

# Application Performance Management 2.0

## Best Practices

Issue 01  
Date 2025-03-28



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI 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.

# Contents

1 Best Practices.....

2 Locating the Causes of Request Errors.....

3 Searching for Span Information.....

4 Connecting to APM.....

4.1 Connecting On-premises Services to APM.....

4.2 Connecting to APM Through a Public Network.....

4.3 Connecting IDE to APM.....

5 Associating Traces with Logs.....

6 Embedding APM Pages into a Self-built System.....

7 Locating Performance Problems Using APM Profiler.....

8 Locating Out of Memory Problems Using Profiler.....

9 Encrypting AK/SK for Deploying an APM Agent in a CCE Container.....

10 Suggestions on APM Security Configuration.....

1

3

5

11

11

12

14

18

20

22

29

31

33

# 1 Best Practices

This document summarizes Application Performance Management (APM) best practices and provides solutions and operation guide to help you easily use APM.

**Table 1-1** APM best practices

Best Practice	Description
<b>Locating the Causes of Request Errors</b>	Sudden request increases or load changes may easily cause application performance problems. APM has powerful analysis tools for cloud application diagnosis. It displays application statuses, call processes, and user operations through topologies and tracing, so that you can quickly locate and resolve faults and performance bottlenecks.
<b>Searching for Span Information</b>	In the distributed architecture, the calls between microservices are complex. If it takes much time to respond to external requests or some requests become abnormal, you can specify a trace ID or set other criteria on the <b>Tracing</b> page to check trace details.
<b>Connecting to APM</b>	<b>Connecting On-premises Services to APM</b> You cannot connect on-premises services to APM using Direct Connect. However, you can do that using a proxy.
	<b>Connecting to APM Through a Public Network</b> Connect to APM through a public network.
	<b>Connecting IDE to APM</b> Connect IDE to APM.
<b>Associating Traces with Logs</b>	Associate trace IDs with logs in Log Tank Service (LTS). If a fault occurs, you can quickly find out the logs based on the associated trace IDs for troubleshooting.

Best Practice	Description
<a href="#">Embedding APM Pages into a Self-built System</a>	APM pages can be embedded into a self-built system. Specifically, create a custom identity broker through the federation proxy mechanism of Identity and Access Management (IAM) and embed a login link into your self-built system. You can then check APM pages on your system without the need to log in to Huawei Cloud.
<a href="#">Encrypting AK/SK for Deploying an APM Agent in a CCE Container</a>	Encrypt the AK/SK when deploying an APM Agent on CCE.
<a href="#">Suggestions on APM Security Configuration</a>	This document provides guidance for enhancing the overall security of APM. You can continuously evaluate the security of APM and combine different security capabilities provided by APM.

# 2 Locating the Causes of Request Errors

## Background

When the number of external requests increases sharply or the load changes abruptly, application performance problems occur frequently, for example, requests cannot be quickly responded or properly handled. Quickly identifying, locating, and handling these problems are required in routine inspection.

APM has powerful analysis tools for cloud application diagnosis. It displays application statuses, call processes, and user operations through topologies and tracing, so that you can quickly locate and resolve faults and performance bottlenecks.

For example, you can view the call relationships between services and quickly locate abnormal instances through topologies. You can also drill down to services and determine root causes based on method tracing.

## Applicable Scenarios

- Routine inspection, covering application metrics such as latency, throughput, and number of errors
- Quick locating of error calls

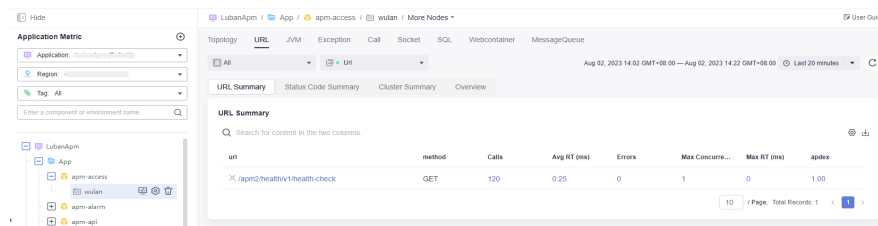
## Procedure

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

**Step 2** In the navigation pane, choose **Application Monitoring > Metrics**.

**Step 3** Click the **URL** tab. On the page that is displayed, check metrics such as the number of calls, number of errors, and latency.

**Figure 2-1** Going to the URL page



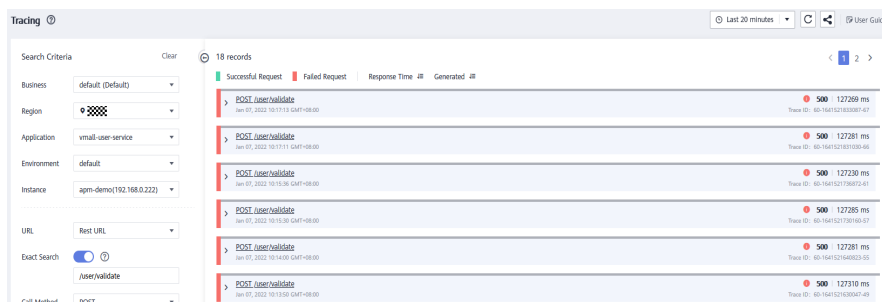
**Step 4** Click the abnormal URL to go to the tracing page.

**Figure 2-2** URL details

Interface-based Summary - Last Data Time 2022-01-07 10:16:35													
url	method	Number of L...	Average ...	umber of...	Maximu...	Slowest L...	0ms-10ms	10ms-10...	100ms-5...	500ms-1s	1s-10s	10s-n	
/user/login	POST	14	128271.36	14	4	128332	0	0	0	0	0	14	
/user/validate	POST	14	127265.21	14	4	127354	0	0	0	0	0	14	

**Step 5** Locate error or slow traces.

**Figure 2-3** Viewing traces



**Step 6** Click the corresponding URL to obtain the trace details and determine the root cause.

**Figure 2-4** Trace details



-----End

# 3 Searching for Span Information

## Background

In the distributed architecture, the calls between microservices are complex. If it takes much time to respond to external requests or some requests become abnormal, you can specify a trace ID or set other criteria on the **Tracing** page to check trace details.

## Procedure

- Step 1 Log in to the APM console.
- Step 2 In the navigation pane, choose **Application Monitoring** > **Tracing**.
- Step 3 Enter the following search criteria and click **Search Trace**.

Figure 3-1 Tracing search result

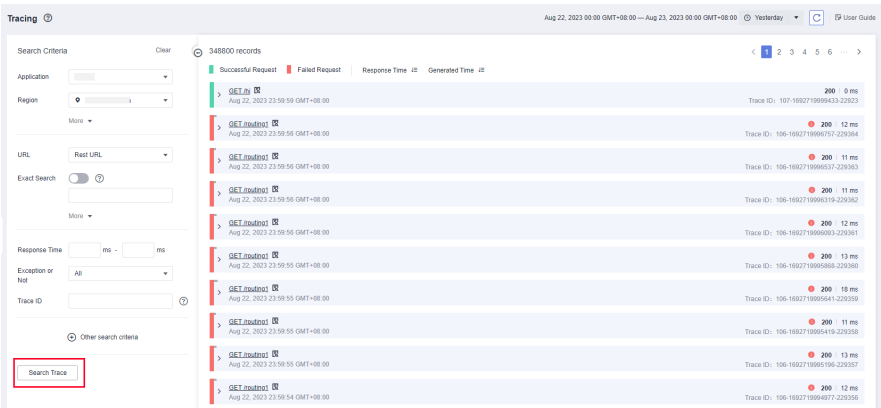


Table 3-1 Search criteria of traces

Search Criterion	Description	Mandatory
Application	Application to which the trace belongs.	Yes
Region	Region where the trace is located.	Yes



Search Criterion	Description	Mandatory
Component	Component to which the trace belongs.	No
Environment	Environment to which the trace belongs.	No
Instance	Instance to which the trace belongs.	No
URL	Trace URL, which can be a REST or real URL. A REST URL contains a variable name, for example, <code>/apm/get/{id}</code> . A real URL indicates an actual URL.	No
Exact Search	Whether to perform exact match on URLs. If this option is selected, exact match is performed. If this option is not selected, fuzzy match is performed.	No
Call Method	HTTP method of the trace.	No
Status Code	HTTP status code returned by the trace.	No
Response Time	Response time range of the trace. You can specify the minimum and maximum response time to search for traces or leave them empty.	No
Exception or Not	Whether to filter the traces that are regarded as exceptions.	No
Trace ID	ID of a trace. If you specify this parameter, other search criteria become invalid and the search will be performed based on the trace ID you specify.	No

**Step 4** Click **Other search criteria**. **Custom Parameter**, **Global Trace ID**, and **Application Code** are displayed.

