腾讯云可观测平台

前端性能监控

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

前端性能监控
前端性能监控简介
前端性能监控概述

最近更新时间：2024-11-05 18:18:51

前端性能监控（Real User Monitoring，RUM）是一站式前端监控解决方案，专注于 Web 和小程序等大前端领域，主要关注用户页面性能（页面测速、接口测速、CDN 测速等）和质量（JS 错误、Ajax 错误等），并且联动腾讯云应用性能监控实现前后端一体化监控。用户只需要安装 SDK 到自己的应用中，通过简单配置化，即可实现对用户页面质量的全方位守护，真正做到了低成本使用和无侵入监控。

产品优势

多平台

前端性能监控目前支持 Web、小程序（微信、QQ）、Hippy、Weex、React Native、 Flutter 和 Cocos 等平台的数据上报，支持无打点首屏测速、资源测速、API 测速、白名单机制和离线日志等功能特性。

无侵入

前端性能监控 SDK 无需在业务代码中打点或者做任何其他操作，可以做到与业务代码充分解耦。 SDK 将会自动监控前端错误，在错误发生时上报错误的具体情况，帮助您快速定位问题。当您开启资源测速时，SDK 将会自动监听页面资源加载情况（耗费时长、成功率等），并在不影响前端性能的前提下收集前端的性能数据，帮助您快速定位性能短板，提升用户体验。

低成本

前端性能监控支持 Web 和小程序的前端真实体验监控服务，学习成本较低，只要您有过基础的前端知识，就可以放心的使用。

功能介绍

最近更新时间：2024-11-06 16:03:01

日志上报

前端性能监控支持开发者上报任意数据到前端监控平台，包括一般性日志、自定义事件、自定义测速等。用来满足前端开发者日志收集的诉求。

错误收集

前端性能监控针对浏览器执行 JS 代码报错、接口信息报错、资源加载异常、Promise 异常等报错都有 SDK 进行主动收集，开发者只需接入 SDK 就可以实现所有错误收集的功能。

性能数据

前端性能监控包括众多页面性能数据，其中包括首屏耗时、建立 TCP 连接耗时、TTFB 耗时和 SSL 耗时等。除此之外，最新的 webvitals（谷歌针对网页加载速度和体验所提出的一套指标）标准已支持页面性能三大核心指标等数据，也可进行数据采集和上报。

页面访问

前端性能监控支持查看页面访问 PV（页面访问量）和 UV（页面独立访问量），通过运营商、访问来源和访问地域等维度，实时了解并分析用户访问情况。

资源测速

前端性能监控支持资源测速，包括图片加载耗时和 CDN 资源耗时等。开发者可以查看某个页面下具体使用了哪些资源，每个资源的耗时情况等信息。同时前端性能监控支持对地域信息统计，ISP 信息统计等功能。

接口测速

前端性能监控可通过接口测速统计页面上所有调用接口的信息，包括接口耗时和状态码统计等信息。同时前端性能监控支持对地域信息统计和 ISP 信息统计等功能。

智能告警

前端性能监控支持错误告警、页面耗时告警和性能数据告警等功能。

应用场景

最近更新时间：2024-11-05 14:33:11

提升用户体验

前端性能监控通过页面报错、页面加载、接口成功率等关键性能指标变化趋势，实时了解用户使用情况。基于关键性能指标变化趋势和用户健康度评分，改善用户浏览页面体验。

故障定位

前端性能监控支持 JS 代码报错、接口信息报错、资源加载异常、Promise 异常等错误主动收集和慢页面详情监控（首屏加载耗时、请求响应等），通过页面加载瀑布图、Core Web Vitals（谷歌提出的三个网络性能指标的视图）、地区视图等多维度关联分析异常指标，定位异常原因。

性能优化

前端性能监控支持地区、运营商、网络、设备/机器类型等多维度视图。您可以结合用户特征和多维度视图进行对比分析，实现有针对性地优化 Web 应用性能。

接入指南
Web 应用场景
安装和初始化

最近更新时间：2024-11-05 16:48:31

支持 CDN 和 NPM 两种方式安装并初始化 SDK 。

方式1：以 CDN 方式安装并初始化 SDK

1. 在 HTML 页面的 `<head></head>` 标签中引入下列代码。

```
<script src="https://tam.cdn-go.cn/aegis-sdk/latest/aegis.min.js"></script>
```

该 cdn 使用 “h3-Q050” 协议，默认 cache-control 为 max-age=666，如果需要修改 cache-control，可以添加参数 max_age，例如：

```
<script src="https://tam.cdn-go.cn/aegis-sdk/latest/aegis.min.js?max_age=3600"></script>
```

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

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

方式2：以 NPM 方式安装并初始化 SDK

1. 执行下列命令，在 npm 仓库安装 aegis-web-sdk。

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

2. 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-web-sdk';

const aegis = new Aegis({
    id: 'pGUVFTCZyewhxxxxxx', // 应用ID，即上报ID
    uin: 'xxx', // 用户唯一 ID（可选）
    reportApiSpeed: true, // 接口测速
    reportAssetSpeed: true, // 静态资源测速
    hostUrl: 'https://rumt-zh.com', // 上报域名，中国大陆 rumt-zh.com
    spa: true // spa 应用页面跳转的时候开启 pv 计算
})
```

说明：

- 为了不遗漏数据，须尽早进行初始化。当您完成安装并初始化 SDK 之后，可以开始使用前端性能监控提供的以下功能：
  - 错误监控：JS 执行错误、Promise 错误、Ajax 请求异常、资源加载失败、返回码异常、PV 上报、白名单检测等。
  - 测速功能：页面性能测速、接口测速、静态资源测速。
  - 数据统计和分析：可在 前端性能监控 > 数据总览 上进行各个维度的数据分析。
- aegis-sdk 默认使用 `https://aegis.qq.com` 作为上报域名，您可以通过 `hostUrl` 参数来控制上报域名。
  - 国内可以选择使用 `https://rumt-zh.com` 作为上报域名。
  - 新加坡地区可以选择使用 `https://rumt-sg.com` 作为上报域名。
  - 硅谷地区可以选择使用 `https://rumt-us.com` 作为上报域名。

日志上报

最近更新时间：2024-11-06 18:43:11

日志查询功能需您上报日志后，才能正常使用。本文将为您介绍如何上报日志到前端性能监控。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

配置 SDK

引入下列参数，配置 SDK ，完成日志上报。

```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.logType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

Aegis.logType 枚举值如下：

```
export enum LogType {
  API_RESPONSE = '1', // 白名单中的用户，页面上的所有 API 返回都将会被上报
  INFO = '2', // aegis.info、aegis.infoAll 上报的日志
  ERROR = '4', // js 错误
  PROMISE_ERROR = '8', // promise 错误
  AJAX_ERROR = '16', // ajax 错误
  SCRIPT_ERROR = '32', // script 加载失败
  IMAGE_ERROR = '64', // 图片加载失败
  CSS_ERROR = '128', // css 加载失败
  CONSOLE_ERROR = '256', // console.error 监控（目前暂未支持）
  MEDIA_ERROR = '512', // 音视频加载失败
  RET_ERROR = '1024', // retcode 返回码异常
  REPORT = '2048', // aegis.report 上报日志默认level
  PV = '4096', // 页面 PV
  EVENT = '8192', // 自定义事件
  PAGE_NOT_FOUND_ERROR = '16384', // 小程序 页面不存在
  WEBSOCKET_ERROR = '32768', // websocket错误
  BRIDGE_ERROR = '65536', // js bridge 错误
}
```

aid

最近更新时间：2024-11-06 16:48:31

aid 是 Aegis SDK 为每个用户设备分配的唯一标识，存储在浏览器的 localStorage ，用于区分用户计算 UV 等。只有用户清理浏览器缓存后， aid 才会更新。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

自定义 aid 上报规则

您可以根据下列算法自定义 aid 上报规则：

```
async getAid(callback: Function) {
// 某些情况下操作 localStorage 会报错.
  try {
    let aid = await localStorage.getItem('AEGIS_ID');
    if (!aid) {
    aid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
      const r = (Math.random() * 16) | 0;
      const v = c === 'x' ? r : (r & 0x3) | 0x8;
      return v.toString(16);
    });
    localStorage.setItem('AEGIS_ID', aid);
    }
    callback?.(aid || '');
  } catch (e) {
    callback?.('');
  }
}
```

说明

若您需要使用自己构造的 aid 作为上报规则，需要后端对 aid 的校验规则，规则如下：`/^[@=.0-9a-zA-Z_-]{4,36}$/` 。

实例方法

最近更新时间：2025-06-11 10:35:21

前端性能监控为您提供多种实例方法用于上报数据，您可以通过实例方法修改实例配置、自定义上报事件、自定义上报测试资源等。
本文为您介绍目前 RUM 提供的 Aegis 实例方法。

参数	用途
setConfig	传入配置对象，包括用户 ID 和 UIN 等信息。
info	主要上报字段，用于上报白名单日志。 下列两种情况日志才会报到后台： 1. 在白名单的用户打开了前端页面。 2. 对应的页面发生了错误。
infoAll	主要上报字段，用于上报白名单日志。该上报与 info 唯一的区别： info 指定用户上报。
error	主要上报字段，用于上报错误信息。
report	用来上报任意类型的日志信息。
reportEvent	上报自定义事件。
reportTime	上报自定义测速资源。
time	上报自定义测速资源，与 timeEnd 共同使用。适用于两个时间点之间时长的计算并上报。
timeEnd	上报自定义测速资源，与 time 共同使用。适用于两个时间点之间时长的计算并上报。
destroy	销毁 aegis 实例。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

实例方法

setConfig

该方法用于修改实例配置，使用场景如下。
1. 可获取到用户 UIN，可同时传入配置用户 ID 和 UIN 两个实例对象，进行实例化。

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  uin: '777'
})
```

2. 通常情况下，我们并不能一开始就获取到用户的 `uin` 。若在获取 UIN 的这段时间不进行实例化，这期间发生的错误前端性能监控将无法监听。针对这种情况，我们可以先传入 ID 进行实例化，引用 setConfig 传入 UIN，示例如下：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx'
})

// 拿到 uin 之后...
aegis.setConfig({
```

```
  uin: '6666'
})
```

注意：该方法对 config 进行覆盖时采用浅合并逻辑，并不会做深度合并。

```
// 初始化时传入如下配置
const aegis = new Aegis({
    spa: true,
    reportApiSpeed: true,
    pagePerformance: {
      firstScreenInfo:  true
    }
})


// 调用 setConfig 修改配置
aegis.setConfig({
    spa: false,
    reportAssetSpeed: true,
    pagePerformance: {
      urlHandler() {
        return 'xxx'
      }
    }
})

// 结果如下：
{
    spa: false,
    reportApiSpeed: true,
    reportAssetSpeed: true,
    pagePerformance: {
      urlHandler() {
        return 'xxx'
      }
    }
}
```

info、infoAll、error 和 report

这几个方法是前端性能监控提供的主要上报手段。

```
// info 可以上报任意字符串、数字、数组、对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传 ext 参数和 trace 参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

