Performance Log Framework Technical Design Spec

Technical Design Spec

Change Record

Date	Author	Version	Description
2015-02-27	Tim Ke	1.0	Initial Draft
2015-03-11	Tim Ke	1.1	Updated package names for the APIs.
2016-07-06	James Liu	1.2	Add fields description for resource monitor
2016-09-22	James Liu	1.3	Add web.xml Servlet configuration
2016-10-10	James Liu	1.4	Add total SQL
2017-07-19	James Liu	1.5	Add http response code and session id
2018-05-28	James Liu	1.6	Add Bad SQL description
2018-06-30	James Liu	1.7	Add module page info to Request level, Add Global ID description
2018-09-20	James Liu	1.8	Add Key Value Pair format
2018-10-09	James Liu	1.9	Add LogStash sample configuration for Key Value Pair format
2018-10-31	James Liu	1.10	Add JDBC detail in Call Stack
2018-11-05	James Liu	1.11	Add Splunk Query sample when both key value pair and fix position data exist
2018-11-07	James Liu	1.12	Add PerfLog customized API interface description
2018-11.21	James Liu	1.13	Add Factory Type for PerfEventFactory usage, so we could know whether it is Job or other module
2018-12.24	James Liu	1.14	Add Request DB correlation feature
2019-5.27	James Liu	1.15	Change event logic for PerfLog Stateless change
2019-6-21	James Liu	1.16	Add Sub thread Call Stack
2019-08-01	Wilson Deng	1.17	Add customized information into PerfLog
2019-09-19	Frances Wang	1.18	Add the flag CallStackLevel into PerfLog
2019-10-18	James Liu	1.19	Add event for Page Performance measurement
2019-10-18	James Liu	1.20	Output module name for call stack node
2020-01-02	James Liu	1.21	User Behavior Analysis based on PerfLog
2020-02-17	James Liu	1.22	Event valid check
2020-02-18	Wilson Deng	1.23	Remove partial bad sql pattern(JET-7867)
2020-03-18	Wilson Deng	1.24	Add hash session id(PPF-209)
2020-03-25	James Liu	1.25	Call stack function self time, GID generating enhance, EETC for initial loading page
2020-04-14	James Liu	1.26	IUID added, UID, UN and PUID may be hash value based on DC
2020-08-14	James Liu	1.27	Add PerfLogAround for easy PerfLog usage, add PerfLogExtendedProxy for instance that don't have interface defined
2020-09-17	James Liu	1.28	Add Monitor Shared Library
2021-01-13	James Liu	1.29	Update Event valid check logic, DT field description.
2021-01-13	James Liu	1.30	Add RQC for PAGELOADING and SUBEVENT
2022-06-06	James Liu	1.31	Update for SAP Machine 11 upgrade

Table of Contents

1 Introduction

SuccessFactors has more and more users and modules. It becomes more challenging to monitor or diagnose our application's performance. Performance Log Framework is designed for following purposes:

1. Collect application performance metrics in production.
2. Find top slowness and guide performance optimization work efficiently.
3. Help to diagnose performance issues. The performance log will have multiple levels and have some detail info to help troubleshooting.

Performance team initiates this project to provide above features to all modules. Once the framework is complete, individual modules will be able to leverage this framework to monitor, diagnose and improve their performance. This framework won't affect the functions of each module, and it is inaccessible for customers. It is purely an internal framework.

2 Functional Specification

The Performance Log framework is designed to monitor overall application performance and investigation on specific performance issues. It offers two levels of logs: event and request.

- Event: an event in PerfLog is the unit for a user or non-user triggered activity. For example, user making a login, user opening a specific form, etc.
- Request: a request is the unit for communication between server and browser. An event could trigger multiple requests.

Perflog data now have two kind of data format, which is controlled by sf.sfv4.perflog.keypair.enabled parameter, if it is false, perflog data will be generated as Fix Position format, if it is true, perflog data will be generated as Key Value pair format, for key value pair format, to reduce log size, we abbreviated keys for all fields, key definition could be find in 2.4. By default, the parameter is false.

2.1 Event Log field definition

An event log entry contains the attributes below:

1. Time
2. Client IP
3. User Context: company id, company name, schema name, db pool name, user id, user name, locale
4. Event Name

5. Event Id
6. Render Time
7. Server Time
8. End2End time
9. # of JS resource on the page
10. # of CSS resource on the page
11. # of Ajax requests sent out from client in the event
12. Call Id
13. Module Id
14. Page Id
15. Page Qualifier
16. Consumed memory
17. Total CPU time
18. User CPU time
19. System CPU time
20. File bytes read from file system
21. File bytes written to the file system
22. Network bytes read
23. Network bytes written
24. Opened file but not closed
25. Opened socked but not closed
26. Total SQL invoking #
27. Total SQL invoking time
28. Time to Page Render Start
29. Time to Interactive Page
30. Time to Main Content Load
31. Time to Supplemental Content Load

Event level log with fix position format:

```
--Log that does not have Phase of page loading data

2015-01-31 23:43:07,618 [50.202.74.56]  [LMSAUTOIS5,LMSAUTOIS5,qacandrot_LMSAUTOIS5.,dbPool1,admin,admin,en_US]  PLT-HOMEPAGE  EVENT-PLT-HOMEPAGE-vsa118143.lab.od.sap.biz -20150131234254-36622 1988   4200 12789  145 26       9116334707-0 HOME HOME_TAB - [[89MB 1540ms 1290ms 250ms 28KB 0KB 216KB 184KB 0 0 ]] 13 1341 - - - -

--Log that have Phase of page loading data

2017-07-19 08:42:59,156 [222.126.176.10]  [CDC12,CDC12,qacandrot_CDC12.,dbPool1,cgrant1,cgrant,en_US]  UNKNOWN-UNKNOWN EVENT-UNKNOWN-UNKNOWN-vsa118143.lab.od.sap.biz-20170719084257-67700 1320 943 1490 137 18 [Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36]     9655987228-3 EMPLOYEE_FILE EMPLOYEE_FILE PP3_MAIN [[77915KB 240ms 220ms 20ms 48KB 0KB 80KB 71KB 0 0 ]] 38 501 7278 7878 11119 11308
```

Event level log with key value pair format:

```
2018-09-14 13:53:45,714                PLV=EVENT CIP=222.126.176.10 CMID=BizXTest CMN="BizXTest" SN=BIZX_BizXTest. DPN=dbPool1 UID=cgrant1 UN=cgrant LOC=en_US EID=EVENT-PLT-HOMEPAGE-H211252FrtbbdrrebbsnqrFbnl-20180914135329-0014 AGN="[Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36]" RDT=11926 SVT=31445 EET=16309 JSC=125 CSSC=29 CAID=6517042289-0 MID=HOME PID=HOME_TAB PQ=HOME_V2 MEM=1480067 CPU=5160 UCPU=4300 SCPU=860 FRE=5246 FWR=0 NRE=2522 NWR=625 SQLC=686 SQLT=9326 TRS=- TIP=- TML=- TSL=-
```

2.2 Request Log field definition

And a request log entry contains the attributes below:

- Time
- Client IP
- User Context: company id, company name, schema name, db pool name, user id, user name, locale
- Event Name
- Event Id
- Request Id
- Request Method: GET/POST
- URL
- Request Server time
- Consumed memory
- Total CPU time
- User CPU time
- System CPU time
- File bytes read from file system
- File bytes written to the file system
- Network bytes read
- Network bytes written
- Opened file but not closed

- Opened socked but not closed
- Total SQL invoking #
- Total SQL invoking time
- Response status code
- Session ID : You could find there is * in the session id in below example, it comes from Security reason, we need to mask part of session. There is similar logic in API and Ngnix.
- Module Id
- Page Id
- Page Qualifier
- IsSubrequest : 0 means request from Load page, 1 means it is request that sends out after the first page load.
- Global id: it could be used to link with server log or access log.
- Parameter Names
- Call Stack