Figure 3-2 Other search criteria

Search Criteria

Clear

Application

Region

More

URL

Rest URL

Exact Search

More

Response Time

ms

-

ms

Exception or Not

All

Trace ID

Custom Parameter

Global Trace ID

Application Code

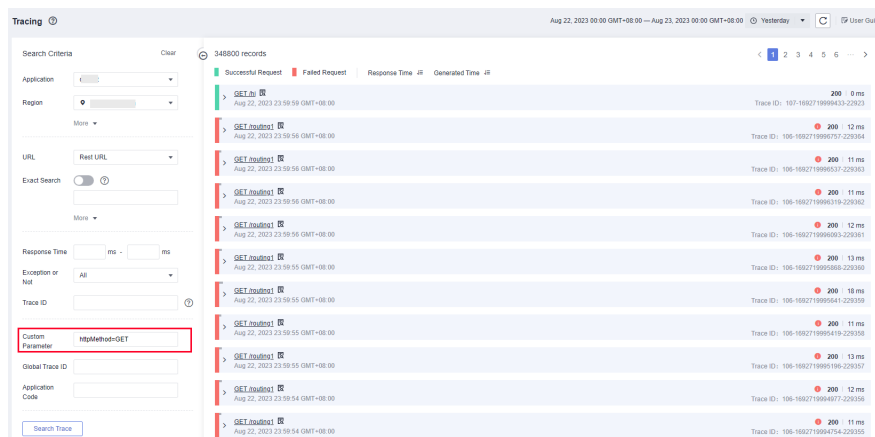
Table 3-2 Search criteria of traces

Search Criterion	Description	Mandatory
Custom Parameter	If you have configured <b>Key for Header Value Interception</b> , <b>Key for Parameter Value Interception</b> , and <b>Key for Cookie Value Interception</b> for URL monitoring, you can set <b>key=value</b> to search.	No
Global Trace ID	Global ID of a trace. If you specify this parameter, other search criteria become invalid and the search will be performed based on the trace ID you specify.	No
Application Code	If you have configured <b>Service Code Length</b> , <b>Key for Service Code Interception</b> , and <b>Normal Service Code</b> , corresponding application codes will be collected. You can search information based on the application codes.	No

- **Custom Parameter**


## Usage Instructions

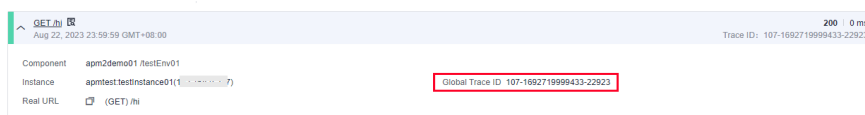
- a. Configure **Key for Header Value Interception**, **Key for Parameter Value Interception**, and **Key for Cookie Value Interception** for URL monitoring. For details, see [Configuring the URL Monitoring Item](#).
- b. In the **Custom Parameter** text box, set the parameters and values.
- c. Click **Search Trace**. The results are displayed on the right.

**Figure 3-3** Results of querying traces based on the custom parameters

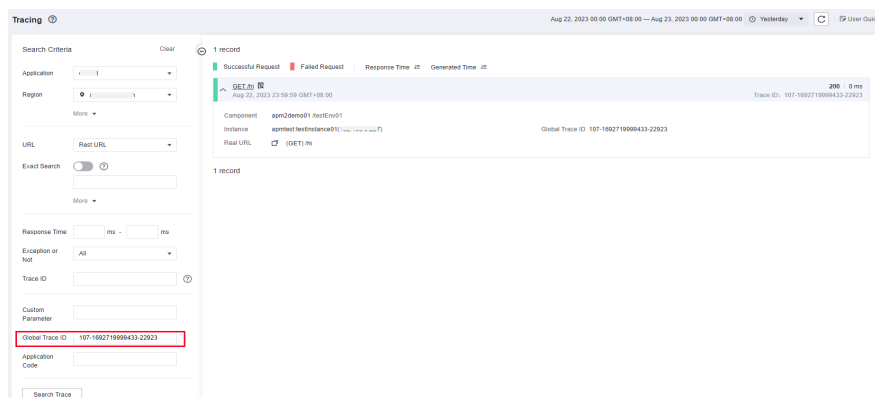
- **Global Trace ID**

## Usage Instructions

- a. Click  next to the target trace to view the global trace ID.

**Figure 3-4** Obtaining the global trace ID

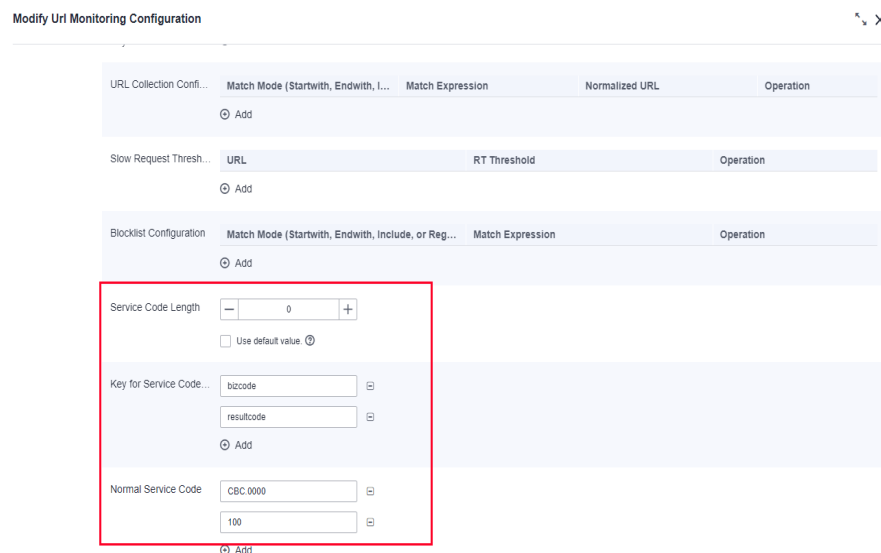
- b. In the **Global Trace ID** text box, enter the global trace ID.
- c. Click **Search Trace**. The results are displayed on the right.


