前端性能监控

实践教程

腾讯云

文档目录

实践教程

Aegis SDK 支持获取请求头和返回头

最近更新时间：2024-06-25 16:23:41

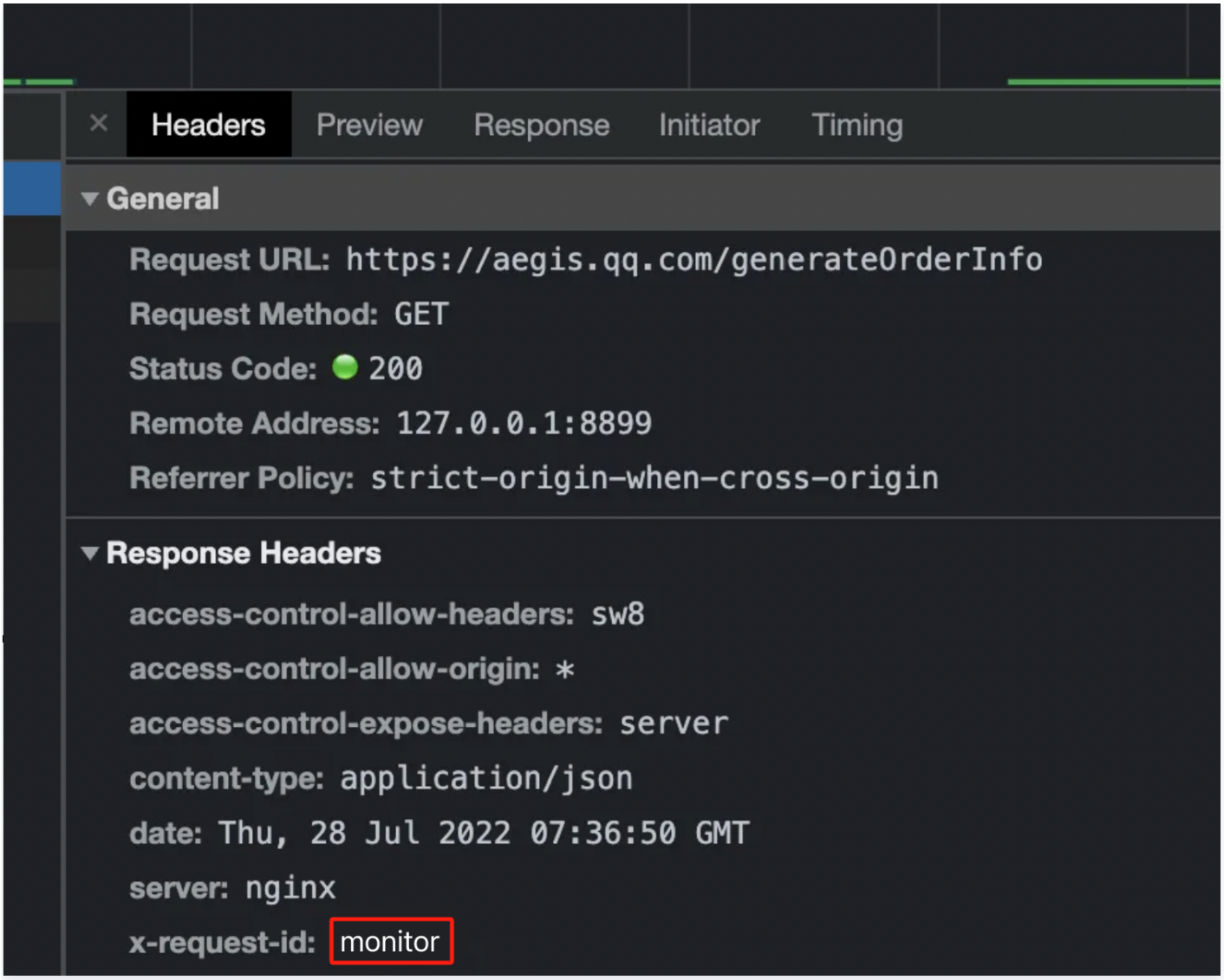
通过记录自定义请求头和响应头的方式，打通前后端链路。

> **说明：**
> 目前仅支持 aegis-web-sdk 和 aegis-mp-sdk。

操作步骤

获取 Response Headers

假设有一个接口 `https://aegis.qq.com/generateOrderInfo`，该接口的有一个自定义的 Response Headers `x-request-id`，值是 `monitor`，为前后端打通的唯一标识。



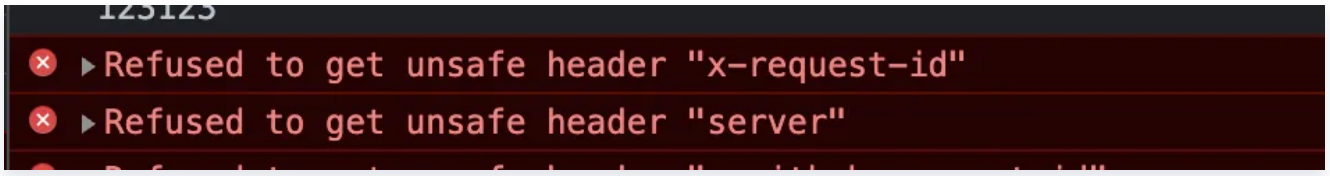
假设需要通过记录这个值到 RUM 来打通前后端的链路，实践步骤如下：

1. 将 SDK 升级到最新版本。
2. 修改初始化 SDK 的方式。

```
new Aegis({
        id: 'xxx',
        reportApiSpeed: true,
        api: {
            apiDetail: true, // 上报接口请求参数和返回值
            reportRequest: true, // 全量接口上报
            reqHeaders: ['sw8'], // 记录 request header
            resHeaders: ['x-request-id', 'content-type', 'server'] //
记录 response header，注意使用小写
        },
    });
```

3. 暴露 unsafe header

完成步骤1和步骤2后，会发现 `content-type` 赋值了，但是 `x-request-id` 和 `server` 还是没有赋值。此时使用 xhr.getResponseHeader() 去获取相关 header，会发现并没有获取值，而且浏览器会出现报错信息。

