应用性能管理
2.0
最佳实践

文档版本 01

发布日期 2024-06-20

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 定位请求异常原因

背景信息

在外部请求激增、负载突变等场景下，极易出现应用性能问题，比如外部请求响应变慢、部分请求异常等。快速识别发现、定位处理应用性能问题成为越来越常见的日常运维场景。

APM作为云应用性能问题诊断服务，拥有强大的分析工具，通过拓扑图、调用链可视化地展现应用状态、调用过程、用户对应用的各种操作，快速定位问题和改善性能瓶颈。

例如，通过APM拓扑功能可视化服务间的调用关系，迅速找到有问题的实例；通过APM调用链功能下钻到服务内部，根据出现问题的方法调用链路，确认问题根因。

适用场景

- 应用日常巡检，监控应用时延、吞吐量、错误数等性能指标。
- 应用异常调用快速定位。

操作步骤

步骤1 登录应用性能管理控制台。

步骤2 在左侧导航栏选择“应用监控 > 指标”。

步骤3 选择“接口调用”页签，进入监控页面，查看接口调用页面中各类指标，调用次数、错误次数、时延等信息。

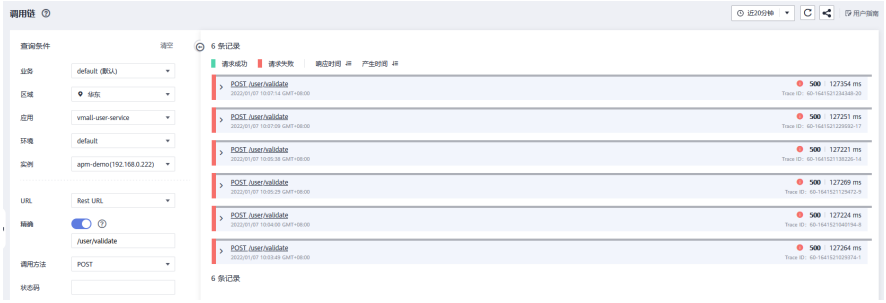
图 1-1 查看接口调用

步骤4 单击出现问题的url请求，进入调用链搜索界面。

图 1-2 接口详情

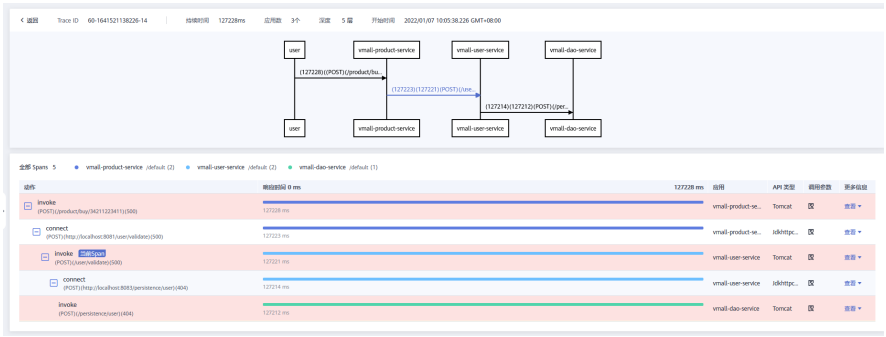
步骤5 在调用链搜索界面，查看失败/高时延调用链。

图 1-3 查看调用链



步骤6 单击url，获取调用链详细信息，确定问题根因。

图 1-4 调用链详情



----结束

2 调用链搜索 span 信息

背景信息

在分布式架构下，微服务之间的调用情况日趋复杂，在外部请求响应变慢、部分请求异常等场景下，想要快速定位哪个环节存在异常，您可以在业务在调用链路查询页面，通过TraceId精确查询调用链路详细情况，或结合多种条件筛选查询调用链路。

操作步骤

步骤1 登录APM控制台。

步骤2 在左侧导航栏选择“应用监控 > 调用链”，进入调用链界面。

步骤3 输入如下查询条件，单击“查找Trace”，页面右侧展示查找结果。

图 2-1 调用链查询结果

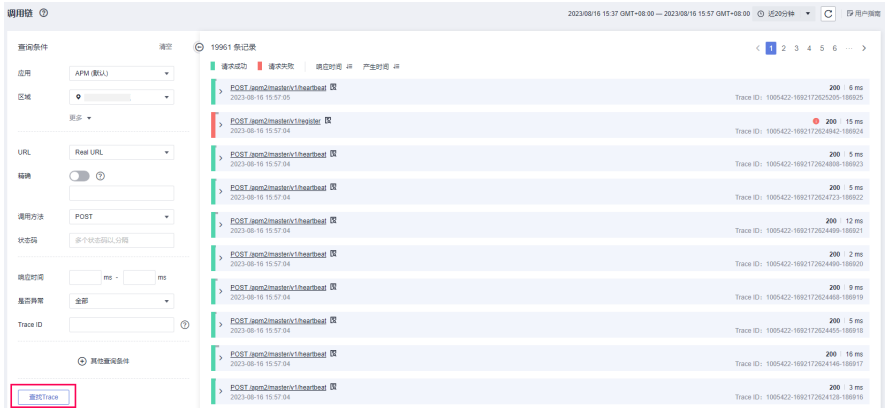


表 2-1 调用链查询条件

查询条件	具体含义	必填/选填
应用	调用链所在应用。	必填
区域	调用链所在区域。	必填
组件	调用链所在组件。	选填

查询条件	具体含义	必填/选填
环境	调用链所在环境。	选填
实例	调用链所在实例。	选填
URL	调用链的URL，分为Rest URL和Real URL两种搜索条件：Rest URL为restful风格的URL，URL中带有变量名称，如/apm/get/{id}；Real URL为实际访问的URL。	选填
精确	对URL是否精确匹配，开启状态下为精确查询URL，不开启则进行模糊查询。	选填
调用方法	调用链的HttpMethod。	选填
状态码	调用链返回的HTTP状态码。	选填
响应时间	调用链的响应时间范围，可以填写最小响应时间和最大响应时间搜索调用链，两个值都可以为空。	选填
是否异常	调用链是否有异常。	选填
Trace ID	调用链的TraceID，填写该搜索条件后，其他搜索条件全部失效，只根据该TraceID搜索。	选填

