Application Performance Management (APM)

Best Practicesy

Issue	01
Date	2026-01-30

HUAWEI

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 Best Practices

This document summarizes Application Performance Management (APM) best practices and provides guidance for you to use APM.

Table 1-1 APM best practices

Best Practice	Description
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 provided by APM.
Associating Traces with Logs	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.
Locating Performance Problems Using APM Profiler	Use APM Profiler (a performance profiling tool) to locate the code that consumes excessive resources.
Locating Out of Memory Problems Using Profiler	Use APM Profiler to locate memory overflows.
Locating the Causes of Request Errors	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.

Best Practice	Description
Embedding APM Pages into a Self-built System	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.
Encrypting AK/SK for Deploying an APM Agent in a CCE Container	Encrypt the AK/SK when deploying an APM Agent on CCE.
Searching for Span Information	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.
Connecting to APM Using OpenTelemetry	Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent.

2 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**.

3 Associating Traces with Logs

Application Scope

Common log frameworks include Logback and Log4j.

Example

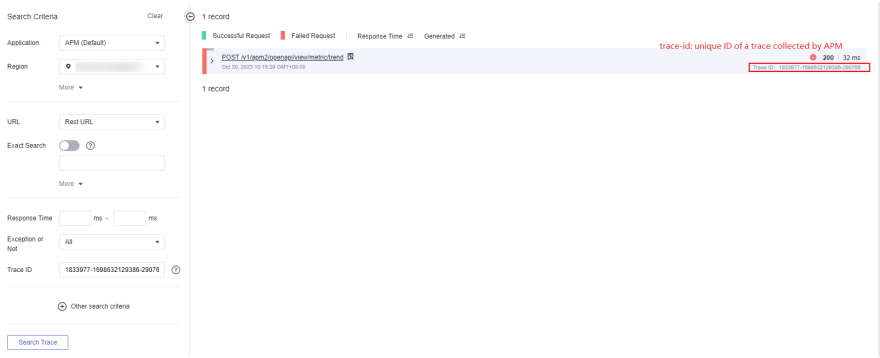
<property name="LOG_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS}} | gtraceid: **%X{apm-gtraceid}** | traceid: **%X{apm-traceid}** | spanId: **%X{apm-spanid}**">

</property>

Trace Parameters

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

   Figure 3-1 Unique ID of a trace

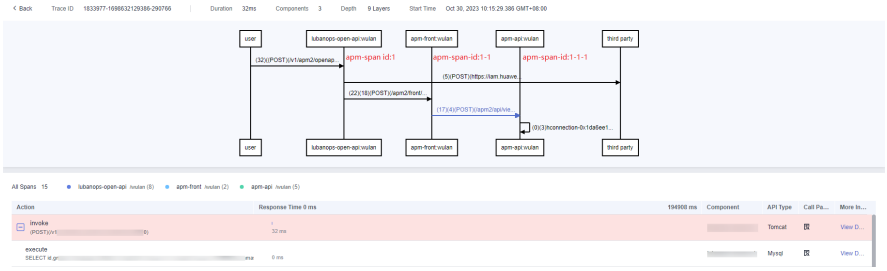
   

2. **apm-gtraceid**: unique ID of a trace which is not sampled.

   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 3-2 Calls between microservices



4 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.

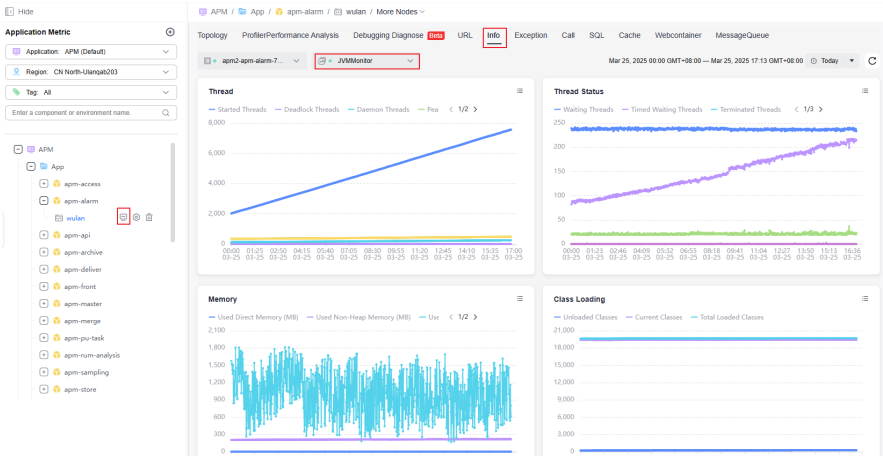
Handling High CPU Usage

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

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

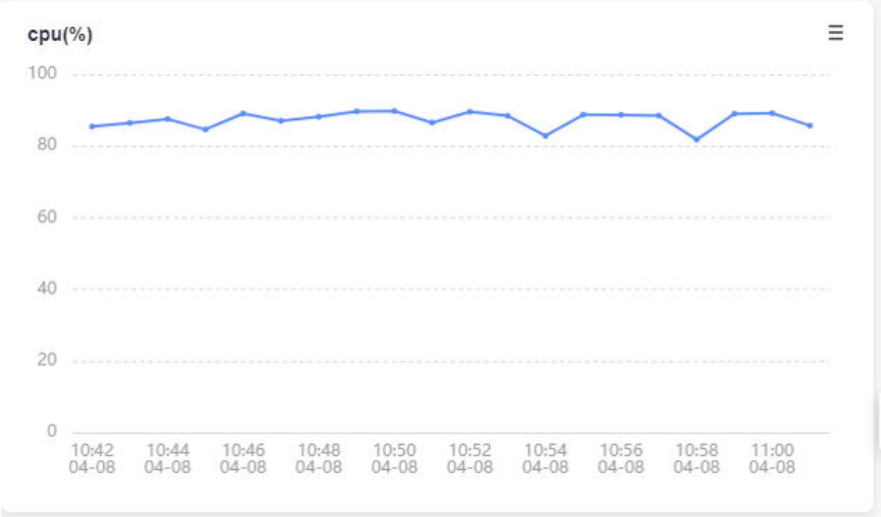
Step 3 Click the **JVM** tab. On the displayed page, select **JVMMonitor** from the monitoring item drop-down list.

Figure 4-1 Viewing JVM monitoring data



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

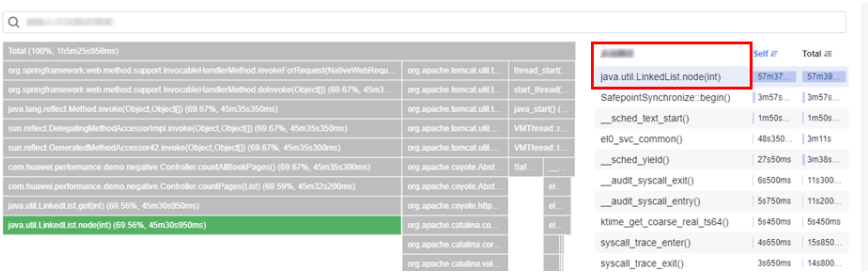
Figure 4-2 CPU (%)



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

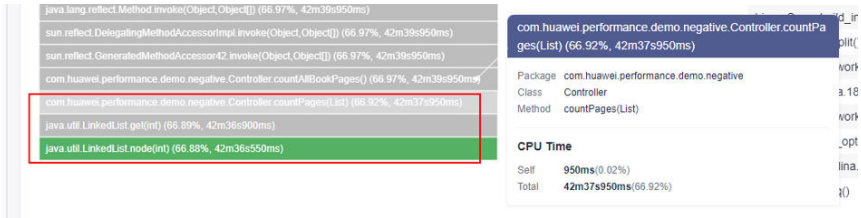
Step 6 Click **Performance Analysis**. On the displayed page, select **CPU Time** for **Type**.