**Figure 3-5** Results of querying traces based on the global trace ID

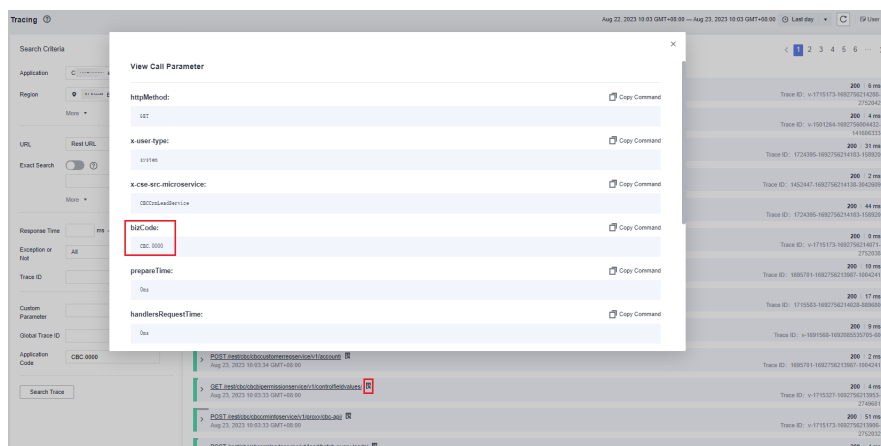
- **Application Code**

## Usage Instructions

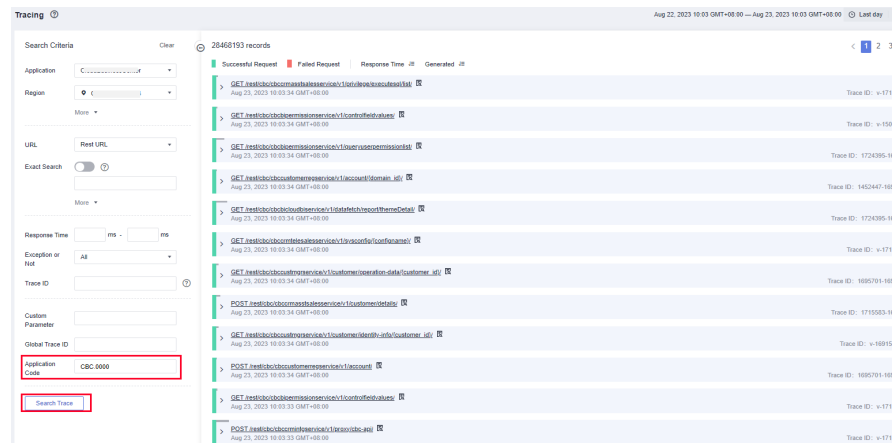
- a. Configure **Service Code Length**, **Key for Service Code Interception**, and **Normal Service Code** for URL monitoring. For details, see [Configuring the URL Monitoring Item](#).

**Figure 3-6** URL monitoring

- b. In the navigation pane, choose **Application Monitoring > Tracing**.
- c. Click  to view the value of the service code, which corresponds to the application code.

**Figure 3-7** Viewing the service code

- d. In the **Application Code** text box, enter the application code.

**Figure 3-8** Searching for the traces corresponding to the code

- e. Click **Search Trace**. The results are displayed on the right.

----End

# 4 Connecting to APM

---

## 4.1 Connecting On-premises Services to APM

### Background

You cannot connect on-premises services to APM using Direct Connect. To access APM, configure a proxy.

### Configuration Method

If the network between your host and APM is disconnected, configure a proxy.

**Step 1** Configure a proxy. You need to log in to the AOM 2.0 console to configure the proxy. For details, see [Configuring a Proxy Area and Proxy](#).

**Step 2** Configure the JavaAgent.

1. Download the JavaAgent package to any directory of the host to be connected to APM.

Example command:

```
curl -O https://xxx/apm-javaagent-x.x.x.tar
Download Agent 2.4.1: curl -k https://apm2-javaagent-xx-xx-x.obs.xx-xxx-4.xxxx.xxx/
apm_agent_install2.sh -o apm_agent_install.sh && bash apm_agent_install.sh -ak {APM_AK} -sk
{APM_SK} -masteraddress https://xx.xx.xx.xx:41333 -obsaddress https://apm2-javaagent-xx-xxxx-
x.obs.xx-xxxx-x.xxxx.xxx -version 2.4.1; history -cw; history -r
```

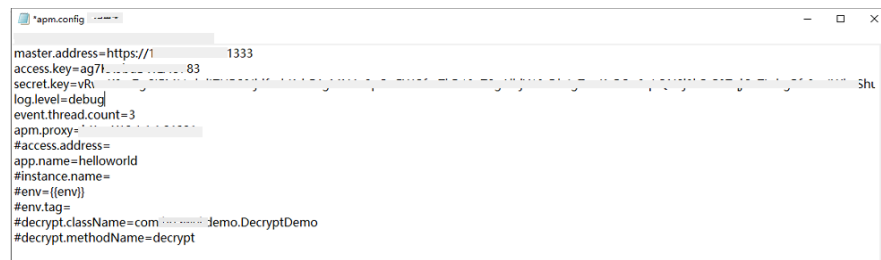
2. Run the **tar** command to decompress the JavaAgent package.

Example command:

```
tar -xvf apm-javaagent-x.x.x.tar
```

3. Modify the **apm.config** file in the JavaAgent package. Add **apm.proxy** to the configuration file, as shown in the following figure.

**Figure 4-1** Configuration file



- Agents of 2.4.1 and later support access through a proxy. Format: **apm.proxy=ip:port** (Obtain *ip:port* from the AOM console.)
- To obtain an AK/SK, see [Access Keys](#).
- To obtain the **master.address**, see [Access Addresses](#).

**Step 3** Restart the application.

1. Modify the startup script of the Java process.

Add the path of the **apm-javaagent.jar** package and the component name of the Java process to the end of the Java command in the service startup script.

Example of adding **-javaagent** parameters:

```
java -javaagent:/xxx/apm-javaagent/apm-javaagent.jar=appName={appName}
```

2. Restart the application.

----End

## 4.2 Connecting to APM Through a Public Network

### Prerequisites

1. You have purchased an ECS as the jump server.
2. You have bound an EIP to the ECS.
3. You are advised to use **CentOS 6.5 64bit** or later images. The minimum specifications are **1 vCPU | 1 GB** and the recommended ones are **2 vCPUs | 4 GB**.
4. You are advised to use iptables for the jump server to forward data.

### Procedure

Purchase an ECS as the jump server and perform the following operations:

**Step 1** Log in to the ECS and modify its security group rules.

1. On the ECS details page, click the **Security Groups** tab.
2. Click a security group name and click **Modify Security Group Rule**.
3. On the security group details page, click the **Inbound Rules** tab and then click **Add Rule**. On the page displayed, add a security group rule by referring to [Table 4-1](#).

**Table 4-1** Security group rule

Direction	Protocol	Port	Description
Inbound	TCP	41333,41335	JavaAgent will send data to the jump server through the listed ports.

**Step 2** Obtain the APM report address. For details, see [Access Address \(master.address\)](#).

**Step 3** Log in to the jump server as the **root** user and run the iptables forwarding command.

If the iptables service does not exist, run the following commands to install it:

```
yum install iptables-services
systemctl stop firewalld.service
systemctl disable firewalld.service
systemctl mask firewalld.service
```

1. Enable data forwarding.

```
# Edit the file.
vim /etc/sysctl.conf
# Add the following content:
net.ipv4.ip_forward=1
# Enable data forwarding.
sysctl -p
```

2. Forward the data from the port of the local host (jump server) to the port for reporting data to APM.

```
# Edit the file.
vim /etc/sysconfig/iptables
# *Add filters.
-A INPUT -p tcp -m state --state NEW -m tcp --dport 41333 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 41335 -j ACCEPT
# *Add NAT rules.
-A OUTPUT -p tcp --dport 41333 -j DNAT --to-destination {IP address of the host that reports data to APM}:41333
-A PREROUTING -p tcp --dport 41333 -j DNAT --to-destination {IP address of the host that reports data to APM}:41333
-A POSTROUTING -d {IP address of the host that reports data to APM}/32 -p tcp --dport 41333 -j SNAT --to-source {IP address of the jump server}

-A OUTPUT -p tcp --dport 41335 -j DNAT --to-destination {IP address of the host that reports data to APM}:41335
-A PREROUTING -p tcp --dport 41335 -j DNAT --to-destination {IP address of the host that reports data to APM}:41335
-A POSTROUTING -d {IP address of the host that reports data to APM}/32 -p tcp --dport 41335 -j SNAT --to-source {IP address of the jump server}

# If the following rule exists, delete it:
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
```

