AIR for Android 開発メモ

2011/01/22

吉岡梅(yoshioka@ciruelo.jp)

AIR for Android をプレリリース時からテストしてきた体験から、AIR for Android に関するよもやま話を集めてみました。基本的な仕組みや、スマートフォンのデバイスにアクセスするための方法などは他の方が解説してくださっているかと思いますので、その他の話題や、個人的にハマって苦労した点に的を絞ってご紹介します。まとまりもとりとめもなく、裏をきちんと取ってないので間違いなどもあるかもしれないレポートですが、悪しからずご了承ください。

1. AIR for Android の歩みと現状

AIR for Android(以下、「AFA」)の開発はどのような経緯で進んできたのでしょうか。以下、ざっとまとめてみました。

1	iPhone 発売、大人気！当時は iOS3.0
2	Adobe の AIR チームさん、iPhone 用に Flash で作成したアプリを書き出せる、 「Packager for iPhone(PFI)」の開発スタート。
3	PFI のベータテスト始まる。 ←私はこのへんから参加
4	PFI が目玉機能の FlashCS5 の発売決定！PFI は AIR2.0 相当の機能。
5	AIR チーム、Android 向けの AIR「AIR for Android(AFA)」の開発にも着手。 確か当時の Android の OS は 2.0(Éclair)、対応 AIR バージョンは 2.0。
6	Apple 社、CS5 発表 **1 日前**にサードパーティ製のコンパイラで作成した iOS 向けアプリの登録を禁止。事実上、PFI が死に機能に。本当にびっくりしました。
7	CS5 発売。
8	AIR チームさん、iOS 向けの開発をストップし、AFA へ注力。
9	Google、Android 向けの OS2.2(Froyo)を発表(この辺時系列あいまい)。
10	AIR チーム、AFA を Froyo 向けに変更、さらに対応 AIR バージョンを 2.5 に。パフォーマンスが一気に上がる。本当に一気に。「これはいけるわー」という感触。
11	AFA リリース！
12	Apple 社、アプリ登録の規約を変更。PFI のアプリでも登録可能に。 が、この時点で PFI は AIR2.0 ベースのままストップ中。Android の 2.5 ベースと差が生まれる。
13	Apple 社、iOS を 4.0 に更新。同時にアプリのアップロードには専用のアップローダー(MAC の Cocoa 環境で動作)となる。事実上、MAC がないと登録不可に。
14	CS5 の PFI で書き出した iOS 向けファイルは Application Loader ではアップロードできないことが判明。
15	Adobe さん、PFI アップデート。PFI で書き出したアプリもアップロードできるように。 Flash Professional CS5 Extension for AIR 2.5 もリリース。

と、いうわけで、現在は Flash では iOS 向け、Android 向けの両方のアプリが書き出せるようになっていますが、一時期 iOS 向けの開発がストップしていたため、iOS 向けと Android 向けでは、使用できる機能に差があります。

■ **Flash でアプリの作成できるスマートフォン向け OS**

OS	AIR	説明
iOS3.0 以上	2.0 相当	CS5 に Packager for iPhone のアップデートを適用すると、AIR2.0 相当の機能を使ったアプリの開発が可能 また、Windows でもアプリ開発自体はできるが、appStore に登録する場合には、MAC が必要
Android2.2 以上	2.5 相当	CS5 に AIR for Android エクステンションを組み込むと、AIR2.5 相当の機能を使ったアプリの開発が可能

iOS 向けのアプリ書き出しは、AIR2.0 ベースです。それに対して Android 向けのアプリ書き出しは AIR2.5 ベースです。AIR2.0 と AIR2.5 の機能の差を抜粋すると、以下のようなものが挙げられます。

■ **AIR2.5 から使用できる機能**

機能	カメラへのアクセス マイクへのアクセス StageWebView の利用 SD カードへのアクセス　‥等

つまりは、現状では iOS ではカメラ機能を使ったアプリ作成等ができないわけですね。
ただ、この差はすぐに埋められると思われます。
ちなみに、Android OS のバージョンは次のような感じでバージョンアップしています。コードネームはアルファベット順だそうですが、このままだとすぐ Z まで行くんじゃないか、という恐ろしい勢いです。バージョンアップの速度が超早いので、いろいろ検証が大変になりそうですね。

■ **Android のバージョン**

コードネーム	バージョン	メモ
Donut	1.6	ドーナツ
Eclair	2.1	エクレア
Froyo	2.2	フローズンヨーグルト AIR for Android が動作するのはここから
Gingerbread	2.3	ジンジャーブレッド 2011 年はこれが主流になるのかな？
Honeycomb	3？	ハニカム タブレット用と言われているが、詳細不明

2. Android アプリを作成する際にそろえておきたい環境

Flash を使って Android のアプリを開発する際には、いろいろと準備が必要です。単に Flash から Android 向けのアプリを書き出すだけであれば、エクステンションひとつをダウンロードして組み込めば済むのですが、同じ PC 上でいろいろなテストを行いたい場合には、それだけではちょっと足りないのです。

そもそも、Android アプリというものは、Java で作成されてきました。そのため、Java 環境ではいろいろな便利なツールが揃っているのです。これらのツールの一部は、Flash を使ってアプリを作成する際にも有用なのです。

と、いうわけで、「Java 環境を作る」「Java 環境で使える Android アプリ作成用のツールを用意する」という 2 段構えで環境を作成することとなります。

■ 準備する 2 段階の環境

環境	目的
JDK (Java Development Kit)	Java を使った開発行うための環境 Android SDK をインストールして実行する際に必要 ※win の場合は 64 ビットマシンでも 32 ビット用の物をインストールしておいた方が無難だそうです
Android SDK	Android 開発者向けの様々なツール

とりあえず、以下に win の場合の設定をざっとメモしておきます。また、このあたりの環境づくりに関しては、web 等で詳しい方法を検索していただいた方が確実かもしれません。

Java 開発環境を準備する

まずは Java 開発環境を準備します。Oracle さんから JDK をダウンロードして指示に従ってインストールします。

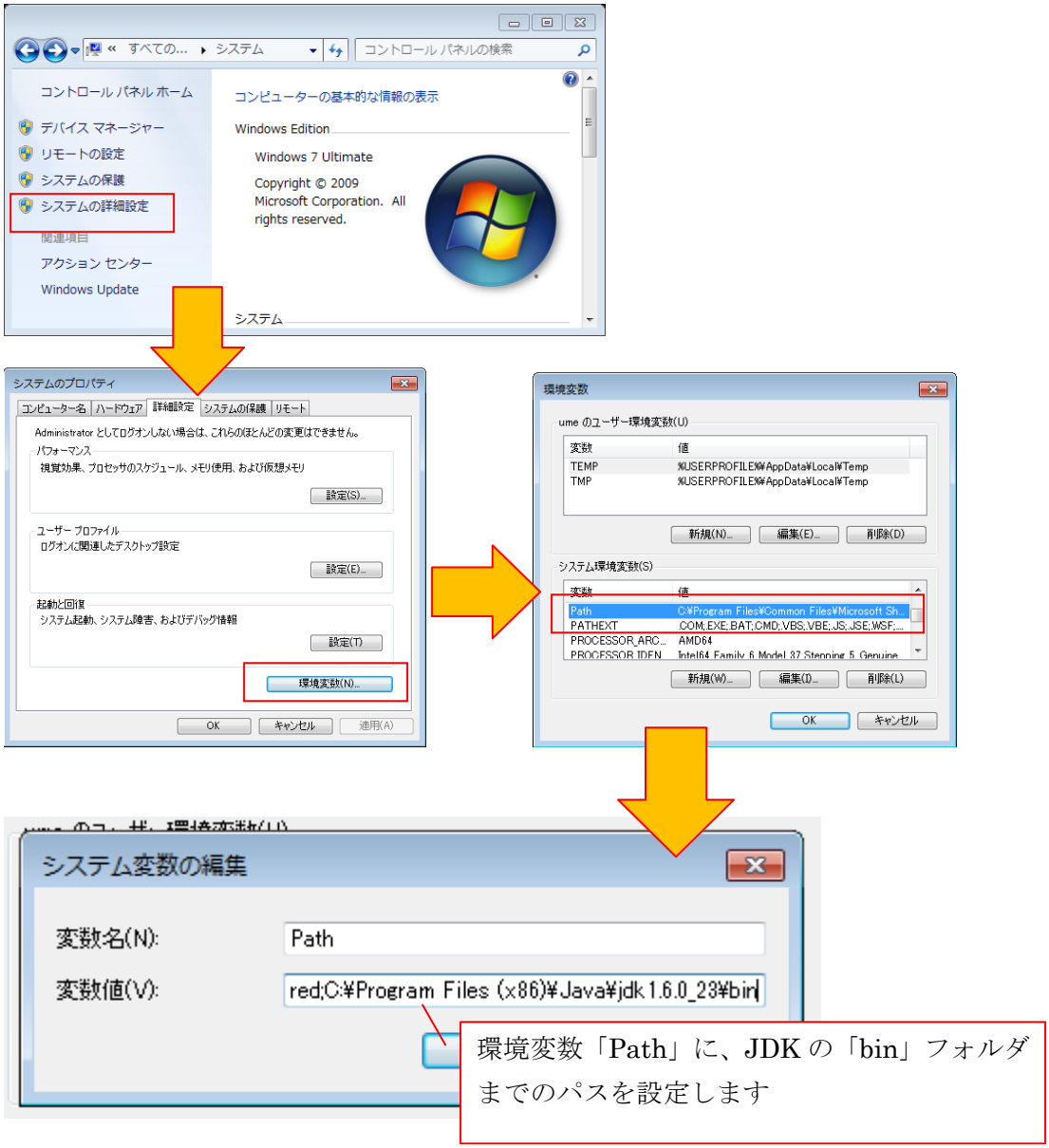
⇒ http://www.oracle.com/technetwork/java/javase/downloads/index.html