Figure 4-3 Profiler flame graph



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

Figure 4-4 Profiler flame graph analysis

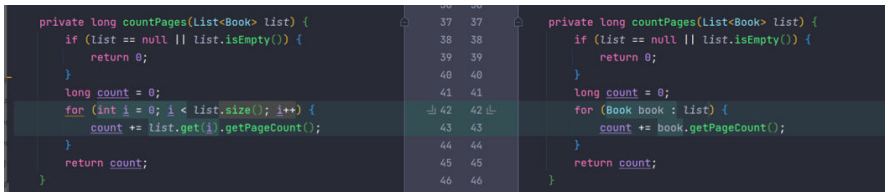


Step 8 Analyze the service code. **countPages(List)** traverses the position indexes of the input parameter set list. However, when the input data is linked lists, position index-based traversal will be inefficient.

Figure 4-5 Code analysis

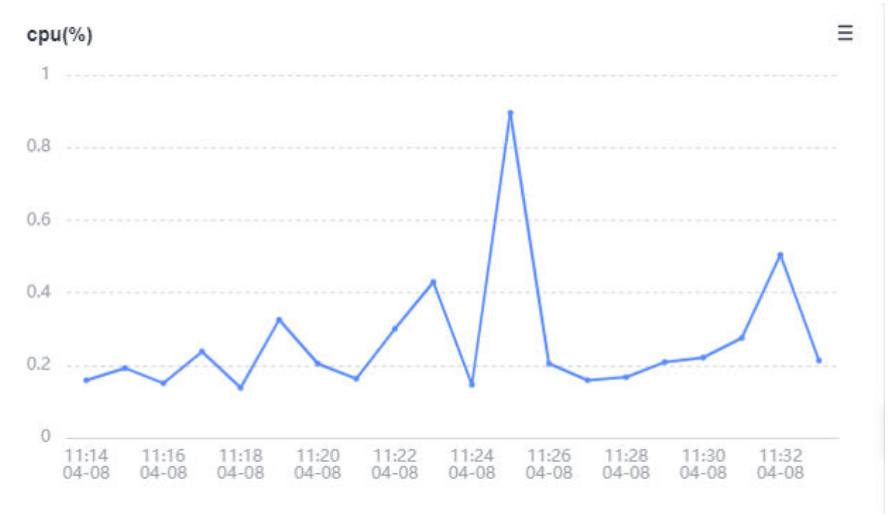
Step 9 Fix the code. Specifically, change the list traversal algorithm to "enhanced for loop".

Figure 4-6 Fixing code



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

Figure 4-7 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.

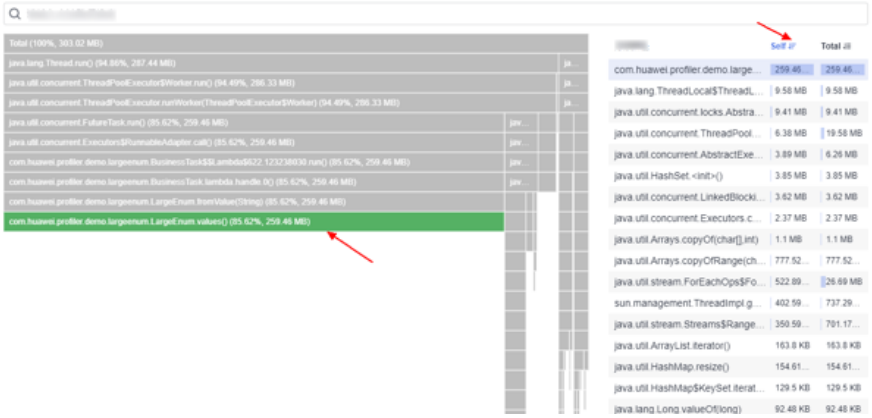
Step 3 Click the **JVM** tab, select the **GC** monitoring item. GC events occur frequently.

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 4-8 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.

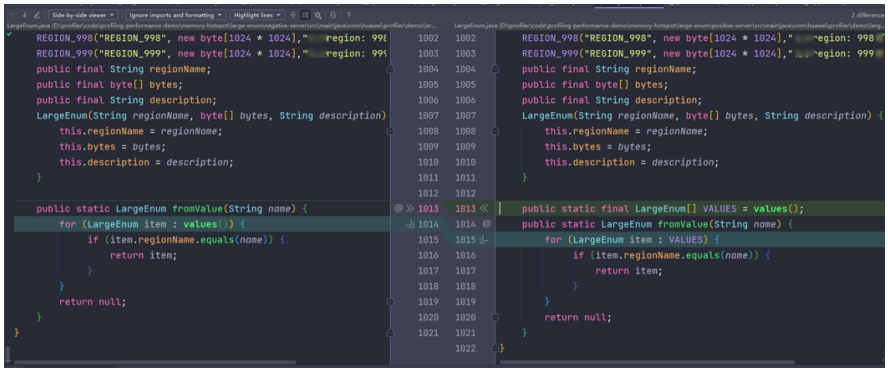
Figure 4-9 Checking the code

```
    REGION_997("REGION_997", new byte[1024 * 1024],"                    "),
    REGION_998("REGION_998", new byte[1024 * 1024],"                    "),
    REGION_999("REGION_999", new byte[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 4-10 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 4-11 Flame graph after optimization

----End

Handling Slow API Response

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

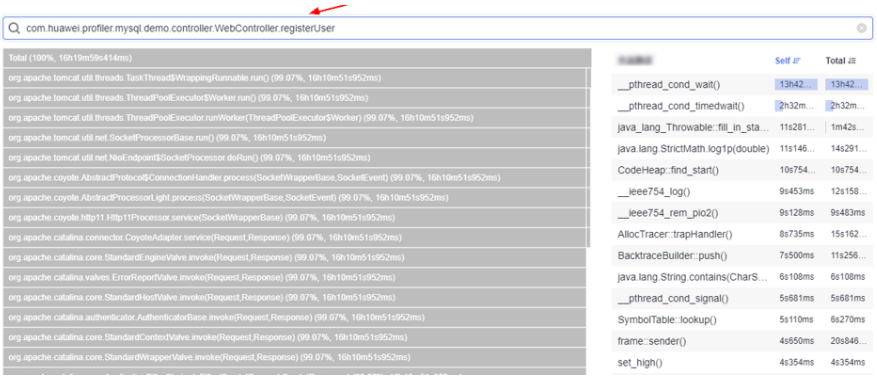
Step 2 In the tree on the left, click  next to the target environment.

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.

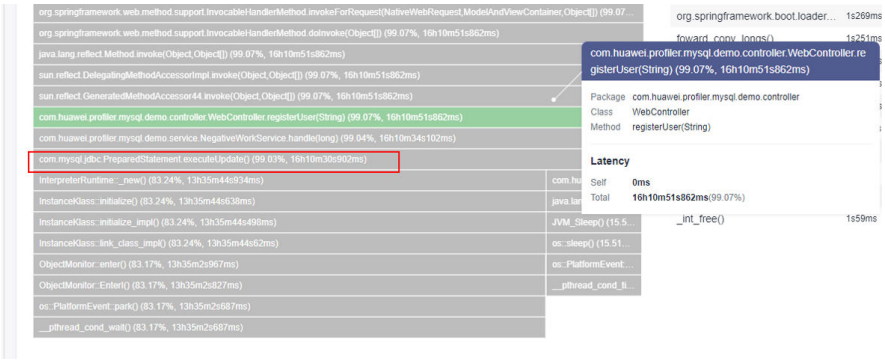
Step 5 Click **Performance Analysis**. On the displayed page, select the **Latency** instance and enter the method of the API.

Figure 4-12 Performance analysis



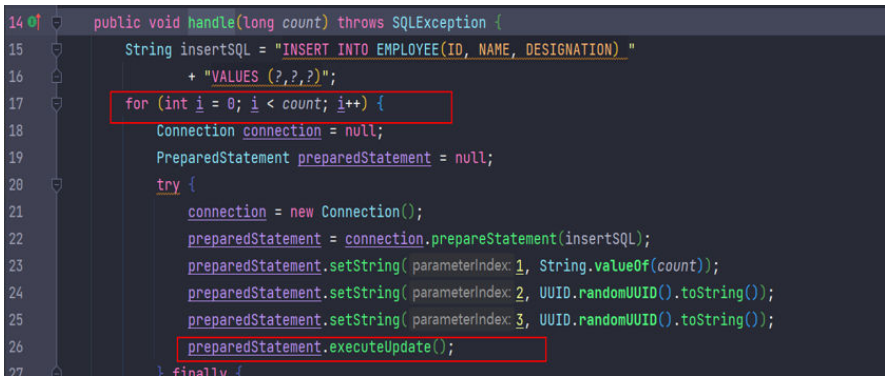
Step 6 Check the call stack and find the time-consuming method. As shown in the following figure, the **executeUpdate()** method in **NegativeWorkService#handle** consumes the most time.

Figure 4-13 Checking the call stack



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

Figure 4-14 Checking NegativeWorkService#handle



