Vol.2013-GN-86 No.21
Vol.2013-CDS-6 No.21
2013/1/17

Android におけるプロセスメモリ管理の改善

野村駿[†1]　服部拓也[†2]　永田恭輔[†2]　中村優太[†1]　山口実靖[†1†2]

Android は，スマートフォン，タブレット PC，音楽プレイヤーなど様々なデバイスで採用されており，重要性が増している．Android には，low memory killer と呼ばれる独自のプロセスメモリ管理システムが搭載されており，メモリ不足時にプロセスを強制終了し，空きメモリを確保する．このプロセスの強制終了により，再度同じアプリケーションを使用する場合にプロセスの再起動が必要となるため，ユーザの利便性が低下することがある．本稿では，low memory killer における強制終了プロセスの選定手法に着目し，新しい選定手法を 2 つ提案する．一つは，アプリケーションの新規起動時間を計測し，新規起動時間の長いアプリケーションが強制終了されづらくする手法である．もう一つはアプリケーション起動順を LRU にて管理し，最後に使用されてからの時間が短いアプリケーションを強制終了されづらくする手法である．評価の結果，両手法とも Android 標準の low memory killer の選定手法よりも合計アプリケーション起動時間を低減させることが可能であり，特に実際のユーザの使用履歴に近い環境で LRU が有効であることが確認された．

Improvement of Process Memory Management in Android

SHUN NOMURA[†1]　TAKUYA HATTORI[†2]　KYOSUKE NAGATA[†2]
YUTA NAKAMURA[†1]　SANEYASU YAMAGUCHI[†1†2]

Android is widely applied to a variety of devices, such as smartphones, tablet PCs, and music players. It has become one of the most important platforms. Android operating system has an original memory management system, which is called “low memory killer”. When enough memory is not available, low memory killer terminates processes according to its policy. However, the selection based on its policy does not always meet users’ requirement. Thus, it sometimes harms users’ convenience. In this paper, new termination policies based on time consumed by re-launch and one based on LRU are proposed. The proposed policies are implemented in Android operating system and are evaluated with application launching experiments. The experimental results have demonstrated that the times consumed by the proposed methods are smaller than that of the standard methods.

1. はじめに

Android はスマートフォン，タブレット PC，音楽プレイヤーなど様々なデバイスの OS として採用されており，2012 年第 2 四半期においてそのシェアは 68%を越え，今も増加中である[1]．この様に Android OS は広く普及しており，重要性が高まっている．Android はオープンソースソフトウェアであるため，一般の企業や開発者でも，Android のソースコードを入手し，ビルドや改変を行うことが可能である．この様に意義においても，Android は重要なプラットフォームになっている．

Android には，low memory killer と呼ばれる独自のメモリ管理システムが搭載されており，独自のルールに従ってメモリの管理を行なっている．low memory killer では，メモリの空き容量を確保するためにプロセスを強制終了する．このプロセスの強制終了により，ユーザが再度同じアプリケーションを使用する場合に，プロセスの再起動が必要となりユーザの利便性を低下させることがある．

本研究では，強制終了するアプリケーションの選定の改善によりプロセス再起動にともなうユーザの利便性の低下を軽減することを目的として，新しい選定手法を提案する．そして提案手法をオープンソースである Android に実装し，評価を行う．

2. Android OS

2.1 アーキテクチャ

Android OS はアプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイム，Linux カーネルで構成されている[2].

アプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイムには Android のために新規に開発された実装が多く含まれている．カーネルには Linux カーネルが使用されており，多くの部分は変更されることなくそのまま使用されている．ただし，次節で述べる low memory killer など Android 固有の機能も追加されている．

2.2 low memory killer

Android には，low memory killer という独自のプロセスメモリ管理システムが搭載されている．これは，メモリの空き容量が閾値まで下がった場合に起動され，*adj* と *minfree* の関係に基づいてプロセスを選定し強制終了するプログラムである． low memory killer が起動されると，強制終了するプロセスの選定のために起動している全てのプロセスの *adj* を比較し，*adj* の数値がより高いプロセスから順に強制

†1 工学院大学情 報通信工学科
Department of information and Communications Engineering, Kogakuin University
†2 工学院大学大学院 電気電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School