Request level log in fix position format:


```
2014-10-15 16:02:13,909 [194.39.218.10] [BOSCH,Bosch PROD,eusfv4_STOCKEUPR001.,dbPool1,10738242,CLA8FE,en_US]  UNKNOWN-UNKN
UNKNOWN-vsa218751.rot.od.sap.biz-20141015155853-691191       REQ-1 POST /xi/ajax/remoting/call/plaincall/genericObjectAutoCompleteCont
[[89MB 1540ms 1290ms 250ms 28KB 0KB 216KB 184KB 0 0 ]] 245 19034 200 6CB4A2BDECC50A244B6F5EF4A4******.pc2bsfapi03t HOME HOME
098bff2f00038c754c947ad6d0f6bdfa ] [] {"n":"/xi/ajax/remoting/call/plaincall/genericObjectAutoCompleteControllerProxy.search.dwr","i":1,"t":2
com.successfactors.genericobject.service.GetObjectDefinition","i":3,"t":1311820,"sub":[{"n":"com.successfactors.platform.dao.impl.oracle.Or
getSysConfig","i":1,"t":898620,"sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(select sys_config_key
sys_config_long_value, sys_config_string_value, null, sys_config_schema from eusfv4_STOCKEUPR001.SYS_CONFIG where sys_config_key
1,"t":701532}]}]},{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(SELECT dbms_transaction.local_transactio
7279278},{"n":"SCA:com.successfactors.genericobject.service.GetAllObjectTypes","i":1,"t":96076},{"n":"SCA:com.successfactors.genericobje
GetRBPRoleIdsByObjectTypeAndPermissions","i":2,"t":24275680,"sub":[{"n":"SCA:com.successfactors.genericobject.service.ExecuteGOSQL
sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(SELECT T0.EXTERNAL_CODE F0, T0.ROW_ID F1 FRO
GENERIC_OBJECT_T T0 left outer join eusfv4_STOCKEUPR001.GENERIC_RELATIONSHIP T2 left outer join eusfv4_STOCKEUPR001.GENERIC
OBJECT_TYPE = ?) AND (T1.OBJECT_TYPE_ID = ?) AND (T2.DESTINATION_INTERNAL_ID = T1.INTERNAL_ID) AND (T1.EFFECTIVE_STATUS =
in (?))  on  (T2.RELATIONSHIP_NAME in (?)) AND (T2.DESTINATION_OBJECT_TYPE = ?) AND (T2.SOURCE_OBJECT_TYPE = ?) AND (T0.ROW_
AND (T0.ROW_ID = T1.PARENT_ROW_ID)  WHERE  (T0.OBJECT_TYPE_ID = ?) AND (T0.OBJECT_TYPE = ?) AND (T0.EFFECTIVE_STATUS = ?)
(?)) AND (T1.SF_FIELD3 = ?) AND (T1.EXTERNAL_CODE = ?) )","i":2,"t":2515001}]},{"n":"com.successfactors.platform.dao.impl.oracle.OracleS
1,"t":1391231,"sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(select sys_config_key, sys_config_type
sys_config_string_value, null, sys_config_schema from eusfv4_STOCKEUPR001.SYS_CONFIG where sys_config_key=? and sys_config_type=
SCA:com.successfactors.legacy.service.GetCustomLanguagePack","i":1,"t":907311,"sub":[{"n":"com.successfactors.platform.dao.impl.oracle
getSysConfig","i":1,"t":796354,"sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(select sys_config_key
sys_config_long_value, sys_config_string_value, null, sys_config_schema from eusfv4_STOCKEUPR001.SYS_CONFIG where sys_config_key
1,"t":697037}]}]},{"n":"SCA:com.successfactors.legacy.service.GetSysConfigList","i":1,"t":225232,"sub":[{"n":"SCA:com.successfactors.legac
GetSysConfigListV2","i":1,"t":94337,"sub":[{"n":"com.successfactors.platform.dao.impl.oracle.OracleSystemDAO.getSysConfigList","i":1,"t":
successfactors.legacy.service.GetSysConfig","i":1,"t":111804,"sub":[{"n":"SCA:com.successfactors.legacy.service.GetSysConfigV2","i":1,"t":
successfactors.rbp.app.api.impl.RBPFacadeImpl.getRoleIdsRuleIdsMap","i":2,"t":9776432,"sub":[{"n":"SCA:com.successfactors.rbp.service.G
i":2,"t":6733651,"sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(select rule_id, role_id, access_group_
target_group_level, my_filter, exclude_login_user,\n relation_role, status, include_self\n FROM eusfv4_STOCKEUPR001.RBP_PERM_RULE r w
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,
and r.rule_id in (select rule_id  FROM eusfv4_STOCKEUPR001.RBP_RULE_AGUSR_MAP where users_sys_id = ?)\n  and status=1 \n order by r
2666324}]},{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(SELECT dbms_transaction.local_transaction_id F
{"n":"SCA:com.successfactors.genericobject.service.rbp.GetTCMapByRuleIdsAndObjectType","i":2,"t":4778396,"sub":[{"n":"com.successfact
OracleSystemDAO.getSysConfig","i":6,"t":4318495,"sub":[{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(se
sys_config_type, sys_config_long_value, sys_config_string_value, null, sys_config_schema from eusfv4_STOCKEUPR001.SYS_CONFIG wher
sys_config_type=?)","i":6,"t":4010290}]}]},{"n":"SCA:com.successfactors.genericobject.service.ExecuteGOSQLQuery","i":2,"t":200528664431,
resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(SELECT * FROM (   SELECT a.*, ROWNUM r FROM ( SELECT T0.EXTERNAL_
SF_FIELD1 F2, T0.INTERNAL_ID F3 FROM eusfv4_STOCKEUPR001.GENERIC_OBJECT_T T0 left outer join eusfv4_STOCKEUPR001.GENERIC_
OBJECT_TYPE = ?) AND (T1.OBJECT_TYPE_ID = ?) AND (T0.ROW_ID = T1.PARENT_ROW_ID) AND (T0.EXTERNAL_NAME = TO_CHAR(T1.INTE
EFFECTIVE_STATUS = ?) AND (T1.RECORD_STATUS in (?))  WHERE  ((T0.EFFECTIVE_START_DATE <= TO_DATE(?,'YYYY-MM-DD')) AND (T0.
TO_DATE(?,'YYYY-MM-DD'))) AND (T0.OBJECT_TYPE_ID = ?) AND (T0.OBJECT_TYPE = ?) AND (T0.EFFECTIVE_STATUS = ?) AND (T0.RECOR
(T0.EXTERNAL_CODE||','||T1.SF_FIELD2||','||T1.SF_FIELD1) LIKE ?) ORDER BY UPPER(COALESCE(T1.SF_FIELD1,T1.SF_FIELD2)) ASC,T0.INTI
ROWNUM <= ?)  WHERE  r >= ? )","i":1,"t":90752762840},{"n":"org.jboss.resource.adapter.jdbc.WrappedPreparedStatement.executeQuery(SEL
ROWNUM r FROM ( SELECT count(1) F0 FROM eusfv4_STOCKEUPR001.GENERIC_OBJECT_T T0 left outer join eusfv4_STOCKEUPR001.GENE
OBJECT_TYPE = ?) AND (T1.OBJECT_TYPE_ID = ?) AND (T0.ROW_ID = T1.PARENT_ROW_ID) AND (T0.EXTERNAL_NAME = TO_CHAR(T1.INTE
EFFECTIVE_STATUS = ?) AND (T1.RECORD_STATUS in (?))  WHERE  ((T0.EFFECTIVE_START_DATE <= TO_DATE(?,'YYYY-MM-DD')) AND (T0.
TO_DATE(?,'YYYY-MM-DD'))) AND (T0.OBJECT_TYPE_ID = ?) AND (T0.OBJECT_TYPE = ?) AND (T0.EFFECTIVE_STATUS = ?) AND (T0.RECOR
(T0.EXTERNAL_CODE||','||T1.SF_FIELD2||','||T1.SF_FIELD1) LIKE ?) ORDER BY UPPER(COALESCE(T1.SF_FIELD1,T1.SF_FIELD2)) ASC,T0.INTI
ROWNUM <= ?)  WHERE  r >= ? )","i":1,"t":109773846170}]}]}
```


Request level log in key value pair format:

2018-09-14 13:51:26,099 PLV=REQ CIP=0:0:0:0:0:0:0:1 CMID=BizXTest CMN="BizXTest" SN=BIZX_BizXTest. DPN=dbPool1 UID=cgrant1 UN=cgrant LOC=en_US EID=EVENT-UNKNOWN-UNKNOWN-H211252FrtbbdrrebbsnqrFbnl-20180914134759-0008 AGN="[Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36]" RID=REQ-[204] MTD=GET URL="/sf/teamoverviewpage" RQT=10912 MID=PERFORMANCE PID=PM2_TEAMOVERVIEW PQ=- SUB=0 MEM=89036 CPU=350 UCPU=290 SCPU=60 FRE=13 FWR=0 NRE=35 NWR=365 SQLC=78 SQLT=411 RPS=200 SID=8DE68088E5A2AF56BE347C0F26**** GID=- PN="[]" STK={"n":"/sf/teamoverviewpage","i":1,"t":10911,"sub":[{"n":"SCA:com.successfactors.homepage.service.GetDefaultSubTab","i":14,"t":225,"sub":[{"n":"SCA:com.successfactors.user.service.preference.GetPreferenceBean","i":14,"t":168,"sub":[{"n":"com.successfactors.user.dao.impl.oracle.OraclePreferencesDAO.getPreferenceBean","i":14,"t":122,"sub":[{"n":"SQL:SELECT lookup_key, lookup_value, type FROM BIZX_BizXTest.ext_profile_inf WHERE users_sys_id = ? AND category = ? AND lookup_key =? and source = ? ","i":14,"t":105}]}]}]}]}**

2.3 Bad sql check logic

Some check was taken to the SQL we gathered to identify bad sql, it includes the following:

1. ~~If the sql contains delete from (TMP_BA_COND|TMP_FB_COND|TMP_EC_COND|TMP_FB_VALID_TO|TMP_BA_COND_DEPTH|TMP_TS_SEARCH|TMP_TS_DATA|TMP_COMPOUND_API|TMP_API_EXPAND), it will be marked as /* BAD_SQL_DELETE_TEMP */~~
   Delete from temp table would block us converting row-store TMP_BA_COND to column-store.
2. If the sql contains "IFNULL", it will be marked as /* BAD_SQL_IFNULL */
   IFNULL function couldn't be handled efficiently by HANA optimizer, which could be rewrite by 2 filters connected by OR
3. If the sql contains "NULL FIRST" or "NULL LAST", it will be marked as /* BAD_SQL_NULL_ORDER */
   Since many queries in our system was converted to HANA version by SQL converter, sometimes NULLS FIRST/LAST automatically appended in the end of query, but certain columns to be ordered isn't nullable, with that option would cause query run into row engine then slow down the execution.
4. ~~If the sql used the following view twice, it will be marked as /* BAD_SQL_TWICE_HIERARCHY_VIEW */ "GENERIC_OBJECT_T_ROWID_HVIEW","GENERIC_OBJECT_T_ROWID_HRVIEW","GO_OBJECT_TYPE_INTERNAL_CODE_HVIEW", "GO_OBJECT_TYPE_EXTERNAL_CODE_HVIEW","WF_REQUEST_HVIEW","OBJECTIVE_HVIEW","ROLE_HVIEW","PICKLIST_HVIEW", "FORM_TEMPLATE_HVIEW","POS_MODEL_DATA_HVIEW","GENERIC_OBJECT_T_HVIEW","USERS_SYSINFO_HVIEW","USERS_SYSINFO_CYCLE_HVIEW", "USERS_SECOND_MANAGER_HVIEW","USERS_SECOND_MANAGER_CYCLE_HVIEW","GENERIC_OBJECT_T_AUDIT_ROWID_HRVIEW"~~
   Reduce HView access in one query could improve query performance
5. If the sql use literal directly, it will be marked as /* BAD_SQL_WITH_LITERAL */
   You can use bind variable to rewrite such pattern.