インストール後に、JDK の bin フォルダへのパスを、Windows の環境変数「Path」へと設定します。win のメニューより、[コンピュータ]を選択し、[システムのプロパティ]を選択し、さらに左側のメニューから「システムの詳細設定」を選択します。

[システムのプロパティ]ダイアログが表示されますので、「詳細設定」タブを選択し、[環境設定]ボタンを押します。[環境設定]ダイアログが表示されますので、「システム環境設定」欄から「Path」という項目を探して選択し、[編集]ボタンを押します。

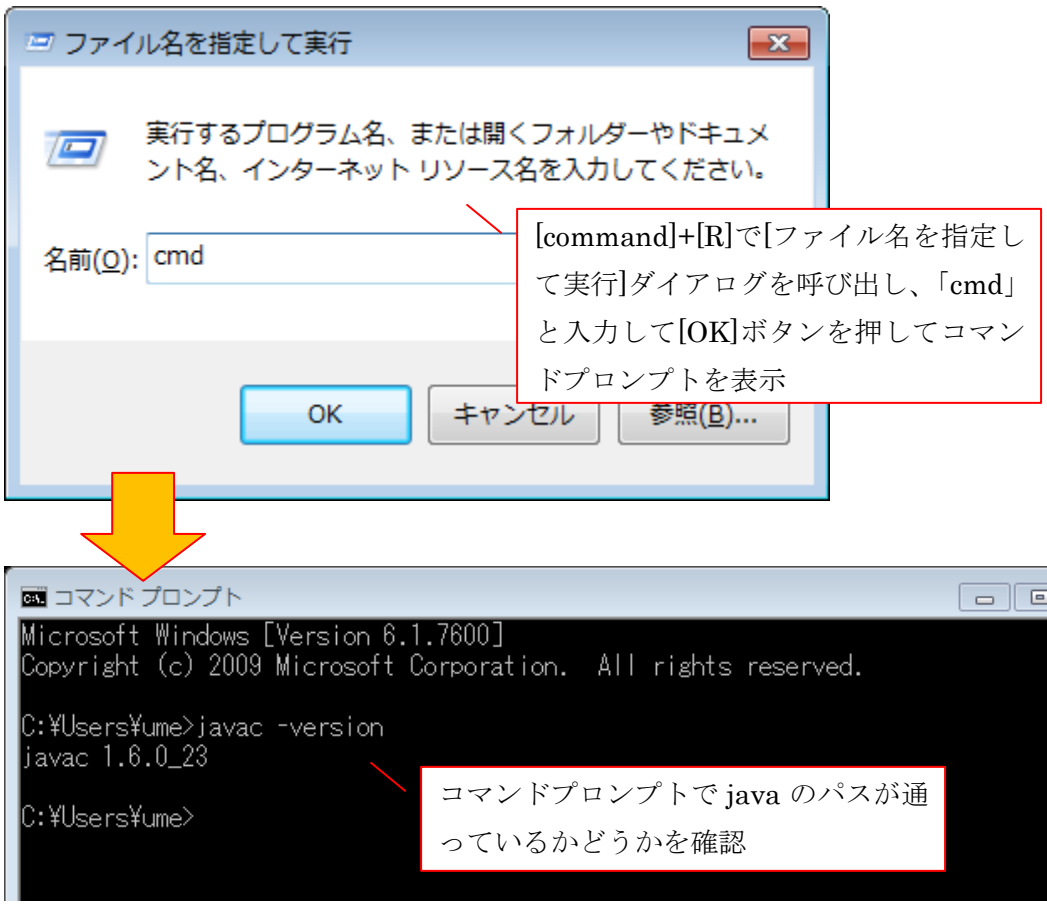
すると、やっと[システム変数の編集]ダイアログが表示されます。お疲れ様！もうちょいです。おそらくこの時点で「編数値」欄には、既に何らかの文字列が入力されているかと思います。この末尾に、区切り文字である「;」を加え、さらに JDK の bin フォルダへのパスを追加します。これで完成です。

■　環境変数を通す



さて、パスがきちんと通ったかどうかを確認してみましょう。[cmd]+[R]で「cmd」入力などの方法でコマンドプロンプトを呼び出し、「javac -version」と入力して[Enter]キーを押します。ここで、「javac 1.6.0_23」等のバージョン番号が表示されれば OK です。

■ javac –version でパスが通っているかを確認



Android SDK を準備する

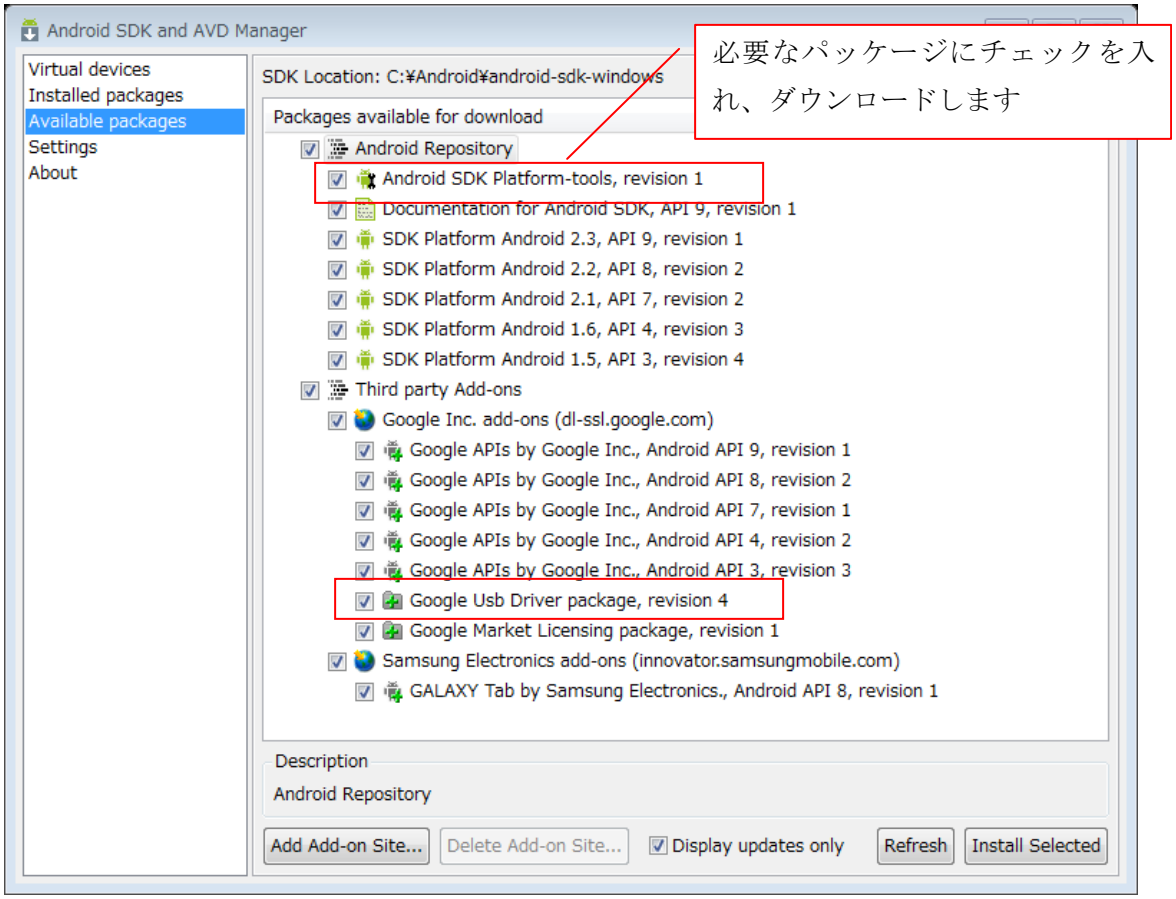
Java の開発環境が整ったら、今度は Java を使った Android アプリの開発環境である「Android SDK」をインストールします。

⇒ http://developer.android.com/intl/ja/sdk/index.html

自分で展開する ZIP 版と、インストーラ付きの exe 版がありますが、どちらをダウンロードしても構いません。

ダウンロードが済んだら、ZIP 版の場合は展開し、展開された「android-sdk-windows」フォルダを適当なディレクトリへと配置して下さい。さらにフォルダ内にある「SDK Manager.exe」を実行すると、Android SDK のうち、必要なパッケージを選択&ダウンロードするためのマネージャーが表示されます(exe 版の場合には自動実行されます)。

