Troubleshooting issues with debugging and profiling (with PDS for OE)

Nischal Y
Principal Software Engineer
November 16th 2017
Software maintenance typically consumes 40 to 80% (60% average) of software costs

Source: "Frequently Forgotten Fundamental Facts about Software Engineering" by Robert L. Glass, (an article in IEEE Software May/June 2001)
Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.

Agenda

- Part 1: Debugging tools
  - To correct faults
- Part 2: Profiling tools
  - To improve the performance
Session goals

- **What** are the different tools available to debug and profile ABL applications?
- **How** do you use these tools to troubleshoot issues?
- **When** to use what?
Part 1: Debugging tools

- Tools available
- Why PDS Debugger
- Basics of PDS Debugger
- Debugging unit test cases
- Debugging Servers
  - Classic AppServer
  - Progress Application Server (PAS)
What tool are you using?
Tools available

Standalone Debugger

PDS Debugger
Why PDS Debugger

- Battle tested Eclipse tooling
- Supports debugging client, servers, unit testcases etc.
- Uses sources not debug listings
- Concurrent debugging sessions
- Persists breakpoints & configurations
- Inspecting complex objects
- Debugging Agents and sessions is a breeze
Debugging different applications

- **PDS Debugger**
  - **Client**
    - Local
  - **Unit Testcases**
  - **Servers**
    - Remote (Attachable debugger)
Basics of PDS Debugger (Client debugging)

- Enable debugging

![Image of command line output showing prodebugenable -enable-all]

![Image of Enable Debugger window explaining The OpenEdge AVM is not enabled for debugging]

© 2017 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.
Debugging user flow

Breakpoints

Launch configuration

Debug
Debugging remote clients

- Debug mode
- Breakpoints
- Remote debug launch
- Debug
ABLUnit unit testing framework

- Introduced in 11.4
- Based on xUnit framework
- Object oriented and Procedure paradigms
  - Test methods and Test procedures
- Has good tooling support from PDS for OE
- Can be integrated with Continuous integration
- Integrated with PDS debugger (11.7)
Debugging Unit testcases
Let's see a short demo
Server debugging

PDS Debugger

Classic AppServer

Progress Application Server

Local

Remote (Attachable debugger)
Case study: Debugging AppServer agents

asbroker1.server.log
This process has been made ready using port 9999 to be used by the attachable Debugger. (11826)
This process has been made ready using port 9998 to be used by the attachable Debugger. (11826)

Customer

Remote machine

Debug Ports

9999
9998
9997

Agents

A1
A2
A3

AppServer

[UBroker.AS.asbroker1]
minSrvrInstance=3
debuggerEnabled=1
svrStartParam=-debugReady 0

ubroker.properties
Can you make it simple?
Solution: AppServer broker debugging
The image shows a configuration panel for a remote machine named `broker debug`. The following properties are configured:

- **Connection type**: AppServer broker
- **Host**: localhost
- **Debug port**: 3099
- **Password**: Not specified

Additionally, there is a diagram illustrating the connection between a broker debugger and remote machines. The diagram includes:

- A broker debugger labeled `3099` and `Agents` connected to an `AppServer` on a remote machine.
Remote machine

**Debug**

- **Servers**
  - New_configuration (1) [Progress External OpenEdge AVM]
    - asbroker1 at localhost:3099
      - "C:\Progress\OpenEdge117\bin\proapsv.exe" 10424
        - OnServer.p( ) line: 16
      - "C:\Progress\OpenEdge117\bin\proapsv.exe" 10372
        - OnServer.p( ) line: 16
  - Client [Progress OpenEdge Application]
    - C:\Progress\OpenEdge117\bin\prowin.exe (29 Oct, 2017 6:00:07 PM)
  - Client [Progress OpenEdge Application]
    - C:\Progress\OpenEdge117\bin\prowin.exe (29 Oct, 2017 6:00:24 PM)
Progress Application Server (PAS) debugging

- `oedbg.war` should be deployed to enable debugging
  - Location: `$DLC/servers/pasoe/extras/oedbg.war`
- Can be deployed onto existing PAS instance
- Created in new instance with `oedbg.war` with `-f` option
PAS Local vs Remote

Remote

Local

New_configuration (1) [Progress External OpenEdge AVM]
oepas1 at localhost:8810/oedbg/debugger, Connected Agent [ _mproapsv 16804 ]

Session [ ID : 7 ] (idle)
Session [ ID : 4 ] (idle)
Recap

- Basics of PDS debugger
- How to debug
  - Client
  - Unit testcases
  - Servers
Part2: Profiling

Premature optimization is the root of all evil

- Donald Knuth

[Source](#)
Part2: Profiling Agenda

- What is profiling?
- Types of profiling
- ABL Profiler introduction
- Profiler tools
- Ways to profile an application
  - Progress developer studio
  - -profile parameter
  - PROFILER handle
- Other tools
What is profiling?

“Profiling… is the investigation of programs behavior using information gathered as the program executes. The usual purpose of this analysis is to determine which sections of a program to **optimize** – to increase its overall *speed*, decrease its *memory consumption* or sometimes both.”

