Application Performance Management

FAQs

Issue	01
Date	2026-07-29

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 JavaAgent Connection FAQs

1.1 Why Is the Application Service Startup Slow After JavaAgent Is Connected?

The following table lists the causes and solutions for slow application service startup after JavaAgent is connected.

Root Cause	Solution
The configured memory of the application service is small, causing full GC. (If the memory allocated to the application is too small, the application may fail to start.)	Increase the memory and ensure that 200–250 MB memory is reserved.
The host name is not configured in the **/etc/hosts** file. It takes much time to obtain the local IP address.	Add the host name to the **hosts** file.

1.2 Why Is Tomcat API Data Not Collected?

Cause: The **threadLocal** data is completely cleared.

Solution: Specify the **threadLocal** data to be cleared.

1.3 Why Are My Services Affected After Connecting to the Agent During AOP Code Execution?

Cause: APM intercepts the runnable implementation class and adds two methods. However, no method name is checked during aspect oriented programming (AOP) interception. As a result, the two methods added by APM are intercepted.

Solution: You need to determine the name of the method to be intercepted and intercept only the run method of the runnable implementation class.

1.4 Why Is "apm Service Not Register for Region:xxxx" Displayed?

Symptom: The Agent fails to be registered, and the error message "apm service not register for region:xxxx" is displayed.

Solution: Enable APM 2.0 in the corresponding region.

1.5 What Can I Do If the APM Agent Fails to Be Connected and "sk not match" Is Displayed?

Cause 1: The SK is incorrect.

Solution: Ensure that the SK in the **apm.config** file is the same as that on the page.

Cause 2: The SK has been encrypted, but the decryption fails.

Solution: Place the decryption package in the **apm-javaagent/ext** directory and try again. For details, see **Access Keys**.

1.6 Why Can't Classes Be Found When Multiple Agents Are Connected?

Cause: When multiple Agents such as Java Code Coverage (JaCoCo) and APM Agents are connected, the Agents cannot find class exceptions.

Solution: Change the positions of the Agents. Specifically, place the JaCoCo Agent in front of the APM Agent. The APM Agent has no requirement on positions.

1.7 What Can I Do If No Tomcat 9 Data Is Reported?

To solve the problem, add the following configuration:

server.tomcat.mbeanregistry.enabled=true

Then Tomcat 9 data can be properly reported.

1.8 How Do I Troubleshoot Network Problems?

You can run the **curl -kv** command to check the network.

```
curl -kv masterAddress:port
```

For example, if you select region **CN-Hong Kong** and set **Access Mode** to **Enhanced Agent**, log in to the host where the application is deployed and run the **curl -kv 100.125.6.106:41333** command to check the network connectivity. For details about access addresses in other regions, see **Agent Access Addresses**.

If the connection fails, check whether the ICMP security group of the firewall is enabled. Run the **curl** command to check whether the security groups of ports 41333 and 41335 are enabled. If the connection still fails, check the network on the service side and contact APM on-call personnel.

1.9 Why Do Agents Go Offline When External Users Use APM?

When external users use APM, Agents may go offline. To solve the problem, do as follows:

- Check whether the edition is free. The free edition supports a maximum of 10 Agents. If a CCE cluster is connected to APM, the number of Agents may temporarily exceed 10. As a result, other Agents stop collecting data and become gray.
- Check whether the account is in arrears. If the account is in arrears, all Agents are automatically stopped.
- Check whether the Agents are manually stopped. You can log in to the system as a resource tenant to view the number of valid Agents and active Agents of the user.

1.10 Why Is the Data of a Single Instance Displayed with a Great Delay on the APM Console?

Symptom: When a user tries to view all instances on the APM console, no data is displayed. However, when the user selects a single instance, data is displayed, but there is a great delay. For example, the current time is 11:30, but the data of 11:10 is displayed.

Solution: Run the **date** command to check whether the server time is correct. If the time is incorrect, calibrate it.