3. Restart iptables.

```
systemctl restart iptables
```

4. Check whether port forwarding is successful.

```
curl -kv https://{IP address of the jump server}:41333
curl -kv https://{IP address of the jump server}:41335
```

**Step 4** Modify the **apm.config** file in the JavaAgent package.

```
master.address=https://{Public IP address of the jump server}:41333
access.address={Public IP address of the jump server}:41335
```



**Step 5** Restart the application.

-----End


## 4.3 Connecting IDE to APM

### Prerequisite

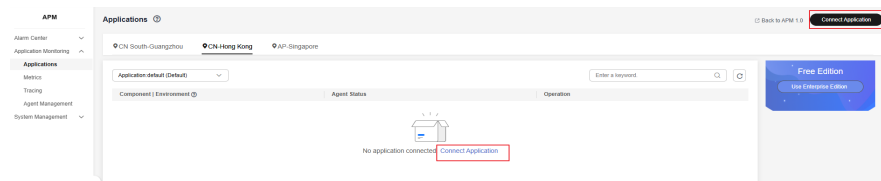
The public network has been connected.

### Procedure

**Step 1** Download the APM Agent.

1. Log in to the management console.
2. Click  on the left and choose **Application > Application Performance Management**.
3. In the navigation pane, choose **Application Monitoring > Applications**.

**Figure 4-2** Connecting an application



4. On the displayed page, click **Connect Application**.

**Figure 4-3** Connecting an application

Basic Info

Region

AP-Hong-Kong

Application

default

Create Application

Select Access Mode

Access Mode

Enhanced Agent

Code-level performance analysis and riche...

Backend Language

Java

Data Access

VM Access

OCE Access

1. Download and install JavaAgent.

- In the navigation pane on the left, choose **System Management > Access Keys**. On the displayed page, obtain an AK and SK for the JavaAgent. For details, see [Access Keys](#).
- Replace *APM\_AK* and *APM\_SK* in the installation command with the AK and SK obtained from the **Access Keys** page.

**Step 2** Run the **git bash** command. In the Agent directory of drive **D** on the local PC, run the copied command to install JavaAgent.

**Step 3** Change the values of **master.address**, **access.address**, and **business** in the **apm.config** file.

### Figure 4-4 Modifying the configuration file

```
master.address=https://100.94.71.117:83
access.address=100.94.71.117:83
access.key=mq9g1qq5r228oswp
secret.key=8e3hkak3f3gs8fcf2k3dwy86tpgiciu
log.level=info
event.thread.count=3

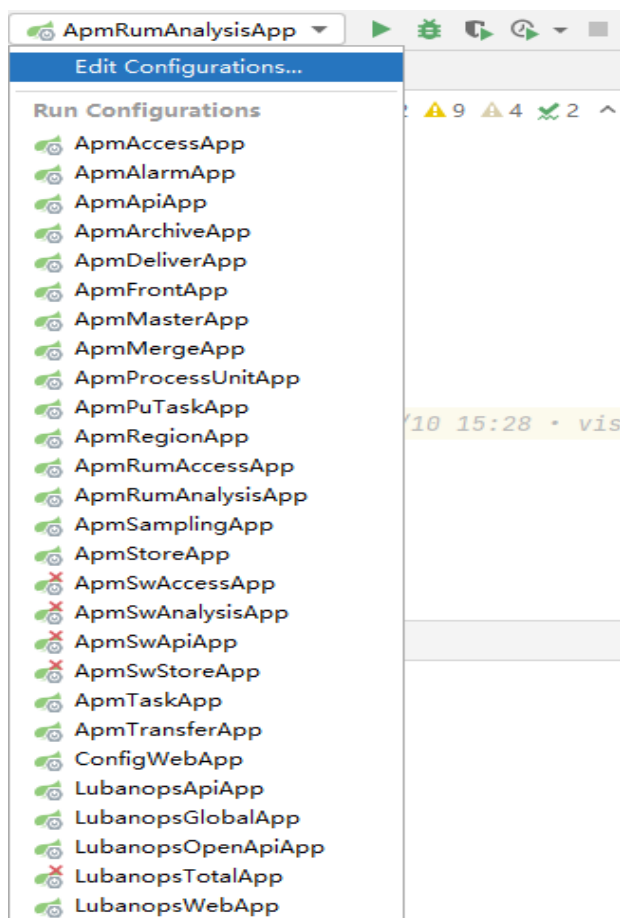
#access.address=
#app.name={{app_name}}
#instance.name=
#env={{env}}
#env.tag=
#business=APM
#sub.business={{sub_business}}
#env.secret=

collect.body=true
collect.sql.result=true
collect.method.body=true
collect.httpClient.body=true
transaction.enable=true
```

**Step 4** Click the drop-down list and choose **Edit Configurations**.

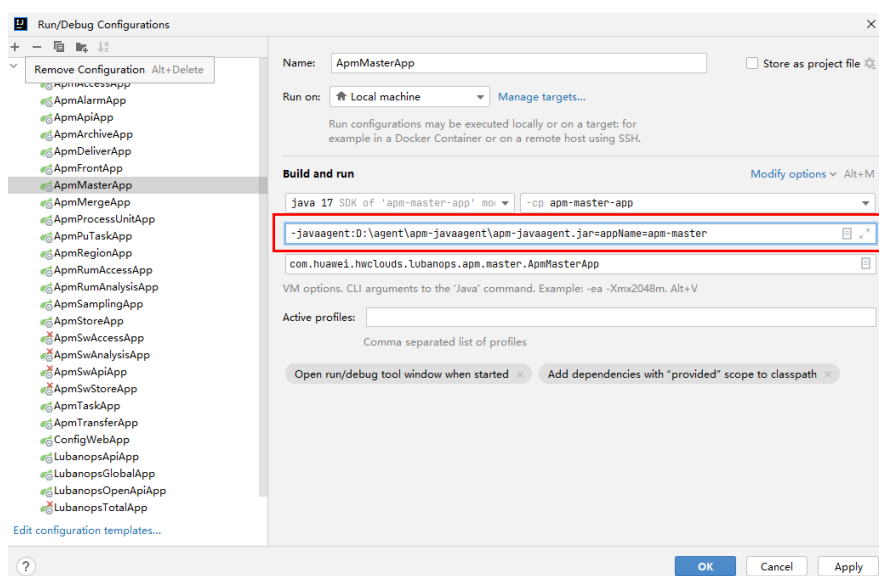
1. Modify the running/debugging configuration of IDE.

**Figure 4-5** Modifying the running/debugging configuration



2. On the **Run/Debug Configurations** page, choose **ApmMasterApp** from the navigation pane. Add `-javaagent:D:\agent\apm-javaagent\apm-javaagent.jar=appName=apm-master` to **Build and run**.

**Figure 4-6** Modifying the configuration under "Build and run"



3. Click **OK**.

**Step 5** Restart the service. If **APM** is displayed in the **Application** drop-down list, the connection is successful.

----End

# 5 Associating Traces with Logs

## Application Scope

Common log frameworks, such as Logback and Log4j.

