計情報を確認するには，次のコマンドを入力します。

```
$ MONITOR SYS/ALL
```

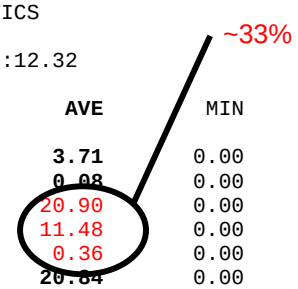
アプリケーションによっては，ファイルの作成，ディレクトリ内のファイルの存在監視，ファイル・メソッドを使ったファイル・プロパティのテストにかなりの時間を費やしているものがあります。このように監視対象ファイルの状態を常時取得するために，1秒間に数百回も`stat()`関数が無駄に呼び出される可能性があります。

例
以下は，CPU 時間の 1/3 近くが Exec モード (カーネル・モード，エグゼクティブ・モード，スーパバイザ・モード) で消費されているシステムの監視統計情報の例です。

```
OpenVMS Monitor Utility
                              SYSTEM STATISTICS
                                   on node
                            18-APR-2002 21:13:12.32

                                    CUR         AVE         MIN         MAX

Interrupt State                    4.83        3.71        0.00        7.16
MP Synchronization                 0.16        0.08        0.00        0.33
Kernel Mode                       26.50       20.90        0.00       37.00
Executive Mode                    13.50       11.48        0.00       24.83
Supervisor Mode                    0.66        0.36        0.00        0.83
User Mode                         25.00       20.84        0.00       39.33
Compatibility Mode                 0.00        0.00        0.00        0.00
Idle Time                        129.33      142.59       99.83      200.00
Process Count                     57.00       57.28       57.00       58.00
Page Fault Rate                    0.00        8.38        0.00      110.83
Page Read I/O Rate                 0.00        0.55        0.00       13.66
Free List Size                232502.00   232746.15   232111.00   233402.00
Modified List Size             47748.00    48753.55    47748.00    50740.00
Direct I/O Rate                    0.83        9.01        0.00      176.66
Buffered I/O Rate                797.00      621.12        1.00     1072.16
```



✓推奨内容
アプリケーションでこのような操作を行っている場合は，論理名`JAVA$CACHING_INTERVAL`を使用することで性能が向上します。これを使うと，`stat()`で収集した情報がキャッシュされます。

本項の「キャッシュ間隔の拡大」を参照してください

クラス数に応じたクラスパスの設定

ディレクトリ一覧を基にしてクラスパスを設定するのが一般的ですが，クラス数が多いものから少ないものの順でクラスパスを設定すれば，性能が向上します。

✓推奨内容
クラスの数が多いファイルをクラスパスの先頭に指定します。

例
`zz.jar` ファイルには300個のクラスが含まれており，`special.jar` ファイルには20個しかクラスが含まれていないとした場合，次のように，`zz.jar` をクラスパスの最初に指定します。

```
$ define JAVA$CLASSPATH zz.jar, special.jar
```

クラス・ファイルを検索する際，JVM は定義された順でクラスパスを検索します。したがってこの場合，zz.jar 内にクラスが見つかると，JVM はそこで検索を終了します。

警告
アプリケーションによっては，クラスパスが特定の順序になっていなくてはならないものがあります。たとえば，アプリケーションがメイン・クラスの前に ECO クラスを見つける必要がある場合には，クラスパスの先頭に ECO の jar ファイルを指定し，その次にメインの jar ファイルを設定します。

子プロセス

OpenVMSでのプロセスの作成は，UNIXでのプロセスの作成と比べて多くのリソースを消費します。

✓推奨内容
SDKで，つまり同じプロセス内でその機能を実行できる場合には，子プロセスを作成しないようにします。

Webアプリケーション・サーバによっては，JSP-to-servlet 形式からコンパイルする際に，子プロセスを生成するか，子プロセスを生成せずにクラスをコンパイルするかを選択できるものがあります。このようなオプションがある場合には，子プロセスを作成しないように設定します。

子プロセスを使わなくてはならない場合は，JAVA$FORK_PIPE_STYLE に 2 を指定して，デフォルトのバッファリング方式をメールボックス・ベースからソケット・ベースに変更することを検討してください。

「親子プロセスの最適化」の「JAVA$FORK_PIPE_STYLEの使用」の項を参照してください

ガベージ・コレクション時間の短縮

大きなオブジェクト (複雑にネストした構造体や文字列など) をすぐに再利用するようなアプリケーションでは，ガベージ・コレクション時間が短いほうが有利です。SDK v1.3.1-1 以降の Fast VM で利用できるデフォルト以外のガベージ・コレクション方式を使うことで，その他のアプリケーションも性能が向上する可能性があります。

説明
SDK v1.3.1-1 では，従来のガベージ・コレクション方式に加えて，Fast VM 用の新しい方式が導入されました。この「コンパクティング・ガベージ・コレクタ」は，デフォルトのコレクタのようにデータをコピーするのではなく，使用中のデータはその場でコンパクト化します。マーク・スイープ/マーク・コンパクトのコレクション方式を使用することで，無駄なデータの移動を防ぎます。長時間使用されているデータが高い割合でヒープに残っているようなアプリケーションでは，性能特性が向上し，ヒープ・サイズも小さくて済みます。このコレクタは，使用中のデータの割合が適度な場合には，ヒープ全体のコレクションを行うのではなく，マイナー・コレクションを行うことも可能です。スイープを行って，データをまったく移動せずに領域を空けることもできます。したがって，短時間しか使用されないオブジェクトの割合が高い場合は別として，長時間使用されるデータが適度にあるアプリケーションでは性能が向上します。

✓推奨内容
上記のようなアプリケーションでは，次にように入力して，Fast VM を使ったコンパクティング・ガベージを試してみることをお勧めします。

```
-Xgc:compacting
```

参考： -Xgc:copying を指定すると，Fast VMはデフォルトの (「コピー方式の」) コレクタを使用します。

ファイルのチューニング

ファイル名マッピングの削減

SDKは，UNIX形式のファイル名をODS-2ファイル・システムに合わせるために，ファイル名のマッピングを行いますが，このマッピングの量は，論理名JAVA$FILENAME_CONTROLSによって決まります。デフォルトでは，ODS-2とODS-5のファイル構造がサポートされます。