1.11 What Operation Do I Need to Take When Deploying a WAR Package?

Linux/Unix Environment

Add the following content to **catalina.sh**. The **-javaagent** parameter specifies the path of your JavaAgent JAR file:

```
JAVA_OPTS="$JAVA_OPTS -javaagent:/root/my-dir/apm-javaagent/apm-javaagent.jar=appName=my-service,env=dev, business=dev"
```

Windows Environment

Add the following content to **catalina.bat**. The **-javaagent** parameter specifies the path of your JavaAgent JAR file:

```
set JAVA_OPTS="%JAVA_OPTS% -javaagent:/root/my-dir/apm-javaagent/apm-javaagent.jar=appName=my-service,env=dev, business=dev"
```


1.12 What Should I Do If "business name not exists:xxx" Is Displayed?

To solve the problem, go to the APM console and check whether the application exists.

1. If the application does not exist, create an application with the same name as the parameter value and restart the application. For details, see **Creating an Application**.
2. If the application exists, check whether the application name is the same as the parameter value. For details, see **Configuring an Application and Sub-application**.

1.13 Why Is an Error Reported When I View the Trace Details?

Symptom: When I click **Detail** in the error trace column on the **Exceptions** tab page, an error is reported on the trace page. There is no topology, and the root span in the trace list is marked as fake. Alternatively, neither a topology nor a trace list is displayed.

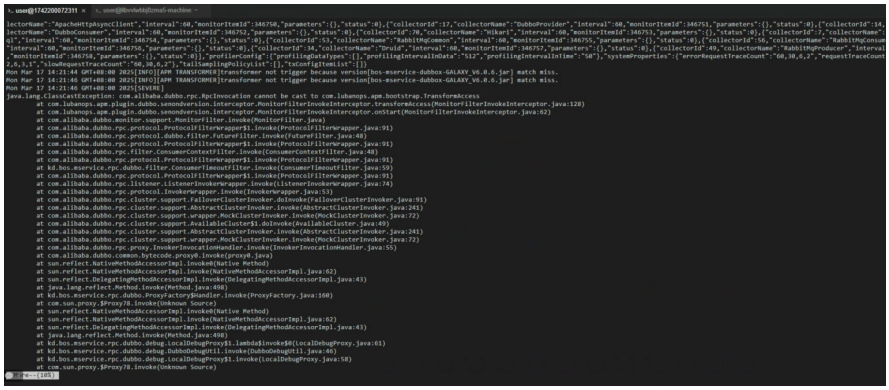
During metric reporting, if an exception occurs, the trace ID will be recorded in the metric and reported. In this case, the **Detail** button is available on the UI. Click the **Detail** button and query the trace details by trace ID. However, if the traces fail to be reported, an error occurs. There are two symptoms:

1. The topology is not displayed, but the trace list is displayed. However, the root trace is filled with fake data.
2. Neither the topology nor the trace list is displayed.

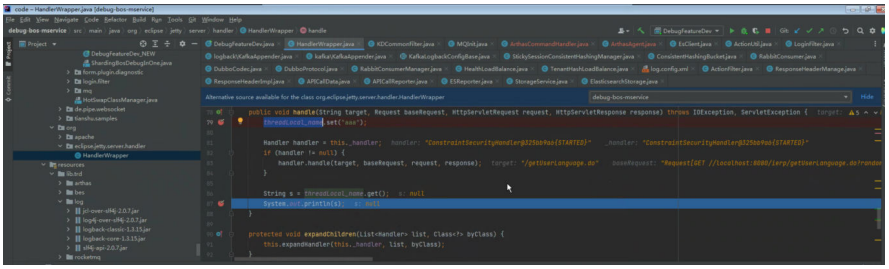
If the child span of the trace is reported but the root span fails to be reported, symptom 1 occurs. If there is no child span or neither the child span nor the root span is reported, symptom 2 occurs.