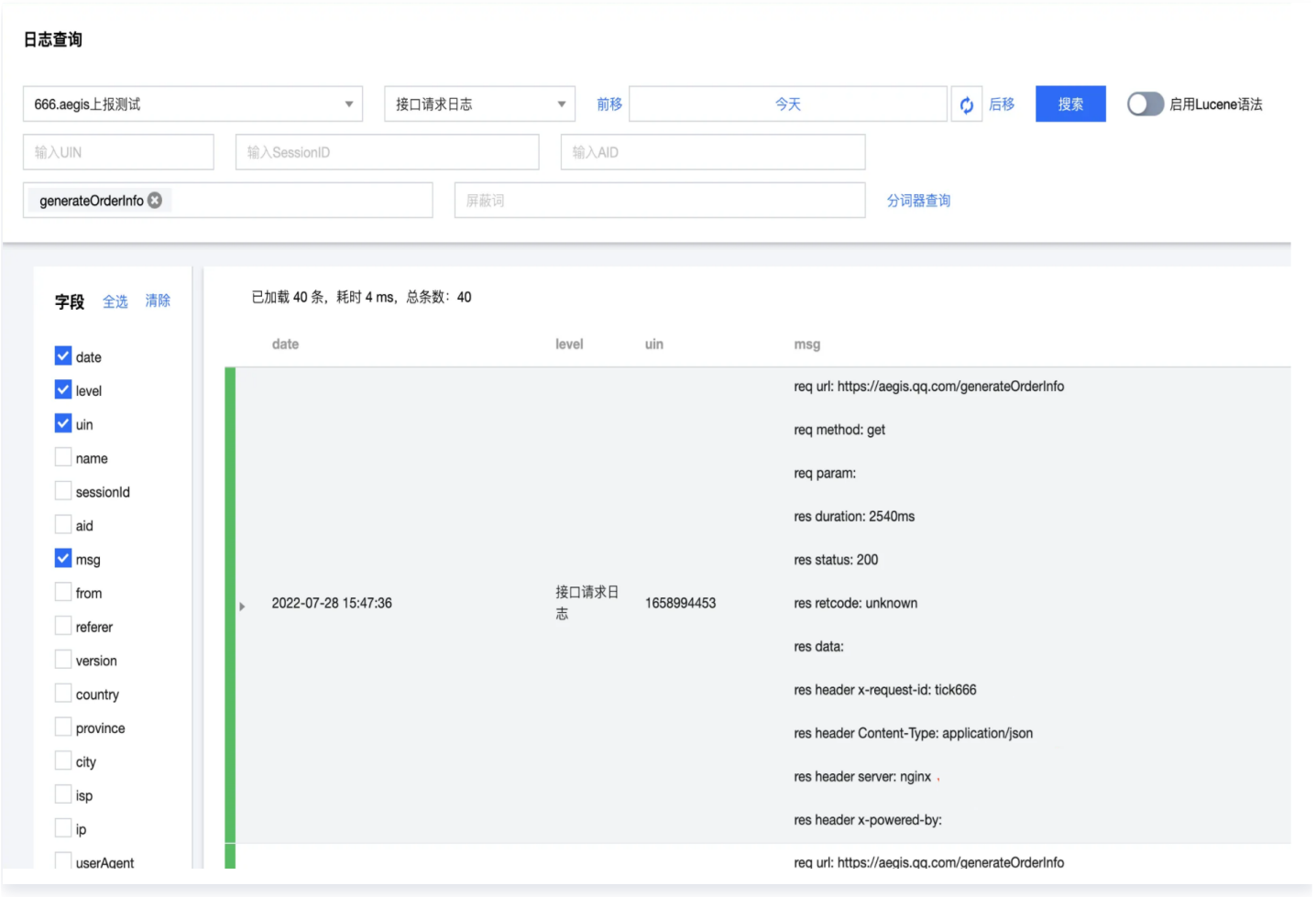


> **说明：**
> 具体报错原因可参见 相关文档 。

此时需要暴露 unsafe header，给需要上报的接口加上 Access-Control-Expose-Headers，如下所示：

```
    location /generateOrderInfo {
         add_header Access-Control-Allow-Origin *;
         add_header Access-Control-Allow-Headers 'sw8';
         add_header Access-Control-Expose-Headers 'x-request-id,
server';
         proxy_pass http://9.139.xx.xx:9301/generateOrderInfo;
     }
```

上报效果如下：

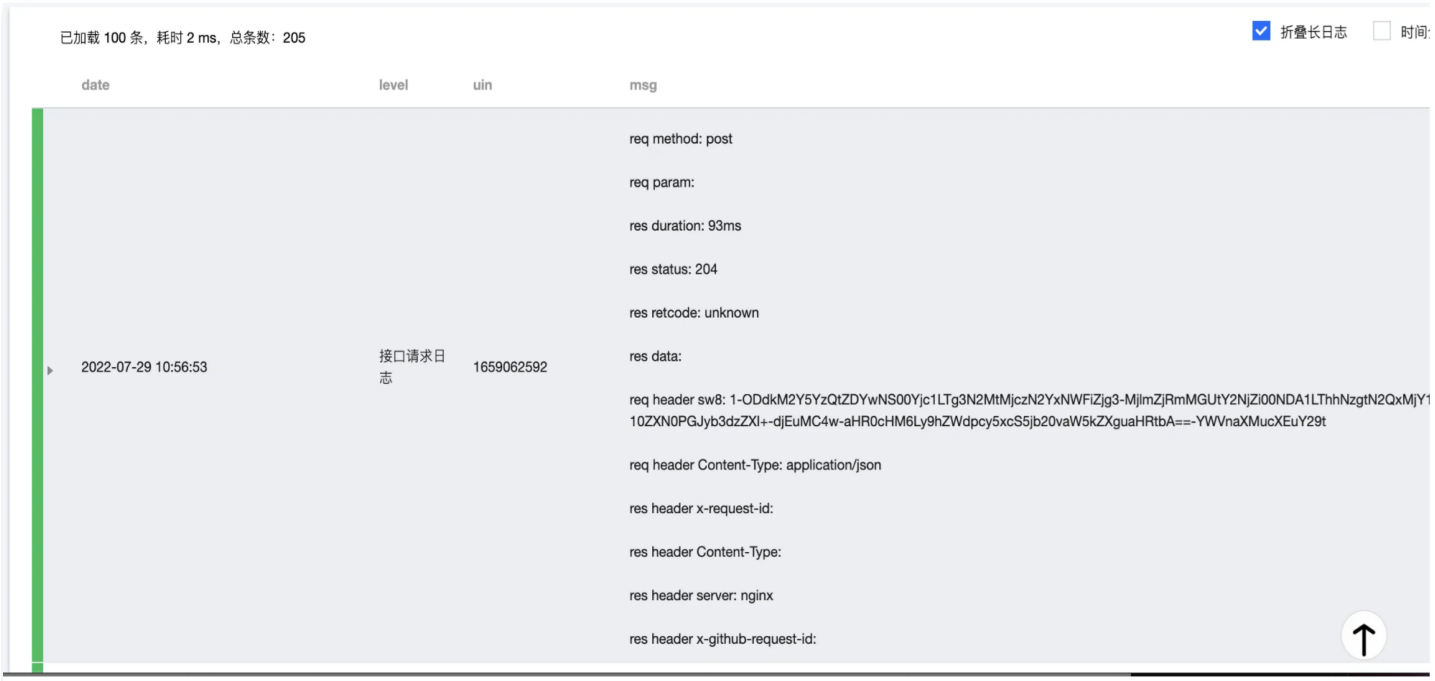


获取 Request Headers

获取 request headers 的使用方式获取 response headers 的方式大致相同，只需要在初始化的时候传入需要标记的 header 即可获取。

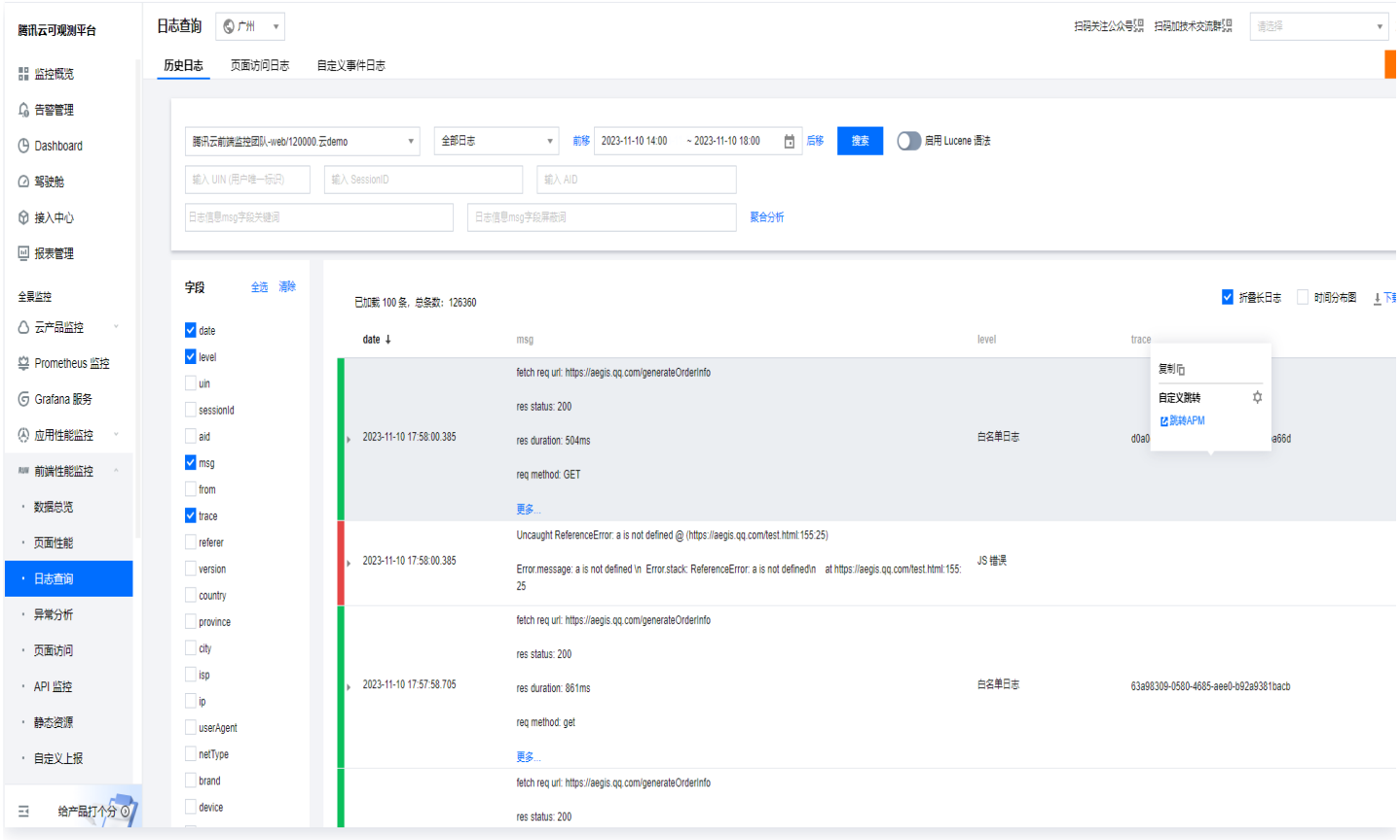
```
new Aegis({
        id: 'xxx',
        reportApiSpeed: true,
        api: {
            apiDetail: true, // 上报接口请求参数和返回值
            reportRequest: true, // 全量接口上报
            reqHeaders: ['sw8'], // 记录 request header
        },
    });
```