✓推奨内容

最高の性能を得るためには，アプリケーションが必要とする機能だけを(JAVA$FILENAME_CONTROLS のビットの設定により)使用します。ファイル名マッピングが起きたときに，個々のマッピングが表示されるようにするには，論理名 JAVA$SHOW_FILENAME_MAPPING を次のように定義します。

```
$ DEFINE JAVA$SHOW_FILENAME_MAPPING 1
```

JAVA$FILENAME_CONTROLS のデフォルト値を変更するには，次の DCL ファイルを編集します。

```
$ SYS$COMMON:[JAVA$131.COM]JAVA$FILENAME_CONTROLS.COM
```

最高の性能

- ODS-5 ディスクだけを使って，JAVA$FILENAME_CONTROLS に 0 または 1 を定義します。

また

- 以下の論理名をサポートしている，最新のC RTL ECOを適用します。

```
DECC$ARGV_PARSE_STYLE
DECC$EFS_CASE_PRESERVE
DECC$EFS_CASE_SPECIAL
DECC$EFS_CHARSET
```

注意

アプリケーションには，100**パーセント**UNIX形式のファイル名とファイル・システムの使用が期待されています。

「メモリの管理」の「ファイル名マッピングの削減」も参照してください

アプリケーションにとってどのファイル名マッピングが適当か

付加的な SDK の変換をすべて無効にすれば最高の性能が得られますが，すべてのファイル名マッピングを無効にするのが実際的でない場合もあります。そこで，アプリケーションの要件を理解する必要があります。たとえば，ODS-2 ディスクからファイルを読み込む必要があるかといったことです。もし読み込む必要がある場合は，付加的なファイル名のマッピングが必要です。指定可能なファイル名マッピングの値を次の表に示します。

ファイル名マッピング

オプション	値
UNIX および VMS のファイル名のサポート	%x00000008
ファイル名中のディレクトリのサポート	%x00000200
ファイル名中の正当な文字のサポート	%x00001000
隠しファイル名のサポート (アンダスコアで置き換え)	%x00004000
隠しファイル名のサポート (ドットを削除)	%x00008000

ディレクトリ中の複数ドットのサポート (アンダスコアで置き換え)	%x00020000
ディレクトリ中の複数ドットのサポート (ドットを削除)	%x00040000
ファイル中の複数ドットのサポート。最後のものを保持	%x00100000
ファイル中の複数ドットのサポート。最初のものを保持	%x00200000
切り詰めによる，40 文字以上のサポート	%x04000000
ドットの移動による，40 文字以上のサポート	%x08000000
ディレクトリの再マッピングのサポート	%x20000000

『SDK Release Notes』の各設定の説明を再度確認し，どの設定や設定の組み合わせがアプリケーションに最適かを理解してください。
http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#unix_style
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#unix_style

キャッシュ間隔の拡大

アプリケーションによっては，以下のようなファイル・メソッドを使って，ファイルの作成，ディレクトリ内にファイルが存在するかの監視，ファイルのプロパティのテストにかなりの時間を費やすものがあります。

```
File file = new File(name);
file.exists()
file.isDirectory()
file.isAbsolute()
```

これらのメソッドでは，内部で C RTL 関数 `stat()`を間接的に呼び出しています。

典型的なシーケンスは，以下のようなものです。

```
if (file.exists()) { // stat()を呼び出すことになる
if (file.isDirectory()) // また stat()を呼び出すことになる
          ...
          }
          }
```

あるファイル群の状態を監視するため，1 秒間に何百回も呼び出しを行っている Web サーバもあります。そのような場合には，キャッシュ間隔を拡大し，連続的なポーリングを削減することで，全体的にかなり高速化される場合があります。

✓推奨内容

アプリケーションがこのような操作 (監視対象のファイルの変化を連続的にポーリングする) を行っていて，SDK v1.3.1-3 を使用している場合には，この最適化を試してください。

デフォルトでは，この最適化は有効になっていません。`JAVA$CACHING_INTERVAL`[1] 論理名に対して，キャッシュ間隔を示すゼロ以外の正の値(秒数)を指定することで有効になります。

たとえば，キャッシュ間隔を 1 分に設定するには，次のようにします。

```
$ DEFINE/JOB JAVA$CACHING_INTERVAL 60
```

`stat()`関数が収集した情報は，ここで示した期間だけキャッシュされ，それが過ぎると古いと判断されて，キャッシュは無効になります。

したがって，上記のコード例の問い合わせ `file.isDirectory()`では，キャッシュから情報が取得され，実際の入出力は行われません。これにより，速さに大きな差が出ます。

使用上の注意

以下の状態の後では，キャッシュは無効となり，`stat()`に対する実際の呼び出しが行われます。

[1] `JAVA$CACHING_INTERVAL`は，SDK 1.3.1-3 で新たに追加された論理名です。

- 現在のキャッシュ間隔が過ぎた。
- 現在のアプリケーションが，キャッシュされた情報を無効にする可能性がある，明示的な動作を行った場合。この動作には以下のものが含まれます。
  - ファイルが READ 以外でオープンされた。
  - ファイルまたはディレクトリが作成または削除された。
  - 子プロセスが Runtime.exec()により生成されたか，完了した。

キャッシュによりポーリングを減らしたときの主な欠点は，自分のアプリケーションとは協調動作しない他のアプリケーション (たとえば FTP) が，監視対象内にファイルをコピーしても，上記の理由のどれかによりキャッシュが更新されるまで，RTE がそれに気づかないという点です。

2 度目の stat()呼び出しの回避

OpenVMS Alpha 上でのファイル名マッピングの動作方法と，ドットを含むディレクトリの処理方法が原因で，探しているクラス・ファイルが最初の stat()呼び出しで見つからなかった場合には，RTE は再度 stat()関数を呼び出す必要があります。

例

他のプラットフォームから，ディレクトリ構造全体の中に project-3.1-A という名前のディレクトリを含むアプリケーションをインポートするとします。動作の中で，ディレクトリが存在することを確認する必要があります。ODS-2 を使っている場合，project-3.1-A という名前のディレクトリは表現できません。その結果，アプリケーションがディレクトリの存在をチェックすると，RTE は最初にその存在しないディレクトリ[projects.project-3.1-a]にアクセスしようとします。これが失敗すると，ディレクトリ名からすべてのドットを取り除いた変更後のパス[projects/project-3_1-A]を使って，再度 stat()を呼び出します。

