T-JV入門

はじめに

Javaは、Sun Microsystems Inc.社が開発したオブジェクト指向プログラミング言語です。Javaで開発されたアプリケーションはプラットフォームの差を意識することなく実行できることを目指して設計されています。現在、PCの世界だけでなく、ほとんどの携帯電話にはJavaが搭載されているように、組込みの世界でもJavaが急激に利用されてきています。

Java 2プラットフォームには、Java 2 Enterprise Edition（J2EE）、Java 2 Standard Edition（J2SE）、Java 2 Micro Edition（J2ME）の3つのシリーズがあり、組込みの世界で主に利用されているのは、このうちJ2MEです。J2MEは、Connected Device Configuration（以下CDC）とConnected Limited Device Configuration（以下CLDC）という2種類のコンフィギュレーションに分かれており、CDCはカーナビやPDA等の比較的ハイエンドな環境をターゲットとしたコンフィギュレーション、CLDCは従来の携帯電話をはじめとして比較的ローエンドな環境をターゲットとしたコンフィギュレーションとなっています。今回ご紹介する2つのT-Engine向けのJava実行環境（以下T-JV）は、双方ともJ2ME CDCをベースとしたものです。今回は、それぞれのT-JVのビルド、インストールおよびアプリケーションの実行方法についてご説明します。

T-JVの入手方法

T-JVは、T-EngineフォーラムのA会員、B会員、および学術会員に公開されています。T-JVを入手するためには、各会員向けサイトに設けられたリンクからT-JVのダウンロード申請サイトへ移動し、ライセンス契約に同意した場合のみ入手することが可能となります。入手するためには、主に以下の2つの項目を満たす必要があります。

・Sun Community Source License（SCSL）のCDC/FPおよびPersonal Basis Profile（以下PBP）にSunのサイト上で同意済であること（SunのサイトでCDC/FPおよびPBPの各ソフトウェアをダウンロードしようとするとSCSLへの同意を求められますので、内容を確認して同意する場合は「accept」ボタンをクリックします）

・T-EngineフォーラムのA会員、B会員、または学術会員であること

これらの条件を満たした者に対して、研究評価目的に限り、T-JVを複製、使用および改変することが許諾されています。

では、各T-JVの利用方法をご説明いたします。

T-JV（Aplix＋UNL）

T-JV（Aplix＋UNL）は、株式会社アプリックスとYRPユビキタ

ス・ネットワーキング研究所が共同で構築したT-Engine上のJava実行環境であり、T-Engine/SH7727上で動作します。

●ビルド環境のインストール

T-JVの入手方法の節で示したライセンスに同意し、T-JV（Aplix＋UNL）を入手して解凍すると、下記ファイルが展開されます。

・jbstart_te_sh7727.tar.gz
　実行イメージのビルド環境
・jcnlib_te_sh7727.tar.gz
　VMライブラリ部分のビルド環境
・t-jv_aplix.unl_readme_v010000.pdf
　日本語マニュアル
・t-jv_aplix.unl_readme_v010000_en.pdf
　英語マニュアル

開発ホストとなるLinuxマシン上に、「jbstart_te_sh7727.tar.gz」および「jcnlib_te_sh7727.tar.gz」の各圧縮ファイルを展開します。

```
% tar xvfz jbstart_te_sh7727.tar.gz......
% tar xvfz jcnlib_te_sh7727.tar.gz......
```

T-JV（Aplix＋UNL）のビルド環境は、VMライブラリ部分のビルド環境と実行イメージのビルド環境に分かれており、互いに独立しています。VMライブラリ部分のビルドは、VMライブラリの生成だけを行い、実行イメージのビルドは、ターゲットデバイス依存部分を定義しており、これとVMライブラリをリンクして実行イメージを生成します。ネイティブメソッドの組込みなど、カスタマイズの多くは実行イメージのビルド環境のみの変更で行えます。

●VMライブラリ部分のビルド

VMライブラリ部分をビルドするためには、開発ホスト上に「T-Engine/SH7727開発キット GNU開発環境」、「Java 2 SDK, Standard Edition, v1.3.1」がインストールされ、環境変数（BD、GNUs、GNU_BD、GNUsh、GCC_EXEC_PREFIX）が正しく定義されている必要があります（表1）。