Below is one example perflog output when this feature is enabled and bad pattern found, you could see there is /* **BAD_SQL_WITH_LITERAL** */ appended in the sql.

2020-05-19 06:04:24,433 PLV=REQ CIP=169.145.2.13 CMID=CQAIASMIG1 CMN="CQAIASMIG1" SN=QACANDROT_CQAIASMIG1. DPN=dbPool2 UID=5CBA6B9A8875B5 UN=E205F8D2C5CB83 IUID=91 LOC=en_US EID="EVENT-UNKNOWN-UNKNOWN-InFeaab7bed5-20200519060424-3939" AGN="[Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko]" RID=REQ-[3206] MTD=GET URL="/dpconsent" RQT=280 MID=- PID=- PQ=- SUB=- MEM=36775 CPU=170 UCPU=170 SCPU=0 FRE=0 FWR=0 NRE=36 NWR=215 SQLC=9 SQLT=66 RPS=302 SID=B4B8F409C3610BF79C4E8FC12A****.mo-fbba8afe6 GID=846ff5d8e2ff5e6325077812ebae1164 PN="[]" HSID=8fe4739949372874c77c57dcd7e1daccf57c9bd31dac75da34a364daaedc247f CSL=FULL STK={"n":"/dpconsent","i":1,"t":280,"slft":67,"sub":[{"n":"SCA:com.successfactors.legacy.service.GetSysConfig","i":31,"t":19,"slft":15,"q":0,"qq":0,"u":0,"uu":0,"m":1101,"sub":[]},{"n":"SCA:com.successfactors.provisioning.api.servicelocal.ProvisioningServiceLocalCall","i":21,"t":58,"slft":50,"q":0,"qq":0,"u":0,"uu":0,"m":18143,"sub":[]},{"n":"SCA:com.successfactors.user.service.FindUserByName","i":1,"t":27,"slft":12,"q":2,"qq":2,"u":0,"uu":0,"m":2534,"sub":[{"n":"com.successfactors.usermanagement.legacy.dao.impl.UserLegacyDaoImpl.findUserByName","i":1,"t":15,"slft":7,"sub":[]}]},{"n":"SCA:com.successfactors.dpcs.service.GetDpcsCurrentVersionsCmd","i":1,"t":42,"slft":18,"q":2,"qq":2,"u":0,"uu":0,"m":2751,"sub":[{"n":"SQL:select * from QACANDROT_CQAIASMIG1.DPCS a left join QACANDROT_CQAIASMIG1.DPCS_VERSION b on a.CURRENT_VERSION = b.DPCS_VERSION_ID where a.DPCS_STATUS =#/* BAD_SQL_WITH_LITERAL */","i":1,"t":17,"slft":17,"st":17,"m":371,"nr":0,"rt":0,"rn":1,"fs":0}]},{"n":"SCA:com.successfactors.dpcs.service.GetDefaultDpcsCmd","i":1,"t":53,"slft":22,"q":4,"qq":4,"u":0,"uu":0,"m":4154,"sub":[{"n":"SQL:select * from QACANDROT_CQAIASMIG1.DPCS where DPCS_STATUS =#/* BAD_SQL_WITH_LITERAL */","i":1,"t":2,"slft":2,"st":41,"m":2307,"nr":25,"rt":0,"rn":1,"fs":0},{"n":"SQL:select * from QACANDROT_CQAIASMIG1.DPCS_CONTENT where DPCS_VERSION_ID = ?","i":1,"t":19,"slft":19,"st":19,"m":817,"nr":21,"rt":0,"rn":11,"fs":0}]}]}**

2.4 Key Value Pair key definition

Java Source Code Reference

- https://github.wdf.sap.corp/bizx/idl-perflogbase/blob/master/idl-perflogbase-service/src/main/java/com/successfactors/perflog/PerfLogField.java

1. PLV: Perf Level: EVENT, REQ or FACTORY (the log comes from PerfEventFactory)
2. FT: Factory Type, Sub type for PLV FACTORY, e.g. S2JOB
3. DT: Event Data Type, more detail in sub section.
4. CIP: Client IP
5. CMID: company id
6. CMN: company name

7. SN: schema name
8. DPN: db pool name
9. UID: user id
10. UN: user name
11. IUID: internal user id
12. LOC: locale
13. PUID: Proxy user id
14. EID: Event Id
15. AGN: Agent
16. RID: Request Id
17. MTD: Request Method
18. URL: URL
19. RQT: Request execution time [ms]
20. RDT: Render Time [ms]
21. SVT: Server Time [ms]
22. EET: End2End Time [ms]
23. EETC: End2End Time [ms], this one only exists in the initial loading page, start time is from timing api, navigation start, which means this one includes network time.
24. JSC: # of JS resource on the page
25. CSSC: # of CSS resource on the page
26. RQC: # of Ajax requests sent out from client in the event
27. CAID: Call Id
28. MID: Module Id
29. PID: Page Id
30. PQ: Page Qualifier
31. ACT: Page Action
32. SUB: 0 means request from Load page, 1 means it is request that sends out after the first page load
33. MEM: Consumed memory [KB]
34. CPU: Total CPU time [ms]
35. UCPU: User CPU time [ms]
36. SCPU: System CPU time [ms]
37. FRE: File bytes read from file system [KB]
38. FWR: File bytes written to the file system [KB]
39. NRE: Network bytes read [KB]
40. NWR: Network bytes written [KB]
41. SQLC: Total SQL invoking #
42. SQLT: Total SQL invoking time [ms]
43. RPS: Http Response status code
44. SID: Session ID, for security reason, part it SID is masked by *
45. HSID: Hash Session ID based on real Session ID.
46. GID: Global id, this id exists in access log, server log and perflog, it could be used to correlate them.
    If request comes though Nginx, GID will be generated by Nginx, and it could be found in the 3 log files., if request does not come though Nginx, PerfLog will generate GID, and it could be found in server log and PerfLog.PN: Parameter Names, if the GID is generated by PerfLog, it will start with GENR-
47. TRS: Time to Page Render Start [ms]
48. TIP: Time to Interactive Page [ms]
49. TML: Time to Main Content Load [ms]
50. TSL: Time to Supplemental Content Load [ms]
51. ST: start time.
52. ET: end time.
53. TTB: time to first byte. [ms]
54. RED: Redirect count
55. RSR: Render Start Time
56. DIA: DOM Interactive
57. DCP: DOM Complete
58. CSL:Call Stack Level
59. CCON: Total CONCURRENT Cache Time
60. CSUP: Total SUPERSFEDIS Cache Time
61. CLOC: Total LOCALSFEDIS Cache Time
62. CEXT: Total EXTREMESFEDIS Cache Time
63. CREM: Total REMOTESFEDIS Cache Time
64. MAZID: Swimlane and AZ information from Consul agent Service, refer PPF-264 for more detail.
65. RTT: Response Total Time - This is useful when used with the PerfPhases so that we can derive the UI-focused timings (window.performance.timing.responseEnd - window.performance.timing.navigationStart)
66. CRT - Client Request time, this can be thought of as the "network time" (window.performance.timing.responseStart - window.performance.timing.navigationStart)
67. SRT - Server Response Time, this can be thought of as the "server-side time" (window.performance.timing.responseEnd, window.performance.timing.responseStart)
68. UTML - UI-normalized time to main content load, this can be thought of as the "client render time" (TML - RTT)
69. VER: perfLog.js version, 1 is old version, 2 is new version from verp

2.4.1 Extention Fields

BizX S2Job has integration with PerfLog, they have some specific fields only for S2Job, FT field for S2Job is S2JOB.

- s2_type_name: Job type name
- s2_exec_id: Job Execution ID
- s2_job_name: Job name

2.4.2 User Information Fields

UID, UN, and PUID are fields that are related to the user who logged in. For security reason, we can not show them in PerfLog in some DC. So for DC that has security request, hash value for them will be stored in PerfLog, for DC without security request, PerfLog will still show original value.

So if someone need to search Splunk for some specific user, they could use below link to get hash value for the user name he knows.

https://cloudsearch-dc13.cld.ondemand.com/en-US/app/search/log_security_tool

For another scenario, if someone need to get user related information from Perflog, IUID in PerfLog could be used, it is Internal User ID. Customer could get real user name based this IUID.

2.4.3 DT Data Type Field

DT field is only available for Event record, its value is used to identify different PerfLog Event, to have better understanding about how EET is calculated for PAGEPERFORMANCE and PAGELOADING, you can check PerfLog E2E time.

it has below values:

- PAGEPERFORMANCE: Event generated for Page Performance, this value highly depends on DOM readyState or TML.
  - PLV=EVENT CAID=*-X
- MANUALEVENT: This is user behavior analysis, detail in 2.8UserBehaviorAnalysisbasedonPerfLog.
  - PLV=EVENT CAID=*-Y
- PAGELOADING: To identify initial page loading request, which could be used by Ops dashboard, other values should not be used to do page performance checking.
  - PLV=EVENT CAID=*-0
- SUBEVENT: To identify event happened after initial page loading on the same page without whole page refresh.
  - PLV=EVENT CAID!=*-X CAID!=*-Y CAID!=*-0
- LONGEET: If the event EET or EETC is longer than 30mins, we will use this tag, further investigation on these data needed.
  - PLV=EVENT AND (EET>30*60*1000 OR EETC>30*60*1000)

You can find data characteristics for each data type, but in splunk search, just use DT= is enough to filter out correct data.

e.g. To check page performance, you just need to get data using DT=PAGELOADING

2.4.3 Change From SAP Machine 11

Bizx will upgrade to use SAP Machine 11 from b2211, with SAP Machine 11, original SAP Monitoring API is not available anymore.