説明

2 度目の呼び出しは，ドットを含むディレクトリのマッピングに関する OpenVMS の方式をサポートするために必要です。一般に，どのような理由で stat()が失敗したとしても，RTE は変更したパス名を使って再度 stat()を呼び出します。この動作が不要な場合は，論理名を使って抑制できます。

2 度目の stat()呼び出しを行うことの影響

一般に最新の SDK アプリケーションは，複数のベンダから提供されたクラス・ファイル (ライブラリ) を集めて作成されます。実行時にこれらを結びつけるためには，クラスパスには数百ものパス名が含まれている可能性がある長いクラスパス文字列を指定しなくてはなりません。

たとえば projects/classes/MyApp.class の場合，RTE はまず[projects.classes]MyApp.class に対して stat() を実行します。何らかの理由でそれが失敗すると，[projects.classes]MyApp_class に対して実行します。

アプリケーションが特定のクラス・ファイルを探している場合には，見つかるまでに一般に約 50%のパスを探す必要があります。そして，クラス・ファイルの検索に失敗するパスごとに，RTE は 2 回ずつ stat()を実行しています。

RTE が，探しているクラスを含まないパスを検索するたびに，最初の stat()が失敗すると 2 度目の stat()が実行されることになります。2 度目の stat() 呼び出しに対し，一般的なアプリケーションの起動時にロードされる何百ものクラスを掛けると，2 度目の stat() の呼び出しの合計が，アプリケーションの起動時間のかなりの部分を占めることがわかります。

✓推奨内容

アプリケーションが ODS-5 形式のディスクに排他的にアクセスすることがわかっている場合，またはドットを含むようなディレクトリ名を**絶対に**扱わないことがわかっている場合には，2 度目の stat()呼び出しを無効にすることができます。

2 度目の stat()呼び出しを抑制するには，JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT 論理名を使用します。

```
$ DEFINE/NOLOG JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT TRUE
```

ファイル共用の制限

ほとんどのSDKアプリケーションは，UNIXのファイル共用モードを想定していますが，最適な性能を得るためには，アプリケーションが必要とするファイル共用レベルだけを有効にする必要があります。

✓推奨内容

以下の段階 (最高の性能から最高の機能へ) に応じてファイル共用を設定します。

1. **最高の性能 – 平均的な共用**
   DECC$FILE_SHARING ENABLED
   ファイルは共用でオープンされます (最高の性能と共用。推奨)。
2. **平均以上の性能 – 平均以上の共用**
   JAVA$FILE_OPEN_MODE 3
   ファイルは共用でオープンされ，さらにRMSバッファのフラッシュも使用します。
3. **平均または平均以下の性能 – 最高の共用モード**
   JAVA$FILE_OPEN_MODE 2
   すべての読み書きはディスクと同期されます (性能は最低)。

(**共用なし**
JAVA$FILE_OPEN_MODE 0
共用なしは，ほとんどのアプリケーションでお勧め*できません*。)

「SDK の各種論理名の使用方法」の「JAVA$FILE_OPEN_MODE」も参照してください

大文字/小文字の区別を無効にするための論理名

SDKでは，大文字と小文字を区別し，ODS-2 および ODS-5 のディスク形式をサポートする必要があるため，File.list()関数は .java および .class で終わるファイルをすべてオープンして読み込み，「実際の」SDKまたはクラス名と対応付けようとします。このため，.java や .class ディレクトリが大きい場合，アプリケーションの動作が遅くなります。

✓推奨内容

ODS-5 ディスクを使っている場合には，JAVA$READDIR_CASE_DISABLE に True を設定します。この論理名はドキュメントには記述されていませんが，これを使うことで，ODS-5 ディスクでは不要な，性能を落とす処理を抑制することができます。

補足

JAVA$FILENAME_CONTROLS にも 0 または 1 を定義する必要があります。

本項の「ファイル名マッピングの削減」の「最高の性能」を参照してください

バージョン数の制限

多くのアプリケーションではSDK Swing File Chooserを使用していますが，これは，ディレクトリ全体を読み込んで，ディレクトリ内の各ファイルに対してFile.isDirectoryメソッドを呼び出します。アプリケーションによっては，ファイルの存在を確認する独自の関数を持っているものもあります。たとえば，NetBeansのfile_existsメソッドなどです (このメソッドは文字列を受け取り，文字列に一致するものが見つかるまでディレクトリ一覧を読み込みます)。ファイルのバージョンが複数あると，これらすべてのメソッドが遅くなります。

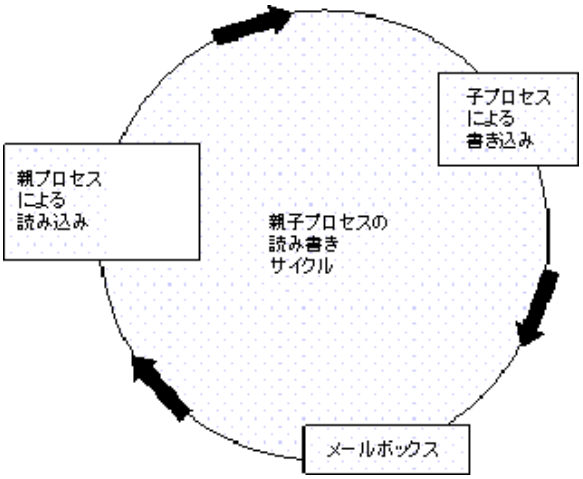
✓推奨内容

古いバージョンのファイルを削除してディレクトリ内のファイルの数を減らすほど，SDKの性能は向上します。

4. 親子プロセスの最適化

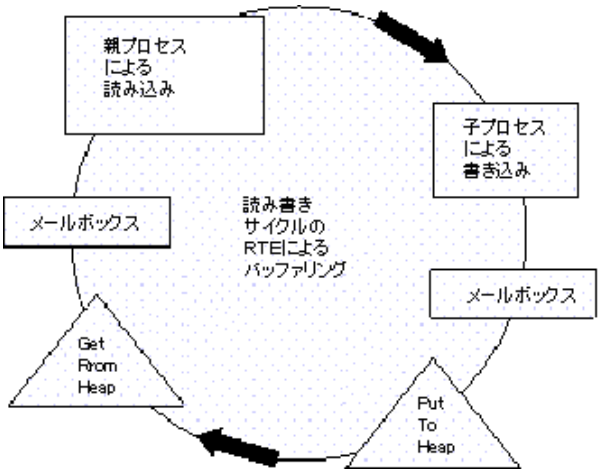
SDK アプリケーションを開発する際には，まず全プロセスを SDK のメモリ空間内で実行できるかどうか，また，SDK 外のプロセスが必要かどうかを明確にします。