步骤4 单击“其他查询条件”，展示“自定义参数”、“全局Trace ID”以及“应用码”三个查询条件。

图 2-2 其他条件

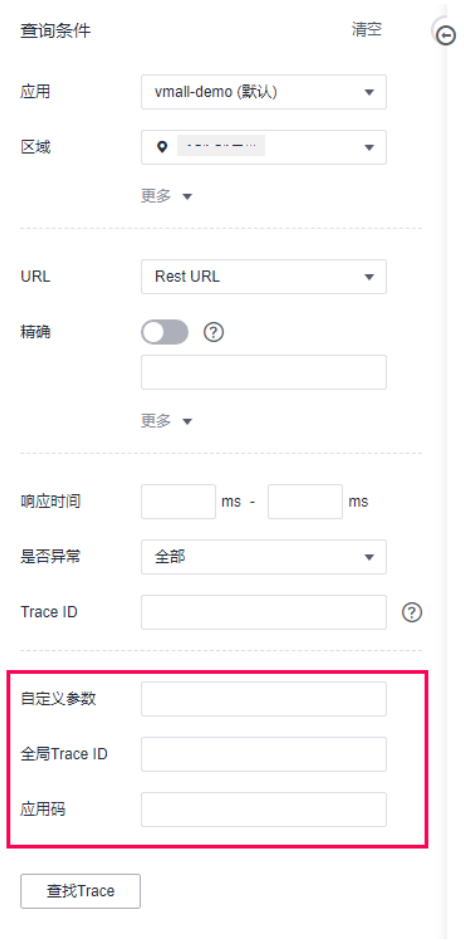


表 2-2 调用链查询条件

查询条件	具体含义	必填/选填
自定义参数	已配置url监控项的拦截header指定key值、拦截url参数指定key值、拦截cookie指定key值参数后，在这里可以设置key=value进行搜索。	选填
全局Trace ID	调用链的全局TraceID，填写该搜索条件后，其他搜索条件全部失效，只根据该全局TraceID搜索。	选填
应用码	已配置url监控项的业务code采集长度限制、解析业务code的key、业务code的正确值参数后，会采集相应的应用码，这里可以根据应用码进行搜索。	选填

- 自定义参数

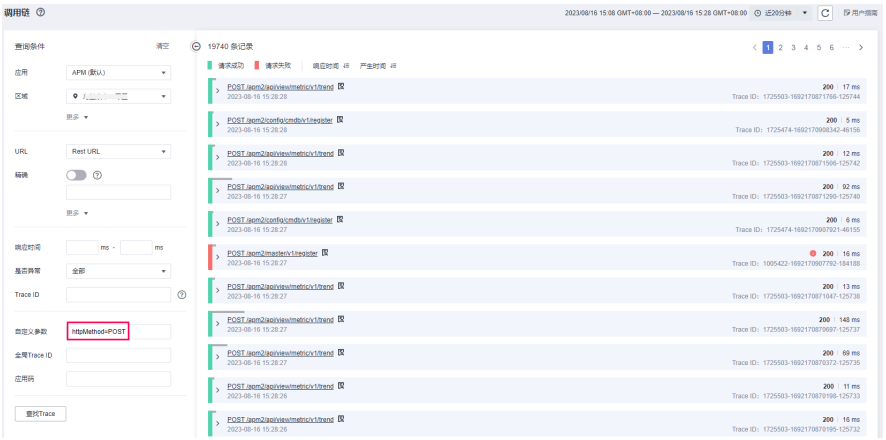
  使用方法

  a. 配置url监控项的拦截header指定key值、拦截url参数指定key值、拦截cookie指定key值参数。具体方法参见**配置url监控项**。

b. 在“自定义参数”后的框中，填写对应的参数以及参数值。

c. 单击“查找Trace”，页面右侧展示查找结果。

图 2-3 自定义参数查询结果

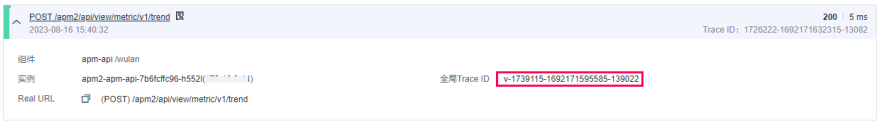


- 全局Trace ID使用方法

  使用方法

  a. 单击待查看的调用链前的 ，查看全局Trace ID。

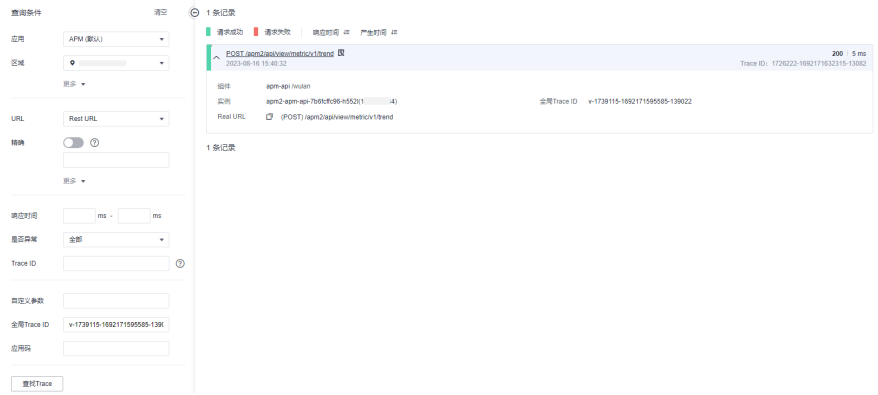
图 2-4 获取全局 Trace ID



b. 在“全局Trace ID”后的框中，填写全局Trace ID。

c. 单击“查找Trace”，页面右侧展示查找结果。

图 2-5 查询全局 Trace ID 结果



- 应用码使用方法

  使用方法

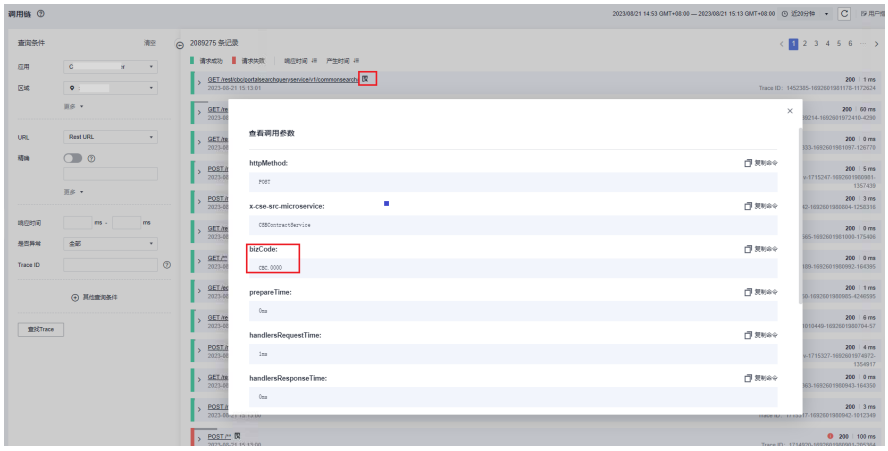
  a. 配置url监控项的业务code采集长度限制、解析业务code的key、业务code的正确值参数。具体方法参见**配置url监控项**。