```
14  public void handle(long count) throws SQLException {
15      String insertSQL = "INSERT INTO EMPLOYEE(ID, NAME, DESIGNATION) "
16              + "VALUES (?,?,?)";
17      for (int i = 0; i < count; i++) {
18          Connection connection = null;
19          PreparedStatement preparedStatement = null;
20          try {
21              connection = new Connection();
22              preparedStatement = connection.prepareStatement(insertSQL);
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.executeUpdate();
27          } finally {
```

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

Figure 4-15 Resolving the problem

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

----End

5 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**.

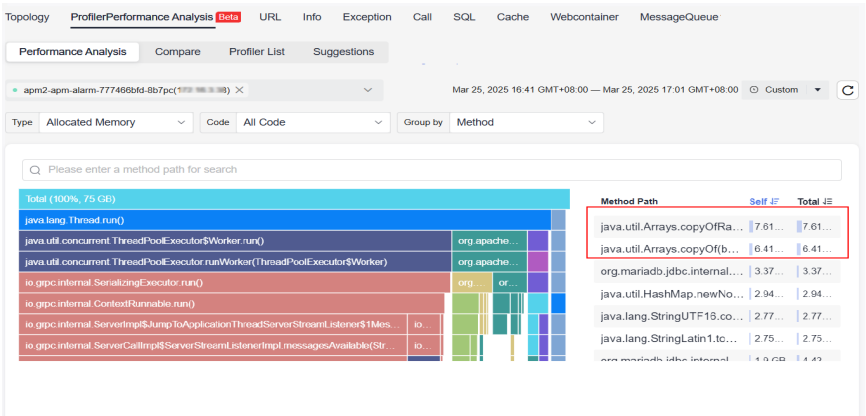
Step 3 In the navigation pane, choose **Application Monitoring** > **Metrics**.

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

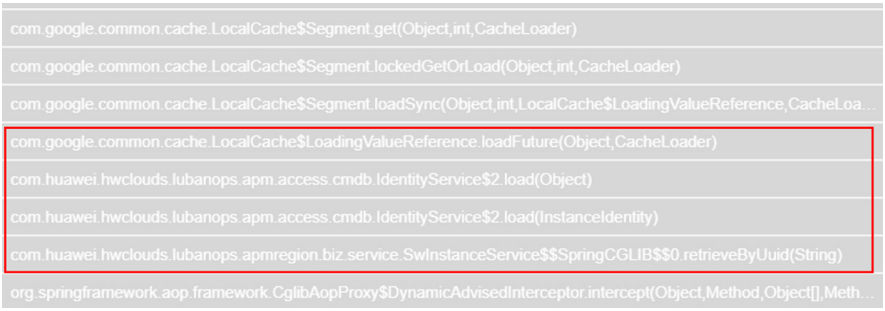
Step 5 Click the **Profiler Performance Analysis** tab.

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.



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.

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

6 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 and locating application performance problems are common in routine O&M.

APM, as a cloud application performance diagnosis service, provides powerful analysis tools. Topologies and traces show the application status and call process, helping you quickly locate 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

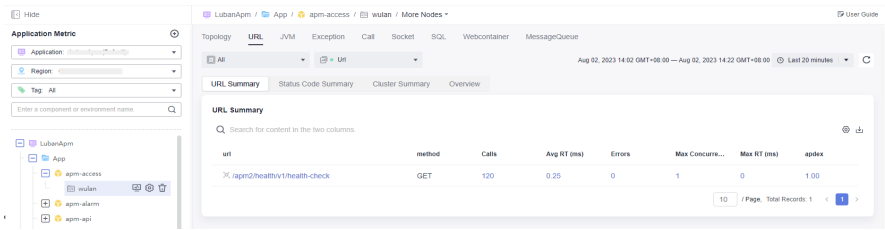
Locating the Causes of Request Errors Based on Traces

Step 1 Log in to the APM console.

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

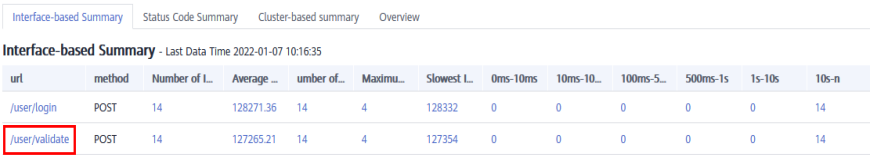
Step 3 Click the **URL** tab. On the page that is displayed, check metrics such as the number of calls, number of errors, and latency.

Figure 6-1 Going to the URL page



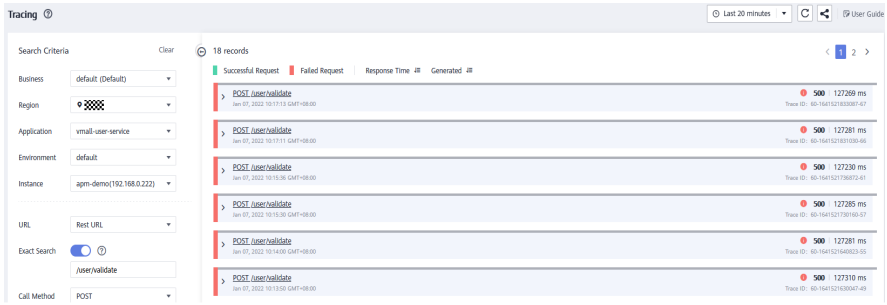
Step 4 Click the abnormal URL to go to the tracing page.

Figure 6-2 URL details



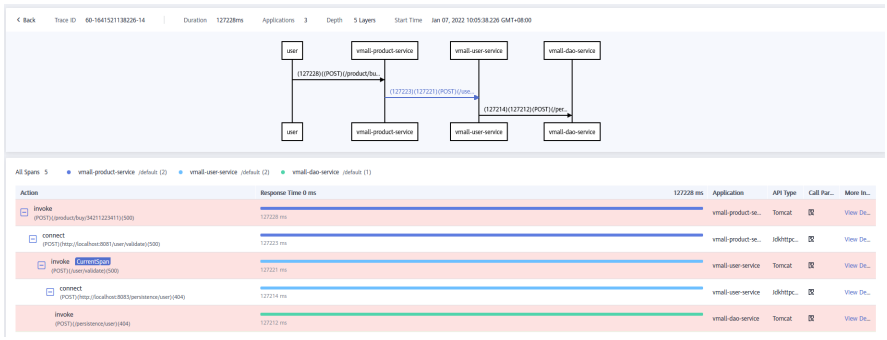
Step 5 Locate error or slow traces.

Figure 6-3 Viewing traces



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

Figure 6-4 Trace details



----End

7 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	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/appindex/business/detail?leftMenuCollapsed=true
Application Monitoring - Tracing	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/appchain?leftMenuCollapsed=true

Service Module	Example URL
Link Trace - Metrics	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/trace/metric/environment/view?leftMenuCollapsed=true
Link Trace - Tracing	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/trace/chain?leftMenuCollapsed=true
Web Monitoring - Websites	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/rum/app?leftMenuCollapsed=true
App Monitoring - Apps	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/mobile/list?leftMenuCollapsed=true

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, **cn-north-4**.
cfModuleHide	**header_sidebar_floatlayer**: Hide the header, footer, and menu bar of the Huawei Cloud console.
leftMenuCollapsed	**true**: Hide the navigation pane on the left. **false**: 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

8 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.

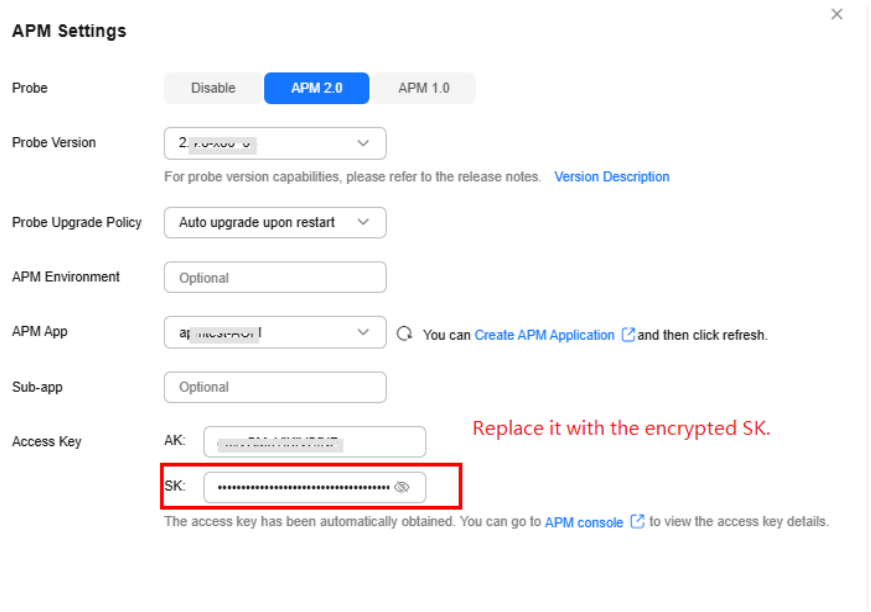
```
initContainers:
  - name: init-secret
    image: swr.cn-north-5.myhuaweicloud.com/hwstaff_pub_apmpaasw3/decrypt:v2
    command:
      - /bin/sh
      - '-c'
      - cp /root/com.demo.DecryptDemo.jar /var/init/secret/apm-javaagent/ext; sed -i 's%#decrypt.className=.*%decrypt.className=com.demo.DecryptDemo%g' /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