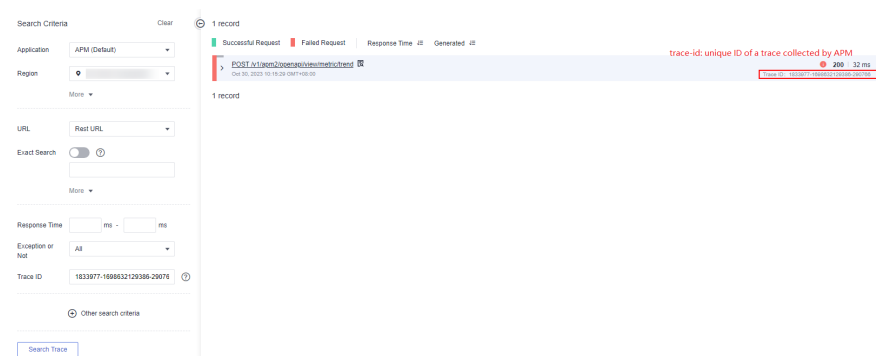
## Example

```
<property name="LOG_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS}} |  
gtraceid: %X{apm-gtraceid} | traceid: %X{apm-traceid} | spanId: %X{apm-  
spanid}">  
  
</property>
```

## Trace Parameters

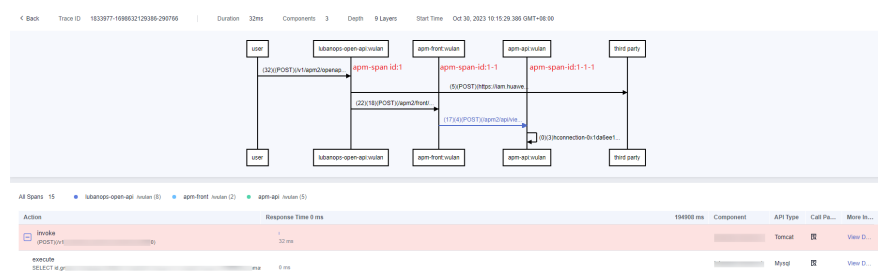
1. **apm-traceid**: unique ID of a trace collected by APM.

Figure 5-1 Unique ID of a trace



2. **apm-gtraceid**: unique ID of a trace which is not sampled.  
APM has a certain sampling ratio. The **apm-gtrace-id** parameter is used to uniquely identify a trace that is not sampled.
3. **apm-spanid**: ID of a microservice called in a trace. Example:

### Figure 5-2 Calls between microservices



# 6 Embedding APM Pages into a Self-built System

## Background

APM pages can be embedded into a self-built system. Specifically, create a custom identity broker through the federation proxy mechanism of Identity and Access Management (IAM) and embed a login link into the customer's self-built system. You can then check APM pages on your system without the need to log in to Huawei Cloud.

## Prerequisite

You have created a custom identity broker and created a `FederationProxyUrl`. For details, see [Creating a FederationProxyUrl Using a Token](#).

## Procedure

After creating a custom identity broker, perform the following steps to embed a page:

- Step 1** Change the value of `console_service_url` in `FederationProxyUrl` to the address of a target service module.

Example of `console_service_url`:

Service Module	Example URL
Application Monitoring - Metrics	<code>https://console.huaweicloud.com/apm2/?region={regionId}&amp;cfModuleHide=header_sidebar_floatlayer#/console/appindex/business/detail?leftMenuCollapsed=true</code>
Application Monitoring - Tracing	<code>https://console.huaweicloud.com/apm2/?region={regionId}&amp;cfModuleHide=header_sidebar_floatlayer#/console/appchain?leftMenuCollapsed=true</code>

Service Module	Example URL
Link Trace - Metrics	<code>https://console.huaweicloud.com/apm2/?region={regionId}&amp;cfModuleHide=header_sidebar_floatlayer#/console/trace/metric/environment/view?leftMenuCollapsed=true</code>
Link Trace - Tracing	<code>https://console.huaweicloud.com/apm2/?region={regionId}&amp;cfModuleHide=header_sidebar_floatlayer#/console/trace/chain?leftMenuCollapsed=true</code>

Parameter	Description
regionId	Current region, which can be obtained from the address box of the browser after you log in to a Huawei Cloud service. For example, <b>cn-north-4</b> .
cfModuleHide	<b>header_sidebar_floatlayer</b> : Hide the header, footer, and menu bar of the Huawei Cloud console.
leftMenuCollapsed	<b>true</b> : Hide the navigation pane on the left. <b>false</b> : Do not hide the navigation pane on the left.

**Step 2** Use inline frames (iframes) to embed an APM page into the self-built system.  
Example:

```
<iframe src="{FederationProxyUrl}" ref="Frame" scrolling="auto" width="100%" height="100%"></iframe>
```

----End



# 7 Locating Performance Problems Using APM Profiler

Use APM Profiler (a performance profiling tool) to locate the code that consumes excessive resources.

## Prerequisites

1. An APM Agent has been connected.
2. The Profiler function has been enabled.
3. You have logged in to the APM console.

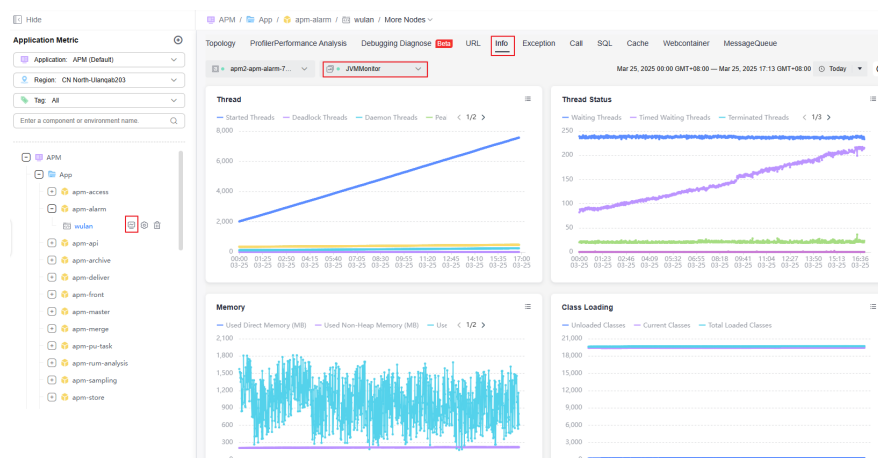
## Handling High CPU Usage

**Step 1** In the navigation pane, choose **Application Monitoring > Metrics**.

**Step 2** In the tree on the left, click  next to the target environment.

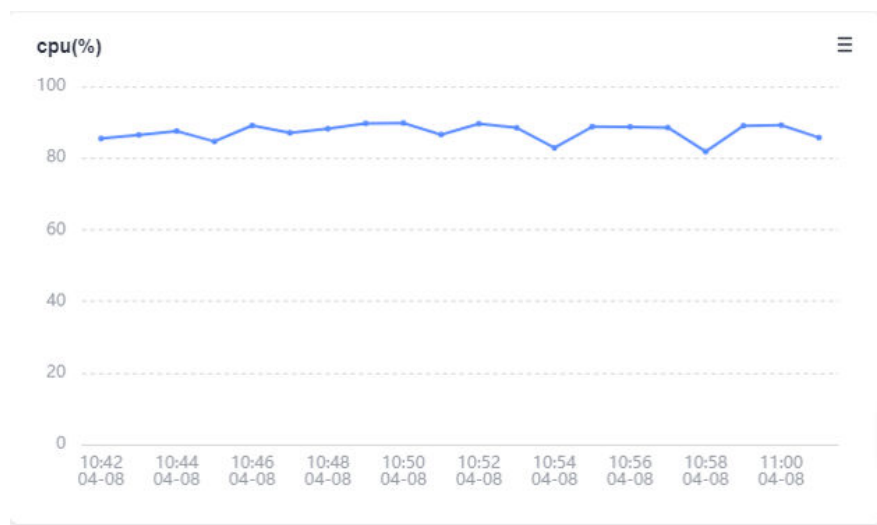
**Step 3** Click the **JVM** tab. On the displayed page, select **JVMMonitor** from the monitoring item drop-down list.

