勤務票システムにおける技術課題

Technical Subject of the Work Record Management System

山 本 史 朗

要 約 国内においても，Java を用いたミッションクリティカルな業務への適用が出始めている．当社でも Java と CORBA という先進技術を用いた勤務状況報告・管理システムを開発した．

この開発経験から得た現在の Java 適用における問題点と対処方法について報告する．

Abstract Applications for mission critical business system running in the Java environment is being implemented and released in Japan.

And also, Nihon Unisys Ltd. developed Employee Attendance Record Management System using the advanced technology of Java and CORBA.

The paper describes the problem and resolution of applying Java experienced in system development phases.

1. はじめに

Java 技術は，Java Applet を用いた WWW（World Wide Web）アプリケーションへの適用に始まり，近年では，サーバ側の構築にも Java を適用すべく拡張 API が急速に実装されてきている．

また，オープン環境下におけるオブジェクト間通信手段として，近年その実用化が急速に進んでいる CORBA（Common Object Request Broker Architecture）も大規模なシステムを構築する際の主要技術として欠かせないものとなってきている．

当社，勤務状況報告・管理システム(以降，勤怠システム)開発では，Java，CORBA という先進的な技術を利用し，全社員規模の基幹システムを構築するという試みを行った．

本稿では，実際に開発に携わった経験から，現在の Java 適用における問題点とその対処方法について報告する．第 2 章から第 5 章では，Java Applet について報告する．第 6 章では CORBA の実装の一つである Java IDL について報告する．第 7 章と第 8 章では，Java プログラムのデバック，パフォーマンス・チューニングについて報告する．第 9 章と第 10 章では，日本語文字の扱いと帳票機能について報告する．

2. プレゼンテーション構築

最近流行の Web アプリケーション・サーバでは，プレゼンテーションを HTML（Hyper Text Markup Language）でクライアント/サーバ間のプロトコルを HTTP（Hyper Text Transfer Protocol）で構築する例が多い．この理由には HTML にする事で比較的ブラウザ間の非互換等に悩まされることが少なくなることと，SSL(Secure Sockets Layer）などセキュリティ技術を適用しやすい環境が整ってきていることが挙げられる．しかし，勤怠システム開発においては，HTML や Java Script のような

クライアント・サイド・スクリプト言語では表現できない高度なGUI部品が必要なこと，イントラネット限定で，それほどセキュリティの要求が厳しくないことなどから，プレゼンテーションをJava Appletで構築している．

Javaではプレゼンテーションの構築にバージョン1.0から存在するAWT（Abstract Windows Toolkit）と，バージョン1.2から標準パッケージに加えられたSwingという二つの選択肢がある．

勤怠システム開発においては，先に挙げた高度なGUI部品を実現するためにSwingを採用している．以下に，Swingを利用する際の，メリットと，デメリットを説明する．

メリットとしては，

- コンポーネントの種類が豊富（AWTに比べ機能的にも高く，拡張性がある）
- Pluggable Look and Feel（MVCモデル）
- Pure Java（プラットフォーム独立）
- JavaBeansとして利用可能（統合開発環境ツールでインポート可能）

などが挙げられる．さらに，Windows 95などでAWTを使う場合のシステム・リソースの制限などからも解放される．

一見良いことばかりに見えるSwingだが，その性能面がデメリットとも言える．

Swingは高機能をJava内部で行うことで，上記のようなメリットを生み出しているが，逆にJavaが内部的に果たす仕事が莫大に増えていることも事実である．

表1は，WindowsNTおよびJDK（Java Development Kit）1.2.1上でAWT，Swingで作成したフレームにボタンを追加表示していくプログラムのJavaHeapサイズ（プログラム内で，ボタン追加直後に，Sytem.gc（），（Runtime.totalMemory（）—Runtime.freememory（））で算出），Javaプロセスサイズ（タスクマネージャ参照）をまとめたものである．Swingは AWTに比べ，JavaHeapで数倍，プロセスメモリで，2倍近く資源を必要としていることが分かる．（値は，プラットフォームや追加するGUIウィジットによって数値は異なってくる）

