新仮想マシン搭載機種に対応するｉアプリを作成する際の留意点

第 1.00 版

平成 22 年 5 月 21 日
株式会社ＮＴＴドコモ

目　次

1 はじめに

2010 年春以降に発売される i アプリ対応携帯電話の一部機種では、搭載される i アプリ実行環境の内部仕様（アプリケーションプログラムを実行する仮想マシンの詳細挙動を含む）が一部変更となります。これに伴い、アプリケーションプログラム実行時の挙動に影響が生じる場合があります。

本書では、上記仕様変更に伴い留意すべき点について解説します。

なお、本書では 2010 年春以降の機種に搭載される新しい i アプリ実行環境の仮想マシンを「新仮想マシン」、従来の機種に搭載されている i アプリ実行環境の仮想マシンを「旧仮想マシン」と表記することとします。

2 スレッド制御に関する留意点

マルチスレッドシステムにおけるスレッド切り替え制御の詳細は、仮想マシンの実装によって異なります。新仮想マシンと旧仮想マシンでもスレッド切り替えの詳細挙動は異なっており、旧仮想マシン上で問題なく動作していたアプリケーションプログラムが、新仮想マシン上ではスレッド切り替え制御の違いにより、意図通りの動作をしない（※）可能性があります。

※タイミングの調整に空ループを用いるなど、システムによる暗黙のスレッド切り替えを期待したアプリケーションプログラムでは、スレッド間のタイミングが合わずに動作不良を引き起こしたり、アプリケーションプログラマが期待するようなタイミングや頻度でスレッド切り替えが発生せず実行性能低下やフリーズが発生したような見え方をするケースもあります。

この問題を避けるため、アプリケーションプログラムでは以下のガイドラインに沿うようにしてください。

(1) スレッド切り替えタイミングに依存したコードの排除

アプリケーションプログラムでは、スレッド切り替えタイミングに依存したコードを記述しないようにしてください。

例：

```
run() {  //Main ループ
    do() {
        ....
        if(flag) continue;
        ....
        //演算等(この期間で paint()が呼ばれると描画が乱れると仮定)
        ....
        if(x) {  //フレーム処理回数が閾値を下回ると repaint()を呼ぶ
            repaint();
            Thread.yield();
        }
    }while(true);
}
paint(Graphics g) {
    flag = true;
    //描画等
    flag = false;
}
```

上記の i アプリは、「演算等」の期間中に `paint()`が呼ばれると描画が乱れるものとします。`flag` によって問題が発生しないよう制御されているように見えますが、`paint()`の呼び出しは i アプリ実行環境が自発的に行うことがあり、それはいつ行われるか不定であるため上記ロジックでは問題が発生する可能性があります。この例の場合、`paint()`処理と「演算等」とを明示的に排他することにより問題が解消します。

(2) ビジーループの排除

新仮想マシンは旧マシンと比較して、ビジーループ（タイミングあわせなどのためにアプリケーションプログラム側で空ループを回すこと）を処理しているスレッドがあると他のスレッドにスレッド制御が切り替わりにくい傾向があります。そのため、アプリケーションプログラム側でビジーループを使用していると、アプリケーションプログラマが期待するようなタイミングや頻度でスレッド切り替えが発生せず実行性能低下やフリーズが発生したような見え方をするケースがあります。

この問題が発生した場合には、ビジーループの処理中に `Thread.yield()`や `Thread.sleep()`を入れることにより旧仮想マシンの動作に近づくことがあります。

3 ガベージコレクタ(GC)に関する留意点

新仮想マシンでは GC 機構として Generational GC（※）を搭載しており、自発的に GC が行われるため、アプリケーションプログラムからの `System.gc()`呼び出しによる GC 起動を原則抑止しています。特定のアプリケーションではメモリ利用状況によっては GC 処理に時間がかかり、アプリケーション実行中に一時的な動作停止を引き起こす場合があるなど好ましくない影響を与えることもあります。