9 Searching for Span Information

Background

In a distributed architecture, the calls between microservices are increasingly complex. When the response to an external request is slow or some requests are abnormal, you need to quickly locate the exception. On the trace page, you can filter traces by trace ID or based on multiple criteria.

Querying Span Information Based on Traces

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 9-1 Tracing search result

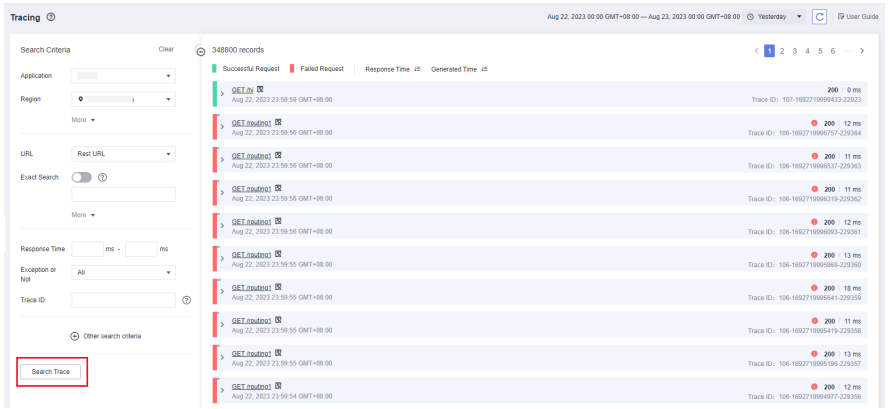


Table 9-1 Search criteria of traces

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	There are REST URLs and real URLs. A REST URL contains a variable name (Example: **/apm/get/***{id}*). A real URL is 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 9-2 Other search criteria

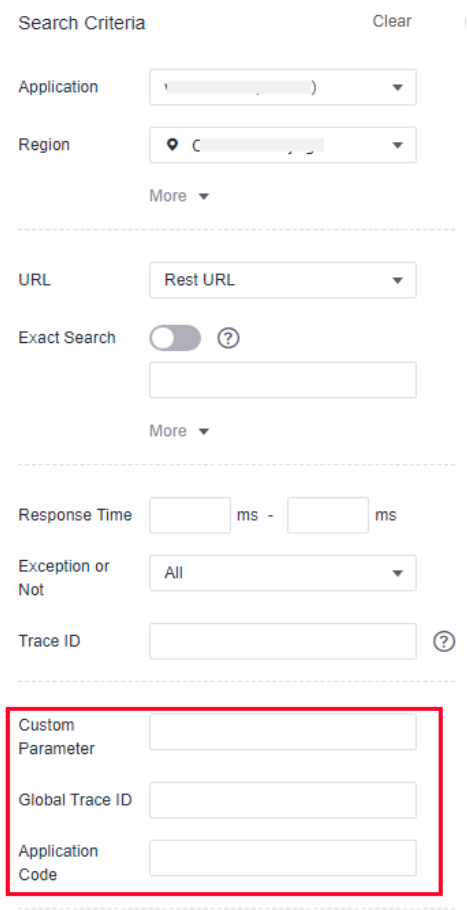


Table 9-2 Search criteria of traces

Search Criterion	Description	Mandatory
Custom Parameter	If you have configured **Key for Header Value Interception**, **Key for Parameter Value Interception**, and **Key for Cookie Value Interception** for URL monitoring, you can set **key=value** 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 **Service Code Length**, **Key for Service Code Interception**, and **Normal Service Code**, corresponding application codes will be collected. You can search information based on the application codes.	No

- **Custom Parameter**

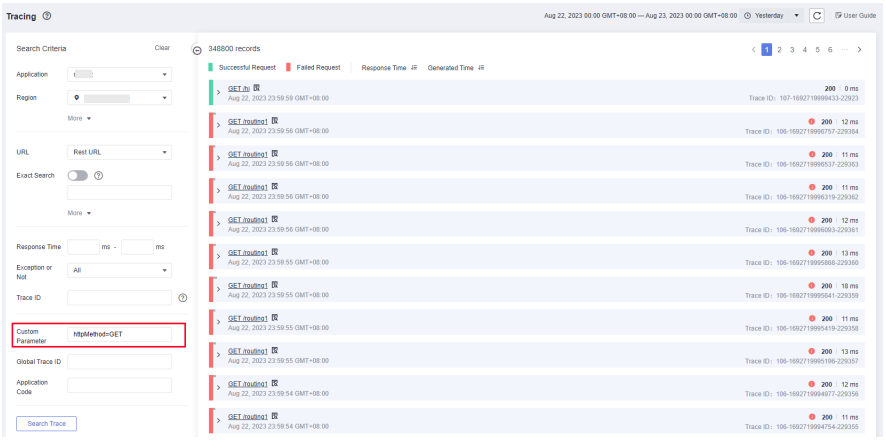
  Usage Instructions

  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 9-3 Results of querying traces based on the custom parameters

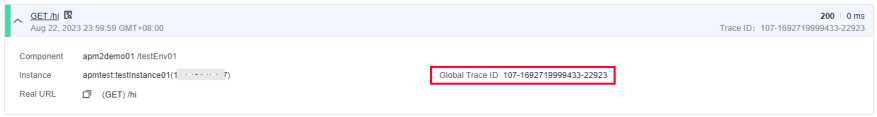
  

- **Global Trace ID**

  Usage Instructions

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

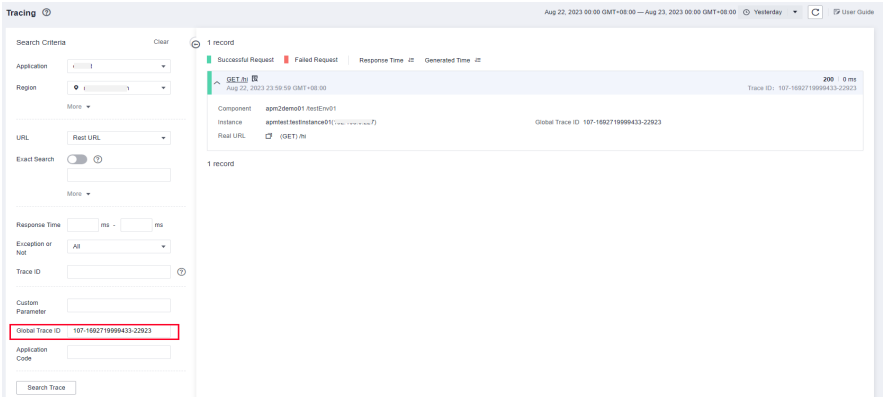
  Figure 9-4 Obtaining the global trace ID