■　**必要な物をダウンロード**



マネージャー左のメニューから「Available packages」を選択し、必要なパッケージにチェックを入れ、ダウンロードします。どれが必要かは、きちんと検証はしていないのですが、AFA で開発を行うのみであれば、最低限、

- Android Repository の Android SDK Platform-tools, revision 1
  (adb という実行ファイルが必要。また ddms という便利なツールも含まれる)
- Third party add-ons の Google Usb Driver package, revision 4
  (win の場合、デバイスを USB 経由でつないだ際のドライバが必要)

の 2 つはダウンロードしておいた方がいいですね(他にも必須項目あるかもしれません)。

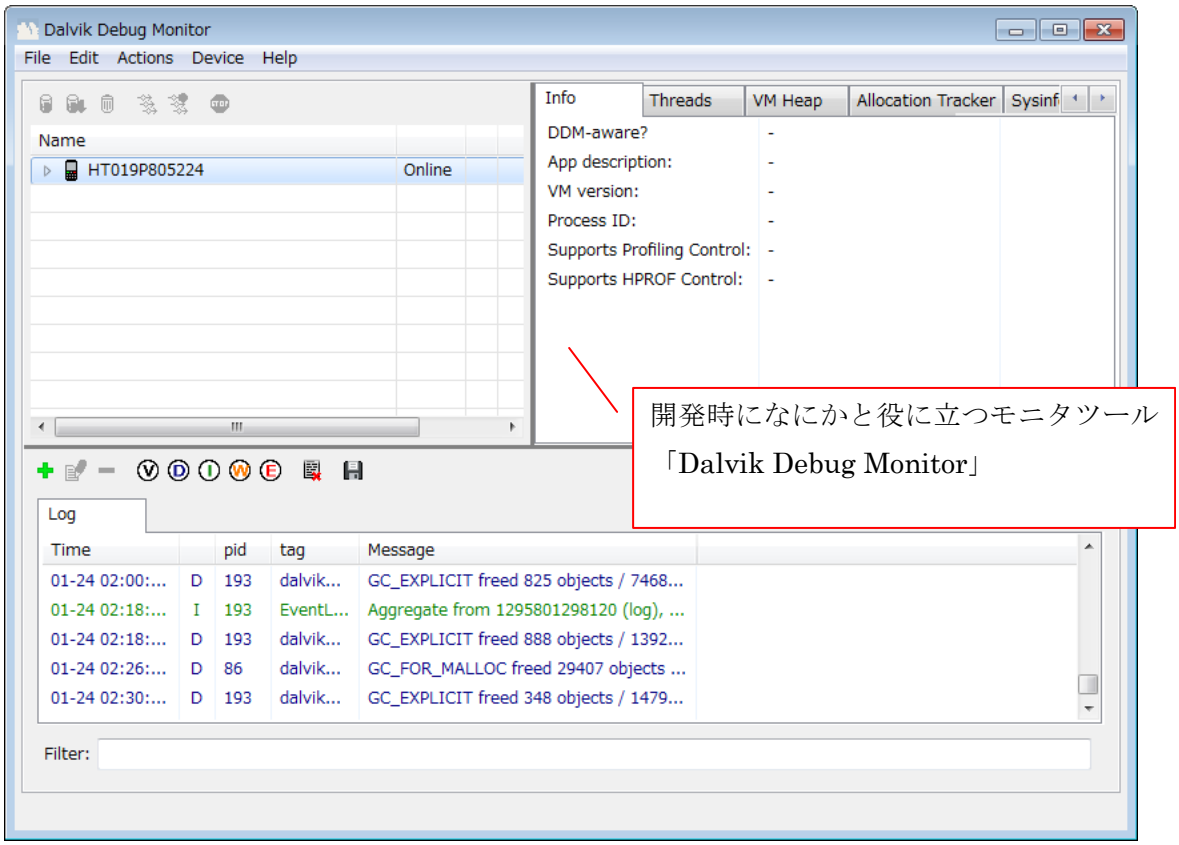
さて、必要なパッケージのダウンロードが済んだら、またもや環境変数「Path」の設定が待っています。おなじみの手順で[システム変数の編集]ダイアログを表示し、今度は Android SDK 内の「tools」フォルダと、「platform-tools」フォルダへのパスを追加します。例えば、C ドライブ直下の「Android」フォルダ内に「android-sdk-windows」を展開した場合には、既存の環境変数 Path に加えて、

```
;C:¥Android¥android-sdk-windows¥tools;C:¥Android¥android-sdk-windows¥platform-tools
```

というような形で追加するわけですね。

必要なパッケージをインストールできたか、パスがきちんと通ったか、を確認するには、コマンドプロンプトで「adb version」と入力してみましょう。「Android Debug Bridge version 1.0.26」のようにバージョン番号が表示されたら成功です。
さらにコマンドプロンプトで「ddms」と入力してみましょう。「Dalvik Debug Monitor」が立ち上がればインストールが成功しています。

■ 「ddms」で「Dalvick Debug Monitor」が立ち上がる



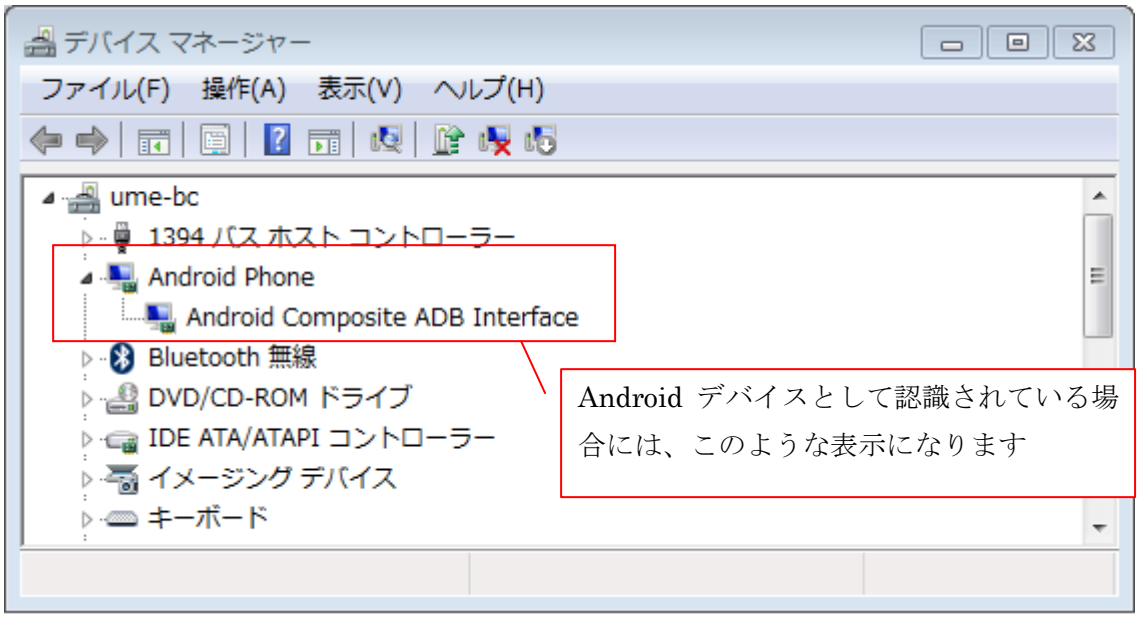
実機を USB 経由で繋いで Android デバイスとして認識されるかをチェック

さて、２つの環境がインストールできました。ここで実機がある場合には、USB ケーブルで実機と PC をつないでみましょう（逆に言うと、この時点まで実機はつながない方が無難です)。
そして、コマンドプロンプトから「adb devices」と入力します。この時点で、端末の番号が表示されれば成功です。USB 経由で実機へと作成したアプリをインストールする事ができます。

■　**デバイス接続の確認**

また、この時、コマンド自体は受け付けられるものの、端末番号が出てこない場合には、対応するドライバがまだ用意できていないか、端末が Android デバイスとして認識されておらず、単なる大容量ハードディスクとして認識されている可能性があります。
比較的新しい端末で開発を行う場合には、まだ Google 側でドライバの配布が追い付いていないケースがありますので、自前でドライバの設定が必要です。このケースでは、端末内に用意されているドライバを使用できるケースもありますが、ケースバイケースですので、web 等でその端末のドライバ設定の情報を検索してみるのが良いでしょう。
また、ハードディスクとして認識されてしまっている場合には、USB ケーブルを抜き差しすると認識しなおしてくれるケースもありますが、それでも認識されない場合には、[コントロールパネル]-[デバイスマネージャー]を使ってチェックしてみましょう。

■　**デバイスマネージャーでチェック**



大容量ハードディスクドライブとして認識されてしまっている場合には、[操作]-[ハードウェア変更のスキャン]を選択して、デバイスを再認識させましょう。すると、「Android Phone」という項目が表示されます(表示されない場合には、大容量ドライブとして認識されているデバイス項目を削除して再実行してみて下さい)。この時点で正しくドライバが認

識されないで、「！」マークが表示されている場合には、該当項目を選択し、ポップアップメニューより、「ドライバソフトウェアの更新」メニューを選択します。そして、手動でのドライバインストールを選択し、Android SDK ディレクトリ内の「google-usb_driver」フォルダを選択すると、対応するドライバが組み込まれます。