※通常のアプリケーションではオブジェクトの生存期間に偏りがあり、一時的に生成される String や種々のバッファのように短期間で不要となるオブジェクト(短命なオブジェクト)とアプリケーション実装クラスのインスタンスや固定の Image のようにアプリケーション実行中のほとんどの期間存在するオブジェクト(長命なオブジェクト)が存在します。Generational GC では前者を young 領域、後者を old 領域と呼ばれるそれぞれ別の領域に保持し、異なる管理を行っています。

Generational GC では young 領域の GC は速い反面､old 領域の GC には時間を要します。特に、描画処理と並行して大きなバッファやオブジェクトの生成　例えば HTTP 通信、ScratchPad アクセス、外部メモリアクセス、メディアデータ処理などを繰り返すようなアプリでは、old 領域で発生する GC 処理に時間を要し、間欠的に描画がスムーズに行われなくなるように見える場合があります。
このような問題が発生する場合には、以下に記載するような対応により速度が改善する可能性があります。

(1) メモリを再利用する

例えば、同じサイズのメモリの確保/解放を繰り返すような処理の場合には、一度確保したメモリを解放せず、以降の繰り返し処理で再利用することを検討してください。

メモリ確保の回数を減らすことができれば、GC の発生頻度を低くすることができます。

(2) 不要になったバッファ/オブジェクトはできるだけ早い時点で参照を断つ

不必要になり再利用の可能性もないバッファ/オブジェクトについては、参照はできるだけ早い時点で断つ（当該オブジェクトを保持している変数に null を代入する）ようにしてください。これにより、GC 発生時に該当のメモリが即空きメモリとして扱われるようになり、メモリフラグメントの広がりを抑止する効果があります。

なおｉアプリ API の中には、アプリケーションプログラムが特定の指針に従うことでその API が利用しているバッファなどを上記方針に従って解放させることができ、メモリフラグメントの抑止につなげることができるものがあります。

- `MediaImage`

  `MediaManger.getImage()`に渡したデータは、`MediaImage.use()`でｉアプリ実行環境内に保持されるので、`use()`から戻った後、そのデータ（バイト配列や `InputStream` など）が不要であれば参照を断ってください。

- `JarInflater`

  `JarInflater` のコンストラクタは、引数で渡したバイト配列や `InputStream` のデータをコンストラクタ処理の中でｉアプリ実行環境内に読み込むので、コンストラクタから戻った後、そのデータ（バイト配列や `InputStream` など）が不要であれば参照を断ってください。また、`JarInflater` では `close()`を呼び出すことでｉアプリ実行環境内に保持したデータを破棄するので、`JarInflater` オブジェクトについては使用が終わったらすぐに `close()`を呼び出してください。

- 外部メモリ API

  外部メモリ API の利用では、`FileEntity.close()`を呼び出したらアプリケーションプログラムで保持している `FileEntity` の参照をすぐに断ってください。外部メモリのリード・ライトに使用していたｉアプリ実行環境内のバッファは、`FileEntity.close()`呼び出し時ではなく、このオブジェクトが消滅するタイミングで解放されます。

(3) 確保するバッファは、様々なケースを想定した大きなサイズではなく実際に必要なサイズとする

アプリケーションプログラム内で処理を共通化している等により、多くのケースで必要とされるサイズ以上のバッファを確保するようになっている処理があれば、必要サイズを確保して、例外的なケースのみバッファを拡張するようにすることで、GC の発生頻度を削減できる可能性があります。

一例として、数 100 バイトのバイト配列データを生成するケースがほとんどであるにもかかわらず、以下のように数 10KByte の `ByteArrayOutputStream` オブジェクトを生成して処理している場合、

```
ByteArrayOutputStream bo new ByteArrayOutputStream(32768);
bo.write(...);    // 数 100 バイト書き込み。
byte[] data = bo.toByteArray();
```

以下のように、引数なし、もしくは実際に使用される場合が多い数 100 バイトのデータサイズで、`ByteArrayOutputStream` オブジェクトを生成することで問題が改善する可能性があります。

```
ByteArrayOutputStream bo new ByteArrayOutputStream(512);
bo.write(...);    // 数 100 バイト書き込み。
byte[] data = bo.toByteArray();
```