```
// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取 JS 异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level:  '4',  // 日志等级，具体取值可参考日志等级小节。
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

Aegis.logType 枚举值如下：

```
export enum LogType {
  API_RESPONSE = '1', // 白名单中的用户，页面上的所有 API 返回都将会被上报
  INFO = '2', // aegis.info、aegis.infoAll 上报的日志
  ERROR = '4', // js 错误
  PROMISE_ERROR = '8', // promise 错误
  AJAX_ERROR = '16', // ajax 错误
  SCRIPT_ERROR = '32', // script 加载失败
  IMAGE_ERROR = '64', // 图片加载失败
  CSS_ERROR = '128', // css 加载失败
  CONSOLE_ERROR = '256', // console.error 监控（目前暂未支持）
  MEDIA_ERROR = '512', // 音视频加载失败
  RET_ERROR = '1024', // retcode 返回码异常
  REPORT = '2048', // aegis.report 上报日志默认level
  PV = '4096', // 页面 PV
  EVENT = '8192', // 自定义事件
  PAGE_NOT_FOUND_ERROR = '16384', // 小程序页面不存在
  WEBSOCKET_ERROR = '32768', // websocket错误
  BRIDGE_ERROR = '65536', // js bridge 错误
}
```

reportEvent

该方法可用来上报自定义事件，系统将会自动统计上报事件的各项指标，例如：PV、平台分布等。
reportEvent 可以支持字符串和对象两种类型上报参数。

字符串类型

```
aegis.reportEvent('XXX请求成功');
```

对象类型

ext1、ext2和 ext3默认使用 new Aegis 的时候传入的参数，自定义事件上报的时候，可以覆盖默认值。

```
aegis.reportEvent({
  name: 'XXX请求成功', // 必填
  ext1: '额外参数1',
  ext2: '额外参数2',
  ext3: '额外参数3',
})
```

> ⚠ **注意：**
> 额外参数的三个 key 是固定的，目前只支持 ext1、ext2和 ext3。

reportTime

该方法可用来上报自定义测速，例如：

```
// 假如‘onload’的时间是1s
aegis.reportTime('onload', 1000);
```

或者如果需要使用额外参数，可以传入对象类型参数，ext1、ext2、ext3会覆盖默认值。

```
aegis.reportTime({
  name: 'onload', // 自定义测速 name
  duration: 1000, // 自定义测速耗时(0 - 60000)
  ext1: 'test1',
  ext2: 'test2',
  ext3: 'test3',
});
```

> ⓘ **说明：**
> `onload` 可以修改为其他的命名。

time 和 timeEnd

该方法同样可用来上报自定义测速，适用于两个时间点之间时长的计算并上报，例如：

```
aegis.time('complexOperation');
/**
 * .
 * .
```

```
 * 做了很久的复杂操作之后。
 * .
 * .
 */
aegis.timeEnd('complexOperation'); /** 此时日志已经报上去了**/
```

说明：

- `complexOperation` 可以修改为其他的命名。
- 自定义测速是用户上报任意值，服务端对其进行统计和计算。由于服务端不能做脏数据处理，建议用户在上报端进行统计值限制，防止脏数据对整体产生影响。
- 目前 Aegis 只支持0 – 60000的数值计算，如果大于该值，建议进行合理改造。

destroy

销毁实例进程，销毁后数据不再上报，并且 Aegis 不再收集用户数据。

```
aegis.destroy();
```

日志等级

键（level）	值（name）
1	'白名单日志'
2	'一般日志'
4	'错误日志'
8	'Promise 错误'
16	'Ajax 请求异常'
32	'JS 加载异常'
64	'图片加载异常'
128	'css 加载异常'
256	'console.error'
512	'音视频资源异常'
1024	'retcode 异常'
2048	'aegis report'
4096	'PV'
8192	'自定义事件'
16384	'小程序 页面不存在'
32768	"websocket 错误'
65536	'js bridge 错误'

白名单

最近更新时间：2024-07-11 15:30:11

白名单功能是适用于开发者针对特定的用户上报更多的信息。为了过滤部分用户，并减少用户的接口请求次数，因此 RUM 提供了白名单功能，并设定了白名单的逻辑。
白名单逻辑如下：

1. 白名单用户会上报全部的 API 请求信息，包括接口请求和请求结果。
2. 白名单用户可以使用 info 接口上报数据。
3. info 和 infoAll：在开发者实际体验过程中，白名单用户可以添加更多的日志，并且使用 info 进行上报。infoAll 会对所有用户无差别进行上报，因此可能导致日志量上报巨大。
4. 通过接口 whitelist 来判断当前用户是否是白名单用户，白名单用户的返回结果会绑定在 aegis 实例上 (aegis.isWhiteList) 用来给开发者使用。
5. 为了减少开发者使用负担，白名单用户是针对业务系统生效。您可以在 应用管理 > 白名单管理 内创建白名单，则业务系统下全部应用都对该白名单用户生效。

钩子函数

最近更新时间：2024-07-05 11:48:11

您可以使用钩子函数对某些资源的测速上报进行自定义配置，系统将会为您计算、统计函数执行时间等。您可以在 自定义上报 页面选择**自定义测速**，进行多维度分析函数执行耗时。

onBeforeRequest

这个钩子函数会在所有请求发送之前被调用，所有请求内容都会传入它的参数中，必须返回待发送的内容。

```
function changeURLArg(url,arg,arg_val) {
  var pattern=arg+' = ([^&]*)';
  var replaceText = arg+'='+arg_val;
  if (url.match(pattern)) {
    var tmp = '/('+ arg+'=)([^&]*)/gi';
    tmp = url.replace(eval(tmp),replaceText);
    return tmp;
  }
  return url;
}
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  onBeforeRequest(log) {
    if (log.type === 'performance') {
      // Page speed test. Here, you can modify the content of `log`; for example, you can modify
the `platform` for page speed test
      log.url = changeURLArg(log.url, 'platform', type)
    }
    return log
  }
});


// SEND_TYPE {
//   LOG = 'log',  // Log
//   SPEED = 'speed', // API and static resource speed test
//   PERFORMANCE = 'performance', // Page speed test
//   OFFLINE = 'offline', // Offline log upload
//   WHITE_LIST = 'whiteList', // Allowlist
//   VITALS = 'vitals', // Web Vitals
//   PV = 'pv', // Custom PV
//   EVENT = 'event', // Custom event
//   CUSTOM = 'custom', // Custom speed test
//   SDK_ERROR = 'sdkError', // SDK error
// }
```

beforeReport

1. 该钩子将在日志报告之前执行（对应上报接口为 `/collect?` ），例如：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    // Listen on the reported error below
    console.log(log); // {level: "4", msg: "An error occurred."}
```

```
    return log;
  }
});

throw new Error('An error occurred.');
```

注意：

上述日志将具有以下字段：

- level：日志级别。例如，'4'表示错误日志。
- msg：日志内容。

日志级别如下：

- { level: '1', name: 'API请求日志（允许的日志）' }
- { level: '2', name: '通用日志（aegis.info或aegis.infoAll）' }
- { level: '4', name: 'JavaScript执行错误' }
- { level: '8', name: 'Promise错误' }
- { level: '16', name: 'Ajax请求异常' }
- { level: '32', name: 'JavaScript加载异常' }
- { level: '64', name: '图像加载异常' }
- { level: '128', name: 'CSS加载异常' }
- { level: '256', name: 'console.error（保留）' }
- { level: '512', name: '音/视频资源异常' }
- { level: '1024', name: '返回码异常' }
- { level: '2048', name: 'aegis报告' }
- { level: 4096, name: 'PV' }
- { level: 8192, name: '自定义事件' }
- { level: 16384, name: '小程序页面不存在' }
- { level: 32768, name: 'websocket 错误' }
- { level: 65536, name: 'js bridge 错误' }

2. 如果该钩子返回 false，则不会报告日志。该功能可用于过滤掉不需要上报的错误，例如：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    if (log.level === '4' && log.msg && log.msg.indexOf('An error that doesn't need to be
reported') !== -1) {
      return false
    }
    return log;
  }
});
throw new Error('An error that doesn't need to be reported'); // This error will not be reported
```

在上面的示例代码中，如果错误内容包含关键字 `An error that doesn't need to be reported`，则不会报告给 RUM 后端。

onReport

该 hook 在日志上报成功后执行，用法与 beforeReport 类似。它们唯一的区别是，该 hook 接收到的参数都是已经上报的日志，而 beforeReport 接收到的参数是待上报的日志。

beforeReportSpeed

1. 该钩子将在速度测试数据报告之前（ `/speed?` ）被执行，例如：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    console.log(msg); // {url: "https://localhost:3001/example.e31bb0bc.js", method: "get",
duration: 7.4, status: 200, type: "static"}
    return msg
  }
});
```

> ⚠ **注意：**
>
> 上面的 msg 将具有以下字段：
>
> - url：资源的请求地址。
> - type：资源类型。
>   - fetch：Aegis 将把该资源报告为 API 请求；
>   - static：Aegis 将把该资源报告为静态资源。
> - duration：资源请求持续时间。
> - method：在请求资源时使用的 HTTP 方法。
> - status：服务器返回的状态码。
> - payload：向您提供的完整资源请求信息（此字段不会被报告给 Aegis 后端，您可以操纵它）
>
> 完整的数据结构如下：
>
> - payload.type: 资源请求类型，用于区分原始请求类型。有效值：'fetch'和'xhr'。
> - payload.sourceURL: 完整的 URL 请求连接。
> - payload.status: 请求状态码。
> - payload.headers: 所有请求头，其值都是字符串。
> - payload.data: 完整的请求资源，您可以自定义它。如果请求类型为 fetch，则为响应对象；如果请求类型为 xhr，则为 XMLHttpRequest 对象。

在上面的示例代码中，每当 Aegis 收集到资源的加载详细信息后，将会使用这些详细信息（即 msg上面返回的）作为参数来调用 `beforeReportSpeed` 钩子。

2. 如果您配置了该钩子，Aegis 最终的上报内容将以钩子的执行结果为准。例如：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    msg.type = 'static';
    return msg;
  }
});
```

上面的代码中，将所有的 msg.type 设置为 static，这意味着所有的资源都将被当成静态资源进行上报，API 请求也将被报至静态资源中。

3. 您可以使用此钩子来纠正 Aegis 类型并判断错误的请求。

如果您有一个 API `https://example.com/api`，其响应标头 Content-Type 为 text/html。正常情况下，RUM 会将此 API 报告为静态资源。但在您的业务中，必须报告为 API 请求，您可以使用以下钩子配置 Aegis 进行修正：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    if (msg.url === 'https://example.com/api') {
      msg.type = 'fetch';
    }
  }
});
```

4. 您还可以屏蔽某些资源的测速上报，例如：

