



BORLAND® DEVELOPER CAMP

チュートリアルセッション#2

Java, Delphi, C++Builderユーザのための メモリリーク, ボトルネックの検出手順

Borland®

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の転載を禁止します。



BORLAND® DEVELOPER CAMP

第2回 ボーランド デベロッパー キャンプ

講師紹介

- 高橋智宏
- 1973年生まれ、京都大学 法学部卒
- エバンジェリスト 兼 コンサルタント 兼 トレーナー...
- 学生の時購入したTurboC++2ndからの熱狂的なボーランドファン
- 参加しているメーリングリストやコミュニティ
 - JBuilder ML, C++Builder ML, Delphi ML, C# ML, CORBA ML 等...
 - ミクシィ
 - http://mixi.jp/show_friend.pl?id=208738
- 「Java読書会」を運営
 - <http://www.javareading.com/bof/>

Borland®

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の転載を禁止します。

アジェンダ

- Javaのメモリリークの落とし穴
 - Optimizelt Profiler for Java
- C++Builder6, C++Builder2006のメモリリークの落とし穴
 - CodeGuard
- Delphi for Win32のメモリリークの検出方法
 - MemCheck, FastMM(BDS2006), 本家FastMM
- Delphi for .NETのメモリリークの落とし穴
 - Optimizelt Profiler for .NET1.1
- CPUプロファイリング手順
 - Java, C++Builder, Delphi for Win32, Delphi for .NET
 - Delphi7 と BDS2006 でメモリマネージャの比較

Borland®

Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。

Javaのメモリリークの落とし穴

- 基本的にJavaにはメモリリークが無いハズ...

```
public class Frame1 extends JFrame {
    JButton jButton1 = new JButton();
    public Frame1() {
        ...
        jblnit();
    }
    private void jblnit() throws Exception {
        ...
        jButton1.setText("フレームを表示");
        jButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jButton1_actionPerformed(e);
            }
        });
    }
}

// フレーム(test.Frame2)を表示
public void jButton1_actionPerformed(ActionEvent e) {
    test.Frame2 f = new test.Frame2(); // Frame2を生成
    f.setVisible(true);                // Frame2を表示
}
```

```
public class Frame2 extends JFrame {
    JButton jButton1 = new JButton();
    public Frame2() {
        ...
        jblnit();
    }
    private void jblnit() throws Exception {
        ...
        jButton1.setText("閉じる");
        jButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jButton1_actionPerformed(e);
            }
        });
    }
}

public void jButton1_actionPerformed(ActionEvent e) {
    setVisible(false); // Frame2(自分自身)を閉じる
}
```

Borland®

Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。

Optimizelt Profiler for Java で監視してみる

- 何故か test.Frame2 を含む様々なインスタンスがガベコレされない...

メモリプロファイラで見ると
インスタンス数が増えたまま

Borland®

クラス名	インスタンス数	メモリ	サイズ	サイズの差分
javax.swing.plaf.basic.BasicRootPaneUI\$RootPaneInput	10	+	240 B	+ 120 B
javax.swing.JLayeredPane	10	+	3280 B	+ 1640 B
java.awt.FlowLayout	10	+	320 B	+ 160 B
javax.swing.plaf.basic.BasicButtonListener	10	+	240 B	+ 120 B
java.lang.Integer	427	+	6832 B	+ 0 B
javax.swing.DefaultButtonModel	10	+	320 B	+ 160 B
sun.awt.windows.Win32SurfaceData	10	+	560 B	+ 280 B
java.awt.Color	53	+	2120 B	+ 200 B
sun.awt.RepaintArea	10	+	160 B	+ 0 B
java.lang.ref.PhantomReference	13	+	312 B	+ 120 B
sun.awt.im.InputMethodContext	10	+	640 B	+ 320 B
sun.awt.windows.WInputMethod	10	+	400 B	+ 200 B
java.awt.Insets	14	+	208 B	+ 104 B
test.Frame2	0	+	3,312 B	+ 1,840 B
javax.swing.JRootPane\$RootLayout	10	+	160 B	+ 80 B
javax.swing.AbstractButton\$Handler	10	+	160 B	+ 80 B
sun.awt.windows.WFramePeer	10	+	1,120 B	+ 680 B
javax.swing.JButton	10	+	4,400 B	+ 2,200 B
sun.awt.im.InputMethod\$Color	11	+	264 B	+ 120 B
java.awt.LightweightDispatcher	10	+	480 B	+ 240 B
javax.swing.JRootPane	10	+	3,800 B	+ 1,800 B
javax.swing.SystemEventQueueUtilities\$ComponentNotif	10	+	160 B	+ 80 B
javax.swing.ComponentInputMap	10	+	240 B	+ 120 B
sun.java2d.DefaultDispatcherRecord	13	+	312 B	+ 120 B