3. Android アプリを作成する前に考えておいた方が良い事

いろいろな環境が準備でき、「AS でアプリ作れるなら今までのノウハウ活かしていっちょやるかー」と、なるわけですが、その前に少し考えておいた方がいい項目が何点かあります。

解像度の変化に対する方針

Android の端末は、さまざまな解像度を持っています。そこで、解像度が変化した場合に、コンテンツの表示をどうするのか、という方針を決めておいた方が安全です。
同じ端末でも端末上部に表示される通知バー部分を表示したままアプリを起動するか、フルスクリーンで起動するかによって、アプリの使用する画面サイズは変化してきます。

■ 通知バー部分の表示・非表示

また、気を付けないといけない点は、端末によってタテヨコ比(アスペクト比)も異なる点です。web のコンテンツでいうところの、ブラウザのサイズ変更に対応してきっちりと各パーツを再配置する、リキッドレイアウトのような考え方が必要になってくるわけですね。
そういう意味では、今まで Flash で web コンテンツを作成されていた方にとっては、「ブラウザ上にムービーをどう表示するか」というノウハウが活かせますね。

■ 画面サイズに関する選択の例

方針	説明
おまかせ作戦	サイズ変更に特に対応しない。 AIR の自動サイズ調整に任せる。
アスペクト比合わせ作戦	アスペクト比に応じて画面内のパーツの位置を調整し、「はみだし」部分が無いようにする。 拡大/縮小に関しては、AIR の自動サイズ調整に任せる。
NO_SCALE 作戦	stage.scaleMode=StageScaleMode.NO_SCALE を適用し、自動サイズ調整による拡大/縮小を適用せず、自前で各パーツのサイズを算出・設定する。
mxml 作戦	(Flash Builder & mxml のレイアウトに任せる)

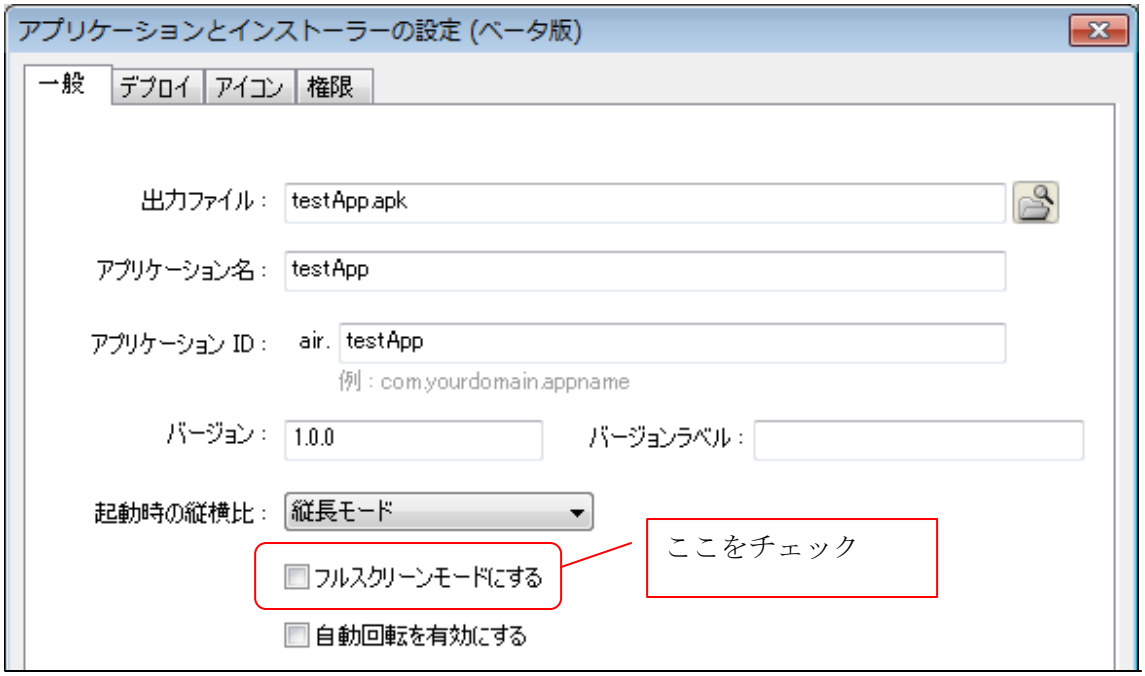
このあたりの処理には以下のようなプロパティやメソッドを使って端末の情報を取得し、対応することになるかと思います。

■　**異なる解像度に対応する際に役立つプロパティ、メソッド**

プロパティ/メソッド	用途
stage.scaleMode	コンテンツ全体の拡大/縮小表示モードを決める
stage.stageWidth stage.stageHeight	表示領域の幅・高さを取得
Capabilities.screenDPI Capabilities.screenResolutionX Capabilities.screenResolutionY	表示端末の解像度情報を取得
stage.align = StageAlign.LEFT;	コンテンツの表示の基準点を左上に設定

また、通知バーの表示/非表示の設定は、[AIR Android 設定]ダイアログの「一般」タブ内の「フルスクリーンモードにする」のチェックで切り替えることができます。

■　**通知バーの表示/非表示の設定**



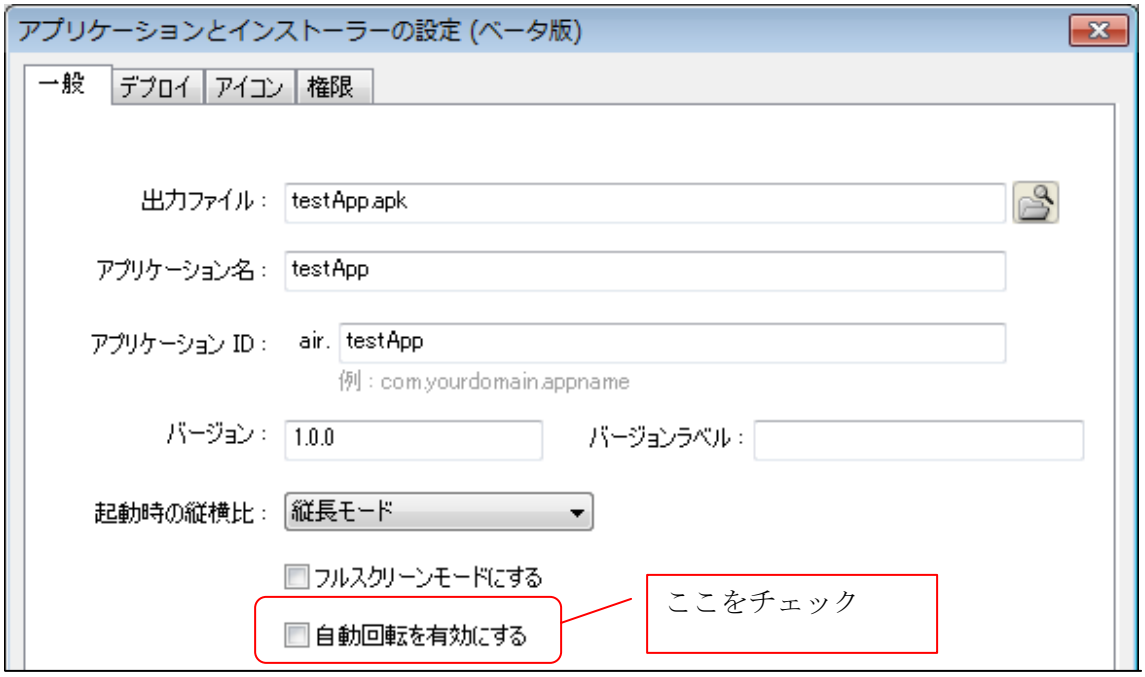
もし、コンテンツの伸縮やパーツの位置調整を行う方針で行く場合には、「resetControls メソッド」のような適当な再配置専用のメソッドをあらかじめ作成しておき、配置関係は集中管理して行うような仕組みを用意しておくのがお勧めです。

デバイスの回転に対する方針

デバイスが回転した際、コンテンツの表示をどうするのか、という方針もあらかじめ決めておいた方が良い物のひとつです。

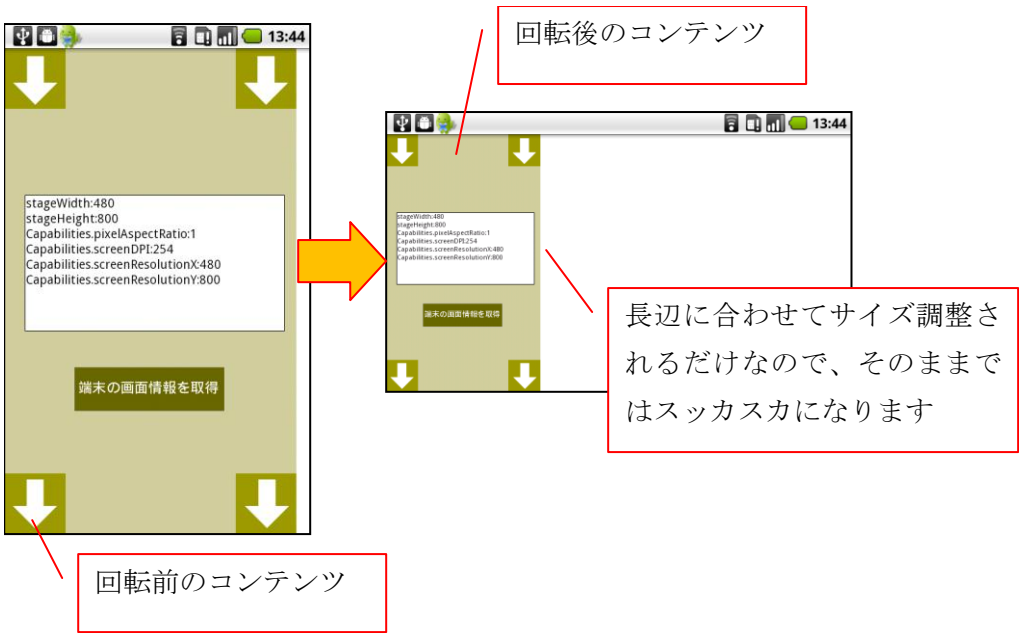
デバイスが回転した際、その向きに合わせてコンテンツも自動的に回転して表示するには、[AIR Android 設定]ダイアログの「一般」タブ内の「自動回転を有効にする」のチェックで切り替えることができます。