Vol.2013-GN-86 No.21
Vol.2013-CDS-6 No.21
2013/1/17

終了する．最高 *adj* のプロセスが複数存在する場合，メモリ使用量を比較し，メモリ使用量のより多いプロセスを強制終了する．

2.2.1 ***adj*** **と** ***minfree***

adj と *minfree* の関係は，`/init.rc` で定義されており，前述の様に，low memory killer において強制終了するプロセスを選定する際に参照される．*adj* はプロセスの状態，種類を表しており，プロセスの状態が変化するたびに数値が増減する．プロセスの状態による *adj* の値を表 1 に示す．*adj* が小さいプロセスほど強制終了されづらく，フォアグラウンドのプロセスが最も強制終了されにくい．*minfree* は，プロセス強制終了実行の閾値となる空きページ数であり，*adj* によってランク分けされている．*adj* と *minfree* の関係を表 2 に示す．表 2 中の *minfree* の値の単位はページ（4kB）である．例えば，メモリ空き容量が 38428[kB]（9607[4kB]）以下になると，*adj* の値が 9 以上の数値を持つプロセスが強制終了される候補に上がる．そして，空きメモリ量が 38428[kB]以上になるか，*adj* が 9 以上のプロセスがなくなるまでプロセスを強制終了する．

表 1 プロセスの状態，種類による *adj* の値

adj	プロセスの状態，種類
0	FOREGROUND_APP
1	VISIBLE_APP
2	PERCEPTIBLE_APP
3	HEAVY_WEIGHT_APP
4	BACKUP_APP
5	SERVICE
6	HOME_APP
7	PREVIOUS_APP
8	SERVICE_B
9	HIDDEN_APP_MIN
15	HIDDEN_APP_MAX

表 2Android4.0.3，NexusS における *adj* と *minfree* の関係

adj	*minfree*[4kB]
0	3674
1	4969
2	6264
4	8312
9	9607
15	11444

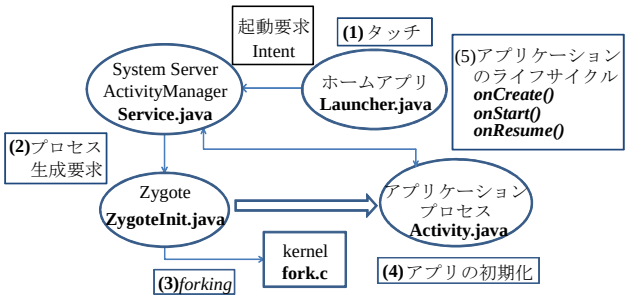

図 1 アプリケーションの起動手順

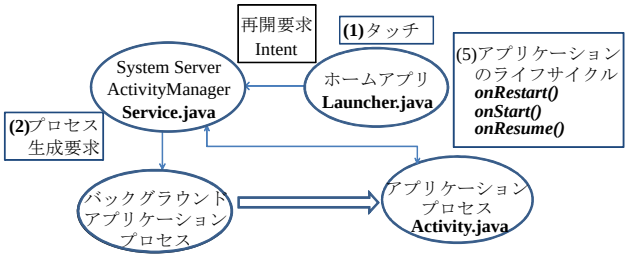

図 2 アプリケーションの再開手順

3. アプリケーションの起動手順

Android のアプリケーションを新規に起動する場合，図 1 の様に以下の手順に従って起動される．①ユーザがアプリケーションのアイコンをタッチする．そして，ホームアプリケーション (Launcher) が起動要求のインテントを ActivityManager に送信する．②ActivityManager が Zygote にプロセス生成要求を送信する．③Zygote が自分自身を fork し，子プロセスを生成する．④新しいプロセスが初期化される．⑤アプリケーションのライフサイクルに従って `onCreate()`, `onStart()`, `onResume()` が呼び出される．

また，起動済みであるがバックグラウンド状態にあるプロセスの再開は，図 2 の様に以下の手順に従って行われる．①ユーザがアプリケーションのアイコンにタッチする．そしてホームアプリケーションが再開要求のインテントを ActivityManager に送信する．②ActivityManager は対象のバックグラウンドアプリケーションプロセスに再開要求を出す．③バックグラウンドアプリケーションプロセスは再開要求を受けてアプリケーションプロセスとして起動しフォアグラウンド状態となる[3]. 本論文では前者を「新規起動」，後者を「再開」と呼ぶ．