Solution:

1. Check for exception information in the **apm.log** file, for example, version incompatibility.

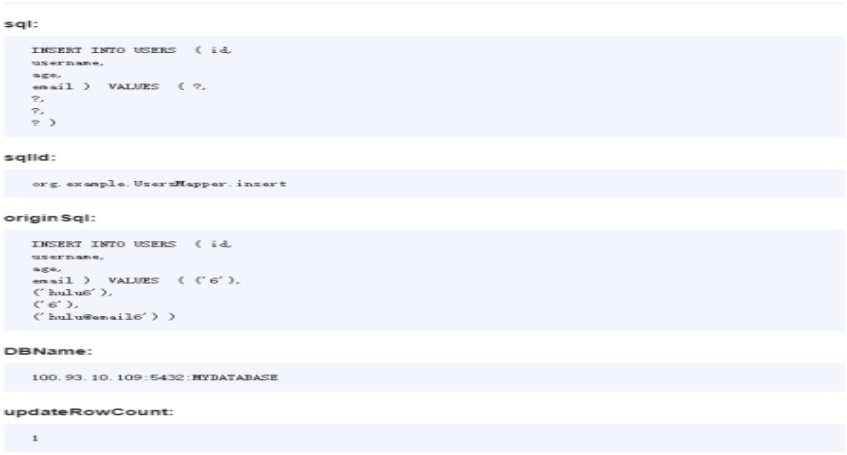


2. If the **apm.log** file does not contain exception information, check whether the problem occurs inevitably. If the problem occurs inevitably, it is usually caused by the client. If the problem does not occur inevitably, the background data may be lost.
3. If the problem occurs inevitably and no exception log about the Agent is generated, the customer's service may clear the **threadLocal** data after the function interception processing. The following figure shows the Jetty framework. In the handle function, the **threadLocal** of the entire thread will be cleared at the last step. As a result, the Agent cannot report the root span of the trace.



1.14 Why Are GaussDB SQL Statements Not Displayed on the Trace Page?

The following figure shows one correct GaussDB statement. Currently, apm-javaagent supports SQL collection for GaussDB 200 and GaussDB-JDBC drivers only.



1.15 Will Enhanced JavaAgents Affect User Performance?

APM Agents use the bytecode enhancement technology to dynamically collect performance data, including method calls, exception information, and distributed traces, helping development and O&M teams monitor and optimize system performance in real time. Agents inevitably cause certain performance loss. By evaluating the performance overhead (such as CPU, memory, and latency) of

Agents and the stability and data collection integrity in heavy load scenarios, we can ensure that Agents run reliably in the production environment and their impact on performance is controllable. This evaluation provides important basis for optimization and deployment.

Test Environments

Tool/ Service	Version/ Specifications	Description
JVisualVM	1.8.0_216	JVisualVM is a performance monitoring tool provided by Java. It monitors and manages JConsole. It can provide the memory, thread, class loading, JVM overview, and real-time information of a Java process.
JMeter	5.3	Apache JMeter is a Java-based pressure test tool developed by Apache. In this test, it is used to simulate multiple users to concurrently call an APM API to query graphs.
JavaAgent	2.4.11-profiler	Stable JavaAgent version
ECS	2u4g	General computing-plus | 2 vCPUs | 4 GiB | c7.large.2, CentOS 7.9
Demo application	benchmark.jar	Sends requests based on the pressure test source and accesses the MySQL and Redis services at the same time. Corresponding values will be returned. Spring Cloud and Dubbo are used in this process.

Constraints

1. The Agent version that supports Profiler must be 2.4.5 or later. For details, see **JavaAgent Updates**.
2. The Agent version that allows users to set sampling policies must be 2.4.11 or later. For details, see **JavaAgent Updates**.

Test Procedure

Step 1 Respectively use 1 TPS, 500 TPS, 1,000 TPS, and 2,000 TPS samples for pressure tests when no Agent is installed. Each pressure test lasts for 30 minutes. The pressure test results are used as the performance baseline.

Step 2 Install an Agent, respectively set the sampling policy to intelligent sampling and 100% sampling, repeat the pressure test process in **Step 1**, and compare differences in CPU, memory, and RT.

Step 3 Install an Agent that supports Profiler, disable the performance profiling function, respectively set the sampling policy to intelligent sampling and 100% sampling, repeat the pressure test process in **Step 1**, and compare the differences in CPU, memory, and RT.