图 2-6 url 监控项

b. 在左侧导航栏选择“应用监控 > 调用链”，进入调用链界面。

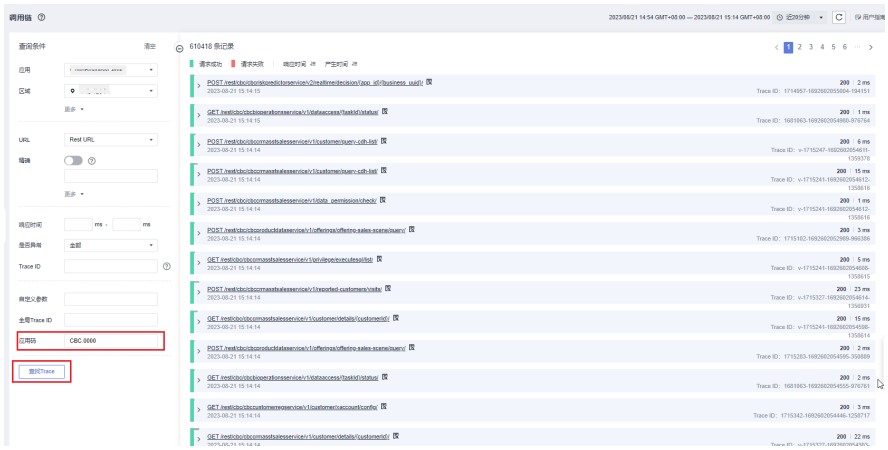
c. 单击 ，查看对应的业务code的值。业务code的值即“应用码”。

图 2-7 查看业务 code



d. 在“应用码”后的框中，填写应用码。

图 2-8 查找 code 对应的调用链



e. 单击“查找Trace”，页面右侧展示查找结果。

----结束

3 云下业务接入云上 APM

背景信息

用户需要云下业务接入云上APM，但云专线无法打通网络。因此，用户需要用代理的方式接入APM，不知道如何操作。

配置方法

接入APM的机器与APM服务网络无法连通，可以接入代理。

步骤1 配置代理

1. 登录AOM 2.0控制台。
2. 在菜单栏选择“采集管理”，进入“采集管理”界面。
3. 在左侧导航栏中，选择“UniAgent管理 > 代理区域管理”，进入代理区域管理页面。
4. 单击“添加代理机”，配置相关参数信息。

图 3-1 添加代理机

表 3-1 添加代理机参数说明

参数	说明	示例
代理区域	选择已创建的**代理区域**。	region
主机	选择已安装的UniAgent主机。	-
代理IP	配置代理机的IP地址	-
端口	端口号，必须小于或等于65535	-

5. 单击“确认”，完成代理机添加。

步骤2 配置JavaAgent

1. 将javaagent下载到需要接入APM机器的任意目录。

   示例命令：

   ```
   curl -O https://xxx/apm-javaagent-x.x.x.tar
   Agent 2.4.1下载方法：curl -k https://apm2-javaagent-cn-north-4.obs.cn-north-4.myhuaweicloud.com/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-cn-north-4.obs.cn-north-4.myhuaweicloud.com -version 2.4.1; history -cw; history –r
   ```

2. 执行tar命令解压javaagent。

   示例命令：

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

3. 修改javaagent中的apm.config配置文件。将apm.proxy写入配置文件中，如下图所示。

   图 3-2 配置文件

   说明

   - Agent 2.4.1及之后版本支持采用代理接入。格式：apm.proxy=ip:port（此处为aom界面的ip:port）。
   - 获取AK/SK的操作步骤，请参见**访问秘钥**。
   - 获取master.address的操作步骤，请参见**接入地址master.address配置**。

步骤3 重启应用。

1. 修改java进程启动脚本。

   在服务启动脚本的java命令之后，配置apm-javaagent.jar包所在路径，并指定java进程的组件名。

添加-javaagent参数示例：

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

2. 重启应用。

----结束

4 调用链与日志关联

适用范围

常用的一些日志框架logback，log4j等。

举例

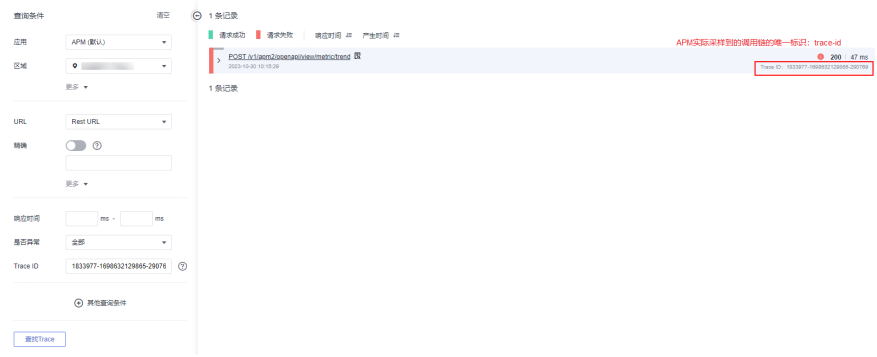
<property name="LOG_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS}} | gtraceid: **%X{apm-gtraceid}** | traceid: **%X{apm-traceid}** | spanId: **%X{apm-spanid}**">

</property>

APM 服务中调用链相关的参数说明

1. apm-traceid: apm服务采集到调用链的唯一标识。

   图 4-1 采集调用链的唯一标识

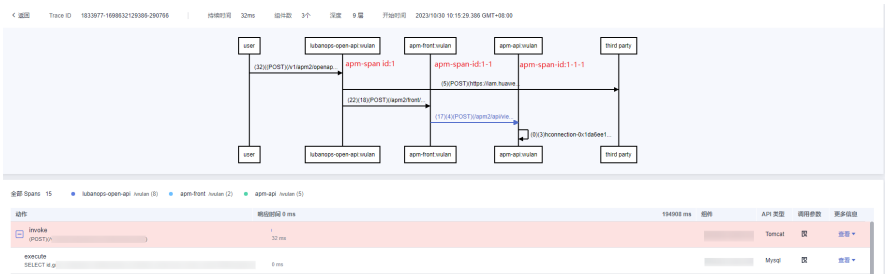
2. apm-gtraceid: apm服务中未被采样到的调用关系的唯一标识。

   说明

   apm服务的调用链具有一定采样率，所以用apm-gtrace-id来表示未被采样的调用链的唯一标识。

3. apm-spanid:在某个调用链的微服务之间调用，表示某一个微服务的id，示例如下。

图 4-2 调用链的微服务之间调用

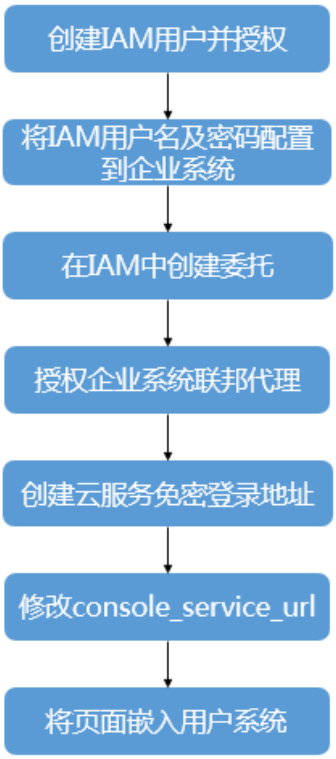


5 结果查询页面内嵌到客户自建系统

背景信息

APM支持将界面嵌入到客户自建系统。通过IAM服务的联邦代理机制实现用户自定义身份代理，再将登录链接嵌入至客户自建系统实现无需在华为云官网登录就可在自建系统界面查看。

操作流程



前提条件

用户自定义创建身份代理并创建登录地址FederationProxyUrl，详细可参考IAM服务的文档“**创建自定义身份代理**”。

操作步骤

用户创建自定义身份代理成功后，您需要执行如下步骤，实现页面的内嵌。

步骤1 将FederationProxyUrl中的console_service_url修改为云服务console地址。

console_service_url示例：

对应业务模块	示例url
应用监控-指标	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/appindex/business/detail?leftMenuCollapsed=true
应用监控-调用链	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/appchain?leftMenuCollapsed=true
链路追踪-指标	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/trace/metric/environment/view?leftMenuCollapsed=true
链路追踪-调用链	https://console.huaweicloud.com/apm2/?region={regionId}&cfModuleHide=header_sidebar_floatlayer#/console/trace/chain?leftMenuCollapsed=true