■　通知バーの表示/非表示の設定



自動回転を有効にした場合には、画面が回転した際にコンテンツの表示を調整する必要が出てきます。何もしないと、本当にそのまんま表示方向が変わるだけです。

■　とりあえず自動回転を有効にした所



通知バーを表示している場合には、通知バーの分だけその向きでのタテ方向の領域が占有される点にも注意です。タテ・ヨコの比が縦向き時と横向き時で変わってくるわけですね。
画面の向きが変更された場合に任意の処理を実行するためには、

```
stage.addEventListener(StageOrientationEvent.ORIENTATION_CHANGE,func);
function func(e:Event):void{
	switch(stage.deviceOrientation){
		case StageOrientation.DEFAULT:
			break;
		case StageOrientation.ROTATED_LEFT:
			break;
		case StageOrientation.ROTATED_RIGHT:
			break;
		case StageOrientation.UPSIDE_DOWN:
	}
}
```

と、orientationChange イベントで向きの変更を拾い、stage.deviceOrientation で変更後の向きを確認し、対応した処理を実行します。ここからコンテンツ上のパーツを再配置する関数を呼んであげてもいいですね。
また、この設定のオン/オフにかかわらず、デバイスは回転します。当たり前ですが、横むければ横向きますし、逆さにすれば逆さになります。そこで、回転に対応する別の手法として、enterFrame イベント等で定期的に stage.deviceOrientation の値をチェックし、変更があったら任意の処理を実行してもいいですね。
たとえば、メモ用アプリを作成し、写真を表示できる機能を作成したとします。このようなケースでは、デバイスの向きを変更した際に、コンテンツ全体の向きを変更する必要はありませんが、写真を横向きにした方が見やすいですよね。そこで、stage.deviceOrientation の値によって、コンテンツ的な向きはそのままで、表示している写真や一部のコントロールの向きだけを変更してしまう、といった対応方法を取る、といった事もできます。

■　**コンテンツの向きはそのままで、デバイスの向きの変化に対応した処理を作成**

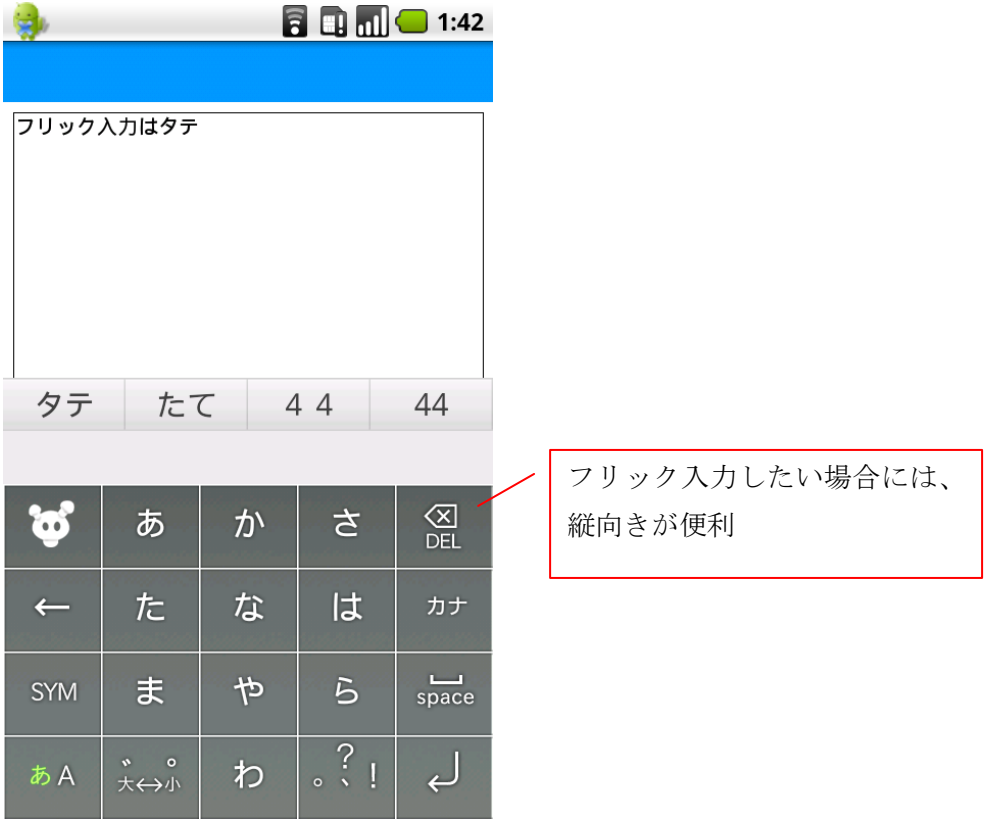
また、コンテンツの向きは、自動回転させるだけではなく、任意のタイミングでコードから変更することもできます。

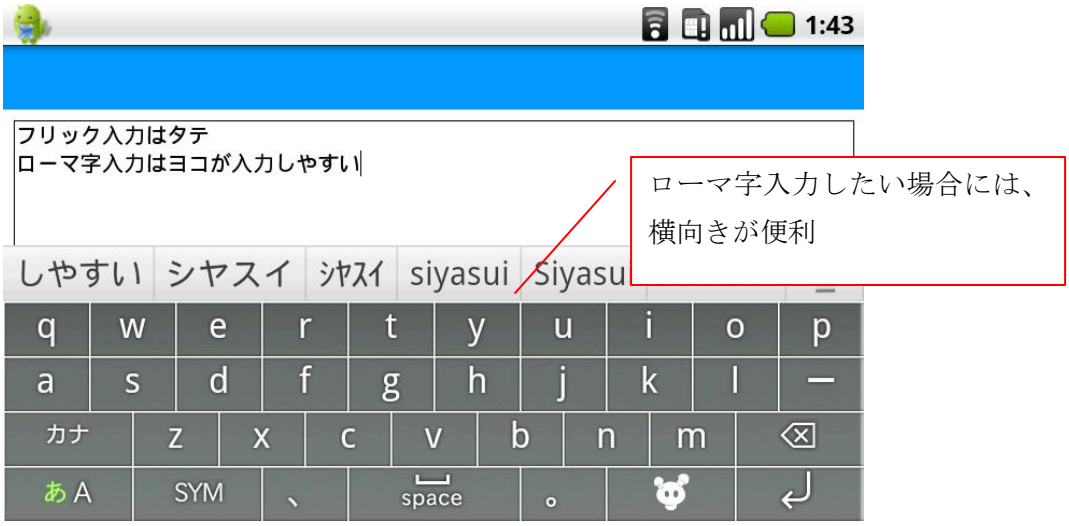
```
stage.setAspectRatio(StageAspectRatio.LANDSCAPE);　　//横向きに
// stage.setAspectRatio(StageAspectRatio.PORTRAIT);　　//縦向きに
```

のように「stage.setAspectRatio メソッド」の引数に、「StageAspectRatio クラス」の 2 つの定数のどちらかを指定して実行します。つまり、コードから任意の向きに変化させる場合には、「タテかヨコか」のどちらかにしか変更できません。

これまたピンポイントで端末の向きを変更したい時に便利ですね。特にスマートフォンでは、端末の向きが重要な意味を持つ場合があります。例えば、ソフトウェアキーボードです。

■　**端末の向きによってソフトウェアキーボードの方式が変化することも**





画像は「Simeji」という Android 界隈では超メジャーな日本語 IME を使用している場合のテキスト入力画面です。縦向きと横向きの場合のソフトウェアキーボードの状態に注目してください。縦向き時はフリック、横向き時はローマ字入力となっていますね。
テキスト入力の方式は、ユーザーさんによって得手不得手がありますので、長いメモを取るようなアプリを作成する場合、ピンポイントで setAspectRatio メソッドを使用すれば、コードから端末の向きを変更し、ユーザーさんの得意な方式のキーボードを表示できるようにする、等の処理が作成できますね。