```
  const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    // Speed test logs for resources whose address contains `https://example.com/api` will not be
reported
    if (msg.url.indexOf('https://example.com/api') !== -1) {
      // If `false` is returned, the reporting of the speed test log will be blocked
      return false
    }
  }
});
```

beforeRequest

该钩子将会在所有的数据上报前执行，来帮助用户在数据上报前对其进行屏蔽和修改，当 beforeRequest 返回 false 就可以不上报该条日志，返回修改过后的 data 就可以实现对上报数据的修改，例如：

> **注意**
> SDK 版本应大于等于1.24.44。

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeRequest: function(data) {
    // 入参 data 的数据结构：{logs: {…}, logType: "log"}
    if (data.logType === 'log' && data.logs.msg.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 log，且内容包含 otheve.beacon.qq.com 的请求
      return false;
    }
    // 入参 data 数据结构：{logs: {}, logType: "speed"}
    if (data.logType === 'speed' && data.logs.url.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 speed，并且接口 url 包含 otheve.beacon.qq.com 的请求
      return false;
    }
    if (data.logType === 'performance') {
      // 修改：将性能数据的首屏渲染时间改为2s
      data.logs.firstScreenTiming = 2000;
```

```
    }
    return data;
  }
});
```

其中，data 将会有以下几个字段：

- logType：日志类型，有以下值：
  - custom：自定义测速
  - event：自定义事件
  - log：日志
  - performance：页面测速
  - pv：页面 PV
  - speed：接口和静态资源测速
  - vitals：web vitals
- logs：上报的日志内容：
  - 当 logType 为 'custom' 时，logs 数据类型为 `{name: "白屏时间", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}`
  - 当 logType 为 'event' 时，logs 数据类型为 `{name: "ios", ext1: "", ext2: "", ext3: ""}`
  - 当 logType 为 'performance' 时，logs 数据类型为 `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 2660, ssl: 4, tcp: 4, ttfb: 5}`
  - 当 logType 为 'speed' 时，logs 数据类型为 `{connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "static", url: "https://xxxxxx", urlQuery: "max_age=1296000"}`
  - 当 logType 为 'vitals' 时，logs 数据类型为 `{delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "LCP", value: 1100}`
  - 当 logType 为 'log' 时，logs 数据类型为 `{msg: "日志详情", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}`

说明

其中 level 枚举值如下：

- `{ level: '1', name: '接口请求日志（白名单日志）' }`
- `{ level: '2', name: '一般日志(aegis.info 或者 aegis.infoAll)' }`
- `{ level: '4', name: 'JS 执行错误' }`
- `{ level: '8', name: 'Promise 错误' }`
- `{ level: '16', name: 'Ajax 请求异常' }`
- `{ level: '32', name: 'JS 加载异常' }`
- `{ level: '64', name: '图片加载异常' }`
- `{ level: '128', name: 'css 加载异常' }`
- `{ level: '256', name: 'console.error (未启用)' }`
- `{ level: '512', name: '音视频资源异常' }`
- `{ level: '1024', name: 'retcode 异常' }`
- `{ level: '2048', name: 'aegis report' }`
- `{ level: 4096, name: 'PV' }`
- `{ level: 8192, name: '自定义事件' }`
- `{ level: 16384, name: '小程序 页面不存在' }`
- `{ level: 32768, name: 'websocket错误' }`

- `{ level: 65536, name: 'js bridge 错误' }`

该钩子返回 false 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，可以用来过滤不希望上报的日志。

afterRequest

该钩子将会在测速数据上报后被执行，例如：

> **注意**
> SDK 版本应大于等于1.24.44。

```
const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx",
  afterRequest: function(data) {
    // {isErr: false, result: Array(1), logType: "log", logs: Array(4)}
    console.log(data);
  }
});
```

其中，data 将会有以下几个字段：

- isErr：请求上报接口是否错误
- result：上报接口的返回结果
- logs：上报的日志内容
- logType：日志类型，同 beforeRequest 中的 logType。

错误监控

最近更新时间：2024-11-22 14:19:22

前端性能监控的 Aegis 的实例会自动进行 JS 执行错误、Promise 执行错误、Ajax（Fetch）请求异常等监控。本文将为您介绍各错误监控逻辑及处理方式。

> ⚠ **注意：**
> Aegis 实例会对这些异常进行监控，当您只是引入了 SDK 而没有将其实例化时，Aegis 将不会上报数据。

JS 执行错误

Aegis 通过监听 `window` 对象上的 `onerror` 事件来获取应用中的报错，并且通过解析错误和分析堆栈，将错误信息自动上报到后台服务中。该上报等级为 error ，所以当自动上报的错误达到阈值时，Aegis 将会自动告警，帮助您尽早发现异常。由于上报等级为 error ，自动上报也将影响应用的评分。

- 如果页面上引入了跨域的 JS 脚本，需要给对应的 `script` 标签添加 `crossorigin` 属性，否则 Aegis 将无法获取详细的错误信息。
- 如果用户使用的是 VUE 框架，请引入下列代码，获取错误并且主动上报。

```
Vue.config.errorHandler = function(err, vm, info) {
  console.log(`Error: ${err.toString()}\nStack: ${err.stack}\nInfo: ${info}`);
  aegis.error(`Error: ${err.toString()}\nStack: ${err.stack}\nInfo: ${info}`);
};
```

Promise 执行错误

通过监听 `unhandledrejection` 事件，捕获到未被 `catch` 的 Promise 错误，为了页面的稳定性，建议您 `catch` 住所有的 Promise 错误。

Ajax（Fetch）请求异常

Aegis 将会改写 `XMLHttpRequest` 对象，监听每次接口请求，Aegis 认为以下情况是异常情况：

- `http status` 大于等于400。
- 请求超时，abort，跨域，cancel。
- 请求结束时 `http status` 仍然是0，通常发生于请求失败。

> ⚠ **注意：**
> Aegis SDK 在错误发生的时候，不会主动收集接口请求参数和返回信息，如果需要对进口信息进行上报，可以使用 API 参数里面的 apiDetail 进行开启。

```
new Aegis({
  api: {
    apiDetail: true,
  },
});
```

retcode 异常

Aegis 改写 `XMLHttpRequest` 对象之后，将获得 API 返回的内容，并尝试在内容中获取到本次请求的 `retcode` 。

- retcode 的值默认会从用户返回 response body 的第一层（如果第一层取不到，再取第二层）的 code、ret、retcode、errcode 中获取。
- Aegis 默认 retcode 的值为0是正常的，非0都是异常的。当 `retcode` 发生异常的时候，会上报一个 retcode 异常的日志。
- 用户可以通过 api.retCodeHandler 对这个值和是否异常进行修正。

> **说明：**
> 如何获取 `retcode` 以及哪些 `retcode` 是正常的，详情请参见 配置文档 。

资源加载失败

页面元素发出的请求如果失败，将会被 `window.onerror` 事件捕获到（捕获阶段），Aegis 正是通过这个特性监听的资源加载失败。Aegis 监听了以下资源：

- `<link>` 标签请求的 css、font 等。
- `<script>` 标签请求的脚本。
- `<audio>` 、`<video>` 标签请求的多媒体资源。

websocket 异常

Aegis 默认不会对全局 websocket 对象进行重写，默认不会监控 websocket 相关错误，如果需要打开相关功能可以在创建 Aegis 对象时设置 websocketHack: true。

```
new Aegis({
  id: '',
  websocketHack: true
});
```

性能监控

最近更新时间：2024-11-05 11:01:41

您可以通过本文了解页面测速、接口测试、资源测速的统计方式和传入配置等信息。

页面测速

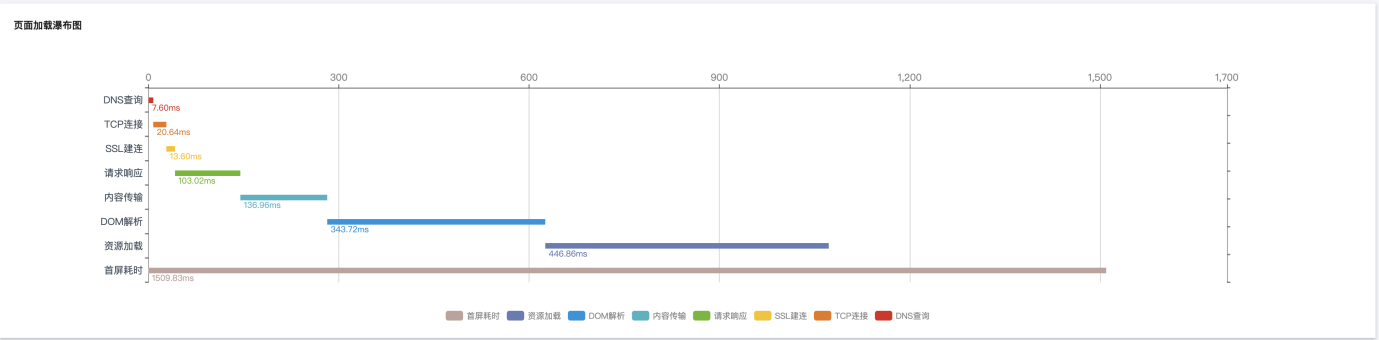
> **说明：**
> RUM 默认为您开启页面测速功能。

当您成功安装和初始化 SDK 后，Aegis 实例默认会上报以下指标：

1. **DNS 查询**：domainLookupEnd - domainLookupStart；
2. **TCP 连接**：connectEnd - connectStart；
3. **SSL 建连**：requestStart - secureConnectionStart；
4. **请求响应**：responseStart - requestStart；
5. **内容传输**：responseEnd - responseStart；
6. **DOM解析**：domInteractive - domLoading；
7. **资源加载**：loadEventStart - domInteractive；
8. **首屏耗时**：监听页面打开3s内的**首屏** DOM 变化，并认为 DOM 变化数量最多的那一刻为首屏框架渲染完成时间（SDK 初始化后 setTimeout 3s 收集首屏元素，由于 JS 是在单线程环境下执行，收集时间点可能大于 3s）；
9. **页面完全加载时间**：TCP、DNS、SSL、TTFB、DOM 解析和资源加载的时间之和。

> **说明：**
> 1-7 项页面打开性能指标计算说明可从 PerformanceTiming 获取。首屏耗时对应的 DOM 元素，可以通过打印 aegis.firstScreenInfo 查看。如果 DOM 元素不能代表首屏，可以添加属性 `<div aegis-first-screen-timing></div>`，把某个元素识别为首屏关键元素，SDK 认为只要用户首屏出现此元素就是首屏完成。也可以添加属性 `<div aegis-ignore-first-screen-timing></div>`，把该 DOM 列入黑名单。

根据以上数据，前端性能监控为用户绘制了页面加载瀑布图：



> **说明：**
> - 总体来说，页面首屏和页面完全加载时间是正相关的。大多数情况下，用户的首屏时间是小于页面完全加载的。Aegis SDK 根据用户页面 DOM 变化计算首屏时间，如果用户页面完全加载后，还继续发生 DOM 变化，就有可能发生首屏时间晚于页面完全加载的情况。
> - 在服务端场景，瀑布图会出现首屏时间大于 DOM 解析的情况，这是由于移动端设备兼容性问题，有些设备无法获取到 DNS 查询、TCP 连接、SSL 建连时间，这三个指标汇总后的平均值偏小，导致除了首屏时间外的其他指标都往左偏移。
> - 如果用户发现 RUM 性能数据里面首屏耗时大于页面完成加载的问题，并且上报的 performance 接口中 firstScreenTiming 为 0 还有可能是 SDK 初始化时间太晚导致的，如果 SDK 初始化时，页面首屏已完成，则无法获取到正常的 firstScreenTiming 的值。建议用户把 SDK 初始化放在 head 中进行，尽早初始化，然后再确认上报数据是否正常。