----End

Performance limit metrics when no Agent is installed

No.	Pressure Test Samples	RT (ms)	CPU (%)	Memory (MB)
1	1 TPS	77.05	0.4	200
2	500 TPS	77.42	11	250
3	1,000 TPS	79.17	23	300
4	2,000 TPS	83.19	45	350

Performance limit metrics when an Agent is installed

Table 1-1 Intelligent sampling

No.	Pressure Test Samples	RT (ms)	CPU (%)	Memory (MB)
1	1 TPS	78.36	0.4	250
2	500 TPS	79.05	18	300
3	1,000 TPS	81.84	31	350
4	2,000 TPS	86.81	55	400

Table 1-2 100% sampling

No.	Pressure Test Samples	RT (ms)	CPU (%)	Memory (MB)
1	1 TPS	78.38	0.4	250
2	500 TPS	80.39	20	300
3	1,000 TPS	84.51	33	450
4	2,000 TPS	88.72	65	500

Agent Performance Overhead Comparison

No.	Pressure Test Samples	Intelligent Sampling			100% Sampling		
-		RT	CPU	Memory	RT	CPU	Memory

No.	Pressure Test Samples	Intelligent Sampling			100% Sampling		
1	1 TPS	+1.31 ms	+0%	+50 MB	+1.33 ms	+0%	+50 MB
2	500 TPS	+1.63 ms	+7%	+50 MB	+2.97 ms	+9%	+50 MB
3	1,000 TPS	+2.67 ms	+8%	+50 MB	+5.34 ms	+10%	+150 MB
4	2,000 TPS	+3.62 ms	+10%	+50 MB	+5.53 ms	+20%	+150 MB

Conclusion

1. Enhanced JavaAgents have very little impact on RT.
2. Intelligent sampling increases the CPU and memory overhead, but the increase is less than 10% of the total host overhead. The overhead may increase along with the application complexity.
3. The performance overhead of 100% sampling is slightly higher than that of intelligent sampling. Do not enable 100% sampling in heavy load scenarios.

2 Profiler FAQs

If no data is displayed in the Profiler flame graph or other faults occur, view the Profiler logs to locate the cause.

Log path:

```
cd ~/apm/instances/<application_name>/logs/profiler/
```

2.1 Why Is "No access to perf events" Displayed?

Symptom

CPU Profiler depends on the **perf_event_open** system call. However, due to the Syscall security policy (**seccomp**) control of the Linux kernel, processes may be prohibited from calling specific Syscalls.

Error message:

```
[ERROR] xxxx Failed to execute
'start,jfr=7,jstackdepth=100,threads=true,event=cpu,interval=50ms,alloc=512k,wall=50ms,file=xxxx.jfr'
[ERROR] xxxx Failed to start Continuous Profile Collector
[ERROR] xxxx No access to perf events. Try --fdtransfer or --all-user option or 'sysctl
kernel.perf_event_paranoid=1'
```

Solution

- Docker environment: Run the following command to run the container. To configure fine-grained system call control, see the **official document**. Enabling privileged containers may cause container escape. Please exercise cautions.
  ```
  docker run --security-opt seccomp=unconfined  XXX
  ```
- Kubernetes environment: Set **privileged** to **true** for a privileged container. The privileged container is always in the **Unconfined** state. If **privileged** cannot be set to **true**, you can modify **securityContext** in the Kubernetes YAML file to use the default system call control. For details, see the **official document**. Enabling privileged containers may cause container escape. Please exercise cautions.
  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: default-pod
  ```



```
  labels:
    app: default-pod
spec:
  containers:
    - name: test-container
      image: hashicorp/http-echo:1.0
      args:
        - "-text=just made some more syscalls!"
      securityContext:
        seccompProfile:
          type: RuntimeDefault
        capabilities:
          add: [ "SYS_ADMIN" ]
        privileged: false
```

2.2 Why Is "No AllocTracer Symbols Found. Are JDK Debug Symbols Installed?" Displayed?

Problem

The memory Profiler depends on the JDK symbol information. If JDK does not contain symbol information, the following error may occur:

[ERROR] xxxx Failed to start Continuous Profile Collector

[ERROR] xxxx No AllocTracer symbols found. Are JDK debug symbols installed?

Solution

If the Java process runs in a container environment and the preceding error is reported or no data is displayed, the possible cause is that the Alpine base image is used. The Alpine base image removes JDK debug symbols to reduce the size, which affects the Profiler function. You are advised to install debug symbols for JDK in the base image (for some JDK versions, the installation may fail due to the lack of debug symbol packages) or use a non-Alpine base image.

If the application is deployed on a CentOS physical machine, perform the following steps to install debug symbols:

Step 1 Run the following command to check whether the debuginfo source has been configured:

```
yum repolist all | grep -i debug
```

Check whether the source configuration is as follows:

```
debuginfo/7/x86_64 CentOS-7 - debuginfo - mirrors.huaweicloud.com Enabled: 8,760
```

If the debuginfo source has not been configured, go to **step 2** to add the configuration. If the debuginfo source has been configured, go to **step 3**.

Step 2 Add the debuginfo configuration to the Yum configuration file.

```
vi /etc/yum.repos.d/CentOS-Base.repo
[debuginfo]
name=CentOS-$releasever - debuginfo - mirrors.huaweicloud.com
failovermethod=priority
baseurl=http://mirrors.huaweicloud.com/centos-debuginfo/$releasever/$basearch/
gpgcheck=1
enabled=1
gpgkey=http://mirrors.huaweicloud.com/centos/RPM-GPG-KEY-CentOS-Debug-7
```


After saving the configuration, run the following command:

```
yum clean all
yum makecache
yum-config-manager --enable debuginfo
```

Step 3 Install the symbol information corresponding to the JDK version.

```
JDK 8
yum list installed|grep openjdk    #Query the JDK version.
java-1.8.0-openjdk.x86_64          1:1.8.0.332.b09-1.el7_9          @updates
debuginfo-install -y java-1.8.0-openjdk      #Install the symbol information.
JDK 11
yum list installed|grep openjdk    #Query the JDK version.
java-11-openjdk.x86_64             1:11.0.15.0.9-2.el7_9            @updates

debuginfo-install -y java-11-openjdk      #Install the symbol information.
```

Step 4 Check whether the JDK symbol information is successfully installed.

```
gdb $JAVA_HOME/lib/server/libjvm.so -ex 'info address UseG1GC'
gdb /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.332.b09-1.el7_9.x86_64/jre/lib/amd64/server/libjvm.so -ex 'info
address UseG1GC'
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.332.b09-1.el7_9.x86_64/jre/lib/amd64/server/
libjvm.so...Reading symbols from /usr/lib/debug/usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.332.b09-1.el7_9.x86_64/jre/lib/amd64/server/libjvm.so.debug...done.
done.
```

If the installation is correct, the following information is displayed:

```
Symbol "UseG1GC" is static storage at address 0x107f993.
```

If the installation fails, the following information is displayed:

```
No symbol "UseG1GC" in current context.
```

----End

2.3 Why Is "perf_event mmap failed..." Displayed?

Symptom

This error is generally displayed in the standard output of JVM. When a user uses the Profiler function to collect CPU hotspot samples, both native (Linux kernel + JVM + C/C++) and Java stacks will be collected. To collect native stacks, the system will implement memory map (MMap) for the perf_event file descriptor (FD) of each Java thread. The Linux kernel limits the total memory of MMap related to perf_event (516 KB by default). When there are a large number of Java threads, the limit is triggered and the warning "perf_event mmap failed..." is printed in the Java standard output. This warning will not impact Java running or services. The only impact is that the native stacks cannot be viewed in the flame graph. Generally, when locating CPU hotspot problems, you only need to view the Java method stacks. Therefore, this warning can be ignored.

Solution

To solve this warning, perform the following steps:

Step 1 Run the following commands on the server:

```
echo 1028 > /proc/sys/kernel/perf_event_mlock_kb
```

The default threshold is 516. Gradually increase the value until the warning is not displayed. If possible, the value should meet the following requirement: 8 × N + 4 (where N is a natural number). For example, 516 = 512 + 4 or 1,028 = 1,024 + 4.

Step 2 Restart the Docker. The problem can then be solved.

----End

2.4 Why Is "libz.so.1: version `ZLIB_1.2.9' not found" Displayed?

Symptom

Error: java.lang.UnsatisfiedLinkError: /jre/lib/amd64/libfontmanager.so: /apm-javaagent-profiler/apm-javaagent/native-agent/x86/libz.so.1: version `ZLIB_1.2.9' not found (required by /usr/lib64/libpng16.so.16)

This error occurs in the service log when the service program calls the method in **libfontmanager.so** of JDK. This **.so** depends on the libz.so.1 of version 1.2.9, which is incompatible with the libz.so.1 version in **javaagent**.

Solution

1. Manually delete the **libz.so.1** file from **javaagent**. In this way, the service program uses the **libz.so.1** file in the system directory.
2. Upgrade the Agent version to 2.4.14-profiler or later.

3 CCE Access FAQs

3.1 CCE Access FAQs and Solutions

Agent Fails to Be Installed Through CCE, and No Log Is Printed

Cause: **JAVA_TOOL_OPTIONS** does not take effect. The user starts the service using **sudo**. The script started by using **sudo** ignores all environment variables.

Solution: Run the following command to open the file and change **Defaults env_reset** to **Defaults !env_reset**:

```
/etc/sudoers
```

Agent Can Be Started Through CCE, But It Goes Offline Later and No Data Is Reported

Cause: The **adt** user is specified to conduct the startup in the script. However, when the script is started from CCE, the **root** user executes the startup script and then the user is switched to the **adt** user. As a result, the Agent can be started from CCE, but it goes offline later and no data is reported.

- **apm.log.0** exists in the **root** directory of the **root** user.
- JavaAgent automatically goes offline. There is no heartbeat or log.
- If the user deletes the corresponding instance when the Agent goes offline, it seems that no application is connected.

Solutions:

- Workaround: Switch to the user that matches the script, for example, **adt**, in the container, and then execute the startup script. In this way, the Agent can be started and data can be reported.
- Solution: Modify the permission of the **adt** user to write logs to the **/home** directory, or manually specify **log.path** in the **apm.config** file.

No Data Is Found When CCE is Connected to APM Through Fuxi

Check for the **apm.log** file. If the file does not exist, the Agent is not connected during startup.

Check the main configuration of Fuxi.