表 1　AWT，Swingにおけるプロセス使用量の比較

追加ボタン数	AWT JavaHeapサイズ（Byte）	AWT Javaプロセスサイズ（Kbyte）	Swing JavaHeapサイズ（Byte）	Swing Javaプロセスサイズ（Kbyte）
0	696808	8512	954304	10712
500	993832	9168	2074672	12792
1000	1238600	9796	3044088	14228
1500	1495392	10560	4018328	15584
2000	1732904	11260	4975520	17360
3000	2244072	12532	6916072	20544

このように，Swingには性能面で，より多くのメモリ資源が必要になることがおわかり頂けたと思う．さらに，Swing自体のバグによりdisposeしたはずのフレームやDialogのリソースが残る，いわゆるメモリ・リークなどにより，莫大にメモリが消費され，コンピュータがハング状態に陥ることなどもあり得る．

勤怠システム開発においては，メモリ・リークについて，後で説明するプロファイラ・ツール等を使ってリーク・オブジェクトを解析し，できるだけ内部要素を破棄するような考慮をコードに埋め込むことで，いくらかの改善を試みている．ただし，本質的な解決はバグフィックスを待つしか無い．

安易に高級な画面を設計してしまうと後で取り返しがつかなくなるので，必ずプロトタイプを作成し，性能評価を前もって行うことを強くお勧めする．

3. Java VM の非互換

Java でプレゼンテーションを構築する際，良く問題になるのが，Java VM（Virtual Machine）の非互換問題である．

この問題には，

- ベンダ間の互換性
- バージョン間の互換性

の二つが存在する．

前者は，Java VM を提供するベンダ間の非互換であり，例えば，Sun Microsystems 社と Microsoft 社では，AWT の振る舞いが異なる場合がある．後者は，バージョンの違いによる非互換であり，Netscape Navigator などは標準で Java VM を搭載しているが，最新の Java 2(以前は JDK 1.2 と呼ばれていた)には対応していないので Java 2 で作られたプログラムは動作しない可能性がある．

Java でプレゼンテーションを構築し，対象クライアントがマルチプラットフォームになると必ず問題になっていたと言っても過言ではない．

これに対し，Sun Microsystems 社は Java Plug-in で，タイムリに Netscape Navigator や Internet Explorer に最新 Java VM を搭載できるような仕組みを提供し始めた．これにより，最新の JDK バージョンが，Netscape 社や，Microsoft 社の対応を待つことなく利用できる．また，同じ Java VM を使うことになるので，先のベンダ間の非互換も基本的には無くなるはずである．

勤怠システム開発においても，開発当初から Java Plug-in を採用することを決めており，現在の動作環境でも Java Plug-in を必須環境としている．なお，運用面からサポート対象は，Netscape Navigator のみとしているが，Internet Explorer でも簡単な動作確認はできている．

4. Java Applet の起動時間

Java Applet の良いところは，アプリケーションの配布機能で都度ダウンロードすることで最新のコードであることを保証し，運用コストを削る期待ももたれている．

しかし，勤怠システム・アプレットのプログラム・サイズは，おおよそ 1.5 Mbyte である．これを，起動毎にダウンロードしていたのでは，起動時間もかかる上に，ネットワークにも負荷がかかることになる．特に，ネットワーク回線の細い営業所や事業所への展開を考えた場合，現実的な運用は厳しいものがある．

そこで勤怠システム開発では，Java 2 の新機能であるダウンロード・エクステンション機能を使って，事前にコア・プログラムをダウンロードしておき，必要最低限度

のプログラムだけ，都度ダウンロードする方式を採った．これにより，都度ダウンロードに必要なプログラム・サイズは，約 30 KByte にまで減少した．

この方式の問題は，事前にダウンロードしておくコア・プログラムの鮮度である．冒頭にもあるような，Java Applet の利点が生かせないのである．そこで，これも Java 2 の新機能であるバージョンチェック機能を使って，コア・プログラムにバージョンを付加し配布，プログラム実行時に，バージョンをチェックし，もしも期待していないバージョンであれば，コア・プログラムをダウンロードするようにユーザに促す処理を入れている．