test.Frame2インスタンスは確かに参照されている...

- 自分のコード(イベントハンドラ)が参照しているのは当たり前だが...

参照ツリーを使って、
参照されている様子を確認

- ネイティブピアからの参照が残らないよう、dispose()またはJFrame.DISPOSE_ON_CLOSEオプションが必要

Borland®



 第2回 ボーランド デベロッパーズ キャンプ

C++Builder特有の落とし穴

- try/ __finally でメモリリークを防ぐ

```

class MyClass {
};

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    MyClass* p = NULL;
    try {
        p = new MyClass();
        try {
            int x = StrToInt("xyz");
        }
        catch(const Exception& ex) {
            return;
        }
        catch(...) {
            return;
        }
    }
    __finally {
        if(p) {
            delete p; // 解放!!
        }
    }
}
          
```

- VCLインスタンスのリーク

```

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    new TButton((TComponent*)NULL); // Owner無し!!
}
          
```



Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



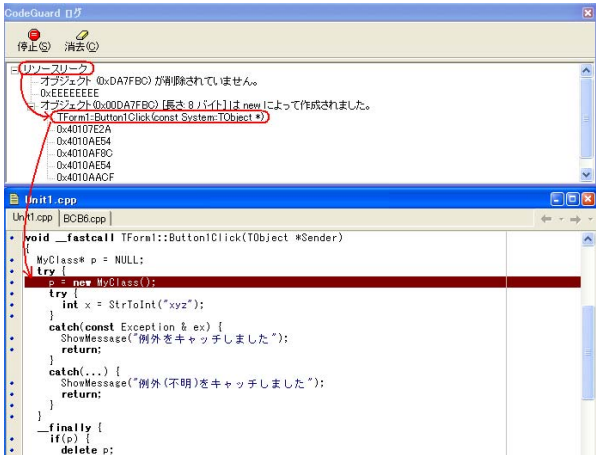
 第2回 ボーランド デベロッパーズ キャンプ

C++Builder6 の CodeGuard で確認してみる

- C++Builder6では
 - なぜか __finally が呼び出されない(仕様?)
 - TButtonインスタンスのリークは見つけれない(仕様?)