表1　VMライブラリ部分のビルドに必要なもの

・T-Engine/SH7727 開発キット GNU 開発環境（パーソナルメディア）
　– Linux 用プラットホーム共通部分（Rel 04）
　– Linux 用 SH 対応部分（Rel 04）
　– T-Kernel リソース部分（Rel 1.4）
・Java 2 SDK, Standard Edition, v1.3.1
・上記開発環境がインストールされた Linux マシン
・VM ライブラリ部分のビルド環境

●VMライブラリ部分のビルド手順

「jcnlib_te_sh7727/cvm.sh」の中で、「JDK_HOME」を指定している以下の部分を、「Java2 SDK, Standard Edition, v1.3.1」をインストールしているディレクトリに修正します。

```
export JDK_HOME=/tools/java/jdk1.3.1
```

ディレクトリ「jcnlib_te_sh7727」でmakeを実行します。

```
% cd jcnlib_te_sh7727
% sh cvm.sh
```

ビルドに成功すると、「jcnlib_te_sh7727/build/TE_SH7727/bin/libjcn.a」が生成されます。これがVMライブラリになります。

ビルドしたVMライブラリを実行イメージに反映させるため、「jcnlib_te_sh7727/ build/TE_SH7727/bin/libjcn.a」を「jbstart_te_sh7727/jcn/lib」ディレクトリにコピーして、実行イメージのビルドを行います。

●実行イメージのビルド

実行イメージをビルドするためには、開発ホスト上に「T-Engine/SH7727開発キット GNU開発環境」、「KASAGO for T-Engine開発キット」がインストールされ、環境変数（BD、GNUs、GNU_BD、GNUsh、GCC_EXEC_PREFIX）が正しく定義されている必要があります（表2）。

表2　実行イメージのビルドに必要なもの

・T-Engine/SH7727 開発キット GNU 開発環境（パーソナルメディア）
　– Linux 用プラットホーム共通部分（Rel 04）
　– Linux 用 SH 対応部分（Rel 04）
　– T-Kernel リソース部分（Rel 1.4）
・KASAGO for T-Engine（エルミックシステム）
　– KASAGO for T-Engine 開発キット SH7727 用 V3.78.05
・上記開発環境がインストールされた Linux マシン
・実行イメージのビルド環境

●ビルド時の環境設定

実行するアプリケーションの各パラメータ情報を「jbstart_te_sh7727/h/jbstart.h」に記述して、実行イメージのビルドを行う必要があります（表3）。

また、ネットワークに接続する場合は、同様にネットワーク情報を「jbstart_te_sh7727/h/jbstart.h」に記述して、実行イメージのビルドを行う必要があります（表4）。

表3　アプリケーションパラメータの設定

設定値	初期値	説明
DEFAULT_CLASSPATH	"/SYS/classes"	Java クラスパスを指定します。ディレクトリのほかにjar ファイルや zip ファイルも指定できます。 複数のパスを指定する場合はコロン（“：”）で区切って並べてください。
DEFAULT_VMARGS	""	VM の起動オプションを指定します。
DEFAULT_APPNAME	"aplix.pbp.TestLauncher"	起動するアプリケーションのメインクラス名を指定します。
DEFAULT_APPARGS	""	アプリケーションの引数を指定します。
DEFAULT_HOMEPATH	"/SYS/home"	ホームディレクトリを指定します。 ここで指定されたディレクトリが、Java プログラム中での相対パス指定時のカレントディレクトリとなります。 また、この文字列は java.lang.System.getProperty() で key 値を“user.home”、“user.dir”とした場合に取得できるディレクトリ名となります。

表4　ネットワーク情報の設定

設定値	初期値	説明
NETWORK_IPADDR	"192.168.10.10"	T-Engine ボードの IP アドレスを設定します。
NETWORK_NETMASK	"255.255.255.0"	ネットマスクを設定します。
NETWORK_GATEWAY	"192.168.10.0"	デフォルトゲートウェイの IP アドレスを指定します。
NETWORK_1STDNS	"192.168.10.1"	プライマリ DNS サーバのアドレスを指定します。
NETWORK_2NDDNS	"192.168.10.2"	セカンダリ DNS サーバのアドレスを指定します。
NETWORK_HOSTNAME	"host.domain.com"	T-Engine ボードのホスト名を設定します。