OpenVMS Alpha システムでは，SDK プロセスの作成と親子関係はネイティブ・スレッドで管理されます。その間でのデータのやり取りは，*メールボックス*を通じて行われます。たとえば，典型的な例としては，子プロセス (メールボックスの*作成者*) が出力ストリームを生成し，親プロセス (メールボックスの*消費者*) がこれを入力ストリームとします。



アプリケーションによっては同期型として作られているものがあり，親プロセスは，子プロセスを開始した後，子プロセスの終了を待ってから，データ・ストリームの読み込みを開始します。また，別のアプリケーションは非同期型として作られており，親プロセスは可能になりしだい読み込みを開始しますが，子プロセスの速度に追いつけないものがあります。メールボックスの大きさは有限なため，このどちらのアプローチも問題が発生する可能性があります。

メールボックスがいっぱいになると，生成者はブロックされそれ以上処理ができなくなります。その一方で，消費者は生成者が終了するのを待っており，こちらも先に進むことができません。この状態は RWMBX (Resource Wait: MailBoX) 待ち状態と呼ばれ (RWINS (Resource Wait: INner mode Semaphore) とも呼ばれる)，アプリケーションを実行しているプロセスがブロックされます。

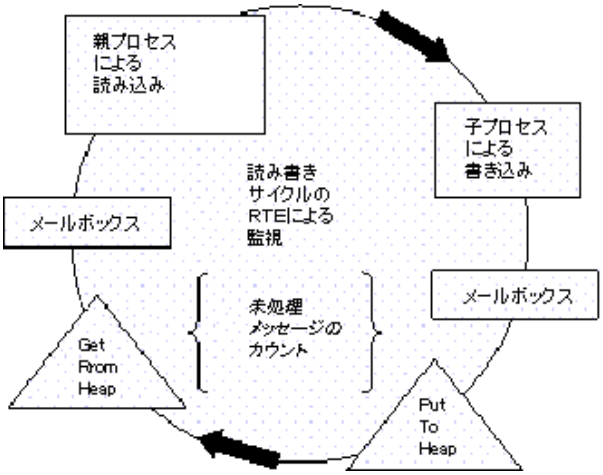


RTE では，メールボックスとメールボックス消費者の間でバッファリングを行って，このような状態になるのを防ごうとします。あるスレッドがメールボックスからデータを読み込み，ヒープ・ストレージに格納します。別のスレッドがバッファリングされたデータをヒープ・ストレージから読み込み，要求があったときに親プロセスに渡します。

親プロセスが子プロセスの終了を待つケースでは，このバッファリングによりメールボックスが空になり，生成者は実行を継続して終了できるため，消費者も同様に継続できるようになります。親プロセスが継続的に読み込むケースでは，一時的に処理が逼迫した場合でも，バッファリングにより生成と消費のデータ・フローが滑らかになります。

ただし，もし子プロセスが完了するまで親プロセスが一切の情報ストリームをまったく消費せずに待っているとした場合，出力量が利用可能なヒープ・ストレージ量を上回ると，このバッファリング機構自体が行き詰まってしまいます。利用できるヒープがなくなると，`PutToHeap` はメールボックスを空にすることができなくなり，この方式でも `RWMBX` 待ち状態になります。ブロックされた状態は，非同期のエージェントを使ってヒープ・ストレージの一部を解放するなど，関連するリソースを変更するまで続きます。

しかし，ほとんどの場合，ヒープ・ストレージによって追加される容量は，子プロセスが完了して親プロセスが継続するのに十分です。このデータ・フローの制御に対するさらなる改良として，RTE は，パイプライン上に未処理のまま残っているメッセージの量を監視します。



書き込まれたメッセージの数から読み込まれたメッセージの数を引いたものが，1024 と論理名 JAVA$FORK_MAILBOX_MESSAGES のどちらか小さいほうに等しくなると，RTE は書き込みを停止して RWMBX 状態にならないようにします。JAVA$FORK_MAILBOX_MESSAGES は，定義されていない場合には，8 と見なされます。

このように一時的に生成を止めることで，親スレッドが追いつくことができ (メッセージ超過のしきい値を下回る)，アプリケーション全体が処理を続行することができます。

メールボックス値を増やす

SDKアプリケーションでのプロセス作成は，論理名JAVA$FORK_MAILBOX_MESSAGESおよびJAVA$FORK_PIPE_STYLEまたはJAVA$EXEC_USE_PIPES[2]を使って調整できます。

JAVA$FORK_MAILBOX_MESSAGES のデフォルト値の 8 では，生成者が消費者より 8 レコードだけ先に進むと，生成者によるバッファ・ストリームの生成は一時的に停止されます。

アプリケーションが小さなレコードを多数生成する場合には，JAVA$FORK_MAILBOX_MESSAGES に大きな値を設定することをお勧めします。典型的なメールボックス・レコード・サイズで 1024 を割り，それに近い値を設定します。

この値をゲーティング因子と考えると，値が大きいほど，生成者のスレッドが開始したり停止する回数が減るため，動作の効率が上がります。ただし，あまり大きくしすぎると，RWMBX 状態を誘発します。JAVA$FORK_MAILBOX_MESSAGES を明示的に指定していない状態で，SDK アプリケーションが RWMBX 状態になる期間があるようであれば，デフォルトの 8 より小さな値を設定してみてください。