MyClassオブジェクトが削除されていないと報告されました。

- オススメは
 - std::auto_ptr
 - boost::scoped_ptr



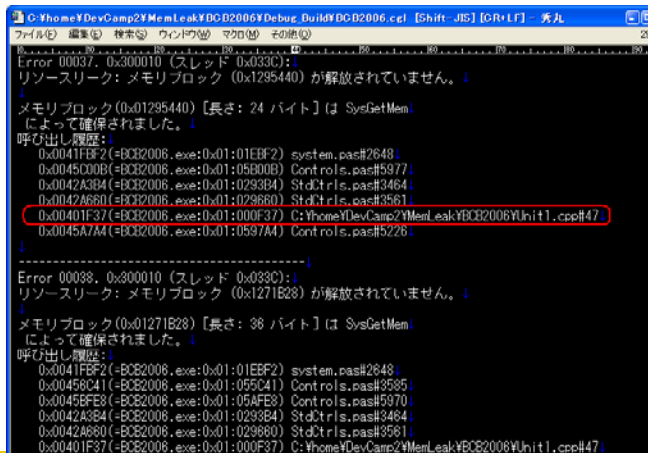


Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。

C++Builder2006では？

- catchの中からreturnしても __finallyが呼び出されるようになりました
- VCLのインスタンスのリークも報告されるようになりました
 - 但し、動的RTL(cc3270xx.dll)とパッケージ(xxxx.bpl)はOFFにする

CodeGuardのログファイルに、
ソースコードと行番号が出力されています



```

C:\Vhome\DevCamp2\MemLeak\BCE2006\Debug_Build\BCE2006.ccl [Shift-JIS] [C-R-L-F] 秀丸
Error 00037, 0x300010 (スレッド 0x033C):
リソースリーク: メモリブロック (0x01295440) が解放されていません。
メモリブロック (0x01295440) [長さ: 24 バイト] は SysGetMem
によって確保されました。
呼び出し履歴:
0x0041F3F2(-BCE2006.exe:0x01:01EBF2) system.pas#2648
0x0045C00B(-BCE2006.exe:0x01:05800B) Controls.pas#5877
0x0042A3B4(-BCE2006.exe:0x01:0293B4) StdCtrls.pas#3464
0x0042A660(-BCE2006.exe:0x01:029660) StdCtrls.pas#3561
0x00401F37(-BCE2006.exe:0x01:000F37) C:\Vhome\DevCamp2\MemLeak\BCE2006\Unit1.cpp#47
0x0045A7A4(-BCE2006.exe:0x01:0597A4) Controls.pas#5226
Error 00038, 0x300010 (スレッド 0x039C):
リソースリーク: メモリブロック (0x01271B28) が解放されていません。
メモリブロック (0x01271B28) [長さ: 36 バイト] は SysGetMem
によって確保されました。
呼び出し履歴:
0x0041F3F2(-BCE2006.exe:0x01:01EBF2) system.pas#2648
0x0045B4C1(-BCE2006.exe:0x01:055C41) Controls.pas#3585
0x0045BFE8(-BCE2006.exe:0x01:05AFB8) Controls.pas#5970
0x0042A3B4(-BCE2006.exe:0x01:0293B4) StdCtrls.pas#3464
0x0042A660(-BCE2006.exe:0x01:029660) StdCtrls.pas#3561
0x00401F37(-BCE2006.exe:0x01:000F37) C:\Vhome\DevCamp2\MemLeak\BCE2006\Unit1.cpp#47
    
```

Borland®

C++Builder と catch(...)

- 例外の補足に catch(...) のみを使用するのはどう？
- VCL例外に関連するメモリがリークするので注意!! (バグ?)
 - C++Builder2006のCodeGuardでチェックしてみよう

誤

```

try {
    int x = StrToInt("xyz");
}
catch(...) {
    ShowMessage("...");
}
    
```

正

```

try {
    int x = StrToInt("xyz");
}
catch(const Exception& ex) {
    ShowMessage("...");
}
catch(...) {
    ShowMessage("...");
}
    
```

Borland®



 第2回 ボーランド デベロッパーズ キャンプ

Delphi5～Delphi2005(Win32) - MemCheck

- Delphi2005までは MemCheck がオススメ
 - <http://v.mahon.free.fr/pro/freeware/memcheck/>
- 無料(MemCheck.pasのみ)
- Delphi2006では利用不可！！
 - Delphiコンパイラやメモリマネージャの変更による影響の為



Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



 第2回 ボーランド デベロッパーズ キャンプ

Delphi5～Delphi2005(Win32) - MemCheck (続き)