  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 9-5 Results of querying traces based on the global trace ID

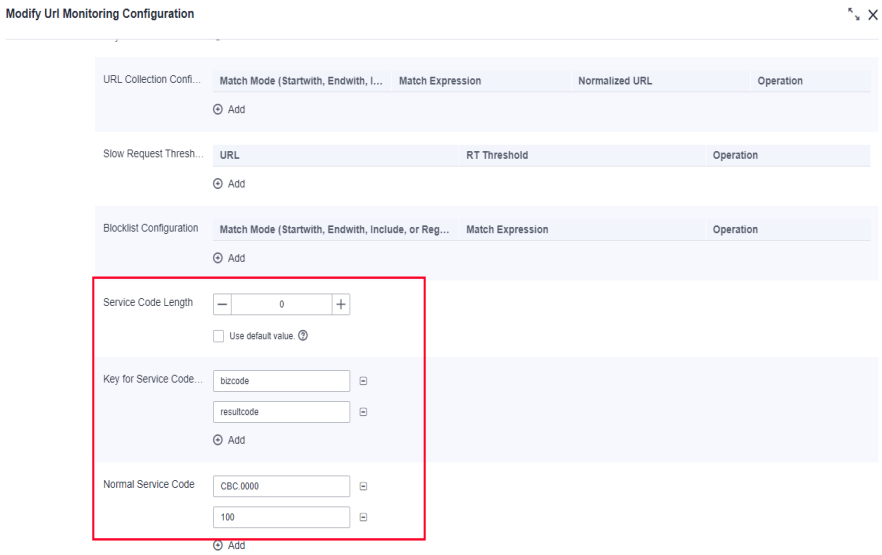
  

- **Application Code**

  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**.

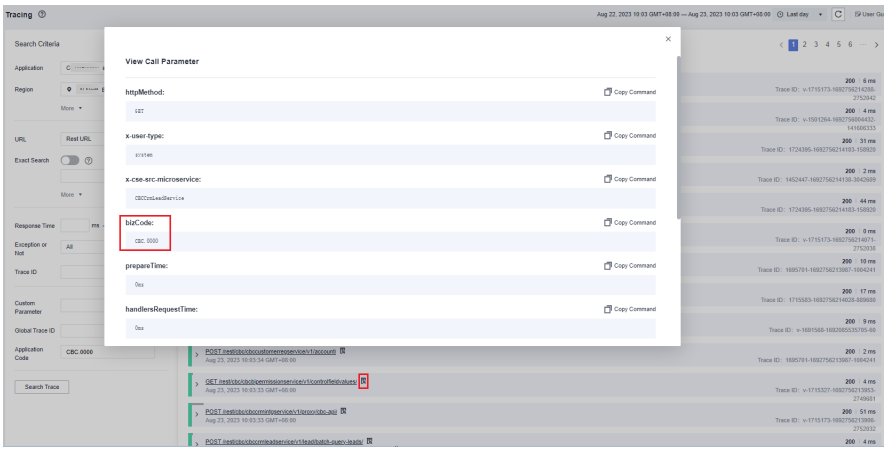
  Figure 9-6 URL monitoring

  b. In the navigation pane, choose **Application Monitoring** > **Tracing**.

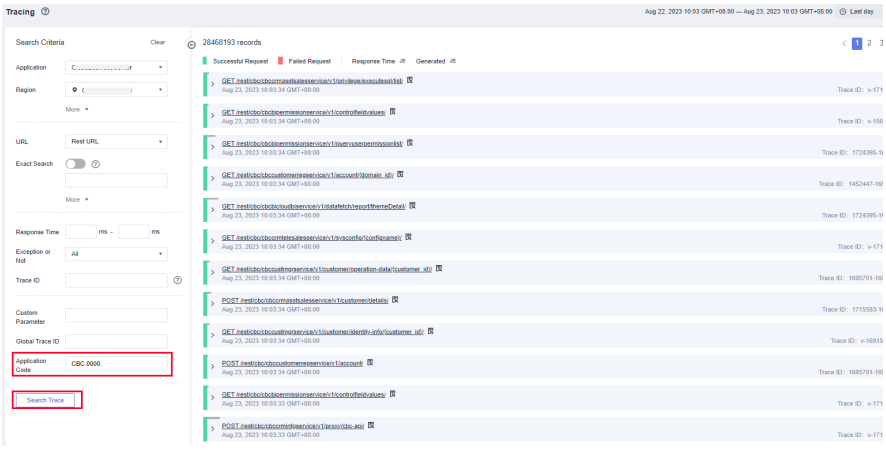
  c. Click  to view the value of the service code, which corresponds to the application code.

  Figure 9-7 Viewing the service code

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

Figure 9-8 Searching for the traces corresponding to the code



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

----End

10 Connecting to APM Using OpenTelemetry

10.1 Reporting Java Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a Java application to APM and report trace data.

Demo

Use Spring Boot to implement a simple dice roller application.

Step 1 Compile service code and initialize a Spring Boot project.

Create a **RollDice.java** file:

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.util.Optional;
import java.util.concurrent.ThreadLocalRandom;

@RestController
public class RollDice {

  @GetMapping("/rolldice")
  public String rollDiceAndSave(@RequestParam("player") Optional<String> player) {
     int result = 0;
     //Roll the dice.
     try {
        Thread.sleep(100);
        result = ThreadLocalRandom.current().nextInt(1, 7);
     } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
     }
     saveResult(player.orElse("Anonymous player"), result);

     return Integer.toString(result);
  }
```

```
    private void saveResult(String player, int dicePoint) {
        //Simulate the time required for performing operations on the database.
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Content of **application.properties**:

```
server.port=8080
```

Step 2 Run the startup command:

```
java -jar otel-demo-0.0.1-SNAPSHOT.jar &
curl http://localhost:8080/rolldice?player=Alex
```

----End

Using Java Agent for Auto Tracking

The OpenTelemetry Java Agent supports non-intrusive tracking for some common libraries/frameworks. For details, see the **Libraries/Frameworks supported by auto tracking**.

Step 1 **Obtain the Agent**.

Step 2 Add Agent startup parameters and restart the application. Command:

```
java -javaagent: Agent installation path  \
  -Dotel.exporter.otlp.protocol=grpc  \
  -Dotel.exporter.otlp.traces.endpoint= Reporting address \
  -Dotel.exporter.otlp.headers=Authentication= Authentication information  \
  -Dotel.service.name= Application name.Component name.Environment name  \
  -Dotel.metrics.exporter=none  \
  -Dotel.logs.exporter=none  \
  -jar <yourApp>.jar  &

curl http://localhost:8080/rolldice?player=Alex
```

Step 3 Log in to the **APM console**.

Step 4 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 5 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 6 In the navigation tree on the left, click an environment, and then click **Overview**. On the displayed page, view the monitoring data of the instance. For details, see **Metrics**.

----End

Using Java SDK for Manual Tracking

If auto tracking does not meet service requirements or you need to create custom tracking logic, use the OpenTelemetry SDK for manual tracking.

Step 1 Add Maven dependencies.