参数名称	说明
regionId	表示当前您所在的区域，可在华为云官网登录云服务后在浏览器的地址栏中获取。例如cn-north-4。
cfModuleHide	值header_sidebar_floatlayer，表示隐藏华为云console页面页头页脚和菜单栏。
leftMenuCollapsed	true表示隐藏左侧菜单，false表示不隐藏左侧菜单。

步骤2 使用iframe将apm页面嵌入用户系统，示例代码如下：

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

----结束

6 公网如何接入 APM

前提条件

1. 已购买华为云弹性云服务器ECS作为跳板机。
2. 弹性云服务器已绑定弹性IP地址。

说明

- 推荐CentOS 6.5 64bit及其以上版本的镜像， 最低规格为1vCPUs | 1GB，推荐规格为2vCPUs | 4GB。
- 推荐使用iptables作为跳板机转发实现。

操作步骤

请先在华为云上购买一台弹性云服务器作为跳板机，然后执行如下操作。

步骤1 登录弹性云服务器，修改跳板机ECS的安全组规则。

1. 在ECS详情页，单击安全组页签，进入安全组列表页。
2. 单击具体的安全组名，单击“更改安全组规则”，进入安全组详情页。
3. 在该安全组详情页，单击“入方向规则 > 添加规则 ”，按表6-1添加安全组规则。

表 6-1 安全组规则

方向	协议	端口	说明
入方向	TCP	41333,41335	JavaAgent发送数据到跳板机的端口列表。

步骤2 获取APM上报地址，参见接入地址master.address配置。

步骤3 以root用户登录跳板机，执行iptables转发命令。

说明

如果没有iptables相关服务，需要先安装，命令如下。

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

1. 开启数据转发功能。

```
# 编辑文件
vim /etc/sysctl.conf
# 增加一行
net.ipv4.ip_forward=1
# 使数据转发功能生效
sysctl -p
```

2. 将本机的端口转发到其他地址和端口。

```
# 编辑文件
vim /etc/sysconfig/iptables
# *filter新增
-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
# *nat新增
-A OUTPUT -p tcp --dport 41333 -j DNAT --to-destination {APM上报ip}:41333
-A PREROUTING -p tcp --dport 41333 -j DNAT --to-destination {APM上报ip}:41333
-A POSTROUTING -d {APM上报ip}/32 -p tcp --dport 41333 -j SNAT --to-source {跳板机ip}

-A OUTPUT -p tcp --dport 41335 -j DNAT --to-destination {APM上报ip}:41335
-A PREROUTING -p tcp --dport 41335 -j DNAT --to-destination {APM上报ip}:41335
-A POSTROUTING -d {APM上报ip}/32 -p tcp --dport 41335 -j SNAT --to-source {跳板机ip}

# 如果存在以下规则，需要删除
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
```

3. 重启iptables。

```
systemctl restart iptables
```

4. 验证端口转发是否成功。

```
curl -kv https://{跳板机ip}:41333
curl -kv https://{跳板机ip}:41335
```

步骤4 修改javaagent中的apm.config配置文件。

```
master.address=https://{跳板机公网ip}:41333
access.address={跳板机公网ip}:41335
```

步骤5 重新部署应用。

----结束

7 Kubernetes 如何接入 APM

前提条件

部署APM Agent时，必须确保接入APM的机器与APM服务网络连通，Agent才能正常工作。

可使用Telnet命令测试目标机器与APM服务器网络是否连通。例如，以检查华北-北京四区域的连通性为例，请登录应用所部署的机器，并输入命令**telnet 100.125.12.108 41333**。

操作步骤

编辑deployment.yaml。

步骤1 在volumes中增加一个emptyDir。

```
volumes:
- name: paas-apm2
emptyDir: {}
```

步骤2 在containers.volumeMounts中增加moutPath。

```
volumeMounts:
- name: paas-apm2
mountPath: /paas-apm2/javaagent/
```

步骤3 在env中增加JAVA_TOOL_OPTIONS环境变量。

```
env:
- name: JAVA_TOOL_OPTIONS
value: '-javaagent:/paas-apm2/javaagent/apm-javaagent/apm-javaagent.jar'
```

步骤4 新增initContainers。

```
initContainers:
- name: init-javaagent
image: '{swrAddress}/op_svc_apm/javaagent:2.4.7.2-x86_64'
command:
- /bin/sh
- '-c'
- 'cd /paas-apm2/javaagent/apm-javaagent; /bin/sh init-config.sh -master_address {masterAddress} -app_name {appName} -access_key {accessKey} -access_value {secretKey}'
resources:
limits:
cpu: 250m
memory: 250Mi
requests:
cpu: 250m
memory: 250Mi
```



```
volumeMounts:
- name: paas-apm2
mountPath: /var/init/javaagent
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
imagePullPolicy: Always
```

----结束

8 IDE 如何接入 APM

前提条件

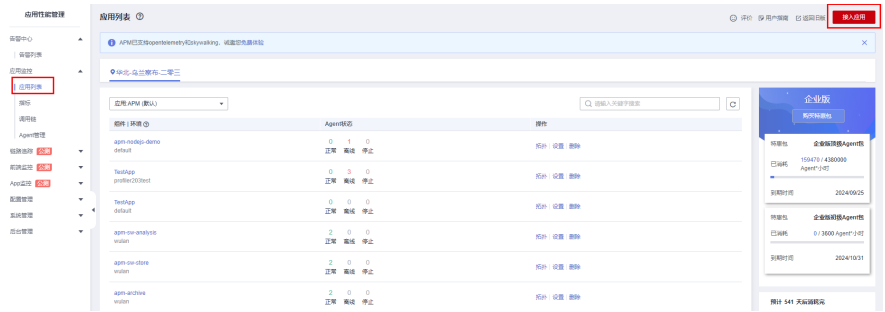
1. 已按APM公网接入指导完成公网接入。

操作步骤

步骤1 下载APM探针。

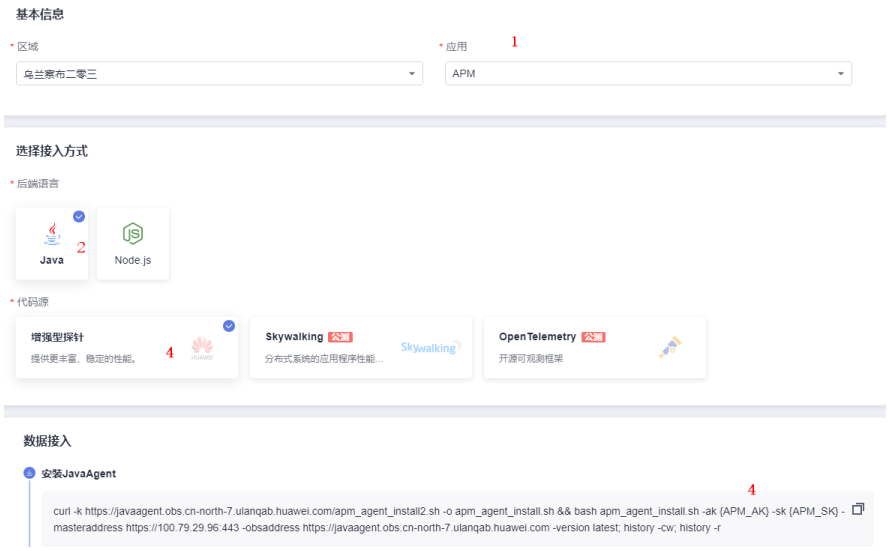
1. 登录管理控制台。
2. 单击左侧 ☰ ，选择“管理与监管 > 应用性能管理 APM”，进入APM服务页面。
3. 在左侧导航栏中选择“应用监控 > 应用列表”。