接口测速

> **说明：**
> 打开方式：初始化时传入配置 `reportApiSpeed: true`

Aegis 通过劫持 `XHR` 及 `fetch` 进行接口测速。

资源测速

> **说明：**
> 打开方式：初始化时传入配置 `reportAssetSpeed: true`

Aegis 通过浏览器提供的 `PerformanceResourceTiming` 进行资源测速。

环境控制

最近更新时间：2024-11-05 15:01:41

aegis 默认把数据所在环境当作 `production` 进行上报，如果需自行修改，可以使用 env 参数进行修改。

```
new Aegis({
  id: 'pGUVFTCZyewhxxxxxx',
  env: Aegis.environment.gray
})
```

Aegis.environment 枚举值如下：

```
export enum Environment {
  production = 'production', // 生产环境
  development = 'development', // 开发环境
  gray = 'gray', // 灰度环境
  pre = 'pre', // 预发布环境
  daily = 'daily', // 日发布环境
  local = 'local', // 本地环境
  test = 'test', // 测试环境
  others = 'others' // 其他环境
}
```

修改 env 参数后，aegis 上报的数据都会带上该参数，方便开发者区分不同环境的数据，但是只有 production 环境的数据会参与应用得分的计算。

配置文档

最近更新时间：2024-11-06 14:27:11

配置说明

配置文档各配置项说明如下：

配置	描述
id	必须，string，默认无。 前端性能监控分配的应用 ID。
uin	建议，string，默认取 cookie 中的 UIN 字段。 当前用户的唯一标识符，白名单上报时将根据该字段判定用户是否在白名单中，字段仅支持 `字母数字@=._-` ，正则表达式： `/^[@=.0-9a-zA-Z_-]{1,60}$/` 。
reportApiSpeed	可选，boolean 或者 object，默认 false。 是否开启接口测速。
reportAssetSpeed	可选，boolean，默认 false。 是否开启静态资源测速。
pagePerformance	可选，boolean 或者 object，默认 true。 是否开启页面测速。
webVitals	可选，boolean，默认 true。 是否开启 web vitals 测速。
onError	可选，boolean，默认 true。 当前实例是否需要进行错误监听，获取错误日志。
aid	可选，boolean，默认 true。 当前实例是否生成 aid。
random	可选，number，默认 1。 0 - 1 抽样率。 0 表示全部不上报。
spa	可选，boolean，默认 false。 当前页面是否为单页应用，若为 true，则将监听 hashchange 及 history api，在页面跳转时进行 PV 上报。
pageUrl	可选。默认是 location.href。 修改上报数据中页面地址，开发者可以主动对数据进行聚合和降低维度。
version	可选，string，默认 SDK 版本号。 当前上报版本，当页面使用了 pwa 或者存在离线包时，可用来判断当前的上报是来自哪一个版本的代码，仅支持 `字母数字.,:_-` ，长度在 60 位以内 `/^[0-9a-zA-Z.,:_-]{1,60}$/` 。
delay	可选，number，默认 1000ms。 上报节流时间，在该时间段内的上报将会合并到一个上报请求中。
repeat	可选，number，默认 5。 重复上报次数，对于同一个错误超过多少次不上报。同一个接口上报测速数据的次数。
env	可选 enum，默认 Aegis.environment.production。当前应用运行所处的环境，详情请参见说明文档 环境控制 。
websocketHack	可选，boolean，默认 false。 是否开启 websocket 监控。开启该选项后，将会对 websocket 连接断开状态进行监控，但是新建操作仍需要自定义监控。
api	可选，object，默认为 `{}` 。相关的配置： • apiDetail：可选，boolean，默认false。上报 api 信息的时候，是否上报 api 的请求参数和返回值。

	• retCodeHandler：`Function(data: String, url: String, xhr: Object):{isErr: boolean, code: string}`，返回码上报钩子函数。会传入接口返回数据，请求 url 和 xhr 对象；详情请参见示例 api.retCodeHandler。 • reqParamHandler：Function(data: any, url: String) 上报请求参数的处理函数，可以对接口的请求参数进行处理，方便用户过滤上报请求参数的信息。 • resBodyHandler：Function(data: any, url: String) 上报 response 返回 body 的处理函数，可以对接口返回值的 response body 进行处理，只上报关键信息。 • resourceTypeHandler：Function，请求资源类型修正钩子函数会传入接口 url，返回值为 `static` 或 `fetch`。 • reportRequest：boolean，默认：false。开启后，aegis.info 会变成全量上报，不需要白名单配置，并且会上报所有接口的信息（上报接口需开启 reportApiSpeed）。 • reqHeaders：Array 需要上报的 HTTP 请求 request headers 列表。 • resHeaders：Array 需要上报的 HTTP 请求 response headers 列表。如果开发者需要获取的字段非“simple response header”，需要给在返回头中给字段添加 Access-Control-Expose-Headers。 • usePerformanceTiming：boolean，默认：false。aegis sdk 默认使用打点的方式计算接口耗时，该方式在移动端存在一些误差，开启 usePerformanceTiming 后，aegis sdk 会从 performance 中重新获取接口耗时，让接口耗时统计更为准确。需要注意的是，开启该参数的前提是用户业务接口请求 url 是独立且唯一的，用户可以在接口请求 url 中添加时间戳等方式来保证 url 唯一性。如果 url 不唯一，可能导致同 url 接口耗时获取错误。 • injectTraceHeader：可选，string，且必须为枚举值 'traceparent'、'sw8'、'b3'、'sentry-trace' 中的一种，开启该参数后，Aegis 会在用户请求头中注入相关 trace header。详细用法参考 腾讯云可观测平台全链路接入。 • injectTraceUrls：可选，Array，数组中传入 string 或者 RegExp。标记哪些请求 url 需要注入 trace 请求头。 • injectTraceIgnoreUrls：可选，Array，数组中传入 string 或者 RegExp。标记哪些请求 url 不需要注入 trace 请求头。
speedSample	可选，boolean，默认 true。 测速日志是否抽样（限制每条 url 只上报一次测速日志）。
hostUrl	可选，默认是 `https://aegis.qq.com`。 影响全部上报数据的 host 地址，设置 hostUrl 后，下面几个 URL 地址都会被覆盖。
url	可选，string，默认 `https://aegis.qq.com/collect`。 日志上报地址。 设置为空字符串可以不进行日志上报。
pvUrl	可选，string，默认 `https://aegis.qq.com/collect/pv`。 pv 上报地址。 设置为空字符串可以不进行 pv 上报。
whiteListUrl	可选，string，默认 `https://aegis.qq.com/collect/whitelist`。 设置白名单确认接口。进入 前端性能监控控制台，选择左侧导航栏的**应用管理>白名单管理**进行设置。 设置为空字符串可以关闭白名单接口请求。
eventUrl	可选，string，默认 `https://aegis.qq.com/collect/events`。 设置自定义事件上报地址。 设置为空字符串可以不进行自定义事件上报。
speedUrl	可选，string，默认 `https://aegis.qq.com/speed`。 测速日志上报地址。 设置为空字符串可以不进行测速数据上报。
customTimeUrl	可选，string，默认 `https://aegis.qq.com/speed/custom`。 设置自定义测速上报地址。 设置为空字符串可以不进行自定义测速上报。
performanceUrl	可选，string，默认 `https://aegis.qq.com/speed/performance`。 设置页面性能日志上报地址。

	设置为空字符串可以不进行页面性能上报。
webVitalsUrl	可选，string，默认 `https://aegis.qq.com/speed/webvitals` 。 设置 webvitals 上报地址。 设置为空字符串可以不进行 web-vitals 上报。
dbConfig	可选，Object。
ext1	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext2	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext3	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。

示例

api.retCodeHandler

假如后台返回数据为：

```
{
   body: {
      code: 200,
      retCode: 0,
      data: {
          // xxx
      }
   }
}
```

业务需要：code 不为200，或者 retCode 不为0，此次请求就是错误的。此时只需进行以下配置：

```
new Aegis({
    // xxx
    reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
    reportAssetSpeed: true,
    api: {
        retCodeHandler(data, url, xhr) {
            // data 是string类型，如果需要对象需要手动parse下
            // url 为请求url
            // xhr 响应，可以通过xhr.response拿到完整的后台响应
            try {
                data = JSON.parse(data)
            } catch(e) {}
            return {
                // isErr 如果是 true 的话，会上报一条 retcode 异常的日志。
                isErr: data.body.code !== 200 || data.body.retCode !== 0,
                code:  data.body.code
            }
        }
    }
})
```

api.resourceTypeHandler

假如接口为：`http://example.com/test-api` 。返回的 `Content-Type` 为 `text/html` ，这将导致 Aegis 认为该接口返回的是静态资源，可以通过以下方法修正：

```
new Aegis({
    reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
    reportAssetSpeed: true,
    api: {
        resourceTypeHandler(url) {
            if (url?.indexOf('http://example.com/test-api') != -1) {
                return 'fetch';
            }
        }
    }
})
```

reportApiSpeed.urlHandler

假如您页面中有 restful 风格的接口，例如：
`www.example.com/user/1000`
`www.example.com/user/1001`
在上报测速时需要将这些接口聚合：

```
new Aegis({
    // xxx
    reportApiSpeed: {
        urlHandler(url, payload) {
            if ((/www\.example\.com\/user\/\d*/).test(url)) {
                return 'www.example.com/user/:id';
            }
            return url;
        }
    }
    // xxx
})
```

pagePerformance.urlHandler

假如您的页面 URL 是 restful 风格的，例如：
`www.example.com/user/1000`
`www.example.com/user/1001`
在上报页面测速时需要将这些页面地址聚合：

```
new Aegis({
    // xxx
    pagePerformance: {
        urlHandler() {
            if ((/www\.example\.com\/user\/\d*/).test(window.location.href)) {
                return 'www.example.com/user/:id';
            }
        }
    }
    // xxx
})
```

浏览器指纹

最近更新时间：2024-05-24 14:25:11

Aegis 指纹版 SDK 使用浏览器指纹算法作为用户唯一标识 aid，提升 uv 准确率；用户通过引用指纹版 Aegis SDK 即可使用该功能。

引入步骤

1. 引入指纹版 SDK。
   - cdn 引入：

```
<script src="https://tam.cdn-go.cn/aegis-sdk/latest/aegis.f.min.js"></script>
```