簡単なフローを図 1 に示す．

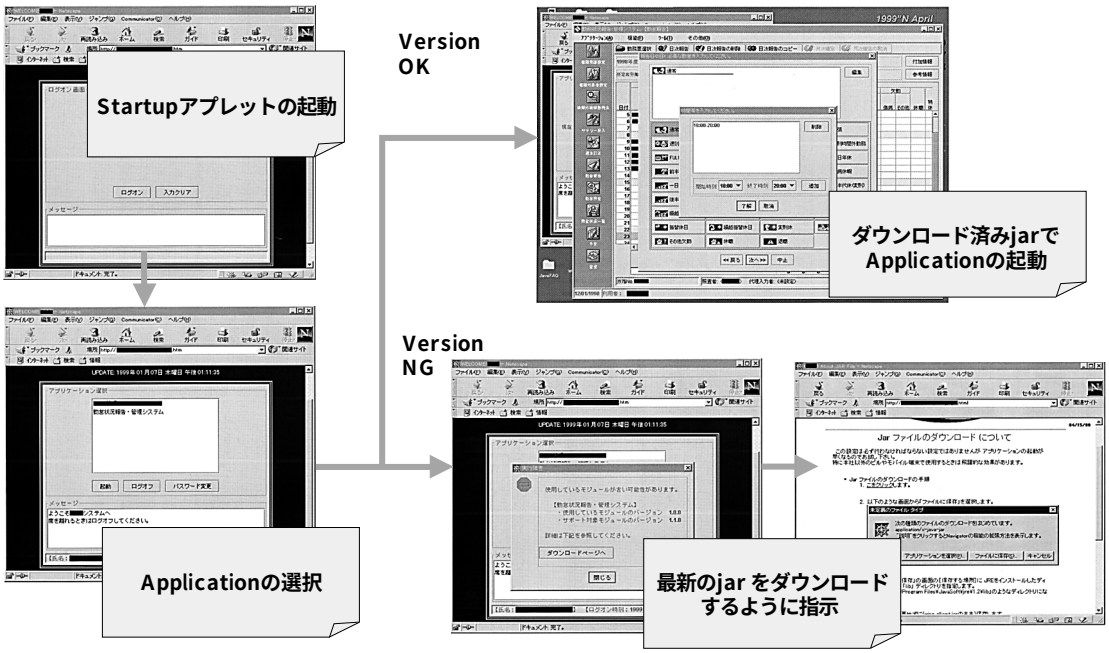

図 1　Applet ダウンロード・フロー

バージョン情報は，jar ファイルのマニフェストファイルに以下のような形式で付加する．

```
Manifest-Version: 1.0
Created-By: 1.0 (Nihon Unisys Ltd.)
Implementation-Vendor: NUL          アーカイブ全体に対して有効な記述
Implementation-Title: TITLE
Implementation-Version: build0
Specification-Vendor: NUL_S
Specification-Title: TITLE_S
Specification-Version: 1.0.0

Name: pkg1/                         特定のパッケージに対して有効な記述
Implementation-Vendor: NUL1
Implementation-Title: TITLE1
Implementation-Version: build1
Specification-Vendor: NUL1_S
Specification-Title: TITLE1_S
Specification-Version: 1.0.1
```

各エントリの意味は表2に示す.