起動済みプロセスが強制終了されることなく再利用された場合は，上記の「再開」の手続きが行われるが，low memory killer によりプロセスが強制終了されてしまった後に再利用されたときは「新規起動」の手続きが行われる．通常，「再開」よりも「新規起動」に要する時間の方が長いため，後に再利用するプロセスの強制終了はユーザの利便性を低下させることになる

Vol.2013-GN-86 No.21
Vol.2013-CDS-6 No.21
2013/1/17

4. 提案手法

本章で，アプリケーション起動時間をもとに強制終了するプロセスを選定する「起動時間ベース手法」と LRU に基づいて選定する「LRU 手法」を提案する．

4.1 起動時間ベース手法

本節で，アプリケーションの新規起動時間を考慮して強制終了するプロセスを選定する起動時間ベース手法を提案する．本手法では，アプリケーションの新規起動時間を計測する．そして，新規起動時間が長く，強制終了した場合にユーザの利便性を大きく損ねるアプリケーションを強制終了されにくくし，新規起動時間が短く利便性低下の程度が小さいアプリケーションを優先的に強制終了する．

アプリケーションの新規起動時間は，アプリケーションのライフサイクルにおける `onCreate()` の呼び出しから `onResume()` の呼び出しまでの時間とし，提案システムでは後述の手法によりアプリケーションごとの新規起動時間の履歴を取る．そして，履歴からアプリケーションの新規起動時間の平均を算出し，新規起動時間が平均より長いアプリケーションのプロセスの *adj* を 3 下げ強制終了されにくくする．これにより，アプリケーションの起動に長い時間を要しユーザの利便性を大きく損なうことが回避できると期待される．

後述の評価用実装では，アプリケーションの新規起動時間の履歴は最新の 10 個を保持し，それより古いもの破棄している．よって，長時間前に起動し最近使用していないアプリケーションが長期にわたり強制終了の対象とならないことが回避できる．履歴の数は状況に応じて変更することが可能である

4.2 LRU 手法

本節で，LRU 方式を用いて強制終了するアプリケーションを選定する手法を提案する．

本手法では，ユーザが起動したアプリケーションを LRU により管理し，LRU 順にて新しいものの *adj* を下げ強制終了の対象となりにくくする．後述の評価用実装では，以下の様に実装されている．長さ 15 の配列を用意し，ユーザが起動したアプリケーション履歴を保持する．配列内の順は LRU にて管理する．そして，配列の先頭(0 番目)から 3 番目までに格納されているアプリケーション(最近使われたアプリケーション)の *adj* を-5 する．同様に，4 番目から 9 番目に格納されているアプリケーションの *adj* を-3, 10 番目から 12 番目に関して *adj* を-2, 13 番目から 14 番目に関して *adj* を-1 する．履歴内に記録のないアプリケーションの *adj* は変更を加えない．履歴の長さや，履歴順の閾値，*adj* の下げ幅は変更可能である．

4.3 実装

提案手法を実装するために，Android のカーネルおよびフレームワークに変更を加えた．以下にその詳細について述べる．

4.3.1 カーネル内部

カーネル内部ではまず，`lowmemorykiller.c` において low memory killer が強制終了するプロセスを選定する箇所に変更を加えた．起動時間ベース手法に関しては，ユーザ空間より/proc インターフェイスを介してアプリケーションの起動時間の情報を受け取り，それをカーネル内の配列に格納して履歴として管理する．そして，メモリが不足し強制終了プログラムを選定する際には，履歴内の起動時間の平均と，各アプリケーションの起動時間を比較し，起動時間が平均より長いアプリケーションに関しては *adj* を 3 減らして評価する(4.1 節参照)．

LRU 手法では，ユーザ空間プログラムよりアプリケーション起動時刻の情報を得て，アプリケーションの起動順をカーネル内の配列に LRU にて管理する．

起動時間ベース手法，LRU 手法ともに，有限長の配列にて起動履歴を管理しているため，最近使用していないアプリケーションは履歴から削除される．よって最近使用していないアプリケーションが履歴に残り続けて強制終了されづらくなることとは発生しない．