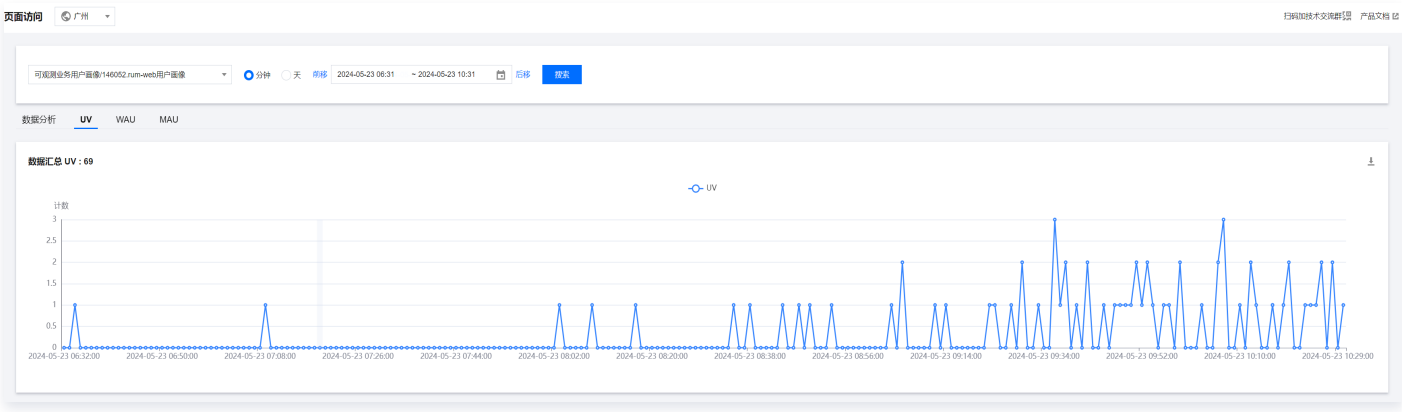
   - npm 引入：

```
$ npm install aegis-web-sdk
```

```
import Aegis from '@tencent/aegis-web-sdk/lib/aegis.f.min';
```

> **说明：**
> SDK version 需要大于 1.38.56 版本。

2. 登录 腾讯云可观测 > 前端性能监控控制台，在左侧菜单栏选择**页面访问**，即可查看 UV。



更多应用场景说明

与此同时，配合业务场景进行技术赋能，可以扩展出更多能力。例如安全部门基于指纹 SDK 扩展出安全审计功能等。

小程序场景
安装和初始化

最近更新时间：2024-05-21 14:48:11

注意事项

- 小程序仅支持 NPM 方式安装 SDK
- 本 SDK 支持微信小程序和 QQ 小程序
- 正式环境接入小程序需要将上报域名添加到安全域名中

> **说明**
> aegis-sdk 默认使用 `https://aegis.qq.com` 作为上报域名，您可以通过 `hostUrl` 参数来控制上报域名。
> 国内可以选择使用 `https://rumt-zh.com` 作为上报域名。
> 新加坡地区可以选择使用 `https://rumt-sg.com` 作为上报域名。
> 硅谷地区可以选择使用 `https://rumt-us.com` 作为上报域名。

开发者需要根据小程序 network 中 aegis sdk 上报接口所使用的域名来判断需要将哪个域名添加到小程序安全域名中。

安装 SDK

执行下列命令，在 npm 仓库安装 aegis-mp-sdk。

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

初始化

参见下列步骤新建一个 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: "pGUVFTCZyewxxxxx", // RUM 上申请的上报 key
  uin: 'xxx', // 用户唯一 ID（可选）
  reportApiSpeed: true, // 接口测速
  reportAssetSpeed: true, // 静态资源测速
  hostUrl: 'https://rumt-zh.com', // 上报域名，中国大陆 rumt-zh.com
  spa: true, // 页面切换的时候上报 pv
});
```

> **说明**
> 为了不遗漏数据，须尽早进行初始化。如果您的小程序应用中使用了 `miniprogram-api-promise` 来封装 `wx.request` 请求，需要注意：由于我们是通过重写 `wx.request` 来进行接口监控的，所以需要您将初始化 Aegis 放在引入这个包之前执行，否则可能会导致接口信息无法完整收集。
> 当您完成安装并初始化 SDK 之后，可以开始使用前端性能监控提供的以下功能：
> 1. 错误监控：JS 执行错误
> 2. 测速功能：接口测速
> 3. 数据统计和分析：可在 数据总览 页面上进行各个维度的数据分析

日志上报

最近更新时间：2024-07-11 15:30:11

日志查询功能需您上报日志后，才能正常使用。本文将为您介绍如何上报日志到前端性能监控。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

日志上报

引入下列参数，配置 SDK ，完成日志上报。

```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.logType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

Aegis.logType 枚举值如下：

```
export enum LogType {
  API_RESPONSE = '1', // 白名单中的用户，页面上的所有 API 返回都将会被上报
  INFO = '2', // aegis.info、aegis.infoAll 上报的日志
  ERROR = '4', // js 错误
  PROMISE_ERROR = '8', // promise 错误
  AJAX_ERROR = '16', // ajax 错误
  SCRIPT_ERROR = '32', // script 加载失败
  IMAGE_ERROR = '64', // 图片加载失败
  CSS_ERROR = '128', // css 加载失败
  CONSOLE_ERROR = '256', // console.error 监控（目前暂未支持）
  MEDIA_ERROR = '512', // 音视频加载失败
  RET_ERROR = '1024', // retcode 返回码异常
  REPORT = '2048', // aegis.report 上报日志默认level
  PV = '4096', // 页面 PV
  EVENT = '8192', // 自定义事件
  PAGE_NOT_FOUND_ERROR = '16384', // 小程序 页面不存在
  WEBSOCKET_ERROR = '32768', // websocket错误
  BRIDGE_ERROR = '65536', // js bridge 错误
}
```

aid

最近更新时间：2024-11-06 16:01:41

aid 是 Aegis SDK 为每个用户设备分配的唯一标识，存储在小程序的 storage。用于区分用户计算 UV 等。只有用户清理小程序缓存 aid 才会更新。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

aid

您可以根据下列算法自定义 aid 上报规则：

```
aid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
const r = (Math.random() * 16) | 0;
const v = c === 'x' ? r : (r & 0x3) | 0x8;
return v.toString(16);
```

实例方法

最近更新时间：2024-11-05 14:33:11

前端性能监控为您提供多种实例方法用于上报数据，您可以通过实例方法修改实例配置、自定义上报事件、自定义上报测试资源等。
目前 RUM 提供的 Aegis 实例方法如下：

参数	用途
setConfig	传入配置对象，包括用户 ID 和 UIN 等信息
info	主要上报字段，用于指定用户上报白名单日志。 下列两种情况日志才会报到后台： 1. 打开页面的用户在名单中。 2. 对应的页面发生了错误。
infoAll	主要上报字段，用于所有用户上报白名单日志。
error	主要上报字段，用于上报错误信息。
report	用来上报任意类型的日志信息。
reportEvent	上报自定义事件。
reportTime	上报自定义测速资源。
time	上报自定义测速资源，与 timeEnd 共同使用。适用于两个时间点之间时长的计算并上报。
timeEnd	上报自定义测速资源，与 time 共同使用。适用于两个时间点之间时长的计算并上报。
destroy	销毁 aegis 实例。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

实例方法

setConfig

该方法用于修改实例配置，使用场景如下：
1. 可获取到用户 UIN ，可同时传入配置用户 ID 和 UIN 两个实例对象，进行实例化：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  uin: '777'
})
```

2. 通常情况下，我们并不能一开始就获取到用户的 `uin` 。若在获取 UIN 的这段时间不进行实例化，这期间发生的错误前端性能监控将无法监听。针对这种情况，我们可以先传入 ID 进行实例化，引用 setConfig 传入 UIN ，示例如下：

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx'
})

// 拿到uin之后...
aegis.setConfig({
  uin: '6666'
```

```
})
```

info、infoAll、error 和 report

这三个方法是前端性能监控提供的主要上报手段。

```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.logType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

Aegis.logType 枚举值如下

```
export enum LogType {
  API_RESPONSE = '1', // 白名单中的用户，页面上的所有 API 返回都将会被上报
  INFO = '2', // aegis.info、aegis.infoAll 上报的日志
```

```
  ERROR = '4', // js 错误
  PROMISE_ERROR = '8', // promise 错误
  AJAX_ERROR = '16', // ajax 错误
  SCRIPT_ERROR = '32', // script 加载失败
  IMAGE_ERROR = '64', // 图片加载失败
  CSS_ERROR = '128', // css 加载失败
  CONSOLE_ERROR = '256', // console.error 监控（目前暂未支持）
  MEDIA_ERROR = '512', // 音视频加载失败
  RET_ERROR = '1024', // retcode 返回码异常
  REPORT = '2048', // aegis.report 上报日志默认level
  PV = '4096', // 页面 PV
  EVENT = '8192', // 自定义事件
  PAGE_NOT_FOUND_ERROR = '16384', // 小程序 页面不存在
  WEBSOCKET_ERROR = '32768', // websocket错误
  BRIDGE_ERROR = '65536', // js bridge 错误
}
```

reportEvent

该方法可用来上报自定义事件，系统将会自动统计上报事件的各项指标，例如：PV、平台分布等。
reportEvent 可以支持字符串和对象两种类型上报参数。

字符串类型

```
aegis.reportEvent('XXX请求成功');
```

对象类型

ext1、ext2 和 ext3 默认使用 new Aegis 的时候传入的参数，自定义事件上报的时候，可以覆盖默认值。

```
aegis.reportEvent({
  name: 'XXX请求成功', // 必填
  ext1: '额外参数1',
  ext2: '额外参数2',
  ext3: '额外参数3',
})
```

> ⚠ 注意
> 额外参数的三个 key 是固定的，目前只支持 ext1、ext2 和 ext3 。

reportTime

该方法可用来上报自定义测速，例如：

```
// 假如‘onload’的时间是1s
aegis.reportTime('onload', 1000);
```

或者如果需要使用额外参数，可以传入对象类型参数，ext1，ext2，ext3 会覆盖默认值：

```
aegis.reportTime({
```

```
  name: 'onload', // 自定义测速 name
  duration: 1000, // 自定义测速耗时(0 - 60000)
  ext1: 'test1',
  ext2: 'test2',
  ext3: 'test3',
});
```

说明

`onload` 可以修改为其他的命名。

time 和 timeEnd

该方法同样可用来上报自定义测速，适用于两个时间点之间时长的计算并上报，例如：

```
aegis.time('complexOperation');
/**
 * .
 * .
 * 做了很久的复杂操作之后。
 * .
 * .
 */
aegis.timeEnd('complexOperation'); /** 此时日志已经报上去了**/
```

说明

- `complexOperation` 可以修改为其他的命名。
- 自定义测速是用户上报任意值，服务端对其进行统计和计算。由于服务端不能做脏数据处理，建议用户在上报端进行统计值限制，防止脏数据对整体产生影响。
- 目前 Aegis 只支持 0 - 60000 的数值计算，如果大于该值，建议进行合理改造。

destroy

销毁实例进程，销毁后数据不再上报，并且 Aegis 不再收集用户数据。