图 8-1 接入应用



4. 单击“接入应用”，进入接入应用页面。

图 8-2 探针接入



5. 在左侧导航栏选择“系统管理 > 访问密钥”，进入访问密钥页面，获取JavaAgent的APM_AK和APM_SK。详细操作参见**访问密钥**。
6. 替换安装JavaAgent的APM_AK和APM_SK。

步骤2 执行git bash命令。在本地电脑D盘agent目录下，将复制的安装JavaAgent命令执行。

步骤3 修改apm.config文件中的master.address、access.address以及business参数值。

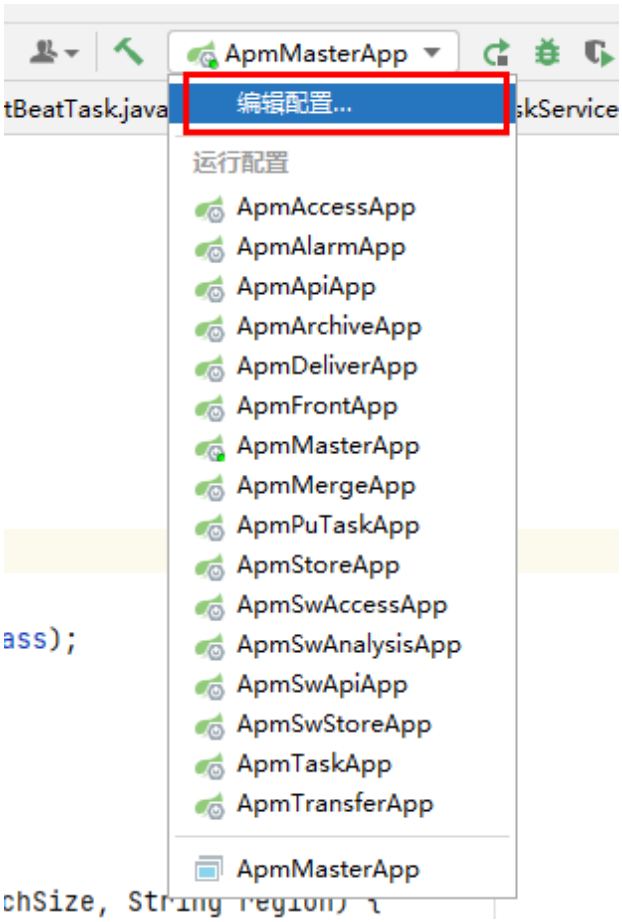
图 8-3 修改配置文件

```
master.address=https://[redacted]
access.address=[redacted]
access.key=mq9g1qq5r228oswp
secret.key=8e3hkak3f3gs8fcf2k3dwy86tpgici1u
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
```

步骤4 单击下拉菜单，选择“编辑配置”。

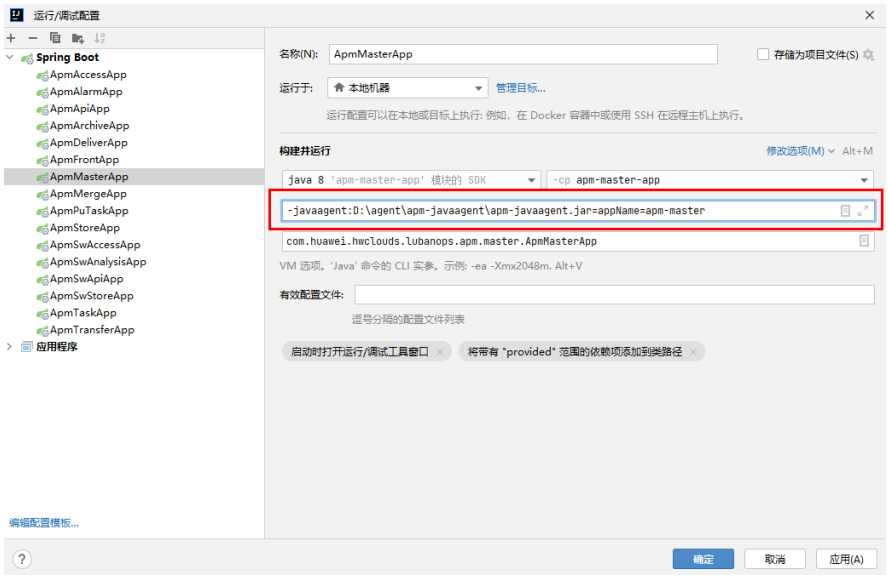
1. 修改IDEA的运行/调试配置。

图 8-4 修改调试配置



2. 在运行/调试配置页面，左侧导航栏中，选择“ApmMasterApp”。右侧“构建并运行”中添加“-javaagent:D:\agent\apm-javaagent\apm-javaagent.jar=appName=apm-master”。