4.3.2 ユーザ空間部

ユーザ空間部では，アプリケーションフレームワークの実装にアプリケーションライフサイクルメソッド(`onCreate()` など)の呼び指し時刻を記録する機能と，記録されたメソッド呼び出し時刻とアプリケーション起動時間を/proc インターフェイスを通じてカーネル内の low memory killer に伝える機能を追加した．起動時間ベース手法ではアプリケーション起動時刻と起動時間を，LRU 手法ではアプリケーションの起動時刻のみを伝えている．

5. 評価

5.1 評価方法

標準 low memory killer，起動時間ベース手法，LRU 手法が実装されている Android スマートフォンにおいて，複数のアプリケーションを順に起動し，その起動に要した時間を測定した．実験は，表 3 に示す仕様のスマートフォンを用いて行った．

表 3 実験環境

Device name	Nexus S
OS	Android4.0.3
CPU	Cortex A8（Hummingbird）Processer 1GHz
Memory	512MB

5.2 評価シナリオ

起動するアプリケーションの順番を定めたものを本稿では“シナリオ”と呼び，本実験では図 3 のシナリオ A~C とシナリオ D の 4 種類のシナリオにより，評価を行った．シナリオ A は，使用頻度が高いと思われるアプリケーションを我々が選定し，それらを連続して使用するシナリオで

Vol.2013-GN-86 No.21
Vol.2013-CDS-6 No.21
2013/1/17

ある．現実の使用順との類似性は考慮していない．ベンチマークアプリケーション 1，2 は測定用に我々が作成した，アプリケーションであり、メモリを消費する起動時間の長いアプリケーションである．シナリオ B は現実の使用順を模倣して我々が作成したシナリオである．シナリオ C は，ユーザの許可を得て取得した実際のユーザの使用履歴である．シナリオ D は，Opera Mobile browser，Browser（Android 標準），2chMate，TkMixiViewer，Gmail，Camera，twicca の中から連続しないようランダムに 100 個選定しシナリオとした．ランダムな選定であるが，標準 low memory killer，起動時間ベース手法，LRU 手法の全てに同一のシナリオを適用して評価を行った．

5.3 評価結果

シナリオ A～D における評価結果を図 4～7 に示す．各図の縦軸は，シナリオのホームアプリケーション（Launcher）以外の全アプリケーションの起動時間の合計であり，少ないほど優れていると言える．

図より，全てのシナリオにおいて標準手法より両提案手法の方が合計起動時間が短く，両提案手法ともに標準手法より優れた手法であると言える．起動時間ベース手法と LRU 手法を比較すると，実際のユーザの使用順に近いシナリオにおいて，特に LRU が優れていることがわかる．

6. 考察

今回提案した手法では，アプリケーションの新規起動時間は `onCreate()` の開始時刻から `onResume()` の開始時刻までとしており，再開時間は `onRestart()` の開始時刻から `onResume()` の開始時刻までとしている．しかし，アプリケーションによっては `onResume()` が開始した後にデータの読み込みやページの読み込みを行い，その間ユーザが待たされることになる．この様な例の場合，実際にユーザの利便性を損なっている程度は，本稿で定義したアプリケーション起動時間を大きく上まわることになり，提案手法により得られるユーザの体感的利便性の向上は前章の評価を大きく上まわるものと期待できる．例えば，Web ブラウザアプリケーションの多くがこれに該当し，プロセスが強制終了されてしまった後にアプリケーションを使用すると，`onCreate()` 呼び出しから `onResume()` 呼び出しまでの間ユーザが待たされ，さらにその後に以前閲覧していたページの再読み込みが発生し，ユーザはさらに長い時間待たされることになる．そのため，本稿で定めたアプリケーションの起動ではなく，`onResume()` の開始後に発生する処理に要する時間も考慮した手法を適用することでユーザの利便性の損失のさらなる軽減を図ることができると予想される．