- MemCheck.pasをプロジェクトに追加
- MemChk; の呼び出しを追加
- 「スタックフレームの生成」をON
- 「TD32デバッグ情報を含める」をON
- プログラム終了時にログファイルが生成される

```

begin
  MemChk; // チェック開始
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.

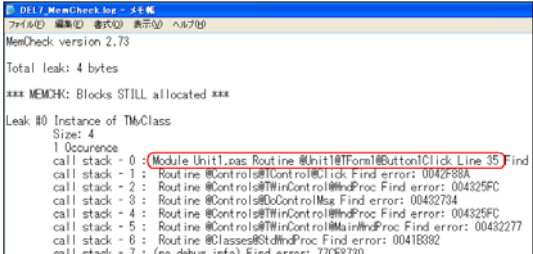
```

```

type
  TMyClass = class
  end;

procedure TForm1.Button1Click(Sender: TObject);
var
  p: TMyClass;
begin
  p := TMyClass.Create; // 破棄されていない
end;

```



Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



BORLAND® DEVELOPERCAMP
第2回 ボーランド デベロッパー キャンプ

Delphi2006 for Win32 - FastMM

- BDS2006から、メモリマネージャがFastMMベースに置き換わった
 - BDNの記事 <http://bdn.borland.com/article/33624>
 - 今までより高速。後ほど確認します
- メモリリーク報告機能が付属している
 - `ReportMemoryLeaksOnShutdown := True;` の行を追加するだけ
 - プログラム終了時にダイアログボックスが表示される
 - しかし、インスタンスの生成場所までは教えてくれない！

```
begin
  ReportMemoryLeaksOnShutdown := True; // これだけ
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Unexpected Memory Leak

 An unexpected memory leak has occurred. The unexpected small block leaks are:
1 - 12 bytes: TMyClass x 1

OK

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



BORLAND® DEVELOPERCAMP
第2回 ボーランド デベロッパー キャンプ

Delphi2006 with 本家FastMM

- リークしたインスタンスの生成場所は、実は Delphi2006 でも調べることができる!!
 - BDS2006付属のFastMMを、本家FastMMの最新版に置き換えよう
 - http://sourceforge.net/project/showfiles.php?group_id=130631
 - ライセンスはMPL
 - 最新版は4.70 (※)

(※)Delphi2006以前のバージョンでも
利用可能

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。

Delphi2006 with 本家FastMM (続き)

- usesに ShareMemユニット を追加して、borlndmm.dll(共用メモリマネージャ) がロードされるように変更する
- 本家FastMMの「Debug版 BorlndMM.dll」がロードされるようにする
- 本家FastMMの「FastMM_FullDebugMode.dll」が実行時に必要
- 「スタックフレームの生成」をON
- 「TD32デバッグ情報を含める」をON
- BDS2006のIDEが起動している必要あり
- プログラム終了時にダイアログボックスが表示される
- プログラム終了時にログファイルが生成される
 - borlndmm_MemoryManager_EventLog.txt
- プログラムは最終的にAVで停止することがあるが「気にせず」タスクマネージャやIDE上で無理矢理終了させる

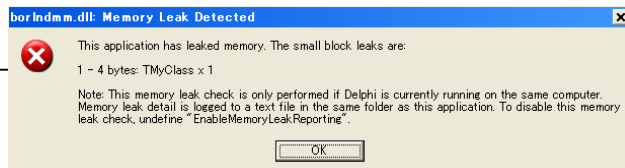
Borland®

Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。

Delphi2006 with 本家FastMM (続き)

```
program XXXX;

uses
  ShareMem, // これだけ
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
...
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```



--- borlndmm_MemoryManager_EventLog.txt -----
A memory block has been leaked. The size is: 4

Stack trace of when this block was allocated (return addresses):
402C9A [System][@GetMem]
40383B [System][TObject.NewInstance]
403BAA [System][@ClassCreate]
403870 [System][TObject.Create]
403A33 [System][@IsClass]
453693 [Unit1.pas][Unit1][TForm1.Button1Click][34]
437052 [Controls][TControl.Click]
426537 [StdCtrls][TButton.Click]
426635 [StdCtrls][TButton.CNCommand]

The block is currently used for an object of class: TMyClass
...
...