JAVA$FORK_PIPE_STYLE の使用

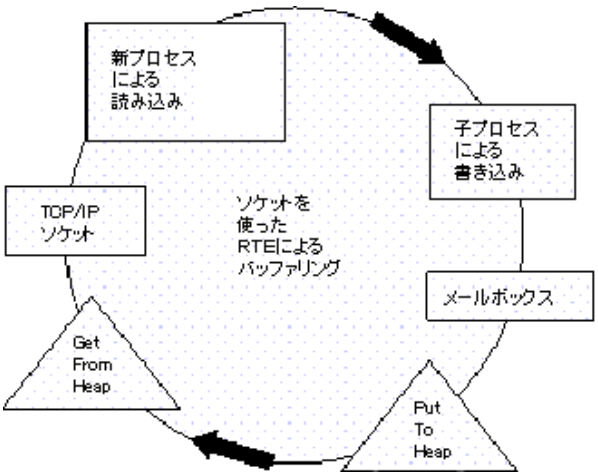
SDK のバージョン 1.3.1-2 で，新しい論理名 JAVA$FORK_PIPE_STYLE が導入されました。この論理名は，JAVA$EXEC_USE_PIPES の動作を拡張し，オプションとしてメールボックスなしのバッファリング方式が利用できるようになっています。JAVA$FORK_PIPE_STYLE の値は JAVA$EXEC_USE_PIPES の値と同じではないことに注意してください。

JAVA$FORK_PIPE_STYLE

[2] SDK 1.3.1-2 以降のユーザは，論理名JAVA$FORK_PIPE_STYLEを使用することをお勧めします。

値	性能	動作
1	普通	デフォルトのメールボックス動作
2	最高	ヒープと SDK アプリケーションの親プロセスによる読み込みの間でデータをバッファリングする際に，メールボックスではなくソケットが使用される (TCP/IP スタックが必要)

JAVA$FORK_PIPE_STYLE に値 2 を指定すると，ヒープと SDK アプリケーションの親プロセスによる読み込みの間でデータをバッファリングする際に，メールボックスではなくソケットが使用されます。これには，TCP/IP スタックが必要です。



この方法は，UNIX のパイプ機能を最も近い形でシミュレートするものです。SDK アプリケーションはソケット・デバイスから読み込むため，バッファリングなしの I/O， ioctl()機能，I/O 操作のキャンセルなどが使用できます。子プロセスからのデータを処理する際に，性能が大幅に向上します。

5. SDK の各種論理名の使用方法

ここでは，アプリケーションの最適化に役立つものの，それ自体は直接性能とは関係しない SDK の論理名の使用方法を説明します。

(JAVA$CACHING_INTERVAL，JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT，および JAVA$FORK_MAILBOX_MESSAGES など，性能に関係する論理名の説明は，前の項にあります。)

JAVA$DIRECTORY_MAPPING_NN

ODS-2 形式のディスク上で 8 階層よりも深いディレクトリ構造をサポートします。ODS-5 では不要です。

アプリケーションで，OpenVMS の最大である 8 階層よりも深くネストしたディレクトリを使う必要がある場合には，JAVA$DIRECTORY_MAPPING_NN に必要な値を定義します。

例

```
$ def java$directory_mapping_01 -
_$ "/foobar/redirect/testing/testing2/testing3=/foobar/cont123"
```

これは，たとえば/foobar/redirect/testing/test2/testing3/の中にある data.txt というファイルをオープンする際には，代わりに/foobar/cont123/data.txt をオープンすることを SDK に指示します。

JAVA$ENABLE_ROOT_WITH_000000

これは，SDK v1.3.1-1 の動作に対する互換性のない変更です。

SDK v1.3.1-1 では，次のような最上位論理名を使ったファイルをオープンしようとした場合に起きる問題が修正されています。

```
File f = new File ("/sys$sysroot");
```

最上位論理名 (たとえば sys$sysroot)，またはディスク名 (たとえば /sys$sysdevice や /node$dka200) のディレクトリ一覧の取得を試みるとします。現在の C RTL では，RTE はこれらの名前に内部的に/000000 を追加して，ディレクトリ・リスト取得の操作が正しく動作するようにします。

この対処により，/sys$errorlog のような最上位でない論理名で問題が起こります。弊社では，新しい C RTL でこれらの特殊ケースを内部的に処理し，RTE でこの対処が不要になるようにしています。この移行を期待して，最新の RTE では，デフォルトの動作では /000000 を自動的に追加しなくなりました。最新の RTE で，アプリケーションがこの機能を必要としている場合には，次の論理名を定義して明示的に要求する必要があります。

```
$ define/job JAVA$ENABLE_ROOT_WITH_000000 TRUE
```

JAVA$EXEC_TRACE

この論理名は，Runtime.exec()関数呼び出しのデバッグに役立つ新しい論理名です。

論理名 JAVA$EXEC_TRACE は，OpenVMS 上での Runtime.exec()呼び出しのデバッグに役立ちます。

```
$ define/job JAVA$EXEC_TRACE true
```

この論理名を次のように定義すると，C RTL の exec の一覧とその引数の一覧が表示されます。

JAVA$FILE_OPEN_MODE

RTE は，限定的なファイル共用をサポートしており，ユーザが選択可能ないくつかのファイル共用モードを持っています。RTE のファイル共用モードを有効にするには，論理名 JAVA$FILE_OPEN_MODE に以下の値のいずれかを指定します。

0 (または定義なし)	ファイル共用なし
2	同期書き込み – すべてのファイル書き込みで，ファイルとディスクが同期される。 **注意** このモードでは，SDKの期待に近いファイル共用が可能です。ただし，通常は数秒で終わる処理が，数分かかる場合があります。
3	SDKは，オープンされている全ファイル・ディスクリプタのテーブルを保持する。SDKはアプリケーションのファイル書き込みを監視し，write_pendingフラグを設定する。各読み込み操作の前に，SDKはオープンされているファイル・ディスクリプタの一覧をスキャンする。書き込みが保留されているものに一致した場合には，読み込みの前に，保留されている内容がディスクに書き込まれる。

注意
値1は，DECC$FILE_SHARING 論理名で置き換えられました。

JAVA$KEYBOARD_TYPE_DEC

RTE がキー入力に応答する際，デフォルトでは，数字キーパッドの上段が「NumLock」，「/」，「*」 および 「-」 になっている PC キーボードからの入力と見なされます。

DEC キーボードを使っている場合は，キーパッドの上部には異なるキー (PF1，PF2，PF3，および PF4) があるため，このようなキー解釈は正しくありません。

次のように JAVA$KEYBOARD_TYPE_DEC 論理名を定義すると，DEC キーボードのキーが正しく解釈されます。

```
$ define/log JAVA$KEYBOARD_TYPE_DEC true
```