获取 request header 不会有浏览器的安全限制，添加较为方便。
添加后若接口请求中有 request header，即可在日志中看到相关数据。



自动获取 trace header

目前 Aegis SDK 已经实现自动解析 **opentelemetry**、**skywalking**、**sentry** 等 trace 协议的 header，并且自动进行上报，如果用户同时接入具有 trace 能力的 SDK，可以实现 traceid 自动识别功能。



跨域问题

如果接口添加了自定义 header，会导致接口请求跨域，需要对接口进行处理才能上报。例如：下列例子中需在项目的请求添加 `sw8`，在 nginx 中的配置添加 Access-Control-Allow-Headers 。

```
location /generateOrderInfo {
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Headers 'sw8';
        add_header Access-Control-Expose-Headers 'x-request-id, server';
        proxy_pass http://9.139.xx.xx:9301/generateOrderInfo;
    }
```

常见问题

什么情况下会用到记录请求 request header？

例如您通过随机 key 给请求添加标记，作为 traceid 的情况；或者您接入除了 aegis 以外的第三方可观测 SDK。具体使用 skywalking 打通前后端的方式可以参见 前后端链路打通 。
结合记录请求 request header，并通过 skywalking 打通前端和后端 API 调用链路后，即可在 RUM 上看相关前端请求信息、前端请求错误信息和前端性能数据等。

如何优化小程序用户体验

最近更新时间：2024-05-21 14:45:31

实践背景

对于有一定用户量的小程序，存在一些服务异常是不可避免的。如何能快速发现并有效分析定位问题?
小程序一般会存在的问题：

- **加载时间慢（白屏）：**
  小程序如果入口需要加载比较多的信息，间接导致首屏接口请求多，加载时间慢。整个小程序的首次冷启动时间可能超过 5s，而加载时长又直接影响到小程序的到达率。根据《High performance iOS Apps》中用研结论，25% 的用户在应用启动时间超过 3s 时会放弃使用。
- **卡顿/闪退：**
  一般有历史查询类场景会遇到长列表页面查询卡顿，滚动不流畅，随着历史消息加载，小程序出现闪退等问题。

这些问题带来了极大降低用户体验，从而降低用户留存。下列将会介绍小程序从接入到问题优化的实践步骤，作为实践案例供您参考。

实践步骤

接入应用

步骤1：创建业务系统

1. 登录 前端性能监控控制台 。
2. 在左侧菜单栏单击**应用管理 > 业务系统**。
3. 在业务系统管理页单击**创建业务系统**，在弹框中填写业务名称并勾选相关协议即可。

步骤2：创建应用

1. 在左侧菜单栏中单击**数据总览**。
2. 在数据总览页单击**应用接入**，填写并选择应用相关信息。

步骤3：安装和初始化 RUM-SDK

1. 安装：执行下列命令，在 npm 仓库安装 aegis-mp-sdk；

```
$ npm install --save aegis-mp-sdk
```

2. 初始化：参考下列步骤新建一个 Aegis 实例，传入相应的配置，初始化 SDK。

```
import Aegis from 'aegis-mp-sdk';
```

```
const aegis = new Aegis({
    id: 'vq6meu47xp638dLL95', // RUM 申请的上报 key
    uin: 'xxx', // 用户唯一 ID（可选）
    reportApiSpeed: true, // 接口测速
    reportAssetSpeed: true, // 静态资源测速
    spa: true, // spa 应用页面跳转的时候开启 pv 计算
    hostUrl: 'https://rumt-zh.com'
});
```

步骤4：验证是否接入成功

进入数据总览，若有数据则证明接入成功。

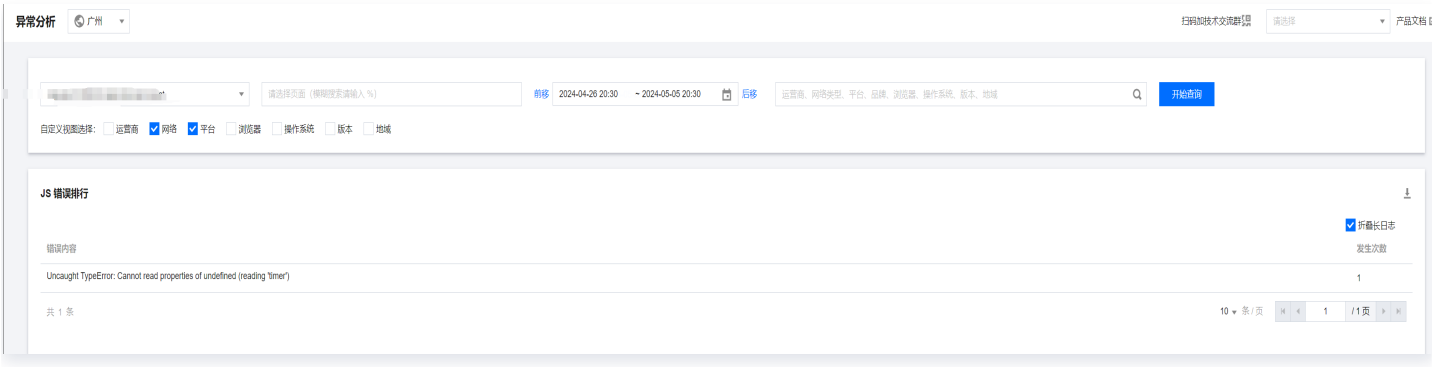
小程序优化

白屏问题处理

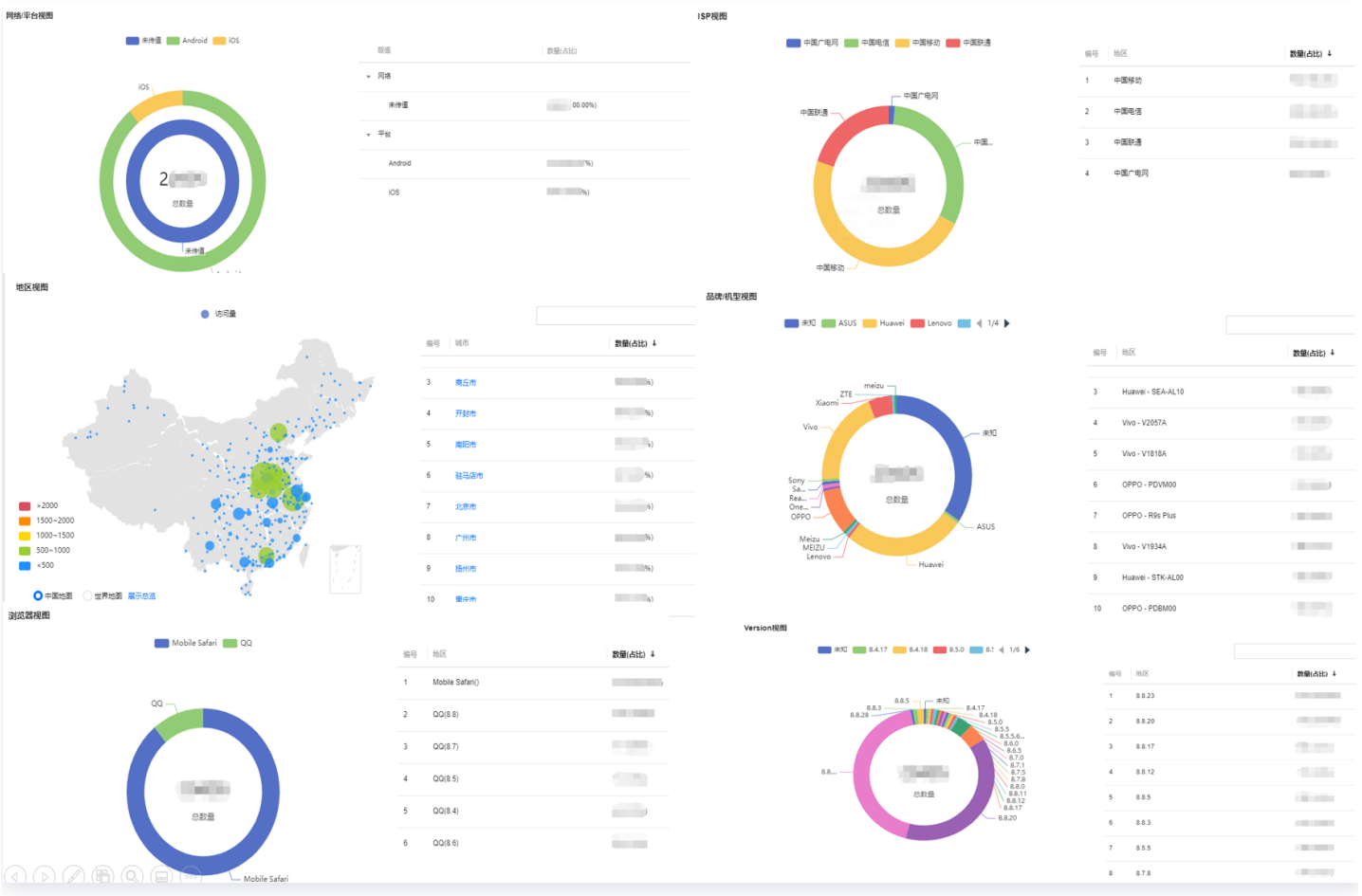
使用 RUM 日志查询 定位异常点。RUM 默认将错误异常全量上报，经过日志上报查询，可分钟级快速定位分析异常，并根据具体报告解决异常。