Borland®

Copyright © 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



 第2回 ボーランド デベロッパー キャンプ

Delphi for .NET特有の落とし穴

- 基本的には.NETにはメモリリークが無いハズ...

```

unit Unit1;
...
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    ...
  end;
...
implementation

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TForm2;
begin
  f := TForm2.Create(nil); // TForm2を生成(Ownerはnil)
  f.ShowModal;           // TForm2を表示
end;

```

```

unit Unit2;
...
type
  TForm2 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    ...
  end;
...
implementation

procedure TForm2.Button1Click(Sender: TObject);
begin
  Close; // TForm2(自分自身)を閉じる
end;
end.

```

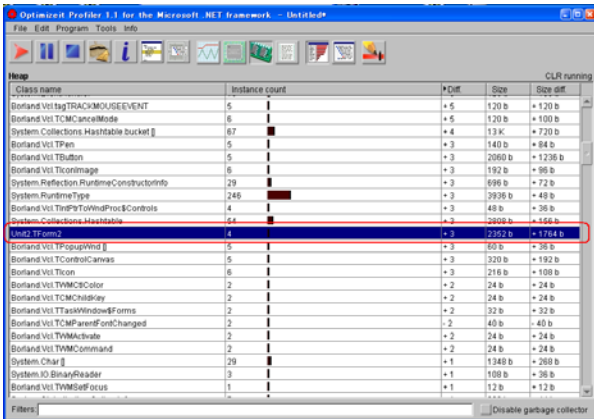

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



 第2回 ボーランド デベロッパー キャンプ

Optimizelt Profiler for .NET1.1(※) で監視してみる

- 何故か Unit2.TForm2 等のインスタンスがガベコレされない...



- Ownerがnilでも、デフォルトのOwnerがいるので、x.Free;またはFreeAndNil(x);の呼び出しは必須


(※)単体販売は終了しています。Delphi2005 Architect版にのみライセンスが付属しています。

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



第2回 ボーランド デベロッパーズ キャンプ

CPUプロファイリング - Java

```

public class Frame1 extends JFrame {
    ...
    JButton jButton1 = new JButton();
    public Frame1() {
        jblnit();
    }

    private void jblnit() throws Exception {
        ...
        jButton1.setText("jButton1");
        jButton1.addActionListener(new Frame1_jButton1_actionAdapter(this));
        contentPane.add(jButton1, java.awt.BorderLayout.SOUTH);
    }

    public void jButton1_actionPerformed(ActionEvent e) {
        long st = System.currentTimeMillis();
        long et = st;
        while( (et-st) < 5000 ) {
            Integer.parseInt("123");
            et = System.currentTimeMillis();
        }
        JOptionPane.showMessageDialog(this, "終わりました");
    }
}
        
```

5秒間のループ処理



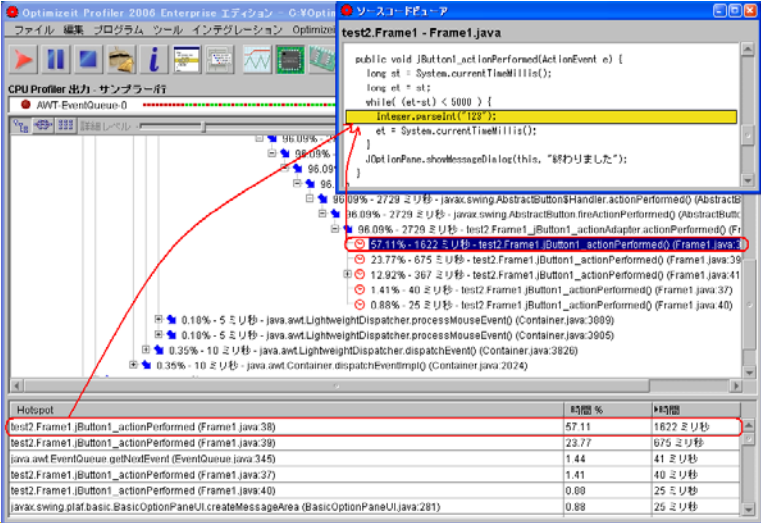
Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の複製を禁じます。