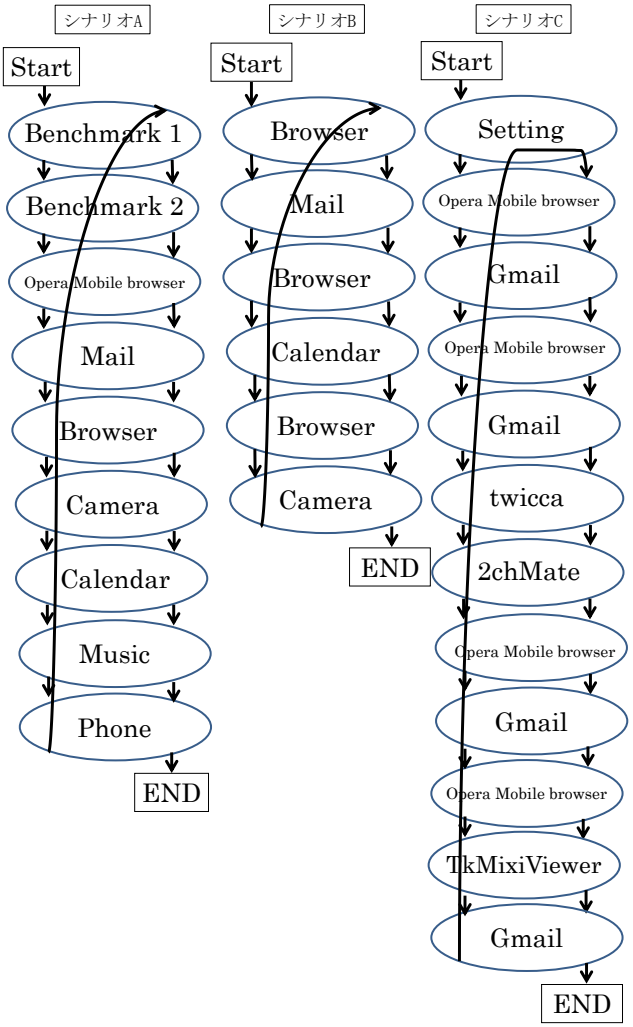

図 3 評価シナリオ A～C

7. 関連研究

Android アプリケーションの起動時間の調査方法に関する研究としては，文献[2][4]の研究がある．これらにおいてユーザによるスクリーンのタッチからアプリケーション起動の完了までの時間の計測方法が示されている．また，実際のアプリケーション起動時間の測定結果と考察が示されている．本稿における起動時間の測定は当該研究で提案されている手法に基づき行われている．

Linux カーネルのモニタリングツールとしては，FTrace[5][6]，SystemTap[7][8]，LTTng[9][10]，OProfile[11]がある．しかし，これらは Linux 用に構築されているため Android の解析には適さず，本研究で必要となるアプリケーションフレームワークや Dalvik VM の動作の解析ができない．具体的には，アプリケーションフレームワーク内におけるアプリケーションライフサイクル(`onCreate()`，`onRestart()`，`onStart()`，`onResume()`)の時刻を取得することができず，アプリケーション起動時間の調査をすることができない．また，我々の手法と比較して，これらの手法は計測のオーバーヘッドが大きくなっている．

アプリケーション起動時間の短縮に関する研究としては，Zygote の preload クラスの増加による起動時間の短縮の研究[12]がある．当該研究は，Zygote による読込済みクラスの数を増加させることによりアプリケーション起動時にストレージから読み込むクラスを減少させ，起動時間の短縮を実現している．しかし，強制終了されたアプリケーションの起動時間の短縮を考える当該研究の手法と，強制終了するアプリケーションの最適化を考える本稿の手法は排他的な関係になく，両手法を同時に用いていくことが好ましいと考えられる．

8. おわりに

本論文で我々は low memory killer における強制終了プロセスの選定手法に着目し，アプリケーションの起動時間を考慮した選定手法と LRU による選定手法を提案した．そしてこれらの手法を評価し，提案手法が標準手法よりもアプリケーション起動時間を短縮できることを確認した．特に，実際のユーザの使用順に近いシナリオにおいては，LRU による手法が優れていることを確認した．