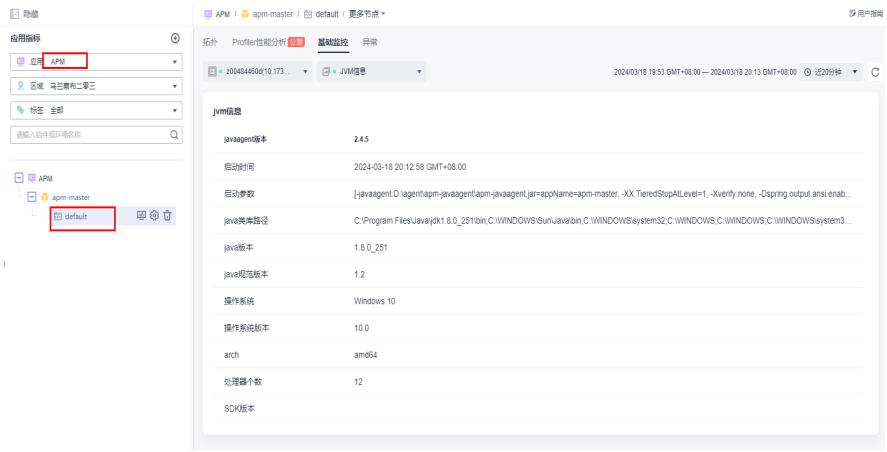
图 8-5 修改构建并运行



3. 单击“确定”。

步骤5 重新运行服务并查看。如果在应用下拉菜单中找到创建的“APM”，则接入成功。

图 8-6 接入成功



----结束

9 如何使用 APM Profiler 定位性能问题

APM profiler 是一种持续性能剖析工具，可以帮助开发者准确找到应用程序中消耗资源最多的代码位置。

前提条件

1. APM Agent 已接入，操作方法参见**开始监控JAVA应用**。
2. Profiler功能已开启，操作方法参见**Profiler性能分析**。
3. 登录应用性能管理控制台。

如何查询并解决 CPU 升高问题

步骤1 在左侧导航栏选择“应用监控 > 指标”。

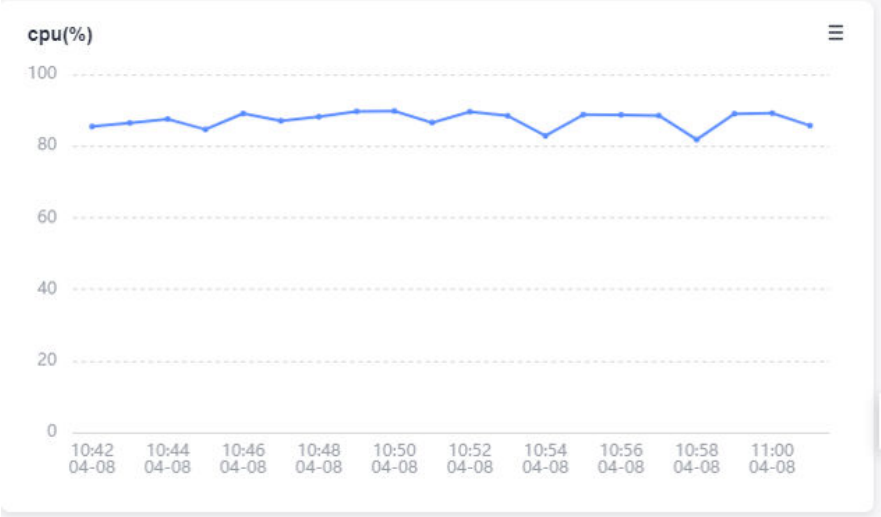
步骤2 在界面左侧树单击待查看基础监控环境后的 。

步骤3 单击“基础监控”，切换至基础监控页签，监控项选择“JVM监控”。

图 9-1 查看 JVM 监控

步骤4 找到“cpu(%)”，发现CPU持续高达80%以上。

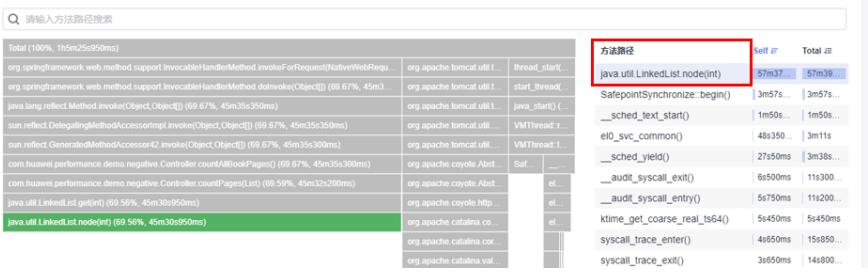
图 9-2 cpu(%)



步骤5 单击“Profiler性能分析”，切换至Profiler性能分析页签。

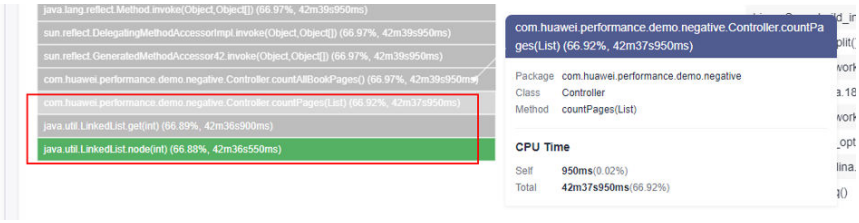
步骤6 单击“性能分析”，Profiler性能分析页面，实例选择“CPU Time”。

图 9-3 Profiler 火焰图



步骤7 分析火焰图数据 从火焰图中可以看到，java.util.LinedList.node(int) 方法占用了 66% 的 CPU，而相应的业务代码方法是 countPages(List)。

图 9-4 Profiler 火焰图分析

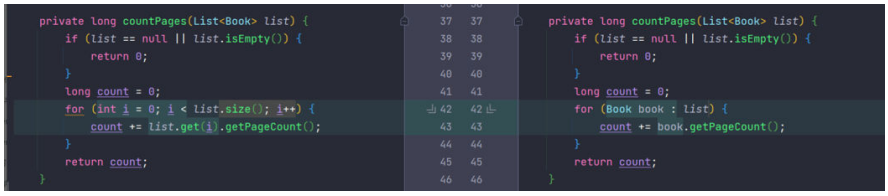


步骤8 分析业务代码，结合代码可以发现该方法countPages(List)是对入参集合list进行下标遍历，而通过火焰图运行时数据发现，传入的是 LinkedList，而LinkedList底层数据结构是链表，通过下标遍历效率会非常差。

图 9-5 代码分析

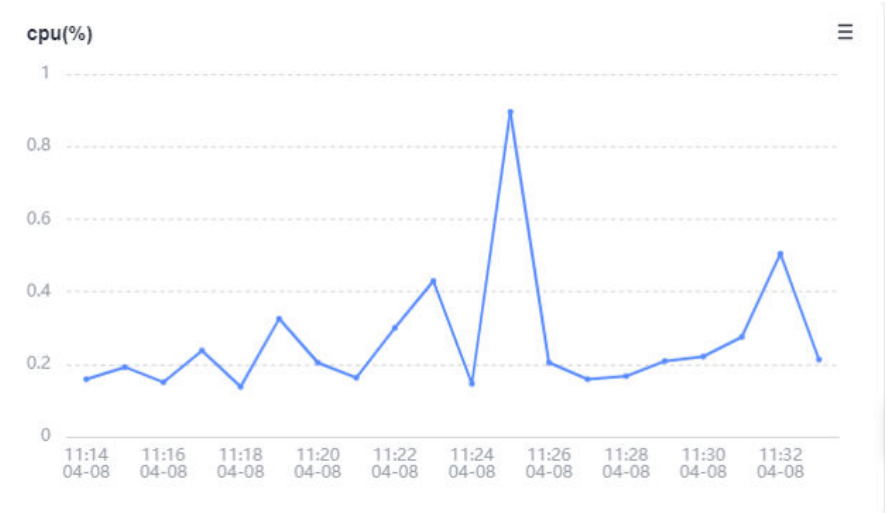
步骤9 修复代码，将list的遍历算法从普通的下标for循环改为增强的for循环。

图 9-6 修复代码



步骤10 优化后，重复**步骤4-步骤5**，发现CPU使用率<1%。

图 9-7 优化后 CPU(%)