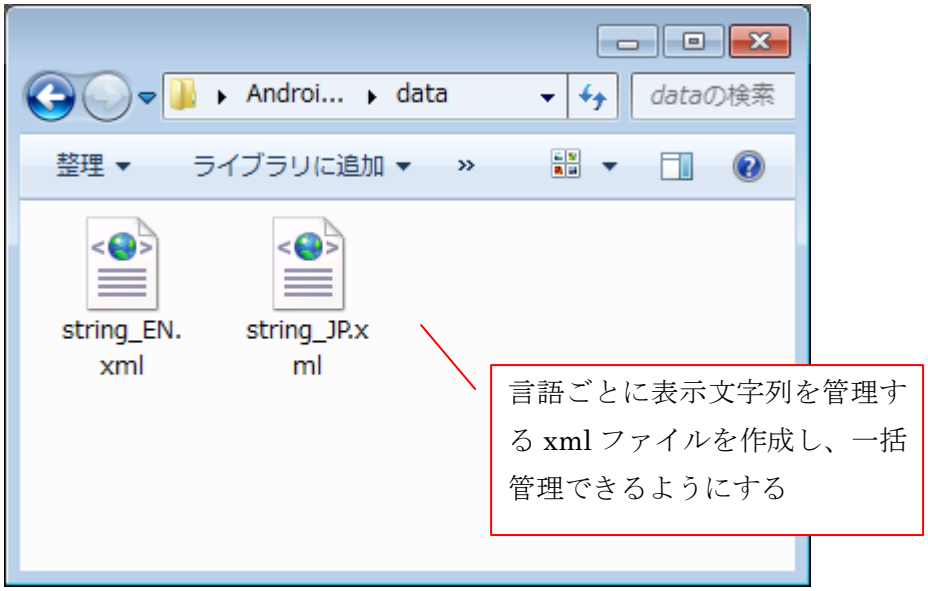
ちなみに、実はこのあたりの「コードからの向きの変更」処理、少し前までは

「stage.setOrientation メソッド」で実行できました。が、なんやかんやあって、今は setAspectRatio メソッドで向きを変更するようになっています。ヘルプ等も含め、少し前の資料には stage.setOrientation メソッド方式の手順を紹介しているものも存在していますのでご注意を。

多言語対応に関する方針

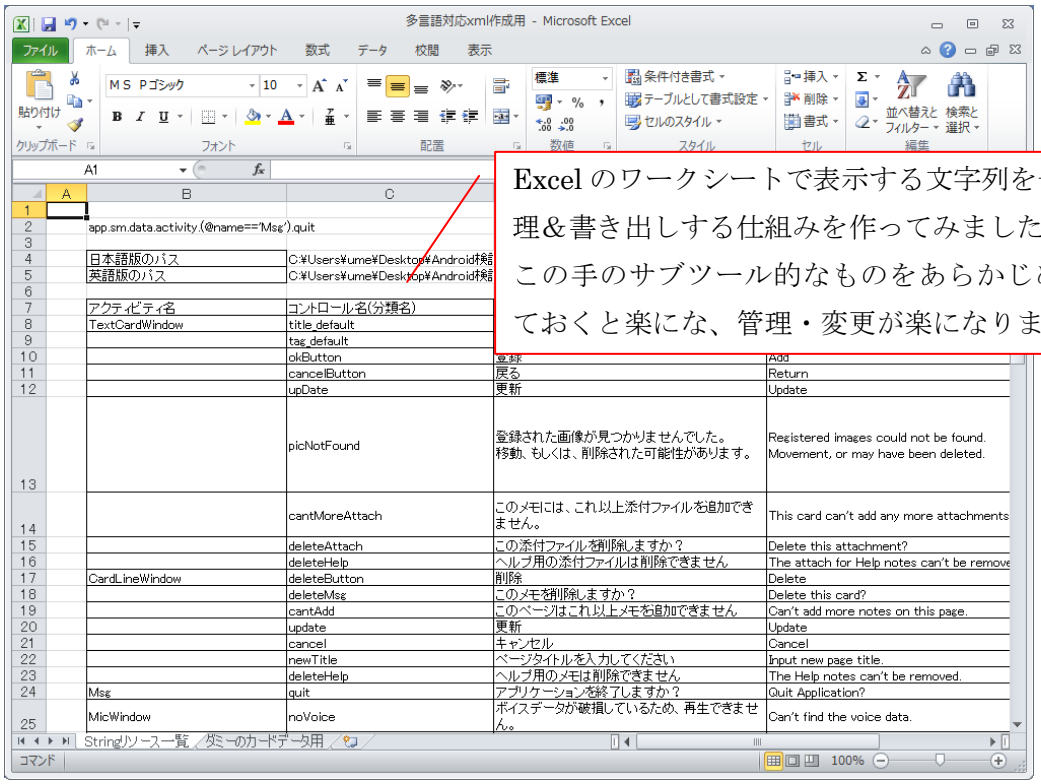
Java で Android アプリを作成する場合、多言語対応に対する対策として、あらかじめ言語ごとに res/values フォルダ内にサブディレクトリを作成し、strings.xml を用意する、といった方法が用意されています。AFA では、この方式は(現状)使用できません。そのため、多言語対応を考えている場合にはあらかじめなんらかの対策を立ててからアプリ作成を始めた方が安全です。
いろいろな方法があるかとは思いますが、筆者の場合には、英語と日本語のテキスト表記を管理する xml ファイルを作成し、アプリに同梱する手法を取ってみました。さらに表示文字列を集中管理するクラスを一つ作成し、そこにユーザーが選択した設定の文字列情報をまとめて読み込んで管理するようにしてみました。そして、アプリに表示する文字列は、必ずこのクラス経由で取得する、というルールを作成し、運用するようにしました。

■ xml ファイルで言語単位で表示文字列を管理



ついでに表示文字列の管理と xml ファイルの作成を一括でできるように、Excel のワークシートで表示文字列を管理し、マクロで各 xml ファイルを作成できるようにしておきました。このあたりは、自分の手慣れたアプリをサブツール的に使用がお勧めです(いっそ AIR で専用サブアプリを作成してしまってもいいですね)。

■　**表示文字列を Excel で一括管理してみました**



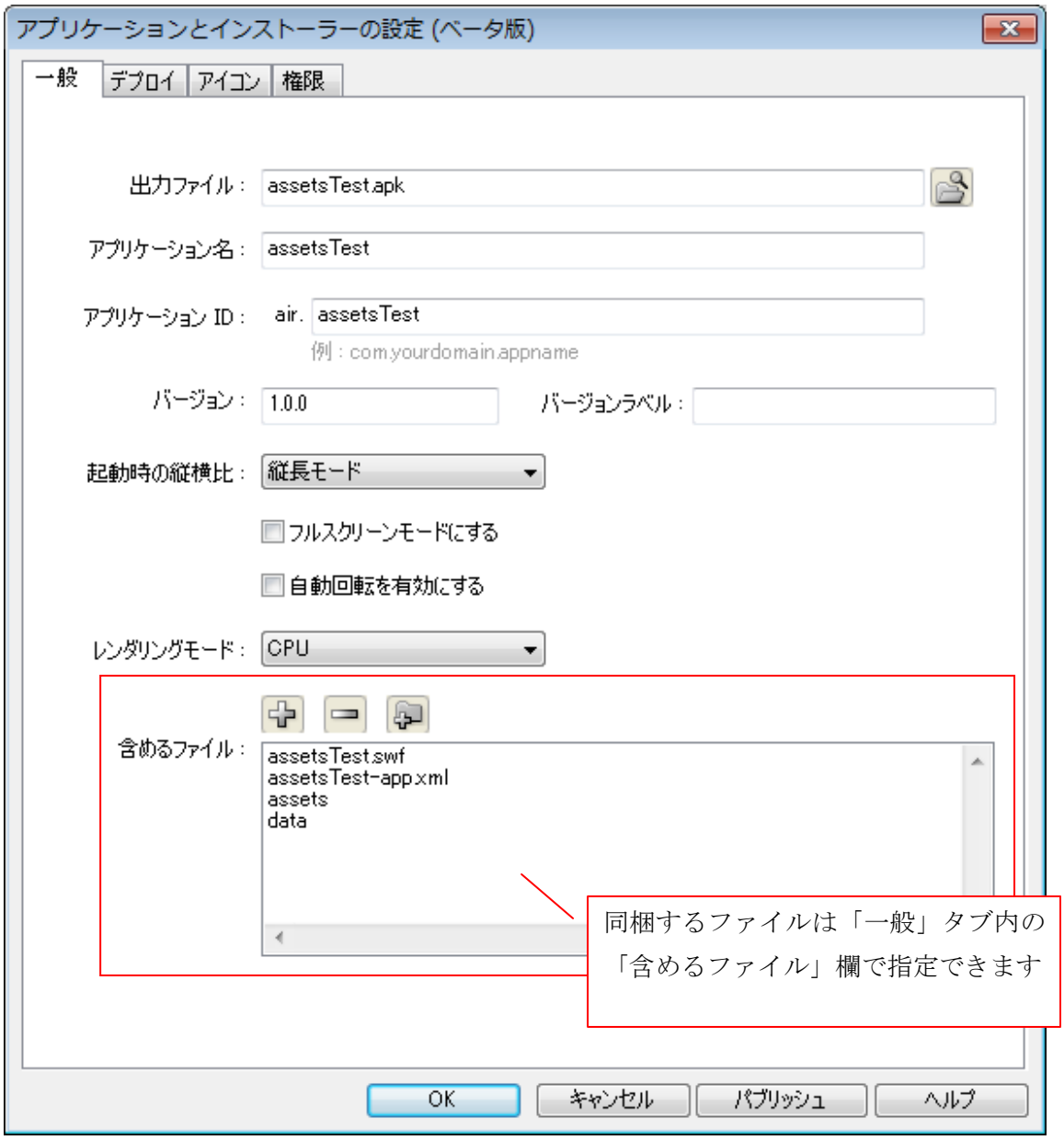
	A	B	C	D	E
1					
2		app.sm.data.activity.(@name=='Msg').quit			
3					
4		日本語版のパス	C:¥Users¥ume¥Desktop¥Android検		
5		英語版のパス	C:¥Users¥ume¥Desktop¥Android検		
6					
7		アクティビティ名	コントロール名(分類名)		
8		TextCardWindow	title_default		
9			tag_default		
10			okButton	登録	Add
11			cancelButton	戻る	Return
12			upDate	更新	Update
13			picNotFound	登録された画像が見つかりませんでした。 移動、もしくは、削除された可能性があります。	Registered images could not be found. Movement, or may have been deleted.
14			cantMoreAttach	このメモには、これ以上添付ファイルを追加できません。	This card can't add any more attachments
15			deleteAttach	この添付ファイルを削除しますか？	Delete this attachment?
16			deleteHelp	ヘルプ用の添付ファイルは削除できません	The attach for Help notes can't be remove
17		CardLineWindow	deleteButton	削除	Delete
18			deleteMsg	このメモを削除しますか？	Delete this card?
19			cantAdd	このページはこれ以上メモを追加できません	Can't add more notes on this page.
20			update	更新	Update
21			cancel	キャンセル	Cancel
22			newTitle	ページタイトルを入力してください	Input new page title.
23			deleteHelp	ヘルプ用のメモは削除できません	The Help notes can't be removed.
24		Msg	quit	アプリケーションを終了しますか？	Quit Application?
25		MicWindow	noVoice	ボイスデータが破損しているため、再生できません。	Can't find the voice data.