また，上述のキー以外に，DIGITAL 形式のキーボードの Find キーと Select キーも期待どおりに動作します。

JAVA$OMIT_CASE_CHECK

JAVAC および JAR のファイル・オペランドの大文字と小文字は，自動的に判断されます。以前弊社では，JAVAC と JAR でワイルドカードのファイル指定を受け付け，名前の大文字と小文字を正しく判断するような機能を追加していました。たとえば次のような指定が可能でした。

```
$ JAVAC *.java
および
$ JAR cvf test.jar *.class
```

SDK 1.3.1 では，これら 2 つのツールでこの機能が任意の .java ファイルと.class ファイル・オペランドに対して拡張され，たとえば次のような指定が可能です。

```
$ JAR cvf test.jar TESTPLOT.CLASS
```

また，名前に大文字と小文字が混在していても気にする必要はありません。

```
$ JAR cvf test.jar "TestPlot.class"
$ JAR cvf test.jar "TESTPLOT.CLASS"
または
$ JAR cvf test.jar "testplot.class"
```

この機能は，DCL を使用してオペランドを自動的に生成する場合に特に便利です。DCL では，オペランドが大文字になるためです。一般にこの機能により，名前の大文字と小文字を正確に入力し，引用符で囲まなくてはならなくなるような状況が減ります。

注意
この機能は，JAVAC と JAR でのみ有効です。たとえば JAVA コマンドなどでは，大文字と小文字を正しく指定する必要があります。

この機能はデフォルトで有効です。既存のコマンド・プロシージャで問題が起きる場合は，次のように設定することで完全に無効にできます。

```
$ define JAVA$OMIT_CASE_CHECK true
```

JAVA$SHOW_FILENAME_MAPPING

ファイル名のマッピングが起きたときに，個々のマッピングを表示させるには，論理名 JAVA$SHOW_FILENAME_MAPPING を定義します。

```
$ DEFINE JAVA$SHOW_FILENAME_MAPPING 1
```

この機能は，ファイル名のマッピングで問題が起き，予期しない「File not found」メッセージが表示されるときに便利です。

VAXC$PATH

SDK v1.3.1-2 よりも前では，クラスパスにディレクトリ・パスを指定しないと，RTE はデフォルトのディレクトリだけを検索していました。SDK v1.3.1-2 からは，デフォルト・ディレクトリ以外で.EXE ファイルや.COM ファイルを探すための OpenVMS の検索パスとして，VAXC$PATH が使えるようになりました。たとえば次のように指定します。

```
$ define VAXC$PATH GNU:[bin],[],sys$common:[java$131.bin]
                        ! open source GNU files
   br = new BufferedReader(new InputStreamReader(
   rt.exec("chmod").getInputStream()));
```

上記のようにすると，VAXC$PATH で定義された 3 つのディレクトリで chmod.，chmod.exe，および chmod.com が検索されるようになります。また，以下のようにすると，javac.，javac.exe，および javac.com が検索されます。

```
   br = new BufferedReader(new InputStreamReader(
   rt.exec("javac").getInputStream()));
```

JAVA$CACHING_ATEXIT_PRINT_STAT

ドキュメントに記述されていないこの論理名により，プログラム終了後に stat()のキャッシュ・インデックスがダンプされます。JAVA$CACHING_INTERVAL のトラブルシューティングの目的で使用するためには，以下の値を指定します。

値 100　すべての情報をダンプする
値 1　　要約をダンプする

JAVA$CREATE_DIR_WITH_OWNER_DELETE

ドキュメントに記述されていないこの論理名は，DECC$file_owner_unix で置き換えられました。

JAVA$CREATE_ONE_VERSION

ドキュメントに記述されていないこの論理名を使うと，ファイルのバージョンが 1 つだけ作成されます。

JAVA$RENAME_ALL_VERSIONS

ドキュメントに記述されていないこの論理名を使うと，ファイルの全バージョンが名称変更されます。通常，C RTL では，ファイルのバージョン番号が最も高いものだけが名称変更されます。

JAVA$WAIT_FOR_CHILDREN XX

ドキュメントに記述されていないこの論理名に XX を指定すると，SDK は `exit()`を呼び出す前に XX 秒間待つようになります。親プロセスが `exit()`を呼び出したときに，子プロセスが停止処理 (ファイルのクローズ) の最中である可能性があります。その場合，`ACCVIO` または `pthread` ダンプとなります。

6. アプリケーションの最適化

SDK アプリケーションの形式や規模はさまざまです。各アプリケーションには個別の特性や要件があります。ここでは，OpenVMS の 4 つのレイヤード・プロダクトについて，性能上の推奨事項を説明します。

BridgeWorks

この推奨事項は，このアプリケーション製品と同じ Alpha システム上で SDK コンポーネントを生成する際に当てはまります。

初期ヒープ・サイズ

複雑な構造体に対して完全にメモリを割り当てるには，SDK のヒープ・メモリ・サイズの初期値を増やす必要があります。たとえば SDK v1.3.1 では，コマンド `java -Xms64M -Xmx256 helloworld` のように指定すると，ヒープ・サイズが最低 64MB，最高 256MB 割り当てられます。(これらオプションの設定方法については，`java -X` で表示されるオプションを参照してください。)

> **注意**
> BridgeWorks ソリューションは一部が SDK のコード，一部がネイティブなコードとなっているため，適切なバランスを見つけるのが非常に重要です。SDK にメモリを割り当て過ぎても，全体の性能は劣化します。

ヒープ値を増やす

SDK のヒープの起動が遅いという現象を解消するには，`MX` と `MS` に非常に大きな値を設定します。

> **注意**
> 物理メモリ・サイズより大きな初期ヒープ・サイズを割り当てると，オペレーティング・システムが仮想メモリを割り当てることになります (一時的にデータをディスクへページングすることになり，性能が大幅に低下します)。

CSWS_JAVA

適切なクォータのバランスを見つけて，Secure Web Server と CSWS_JAVA モジュールの両方の性能を出さなければいけません。

Tomcat および JServ の要件

Jakarta (Tomcat) または JServ サーブレット・エンジンに対するユーザ・アカウントを設定する際には，SDK のクォータの要件を考慮して，SDK アプリケーションで最高の性能が得られるようにします。
HP Secure Web Server のインストールで，`APACHE$WWW` アカウントが設定されますが，このアカウントに対するデフォルトのクォータは，SDK 用に最適化されていません。特に，`FILLM` (および関連する `SYSGEN` パラメータ `CHANNELCNT`)，`PGFLQUO`，および `BYTLM` を増やす必要があります。