第2回 ボーランド デベロッパーズ キャンプ

Optimizelt Profiler for Java によるCPUプロファイリング

- デフォルトは、5ミリ秒間隔のサンプリングを実施



Hotspot	時間 %	時間
test2.Frame1_jButton1_actionPerformed (Frame1.java:38)	57.11	1622 ミリ秒
test2.Frame1_jButton1_actionPerformed (Frame1.java:39)	23.77	675 ミリ秒
java.awt.EventQueue.getNextEvent (EventQueue.java:345)	1.44	41 ミリ秒
test2.Frame1_jButton1_actionPerformed (Frame1.java:37)	1.41	40 ミリ秒
test2.Frame1_jButton1_actionPerformed (Frame1.java:40)	0.88	25 ミリ秒
javax.swing.plaf.basic.BasicOptionPaneUI.createMessageArea (BasicOptionPaneUI.java:281)	0.88	25 ミリ秒



CPUプロファイリング - C++Builder

```
...  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    unsigned long s = GetTickCount();  
    unsigned long e = s;  
    while( (e-s) < 5000 )  
    {  
        char buf[4];  
        sprintf(buf, "%s", "123");  
        e = GetTickCount();  
    }  
    ShowMessage("終わりました");  
}  
...
```

} 5秒間のループ処理

CPUプロファイリング - C++Builder (続き)

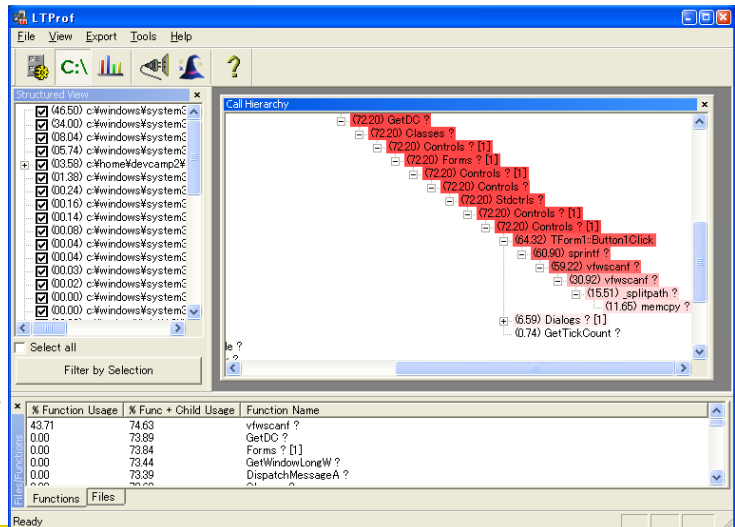
- LTProf の紹介
 - Lightweight Technologies社の製品
 - <http://www.lw-tech.com/>
 - C++Builder/Delphiに対応したCPUプロファイラ
 - 価格は \$49.95(USD)
 - TD32デバッグ情報だけでOK

LTProfによるCPUプロファイリング

- デフォルトは、3ミリ秒間隔のサンプリングを実施

処理時間が長いメソッドが
赤くハイライトされています

Borland®



CPUプロファイリング - Delphi for Win32

```

...
procedure TForm1.Button1Click(Sender: TObject);
var
  s,e: Cardinal;
  i: Integer;
begin
  s := GetTickCount;
  e := s;
  while (e-s) < 5000 do
  begin
    i := StrToInt('123');
    e := GetTickCount;
  end;
  ShowMessage('終わりました');
end;
...