4 System.gc()の効果と使い方

3 項の冒頭に記載したように、新仮想マシンではアプリケーションプログラムからの `System.gc()`呼び出しによる GC 起動を原則抑止しています。ただし、ADF に VmOption キーの指定（値 gc:on を指定）を行うことで、`System.gc()`呼び出しによって GC を起動させることができるようになります。

この機能を用いることで、`System.gc()`によって sweep を行わない場合に自発 GC が行われるような、メモリを繰り返し大量に消費・解放するようなｉアプリの動作を改善できる場合があります。

ADF に VmOption=gc:on が指定されているアプリでは、`System.gc()`を呼ぶと young 領域/old 領域に対して mark&sweep が実行されます（ただしコンパクションは行われません）。mark と

は root からの参照があるオブジェクトに印をつける処理で、その後の sweep において参照の断たれたオブジェクトが回収されます。

mark&sweep が実行されただけの状態ではヒープがフラグメンテーションを起こしている可能性も十分考えられますが、VmOption の指定による `System.gc()`の処理では性能を優先するため、このタイミングではコンパクションによるヒープのフラグメント解消は行われません。しかしその状態でも、その後生成されるオブジェクトがフラグメント内で確保可能なサイズの場合にはコンパクションを行うことなく継続動作が可能なため、結果的にコンパクションの実行機会の低減につながります。

GC の処理中はアプリケーションプログラムの実行は停止しますが、Generational GC において最も時間がかかるのが old 領域のコンパクションです。この old 領域のコンパクションの実行によるアプリの停止がユーザからもっとも違和感を持たれやすいものであるため、この発生頻度をできる限り下げることが重要となります。

old 領域のコンパクションの頻度を下げる（ヒープ使用量の削減を行う）ための一般的な方法は、ｉアプリコンテンツ開発ガイド　詳細編の 3.2 項も併せて参考にしてください。

`System.gc()`による GC の実行には時間がかかるため、`System.gc()`を呼ぶかどうかはそのメリット・デメリットを考慮して決める必要があります。なお、GC 実行によるアプリケーションプログラム実行の停止が不自然にならないようなタイミング（例えば画面表示の切り替えのタイミングなど）が i アプリ内にあれば、そのようなタイミングで `System.gc()`を呼ぶことで、仮想マシンが自発的に行う GC のタイミングを後に延ばすことができます。

5 その他の留意事項

本項では今までに解説した項目に属さない、個別の留意点について解説します。

(1) スレッドのメモリ消費

アプリケーションプログラムがスレッドを生成する際、旧仮想マシンと比較して新仮想マシンの方が多くメモリを消費します。またスレッドの生成/破棄自体には時間がかかるため、無駄なスレッド生成は避けるようにした方がメモリ効率および速度面で有利となります。

(2) 仮想マシンサスペンド時のオーバーヘッド

新仮想マシンは旧仮想マシンと比較してサスペンドの処理が遅くなっています。 i アプリ実行環境からのサスペンド応答待ち中に i アプリから呼び出される API で例外が発生しやすくなっているため、 i アプリは適切に例外を処理する必要があります。

(3) 事前検証プロセスの廃止

旧仮想マシンで動作させるアプリケーションプログラムのクラスファイルは、ビルド時に事前検証を行う必要がありました。これに対し、新仮想マシンでは事前検証が廃止されており、ビルド時の事前検証プロセスは不要となります（事前検証が施された i アプリも実行は可能です）。

これに伴い、NTT ドコモが提供する i アプリ開発環境は、Star-1.3 プロファイル向け以降のものはビルド時に事前検証プロセスを行わないようになっています（新仮想マシンは Star-1.3 プロファイル対応の携帯電話に搭載されます）。このため、Star-1.3 プロファイル向け以降の i アプリ開発環境でビルドした i アプリは、Star-1.2 プロファイル対応の携帯電話では動作しないため注意してください。

※本項目は新仮想マシンに関する留意点ではなく、新仮想マシン向けの i アプリ開発環境を利用する上での留意点となります。