これらはプールされるクォータです。サブプロセスである JServ サーブレット・エンジンを構成する際には，これらのクォータに対する，同じジョブ・ツリー内にある Apache の子プロセスによる影響に注意する必要があります。Jakarta (Tomcat) サーブレット・エンジンは独立プロセスであるため，Apache の子プロセスには影響されません。

JServ ログ・ファイルの高速な更新

`JSERV.LOG` ファイルは，十分なデータがたまってバッファの内容がフラッシュされるまでは空のままです。(サーバをシャットダウンすると，ログ・ファイルは空です。) サーブレットのデバッグをしているなどの理由で，JServ ログ・ファイルをよりタイムリーに更新する必要がある場合には，ファイルのフラッシュと共用のモードを変更できます。そのためには，ファイル `START_JSERV_MANUAL.COM` (`APACHE$COMMON:[000000]`にあります) 内の論理名 `JAVA$FILE_OPEN_MODE` を変更します。

JAVA$FILE_OPEN_MODE には，以下の値を指定できます。

0	ファイル共用なし – デフォルト
2	同期書き込み – すべてのファイル書き込みで，ファイルとディスクが同期される。
3	JVMはアプリケーションのファイル書き込みを監視し，読み込み操作の前に書き込み結果をディスクにフラッシュする。ただし，ファイルを確実に共用できるのは1プロセスのみです。

注意

生産環境では JAVA$FILE_OPEN_MODE に 2 を設定しないでください。性能に悪影響が出ます。

「性能に関するその他のヒント」の「ファイルのチューニング」の下の「ファイル共用の制限」も参照してください

NetBeans

NetBeans Launcher により，初期スタック・サイズ，初期ヒープ・サイズ，最大ヒープ・サイズのデフォルトの値が選択されます。これらの値は，平均的なワークロードをシミュレートするテストに基づいて選択されたものです。

もし NetBeans プロセスがヒープまたはスタック領域を使い果たしてしまう場合は，スタック・サイズまたは最大ヒープ・サイズとしてより大きな値を指定して NetBeans を起動する必要があります。

これは，最大ヒープ・サイズとして 350MB，スタック・サイズとして 2MB を設定する例です。

```
NetBeans "" "-Xmx350m –Xss2m"
```

BEA WebLogic Server

weblogic600 または weblogic610 アカウントでは，最低クォータを以下のようにする必要があります。

```
PGFLQUO         1500000
WSDEF           8000
WSEXTENT        32000
WSQUO           16000
BYTLM           300000
FILLM           1024
```

付録 A: Java$ 論理名

SDK の『*Release Notes* 』(v 1.4.0 またはそれ以前), または『*User Guide* 』 (v 1.4.1 以降) の「Using the Fast VM」を参照してください。
v 1.4.1 の場合:
http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#usingfastvm
v 1.3.1-6 の場合:
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#usingfastvm

このドキュメントにのみ記載されている論理名は，「文書化されていない」とみなされます。そのような論理名はサポートされませんので，各自の責任でご使用ください。

論理名	SDK 1.3.1-6 または 1.4.1 での新規	文書化	Web ベースのリリース・ノートへの参照
JAVA$CACHING_ATEXIT_PRINT_STAT			非サポート
JAVA$CACHING_INTERVAL		✓	1.3.1.6, 1.4.1
JAVA$CLASSPATH		✓	1.3.1.6, 1.4.1
JAVA$CREATE_DIR_WITH_OWNER_DELETE	✓	✓	1.3.1.6, 1.4.1
JAVA$CREATE_ONE_VERSION			非サポート
JAVA$DAEMONIZE_MAIN_THREAD	✓	✓	1.3.1.6, 1.4.1
JAVA$DELETE_ALL_VERSIONS	✓	✓	1.3.1.6, 1.4.1
JAVA$DIRECTORY_MAPPING_COUNT		✓	1.3.1.6, 1.4.1
JAVA$DIRECTORY_MAPPING_NN		✓	1.3.1.6, 1.4.1
JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT		✓	1.3.1.6, 1.4.1
JAVA$ENABLE_ENVIRONMENT_EXPANSION	✓	✓	1.3.1.6
JAVA$ENABLE_ROOT_WITH_000000		✓	1.3.1.6, 1.4.1
JAVA$ENABLE_SIGQUIT_CTRLC	✓	✓	1.3.1.6, 1.4.1
JAVA$ENABLE_SIGQUIT_MAILBOX	✓	✓	1.3.1.6, 1.4.1
JAVA$EXEC_TRACE		✓	1.3.1.6, 1.4.1
JAVA$EXEC_USE_PIPES		✓	1.3.1.6, 1.4.1
JAVA$FILE_OPEN_MODE		✓	1.3.1.6, 1.4.1
JAVA$FORK_MAILBOX_MESSAGES		✓	1.3.1.6, 1.4.1
JAVA$FORK_PIPE_STYLE		✓	1.3.1.6, 1.4.1
JAVA$FSYNC_INTERVAL	✓	✓	1.3.1.6
JAVA$KEYBOARD_TYPE_DEC		✓	1.3.1.6

論理名	SDK 1.3.1-6 または 1.4.1 での新規	文書化	Web ベースのリリース・ノートへの参照
JAVA$OMIT_CASE_CHECK		✓	1.3.1.6, 1.4.1
JAVA$PRINT_COMMAND_ARGS	✓	✓	1.3.1.6, 1.4.1
JAVA$READDIR_CASE_DISABLE	✓	✓	1.3.1.6, 1.4.1
JAVA$RENAME_ALL_VERSIONS	✓	✓	1.3.1.6, 1.4.1
JAVA$SHOW_FILENAME_MAPPING	✓	✓	1.3.1.6, 1.4.1
JAVA$WAIT_FOR_CHILDREN			非サポート
VAXC$PATH		✓	1.3.1.6, 1.4.1

付録 B: DECC$ 論理名

この一覧にある論理名は，Web ベースの『*C ランタイム・ライブラリ・リファレンス・マニュアル*』または最新の ACRTL ECO の readme ファイルに記載されています。