Add OpenTelemetry dependencies to the **pom.xml** file.

```
<dependencies>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-api</artifactId>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-sdk-trace</artifactId>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-exporter-otlp</artifactId>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-sdk</artifactId>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-semconv</artifactId>
      <version>1.30.0-alpha</version>
    </dependency>
  </dependencies>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.opentelemetry</groupId>
        <artifactId>opentelemetry-bom</artifactId>
        <version>1.30.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

Step 2 Initialize the OpenTelemetry SDK.

1. Write a tool class to configure and create an OpenTelemetry instance.

```
import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.api.trace.propagation.W3CTraceContextPropagator;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.BatchSpanProcessor;
import io.opentelemetry.semconv.resource.attributes.ResourceAttributes;

public class OpenTelemetrySupport {

    static {
        // Obtain the OpenTelemetry tracer.
        Resource resource = Resource.getDefault()
            .merge(Resource.create(Attributes.of(ResourceAttributes.SERVICE_NAME, "Application name.Component name.Environment name", // For Application name, use the name of the OpenTelemetry application created in APM. For Component name and Environment name, enter any value.
                ResourceAttributes.SERVICE_VERSION, "1.0.0", // Version number.
                ResourceAttributes.DEPLOYMENT_ENVIRONMENT, "test", // Deployment environment. Enter any value.
                ResourceAttributes.HOST_NAME, "127.0.0.1" // Replace ${host-name} with your host name or enter any value.
            )));

        SdkTracerProvider sdkTracerProvider = SdkTracerProvider.builder()
            .addSpanProcessor(BatchSpanProcessor.builder(
```

```
            OtlpGrpcSpanExporter.builder().setEndpoint("Access address") // Access address.
                .addHeader("Authentication", "Application authentication information") // Application
authentication information.
                .build()).build())
        .setResource(resource)
        .build();

    OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
        .setTracerProvider(sdkTracerProvider)
        .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
        .buildAndRegisterGlobal();

    tracer = openTelemetry.getTracer("OpenTelemetry Tracer", "1.0.0");
  }

  private static Tracer tracer;

  public static Tracer getTracer() {
    return tracer;
  }
}
```

2. Create spans.

```
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.SpanKind;
import io.opentelemetry.api.trace.StatusCode;
import io.opentelemetry.context.Context;
import io.opentelemetry.context.Scope;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.Optional;
import java.util.concurrent.ThreadLocalRandom;

@RestController
public class RollDice {

  @GetMapping("/rolldice")
  public String rollDiceAndSave(@RequestParam("player") Optional<String> player) {
    int result = 0;
    Span span =
OpenTelemetrySupport.getTracer().spanBuilder("rolldice").setSpanKind(SpanKind.SERVER).startSpan();
    try (Scope scope = span.makeCurrent()) {
      span.setStatus(StatusCode.OK);
      span.setAttribute("dice.player", player.orElse("Anonymous player"));
      span.addEvent("ev",
          Attributes.builder().put("event1", "value1").build());  // event1 and value1 will be
displayed on the trace span details page.
      span.setAttribute("http.target", "rolldice"); // This field is mandatory when the service serves
as the server.
      span.setAttribute("http.method", "GET");  // This field is mandatory when the service serves
as the server.
      span.setAttribute("http.route", "{path }/**"); // Optional.
      span.setAttribute("http.status_code", 200); // Optional.
      Attributes attributes = Attributes.builder()
          .put("exception.type", "NullPoint")
          .put("exception.message", "I am exception message")
          .build(); // (Optional) This field is displayed on the trace span details page.
      span.addEvent("exception", attributes);
      //Roll the dice.
      try {
        Thread.sleep(100);
        result = ThreadLocalRandom.current().nextInt(1, 7);
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
      saveResult(player.orElse("Anonymous player"), result, span);
    } catch (Throwable t) {
```

```
            span.setStatus(StatusCode.ERROR, "handle parent span error");
        } finally {
            span.end();
        }
        return Integer.toString(result);
    }

    private void saveResult(String player, int dicePoint, Span parentSpan) {
        Span span = OpenTelemetrySupport.getTracer()
            .spanBuilder("saveResult")
            .setSpanKind(SpanKind.CLIENT)    // CLIENT indicates that an external service (such as a database) is called.
            .setParent(Context.current().with(parentSpan))    // Parent span. This helps establish the parent-child relationship and form a complete distributed tracing tree.
            .startSpan();

        try (Scope scope = span.makeCurrent()) {
            span.setAttribute("db.system", "mysql"); // Database type, which is MySQL.
            span.setAttribute("db.statement",
                "INSERT INTO dice_roll (player_name, dice_point) VALUES (" + player + ", " + dicePoint
                    + ")"); // SQL statement.
            span.setAttribute("db.connection_string",
                "127.0.0.1:3306:mysql"); // Connected database.

            //Simulate the time required for performing operations on the database.
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        } catch (Throwable t) {
            span.recordException(t);
            span.setStatus(StatusCode.ERROR, "handle child span error");
        } finally {
            span.end();
        }
    }
}
```

Step 3 Log in to the **APM console**.

Step 4 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 5 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 6 In the navigation tree on the left, click an environment, and then click **Overview**. On the displayed page, view the monitoring data of the instance. For details, see **Metrics**.

----End

10.2 Reporting Python Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a Python application to APM and report trace data.

Constraint

Python 3.9 or later must be used.

Environment Preparation

1. Install Python:
```
sudo apt update
sudo apt install python3 python3-pip python3-venv
```
2. Verify the environment:
```
python3 --version
pip3 --version
```

Demo

Implement a simple dice roller application using the Flask framework.

1. Create a running environment:
```
mkdir demo
cd demo
python3 -m venv demo
source ./demo/bin/activate
```
2. Download dependencies:
```
pip install flask
```
3. Compile the service code. Create an **app.py** file:
```
from random import randint
from flask import Flask, request
import logging

app = Flask(__name__)
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@app.route("/rolldice")
def roll_dice():
    player = request.args.get('player', default=None, type=str)
    result = str(roll())
    if player:
        logger.warning("%s is rolling the dice: %s", player, result)
    else:
        logger.warning("Anonymous player is rolling the dice: %s", result)
    return result

def roll():
    return randint(1, 6)
```
4. Run the startup command:
```
flask run -p 8080 &
curl http://localhost:8080/rolldice
```

Auto Tracking

Step 1 Download dependencies:
```
pip install opentelemetry-distro
pip install opentelemetry-exporter-otlp
opentelemetry-bootstrap -a install
```

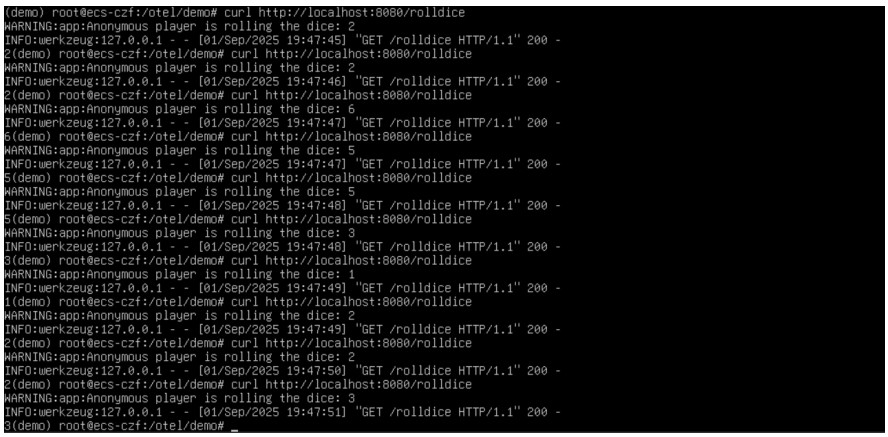
Step 2 Start the Python application:
```
opentelemetry-instrument  \
--traces_exporter otlp  \
--metrics_exporter none \
--service_name Application name.Component name.Environment name  \
```

```
--resource_attributes host.name=hostName  \
--exporter_otlp_endpoint Reporting address  \
--exporter_otlp_headers Authentication=Authentication information    \
flask run -p 8080  &

curl  http://localhost:8080/rolldice
```

Step 3 Trigger a request:

```
curl http://localhost:8080/rolldice
```



Step 4 Log in to the **APM console**.

Step 5 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 6 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 7 In the tree on the left, locate an environment and then go to the **URL** tab page to view the URL calling data.

Step 8 In the navigation pane, choose **Link Trace** > **Tracing**.

----End

10.3 Reporting Go Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a Go application to APM and report trace data.

Constraint

Go 1.23 or later must be used.

Environment Preparation

1. Install Go:

```
wget  https://mirrors.aliyun.com/golang/go1.23.1.linux-amd64.tar.gz
tar -C /usr/local -xzf go1.23.1.linux-amd64.tar.gz
```



```
echo 'export PATH=$PATH:/usr/local/go/bin' >> /etc/profile
source /etc/profile
```

1. Verify the environment:
   ```
   go version
   ```

Demo

Implement a simple dice roller application using the Gin framework.

1. Initialize the project:
   ```
   go mod init demo
   ```
2. Compile the service code:
   ```
   package main

   import (
       "log"
       "math/rand"
       "github.com/gin-gonic/gin"
   )

   func main() {
       // Routes
       r := gin.Default()
       r.GET("/rolldice", rolldice)

       // Run the server
       log.Println("Server starting on :8080...")
       r.Run(":8080")
   }

   func rolldice(c *gin.Context) {
       c.JSON(200, gin.H{
           "roll":    rand.Intn(6) + 1,
           "message": "Dice rolled!",
       })
   }
   ```
3. Download and update the dependencies:
   ```
   go mod tidy
   ```
4. Run the startup command:
   ```
   go run . &
   curl http://localhost:8080/rolldice
   ```

Using OTelSDK to Report Data over the gRPC Protocol

Step 1 Use the OTelSDK for auto tracking.

Step 2 Add dependencies and initialize the auto tracking SDK:

```
package main

import (
    "log"
    "math/rand"
    "github.com/gin-gonic/gin"
    "context"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
    "go.opentelemetry.io/contrib/instrumentation/github.com/gin-gonic/gin/otelgin"
)

var (
```



```
    SERVICE_NAME = "Application. Component. Environment"
    APM_ENDPOINT = "Reporting address"
    APM_TOKEN = "Authentication information"
)

func main() {
    cleanup := initTracer()
    defer cleanup(context.Background())

    // Routes
    r := gin.Default()

    //automatic instrumentation
    r.Use(otelgin.Middleware(SERVICE_NAME))

    r.GET("/rolldice", rolldice)

    // Run the server
    log.Println("Server starting on :8080...")
    r.Run(":8080")
}

func rolldice(c *gin.Context) {
    c.JSON(200, gin.H{
        "roll":    rand.Intn(6) + 1,
        "message": "Dice rolled!",
    })
}

func initTracer() func(context.Context) error {
    headers := map[string]string{"Authentication": APM_TOKEN}
    exporter, err := otlptracegrpc.New(
        context.Background(),
        otlptracegrpc.WithInsecure(),
        otlptracegrpc.WithEndpoint(APM_ENDPOINT),
        otlptracegrpc.WithHeaders(headers),
    )

    if err != nil {
        log.Fatalf("Failed to create exporter: %v", err)
    }

    resources, err := resource.New(
        context.Background(),
        resource.WithAttributes(
            attribute.String("service.name", SERVICE_NAME),
            attribute.String("library.language", "go"),
        ),
    )

    if err != nil {
        log.Fatalf("Could not set resources: %v", err)
    }

    otel.SetTracerProvider(
        sdktrace.NewTracerProvider(
            sdktrace.WithSampler(sdktrace.AlwaysSample()),
            sdktrace.WithBatcher(exporter),
            sdktrace.WithResource(resources),
        ),
    )
    return exporter.Shutdown
}
```

Step 3 Download and update the dependencies:

```
go mod tidy
```

Step 4 Run the demo application:

```
go run . &
curl http://localhost:8080/rolldice
```

Step 5 Log in to the **APM console**.

Step 6 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 7 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 8 In the tree on the left, locate an environment and then go to the **URL** tab page to view the URL calling data.

Step 9 In the navigation pane, choose **Link Trace** > **Tracing**.

----End

10.4 Reporting PHP Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a PHP application to APM and report trace data.

Constraint

For non-intrusive access, PHP 8.0 or later must be used.

Environment Preparation

1. Prepare PHP 8.0+, PECL, and Composer.
2. Run the installation command:
   ```
   apt update
   apt install -y php-pear
   apt install composer
   ```
3. Verify the environment:
   ```
   pecl version
   composer --version
   ```

Demo

1. Initialize the demo application.

   In the empty project directory, initialize the **composer.json** file.
   ```
   composer init \
   --no-interaction \
   --require slim/slim:"^4" \
   ```

```
--require slim/psr7:"^1"

composer update
```

2. Compile the application code. Create an **index.php** file:

```
<?php
use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;
require __DIR__ . '/vendor/autoload.php';

$app = AppFactory::create();

$app->get('/rolldice', function (Request $request, Response $response) {
    $result = random_int(1,6);
    $response->getBody()->write(strval($result));
    return $response; });

$app->run();
```

3. Run the demo application:

```
php -S localhost:8080 &
curl http://localhost:8080/rolldice
```

Non-Intrusive Access

Step 1 Install the required build tool:

```
sudo apt-get install gcc make autoconf
```

Step 2 Install the extended component using the build tool:

```
sudo apt-get install php-dev
pecl install opentelemetry
pecl install grpc   //It takes about 20 to 30 minutes to install the component.
pecl install protobuf-3.21.12  // Compatible with PHP 8.
```

Select the required plug-in version based on PHP version. For example, for PHP 8.0, use OpenTelemetry 1.1.3 or earlier and Protobuf 3.21.0 or a similar version.

Step 3 Enable the extended component.

1. Locate the **php.ini** configuration file loaded in the current environment:

```
php --ini | grep "Configuration File"
```

2. Add the following content to the PHP file:

```
[opentelemetry]
extension=opentelemetry.so[
[PHP]
extensionxtension=grpc.so
extensionextension=protobuf.so
```

Step 4 Perform verification:

```
php --ri opentelemetry
php --ri grpc
php --ri protobuf
```

Step 5 Install the required dependencies for OpenTelemetry auto tracking:

```
composer config allow-plugins.php-http/discovery false
composer require open-telemetry/sdk
composer require open-telemetry/exporter-otlp
composer require open-telemetry/transport-grpc
composer require open-telemetry/opentelemetry-auto-slim
composer update
```

Step 6 Add the cloud service configuration parameters to environment variables and run the startup command:

```
env OTEL_EXPORTER_OTLP_ENDPOINT=http://xxx.xxx.xx.xxx:4317 \
OTEL_EXPORTER_OTLP_HEADERS=Authentication=LV***9l \
```



```
OTEL_EXPORTER_OTLP_PROTOCOL=grpc \
OTEL_PHP_AUTOLOAD_ENABLED=true \
OTEL_SERVICE_NAME=Application name.Component name.Environment name \
php -S localhost:8080 &
```

Step 7 Trigger a request:

```
curl -kv http://localhost:8080/rolldice
```

```
root@ecs-czf:/otel# [Tue Sep  2 17:11:43 2025] PHP 8.3.6 Development Server (http://localhost:8080) started

root@ecs-czf:/otel# curl http://localhost:8080/rolldice
[Tue Sep  2 17:11:51 2025] 127.0.0.1:44176 Accepted
[Tue Sep  2 17:11:51 2025] 127.0.0.1:44176 [200]: GET /rolldice
[Tue Sep  2 17:11:51 2025] 127.0.0.1:44176 Closing
4root@ecs-czf:/otel# curl http://localhost:8080/rolldice
[Tue Sep  2 17:11:52 2025] 127.0.0.1:44190 Accepted
[Tue Sep  2 17:11:52 2025] 127.0.0.1:44190 [200]: GET /rolldice
[Tue Sep  2 17:11:52 2025] 127.0.0.1:44190 Closing
6root@ecs-czf:/otel# curl http://localhost:8080/rolldice
```

Step 8 Log in to the **APM console**.

Step 9 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 10 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 11 In the tree on the left, locate an environment and then go to the **URL** tab page to view the URL calling data.

Step 12 In the navigation pane, choose **Link Trace** > **Tracing**.

----End

10.5 Reporting Node.js Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a Node.js application to APM and report trace data.

Constraint

Node.js 14 or later must be used.

Environment Preparation

1. Prepare the node and npm tools.
2. Run the installation command:
   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
   \. "$HOME/.nvm/nvm.sh"
   nvm install 18
   ```
3. Verify the environment:
   ```
   node -v
   npm -v
   ```

Demo

Use the Express framework to implement the dice rolling API.