```

5秒間のループ処理

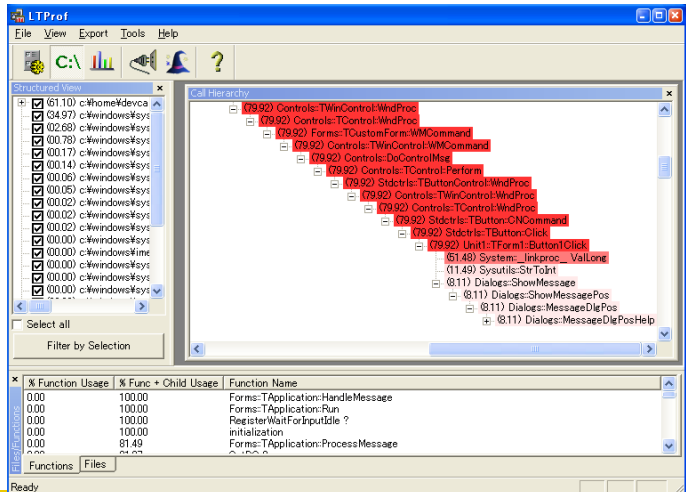
Borland®

LTProfによるCPUプロファイリング

- 「TD32デバッグ情報を含める」をON
- 「デバッグ版DCUを使う」をON

処理時間が長いメソッドが
赤くハイライトされています

Borland®



CPUプロファイリング - Delphi for .NET

```

...
procedure TForm1.Button1Click(Sender: TObject);
var
  s,e: Cardinal;
  i: Integer;
begin
  s := GetTickCount;
  e := s;
  while (e-s) < 5000 do
  begin
    i := StrToInt('123');
    e := GetTickCount;
  end;
  ShowMessage('終わりました');
end;
...

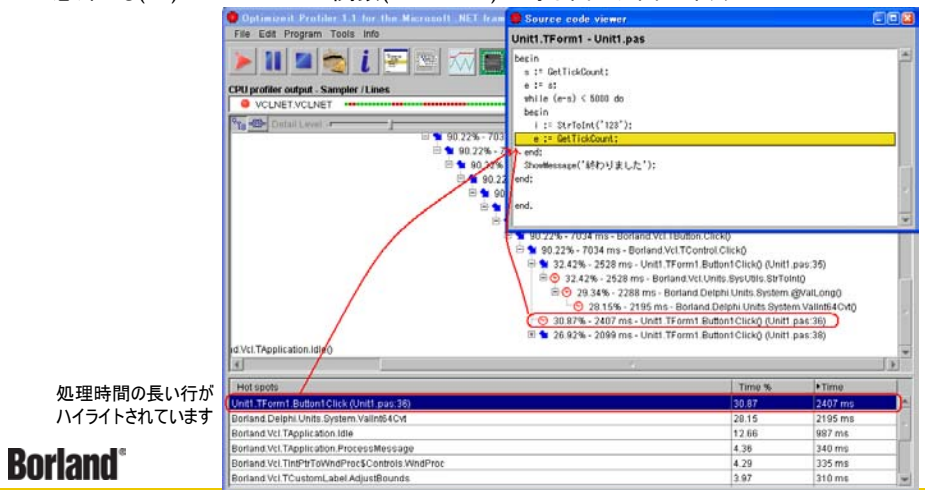
```

} 5秒間のループ処理

Borland®

Optimizelt Profiler for .NET1.1 によるCPUプロファイリング

- デフォルトは、5ミリ秒間隔のサンプリングを実施
- 意外にも(?) GetTickCount関数(Win32API) の呼び出しがボトルネックに



Borland®

Delphi7 と BDS2006 でメモリマネージャの比較

- Delphi7は、従来のメモリマネージャを使用
- BDS2006は、製品に付属するFastMMベースのものを使用

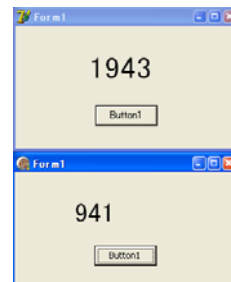
```

procedure TForm1.Button1Click(Sender: TObject);
var
  i,j: Integer;
  s,e: Cardinal;
  list: TStringList;
begin
  s := GetTickCount;
  for i := 1 to 100 do
  begin
    list := TStringList.Create;
    for j := 1 to 10000 do
    begin
      list.Add(IntToStr(j));
    end;
    FreeAndNil(list);
  end;
  e := GetTickCount;

  Label1.Caption := IntToStr(e-s);
end;