●実行イメージのビルド手順

実行イメージ部分を展開したディレクトリ下の、「jbstart_te_sh7727/sh7727/」でmakeを実行します。

```
% cd jbstart_te_sh7727/sh7727
% make
```

ビルドに成功した場合、「jbstart_te_ sh7727/sh7727/jbstart」がT-JVの実行イメージになります。

●ターゲットへのT-JVのインストール

ターゲットへT-JVをインストールし、実行するためには、表5に示す各ハードウェア／ソフトウェアが必要になります。

ターゲットへT-JVをインストールするには、まず作成した「jbstart」をT-Engineボードの「/SYS」ディレクトリに転送します。たとえば、recvコマンドで転送する場合T-Engineボードに接続したターミナルで以下のように実行します。

```
[/SYS]% recv -d jbstart
```

T-JVを実行する際には、CLIのプロンプトから、「jbstart」を起動します。

```
[/SYS]% lodspg jbstart
```

なお、デフォルトのままの設定では、アプリケーションのクラス（aplix.pbp.TestLauncher）が存在しないため、「java.lan g.NoClass DefFoundError: aplix.pbp.TestLauncher」と出力されて終了してしまいます。

表5　T-JVの実行に必要なもの

- T-Engine/SH7727 開発キット T-Engine ボード（パーソナルメディア）
- 上記ハードウェアにシリアル接続したターミナル（PC など）
- T-Engine ボード用起動ディスク（コンパクトフラッシュあるいは USB メモリ）
  - ーターゲット側ソフトウェアは Version 1.1.02 を用いる必要があります
- T-Engine/SH7727 拡張 LAN ボード（パーソナルメディア）
  - ー存在しなくても T-JV は起動しますが、ネットワーク機能は使用できません
- KASAGO for T-Engine（エルミックシステム）
  - ー存在しなくても T-JV は起動しますが、ネットワーク機能は使用できません
- Java 実行環境の実行イメージ（jbstart_te_sh7727/sh7727/jbstart）
- 実行する Java アプリケーションのクラスファイル
  - ー CLI より参照できるファイルシステム上に配置してください

●サンプルプログラム

ここでは、「Hello T-Engine World」とターミナル上に出力するプログラムをファイル名「HelloWorld.java」で作成し、T-JV上で実行するまでの例を示します。

下記内容の「HelloWorld.java」を作成します。

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out. println( “Hello T-Engine
World” );
  }
}
```

ソースファイルをコンパイルします。

```
% javac HelloWorld.java
```

このサンプルでは、実行するアプリケーションのクラスが「HelloWorld」なので、「jbstart.h」の中で定義されているデフォルト値を変更して、実行イメージを再構築します。

```
#define DEFAULT_CLASSPATH  “/SYS/classes”
#define DEFAULT_VMARGS  “”
#define DEFAULT_APPNAME  “HelloWorld”
#define DEFAULT_APPARGS  “”
#define DEFAULT_HOMEPATH  “/SYS/home”
```

コンパイルされた「HelloWorld.class」をT-Engineに転送し、「/SYS/classes」の下に置くとともに、再構築した「jbstart」についても転送します。

```
[/SYS] cd classes
[/SYS/classes] recv HelloWorld.class
[/SYS] recv -d jbstart
```

T-JVは、下記コマンドにより起動します。

```
[/SYS] lodspg jbstart
```

実行結果が以下のように出力されます。

```
[/SYS] lodspg jbstart
      ……（起動ログ）……
Hello T-Engine World
JCN[info] javavm_start: returns 0
jbstart: Can't Loaded System Program -327680
```

T-JV（Aplix＋UNL）をダウンロードしていただくと、ここで紹介した以外に、ネイティブメソッドを利用したデバックボードの8ビットLEDを操作するサンプルプログラムがの実装方法が添付されています。

T-JV（Sun Microsystems＋UNL）

T-JV（Sun Microsystems＋UNL）は、サン・マイクロシステムズ株式会社とYRPユビキタス・ネットワーキング研究所が共同で構築したT-Engine上のJava実行環境であり、T-Engine/SH7727上で動作します。

●ビルド環境のインストール

ライセンス条項に同意し、T-JV（Sun Microsystems＋UNL）を入手して解凍すると、下記ファイルが展開されます。

・cdcfoundation.tar.gz
　CDC/FP＋TRON Codeプロファイルのソースコード
・cvm_bin.tar.gz
　コンパイル済みのJava実行環境
・CVM_DEMO.BPK
　マイクロスクリプトを用いたデモプログラム
・t-jv_buildenv.tar.gz
　T-JV専用ビルド環境
・t-jv_sun.unl_readme_v010000.pdf
　日本語マニュアル
・t-jv_sun.unl_readme_v010000e.pdf
　英語マニュアル

T-JVをビルドするためには、「T-JV専用ビルド環境」をLinux上にインストールする必要があります。すでにパーソナルメディア株式会社より配付されているGNU開発環境がインストールされている場合は、この「T-JV専用ビルド環境」と置き換える必要があります。そのためすでにGNU開発環境がインストールされている場合は、GNU開発環境をバックアップします。