JS/ Ajax 等问题处理

当服务主要依赖明细日志时，很难及时对服务异常情况有总览或主动发现效果。RUM 将异常错误全量上报，并将错误信息按类型汇总统计，您可以使用 异常分析 分析 JS、Ajax 等错误问题并针对性地进行优化。



针对异常还可做不同维度的占比分析，更加精准地知道异常分布情况（网络类型、地域、机型等）。



配置指标监控告警

您可以使用 告警管理 > 策略管理，针对关键指标设置告警。在指标异常时及时通知您优化性能。


配置告警规则
监控类型
云产品监控
应用性能监控
HOT
前端性能监控
HOT
云拨测
HOT
聚焦Web、小程序等大前端领域，关注用户页面性能和质量。了解更多
策略类型
错误日志
页面性能
静态资源
API监控
自定义事件
自定义测速
上报数据量
页面访问-PV
业务日志
所属标签
标签键
标签值
添加
键值粘贴板
筛选条件(与)
地域
=
广州
业务系统
=
应用
=
日志类型
=
JS 错误
告警对象维度
业务系统
应用
日志类型
将对您筛选条件为 地域=广州，业务系统=rum-0JsJZjiOz36qmb(csjietest)，应用=csjiechendemotest，日志类型=JS 错误 的日志类型指标做告警检测

如何根据性能数据优化应用

最近更新时间：2024-05-08 15:36:02

实践背景

对于前端来说，最重要的是体验，而在前端体验中，最为核心的就是性能。本文将指导您如何看懂 RUM 可视化图表，并通过图表性能数据进行应用优化。

实践前提

已 接入应用 。

实践步骤

优化举例

下列某测试应用作为本次优化案例。前端架构错综复杂，不同应用的分析数据和优化方式截然不同，此处仅以个别应用为例，给您提供应用优化思路。

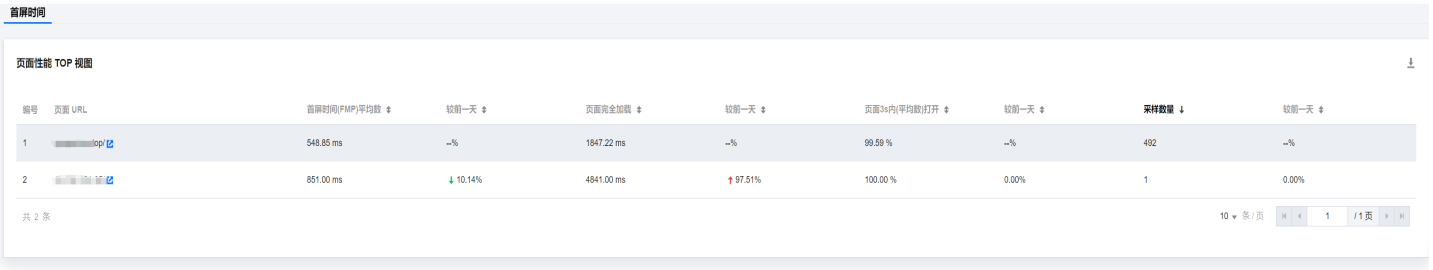
如上图，首屏时间达到了 4.8s，LCP（最大内容绘制）的时间超过了 4s， 指标建议优化等级也仅仅在“POOR（较差）”等级 ，CLS（累积布局移位）为0.64，数据也不容乐观。那我们该如何进一步分析?

> **说明：**

- LCP（Largest Contentful Paint – 最大内容绘制）：LCP 度量从用户请求网址到在视窗中渲染最大可见内容元素所需的时间。最大的元素通常是图片或视频，也可能是大型块级文本元素。
- CLS（Cumulative Layout Shift – 累积布局移位）：CLS 会衡量在网页的整个生命周期内发生的所有意外布局偏移的得分总和。得分是零到任意正数，其中 0 表示无偏移，且数字越大，网页的布局偏移越大。
- FID（First Input Delay – 首次输入延迟）：从用户首次与您的网页互动（点击链接、点按按钮，等等）到浏览器响应此次互动之间的用时。此衡量方案的对象是被用户首次点击的任何互动式元素。

分析 Top 访问页面

进入 前端性能监控 > 页面性能 页面，在**页面性能 TOP 视图**按 “首屏时间” 倒排序，排查是否把多个应用的页面都使用了同一个业务系统进行上报，导致一些比较差的页面对整体数据产生了影响。如有，建议不同应用使用不同的业务系统上报数据，方便定点发现问题。



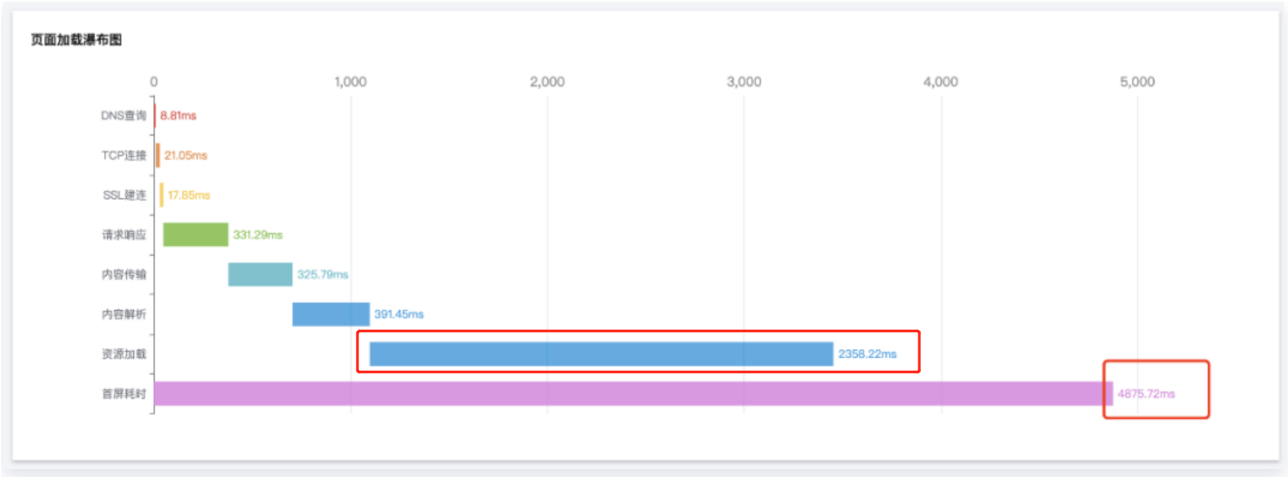
如上图我们发现开发者把多个应用的页面都在同一个业务系统上报了，导致整体性能数据较差。我们再进行下一步排查。