It will impacts below fields:

- FRE
- FWR
- NRE
- NWR

From b2211, these fields value will be "-", which means not available.

2.5 Call Stack Field Definition

In general, you will see n, i, t for most data in Call Stack.

Below is definition for them:

- n: Name for the call node to identify which function is execution, it could be SCA, JDBC, SQL and any other name defined by developer
- i: Total invoke times for current node.
- t: Total invoke time for current call node. [ms]
- slft: execution time taken by the function itself. slft = t - sum(sub.t)

We added below field for SCA node to show more detail information.

Note: SCA could invoke another SCA, for q, qq, u, uu, it only reflect SQL executed in the SCA itself, not includes SQL executed in sub SCA.

- q: Total distinct SQL query count for current call node.
- qq: Total SQL query count for current call node.
- u: Total distinct SQL update count for current call node.
- uu: Total SQL update count for current call node.
- m: Memory consumption of the Thread for current call node [KB]

PerfLog has two kinds of call stack. by default.

- Full Call Stack
  - when request execution time is longer than 10 seconds, whole printable call stack will be outputted.
- Critical Call Stack
  - When request execution time is longer than 1 second, and less than 10 seconds, only critical call stack will be outputted.
  - Critical call stack mean PerfLog will select node with longest execution time for each level and output them, you will see parent has only one child in this call stack.

When request execution time is less than 1 second, no call stack will be outputted, but 1 second, 10 seconds are configurable.

To know the call stack type, you could find CSL field in PerfLog record, it helps end-user to understand what level of details he can expect from PerfLog STK.

- CRITICAL - it is critical call stack.
- FULL - full call stack
- NONE - no call stack

To identify function ownership, now module name for the node is printed out in CallStack.

- mn: module name of the class.

this feature will need module team's change to their classes by adding some annotation, detail could be found in How to add module type to module's own class.

2.5.1 JDBC Detail in Call Stack

To get more idea for JDBC execution, we tracked extra information for Statement and ResultSet execution. Normal entry in Call Stack only have n, i, t 3 fields, for Query SQL, we added more feilds, below is one example:

{"n":"SQL:select sys_config_key, sys_config_type, sys_config_long_value, sys_config_string_value, null, sys_config_schema from BIZX_BizXTest.SYS_CONFIG where sys_config_key=? and sys_config_type=?","i":1,"t":82,"st":99,"m":857,"rt":13,"rn":67,"fs":10}

You could see following new fields:

- st: Another time tracking for sql execution, it starts from statement.execute, ends when statement.close is invoked. [ms]
- m: Memory consumption of the Thread. it starts from statement.execute, ends when statement.close is invoked. [KB]
- nr: Network read for the Thread. it starts from statement.execute, ends when statement.close is invoked. [KB], from b2211, this value will be set to value 0 because of JVM upgrade.
- rt: Accumulated time for rs.next() execution. [ms]. Time tracking logic is removed in b2211, since in Hana this field does not have obvious help, to keep format compatible, in Perflog output, the field is still there , but value will be 0.
- rn: Result number for the query. We get this by counting execution # of rs.next().
- fs: Fetch Size for the query, this field is introduced when BizX is still using Oracle, now it is all Hana, and in Hana, most time, fetch size is not configured, also to avoid necessary Exception, we manually set this field to value 0.

2.5.2 Multiple Thread Call Stack

Now PerfLog call stack could also show call stack for sub thread which is started by main thread, we don't support thread started by sub thread unless there is clear requirement.

For each sub thread, it will be one child node for root node, and node name starts with THREAD:, following name is the class name of the Callable or Work. If two threads have same node name, they will be merged.

For root node of thread, one more field is added.

- mt: Max execution time for sub threads.

To add sub thread call stack, some code change will be needed.

Now PerfLog already enabled the change in com.successfactors.apm.app.api.AsyncWorkManagerFactory and com.successfactors.jobscheduler.jobimpl.ConcurrentJobProcessor , if you are using other basic thread pool, similar change will be needed.

Below is sample code change for one method, there is three method invoking, first one is create sub call stack and create relationship with main thread, the second one is tart first node for the call stack, last one is end first call for the stack.

```
public <T> Future<T> submit(Callable<T> callable, WorkListener listener) {
final Callable<T> c = callable;
final CallStack subCallStack = CallStack.createSubCallStack(c.getClass().getName() );


return executor.submit(new Callable<T>() {

@Override
public T call() throws Exception {
try {
APMTaskUtils.preExecution();
CallStack.startSubThreadCall(subCallStack, c.getClass().getName());
return c.call();
} catch (Exception e) {
log.error("Hit Exception to run APM task {}", e, c.getClass().getCanonicalName());
throw e;
} finally {
CallStack.endSubThreadCall();
APMTaskUtils.postExecution(c.getClass());
}
}
});
}
```

2.5.3 Limitation of SLFT

PerfLog call stack only show function that is tracked by PerfLog and SLFT in PerfLog is not really self-time for the function but self-time + time on unintercepted sub calls. If parent function has long time consuming sub function which is not tracked by PerfLog, based on PerfLog slft, you may think the function itself takes too much time, so when you find some function has big slft, you need to double confirm in source code too.

2.6 Request DB correlation

PerfLog has Event ID, Request ID and Global ID stored in log data, Global ID is also there in Server log and Access Log.

This feature will store Global ID/Event ID and Request ID into related DB table, so when you want to know which UI request or Job execution is related to the SQL you are checking, you could get Global ID/Event ID and Request ID from table and based on that get PerfLog, Server Log and Access Log.

2.6.1 Hana

In Hana DB, when Expensive Statements Trace enabled, based on threshold set there, when query execution time is longer than that, you could query m_expensive_statements table to get expensive sql.

In that table, you could find application_source column,

For normal UI request:

- If Global ID exists, application_source column will store Global ID and text "GlobalID", separated by " : "
- If Global ID not exists, application_source column will store Event ID and Request ID, separated by " : "

For S2 Job Execution:

- application_source column will store Event ID and text "FACTORY", separated by " : ", since event id is unique for S2 Job PerfLog, you just need Event ID when you do the query in splunk.

Below first image, application_source is "XCM2qwoEG8AAAGlhl@AAAABG : GlobalID", so the Global ID is XCM2qwoEG8AAAGlhl@AAAABG.

Below second image, application_source is "-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32] ", so the part of event id is -H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008, and request id is REQ-[32].

Based on these, you could check further detail in splunk, detail step show in below.

There are also other tables that contains similar field, e.g. M_SERVICE_THREADS will be filled by this correlation too, that view could be used in live checking and troubleshooting, m_service_thread_samples is historical table for M_SERVICE_THREADS.

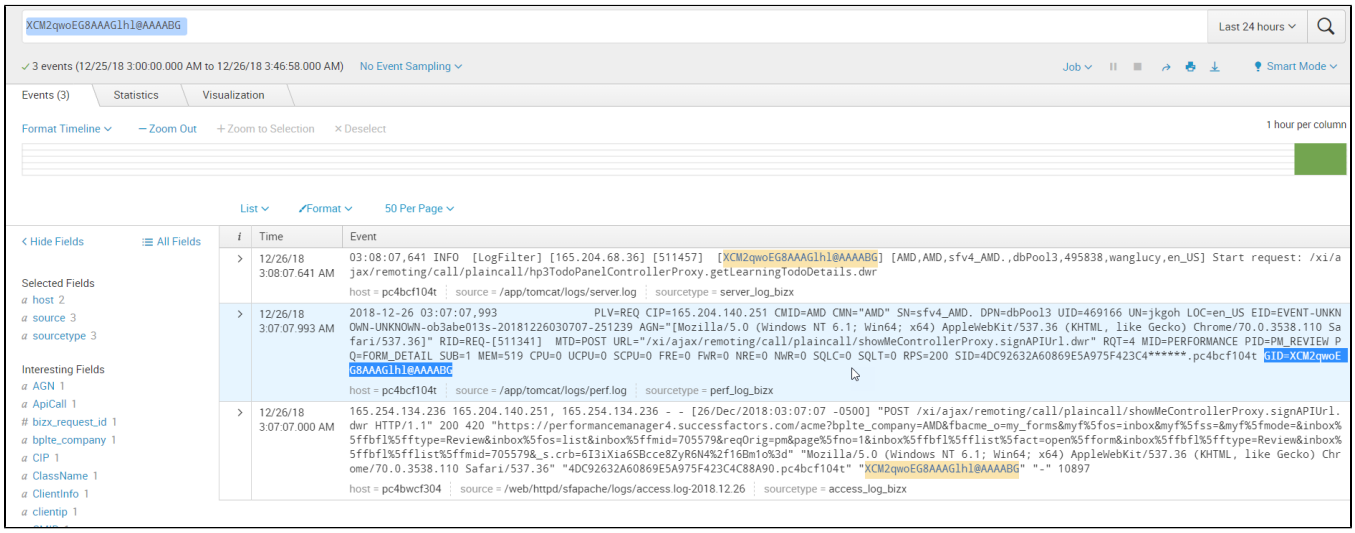
Sample application_source value in Hana db(Global ID)

T_STATEMENT_ID	STATEMENT_HASH	DB_USER	SCHEMA_NAME	APP_USER	START_TIME	DURATION_MICROSEC	OB
	76f246be96028db1e4ac3997b6a07373	SYSTEM	SYSTEM		Jan 16, 2019 5:55:16.490631 AM	1,047	M_
	76f246be96028db1e4ac3997b6a07373	SYSTEM	SYSTEM		Jan 16, 2019 5:55:06.419971 AM	1,535	M_
	76f246be96028db1e4ac3997b6a07373	SYSTEM	SYSTEM		Jan 16, 2019 5:55:06.419971 AM	1,454	M_
	692e184981e9c2c55eef9eadcb3a3c9b	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.711893 AM	11,327	BI
	e20651c7343a4da119687abbecf5d432	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.659956 AM	6,828	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.626631 AM	13,940	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.626631 AM	13,653	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.56756 AM	55,184	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.56756 AM	54,544	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.526849 AM	14,488	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.526849 AM	14,129	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.469165 AM	53,317	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.469165 AM	52,654	BI
	4ace42c36799b30f3d6baa265345cca8	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.405945 AM	10,376	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.346755 AM	16,936	BI
	b6287d1ec14860e0958107fff70617cf	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.346755 AM	16,475	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.247278 AM	89,234	BI
	45b2db084daba0af4478e383ce41b9...	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.247278 AM	87,828	BI
	4c98b7662a8bc3ab5ab5347fe401ee10	BIZX	BIZX	XCM2qwoEG8AAAGlhl@AAAABG : GlobalID	Jan 16, 2019 5:55:01.097943 AM	8,543	BI

Sample application_source value in Hana db(Event ID and Request ID).