```
$ su
Password:
# cd /usr/local
# mv te te_bak
```

T-JV専用ビルド環境のアーカイブ「t-jv_buildenv.tar.gz」を「/usr/local/te」に展開します。また、環境変数（BD、GNUs、GNU_BD、GNUsh、GCC_EXEC_PREFIX）が「T-JV専用ビルド環境」に対して定義されている必要があります。

```
# tar zxpf t-jv_buildenv.tar.gz
# exit
$
```

●ビルド時の環境設定

T-JVのビルド時には、「ksh（Korn Shell）」が必要になるため、

開発ホストにkshがインストールされていない場合はインストールしなければなりません。

ここでは、「pdksh（the Public Domain Korn Shell）」をインストールすることを想定して説明します。pdkshの公式ページから「pdksh-5.2.14.tar.gz」をダウンロードし、解凍します。さらに、展開したディレクトリに入り、以下の手順でインストールを行います。

```
$ su
Password:
# tar zxpf pdksh-5.2.14.tar.gz
# cd pdksh-5.2.14
# ./configure
# make
# make install
# exit
$
```

●ターゲット環境の設定

フラッシュメモリイメージは、Version.1.1.02を用います。T-Engineのフラッシュメモリにはデフォルトでは古いバージョンが入っている可能性があるので、これを書き換える必要があります。

また、T-JVの実行にはT-Shellが必要であるため、起動ディスクにT-Shellをインストールする必要があります。インストール方法の詳細は、「PMC T-Shell/SH7727開発キット 取扱説明書」を参照してください。

●T-JVのビルド

T-JVのソースコードは「cdcfoundation.tar.gz」に含まれています。これを展開しビルドする前に、これまでに説明した環境構築をすませておきます。

まず、「cdcfoundation.tar.gz」を展開します。

```
$ tar zxpf cdcfoundation.tar.gz -C .
```

JDK をインストールした場所に応じて、「cdcfoundation/build/btron3/defs.mk」の「JDK_HOME」を書き換えます。たとえば、「/usr/java/jdk1.3.1_11」にインストールされている場合は、下記のように修正します。

```
...
USE_SUNCC_CLEANUP_ACTION =
$(DEFAULT_CLEANUP_ACTION)
HAVE_64_BIT_IO_CLEANUP_ACTION =
$(DEFAULT_CLEANUP_ACTION)
GPROF_CLEANUP_ACTION = $(DEFAULT_CLEANUP_ACTION)
JDK_HOME = /usr/java/jdk1.3.1_11
CVM_JAVABIN = $(JDK_HOME)/bin
...
```

ビルドは、「cdcfoundation/build/btron3-sh3」ディレクトリに移動しmakeを行います。このとき、

・CVM_DEBUG=true
・J2ME_CLASSLIB=foundation

の2つのパラメータを必ず指定してください。

```
$ cd cdcfoundation/build/btron3-sh3
$ make CVM_DEBUG=true J2ME_CLASSLIB=foundation
```

ビルドが成功すれば、以下のファイルが生成されます。

・cdcfoundation/build/btron3-sh3/bin/cvm
・cdcfoundation/build/btron3-sh3/bin/cvm.map
・cdcfoundation/build/btron3-sh3/lib/foundation.jar
・cdcfoundation/build/btron3-sh3/lib/security/java.policy
・cdcfoundation/build/btron3-sh3/lib/security/java.security

●T-JVのインストール

ビルドの結果生成されたファイルを、T-Engine上の「/SYS/WORK/java」というディレクトリ下にインストールするために、まず、「cdcfoundation/build/btron3-sh3」に移動します。