按网络和区域分析性能数据

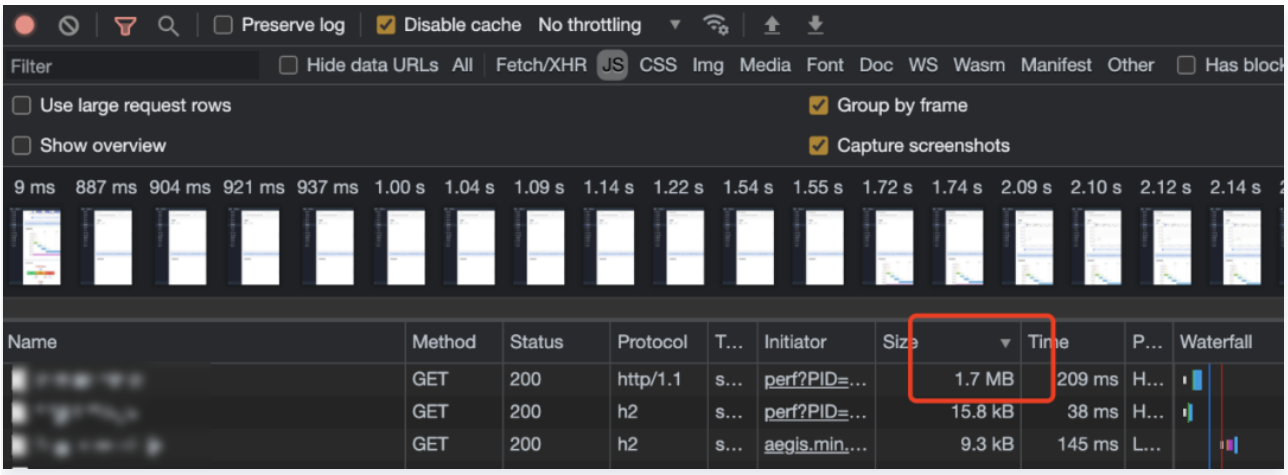
排除了页面的干扰，我们再分析一下网络和区域干扰。从图中可以看出来，网络状况和地区差异对页面首屏时间变化并不大。

再分析页面加载瀑布图

从图中可以看到该应用主要的瓶颈其实在 “资源加载” 的耗时。



通过“F12控制台功能”对用户页面的资源加载情况进行分析，某 JS 文件达到了1.7M导致加载缓慢。



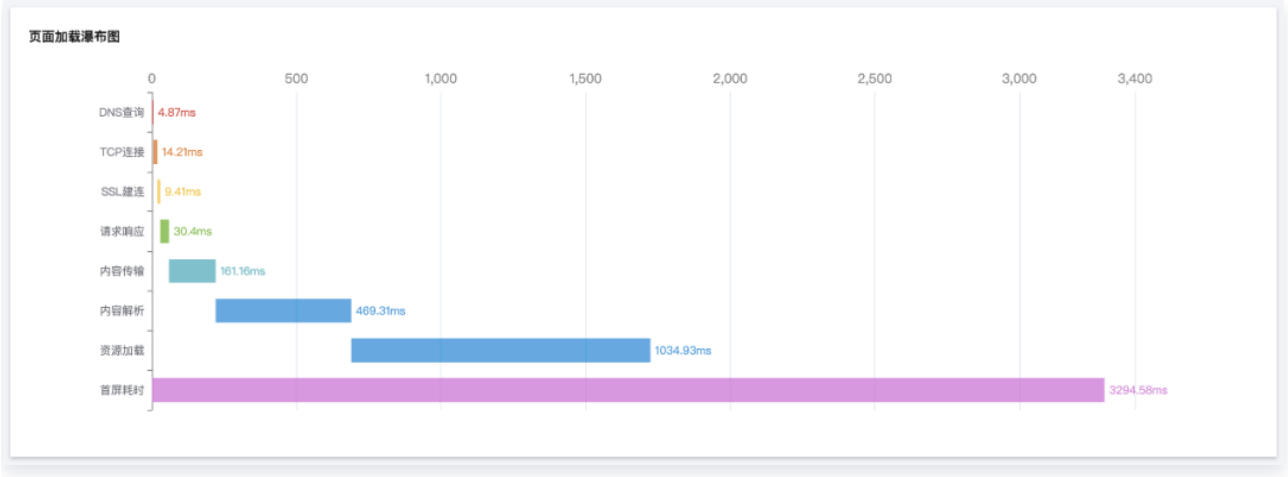
再进一步分析，该测试应用使用的是 React 框架，在没有服务端渲染的情况下，页面是会在加载主 JS 后才渲染的。而用户大部分 JS 文件都打包成一个 bundle ，导致产生了一个超大的 JS 文件，这个 JS 文件就成为了用户页面渲染的瓶颈。除此之外还发现了该 JS 文件没有支持 HTTP2 协议。那我们该如何优化?

资源加载优化

根据上述数据显示，我们建议用户做以下优化：

1. 拆包，通过把公共外部依赖打包成为 vendor，并且对组件做异步加载。
2. 去掉一些非必需的包，例如用户引入了全量的 lodash，让其改成 lodash-es，方便 webpack 做 treeshaking；去掉仅为了把某个时间做格式化而引入的 moment；去掉 jquery，大多数 jquery 仅仅为了查询某个元素。
3. 建议使用 webpack-bundle-analyzer 对打包后的代码进行分析，查看哪些包不需要引用，或者可以单独打包。
4. 网络协议方面全面引入 HTTP2，合并了一些小的静态资源，把一些小的 svg 改成了 base64。

通过上述步骤优化后首屏耗时从 4.8s 优化到 3.2s，最重要的是 “资源加载” 耗时直接下降50%。

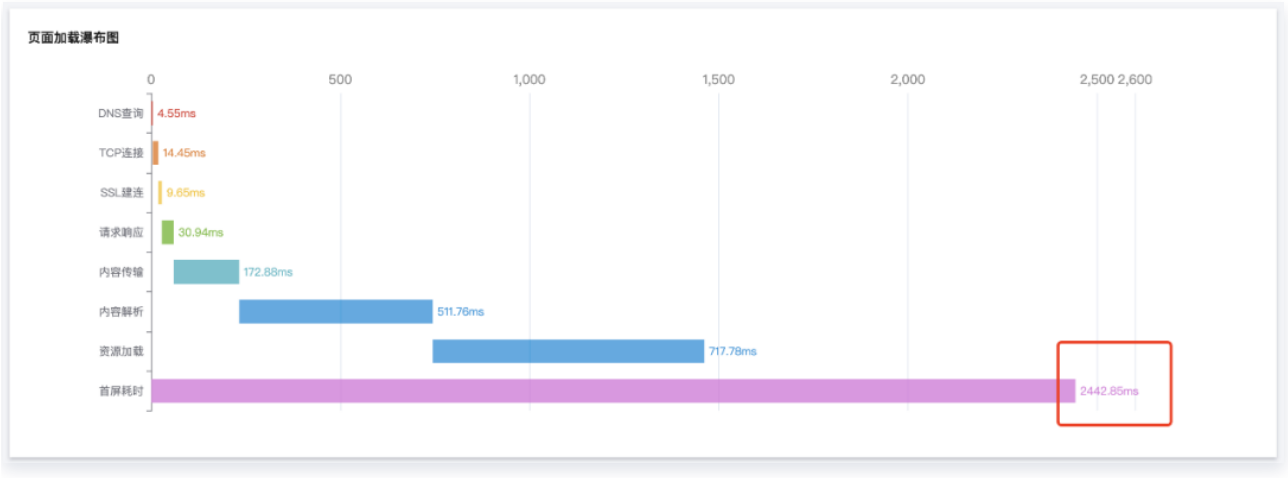


初次优化后，又迎来了一个问题，用户的 “资源加载” 时间已经大幅度降低了，但是为什么 “首屏耗时” 没有相应的同比降低（下降50%）呢?

我们通过对页面分析发现，该页面在加载完成后，会执行非常多的 JS 代码逻辑，包括一些数据上报，用户行为收集，还有加载侧边栏，弹出广告等。这个原因直接导致下列两个问题：

1. 页面主进程阻塞严重，Aegis SDK 的一些逻辑在执行的时候受到了影响，导致实际执行时间要晚于设定的时间，所以上报的“首屏耗时”其实要比实际晚的。
2. 用户的页面会在首屏完成后，继续加载很多 DOM 元素，也就是有很多 DOM 元素的变化，导致了 Aegis SDK 计算出来的首屏时间也要晚于真实的“首屏时间”。

根据上述两个问题，我们对定时器和异步进行了改造，又大幅度提升了页面的“首屏时间”。



此时“首屏耗时”已经是优化之初的 1/2 了，但是 CLS 的得分一直是 “POOR” 的状态。需要我们对 CLS 再进行优化。

CLS 指标优化

CLS 指的是页面布局偏移量，再次简单分析，我们发现该应用有一个长列表是页面主要渲染内容，由于数据不多，一般在 4 – 10 条数据，所以开发者没有对列表做分页。