T_STATEMENT_ID	STATEMENT_HASH	DB_USER	SCHEMA_NAME	APP_USER	START_TIME	DURA
	5f17209c07dbbb76d560b15f267a36ba	BIZX	BIZX	I322363	Jan 16, 2019 5:58:42.447394 AM	
	5f17209c07dbbb76d560b15f267a36ba	BIZX	BIZX	I322363	Jan 16, 2019 5:58:44.264187 AM	
	5f17209c07dbbb76d560b15f267a36ba	BIZX	BIZX	I322363	Jan 16, 2019 5:58:42.767118 AM	
	5f17209c07dbbb76d560b15f267a36ba	BIZX	BIZX	I322363	Jan 16, 2019 5:58:42.447394 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:38.49977 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:38.49977 AM	
	5b4be90a3839843dabd64d3f2390efc7	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:38.069391 AM	
	8aaa1e17d379bb576d0eb95a940014...	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.621105 AM	
	8aaa1e17d379bb576d0eb95a940014...	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.60984 AM	
	ebba894be6b0c7d0d3a240eb23771b...	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.064039 AM	
	d3204de2f9fb21de9117038cc06154ce	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.134083 AM	
	d3204de2f9fb21de9117038cc06154ce	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.134083 AM	
	d3204de2f9fb21de9117038cc06154ce	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.095629 AM	
	ebba894be6b0c7d0d3a240eb23771b...	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.064039 AM	
	ebba894be6b0c7d0d3a240eb23771b...	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:37.054051 AM	
	393ccc1e3ce97a6a50c8872e5b54eb70	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.975723 AM	
	2e7786dc663cdf003b681dd13ce306a2	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.962472 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.937574 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.937574 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.937574 AM	
	581dd60485157a51ef57eb0309931e06	BIZX	BIZX	-H211252FrtbbdrrebbsnqrFbnl-20190116135836-0008 : REQ-[32]	Jan 16, 2019 5:58:36.925671 AM	

If you have Global ID, you could just search for Global ID directly.



If you don't have Global ID, you could make search based on Event ID and Request ID

2.7 Customized Key Value Pair

This is for extend information into perflog via adding request header ***X-Customized-Info*** with key value pair in header value, perflog will parse and log it. (e.g. X-Customized-Info: key1=value1&key2=value2... in request header)

There are some rules as following to using this header:

1. The max key value pair number is **10** and this is configurable via ***sf.sfv4.perflog.max.customized.keypair.number*** , if key value pairs is more than this value setting will only keep the setting numbers k-v pair and ignored rest of it.
2. Key or value length is set to **100** characters, if more than 100 chars will be truncated the rest of it.
3. Key or value do not contains space, if has will be filtered.
4. Key or value is null or empty string will be ignored immediately.

2.8 User Behavior Analysis based on PerfLog

Now PerfLog has module, page, page qualifier to identify Event source, but even with page qualifier, it still does not provide enough detail for Event source. e.g. same page, different user action, may trigger different Ajax call, they may share same page qualifier. To provide further event source detail, PerfLog provides this feature.

1. ACT field is added in PerfLog Event.
   Module team sets Action name for their Ajax call when they initial the Ajax call, with below code, the Page Action will be set as "ActionName1"
   window.PerfLog.pageAction = "ActionName1"
   Then the Action Name will be shown in according PerfLog Event as Below, ACT=ActionName1 shows the action name.

   2020-01-02 13:27:59,494 PLV=EVENT CIP=192.168.106.2 CMID=BizXTest CMN="BizXTest" SN=BIZX_BizXTest. DPN=dbPool1 UID=cgrant1 UN=cgrant LOC=en_US EID="EVENT-PLT-HOMEPAGE-H211252-20200102132758-0012" AGN="[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36]" RDT=404 SVT=282 EET=847 EETC=852 JSC=15 CSSC=8 CAID=5414616194-1 MID=HOME PID=HOME_TAB PQ=HOME_V3 **ACT=ActionName1** MEM=18608 CPU=160 UCPU=130 SCPU=30 FRE=0 FWR=0 NRE=21 NWR=16 SQLC=35 SQLT=49 SID=D719C0BFFC98060FBB07A29A9C****** TRS=429 TIP=- TML=- TSL=- ST=1577942878647 ET=1577942879494 TTB=296 RED=- RSR=307 DIA=521 DCP=-
2. If ACT field alone is not enough for you, you want more detail parameter for the action, you could group all detail parameters into a JSON and assign it to window.PerfLog.pageActionParams when you do the Ajax request.
   window.PerfLog.pageActionParams = {field1: "field1Value", field2: true}

   Then PerfLog will treat first level field in the JSON as customized field in PerfLog Event and generate below log:
   2020-01-02 13:27:59,494 PLV=EVENT CIP=192.168.106.2 CMID=BizXTest CMN="BizXTest" SN=BIZX_BizXTest. DPN=dbPool1 UID=cgrant1 UN=cgrant LOC=en_US EID="EVENT-PLT-HOMEPAGE-H211252-20200102132758-0012" AGN="[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36]" RDT=404 SVT=282 EET=847 EETC=852 JSC=15 CSSC=8 CAID=5414616194-1 MID=HOME PID=HOME_TAB PQ=HOME_V3 ACT=ActionName1 MEM=18608 CPU=160 UCPU=130 SCPU=30 FRE=0 FWR=0 NRE=21 NWR=16 SQLC=35 SQLT=49 SID=D719C0BFFC98060FBB07A29A9C****** TRS=429 TIP=- TML=- TSL=- ST=1577942878647 ET=1577942879494 TTB=296 RED=- RSR=307 DIA=521 DCP=- **C_field1="field1Value" C_field2="true"**

3. Currently PerfLog Event only generated for request that went to server side, if you have pure UI action, which will not send any request to server side, and you want to analyze its execution time, you could use below code to accomplish this.

   //start manual event
   window.PerfLog.startManualEvent()

   //end manual event, parameter is action name, after end is called, start time will reset to null
   window.PerfLog.endManualEvent("action3")

   then server side will generate event:

2019-11-25 17:30:03,267 PLV=EVENT **DT=MANUALEVENT** CIP=192.168.106.2 CMID=BizXTest CMN="BizXTest" SN=BIZX_BizXTest. DPN=dbPool1 UID=cgrant1 UN=cgrant LOC=en_US EID="EVENT-PLT-HOMEPAGE-H211252-20191125172921-0063-1" AGN="[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.97 Safari/537.36]" RDT=- SVT=- EET=20819 JSC=- CSSC=- CAID=8959823930-1-Y MID=HOME PID=HOME_TAB PQ=HOME_V3 **ACT=action3** MEM=- CPU=- UCPU=- SCPU=- FRE=- FWR=- NRE=- NWR=- SQLC=- SQLT=- SID=36610DAD6EA726E9D64C5EE7FE****** TRS=142 TIP=342 TML=342 TSL=3642 ST=1574674182139 ET=1574674202958 TTB=206 RED=- RSR=260 DIA=308 DCP=431

2.9 Event Valid check

perfLogServlet is the request that will collect Event information from client and send back to server side, this request will generate Event accordingly, invalid perfLogServlet will generate wrong EET/EETC in server side.

Know scenarios include:

- Sometimes, external Security testing will mock perfLogServlet request with wrong parameter.
- Customer may record UI flow as automation scripts, and replay it.
- Even the session already timed out, sometimes we can still get perfLogServlet request of that page.

To prevent this, we had two kinds of checking for this.

2.9.1 Hash check

We will add request hash code in request header, which is consistent with the request together with request parameters. If the request is manually changed, and the hash not changed, server side will know the event is invalid, event log for this event will be skipped.

This feature is designed to prevent manually change to request parameters in perfLogServlet.

2.9.2 Session Token check

Server side will generate a token based on session id, following perfLogServlet request will add this token as one of the request parameters, so server side could check whether the request is valid.

This feature is designed to prevent repeatedly sending same recorded perfLogServlet request, also perfLogServlet request from timed out session.

If above checks in 2.9.1 and 2.9.2 failed, we will output log in Server log like below:

[log4j2] 12:04:57,657 INFO [PerfLogBaseServlet] [BizXTest,BizXTest,BIZX_BizXTest.,dbPool1,cgrant1,cgrant,en_US] Event is ignored because of invalid for EVENT-UNKNOWN-UNKNOWN-H211252-20200316120457-0034

2.9.3 Logged in User check

If the session is not for a logged in user(logged in user should have at least company id in session), we will treat it as invalid request and skip it in event log.

- For LOGIN page, it does not have logged in session yet, so we added this page in sf.sfv4.perflog.sessioncheck.skipmodules, then LOGIN page will skip this check

2.10 Hash Session ID

The external users want to analyze the user behaviors via session id in PerfLog, but the last 6 characters of the session id were masked by * for security reason. Here a new field **HSID** was added in PerfLog, which is the hash(a SHA256 algorithm) of the session id. It's identical for each user session, and can be used to group the PerfLogs for a specific user session for further analysis.

2.11 Monitor Shared Library

To have accurate time tracking in Bizx, PerfLog add time tracking to some shared libraries, including Kafka, SOLR, Cache and External Web Request.