**Figure 7-1** Viewing JVM monitoring data



**Step 4** Check the **cpu(%)** graph. The CPU usage remains higher than 80%.

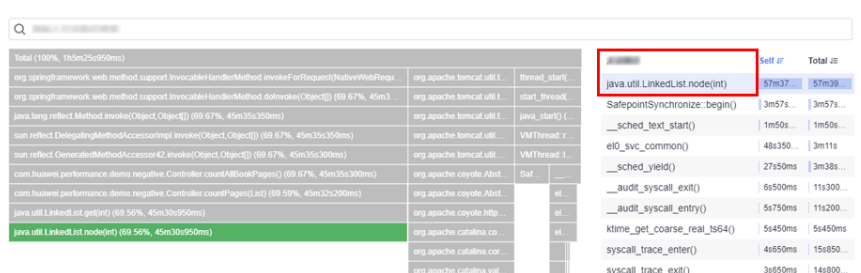
Figure 7-2 CPU (%)



**Step 5** Click the **Profiler Performance Analysis** tab.

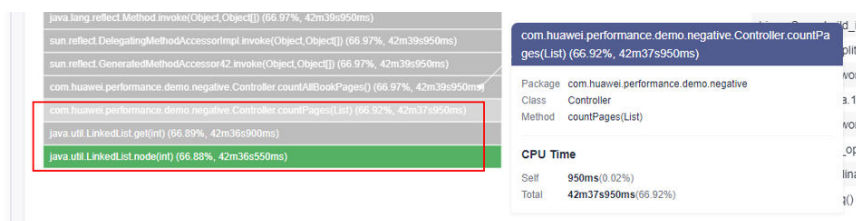
**Step 6** Click **Performance Analysis**. On the displayed page, select **CPU Time** for **Type**.

Figure 7-3 Profiler flame graph



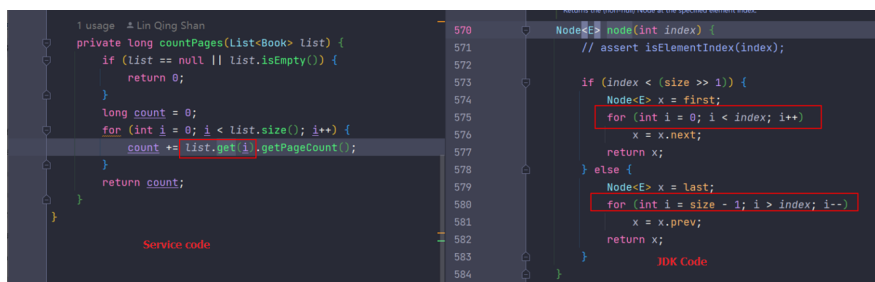
**Step 7** Analyze the flame graph data. The **java.util.LinkedList.node(int)** method occupies 66% of the CPU, and the corresponding service code method is **countPages(List)**.

Figure 7-4 Profiler flame graph analysis



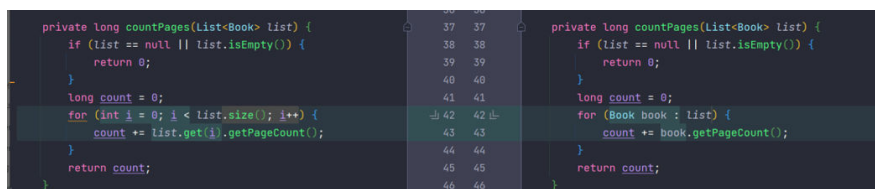
**Step 8** Analyze the service code. **countPages(List)** traverses the position indexes of the input parameter set list. However, when the input data is linked lists, position index-based traversal will be inefficient.

Figure 7-5 Code analysis



**Step 9** Fix the code. Specifically, change the list traversal algorithm to "enhanced for loop".

Figure 7-6 Fixing code



**Step 10** After the optimization, repeat steps 4 and 5. The CPU usage is lower than 1%.

Figure 7-7 CPU usage after optimization



----End

## Handling High Memory Usage

Prerequisite: The test program is started, and the heap size is set to **2g(-Xms2g -Xmx2g)**.

**Step 1** In the navigation pane, choose **Application Monitoring > Metrics**.


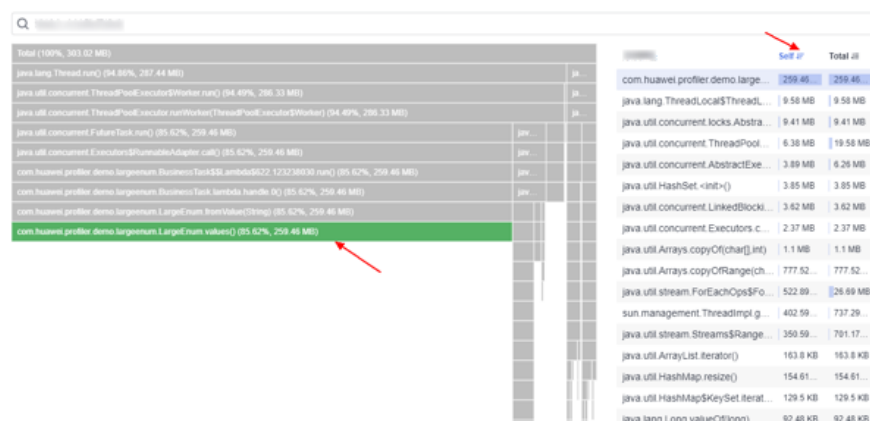
- Step 2** In the tree on the left, click  next to the target environment.
- Step 3** Click the **JVM** tab, select the **GC** monitoring item. GC events occur frequently.
- Step 4** Select the **JVMMonitor** monitoring item to check JVM monitoring data.
- Step 5** Click the **Profiler Performance Analysis** tab.
- Step 6** Click **Performance Analysis**. On the displayed page, select the **Allocated Memory** instance. Locate the method with the most allocated memory based on the **Self** column on the right.

Figure 7-8 Memory flame graph



- Step 7** Check the code. **LargeEnum** is an enumeration class and has defined a large number of constants. The enumeration class method **values()** implements functions through array clone. That is, each time the **values()** method is called, an enumeration array will be copied at the bottom layer. As a result, heap memory is frequently allocated and GC often occurs.

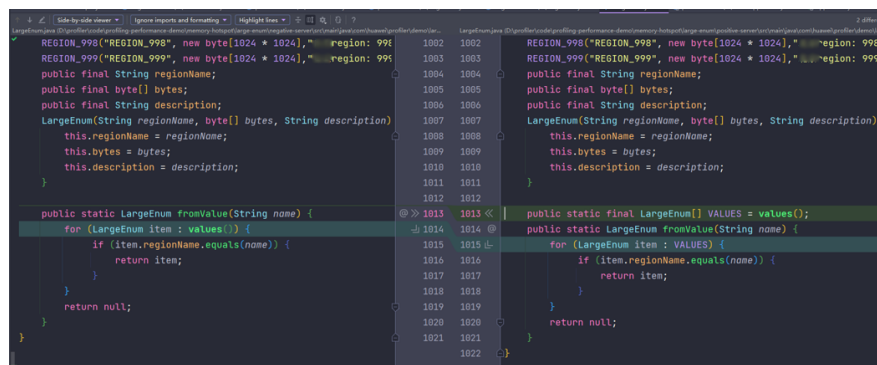
Figure 7-9 Checking the code

```
REGION_997("REGION_997", new byte[1024 * 1024], "LargeEnum: 997(1024 * 1024)");
REGION_998("REGION_998", new byte[1024 * 1024], "LargeEnum: 998(1024 * 1024)");
REGION_999("REGION_999", new byte[1024 * 1024], "LargeEnum: 999(1024 * 1024)");
public final String regionName;
public final byte[] bytes;
public final String description;
LargeEnum(String regionName, byte[] bytes, String description) {
    this.regionName = regionName;
    this.bytes = bytes;
    this.description = description;
}

public static LargeEnum fromValue(String name) {
    for (LargeEnum item : values()) {
        if (item.regionName.equals(name)) {
            return item;
        }
    }
    return null;
}
```

**Step 8** Define **values** as a constant to avoid frequent calling of **enum.values()**.

**Figure 7-10** Resolving the problem



**Step 9** Repeat steps 3 to 6. The number of GC times decreases greatly and there is no memory allocated to **enum.values()** in the flame graph.