多言語対応を考える場合には、アプリ制作前にどのような方針で行くのかをちょいと考えてから進めてみましょう。後でステージ上やライブラリ内のテキストを必死になって変えたりするのは、超めんどくさいです(やりました)。

どんなファイルを同梱するかに関する方針とアクセス方法

AFA で作成するアプリでは、設定用の xml ファイルや、表示したいムービーや画像の swf ファイル、png ファイルなどの各種ファイルを同梱してパッケージングする事ができます。画像などは､開発中に差し替えることを考慮して､png ファイルを適宜読み込んで使用する、なんて運用をする事が多いかと思いますが、そのまんまパッケージングできるわけですね。同梱するファイルを決めるには、Flash の[AIR Android 設定]ダイアログボックス内の「一般」タブ内の「含めるファイル」欄で設定を行います。この際、ファイル単位だけでなく、フォルダ単位でも同梱するファイルを指定できます。

■　**ファイルを同梱する際の設定**



つまりは、すべての素材を Flash ドキュメント内で管理しなくても OK なわけですね。
さて、このようにして同梱したファイルにコードからアクセスする場合、ファイルへのパスの指定はどうすればよいのでしょうか。おなじみの Loader&URLRequest で同梱した画像ファイル等を読み込む場合には、そのまま相対パスでアクセス可能です。

```
//「piyo.png」という画像ファイルを同梱した場合
var req:URLRequest = new URLRequest("piyo.png");
var loader:Loader = new Loader();
loader.load(req);
addChild(loader);
```

フォルダごと同梱した場合には、フォルダ名から指定すれば OK です。

```
//「data」フォルダ内に「piyo.png」という画像ファイルを同梱した場合
```

```
var req:URLRequest = new URLRequest("data/piyo.png");
var loader:Loader = new Loader();
loader.load(req);
addChild(loader);
```

AIR アプリでおなじみの File を使ってアクセスしたい場合には、いろいろなアプローチがあるのですが、とりあえず「app:/ファイルパス」のような形式で同梱したファイルへとアクセス可能です。

```
// piyo.png」という画像ファイルを同梱した場合
var f:File = new File("app:/piyo.png");
//ファルを開き、バイト配列へと内容を取得
var s:FileStream = new FileStream();
s.open(f,FileMode.READ);
var data:ByteArray = new ByteArray();
s.readBytes(data);
//取得した値を利用。今回は画像を Loader に表示。
var loader:Loader = new Loader()
loader.loadBytes(data);
addChild(loader);
```

バイナリとして同梱ファイルを扱いたい場合には File を利用すれば OK ですね。
ちなみに、File を利用していろいろな場所からファイルの読み書きを行う場合、次の 3 種類のパスの指定方法を覚えておくと便利です。

■ File のパス指定あれこれ

指定方法	パス
File.applicationDirectory または、 app:/	同梱ファイルのルートです。 読み取り専用ディレクトリです。
File.applicationStorageDirectory または、 app-storage:/	いわゆる「アプリケーション専用ディレクトリ」です。 ここは読み書き可能です。アプリの設定などを保存する場合にはここで OK ですね。
File.desktopDirectory File.documentsDirectory	SD カードのルートです。 SD カード上にフォルダを作成したり、ファイルの読み書きをしたい場合にはこちらを基本にパスを作成していきます。

この仕組みを活用する場合には、何を読み込むのか、また、どんなフォルダ構成で管理するのかをあらかじめ決めておいた方が良いですね。

4.　開発・デバッグの際に知っておくと便利な仕組み

AFA でアプリを開発する場合に便利だった仕組みを 2 点ご紹介したいと思います。ひとつは実機でテストする際に、おなじみの trace 関数やエラーメッセージ等を確認する方法、もうひとつは、条件付きコンパイルを利用したデバッグの方法です。

DDMS を使って trace 関数やエラーメッセージを確認する

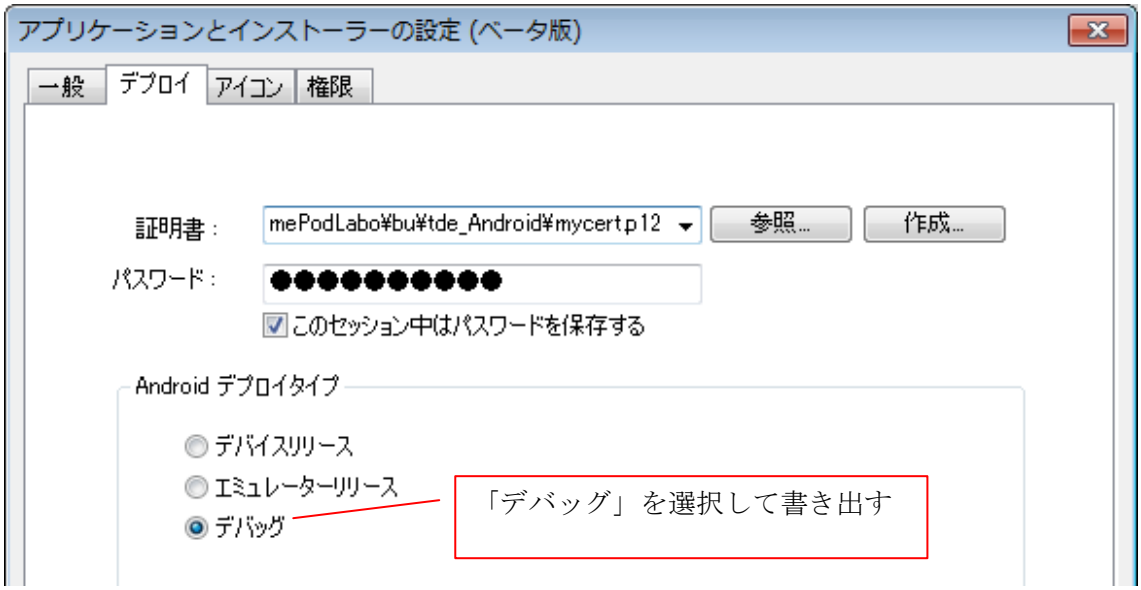
AFA でアプリを作成し、書き出す際には 3 パターンの書き出し方法が用意されています。

■　3 パターンの書き出し方法

タイプ	説明
デバイスリリース	いわゆる本番用 実際に実機にインストールしたり、配布する際に使用
エミュレーターリリース	PC 上で動作する Android エミュレータにインストールする際の方法。PC 上でテストしたいときに使用
デバッグリリース	trace 関数の出力や、エラーメッセージの確認等をしたい際に利用できる方法

このうち「デバッグリリース」で書き出した場合には、tracce 関数やエラーメッセージ等、Flash の「ムービープレビュー」で表示した際に[出力]ウィンドウに表示されるおなじみの情報を、Android のログへと出力してくれます。