```
"JAVA_TOOL_OPTIONS": "-javaagent:/paas-apm2/javaagent/apm-javaagent/apm-javaagent.jar"
```

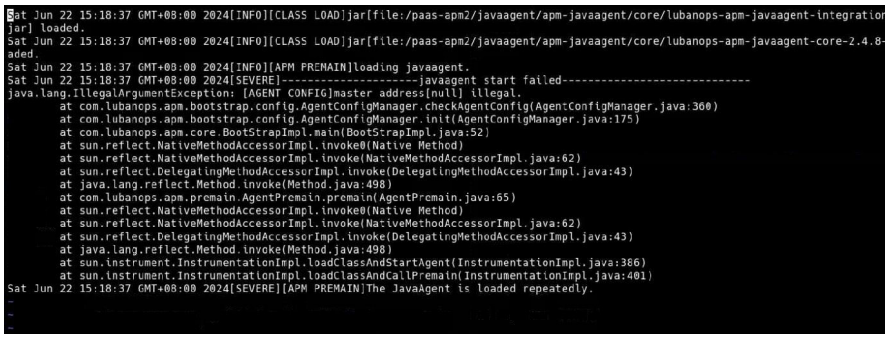
Check whether the **JAVA_TOOL_OPTIONS** environment variable exists in container 1 on CCE.

Check whether container 2 exists on CCE, check the instance startup event, and check whether the SWR image of JavaAgent has been loaded.

Agent Fails to Be Connected Through CCE (Log Shows That the Master Address Is Null)

During connection through CCE, select the Agent of the noroot version. For details, see **Installing Agents for the Java Applications Deployed in CCE Containers**.

"master address illegal" Is Reported When the Agent Is Connected to APM in CCE Access Mode



Solution: Security hardening has been performed on the container. To solve the problem, switch to the Agent of the noroot version.

3.2 Why Is There No Monitoring Data Displayed on APM After the JavaAgent Is Enabled on CCE?

This is because the JavaAgent is of an old version or started using Tomcat.

To solve the problem, do as follows:

1. On the APM console, subscribe to APM 2.0 for free. (Ten Agents are free of charge.) For details, see **Enabling APM 2.0**.
2. Alternatively, purchase the enterprise-edition APM. For details, see **Subscribing to Enterprise-Edition APM**.
3. Use the JavaAgent of the latest version. After the container is restarted, the APM page is displayed properly.

4 Debugging Diagnosis FAQs

4.1 Method Analysis of Debugging Diagnosis Does Not Support Drilldown for Nested Calls of Method Overloading

Symptom

Step 1 Log in to the **APM console**.

Step 2 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 3 In the navigation pane, choose **Application Monitoring** > **Metrics**.

Step 4 In the tree on the left, click next to the target environment.

Step 5 Click the **Debugging Diagnosis** tab.

Step 6 Click **Method Analysis**. The **Method Analysis** page is displayed.

Step 7 Enter class name **com.example.hello.helloController**, select method name **boy**, and click **Confirm**.

Step 8 Click **Drill Down** corresponding to **org.apache.http.impl.client.CloseableHttpClient.execute #-1**.

Step 9 Click **Drill Down** corresponding to **org.apache.http.impl.client.CloseableHttpClient.execute #108**. The system does not respond.

----End

Trigger Conditions

When method overloading exists (that is, multiple functions with the same name exist in the class and nested calls exists), only the first called method can be

traced. That means if the system calls an execute method and then calls its nested execute method, the nested method cannot be drilled down.

Root Cause and Constraint

The **watch** and **trace** commands in the frontend do not specify overload method parameters. Therefore, accurate method information cannot be returned during debugging diagnosis. Only the information about the first method is returned. If a nested execute method is called, its information cannot be returned.

Debugging diagnosis does not support drilldown for nested overloaded methods.

5 Web Monitoring FAQs

5.1 How Can I Connect WeChat?

Different from other applets, WeChat supports the following two access methods.

Method 1: Integrating the SDK using npm

1. Ensure that the project has the **package.json** file. If it does not exist, run the following command in the **root** directory of the project to create one:
   ```
   npm init
   ```
2. Run the SDK installation command to install the SDK software package:
   ```
   npm i apm-mini-sdk
   ```
3. Use the developer tool to build the npm library file of the current project.
4. Use **import agent from 'apm-mini-sdk'** in the **app.js** file.

Method 2: Importing a file

1. Run the SDK installation command to install the SDK software package:
   ```
   npm i apm-mini-sdk
   ```
2. Find the **app.js** file (**node_modules** > **apm-mini-sdk** > **app.js**) in the SDK folder, copy it from **node_module** to the **root** directory, and rename it.

6 Consultation FAQs

6.1 Why Does APM Metric Collection Fail?

1. You can check metric data several minutes after you connect APM Agents.
2. If data collection stops, check the following:
   - Instance level: Agents are stopped on the **Instance** tab page.
   - Monitoring item level: Monitoring items are manually disabled on the **Monitoring Item** tab page.
   - Global level: The **Stop Collecting Data Through Bytecode Instrumentation** option is enabled on the **General Configuration** page of APM.
3. If no data is collected for a long time, check the following:
   - Java 9 prompts that the **sql.time** class cannot be found.

     Cause analysis: Agents are developed using JDK 1.7. However, after Java 9 modularization, no SQL package is provided by default.

     Occurrence: This problem occurs under certain conditions.

     Workaround: Ensure that the component can proactively import **java.sql** to **module-info.java**.
   - Java 11 prompts that "Caused by: java.lang.NoClassDefFoundError: sun/misc/Unsafe class cannot be found."

     Cause analysis: Agents are developed using JDK 1.7, but the Java 11 Unsafe class is categorized to a different package.

     Occurrence: This problem occurs inevitably.

     Workaround: Ensure that the application can proactively import **jdk.unsupported** to **module-info.java**.
   - Java 9 reports an illegal reflective access alarm. (This problem will be solved in versions later than Java 9.)

     Workaround: Set **illegal-access** to **warn** or delete this option.

6.2 Which Logs Can I View in APM?