**Figure 7-11** Flame graph after optimization



----End

## Handling Slow API Response

**Step 1** In the navigation pane, choose **Application Monitoring > Metrics**.

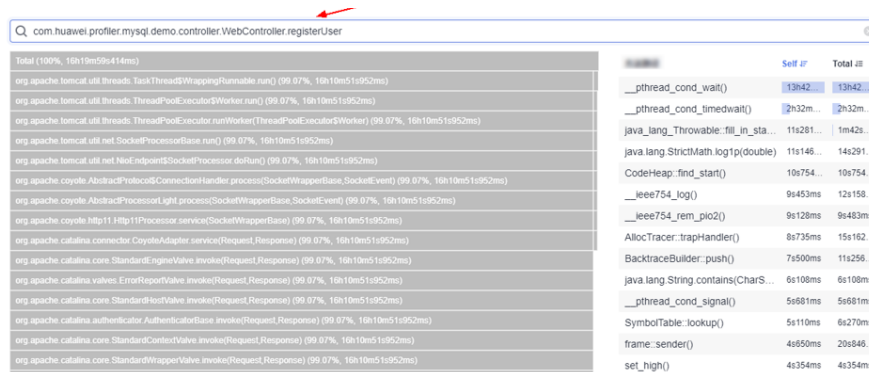
**Step 2** In the tree on the left, click  next to the target environment.

**Step 3** Click the **URL** tab. The API responses are slow. The average response time is about 80s.

**Step 4** Click the **Profiler Performance Analysis** tab.

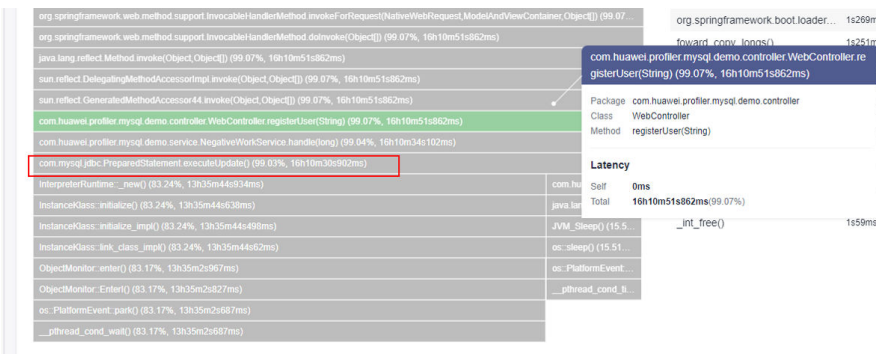
**Step 5** Click **Performance Analysis**. On the displayed page, select the **Latency** instance and enter the method of the API.

Figure 7-12 Performance analysis



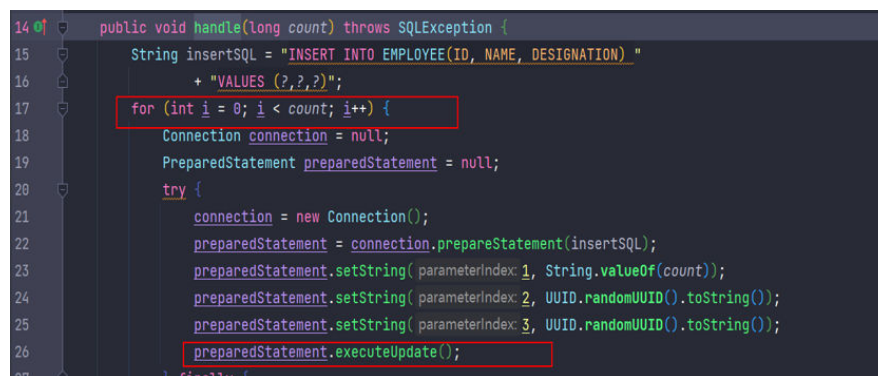
**Step 6** Check the call stack and find the time-consuming method. As shown in the following figure, the **executeUpdate()** method in **NegativeWorkService#handle** consumes the most time.

Figure 7-13 Checking the call stack

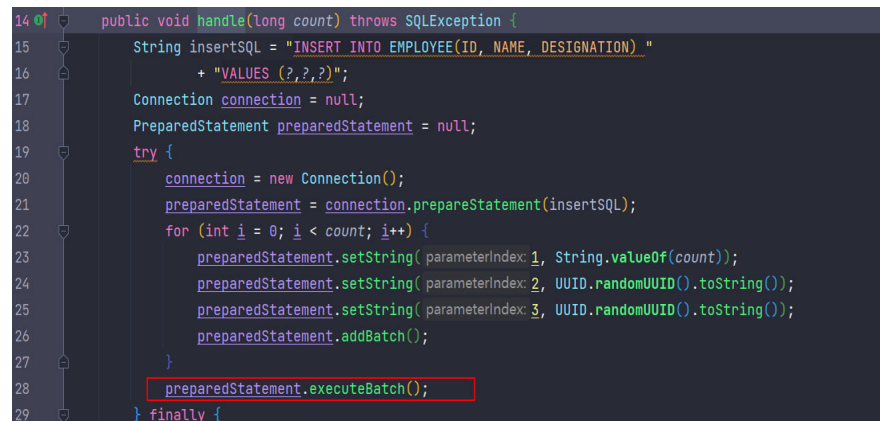


**Step 7** Check the **NegativeWorkService#handle** method. The cause is that database insertion is performed in the loop.

Figure 7-14 Checking NegativeWorkService#handle



**Step 8** Change the configuration to batch data insertion to resolve the problem.

**Figure 7-15** Resolving the problem

```
14 public void handle(long count) throws SQLException {
15     String insertSQL = "INSERT INTO EMPLOYEE(ID, NAME, DESIGNATION) "
16         + "VALUES (?, ?, ?)";
17     Connection connection = null;
18     PreparedStatement preparedStatement = null;
19     try {
20         connection = new Connection();
21         preparedStatement = connection.prepareStatement(insertSQL);
22         for (int i = 0; i < count; i++) {
23             preparedStatement.setString(parameterIndex: 1, String.valueOf(count));
24             preparedStatement.setString(parameterIndex: 2, UUID.randomUUID().toString());
25             preparedStatement.setString(parameterIndex: 3, UUID.randomUUID().toString());
26             preparedStatement.addBatch();
27         }
28         preparedStatement.executeBatch();
29     } finally {
```

**Step 9** Check the average response time. It is reduced from 80s to 0.2s.

----End



# 8

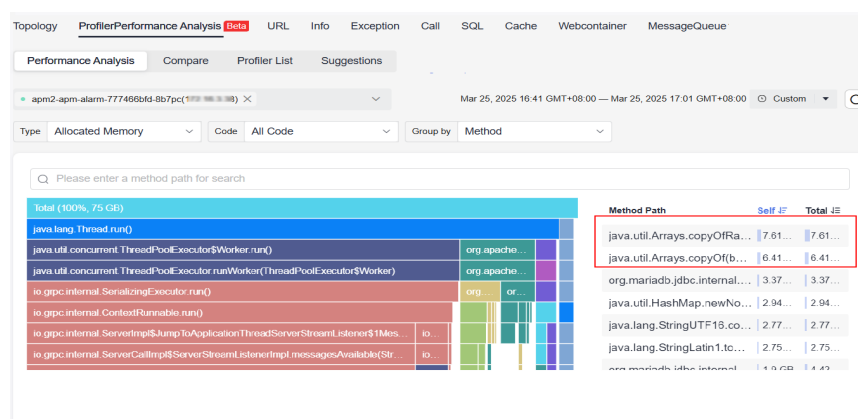
## Locating Out of Memory Problems Using Profiler

### Background

The container where a service is located restarts frequently. Based on self-monitoring, full GC occurs frequently (about 20 times per minute) before each restart.