```
aegis.destroy();
```

白名单

最近更新时间：2024-07-11 15:30:11

白名单功能是适用于开发者针对特定的用户上报更多的信息。为了过滤部分用户，并减少用户的接口请求次数，因此 RUM 提供了白名单功能，并设定了白名单的逻辑。
白名单逻辑如下：

1. 白名单用户会上报全部的 API 请求信息，包括接口请求和请求结果。
2. 白名单用户可以使用 info 接口上报数据。
3. info 和 infoAll：在开发者实际体验过程中，白名单用户可以添加更多的日志，并且使用 info 进行上报。infoAll 会对所有用户无差别进行上报，因此可能导致日志量上报巨大。
4. 通过接口 whitelist 来判断当前用户是否是白名单用户，白名单用户的返回结果会绑定在 Aegis 实例上 (aegis.isWhiteList) 用来给开发者使用。
5. 为了减少开发者使用负担，白名单用户是针对业务系统生效。您可以在 应用管理 > 白名单管理 内创建白名单，则业务系统下全部应用都对该白名单用户生效。

钩子函数

最近更新时间：2024-12-18 10:02:33

您可以使用钩子函数对某些资源的测速上报进行自定义配置，系统将会为您计算、统计函数执行时间等。您可以在 自定义上报 > 自定义测速 多维度分析函数执行耗时。

beforeRequest

该钩子将会在所有的数据上报前执行，来帮助用户在数据上报前对其进行屏蔽和修改，当 beforeRequest 返回 false 就可以不上报该条日志，返回修改过后的 data 就可以实现对上报数据的修改，例如：

> **注意：**
> SDK 版本应大于等于1.24.44。

```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeRequest: function(data) {
    // 入参msg的数据结构：{logs: {…}, logType: "log"}
    if (data.logType === 'log' && data.logs.msg.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 log，且内容包含 otheve.beacon.qq.com 的请求
      return false;
    }
    // 入参data数据结构：{logs: {}, logType: "speed"}
    if (data.logType === 'speed' && data.logs.url.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 speed，并且接口 url 包含 otheve.beacon.qq.com 的请求
      return false;
    }
    if (data.logType === 'performance') {
      // 修改：将性能数据的首屏渲染时间改为2s
      data.logs.firstScreenTiming = 2000;
    }
    return data;
  }
});
```

其中，msg 将会有以下几个字段：

- logType：日志类型，有以下值：
  - custom：自定义测速
  - event：自定义事件
  - log：日志
  - performance：页面测速
  - pv：页面PV
  - speed：接口和静态资源测速
  - vitals：web vitals
- logs：上报的日志内容：
  - 当 logType 为 'custom' 时，logs 数据类型为 `{name: "白屏时间", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}`
  - 当 logType 为 'event' 时，logs 数据类型为 `{name: "ios", ext1: "", ext2: "", ext3: ""}`
  - 当 logType 为 'performance' 时，logs 数据类型为 `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 260, ssl: 4, tcp: 4, ttfb: 5}`

- 当 logType 为 'speed' 时，logs 数据类型为
  ```
  {connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "tatic", url: "https://xxxxxx", urlQuery: "max_age=1296000"}
  ```
- 当 logType 为 'vitals' 时，logs 数据类型为
  ```
  {delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "CP", value: 1100}
  ```
- 当 logType 为 'log' 时，logs 数据类型为 `{msg: "日志详情", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}`

说明

其中 level 枚举值如下

- `{ level: '1', name: '接口请求日志（白名单日志）' }`
- `{ level: '2', name: '一般日志' }`
- `{ level: '4', name: 'JS 执行错误' }`
- `{ level: '8', name: 'Promise 错误' }`
- `{ level: '16', name: 'Ajax 请求异常' }`
- `{ level: '32', name: 'JS 加载异常' }`
- `{ level: '64', name: '图片加载异常' }`
- `{ level: '128', name: 'css 加载异常' }`
- `{ level: '256', name: 'console.error（未启用）' }`
- `{ level: '512', name: '音视频资源异常' }`
- `{ level: '1024', name: 'retcode 异常' }`
- `{ level: '2048', name: 'aegis report' }`
- `{ level: 4096, name: 'PV' }`
- `{ level: 8192, name: '自定义事件' }`
- `{ level: 16384, name: '小程序 页面不存在' }`
- `{ level: 32768, name: 'websocket错误' }`

该钩子返回 false 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，可以用来过滤不希望上报的日志。

afterRequest

该钩子将会在测速数据上报后被执行，例如：

注意

SDK 版本应大于等于1.24.44。

```
const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx",
  afterRequest: function(msg) {
    // {isErr: false, result: Array(1), logType: "log", logs: Array(4)}
    console.log(msg);
  }
});
```

其中，msg 将会有以下几个字段：

- isErr：请求上报接口是否错误。
- result：上报接口的返回结果。
- logs：上报的日志内容。
- logType：日志类型，同 beforeRequest 中的 logType。

错误监控

最近更新时间：2025-01-06 17:33:42

前端性能监控的 Aegis 的实例会自动进行 JS 执行错误、Promise 执行错误、Ajax（Fetch）请求异常等监控。本文将为您介绍各错误监控逻辑及处理方式。

> **注意：**
> Aegis 实例会对这些异常进行监控，当您只是引入了 SDK 而没有将其实例化时，Aegis 将不会上报数据。

JS 执行错误

Aegis 通过监听 `wx` 对象上的 `onerror` 事件来获取应用中的报错，并且通过解析错误和分析堆栈，将错误信息自动上报到后台服务中。该上报的上报等级为 error，所以当自动上报的错误达到阈值时，Aegis 将会自动告警，帮助您尽早发现异常。由于上报等级为 error，自动上报也将影响应用的评分。

request 请求异常

Aegis 会改写 `wx.request`（`qq.request`），并在每次接口请求时进行监听。当 `statusCode` 不存在或者大于等于400时，认为该请求是一个失败的请求，亦或是接口超时，abort 和其他类型的失败请求。
如果您的应用中使用了一些库也会重写 `wx.request`（`qq.request`），如：`miniprogram-api-promise` 或者自己有封装，请一定确保在引入这个库之前完成对 Aegis 的初始化，否则将无法监听到接口失败的情况。

> **注意：**
> Aegis SDK 在错误发生的时候，不会主动收集接口请求参数和返回信息，如果需要对接口信息进行上报，可以使用 API 参数里面的 apiDetail 进行开启。

```
new Aegis({
  api: {
    apiDetail: true,
  },
});
```

retcode 异常

Aegis 在改写 `wx.request`（`qq.request`）时，会获得 API 返回的内容，并尝试在内容中获取到本次请求的 `retcode`。

- retcode 的值默认会从用户返回 response body 的第一层（如果第一层取不到，再取第二层）的 code、ret、retcode、errcode 中获取。
- Aegis 默认 retcode 的值为0，表示正常，非0都是异常。当 retcode 发生异常时，会上报一个 retcode 异常的日志。
- 用户可以通过 api.retCodeHandler 对这个值和是否异常进行修正。

> **说明：**
> 如何获取 `retcode` 以及查看正常的 `retcode` 请参见 配置文档。

websocket 异常

Aegis 默认不会对全局 websocket 对象进行重写，默认不会监控 websocket 相关错误，如果需要打开相关功能可以在创建 Aegis 对象时设置 websocketHack: true。

```
new Aegis({
  id: '',
  websocketHack: true
```

```
});
```

性能监控

最近更新时间：2024-08-16 15:02:21

您可以通过本文了解页面测速、接口测试等信息。

页面测速

Aegis 小程序 SDK 会自动收集页面性能数据并上报。包括：

1. 程序启动时间。
2. 代码注入时间。
3. 页面首次渲染时间。
4. 路由切换时间。

> **注意：**
> - 获取页面性能数据依赖小程序的 Performance API，请保证基础库版本大于 `2.11.0` 。
> - QQ 小程序目前暂时不支持 Performance API，所以不会上报页面性能数据。

接口测速

> **说明：**
> 打开方式：初始化时传入配置 `reportApiSpeed: true`

Aegis 通过劫持 `wx.request || qq.request` 进行接口测速。

配置文档

最近更新时间：2024-11-04 18:22:32

配置说明

配置文档各配置项说明如下：

配置	描述
id	必须，string，默认 `无` 。 开发者平台分配的应用上报 ID。
uin	建议，string，默认取 cookie 中的 UIN 字段。 当前用户的唯一标识符，白名单上报时将根据该字段判定用户是否在白名单中，字段仅支持 `字母数字@=._-` ，正则表达式： `/^[@=.0-9a-zA-Z_-]{1,60}$/` 。
onError	可选，boolean，默认 true。 当前实例是否需要进行错误监听，获取错误日志。
reportApiSpeed	可选，boolean，默认 false。 是否开启接口测速。
reportAssetSpeed	可选，boolean，默认 false。 是否开启静态资源测速。
reportLoadPackageSpeed	可选，boolean，默认 false。 是否开启包下载耗时上报。
pagePerformance	可选，boolean，默认 true。 是否开启页面测速。
version	可选，string，默认 sdk 版本号。 当前上报版本，当页面使用了 pwa 或者存在离线包时，可用来判断当前的上报是来自哪一个版本的代码，仅支持 `字母数字.,:_-` ，长度在 60 位以内 `/^[0-9a-zA-Z.,:_-]{1,60}$/` 。
delay	可选，number，默认 1000 ms。 上报节流时间，在该时间段内的上报将会合并到一个上报请求中。
repeat	可选，number，默认 5。 重复上报次数，对于同一个错误超过多少次不上报，同一个接口上报测速数据的次数。
env	可选，enum，默认 Aegis.environment.production。当前应用运行所处的环境，详情请参见说明文档 环境控制 。
random	可选，number，默认 1。 0 - 1 抽样率。 0 表示全部不上报。
spa	可选，boolean，默认 false。 是否在小程序页面跳转时进行 PV 上报。
pageUrl	可选，string，默认是小程序 getCurrentPages。 修改上报数据中页面地址from，开发者可以主动对数据进行聚合和降低维度
websocketHack	可选，boolean，默认 false。 是否开启 websocket 监控。开启该选项后，将会对 webscoket 连接断开状态进行监控，但是新建操作仍需要自定义监控。
hostUrl	可选，默认是 `https://aegis.qq.com` 。 影响全部上报数据的 host 地址，设置 hostUrl 后，下面几个 URL 地址都会被覆盖。

url	可选，string，默认 `https://aegis.qq.com/collect` 。 日志上报地址。 设置为空字符串可以不进行日志上报。
pvUrl	可选，string，默认 `https://aegis.qq.com/collect/pv` 。 pv 上报地址。 设置为空字符串可以不进行 pv 上报。
whiteListUrl	可选，string，默认 `https://aegis.qq.com/collect/whitelist` 。 白名单确认接口， 设置为空字符串可以关闭白名单接口请求。
eventUrl	可选，string，默认 `https://aegis.qq.com/collect/events` 。 自定义事件上报地址。 设置为空字符串可以不进行自定义事件上报。
speedUrl	可选，string，默认 `https://aegis.qq.com/speed` 。 测速日志上报地址。 设置为空字符串可以不进行测速数据上报。
customTimeUrl	可选，string，默认 `https://aegis.qq.com/speed/custom` 。 自定义测速上报地址。 设置为空字符串可以不进行自定义测速上报。
performanceUrl	可选，string，默认 `https://aegis.qq.com/speed/performance` 。 设置页面性能日志上报地址。 设置为空字符串可以不进行页面性能上报。
setDataReportUrl	可选，string，默认 `https://aegis.qq.com/speed/speed/miniProgramData` 。 设置小程序 setData 性能日志上报地址。
setDataReportConfig	可选，object，默认为 `{}` 。相关的配置： • disabled：可选，Boolean，默认 false。是否禁用 setData 数据上报。 • timeThreshold：可选，Number，单位为ms，默认值为30。上报的耗时阈值，表示仅上报更新耗时超过该阈值的数据。 • withDataPaths：可选，Boolean，默认为 true。是否上报本次更新的字段信息。
api	可选，object，默认为 `{}` 。相关的配置： • apiDetail：可选，boolean，默认 false。上报 api 信息的时候，是否上报 api 的请求参数和返回值。 • enableResDetail：可选，boolean，默认 false。用于控制是否上报 API 返回体中除了 data 之外的包体属性。 • retCodeHandler：`Function(data: String, url: String, xhr: Object):{isErr: boolean, code: string}` ，返回码上报钩子函数。会传入接口返回数据，请求 url 和 xhr 对象；详情请参见示例 api.retCodeHandler 。 • reqParamHandler：Function(data: any, url: String) 上报请求参数的处理函数，可以对接口的请求参数进行处理，方便用户过滤上报请求参数的信息。 • resBodyHandler：Function(data: any, url: String) 上报 response 返回 body 的处理函数，可以对接口返回值的 response body 进行处理，只上报关键信息。若开启 enableResDetail，可以捕获小程序接入安全网关后的全部返回的内容，不仅是 res data，另外在请求日志的写入上也会有对应反馈。 • resourceTypeHandler：Function，请求资源类型修正钩子函数会传入接口 url，返回值为 `static` 或 `fetch` 。 • reportRequest：boolean，默认为 false。开启后，aegis.info 会变成全量上报，不需要白名单配置，并且会上报所有接口的信息（上报接口需开启 reportApiSpeed）。 • reqHeaders：Array 需要上报的 HTTP 请求 request headers 列表。 • resHeaders：Array 需要上报的 HTTP 请求 response headers 列表。如果开发者需要获取的字段非“simple response header”，需要给在返回头中给字段添加 Access-Control-Expose-Headers。

ext1	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext2	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext3	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。

示例

api.retCodeHandler

假如后台返回数据为：