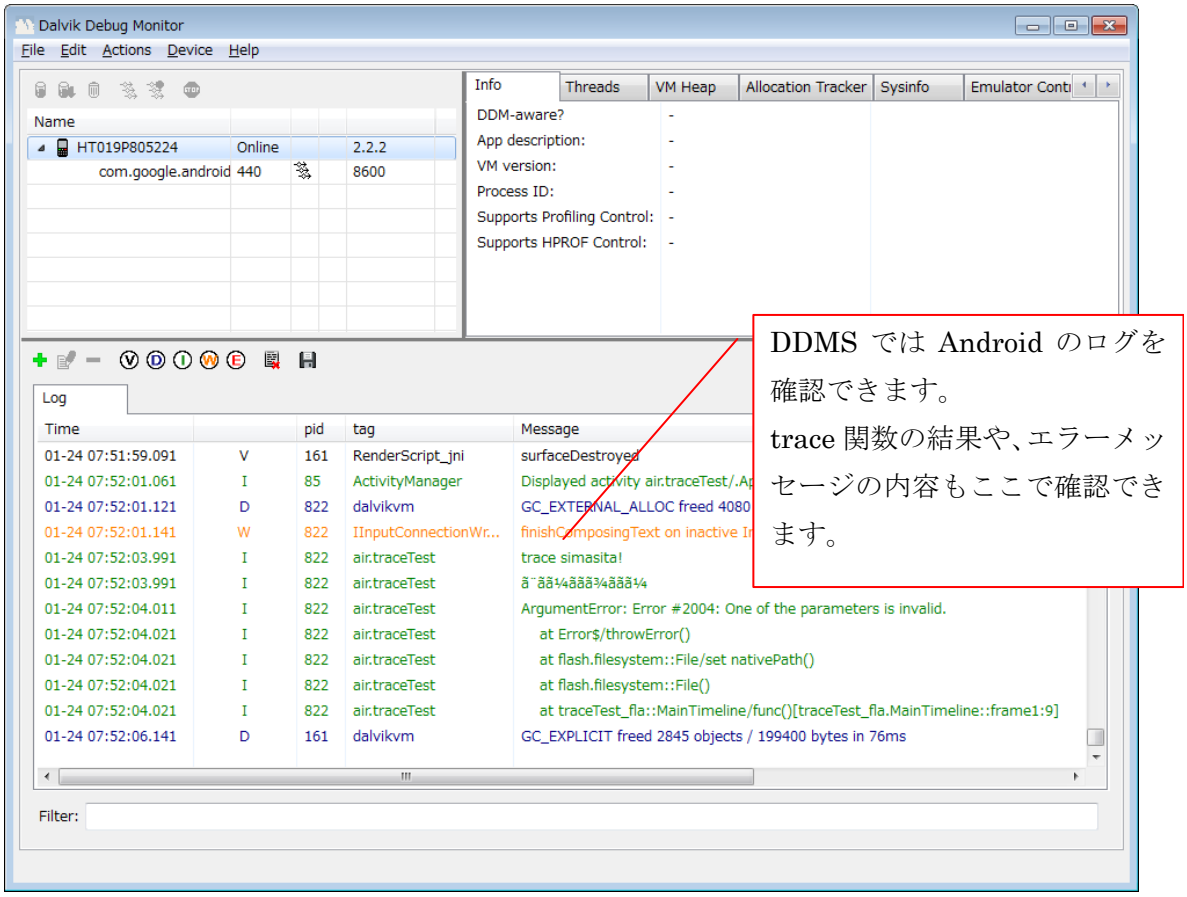
■　デバッグモードで書き出す



そして、この Android に書き出されたログを確認する際に便利なツールが、「DDMS(Dalvik Debug Monitor)」です。DDMS は Android でアプリを作成する際に、様々な用途に使えるモニタツールなのですが、そのうちのひとつに、端末から出力されるログを、リアルタイムで確認できるという機能があります。

つまりは、デバッグモードでアプリを書き出し、実機にインストールし、DDMS を立ち上げておけば、操作に応じて配置した trace 関数の出力を確認したり、エラーの発生やその原因を確認できるというわけですね。

■ Dalvik に表示された trace 関数のメッセージやエラーメッセージ



日本語のメッセージは文字化けしてしまうのですが、アルファベットなら OK です。お手軽に[出力]ウィンドウ代わりに使える便利なツールとして活用してみましょう。

条件付きコンパイルを利用してマルチプラットフォーム対応やデバッグ対応を切り替える

Flash には「条件付きコンパイル」機能が搭載されています。詳しくはアドビの上条さんの blog をご覧ください。

http://cuaoar.jp/2010/10/actionscript-30.html

http://cuaoar.jp/2010/10/actionscript-30-1.html

この仕組みを利用して、iOS 用のコード、Android 用のコード、さらにはデバッグ用のコードなどを切り分けて記述しておき、用途によってコンパイルする箇所を切り替えながら管理すると、同じ Flash ドキュメントやクラスファイル内での、マルチプラットフォーム対応や、デバッグ時のみに走らせたいコードなどを整理整頓することができます。

いちいちコメントアウトしたり元に戻したりしながら確かめるよりも楽ですね！

5.　コンポーネントどうするか問題

AFA を使ってアプリを作成する場合の大きな問題点として、「コンポーネントをどうしよう」という問題点があります。
現状では、AFA では Android アプリの標準的なボタンやメニュー、スクロールするリストといったコンポーネントは提供されていません。Flash Builder の次期プレビュー版である burrito には用意されているのですが、Flash の場合には自前で用意する必要があります。
このコンポーネントに関してですが、ヒム・カンパニーの永井さんが、ご自身のブログでひとつの解決方法をご提示して下さっています。

■AIR Android アプリ作成：Flash, Flex Hero
http://himco.jp/air-for-android/
■2-6-2:Menu コンポーネント
http://himco.jp/air-for-android/?p=286
■翻訳記事 No:106 「ケビン ホイトのコンポーネント」
http://www.himco.jp/articles/pdf/kevinHoytComponents.pdf

上記の翻訳記事の内容を拝見しますと、

本ドキュメントは、Adobe のプラットフォームエバンジェリストであるケビン ホイト（Kevin Hoyt）のブログで公開されている記事「Some Flash Android Components」のコンポーネントの使い方をヒム・カンパニー 永井勝則が自主的に記述したものです。

との記述がされています。日本語で読める AFA 対応のコンポーネントの使い方の記事があるというのは本当にありがたいですね。多謝です。

また、永井さんの blog には、AFA 開発のノウハウやサンプルコードが満載です。びっくりするくらい充実しています。AFA で開発をしようと考えている方は必見です。

6. 現状、AFA ではできないこと

「そういえば AFA であれってできないのかな？」という項目を調べてみて、現状ではどうやらできなさそうな項目をまとめてみました。

■ 現状、AFA ではできないこと

項目	説明
インテントの発行	AFA ではインテントの発行はできません。 URI スキーマは、StageWebView 上では利用できます。 ただし、navigateURL 等では、http:、tel:、sms:で、ブラウザ、電話、ショートメッセージ(メールのような物？日本ではあまり馴染み無い)の各アプリが呼び出せるようです。
インカメラへのアクセス	いわゆる「自撮り」用のカメラへのアクセスはできません。 自撮り用のカメラへのアクセスというか、「２台目のカメラ」へのアクセスができない状態なのかもしれません。
バッテリー残量等へのアクセス	ガラケーの時のコンテンツではおなじみのバッテリー容量情報ですが、AFA ではアクセスできません。
ウィジェットの開発	AFA ではいわゆるウィジェットは加発できません。 フルスクリーンを利用するアプリのみが開発できます。

この他にもまだいろいろとあるかもしれません。
ただ、AFA はスタートしたばかりの技術ですので、今後のアップデートで可能になる物も出てくるかと思います。今後の動向に注目ですね。

7. 描画速度に関して

AFA での開発を進めるにあたって、おそらくみなさんが一番悩むのは描画速度に関する問題になるかと思います。やはりまだまだ PC に比べると、そんなに速度は出ません。
そこで、様々な手法で描画速度を上げるチューニングを行うわけですが、正直言って、この項目に関してはまだ手探り状態です。そこで、有効だと思われる方法を列記するに留め置きたいと思います。

- 表示リストの改装を深くしない。理想は 1~2 階層
- GPU レンダリングを効果的に利用する
- cacheAsBitmap、cacheAsBitmapMatrix を効果的に利用する
- 画面に表示するインスタンスを事前に生成できる場合には、事前に用意しておく(生成と表示のタイミングを切り分ける)
- いわゆる画像プールの仕組みなどを使用し、なるべく BitmapData を使いまわす
- カクついてしまう画面遷移の演出などは避けたり代替のものを利用する
- 画面全体を一気に動かすのではなく、細かくパーツごとの位置を移動する

などなどです。
このあたりはいろいろと実験してみて、有効な方法を模索し続けたいと考えております(azoth 系のサードパーティ製のコンパイラでビットマップの描画部分だけ最適化して手動パッケージングしてみたらどうかなあ…とか。今度試してみます)。

さてさて、本流とはちょっと外れた話題をとりとめもなくいろいろと書き連ねてきましたが、AIR for Android、単純に楽しいですよ！新しいデバイスでのコンテンツ作りというのはやはり楽しいですね。また、身近に持ち歩く端末上で、自分自身が作成したアプリを動作させることができるというのは、本当に便利です。かゆい所に手が届くものも作れそうですね。

今後もいろいろと研究し、コミュニティの皆様方にとって面白い情報を提供できるように頑張りたいと思います。

ではでは。

よしおか