表 2　マニフェストファイルのエントリ

エントリ名	意味
Implement-Title	パッケージのタイトル
Implement-Version	バージョン番号
Implement-Vendor	ベンダー企業または組織
Specification-Title	仕様のタイトル
Specification-Version	バージョン番号
Specification-Vendor	ベンダー企業または組織

プログラム内で，バージョン情報を取り出すには以下のような処理を行う.

```
package pkg1;
  public class A {
    public static void main(String[] args) {
      try {
        Class c   = Class.forName("pkg1.A");
        Package p = c.getPackage();
        if (null != p) {
          System.out.println(p.getImplementationVendor());
          System.out.println(p.getImplementationTitle());
          System.out.println(p.getImplementationVersion());
          System.out.println(p.getSpecificationVendor());
          System.out.println(p.getSpecificationTitle());
          System.out.println(p.getSpecificationVersion());
        }
      } catch (ClassNotFoundException ex) {}
    }
  }
```

このように，Java 2の新機能を使えば，Java Appletの問題とされていた起動時間の短縮も可能である.

5. Java Appletのセキュリティ制限

従来，Java Applet適用の足かせとなっていた問題の一つにセキュリティ問題がある.

バージョン1.0時代は，ローカル・クラスもしくはリモート・クラス，バージョン1.1時代には，ローカル・クラス（署名付きリモート・クラス含む）もしくはリモート・クラスにより，実行クライアント上でのアクセス制御が施されていた．これは，悪意のあるJava Appletを安易に実行クライアント上で動かせないための仕組みである.

しかしながら，この制限が，逆にシステム構築の際には思わぬ足かせにもなっていたわけである．そのいくつかを挙げると，

- Appletホスト（Java Appletをダウンロードしたホスト）としか通信できない
- ローカル資源にアクセスできない

などがある．

Applet ホストとしか通信できないため，とにかく Applet ホストを proxy にしていろいろな通信を行うしかない．（HTTP サーバ配下に複数のアプリケーション・サーバを配置し，目的別のアプリケーション・サーバとクライアントが直接通信することはできない）また，ローカル資源にアクセスできないため，アプリケーション・ログや，各種個人設定ファイルなどを，サーバ・サイドに保管するしかない．インターネット環境ならまだしも，イントラネット環境で，それほどセキュリティに拘らない場合などは，この制限がかえって邪魔になってくるのである．

WISE 開発においては，上記の制限を緩めることが容易な，Java 2 の新セキュリティ・モデルを採用している．

簡単に新セキュリティ・モデルの特徴を挙げると，

- 精密なアクセス制御（SecurityManager クラスと ClassLoader クラスのサブクラス化およびそのカスタマイズなど）．
- 単に設定できるセキュリティポリシ
- 簡単に拡張できるアクセス制御構造
- アプリケーションとアプレットを含む，すべての Java プログラムへのセキュリティチェックの拡張

などがある．Sun Microsystems 社は JDK の中に実装の一つとして PolicyFile クラスを含めており，付属の Policytool（GUI でポリシーファイルを作成する）などで簡単に設定できる．

```
/*   .java.policy の例   */
grant codeBase "http://Applet ホスト/-" {
   permission java.security.AllPermission;
};
```

勤怠システム開発では，Applet ホストからダウンロードして来たクラス（jar 含む）に対しては，全ての権限（通信やローカル資源アクセスなど）を与えるポリシ・ファイルを情報システム部が作成，配布する運用を採っている．

6. JDK の CORBA 実装

図 3 のアーキテクチャで示すように，勤怠システムでは分散オブジェクト技術を主要なモジュール間通信技術として採用している．分散オブジェクト技術には，Sun Microsystems 社の RMI（Remote Method Invocation）や OMG（Object Management Group）の CORBA（実装として当社の SYSTEM ν［nju：］，Sun Microsystems 社の JavaIDL などがある）が存在する．

勤怠システム開発では，性能や機能面などで問題が発生した場合，実装の取り替えが容易な CORBA を採用している．Java 2（JavaPlug-in）に含まれる JavaIDL をクライアント，サーバ・モジュールで，データ・アクセス・モジュールを SYSTEM ν［nju：］で実装している．