```
{
  body: {
    code: 200,
    retCode: 0,
    data: {
        // xxx
  }
}
```

业务需要：code 不为200，或者 retCode 不为0，此次请求就是错误的。此时只需进行以下配置：

```
new Aegis({
// xxx
  reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
  reportAssetSpeed: true,
  api: {
    retCodeHandler(data) {
      // 注意这里拿到的data是string类型，如果需要对象需要手动parse下
      try {
        data = JSON.parse(data)
      } catch (e) {
      }
      return {
        // isErr 如果是 true 的话，会上报一条 retcode 异常的日志。
        isErr: data.body.code !== 200 || data.body.retCode !== 0,
        code: data.body.code
      }
    }
  }
})
```

Aegis SDK 支持获取请求头和返回头

最近更新时间：2024-06-25 18:08:21

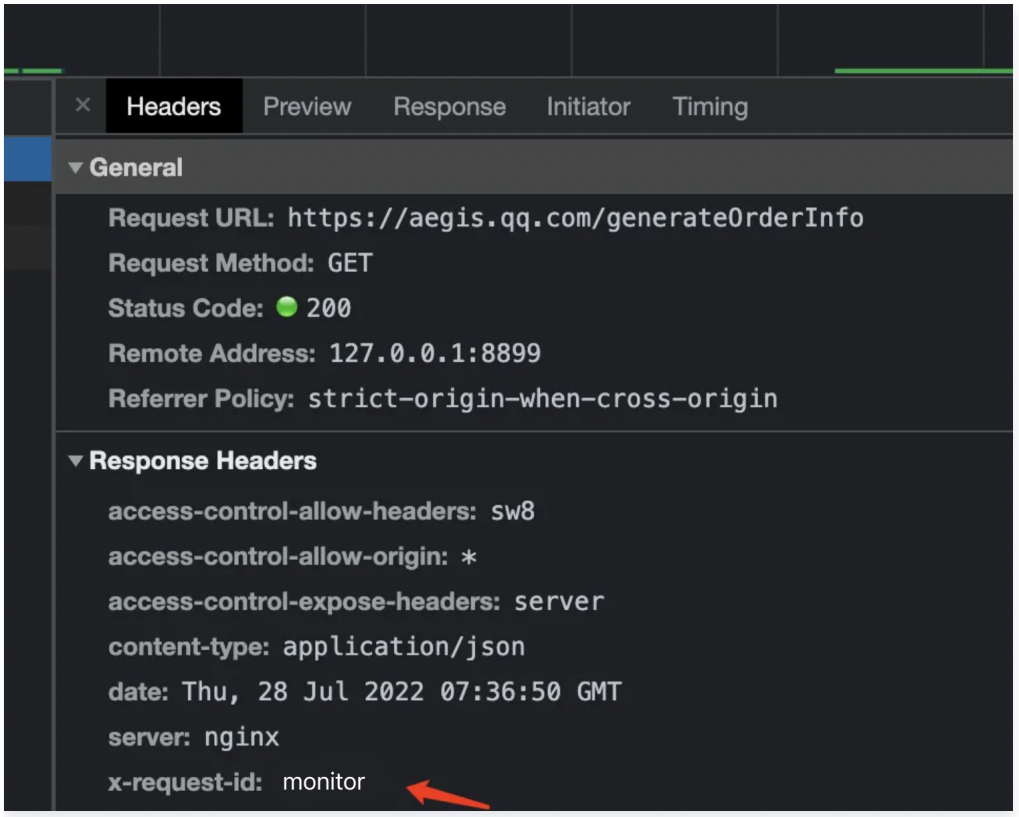
通过记录自定义请求头和响应头的方式，打通前后端链路。

> **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 Header `x-request-id`，值是 `monitor`，为前后端打通的唯一标识。



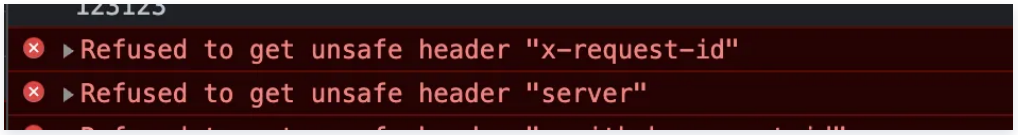
假设需要通过记录这个值到 RUM 来打通前后端的链路，实践步骤如下：

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
      },
  });
```

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。

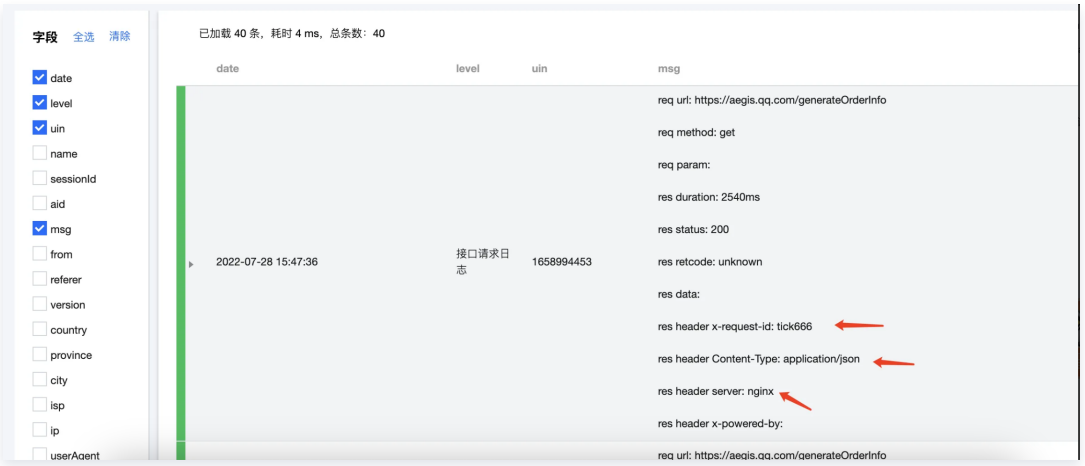


> **说明：**
> 具体报错原因可参见 相关文档 。

此时需要暴露 unsafe header，给要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：

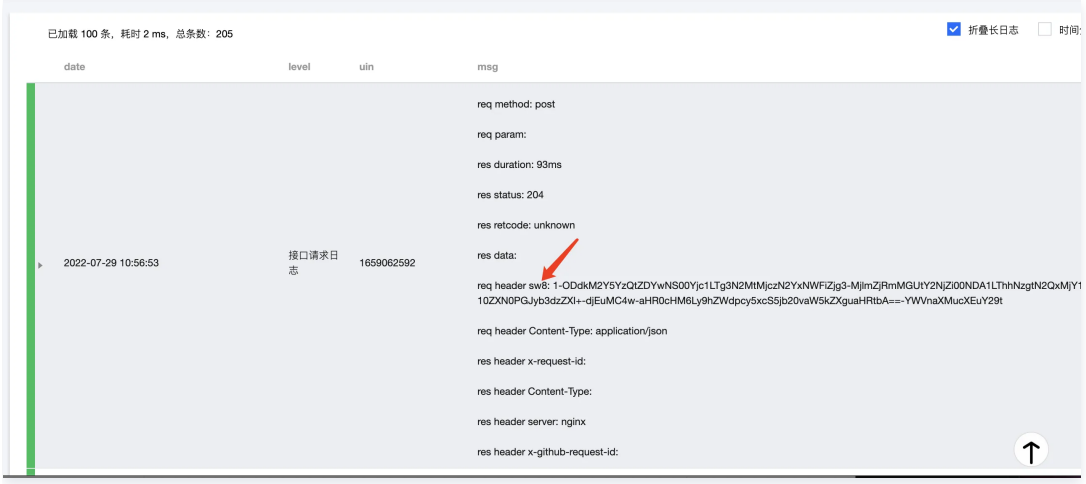


获取 Request Headers

获取 request headers 的使用方式与获取 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 以外的第三方监控 SDK。
具体使用 skywalking 打通前后端的方式可参见 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 上查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。

控制台操作指南
应用接入

最近更新时间：2024-05-10 18:42:31

前端性能监控支持 Web、小程序（微信、QQ）、Hippy、Weex、React Native 、Flutter 和 Cocos 应用类型接入。本文将为您介绍如何进行应用接入。