ECO キー	
4	dec-axpvms-vms721_acrtl-v0400--4
3	dec-axpvms-vms721h1_acrtl-v0300--4
2	dec-axpvms-vms73_acrtl-v0200--4
1	dec-axpvms-vms722_acrtl-v0100--4

論理名	ECO での新規	文書化	Web ベースのリファレンスおよび ECO 情報
DECC$ARGV_PARSE_STYLE	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$DEFAULT_LRL		✓	C RTL リファレンス
DECC$DEFAULT_UDF_RECORD		✓	C RTL リファレンス
DECC$DETACHED_CHILD_PROCESS		✓	C RTL リファレンス
DECC$DISABLE_POSIX_ROOT		✓	C RTL リファレンス
DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION	✓	✓	C RTL リファレンス 4, 3
DECC$EFS_CASE_PRESERVE	✓	✓	C RTL リファレンス 4, 3
DECC$EFS_CASE_SPECIAL	✓	✓	C RTL リファレンス 4, 3
DECC$EFS_CHARSET	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$EFS_FILE_TIMESTAMPS		✓	C RTL リファレンス
DECC$ENABLE_GETENV_CACHE		✓	C RTL リファレンス
DECC$EXEC_FILEATTR_INHERITANCE		✓	C RTL リファレンス
DECC$FILENAME_UNIX_NO_VERSION	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$FILENAME_UNIX_ONLY	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$FILENAME_UNIX_REPORT	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$FILE_OWNER_UNIX		✓	C RTL リファレンス
DECC$FILE_PERMISSION_UNIX		✓	C RTL リファレンス
DECC$FILE_SHARING	✓	✓	C RTL リファレンス 4, 3
DECC$FIXED_LENGTH_SEEK_TO_EOF		✓	C RTL リファレンス

論理名	ECO での新規	文書化	Web ベースのリファレンスおよび ECO 情報
DECC$LOCALE_CACHE_SIZE		✓	C RTL リファレンス
DECC$MAILBOX_CTX_STM		✓	C RTL リファレンス
DECC$PIPE_BUFFER_SIZE		✓	C RTL リファレンス
DECC$POSIX_SEEK_STREAM_FILE	✓	✓	C RTL リファレンス 4, 3
DECC$READDIR_DROPDOTNOTYPE		✓	C RTL リファレンス
DECC$READDIR_KEEPDOTDIR			非サポート
DECC$RENAME_NO_INHERIT	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$SELECT_IGNORES_INVALID_FD	✓	✓	C RTL リファレンス 4, 3
DECC$SET_CHILD_STANDARD_STREAMS		✓	C RTL リファレンス 1
DECC$STDIO_CTX_EOL		✓	C RTL リファレンス
DECC$STRTOL_ERANGE		✓	C RTL リファレンス
DECC$TRACE			非サポート
DECC$THREAD_DATA_AST_SAFE	✓	✓	C RTL リファレンス 4, 3, 2, 1
DECC$TZ_CACHE_SIZE		✓	C RTL リファレンス
DECC$UMASK		✓	C RTL リファレンス
DECC$UNIX_PATH_BEFORE_LOGNAME		✓	C RTL リファレンス
DECC$USE_RAB64		✓	C RTL リファレンス
DECC$V62_RECORD_GENERATION		✓	C RTL リファレンス
DECC$VALIDATE_SIGNAL_IN_KILL		✓	C RTL リファレンス
DECC$XPG4_STRPTIME	✓	✓	C RTL リファレンス 4, 3

機能ごとの DECC$ 論理名

以下の表 (『Cランタイム・ライブラリ・リファレンス・マニュアル』より) は，文書化されている DECC$ 論理名のほとんどを，機能別にまとめ，それぞれのデフォルト設定を示したものです。

性能の最適化	
DECC$ENABLE_GETENV_CACHE	DISABLE
DECC$LOCALE_CACHE_SIZE	0
DECC$TZ_CACHE_SIZE	2
従来の動作	
DECC$V62_RECORD_GENERATION	DISABLE
DECC$XPG4_STRPTIME	DISABLE
DECC$THREAD_DATA_AST_SAFE	DISABLE
ファイル属性	
DECC$DEFAULT_LRL	32767
DECC$DEFAULT_UDF_RECORD	DISABLE
DECC$FIXED_LENGTH_SEEK_TO_EOF	DISABLE
メールボックス	
DECC$MAILBOX_CTX_STM	DISABLE
UNIX **準拠のための変更**	
DECC$STRTOL_ERANGE	DISABLE
DECC$VALIDATE_SIGNAL_IN_KILL	DISABLE
DECC$SELECT_IGNORES_INVALID_FD	DISABLE
一般的な UNIX **拡張機能**	
DECC$ARGV_PARSE_STYLE	DISABLE
DECC$PIPE_BUFFER_SIZE	512
DECC$STDIO_CTX_EOL	DISABLE
DECC$USE_RAB64	DISABLE
UNIX **形式のファイル名をサポートするための拡張**	
DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION	DISABLE
DECC$EFS_CHARSET	DISABLE
DECC$FILENAME_UNIX_NO_VERSION	DISABLE
DECC$FILENAME_UNIX_REPORT	DISABLE
DECC$READDIR_DROPDOTNOTYPE	DISABLE
DECC$RENAME_NO_INHERIT	DISABLE
UNIX **形式のファイル属性をサポートするための拡張**	
DECC$EFS_FILE_TIMESTAMPS	DISABLE
DECC$EXEC_FILEATTR_INHERITANCE	DISABLE
DECC$FILE_OWNER_UNIX	DISABLE
DECC$FILE_PERMISSION_UNIX	DISABLE
DECC$FILE_SHARING	DISABLE
UNIX **準拠モード**	
DECC$FILENAME_UNIX_ONLY	DISABLE
DECC$DETACHED_CHILD_PROCESS	DISABLE
POSIX **準拠のための新しい動作**	
DECC$POSIX_SEEK_STREAM_FILE	DISABLE
DECC$UMASK	RMS のデフォルト

ファイル名の処理	
DECC$READDIR_KEEPDOTDIR	DISABLE
DECC$EFS_CASE_PRESERVE	DISABLE
DECC$EFS_CASE_SPECIAL	DISABLE
DECC$UNIX_PATH_BEFORE_LOGNAME	DISABLE
DECC$DISABLE_POSIX_ROOT	DISABLE