APM allows you to view component exception logs, trace logs, and **apm.log**.

Component Exception Logs

This function monitors application exception logs. Take the monitoring of Java exception logs as an example. Once you use the log system to print logs, they will be collected by APM. The exception collection type varies according to the collector type.

Step 1 Log in to the **APM console**.

Step 2 Click  on the left and choose **Application** > **Application Performance Management**.

Step 3 In the navigation pane, choose **Application Monitoring** > **Metrics**.

Step 4 In the tree on the left, click  next to the target environment.

Step 5 Click the **Exception** tab. By default, exception logs of all instances are displayed. For details about the metrics, see **Table 6-1**. For details, see **Exceptions**.

Table 6-1 Exception and log parameters

Metric Set	Parameter	Description
Exception	Class	Exception class
	Exception Type	Exception type
	Log Type	Exception log type
	Total Exceptions	Number of times that an exception has occurred
	Message	Message returned when the exception occurred
	Error Stack	Error stack
	Error Trace ID	Information about the error trace
Log Version	Log Type	Log type
	Version	Version

----End

Trace Logs

This function is used to search for span information, that is, the root event of a node. A trace can be found in multiple environments. For example, in the scenario

where service A calls service B and then calls service C, the same trace may be found from services A, B, and C.

To view trace logs, perform the following operations:

1. On the LTS console, configure collection. For details, see **section "Ingesting ECS Text Logs to LTS" in the Log Tank Service (LTS) User Guide**.
2. Click **View Logs** to go to the LTS page based on the trace ID. For details about how to search for logs on the LTS console, see **Searching for Logs** .
3. If the function of associating trace IDs with logs is not enabled, a warning dialog box is displayed. Click **Associate Now** to go to the log association settings page. For details about how to associate logs, see **Component Settings**.

apm.log

Some issues may occur during JavaAgent connection. To quickly locate and rectify these issues, view **apm.log** as follows:

- Log in to the node.
  ```
  cd /apm
  ```
- If **apm.log** is not in the **apm** directory, run the following command to search for the **apm.log** file:
  ```
  find / -name apm.log
  ```

6.3 Is Huawei's APM Android SDK Compatible with Other Similar Products?

APM usually implements bytecode instrumentation based on the ASM framework. This technology allows developers to dynamically modify the bytecode of an application to monitor its performance without changing the source code.

However, if multiple APM tools are installed, the code will be instrumented multiple times. Different tools may conflict with each other, causing compilation errors and performance problems. For example, one tool may modify the bytecode of a method, while another tool may attempt to modify the same bytecode. Such conflicts may cause runtime exceptions or inconsistent behavior. In addition, frequent instrumentation may increase the application startup time and running overhead, affecting performance.

Therefore, you are advised to install only one APM tool in a project to ensure application stability and performance.

6.4 What Is Apdex?

Apdex is an open standard developed by the Apdex alliance. It defines a standard method to measure application performance. The Apdex standard converts the application response time into user satisfaction with application performance in the range of 0 to 1.

- Apdex principle

Apdex defines the threshold "T" for application response time. "T" is determined based on performance expectations. Based on the actual response time and "T", user experience can be categorized as follows:

Satisfied: indicates that the actual response time is shorter than or equal to "T". For example, if "T" is 1.5s and the actual response time is 1s, user experience is satisfied.

Tolerable: indicates that the actual response time is greater than "T", but shorter than or equal to "4T". For example, if "T" is 1s, the tolerable upper threshold for the response time is 4s.

Frustrated: indicates that the actual response time is greater than "4T".

- Apdex calculation

  In APM, the Apdex threshold is the maximum response time that makes users satisfied. The application response latency is the service latency. The Apdex value ranges from 0 to 1 and is calculated as follows:

  Apdex = (Number of satisfied samples + Number of tolerable samples × 0.5)/ Total number of samples

6.5 What Is APM's Metric Data Sampling Policy?

After you enable data collection, APM only collects application performance metrics and tracing data. Your personal privacy data will not be collected. For details, see **Data Collection**.

APM Agents collect data about applications, infrastructure, and user experience non-intrusively.

Metric data is collected regularly. The default collection period is 1 minute.

6.6 Are APM Agents Compatible with Other Agents Such as Pinpoint?

No.

Generally, APM implements bytecode instrumentation based on the ASM framework. Installing two Agents means two instrumentation operations on your code. Code instrumentation mechanisms vary according to products. If you install Agents of different products, code conflicts may occur, affecting performance.

Therefore, do not install APM Agents together with other Agents.