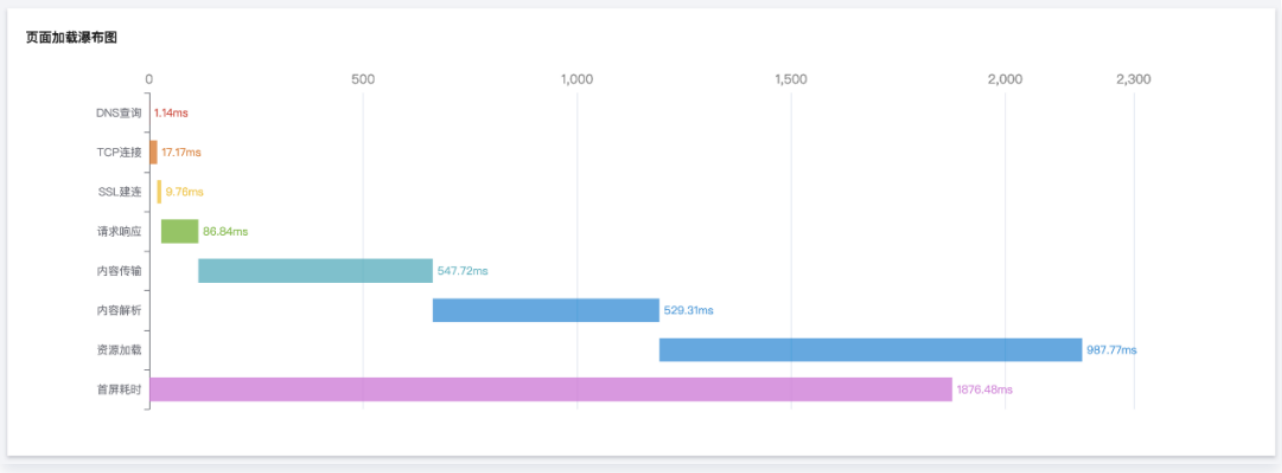
没有分页带来了列表无法在渲染之初就确定长度，导致获取数据后渲染列表的时候页面发生较大的偏移，同时也带来了超多的 DOM 变化。

这个是导致 CLS 数据较大的核心原因，同时也增加了“首屏耗时”的压力 ，除此之外，前面提到的一些异步数据，如广告挂件等也带来了DOM 变化。

于是我们做了如下优化：

1. 在一开始就确定列表高度（加入分页），通过骨架屏优化加载效果，同时减少 DOM 变化。
2. 广告挂件使用绝对布局，使其脱离文档流，减少DOM变化。
3. 一些其他元素，如图片等，确定长度和宽度属性，这些值允许浏览器在将图像渲染到位之前保留视觉空间。
4. 一些元素的变化，通过 CSS 实现，而不是使用 JS 改变元素属性实现。

再次优化后用户页面首屏和 CLS 数据变化惊人，首屏耗时下降了61.5%。下列为优化后的效果：



RUM 自动上传 sourcemap 文件

最近更新时间：2025-06-05 17:47:52

Sourcemap 文件对于前端 js 排障至关重要。Sourcemap 是一个信息文件，里面储存着代码转换前后的对应位置信息，可以解决在打包过程中，代码经过压缩、去空格以及 babel 编译转化后，由于代码之间差异性过大，造成无法 debug 的问题。Sourcemap 文件通常是在 CI/CD 过程中自动生成，如果用户有需求通过 API 调用或者想通过自定义流水线插件使流程自动化，可以参考以下的上传流程。

步骤1：获取调用云 API 的密钥

针对云上产品，前端页面使用的接口和 API 开放平台的接口是一致的。前端页面鉴权方式通常是使用微信/QQ/云梯账号等，而 API 开放平台的鉴权方式需要通过调用 API 对账号以及账号密钥进行鉴权。如果您想了解更多关于云 API 的内容，可以通过 前端性能 API 3.0 版本 进行查看。

步骤2：按照 demo 进行改造

为便于您的使用，您可以参照以下代码进行　写，以此改造自己的插件：

```
// Depends on tencentcloud-sdk-nodejs version 4.0.3 or higher
// 需要引入相关 npm 包
const tencentcloud = require('tencentcloud-sdk-nodejs');
const COS = require('cos-nodejs-sdk-v5');
const fs = require('fs');
var crypto = require('crypto');
const RumClient = tencentcloud.rum.v20210622.Client;
const clientConfig = {
    credential: {
        secretId: '子账号密钥 id',
        secretKey: '子账号密钥 key',
    },
    region: "",
    profile: {
        httpProfile: {
        endpoint: "rum.tencentcloudapi.com", // rum.tencentcloudapi.com
是外网域名，如果上面的密钥对没有开外网访问，使用外网域名将报无权限错误
        },
    },
};
const client = new RumClient(clientConfig);
const params = {};
const projectID = 0; // rum 的项目 id（数字 id 不是上报 key）
```

```
const version = '1.0.0'; // 这里自己填入需要的版本号(比如线上使用的 js 是1.0.0
版本)
const fileName = 'test.js.map'; // sourcemap 文件名
// 读取文件内容
const sourceMapFileContent = fs.readFileSync('./test.js.map');
var fileHash =
crypto.createHash('md5').update(sourceMapFileContent.toString()).digest(
'hex');
const cos = new COS({
    getAuthorization: (options, callback) => {
        client.DescribeReleaseFileSign(params).then(
            (data) => {
                callback({
                    TmpSecretId: data.SecretID,
                    TmpSecretKey: data.SecretKey,
                    SecurityToken: data.SessionToken,
                    ExpiredTime: data.ExpiredTime,
                });
            },
            (err) => {
                console.error("error", err);
            }
        ).catch(err => {
            console.log(err);
        });
    },
});
const timestamp = +new Date();
const fileKey = `${projectID}-${version}-${timestamp}-${fileName}`;
cos.putObject({
    Bucket: 'rumprod-1258344699', /* 必须 */ // 固定值
    Region: 'ap-guangzhou',     /* 存储桶所在地域，必须字段 */ // 固定值
    Key: fileKey,              /* 必须 */
    Body: sourceMapFileContent.toString(), // 上传文件对象
}).then(res => {
    console.log(res);
    client.CreateReleaseFile({
        ProjectID: projectID,
        Files: [{
            Version: version,
            FileKey: fileKey,
            FileName: fileName,
            FileHash: fileHash,
```

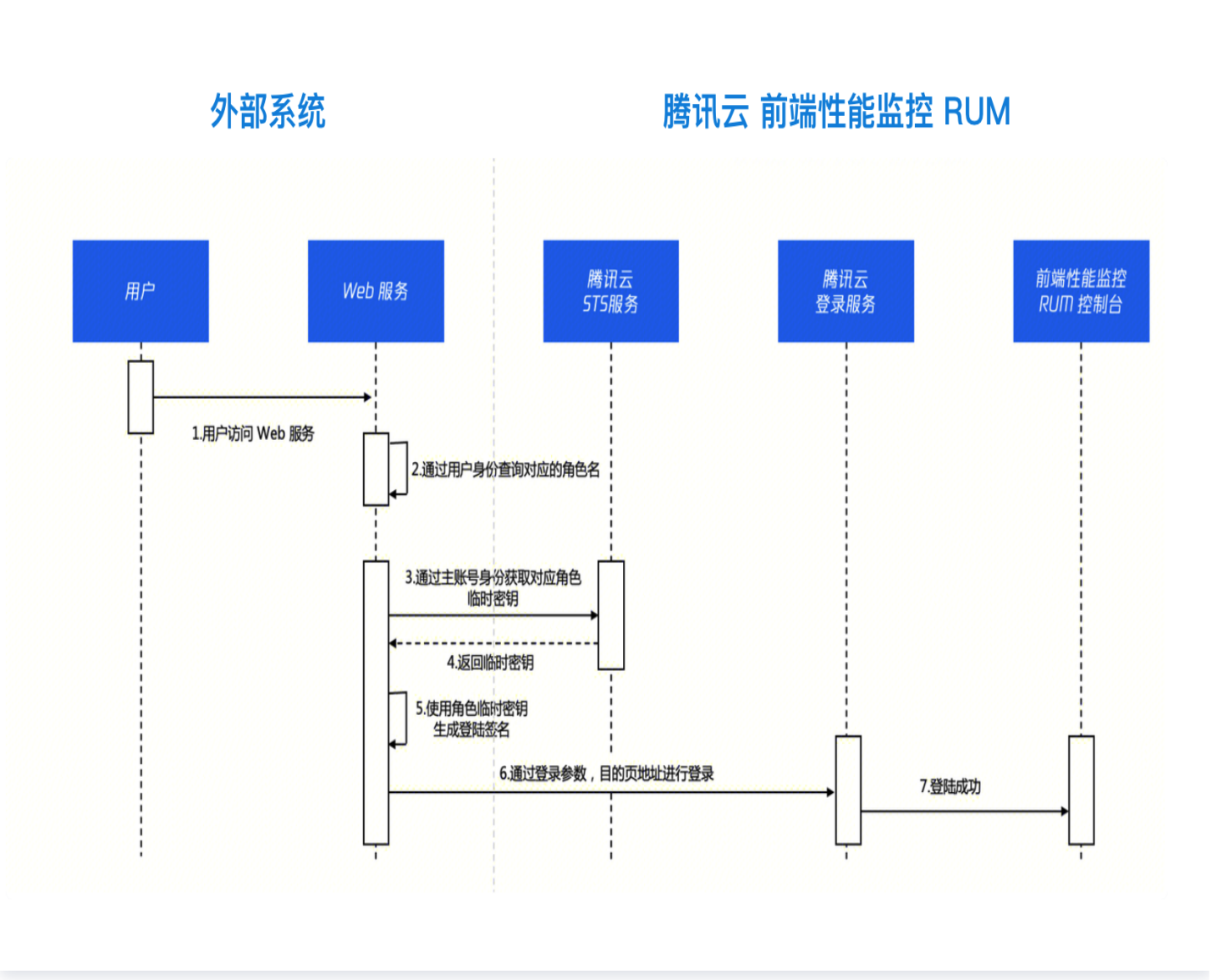
```
            }]
        }).then(
            (data) => {
                console.log('CreateReleaseFile res:', data);
                client.DescribeReleaseFiles({
                    ProjectID: projectID
                }).then(
                    (data) => {
                        // 这里就能查看到自己上传的 sourcemap 文件列表了
                        console.log('DescribeReleaseFiles res: ', data);
                    },
                    (err) => {
                        console.error("DescribeReleaseFiles error", err);
                    }
                );
            },
            (err) => {
                console.error("CreateReleaseFile error", err);
            }
        );
    }).catch(err => {
        console.log('putObject err:', err);
    });
```