今後は，`onResume()` の開始後に発生する処理に要する時間も考慮した選定手法について考察していく予定である．

謝辞

本研究は JSPS 科研費 22700039, 24300034 の助成を受けたものである．

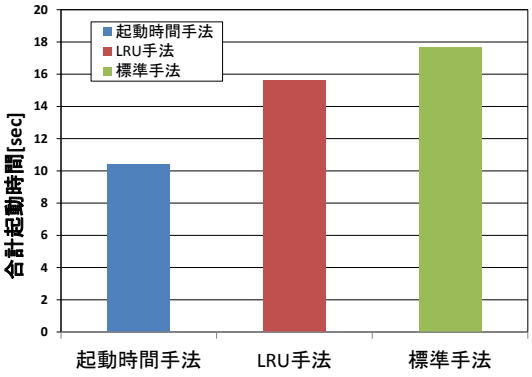

図 4 シナリオ A における評価結果

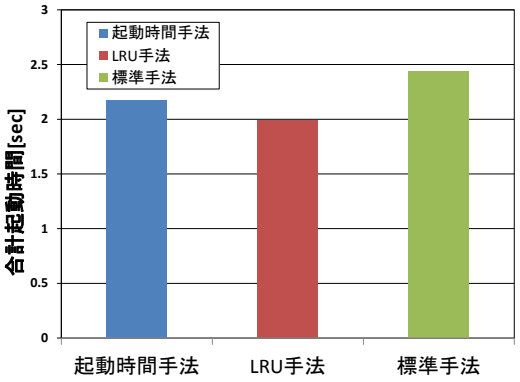

図 5 シナリオ B における評価結果

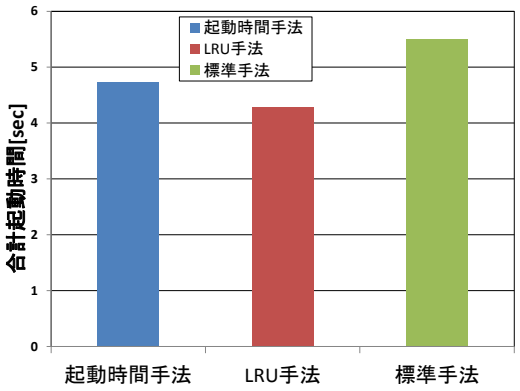

図 6 シナリオ C における評価結果

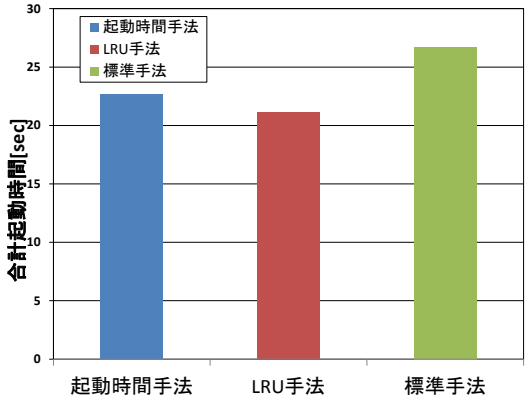

図 7 シナリオ D における評価結果

参考文献

1）Android and iOS Surge to New Smartphone OS Record in Second Quarter, According to IDC : http://www.idc.com/getdoc.jsp?containerId=prUS23638712
2）What is Android?|Android Developers: http://developer.android.com/guide/basics/what-is-android.html
3）永田恭輔，山口実靖：Android アプリケーションの起動性能解析システムとその評価，マルチメディア，分散，協調とモバイル(DICOMO2012)シンポジウム，pp83-90(2012)
4）Kyosuke Nagata, Saneyasu Yamaguchi, "An Android Application Launch Analyzing System ," 8th ICCM: 2012 International Conference on Computing Technology and Information Management, (2012/04/24)
5）Steve Rostedt. ftrace tracing inftrastructure. http://lwn.net/Articles/270971/. SystemTap http://sourceware.org/systemtap/
6）Frank Ch. Eigler. “Problem solving with systemtap,” In Proceedings of the Ottawa Linux Symposium 2006, 2006.
7）LTTng Project http://lttng.org/
8）T. Bird, “Measuring Function Duration with Ftrace,” in Proc. of the Japan Linux Symposium, 2009.
9）M. Desnoyers, M. R. Dagenais, "The LTTng Tracer: A low impact performance and behavior monitor of GNU/Linux" Proceedings of Ottawa Linux Symposium 2006, 2006, pp. 209-223.
10）J. Levon and P. Elie. Oprofile: A system profiler for linux. http://oprofile.sf.net, September 2004.
11）Valgrind http://valgrind.org
12）永田 恭輔，山口 実靖, "Android アプリケーションの起動時間に関する一考察", 情報処理学会 研究報告 システムソフトウェアとオペレーティング・システム（OS），Vol.2012-OS-123 (2012)