JavaIDL を採用した理由には，SYSTEM ν［nju：］との接続実績があること，Java 2 に実装が含まれるため，クライアント・モジュールのダウンロードを必要としない

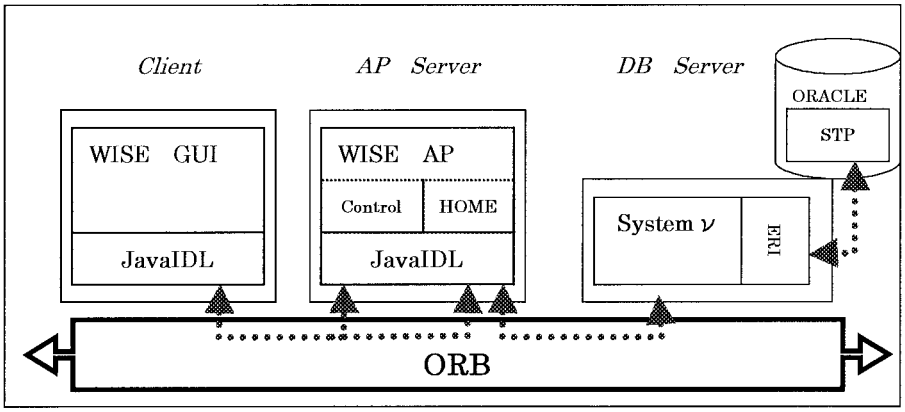

図 3　勤怠システム・アーキテクチャ

ことなどがある．

しかし JavaIDL は，本番システムで利用するには，やや不安な点もある．勤怠システム開発において問題となったのは，大量の接続要求が発生した際に，フリーズ状態となるコネクションが発生することである．原因は，一定量のメソッド同時起動が発生した場合に，スレッドが起こせなくなり（内部的には Java の資源不足を促す Error 例外が発生している），本来は，メソッド起動要求元に例外が返らなければならないはずが，何も返さずフリーズしてしまうというものである．また，コネクション管理も弱く，一旦開設したコネクションは，メソッド起動が発生しなくなっても，どちらかのノードの ORB プロセスが終了しない限り，オープンしたままになってしまう（一定時間での解放などは実装されていない）ので資源が無駄に使われることになる．

上記の問題は，SYSTEM ν［nju：］の流量制御機能を使用することで解決している．流量制御は，メソッド起動要求を，いったん TA（SYSTEM ν［nju：］のトランスファ・エージェント）がキューイングし，管理するコネクション・プールを使って，対象オブジェクトに接続しメソッド起動するものである．この際，同時にメソッド起動できる数を TA で制御することができる．なお，若干オーバーヘッドも発生するが，流量制御用の TA をサーバ・モジュールと同一の物理マシン上に配置することで最小に抑えることができる．

7.　バグの追跡

Java プログラムのデバックは比較的簡単だとよく言われる．これは，C や C++ などに比べポインタ演算が排除されていることが大きく影響している．

プログラムの単体レベルでのデバックについては，個々にプリントアウト・コードを埋め込め込んでも良いが，できることなら使用する統合開発環境ツール（ボーランド社の JBuilder など）のデバック機能を利用することが望ましい．

さて，Java プログラムの不具合を追跡する際に困るのが，問題の切り分けである．自分のコードが悪いのか，API のバグなのか判断しなければならない．仮に問題が

API にあったとして，その場合の回避策など，個人で考えていると開発はおぼつかない．このような場合には以下の WWW サイトが役に立つので是非活用していただきたい．

- Java（tm）House Mailing　http : //java-house.etl.go.jp/ml/
  日本国内で早くから運営されている Java 関係のメイリングリストの主催ページ．過去の議論が簡単に検索できるようになっている．
- BugDatabase　http : //developer.java.sun.com/developer/
  本家 JavaSoft が提供する Bug データベース．バグ情報が満載されている．