2.11.1 Kafka

You could find time tracking for kafka message pulish, this is implemented in com.successfactors.hermes.core.HermesPublisherImpl.

"n":"Spring:com.successfactors.hermes.core.HermesPublisherImpl.publish","i":6,"t":34,

There is one special case for Kafka used in GDPR, which is using another library, time tracking for that will be show as below

"n":"do publish kafka message","i":1,"t":24,

2.11.2 SOLR

You could find time tracking for SOLR related actions, this is implemented in com.successfactors.search.util.solr.SolrSearchServiceImpl, all functions in this class are tracked.

n":"com.successfactors.search.util.solr.SolrSearchServiceImpl.query","i":5,"t":115

2.11.3 External Web Request

In Bizx, most external web request is sent out using JAX-RS or HttpClient, PerfLog tracked many of them, you could see them as blow:

"n":"URL:http://ip-10-116-41-118.lab-rot.saas.sap.corp/rest/v1/jobrequest","i":1,"t":228

Because there are no central Bizx library to do this, we may not cover all invoking to JAX-RS and HttpClient, if you want to add tracking to your library, please refer to Tracking Web Requst in Bizx or contact us directly.

2.11.4 Cache Service

You could find time tracking of Cache Service as below,.

"n":"SUPERSFEDIS-GLOBALSYSCONFIGCACHE-com.successfactors.cachelegacy.impl.MigrationCacheAdapter.get","i":14,"t":13

You could see name pattern for n field, actually it is "Cache Service Type"-"Cache Type"-"Cache Service implementation Class"."Method invoked",

Possbile value for Cache Service Type is:

- CONCURRENT: Local Cache
- SUPERSFEDIS: May access both local and remote Redis cache
- LOCALSFEDIS: Local Cache
- EXTREMESFEDIS: May access both local and remote Redis cache
- REMOTESFEDIS: Remote Redis Cache

Based on Cache Type, you could find cache owner.

Also, PerfLog does a summary to Cache Service type in both Request level and event level, below is part of PerfLog sample, these 5 fields represent total function execution time for these 5 cache service types, detail field definition could be found in 2.4KeyValuePairkeydefinition

CCON=1 CSUP=272 CLOC=0 CEXT=0 CREM=0

3 Design

3.1 Overview

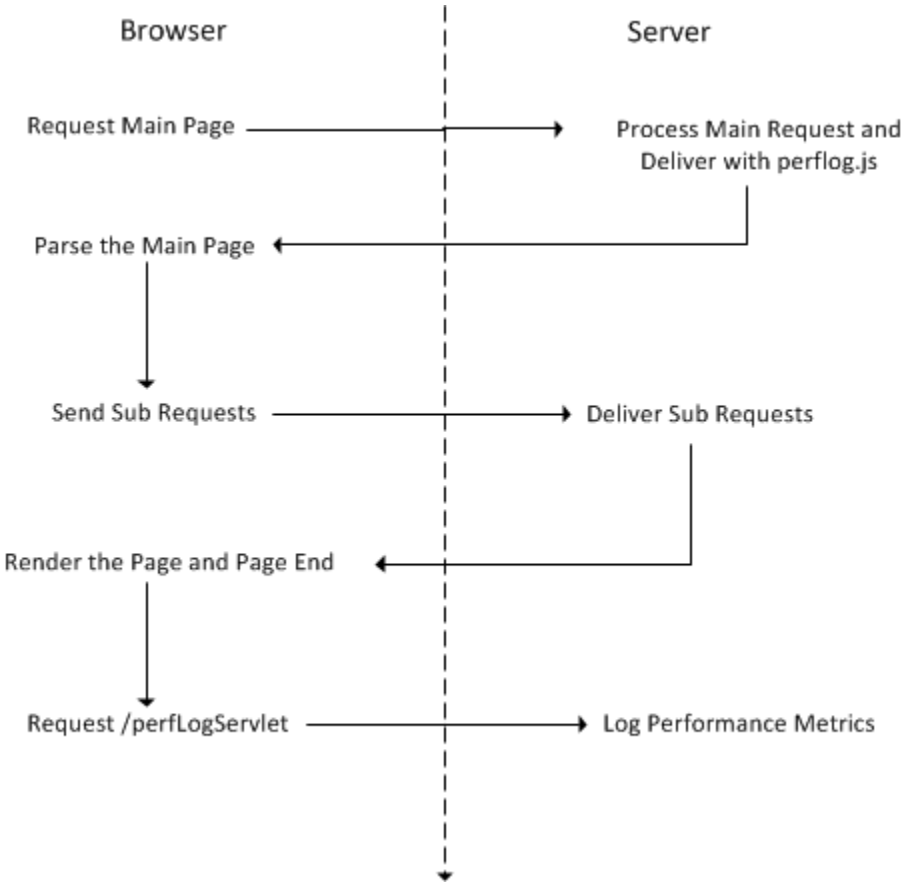
This section describes the design of Performance Log Framework. PerfLog is maind on Java Servlet/Filter technologies, together with client side Javascript.

3.2 Event Logging

Bizx will change to be stateless, Original PerfLog event is designed based on Session, we can not depend on session any more after stateless change.

So Perflog changed event logic to maintain event information in client side, including event start time, server time for each request, SQL time for each request, Memory, CPU, IO, Network usage for each request.

As described in the sections above, PerfLog will capture event level logs with performance metrics. The diagram below illustrates how the flow works:



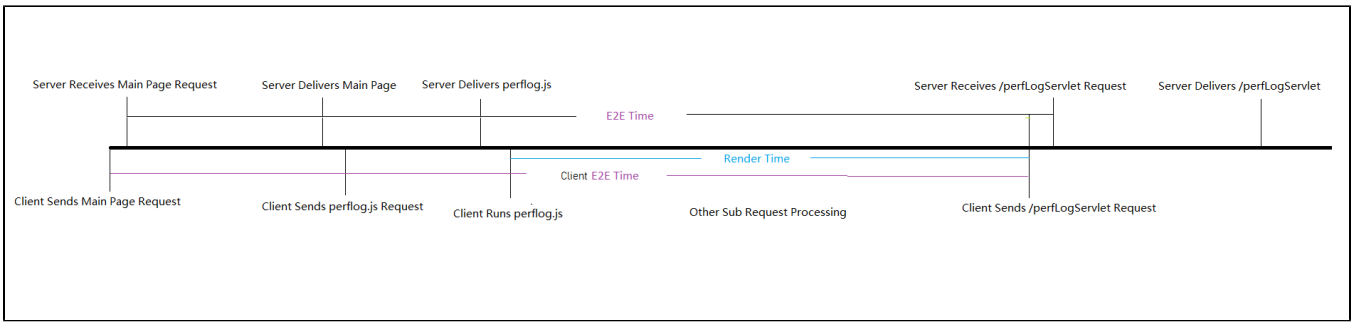
1. Client side initializes the main page request, for example, user makes a login, which will load /sf/home page
2. Server side checks whether there is event ID in http header, for the initial page loading request(the whole page refreshed), there is no event id in http header, so server side create new event, then server side proceeds the main page request and respond back with the HTML content(with perflog.js and tracking code injected), finally, write event id, event start time, Memory usage, network usage, io usage, sql usage, server time, cpu usage in response header.
3. Client side get event info and resource usage from http header, keep it as event basic information. Client side sends sub requests while parsing the HTML dom, with event id in http request header.
4. Server side proceeds the sub requests one by one and responds back to client, when server side find the event id in request http header, it will use same event id, and finally write event id, event start time, Memory usage, network usage, io usage, sql usage, server time, cpu usage in response header.
5. Client side get event info and resource usage from http header, accumulate resource usage if event id is same, finally clients side sends back /perfLogServlet call after page is loaded completely, with the performance metrics collected, including event info and resource usage that comes from http header.
6. Server side logs the performance metrics

The /perfLogSevlet call carries the parameters below:

- Render Time
- # of JS resource on the page
- # of CSS resource on the page
- Call Id
- Module Id
- Page Id
- Page Qualifier

There are 3 different time metrics logged at event level:

- Render Time: between the time when perflog.js is loaded and the time when /perfLogServlet call is sent
- Server Time: the accumulated processing time for all the requests associated with the event
- End2End Time: between the time when Client sent out main page request and the time when Client sent out the /perfLogServlet
    - For the loading page, they have two End to End time logged.
        - EET: start time is server side receivers main page request, end time is server side receives /perfLogServlet request
        - EETC: start time is client sends out main page request(value comes from Timing API navigationStart), end time is client sent out /perfLogServlet request
    - For event generated after initial page loading, they just have one EET, start time is client sends out main page request, end time is client sent out /perfLogServlet request



/perfLogServlet is a Java servlet on the server side to collect performance metrics and log them.

After main page is loaded, user may have further user clicking which will only refresh part of page, this will generate event in perflog.
when client sends out first request, it will initialize the event start time based on client local time and event id from main page event id plus one auto-increased int, since server side perflog event id will use event id in request header if it is there, we will get new event in server side.
When client thinks all requests for the event have finished, it will send out perfLogServlet request with performance metrics and also event ending time based on client local time.

To have accurate E2E time that could reflect initial page performance, now Perflog will send out one special perflogServlet request when TML is ready or DOM readyState is Complete, it will generate one event which has no request related, but E2E time for this event could be used to check page performance.