将 RUM 页面嵌入自建系统

最近更新时间：2024-08-23 14:43:21

RUM 满足不需要登录腾讯云控制台即可查询分析数据的诉求。通过内嵌前端性能监控控制台页面，可以给用户带来以下方便：

- 在外部系统服务中（例如公司内部运维或运营系统）快速集成 rum 数据的查询分析能力。
- 无需管理众多腾讯云子账号，方便将 RUM 数据分享给他人进行查看。



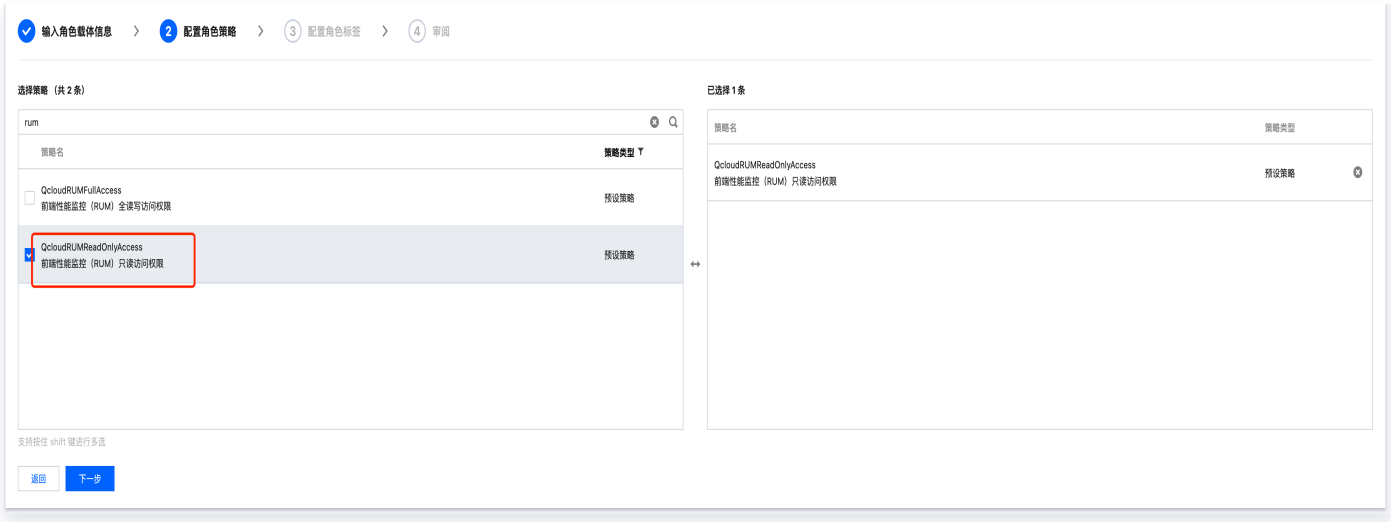
创建用户身份

企业用户（运维或开发人员）根据业务需求申请对应的权限，用户身份对应腾讯云账号角色，可以通过 控制台 或 创建角色API 创建对应的角色：

通过控制台创建 CAM 角色

1. 登录 访问管理 CAM 控制台 。
2. 单击左侧菜单栏中的**角色**，进入角色页面。
3. 选择**新建角色 > 腾讯云账户**，开始新建自定义角色。
4. 选择**当前主账号**并勾选**允许当前角色服务控制台**，单击**下一步**。

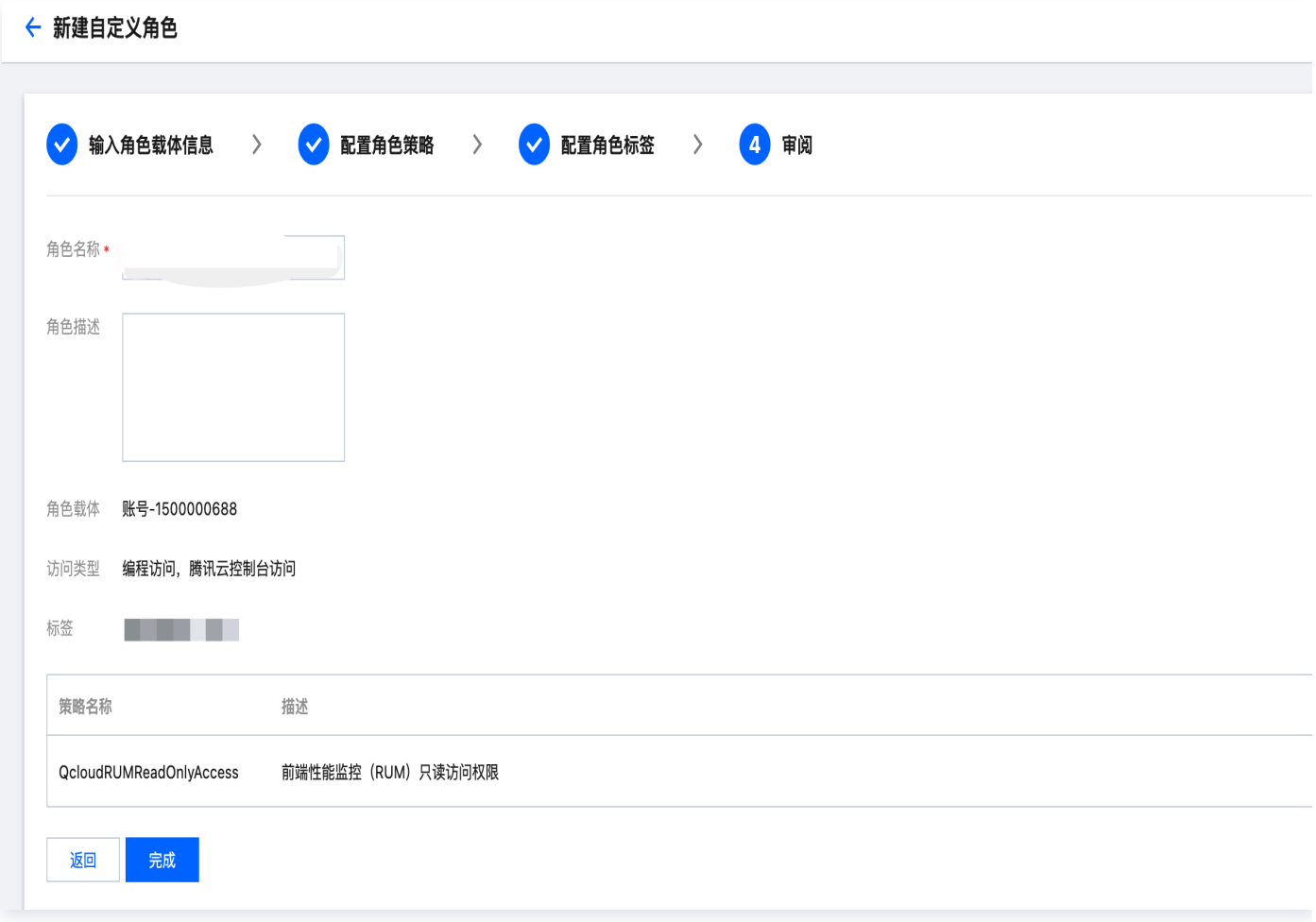
5. 为角色设置访问策略，例如只读策略权限 QcloudRUMReadOnlyAccess，单击**下一步**。