8. パフォーマンス・チューニング

開発するプログラムは，要求される性能要件を満たさなければならない．このためのチューニングを行うための材料を採取する方法には，Java プロファイラー・ツールの活用と，各 OS の資源関係コマンドの活用がある．前者は，Java プログラムを解析し，パフォーマンスのボトルネックを明らかにする．後者は，プラットフォームに於ける Java プロセスの資源の利用状況を明らかにする．

Java プロファイラ・ツールについては，OptimizeIt（http : //www.optimizeit.com）や JProbe（http : //www.klg.com）などがあり，両者はともにグラフィカルなレポート機能を提供する．WISE 開発においては，OptimizeIt を用いて，クライアント Applet のチューニング作業を行っている（CORBA 経由の本番プログラムは上手く動作しなかったため，注目点をいくつかのサンプルプルグラムにして解析結果を反映する方式を採った）．OptimizeIt には，Java プロセス内の，メモリ消費状況を調査する「メモリ・プロファイラ」と，CPU 利用状況を調査する「CPU プロファイラ」が含まれており，アロケートされたオブジェクトのリストや，そのオブジェクトに対する参照関係の表示したり，あるスレッド内でホットスポットとなるメソッドを見つけだし表示してくれる．Java プログラムでは，一般的にオブジェクトの生成や，ガーベッジ・コレクションは負荷が高いと言われており，無駄な一時的なオブジェクトを生成することは極力避けた方が良い. これらの振る舞いもツールを使えば良く理解できるので，本番プログラムへの反映を行うことでパフォーマンスの改善につながる．

資源関係コマンドについては, Java プロセス自身が OS 内部で最適に動作するように調査を行うために使用する. コマンドは, プラットフォームによって異なるが, 勤怠システム開発においては, サーバ機に Solaris を搭載しているため, UNIX の vmstat, mpstat，フリープログラムの se（http : //www.sun.com/sun-on-net/performance/se3/）や top（http : //www.sunfreeware.com）コマンドを使用している．これらのコマンドを使用することで，プラットフォーム上のプロセッサ毎またはプロセス毎の CUP 利用状況や，仮想メモリ状況，プロセス毎のスレッド数などを明らかにすることができる．勤怠システム開発の経験では CPU の利用率が他のプロセスに対し圧倒的に多いことが判明し，サーバ機の CPU を増設したり，Application サーバと，Database サーバを物理的に分割するなどの処置を施している．

パフォーマンスを向上させる方法としては，Java プログラムをネーティブコンパイラ（TowerJ（http : //www.twr.com）などが有名）を採用する方法もある．これ

は，Java バイト・コードを各プラットフォームに最適化された実行コードにコンパイルするもので，特定のプラットフォームに特化してしまう変わりに実行性能を向上させるものである．勤怠システム開発においても，サーバプログラムのチューニング候補として検討課題には挙がったが時間的な問題もあり，評価するまでには至らなかった．

9. 日本語文字の扱い

GUI 上のラベル文字や，テキストフィールドに入力する値など，日本語処理は必須要件である．

Java における日本語処理の問題は，バージョン 1.1 の国際化対応において解決されたと理解されている方も多いと思われるが，まだエンコーディング・タイプ間の非互換問題が残されている．

GUI を例に取って説明する．一般的に GUI を使うアプリケーションは Windows プラットフォームが多いと思われるが，この場合，Microsoft 社が使用している SJIS（Miscrosoft キャラクタセット）と UNICODE の変換マップ（MS 932）と JIS が規格する変換マップ（SJIS）は若干異なるようである．一見 GUI 上では問題なく表示されているようだが，内部的には JIS 規格に合わない形で，UNICODE にマップされているのである．従って，これらを JIS 規格に従った，他のエンコーディング・タイプ（例えば ISO 2022 JP）を使用すると変換不能文字（上記の環境の場合「？（0 x 3 f)」）になってしまう（表 3)．