----结束

如何查询并解决内存升高问题

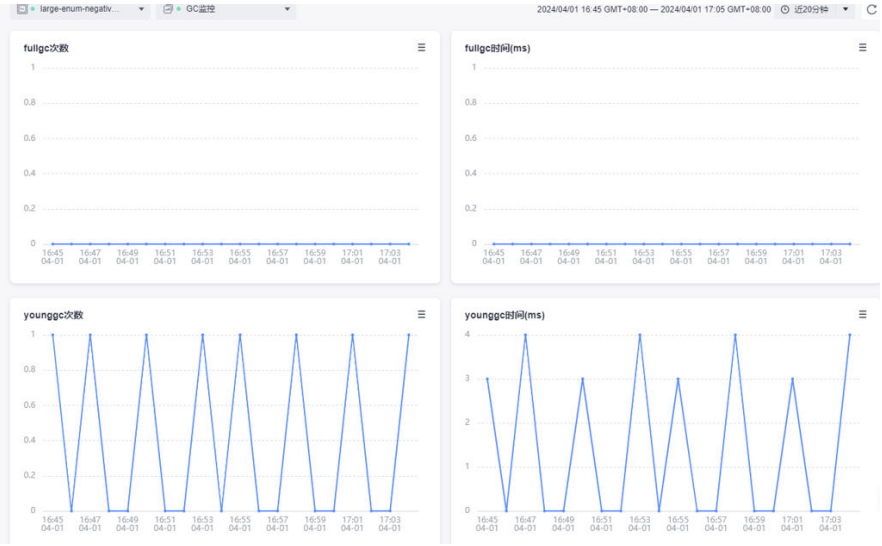
前提条件：开启测试程序，同时设定heap大小为2g（ -Xms2g -Xmx2g ）。

步骤1 在左侧导航栏选择“应用监控 > 指标”。

步骤2 在界面左侧树单击待查看基础监控环境后的 。

步骤3 单击“基础监控”，切换至基础监控页签，监控项选择“ GC监控”，非常频繁的进行gc操作。

图 9-8 查看 GC 监控



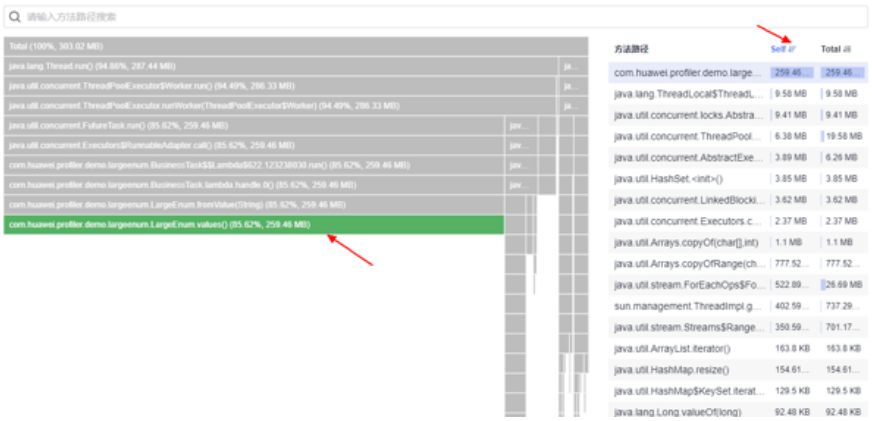
步骤4 监控项选择“JVM监控”，查看JVM监控。

图 9-9 查看 JVM 监控

步骤5 单击“Profiler性能分析”，切换至Profiler性能分析页签。

步骤6 单击“性能分析”，Profiler性能分析页面，实例选择“Memory”。根据右侧Self排序排查，找到分配内存最多的方法。

图 9-10 内存火焰图



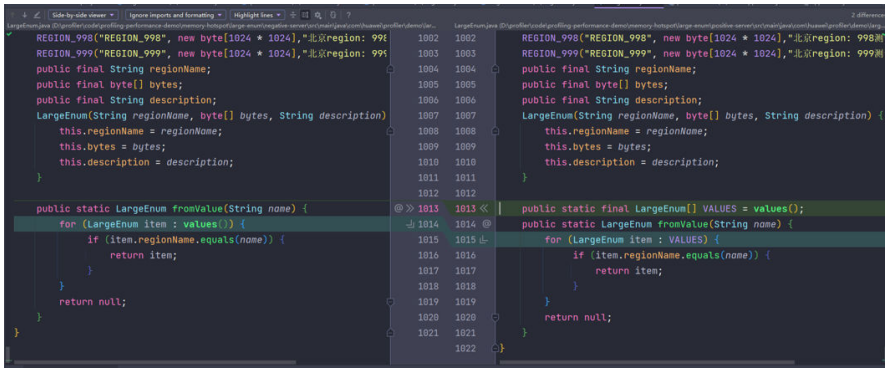
步骤7 查看代码，发现LargeEnum是个枚举类，定义了大量的常量。由于枚举类的方法values() 底层是通过数组clone实现的，即每次调用values()方法，底层会复制一个枚举数组，所以会导致频繁分配堆内存，频繁GC。

图 9-11 查看代码

```
    REGION_997("REGION_997", new byte[1024 * 1024],"北京region: 997测试和开发团队专用"),
    REGION_998("REGION_998", new byte[1024 * 1024],"北京region: 998测试和开发团队专用"),
    REGION_999("REGION_999", new byte[1024 * 1024],"北京region: 999测试和开发团队专用");
    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;
    }
}
```

步骤8 问题修复，将values定义为一个常量，避免频繁调用enum.values()。

图 9-12 问题修复



步骤9 重复**步骤3-步骤6**，发现GC次数大幅下降，并且火焰图中以找不到enum.values()内存分配。

图 9-13 优化后 GC 监控

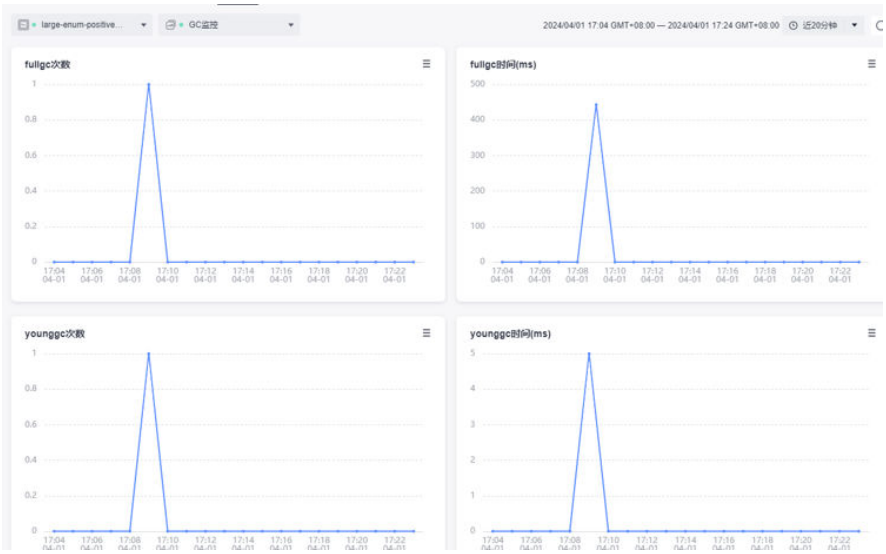


图 9-14 优化后性能分析火焰图

----结束

如何查询并解决接口响应慢问题

步骤1 在左侧导航栏选择“应用监控 > 指标”。

步骤2 在界面左侧树单击待查看基础监控环境后的 。

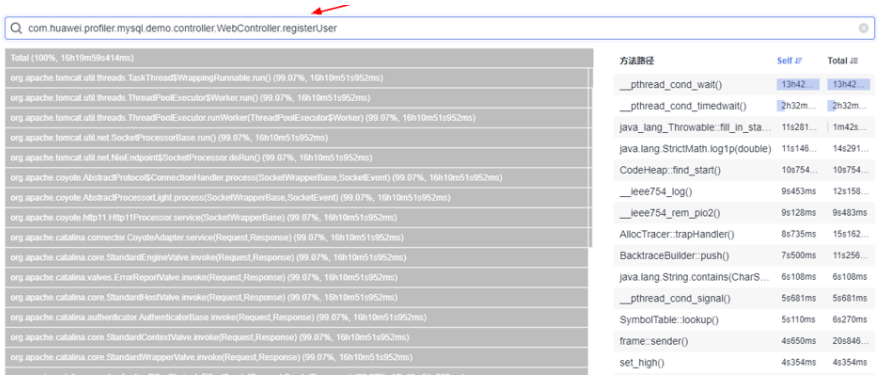
步骤3 单击“接口调用”，切换至接口调用页签。通过APM的接口调用功能发现接口响应慢，平均响应时间80s左右。