- CAID for this event is ended with -X
- It contains DT field with value as PAGEPERFORMANCE

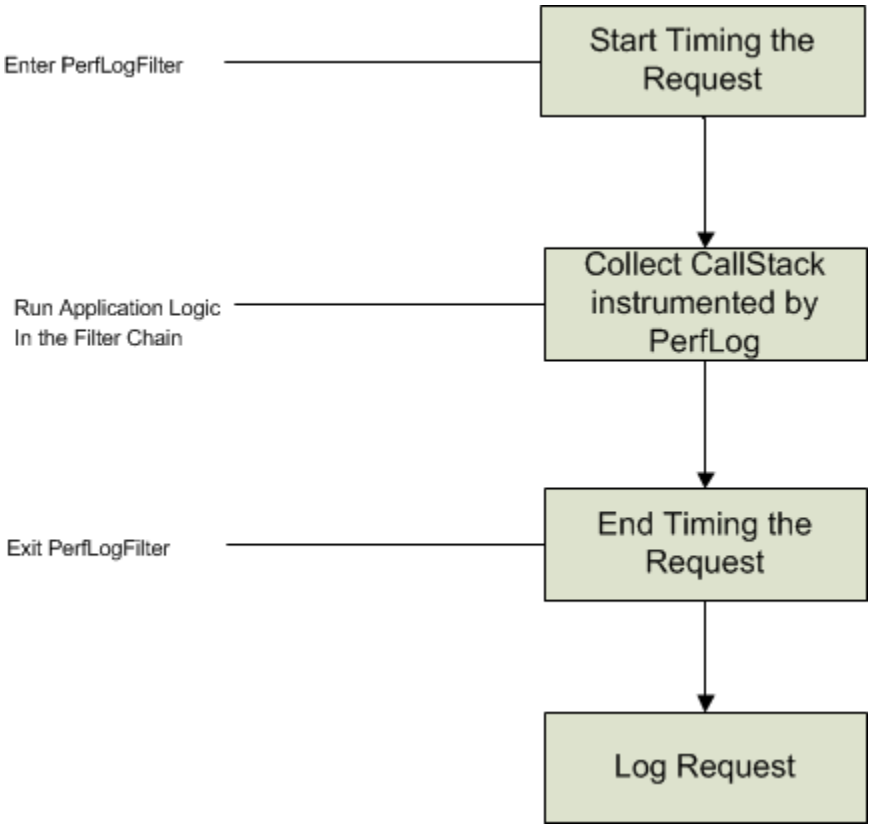
If the page is UI5 and implemented UI5 Performance Metrics, it should have TML data during page loading. So two kinds of scenarios exist:

1. TML is there, PerfLog will check both TML and DOM readyState.
   PerfLog will send out Page Performance event when any of below two events: TML is generated or DOM readyState changes to Complete.
   and PerfLog only sends out one page performance request in one page. During our testing, TML mostly happens before DOM ready.
2. TML is not there. PerfLog will check DOM readyState, when it is Complete, page performance event will be sent out.

3.3 Request Logging

Each HTTP request associated with an event will be logged with performance metrics. The server side is configured with PerfLogFilter in the filter chain the captured the begin and end time for processing a HTTP request.

The diagram below illustrates how request level logging works:



The request level logging is implemented via PerfLogFilter.

1. PerfLogFilter receives a HTTP request
2. PerfLogFilter check if there is an existing event id. If not, generate a new event id and store it in session.
3. PerfLogFilter starts timing
4. PerfLogFilter invoke other filters to run application logic
5. CallStack is collected for the proxied classes by PerfLogProxy
6. Application returns to PerfLogFilter
7. PerfLogFilter ends timing
8. PerfLogFilter writes logs

The PerfLogFilter shall be configured in web.xml like below, if it is tomcat, we should add the filter in tomcat-web.xml.

```
<filter>
<filter-name>PerfLogFilter</filter-name>
<filter-class>com.successfactors.perflog.ui.PerfLogFilter</filter-class>

</filter>

...

<filter-mapping>
<filter-name>PerfLogFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

The filter-mapping section shall be place after Logging Filter filter-mapping in web.xml, because PerfLogFilter need to get request id from Logging FIlter.

To ensure Event could be properly recorded, PerfLogServlet configuration need to be added into web.xml, if it is tomcat, we should add the filter in tomcat-web.xml.

```
<servlet>
<servlet-name>PerfLogServlet</servlet-name>
<display-name>Performance Logging Servlet</display-name>
<servlet-class>com.successfactors.perflog.ui.PerfLogServlet</servlet-class>
</servlet>

...


<servlet-mapping>
<servlet-name>PerfLogServlet</servlet-name>
<url-pattern>/perfLogServlet</url-pattern>
</servlet-mapping>
```

3.4 API

The sections above describe how PerfLog is applied during http requests processing. While for some other application that is not triggered by http requests, PerfLog also offer APIs to track the performance metrics as well as instrument.

Below are the core APIs:

API	Remark
PerfEvent com.successfactors.perflog.PerfEventFactory.initializePerfCallStack(String eventName)	Starts a PerfLog event with factory type as null
PerfEvent com.successfactors.perflog.PerfEventFactory.initializePerfCallStack(String eventName, String FactoryType)	Starts a PerfLog event with factory type defined
void com.successfactors.perflog.PerfEventFactory. endPerfCallStack(PerfEvent event)	Ends a PerfLog event and log the event
Object com.successfactors.perflog.PerfLogProxy.newInstance(Object obj)	Proxy an interface
Object com.successfactors.perflog.PerfLogProxy.newInstance(Object obj, String str)	Proxy an interface

** **Since PerfEventFactory could be used by different module, if you want to identify your perflog from them, Factory Type need to be provided.**

A sample use of these APIs for S2.

In the example, you could also see PerfLogFieldLogger.addCoreLog and PerfLogFieldLogger.addCustomizedLog are invoked, this is used to provide more information to Perflog final

output log, for the field mentioned in section 2.4, you need to invoke PerfLogFieldLogger.addCoreLog to add data, for other module customized field, you could invoke PerfLogFieldLogger.addCustomizedLog

```
PerfEvent event = PerfEventFactory.initializePerfCallStack("[" + job.getJobName() + "]-[" + Long.toString(job.getJobExecutionId()) + "]",
"S2JOB");
PerfLogFieldLogger.addCoreLog(PerfLogField.COMP_ID, job.getCompanyId());
PerfLogFieldLogger.addCustomizedLog("job_name", job.getJobName());
PerfLogFieldLogger.addCustomizedLog("exec_id", Long.toString(job.getJobExecutionId()));
PerfLogFieldLogger.addCustomizedLog("type_name", job.getTypeName());

.....Job related code to execute Job.....

PerfEventFactory.endPerfCallStack(event);
```

After key value pair feature is enabled, perflog could allow module team to add other information to Perflog with following API, please be aware that

there is some constrain when customized log is added, after successfully added, new data will be outputted together with existing perfLog, customized data will be after

PerfLog core data, but before call stack.

```
Class: com.successfactors.perflog.PerfLogFieldLogger

/**
* Add customized data to PerfLog, key could only be alphanumeric and _
*
* @param key
* key for the data
* @param value
* value for the data
* @return 0 means successfully, 2 means key is not alphanumeric,
* 3 means the key is already there in core log
* 4 means the key is already there in extention log
*/

public static int addCustomizedLog(String key, String value){

....

}
```

3.5 Add entry to PerfLog Call Stack

In Bizx, PerfLog enabled call stack collection for Seam, SCA, JDBC, DAO, besides these entries, if module team want to add your own entry, you could following below example.

```
CallStack stack = null;
try {
if (PerfLogProxy.perfLogEnabled) {

stack = PerfLogProxy.stackHolder.get();
if (stack != null) {
stack.startCall("meaningful name of the call");
}

}

..... The logic you want to track with Perflog.....

}
catch (....) {
}
finally {
if (stack != null) {
stack.endCurrentCall();
}
}
```

Live Class Example.

For easy use of Perflog, we wrapped previous code with Java Try with resource, below sample code will have same effect as previous example to add entries to PerfLog Call Stack.

```
import com.successfactors.perflog.PerfLogAround;



try (PerfLogAround perflog = new PerfLogAround("meaningful name of the call")) {
..... The logic you want to track with Perflog.....
}
```

3.5.1 Add entry to PerfLog Call Stack by Annotation

Now PerfLog Supports to add PerfLog entry to Call Stack by Annotation for Spring/Seam.

In Bizx, we created Interceptor for both Spring and Seam, so if PerfLog Annotation is added to Spring/Seam class, its method's execution time will be tracked in Call Stack.

Sample Node name in Call Stack as below, whether it is Seam or Spring depends on config in Bizx.

{"n":"Seam:com.successfactors.genericobject.app.impl.MDFRBPFacadeImpl.getRBPObjectPermission","i":27,"t":478}

{"n":"Spring:com.successfactors.genericobject.app.impl.MDFRBPFacadeImpl.getRBPObjectPermission","i":27,"t":478}

All you need to do is to add com.successfactors.perflog.PerfLog annotation to your Seam/Spring class.

```
import com .successfactors.perflog.PerfLog ;

@Name(MDFRBPFacade.FACADE_NAME)
@javax.inject.Named(MDFRBPFacade.FACADE_NAME)
@Scope(ScopeType.EVENT)
@org.springframework.context.annotation.Scope(SFContextConstant.SCOPE_EVENT)
@PerfLog
public class MDFRBPFacadeImpl implements MDFRBPFacade {

.....

}
```

Live Class Example.

3.5.2 Add entry to PerfLog Call Stack by Proxy

If you want to intercept all methods in a class, you can use PerfLogProxy to add Call Stack, wrap your class with PerfLogProxy as below example.

Please be aware that PerfLogProxy is using Java Dynamic Proxy, so your instance to be proxied need to have interface defined, just like GroupDAO defined methods and OracleGroupDAO do the implementation.

```
import com.successfactors.perflog.PerfLogProxy;

public GroupDAO getGroupDAO() {
return (GroupDAO) PerfLogProxy.newInstance(new OracleGroupDAO());

}
```

Live Class Example.

Based on previous proxy, PerfLog tracking will be added for each method of the proxied instance, the name for the call stack node will be classname. methodname.

Sometimes, you may want to add extra information to the node name, so we provide newInstanceWithNamePrefix function, it will still return a proxy object, but with one extra parameter prefix, which will be added to starting position of the node name.