Profile → Optimize

Profile → Optimize
Without proper tools, profiling is like...
Looking for needle in haystack!
Types of Profiling

- CPU
  - Time/Calls
  - Execution tracing

- Memory
  - Usage
  - Allocation
ABL Profiler - Introduction

- Introduced in 9.0 (-profile)
- Execution time of modules with micro second precision
- Complete tracing information of entire session
- PROFILER handle: to focus on particular modules
- Per-Line execution times
ABL Profiler (User flow)

Enable profiling
- Enable profiling for the session
  - -profile
  - PROFILER handle

Output Generation
- Exit the session
- Profiler generates the profiler output

Analyze
- Analyze profiler output
Sample profiler output file

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>User</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/30/2015</td>
<td>15:46:56</td>
<td>srireddy</td>
<td>C:\Users\srireddy\Progress\Developer Studio 4.3.1\workspace3\Demo2\custom.p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;C:\Users\srireddy\Progress\Developer Studio 4.3.1\workspace3\ProfilerOut\listing\bg_ClassA_01a0397&quot;</td>
</tr>
</tbody>
</table>

```
1 3 2 1
0 0 1 1
1 5 3 1

0 0 1 0.000000 0.003500 0.000000 0.003500
1 0 1 0.000040 0.003500
1 3 1 0.000221 0.000250
2 0 1 0.000029 0.000029
3 0 1 0.001631 0.003193
3 2 1 0.000017 0.003210
3 2 7 0.000023 0.008289
3 2 9 0.000000 0.008291
3 2 7 0.000001 0.008291
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
3 2 7 0.000000 0.008292
3 2 9 0.000000 0.008292
```
Profiler tools

Profiler control tool

PDS Profiler
### PDSOE Profiler (11.6)

#### Module Details

**Execution time of modules**

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Times Called</th>
<th>Avg Time Per Call (sec)</th>
<th>Total Time (sec)</th>
<th>% of Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Regex&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>preProcessCustomerDetails</td>
<td>1</td>
<td>0.254157</td>
<td>0.254157</td>
<td>95.8960</td>
</tr>
<tr>
<td>C:sireddy\ProfileWS\profiler\com\progress\DB\Access</td>
<td>1</td>
<td>0.005486</td>
<td>0.005486</td>
<td>2.0699</td>
</tr>
<tr>
<td>com.progress\DB\Access</td>
<td>1</td>
<td>0.003428</td>
<td>0.003428</td>
<td>1.2934</td>
</tr>
<tr>
<td>doProcessing com.progress\DB\Access</td>
<td>100</td>
<td>0.0000013</td>
<td>0.000013</td>
<td>0.5075</td>
</tr>
<tr>
<td>postProcessCustData com.progress\CustomerData...</td>
<td>101</td>
<td>0.000005</td>
<td>0.000005</td>
<td>0.2070</td>
</tr>
<tr>
<td>Total time taken by the session (sec): 0.265034</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Calling and Called modules for “C:sireddy\ProfileWS\profiler\com\progress\StartProcedure.p”**

**Calling Modules**

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Times Called</th>
<th>% of Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Regex&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>1</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>

**Called Modules**

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Times Called</th>
<th>% of Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Regex&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>preProcessCustomerDetails com.progress\DB\Access</td>
<td>1</td>
<td>95.8960</td>
</tr>
<tr>
<td>postProcessCustDate com.progress\DB\Access</td>
<td>101</td>
<td>2.0699</td>
</tr>
<tr>
<td>doProcessing com.progress\DB\Access</td>
<td>100</td>
<td>1.2934</td>
</tr>
<tr>
<td>com.progress\DB\Access</td>
<td>1</td>
<td>0.5075</td>
</tr>
<tr>
<td>com.unprocessCustomerDataProcessor</td>
<td>2</td>
<td>0.0257</td>
</tr>
</tbody>
</table>

---

© 2017 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.
What does PDSOE Profiler tell you?

- where is the time being spent?
  - which method takes the most time?
  - which method is called the most?

- Program flow ("hierarchical" profiling)
  - do calls to method A cause method B to take too much time?

- Per-line information
  - which line(s) in a given method are the most expensive?
Demo
Words of caution

- Sources are needed for debug listing file generation
- All line numbers are debug listing line numbers
- "Enabling tracing" might generate very large output
  - Especially in app server environments
- Profiling for longer durations can also generate large output
-profile startup parameter

- Profiles the entire session
- Run the session with -profile startup parameter
  -profile config.txt
- -profile takes a configuration file as input
PROFILER handle

- Useful for focussed profiling
- Simple example:

```plaintext
//Enable profiling
ASSIGN
    PROFILER:DESCRIPTION = 'hello profiler'
    PROFILER:FILE-NAME = '/ProfilerDir/profiler.prof'
    PROFILER:LISTINGS = YES
    PROFILER:ENABLED = YES
    PROFILER:PROFILING = YES
.

obj:myCriticalMethod().

//Disable profiling
ASSIGN
    PROFILER:ENABLED = NO
    PROFILER:PROFILING = NO
.

//Write data to output file
PROFILER:WRITE-DATA()
```
Other tools

- Memory Profiling
  - DynObjects log entry type
  - Helps you find memory leaks caused by ABL dynamic objects.

- Execution tracing
  - Prospy plus
  - Log tracing (4GLTrace)

- Manual Profiling
  - etime() mtime()
Conclusion
Questions?