图 9-15 接口调用

步骤4 单击“Profiler性能分析”，切换至Profiler性能分析页签。

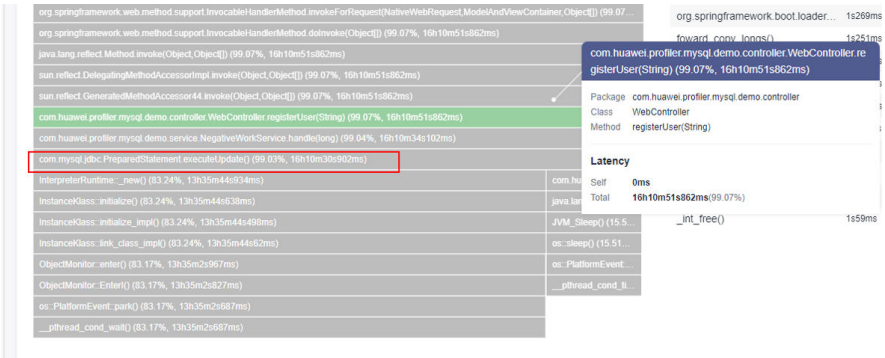
步骤5 单击“性能分析”，Profiler性能分析页面，实例选择“Latency”，输入接口所在的方法。

图 9-16 性能分析



步骤6 排查调用栈，寻找耗时的方法。如下图，NegativeWorkService#handle中executeUpdate()方法耗时最多。

图 9-17 排查调用栈



步骤7 排查NegativeWorkService#handle方法，发现根因是循环内执行数据库插入操作。

图 9-18 排查 NegativeWorkService#handle 方法

```
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 {
```

步骤8 问题修复，改为批量插入数据。

图 9-19 问题修复

```
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 {
```

步骤9 观察接口调用平均响应时间，从80s减少到0.2s。

图 9-20 优化后查询接口调用平均响应时间

----结束

10 如何使用 Profiler 定位 OOM 问题

背景

服务所在容器频繁重启，通过自监控发现重启前fullgc次数会突增（约每分钟20次）。

使用 Profiler 定位 OOM 问题

步骤1 登录管理控制台。

步骤2 单击左侧 ，选择“管理与监管 > 应用性能管理 APM”，进入APM服务页面。

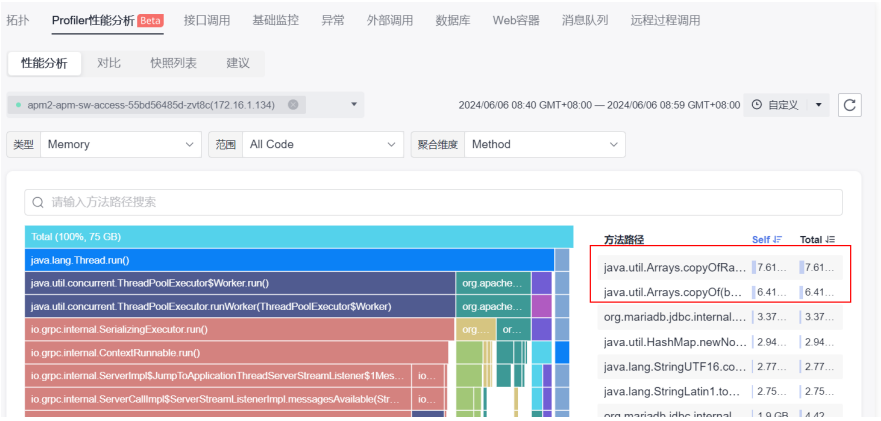
步骤3 在左侧导航栏选择“应用监控 > 指标”。

步骤4 在界面左侧树单击待查看Profiler性能分析环境后的 。

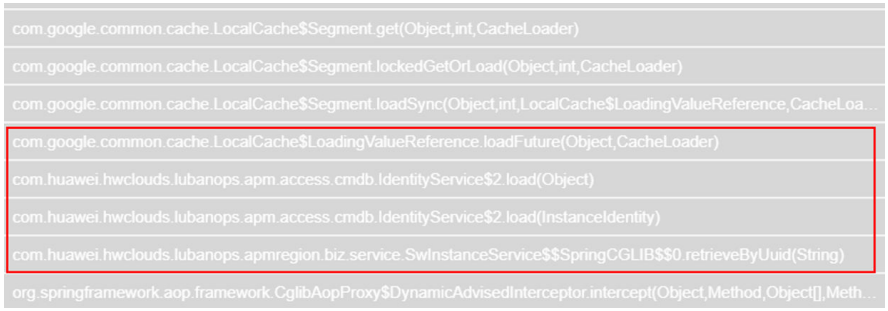
步骤5 单击“Profiler性能分析”，切换至Profiler性能分析页签。

步骤6 单击“性能分析”，进入性能分析页面。

步骤7 选择类型：Memory，范围：All Code，聚合维度：Method，发现有两个方法占用了较多内存。



步骤8 单击“方法路径”列，对应的方法名找到该方法的调用栈，向上找到调用此方法的业务代码。

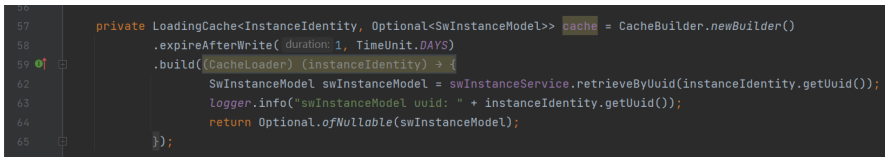


步骤9 定位到业务代码块，该代码使用了一个缓存来存放每个instance的信息。通过自监控查看该sql的调用，发现每分钟会调10万次，进一步证实是缓存失效了。

说明

当查询instance信息时，会先从缓存查，如果查不到再从mysql查。instance信息查到之后存入缓存，防止频繁访问数据库。

步骤10 检查代码发现缓存的key是一个类，该类没有重写equals和hashCode方法。因此，导致缓存通过key去获取value时，会根据key的地址来判断该key是否在缓存中存在。而每次传参进来的key地址都不一样，所以从缓存中查找失败，只能从mysql查找，然后又不断往缓存中存，最终导致OOM问题。



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

----结束

解决方案

给作为key的类重写equals和hashCode方法。该类有个uuid属性，不同instance的uuid不一样，因此可以用uuid来判断两个instance是否相同。