```
$ cd cdcfoundation/build/btron3-sh3
```

T-Engineを起動し、T-Engine上に「/SYS/WORK/java」ディレクトリを作成し、ビルドされた5つのファイルを開発ホスト上の構成のままrecvコマンドを用いてコピーします。

```
[/SYS]% mkf WORK
[/SYS]% cd WORK
[/SYS/WORK]% mkf java
[/SYS/WORK]% cd java
[/SYS/WORK/java]% mkf bin
[/SYS/WORK/java]% cd bin
[/SYS/WORK/java/bin]% recv -d bin/cvm
[/SYS/WORK/java/bin]% recv -d bin/cvm.map
[/SYS/WORK/java/bin]% cd ..
[/SYS/WORK/java]% mkf lib
[/SYS/WORK/java]% cd lib
[/SYS/WORK/java/lib]% recv -d lib/foundation.jar
[/SYS/WORK/java/lib]% mkf security
[/SYS/WORK/java/lib]% cd security
[/SYS/WORK/java/lib/security]% recv -d lib/
security/java.policy
[/SYS/WORK/java/lib/security]% recv -d
lib/security/java. security
```

T-Kernelの環境変数を編集します。ここでは、「/SYS/SYSCONF」中の「TMaxSemId」および「TMaxMtxId」の値をいずれも500に変更します。gterm上から「SYSCONF」ファイルを編集するには、edコマンドを用います。コマンドの詳細は、「T-Engine/SH7727開発キット 取扱説明書」を参考にしてください。

```
...
TMaxSemId 500 #最大セマフォ数
...
TMaxMtxId 500 #最大ミューテックス数
...
```

以上で、T-JVのビルドおよびインストールが完了しました。

●サンプルプログラム

ここでは、「Hello T-Engine World」とターミナル上に出力するプログラムをファイル名「HelloWorld.java」で作成し、T-JV上で実行するまでの例を示します。

下記内容の「HelloWorld.java」を作成します。

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println( “Hello T-Engine World” );
  }
}
```

ソースファイルをコンパイルします。

```
% javac HelloWorld.java
```

T-Engineを起動します。T-Engine上の適当な場所に、「HelloWorld.class」を開発ホストからrecvコマンドにより受信します（ここでは、「/SYS/WORK/java/test」というディレクトリを利用したものとして説明します）。

```
[/SYS/WORK/java/test]% recv -d HelloWorld.class
Target: HelloWorld.class
.. [HelloWorld.class: 680 bytes]
! (680.0 bytes/sec)
[/SYS/WORK/java/test]%
```

「/SYS/WORK/java/bin/cvm」を用いて、Javaプログラムを実行します。ここでは、引数に以下の項目を指定します。

・-Djava.class.path=「実行するプログラムの存在するディレクトリの絶対パス」
・「実行するクラス名」

```
/SYS/WORK/java/bin/cvm -Djava.class.path=/SYS/
WORK/java/test HelloWorld
```

正しく実行されると、ターミナル上に以下のように表示されます。

```
[/SYS/WORK/java/test]% /SYS/WORK/java/bin/cvm -
Djava.class.path=/SYS/ WORK/java/test HelloWorld
......（起動ログ）......
!ello T-Engine World
[/SYS/WORK/java/test]%
```

なお、ここで表示される「!」はTRONコードの言語指定コードによるものです。

T-JV（SunMicrosystems＋UNL）をダウンロードしていただくと、ここで紹介した以外にも、T-Shellと連携し、マイクロスクリプトのGUIを通じてJavaプログラムが簡単な文字列操作を行うプログラム（図1）やネットワーク機能を利用したプログラムの実装方法が添付されています。

図1　マイクロスクリプトとの連携プログラム

終わりに

今回は、2つのT-JVのビルド、インストール方法、および標準出力に文字を出力する簡単なJavaアプリケーションの構築についてご説明いたしましたが、実際にT-JVをダウンロードしていただきますと、さらに詳しいマニュアルとサンプルアプリケーションが付属していますので、T-EngineフォーラムのA会員、B会員、学術会員の方はぜひ試してみてはいかがでしょうか。