表 3　エンコーディング・タイプにより文字化けが発生するコード

	JIS コード	SJIS コード	(MS932 encoding) UNICODE	(SJIS encoding) UNICODE	(ISO2022JP encoding) UNICODE
1 区 33 点　～	2141	8160	FF5E	301C 注 1	301C 注 1
1 区 34 点　∥	2142	8161	2225	2016 注 1	2016 注 1
1 区 61 点　−	215D	817C	FF0D	2212 注 1	2212 注 1
1 区 81 点　¢	2171	8191	FFE0	00A2 注 2	00A2 注 2
1 区 82 点　£	2172	8192	FFE1	00A3 注 2	00A3 注 2
2 区 44 点　¬	224C	81CA	FFE2	00AC 注 2	00AC 注 2

注 1	MS932 エンコーディングを基本とした場合、他のコードに変換する場合は「3F」に置換される。他コードを、SJIS、ISO2022JP でエンコーディングした際の Unicode を解釈できず、GUI 上では「？」表示される。
注 2	MS932 エンコーディングを基本とした場合、他のコードに変換する場合は「3F」に置換される。他コードを、SJIS、ISO2022JP でエンコーディングした際の Unicode は解釈できるようで、GUI 上では正しく表示される。

表 3 では GUI を例に挙げたが，同じような問題が，JDBC や CORBA 経由のシステムを構築する際，発生する可能性がある．

勤怠システム開発における対処策を以下に説明する．

- JDBC（利用者データベースとの連携の際に使用）
  使用文字の制限．使用するデータに上記のコードが含まれないことを事前に確認した．

- Mail（督促状をメール配信する際に使用）
  使用文字の制限．オンラインマニュアル等に記載することで対処した．
- CORBA（メソッド起動の全角文字の受け渡しの際に使用）
  エンコーディング・タイプを MS 932 に統一．CORBA 経由の全角文字列データの受け渡しを octet シーケンスで行う場合，クライアント/サーバ間でエンコーディングを統一することにした．使用頻度やオペレーション等を加味し，Microsoft キャラクターセットに他を合わせることとした．

以上は，すべて回避策であり，本質的な解決策では無いことに注意していただきたい．

10. 帳票機能の実現

業務システム開発において，帳票は欠かせない機能の一つである．

勤怠システム開発では，ペーパーレスが一つの目的であり，当初帳票機能は必要ないと考えていた．しかし，利用者からの要求は多く現在検討を行っている．

Java の印書機能そのものは，下記の例に示すようにバージョン 1.1 から含まれているが，API 的には非常に低水準なものであり，帳票を出すためにはかなりのコードを作成する必要がある．また，一部のプラットフォームにおいてはプリンタ情報を受け取れないなど帳票を作成するためには問題が多かった（java.awt.print パッケージなどで順次補完されつつある）．

最近の構築事例などでも，Java の印書機能は使用せず，CGI などで外部システムと連携し，何らかの形で印書する方式を取っているものが大方である．

```
PrintJob pj = this.getToolkit().getPrintJob(this,"Image Print",(Properties)null);
Graphics pg = pj.getGraphics();
If(pg != null){
   //Graphics メソッドでコンテキストに描画
   pg.dispose();
}
pj.end();
```

11. お わ り に

本稿では，Java と CORBA という先進技術を使った基幹システム開発に携わった経験から，現在の Java 技術の適用について問題点と対処を報告した．特に Java Applet については，雑誌等でいろいろな問題点が指摘されてきたが，報告にもあるように Java 2 の登場により，その多くが解決できることがおわかり頂けたと思う．

最近では，EJB（Enterprise Java Beans），Servlet，JSP（Java Server Pages）などを使ったサーバ・サイド構築が注目されており，今後は，これらの技術を検討し，次システム開発への適用を図りたいと思う．

執筆者紹介　山　本　史　朗（Fumiaki Yamamoto）
1968 年生．1991 年東海大学工学部通信工学科卒業．同年日本ユニシス(株)入社．U-6000 ホスト通信エミュレータの開発，営業支援を経て，現在，生産技術部情報技術室所属．Java を用いたシステム開発支援に従事．