```
import com.successfactors.perflog.
PerfLogProxy;

String prefix = "CacheType:CacheName:"

return (CacheService) PerfLogProxy.
newInstanceWithNamePrefix(
cacheService, prefix);
```

If your instance don't have interface defined, PerfLog has another class PerfLogExtendedProxy provided to do proxy.

Please be aware that:

1. PerfLogExtendedProxy is using Cglib to do proxy, you will need to add gradle dependency to use it.
2. it doesn't support final, if you have final method in the class, you won't get perflog tracking around it.
3. **please only use this method when your class don't have interface defined, and fully tests your methods.**

```
import com.successfactors.perflog.PerfLogExtendedProxy ;

SampleClass instance = (SampleClass) PerfLogExtendedProxy.newInstance(new SampleClass());
```

3.6 Configuration

Configuration is required on both web server and application server to enable PerfLog.

On web server, to allow /perfLogServlet to forward to backend application server, the section below is required to add mod_jk mapping config file pm.properties:

```
JkMount /perfLogServlet prodloadbalancer
```

On application server, the section below need to be added into the log4j config file conf/jboss-log4j.xml:

```
<!-- A time/date based perf log rolling appender -->
< appender name = "PERFLOG" class = "org.jboss.logging.appender.DailyRollingFileAppender" >
    < errorHandler class = "org.jboss.logging.util.OnlyOnceErrorHandler" />
    < param name = "File" value = "${jboss.server.log.dir}/perf.log" />
    < param name = "Threshold" value = "INFO" />

    <!-- Rollover at midnight each day -->
    < param name = "DatePattern" value = "'.'yyyy-MM-dd" />

    < layout class = "org.apache.log4j.PatternLayout" >
       <!-- The default pattern: Date Priority [Category] Message\n -->
       < param name = "ConversionPattern" value = "%d %X{ipAddress} %X{params} %X{perf.eventName} %X{perf.eventId}
%X{perf.event.renderTime} %X{perf.event.serverTime} %X{perf.event.end2endTime} %X{perf.event.jsNum} %X{perf.
event.cssNum} %X{perf.req.id} %X{perf.req.method} %X{perf.req.uri} %X{perf.req.serverTime} %X{perf.req.params}
%X{perf.req.callstack} %m%n" />

    </ layout >
</ appender >

< category name = "PerfLog" additivity = "false" >
   < appender-ref ref = "PERFLOG" />
</ category >
```

And the following JVM parameters are available in PerfLog via run.conf:

- **sf.sfv4.perflog.enabled**=true

Default is true. It controls whether to turn on PerfLog.

- **sf.sfv4.perflog.time.threshold**=10000

Default is 10000ms (10s). Only if a request processing time exceeded the threshold, its request parameters and call stack can be recorded.

- **sf.sfv4.perflog.call.time.threshold**=10

Default is 10ms. If a call aggregately less than this threshold, it won't be printed in the call stack.

- **sf.sfv4.perflog.logsqls.enabled**=true

Default is true. If it is set to false, sql statement won't be output into the call stack.

- **sf.sfv4.perflog.logparameters.enabled**=false

Default is false. There is security concern on logging every http request's parameter values. However parameter names are fine, they are always logged.

- **sf.sfv4.perflog.callstack.maxlevel**=100

Default is 100. It is to limit the memory consumption, a call stack only records limited levels. Any calls deeper than this maximum level won't be recorded.

- **sf.sfv4.perflog.ignorelist**=none

It is used to allow some requests bypass perflog. In case perflog has any conflicts with some requests, we can tell perflog ignore these requests by setting this parameter. Default is empty. The ignorelist is consists of multiple patterns which are separated by ":", and wild chars "*" and "?" can be used in the pattern, for example "**/home:/caree?:/pageHeaderController.*.dwr** ".

- **sf.sfv4.perflog.monitor.enabled**=true

Default is true. but the resource monitor will be in effect only when **sf.sfv4.perflog.enabled**=true, otherwise it still considered as disabled.

- **sf.sfv4.perflog.logmemory.enabled**=true

Default is ture. If it is set to false, it will not log the consumed memory of request & event.

- **sf.sfv4.perflog.logcpu.enabled**=true

Default is ture. If it is set to false, it will not log the consumed CPU information of request & event.

- **sf.sfv4.perflog.logio.enabled**=true

Default is ture. If it is set to false, it will not log the consumed IO information of request & event.

- **sf.sfv4.perflog.trackchildthread.enabled**=false

Default is false. If it is set to true, it will track the child thread consumed resource and try to send it back to its parent thread, so finally we will have a more accurate summarize.

- **sf.sfv4.perflog.childthreadresourcelistener.class**=null

Default is null. If it set to a class that implement the interface **com.successfactors.perflog.monitor.ChildThreadInfoListener,** then resource monitor will try to pass the child thread consumed resource information to this interface.

As it use java reflect to construct an instance of this class, so the class must have a non-parameter constructor. This setting is useful if your monitored code will start child threads and they will not join back to the parent thread, at this situation

the parent thread can not wait for the consumed resource information of child threads, so resource monitor will pass those information to this listener.

- **sf.sfv4.perflog.keypair.enabled=true**

Define perflog log format, false is fix position format, true is key value pair format.

- **sf.sfv4.perflog.jdbcdetail.enabled=true**

Define whether PerfLog tracks JDBC detail mentioned in Section 2.5, by default it is true.

- **sf.sfv4.perflog.sqlrequestmap.enabled=true**

Define Request DB correlation feature, by default it is enabled.

- **sf.sfv4.perflog.logbadsqlpattern.enabled=false**

Define Bad SQL checking feature as described in section 2.3, by default it is disabled. Now it is only enabled in QA, Production is disabled.

- **sf.sfv4.perflog.eventperrequest.enabled=false**

Enable this will generate unique event for each request, and it will not have any session change action. Please be aware, this attribute should not be enabled for normal CF app, or event will be generated with wrong logic.

- **sf.sfv4.perflog.keysteps.enabled=true**

When this is enabled, even the request time is less than threshold of sf.sfv4.perflog.time.threshold, Perflog will still log key steps stack trace. Key steps means we get max time for each layer and print it out.

- **sf.sfv4.perflog.keysteps.threshold=1000**

Threshold for call stack key steps, if request time is less than this, we will not print key steps any more, if request time is longer than sf.sfv4.perflog.time.threshold, will only print real call stack.

- **sf.sfv4.perflog.trackhttprequest.enabled= true**

When it is true, PerfLog will track external Web request implemented by JAX_RS or HttpClient, by default, it is true.

- **sf.sfv4.perflog.noderesourceusage.enabled= true**

When it is true, PerfLog call stack will show SCA memory usage in field m, by default, it is true.

- **sf.sfv4.perflog.mazidpatterninservice="Service":".+?-.+?-(([^"]+?-[^"]+?)|([^"]+?))(-[^"]+)*"**

MazId pattern in Consul agent service response, more detail refering to PPF-264

- **sf.sfv4.perflog.sessioncheck.enabled= true**

if the request module is in sf.sfv4.perflog.sessioncheck.skipmodules, we will skip the session check. if not, only if PerfLogConstants.PARAMS in session and company id there, session is valid.

- **sf.sfv4.perflog.sessioncheck.skipmodules= LOGIN**

if the request module is in sf.sfv4.perflog.sessioncheck.skipmodules, we will skip the session check.

- **sf.sfv4.perflog.long.eet.threshold=1800000**

Threshold to mark an event as LONGEET, if EET or EETC is longer than the threshold, DT will be LONGEET

Kindly notice all above parameter are optional. If they are not set, Tomcat can start up and function as usual.

Currently only one parameter is configured in Tomcat sfserver.properties(Consul in future), other config are all using default value.

- **sf.sfv4.perflog.logbadsqlpattern.enabled**
  configured as true in DC13, false in other environments.

The configuration options below are available in JMX bean PerfLogJMX:

- **sf.sfv4.perflog.enabled**
- **sf.sfv4.perflog.logsqls.enabled**
- **sf.sfv4.perflog.time.threshold**
- **sf.sfv4.perflog.call.time.threshold**
- **sf.sfv4.perflog.logparameters.enabled**
- **sf.sfv4.perflog.monitor.enabled**
- **sf.sfv4.perflog.logmemory.enabled**
- **sf.sfv4.perflog.logcpu.enabled**
- **sf.sfv4.perflog.logio.enabled**
- **sf.sfv4.perflog.trackchildthread.enabled**
- **sf.sfv4.perflog.childthreadresourcelistener.class**

The changes of the options via JMX bean are picked up by PerfLog at runtime.

4 Open Issues

- Since PerfLog utilizes the dynamic proxy mechanism to return a proxied object which implemented all the origin interfaces, but it won't extend the origin class, casting the returned object to a concrete implementation can lead to exception.

5 Related Configuration

5.1 LogStash configuration sample

```
input {
beats {
port => 5044
}
}

filter {
grok {
match => {
"message" =>["%{TIMESTAMP_ISO8601:logdate} (?<msg>((?!STK=).)*)(STK=(?<stacktrace>.*))?"]
}
}
kv {
source => "msg"
field_split_pattern => " "
trim_key => " "
trim_value => " "
}
}

output {
elasticsearch {
hosts => "localhost:9200"
manage_template => false
index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
}
}
```

6 Known Issues

Performance Log Framework Known Issues

7 Release Notes

- PerfLog b2005 Release Notes

8 Reference

- Tools to parse PerfLog call stack developed by Performance team: http://perfpoint.c.eu-de-2.cloud.sap/perflog/parser embedded in performance engineering one-stop shop: PerfPoint http://perfpoint.c.eu-de-2.cloud.sap
- Tools to parse PerfLog call stack: http://10.76.118.52/perf-analyzer/ (original), https://github.wdf.sap.corp/pages/ec-perf/perflog-analyser/ (forked and enhanced)
- PerfLog + Splunk for performance analysis
- KICKS on PerfLog(slides recording)
- Perflog Resource Monitor.pptx
- Performance Log Framework User Guide