```

TStringListを使った重いループ処理



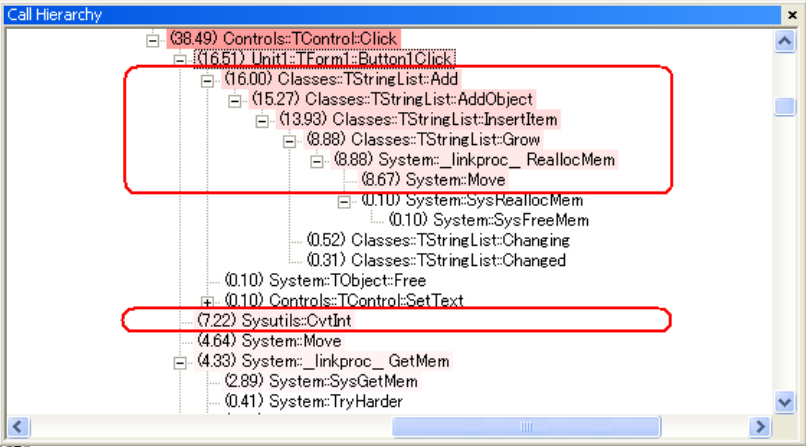
Borland®



 第2回 ボーランド デベロッパー キャンプ

Delphi7だと...

- 大量のTStringList.Addに伴うメモリの再配置処理(ReallocMem,Move)が最も重く
- IntToStrメソッド(Sysutils::CvtInt)がその次に重い



Borland

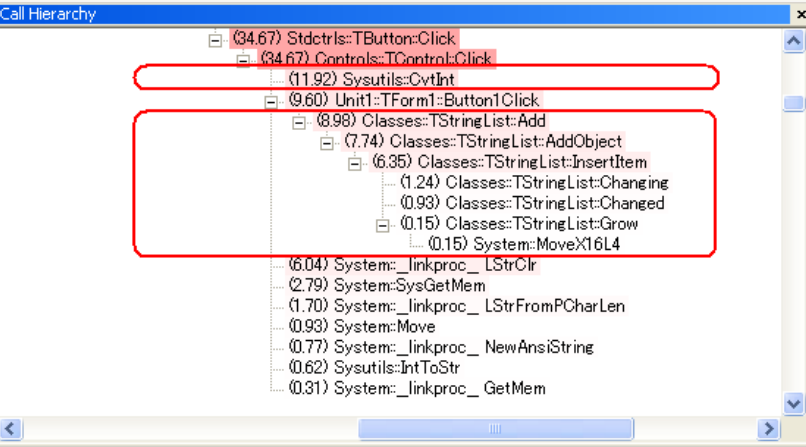
Copyright (C) 2006, Borland Software Corporation. 本記事の一部または全部の複製を禁じます。



 第2回 ボーランド デベロッパー キャンプ

BDS2006だと...

- メモリの再配置処理は軽くなり、逆にIntToStr(Sysutils::CvtInt)が目立つ結果に。
 - Sysutils::CvtIntの実装内容は、Delphi7とBDS2006とで同一です



Borland

Copyright (C) 2006, Borland Software Corporation. 本記事の一部または全部の複製を禁じます。



BORLAND® DEVELOPER CAMP

Q&A

Borland®

Copyright (C) 2006, Borland Software Corporation. 本文書の一部または全部の転載を禁止します。