1. Initialize the project:
   ```
   npm init -y
   ```
2. Install project dependencies:
   ```
   npm install express
   ```
3. Compile the service code. Create an **app.js** file:
   ```
   const express = require('express');
   const PORT = parseInt("8080");
   const app = express();

   app.get('/rolldice', (req, res) => {
     res.send(getRandomNumber(1, 6).toString());
   });

   function getRandomNumber(min, max) {
     return Math.floor(Math.random() * (max - min + 1) + min);
   }

   app.listen(PORT, () => {
     console.log(`Listening for requests on http://localhost:${PORT}`);
   });
   ```
4. Run the demo application:
   ```
   node app.js &
   curl http://localhost:8080/rolldice
   ```

Non-Intrusive Installation

Step 1 Install the dependencies:

```
npm install @opentelemetry/sdk-node \
  @opentelemetry/api \
  @opentelemetry/auto-instrumentations-node \
  @opentelemetry/sdk-metrics \
  @opentelemetry/sdk-trace-node
```

Step 2 Interconnect with OpenTelemetry:

```
export OTEL_TRACES_EXPORTER="otlp"
export OTEL_EXPORTER_OTLP_PROTOCOL='grpc'
export OTEL_EXPORTER_OTLP_ENDPOINT="Access address"
export OTEL_NODE_RESOURCE_DETECTORS="env,host,os"
export OTEL_SERVICE_NAME="Application name.Component name.Environment name"
export OTEL_EXPORTER_OTLP_HEADERS="Authentication= Authentication information"
export NODE_OPTIONS="--require @opentelemetry/auto-instrumentations-node/register"
```

Step 3 Run the startup command:

```
node app.js &
curl http://localhost:8080/rolldice
```

```
root@ecs-czf:/otel# node app.js &
[1] 7169
root@ecs-czf:/otel# Listening for requests on http://localhost:8080

root@ecs-czf:/otel#
root@ecs-czf:/otel# curl http://localhost:8080/rolldice
2root@ecs-czf:/otel# curl http://localhost:8080/rolldice
2root@ecs-czf:/otel# curl http://localhost:8080/rolldice
6root@ecs-czf:/otel# curl http://localhost:8080/rolldice
5root@ecs-czf:/otel# curl http://localhost:8080/rolldice
2root@ecs-czf:/otel# curl http://localhost:8080/rolldice
3root@ecs-czf:/otel# curl http://localhost:8080/rolldice
2root@ecs-czf:/otel# curl http://localhost:8080/rolldice
6root@ecs-czf:/otel#
```

Step 4 Log in to the **APM console**.

Step 5 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 6 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 7 In the tree on the left, locate an environment and then go to the **URL** tab page to view the URL calling data.

Step 8 In the navigation pane, choose **Link Trace** > **Tracing**.

----End

10.6 Reporting .NET Application Data Using OpenTelemetry

Huawei Cloud APM is compatible with OpenTelemetry and can directly receive the trace data reported using the OpenTelemetry SDK or Agent. This section describes how to use OpenTelemetry to connect a .NET application to APM and report trace data.

Constraint

.NET SDK 8 or later must be used.

Environment Preparation

1. Prepare .NET SDK 8.
2. Run the installation command:
   ```
   sudo apt-get update &&  sudo apt-get install -y dotnet-sdk-8.0
   ```
3. Verify the environment:
   ```
   dotnet --version
   ```

Demo

1. Initialize the application:
   ```
   dotnet new web
   ```
2. Replace the content in the **Program.cs** file in the current directory with the following content:
   ```
   using System.Globalization;

   var builder = WebApplication.CreateBuilder(args);
   var app = builder.Build();
   var logger = app.Logger;

   int RollDice()
   {
       return Random.Shared.Next(1, 7);
   }

   string HandleRollDice(string? player)
   {
       var result = RollDice();
       if (string.IsNullOrEmpty(player))
       {
           logger.LogInformation("Anonymous player is rolling the dice: {result}", result);
       }
       else
       {
           logger.LogInformation("{player} is rolling the dice: {result}", player, result);
       }
       return result.ToString(CultureInfo.InvariantCulture);
   }
   ```



```
app.MapGet("/rolldice/{player?}", HandleRollDice);
app.Run();
```

3. Replace the content in the **launchSettings.json** file in the **Properties** directory with the following content:

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:8080",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

4. Build and run the demo application in the directory where **Program.cs** is stored:

```
dotnet build
dotnet run
curl  http://localhost:8080/rolldice
```

Non-Intrusive Access

Step 1 Install the dependencies for auto tracking:

```
curl -L -O https://github.com/open-telemetry/opentelemetry-dotnet-instrumentation/releases/latest/
download/otel-dotnet-auto-install.sh
chmod +x otel-dotnet-auto-install.sh
./otel-dotnet-auto-install.sh
```

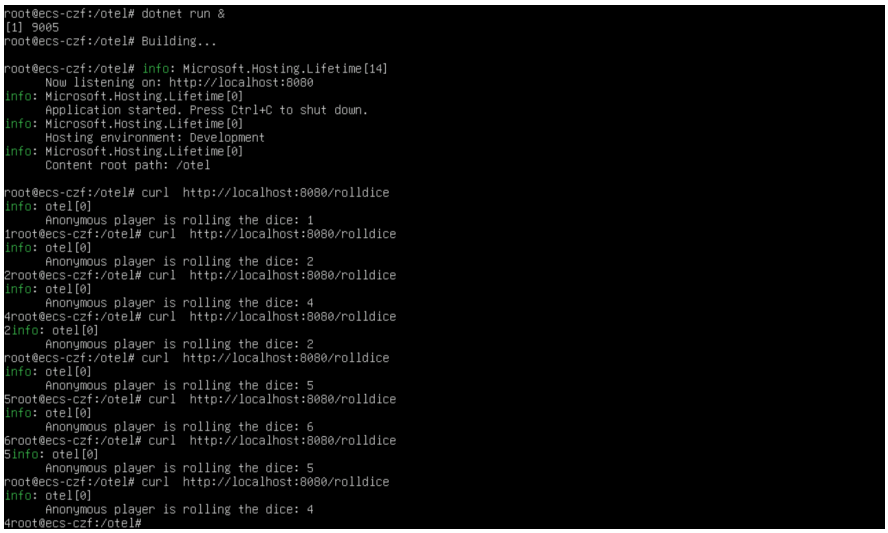
Step 2 Add environment variables:

```
export OTEL_TRACES_EXPORTER=otlp \
  OTEL_EXPORTER_OTLP_PROTOCOL=grpc \
  OTEL_METRICS_EXPORTER=none \
  OTEL_LOGS_EXPORTER=none \
  OTEL_SERVICE_NAME=Application.Component.Environment \
  OTEL_EXPORTER_OTLP_ENDPOINT=Access address \
  OTEL_EXPORTER_OTLP_HEADERS="Authentication=Authentication information"

. $HOME/.otel-dotnet-auto/instrument.sh
```

Step 3 Run the demo application:

```
dotnet run &
curl  http://localhost:8080/rolldice
```



Step 4 Log in to the **APM console**.

Step 5 Click ☰ on the left and choose **Management & Governance** > **Application Performance Management**.

Step 6 In the navigation pane, choose **Link Trace** > **Metrics**.

Step 7 In the tree on the left, locate an environment and then go to the **URL** tab page to view the URL calling data.

Step 8 In the navigation pane, choose **Link Trace** > **Tracing**.

----End