### Procedure

- Step 1** Log in to the management 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 **Profiler Performance Analysis** tab.
- Step 6** Click **Performance Analysis**.
- Step 7** Set **Type** to **Memory**, **Code** to **All Code**, and **Group by** to **Method**. It is found that two methods occupy a large amount of memory.





**Step 8** In the **Method Path** column, find the call stack of the method based on the method name, and find the service code that calls the method.

```
com.google.common.cache.LocalCache$Segment.get(Object,int,CacheLoader)
com.google.common.cache.LocalCache$Segment.lockedGetOrLoad(Object,int,CacheLoader)
com.google.common.cache.LocalCache$Segment.loadSync(Object,int,LocalCache$LoadingValueReference,CacheLoa...
com.google.common.cache.LocalCache$LoadingValueReference.loadFuture(Object,CacheLoader)
com.huawei.hwclouds.lubanops.apm.access.cmdb.IdentityService$2.load(Object)
com.huawei.hwclouds.lubanops.apm.access.cmdb.IdentityService$2.load(InstanceIdentity)
com.huawei.hwclouds.lubanops.apmregion.biz.service.SwInstanceService$$SpringCGLIB$$0.retrieveByUuid(String)
org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(Object,Method,Object[],Meth...
```

**Step 9** Locate the target service code block. A cache is used to store information about each instance. Check the calling of the SQL statement through self-monitoring. It is called 100,000 times per minute, which further proves that the cache is ineffective.

During instance information querying, the system first queries the cache. If the information cannot be found in the cache, the system queries the MySQL database. After the instance information is queried, it is stored in the cache to prevent frequent access to the database.

**Step 10** Check the code. The cached key is a class that does not override the **equals** and **hashCode** methods. Therefore, when the cache obtains the value based on the key, whether the key exists in the cache is determined based on the key address. The key address transferred each time is different. Therefore, the system cannot find related information in the cache and needs to query the MySQL database. It frequently stores the queried information to the cache, causing out of memory.

```
56
57     private LoadingCache<InstanceIdentity, Optional<SwInstanceModel>> cache = CacheBuilder.newBuilder()
58         .expireAfterWrite( duration: 1, TimeUnit.DAYS)
59         .build((CacheLoader) (instanceIdentity) -> {
60             SwInstanceModel swInstanceModel = swInstanceService.retrieveByUuid(instanceIdentity.getUuid());
61             logger.info("swInstanceModel uuid: " + instanceIdentity.getUuid());
62             return Optional.ofNullable(swInstanceModel);
63         });
64
65
```

----End

## Solution

Override the **equals** and **hashCode** methods for the class that is used as the key. This class has the UUID attribute. Different instances have different UUIDs. Therefore, whether two instances are the same can be determined based on UUID.

# 9 Encrypting AK/SK for Deploying an APM Agent in a CCE Container

## Background

When an APM Agent is deployed in a CCE container, the AK/SK must be encrypted for security purposes.

## Procedure

**Step 1** Generate a JAR package that contains the decryption method. Assume that the JAR package name is **demo.jar**, the built-in decryption class is **com.demo.DecryptDemo**, and the decryption method is **decrypt** (which is a static method). Then pack the JAR package into an image and upload it to the image repository. To obtain an access key, see [Access Keys](#).

**Step 2** Add the **initContainer** attribute to the CCE deployment YAML file.

Example:

1. The address of the image uploaded in step 1 is **swr.cn-north-5.myhuaweicloud.com/hwstaff\_pub\_apmpaasw3/decrypt:v2**.
2. The decryption class name is **com.demo.DecryptDemo**. The decryption method is **decrypt**.

The following shows **initContainer**. Replace the information in bold.

```
initContainers:
- name: init-secret
  image: swr.cn-north-5.myhuaweicloud.com/hwstaff_pub_apmpaasw3/decrypt:v2
  command:
  - /bin/sh
  - '-c'
  - cp /root/com.demo.DecryptDemo.jar /var/init/secret/apm-javaagent/ext; sed -i 's
##decrypt.className=.*%decrypt.className=com.demo.DecryptDemo%g' /var/init/secret/apm-
javaagent/apm.config; sed -i 's%#decrypt.methodName=.*%decrypt.methodName=decrypt%g' /var/
init/secret/apm-javaagent/apm.config;
  resources:
    limits:
      cpu: 100m
      memory: 100Mi
    requests:
      cpu: 100m
      memory: 100Mi
    volumeMounts:
```

```
- name: paas-apm2
  mountPath: /var/init/secret
```

By adding **initContainer**, you can copy the JAR package to the **apm-javaagent/ext** directory and modify the configuration file.

**Step 3** Obtain an AK/SK from the APM console, encrypt the SK, and replace the AK/SK in the YAML file.

**Step 4** Save the configuration and upgrade the CCE instance.

----End

# 10 Suggestions on APM Security Configuration

---

This section provides guidance for enhancing the overall security of APM. You can continuously evaluate the security of APM and combine different security capabilities to enhance overall defense. By doing this, stored data can be protected from leakage and tampering both at rest and in transit.

Consider the following aspects for your security configurations:

- [Properly Using APM Access Keys and Encrypting Them](#)
- [Granting User Permissions Using Access Control Capabilities](#)
- [Protecting Privacy and Sensitive Information Through Data Masking](#)
- [Using the Latest Agent for Better Monitoring Experience and Security Capabilities](#)

## Properly Using APM Access Keys and Encrypting Them

1. **Keeping APM access keys secure and changing them regularly**  
Access Key ID (AK) and Secret Access Key (SK) are your long-term identity credentials. Agents report data with an AK. An AK is used together with an SK to sign requests cryptographically, ensuring that the requests are secret, complete, and correct. Keep your access keys secure. You can also create, delete, and disable access keys on the **Access Keys** page.
2. **Using custom functions to encrypt authentication information**  
To better protect your identity authentication information, APM supports custom encryption for keys. You can customize a function to encrypt an AK/SK and place the decryption function in the specified directory of an Agent. When a service is started, the Agent uses the custom decryption function to parse the key, protecting privacy and preventing identity authentication information leakage. For details, see [Access Keys](#).

## Granting User Permissions Using Access Control Capabilities

1. **Granting IAM users with different roles to prevent data leakage or misoperations caused by excessive permissions**  
To better isolate and manage permissions, you are advised to configure independent IAM administrators and grant them permissions to manage IAM

policies. An IAM administrator can create different user groups based on your service requirements. User groups correspond to different data access scenarios. By adding users to user groups and binding IAM policies to user groups, the IAM administrator can grant different data access permissions to employees in different departments based on the principle of least privilege. For details, see [Login Protection](#) and [Login Authentication Policy](#).

2. **Using enterprise projects to isolate services logically**

After creating IAM user groups for employees, you can create enterprise projects on the Enterprise Management console and grant permissions to the user groups in the enterprise projects to implement personnel authorization and permission control. You can create enterprise projects. Then you can manage resources across different regions by enterprise project, grant different permissions to user groups, and add them to enterprise projects. For details, see [Creating a User and Granting Permissions](#).

## **Protecting Privacy and Sensitive Information Through Data Masking**

When a service request includes sensitive information, you are advised to use the data masking function. On the data masking page, create masking configurations for your components. The platform will then replace sensitive information in traces with a globally unique random character string (**Hash code** mode) or a fixed number of asterisks (\*) (**Mask** mode). After the configuration takes effect, you can go to the tracing page to view the trace details.

## **Using the Latest Agent for Better Monitoring Experience and Security Capabilities**

Regularly update your Agent versions for better monitoring experience and security capabilities. To download the latest Agent, see [JavaAgent Updates](#).