6. 为角色配置不同维度标签，使用标签对用户进行分类管理。


输入角色载体信息
配置角色策略
3
配置角色标签
4
审阅
标签是腾讯云提供的用于标识云上资源的标记，是一个键值对（Key-Value）。
您可以为子用户设置不同维度的标签，如职位、部门、籍贯等，使用标签对用户进行分类管理。
vwy92ox9
A
添加
键值粘贴板
返回
下一步

7. 输入角色名，完成创建。


新建自定义角色
输入角色载体信息
配置角色策略
配置角色标签
4
审阅
角色名称 *
角色描述
角色载体
账号-1500000688
访问类型
编程访问，腾讯云控制台访问
标签
策略名称
描述
QcloudRUMReadOnlyAccess
前端性能监控（RUM）只读访问权限
返回
完成

通过 API 创建 CAM 角色

1. 获取当前用户的访问密钥，可参见 主账号访问密钥管理 。
2. 创建角色，详情请参见 创建角色 API ，其中 ConsoleLogin 需要填入1，允许角色登录控制台。
3. 绑定 QcloudRUMReadOnlyAccess 的权限策略到角色，详情参见 绑定权限策略到角色 。

获取用户身份访问密钥

根据角色名访问腾讯云 STS 服务，调用 AssumeRole 接口，申请角色 CompanyOpsRole 的临时密钥。

> **注意：**
> 设置中可能存在以下风险，请参考安全意见进行操作：
> - 临时密钥有效期请勿设置过长，建议设置在 5 分钟以内，可通过 AssumeRole 参数 DurationSeconds 指定。
> - 包含参数的完整登录地址（https://cloud.tencent.com/login/roleAccessCallback?algorithm...）请勿暴露在公网。
> - 用户侧用于生成登录地址的系统需设置鉴权，例如内网身份校验，请勿设置成公开权限访问。

生成 APM 访问链接

生成签名串

1. 拼接参数。对要求签名的参数按照字母表或数字表递增顺序的排序，先考虑第一个字母，在相同的情况下考虑第二个字母，以此类推。

参数名称	必选	类型	描述
action	是	String	操作动作，固定为 roleLogin
timestamp	是	Int	当前时间戳
nonce	是	Int	随机整数，取值10000-100000000
secretId	是	String	STS 返回的临时 AK

拼凑参数示例： `action=roleLogin&nonce=67439&secretId=AKI***PLE&timestamp=1484793352`

2. 拼接签名串。按 **请求方法** + **请求主机** +**请求路径** + ? + **请求字符串** 的规则拼接签名串。

参数	必选	描述
请求主机和路径	是	固定为 cloud.tencent.com/login/roleAccessCallback
请求方法	是	支持 GET 或 POST

拼接签名串示例：

```
GETcloud.tencent.com/login/roleAccessCallback?
action=roleLogin&nonce=67439&secretId=AKI***PLE&timestamp=1484793352
```

3. 生成签名串。使用 HMAC-SHA1 算法对字符串签名，目前支持 HMAC-SHA1 和 HMAC-SHA256，以 Python 语言为例：

```
sts_secret_id = "AKI***PLE"
sts_secret_key = "IF***Wn3"
sig_str = 'GETcloud.tencent.com/login/roleAccessCallback?
action=roleLogin&nonce=' + str(nonce) + '&secretId=' + sts_secret_id +
'&timestamp=' + str(timestamp)
sign_str = base64.b64encode(hmac.new(bytes(sts_secret_key,
encoding='utf-8'), bytes(sig_str, encoding='utf-8'),
hashlib.sha1).digest())
```

拼凑最终访问链接

1. 获取 RUM 控制台页面：https://console.cloud.tencent.com/monitor/rum?hideWidget=true&hideTopNav=true

   URL 参数说明：

参数名称	必选	类型	描述
hideWidget	否	Boolean	是否隐藏智能客服图标：默认不隐藏，true 表示隐藏
hideTopNav	否	Boolean	是否隐藏腾讯云控制台顶部导航栏：默认不隐藏，true 表示隐藏
hideLeftNav	否	Boolean	是否隐藏腾讯云控制台左侧导航栏：默认不隐藏，true 表示隐藏

2. 拼接完整登录信息以及目的页地址进行登录，参数值需要 urlencode 编码。

```
https://cloud.tencent.com/login/roleAccessCallback
?algorithm=<签名时加密算法，目前只支持 sha1 和 sha256 ，不填默认 sha1
&secretId=<签名时 secretId>
&token=<临时密钥 token>
&nonce=<签名时 nonce>
&timestamp=<签名时 timestamp>
&signature=<签名串>
&s_url=<登录后目的 URL>
```

3. 使用生成的最终链接，访问腾讯云 APM 控制台页面。

```
https://cloud.tencent.com/login/roleAccessCallback?
algorithm=sha1&secretId=AK***Lb&token=yXJYBcXqi***qPos_52PCpauvYykeiSp
VZ7w5g2qOvV1Azs&nonce=67439&timestamp=1484793352&signature=AJ***3D&s_u
rl=https%3A//console.cloud.tencent.com/monitor/rum%3FhideWidget%3Dtrue
%26hideTopNav%3Dtrue
```

完整示例代码

Python 语言完整实例代码：

```
import json
import random
import time
import base64
import hmac
import hashlib
from urllib.parse import quote
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.common.exception.tencent_cloud_sdk_exception import
TencentCloudSDKException
from tencentcloud.cam.v20190116 import cam_client, models as cam_models
from tencentcloud.sts.v20180813 import sts_client, models as sts_models

try:
    role = "CompanyOpsRole"
    uin = 100020507208

    # 步骤一：创建角色
    secret_id = "AK***NO"
    secret_key = "lG***Zx"
    cred = credential.Credential(secret_id, secret_key)
    httpProfile = HttpProfile()
    httpProfile.endpoint = "cam.tencentcloudapi.com"

    clientProfile = ClientProfile()
    clientProfile.httpProfile = httpProfile
    client = cam_client.CamClient(cred, "", clientProfile)

    # 步骤二：创建 CAM 角色
```

```
    req = cam_models.CreateRoleRequest()
    params = {
        "RoleName": role,
        "PolicyDocument": "{\"version\":\"2.0\",\"statement\":
[{\"action\":\"name/sts:AssumeRole\",\"effect\":\"allow\",\"principal\":
{\"qcs\":[\"qcs::cam::uin/" + str(
            uin) + ":root\"]}}]}",
        "ConsoleLogin": 1
    }
    req.from_json_string(json.dumps(params))
    client.CreateRole(req)

    # 步骤三：绑定权限策略到角色
    req = cam_models.AttachRolePolicyRequest()
    params = {
        "AttachRoleName": role,
        "PolicyName": "QcloudRUMReadOnlyAccess"
    }
    req.from_json_string(json.dumps(params))
    client.AttachRolePolicy(req)

    # 步骤四：请求 AssumeRole
    client = sts_client.StsClient(cred, "ap-shanghai")
    req = sts_models.AssumeRoleRequest()
    req.RoleArn = "qcs::cam::uin/" + str(uin) + ":roleName/" + role
    req.RoleSessionName = "test"
    resp = client.AssumeRole(req)

    # 步骤五：生成签名串
    sts_secret_id = resp.Credentials.TmpSecretId
    sts_secret_key = resp.Credentials.TmpSecretKey
    token = resp.Credentials.Token
    nonce = random.randint(10000, 100000000)
    timestamp = int(time.time())

    sig_str = 'GETcloud.tencent.com/login/roleAccessCallback?
action=roleLogin&nonce=' + str(
        nonce) + '&secretId=' + sts_secret_id + '&timestamp=' +
str(timestamp)
    sign_str = base64.b64encode(
        hmac.new(bytes(sts_secret_key, encoding='utf-8'), bytes(sig_str,
encoding='utf-8'), hashlib.sha1).digest())
```

```
    # 步骤六：拼凑最终访问链接
    result = 'https://cloud.tencent.com/login/roleAccessCallback?
algorithm=sha1&secretId=' + \
             quote(sts_secret_id) + '&token=' + quote(token) + '&nonce='
+ str(nonce) + '&timestamp=' + str(
        timestamp) + '&signature=' + quote(sign_str) + \
             '&s_url=' +
quote('https://console.cloud.tencent.com/monitor/rum?
hideWidget=true&hideTopNav=true')
    print(result)
except TencentCloudSDKException as err:
    print(err)
```