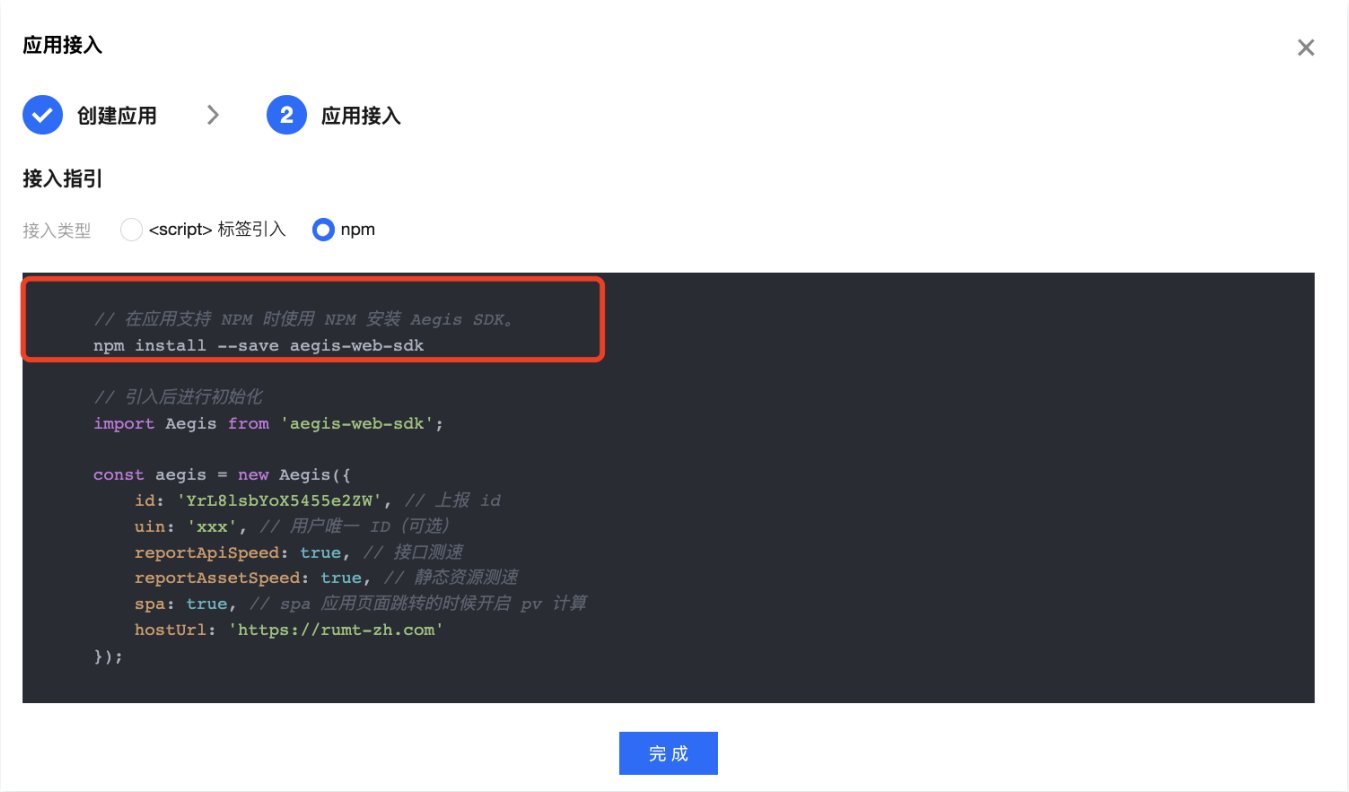
操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击**前端性能监控 > 数据总览**。
3. 在数据总览页中单击**应用接入**，根据下列表格配置创建应用信息。

配置项	说明
应用名称	自定义应用名称，方便您在前端监控平台辨识该应用。
应用描述	填写应用描述 ，如应用用途、应用简介等，方便其它用户了解该应用。
应用类型	支持 Web、小程序（微信、QQ）、Hippy、Weex 、React Native 、 Flutter 和 Cocos 应用类型接入。
应用仓库地址（可选）	填写您的应用仓库地址，可不填写。
数据上报域名(填 * 表示不做任何校验)	默认 * 不作任何校验。
所属业务系统	该功能用于分类管理您接入的应用，您可以根据研发团队、业务逻辑、应用类别等进行应用分类管理。若您没有可用团队，您可以单击右侧的创建链接，填写完信息后，单击**购买**即创建成功。

4.配置完后单击**下一步**，参考下列说明选择一种方式安装 SDK 。

- **npm** 方式安装 SDK（所有应用类型均可使用该方式接入）。下面以 Web 应用为例说明如何通过 npm 方式接入 SDK。
  i. 在接入指引页面中复制提供的首行命令，引入 npm 包。



ii. 在接入指引页面中复制提供的代码初始化 SDK。

- **<script> 标签引入方式接入 SDK（仅支持 Web 接入类型）。**
  i. 在接入指引页面复制提供的 `<script>` 标签 代码。
  ii. 把**<script> 标签引入**类型下的代码引入到 `<head></head>` 标签中即可。

说明

按照上述步骤接入后即可使用数据总览、页面性能、异常分析、页面访问（PV、UV）、API 监控和静态资源功能。如需使用日志查询、自定义测速和自定义事件，需参见 接入指引 上报数据。

数据总览

最近更新时间：2024-07-10 17:56:21

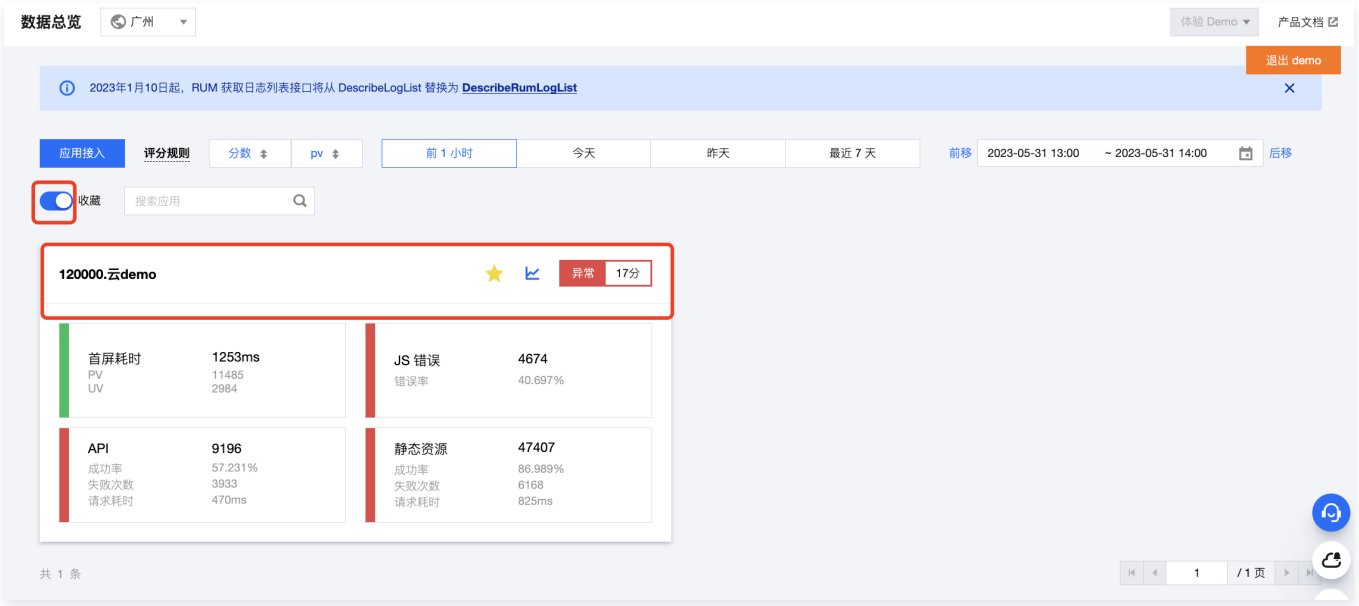
本文将为您介绍如何查看数据总览。

前提条件

在您开始执行操作前，请确认已在前端性能监控接入应用。如何接入应用，请参见 应用接入 。

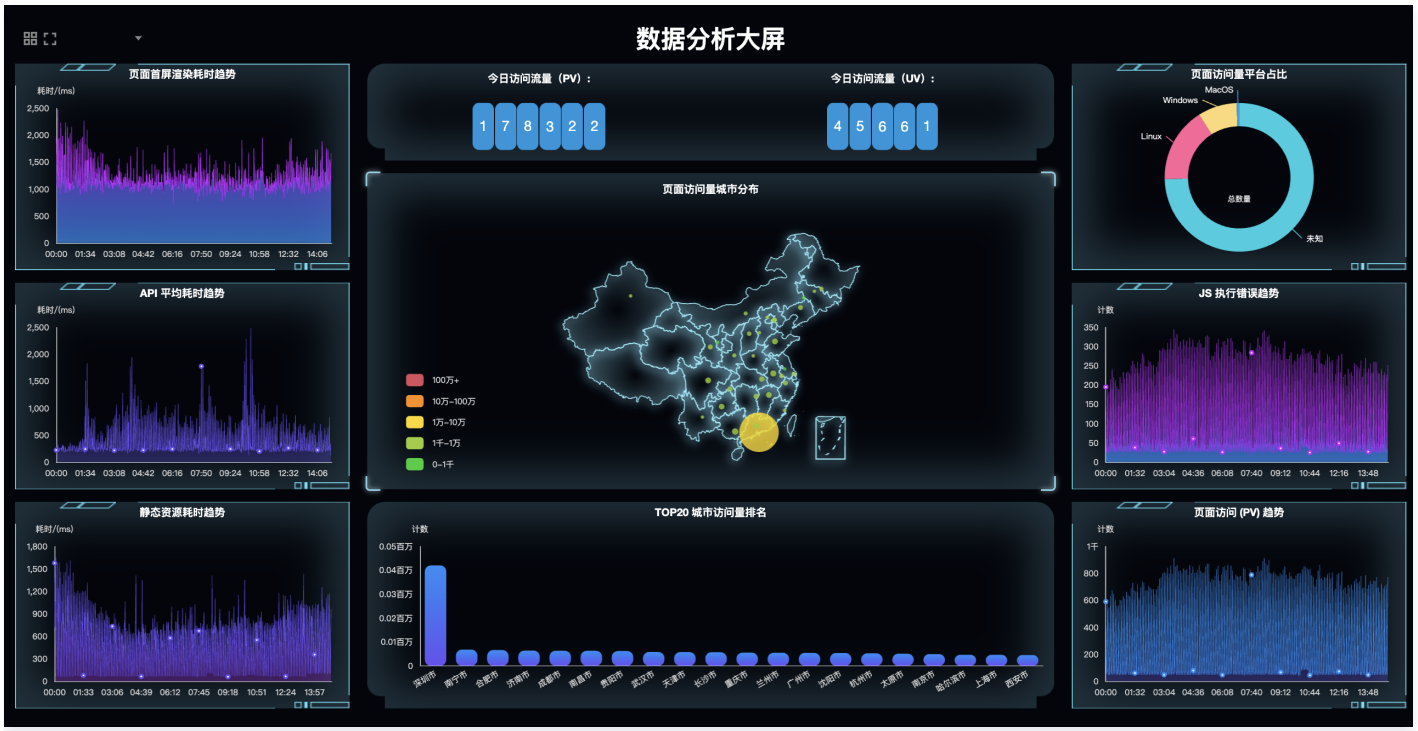
操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击**前端性能监控 > 数据总览**。
3. 在数据总览页面，即可查看所有应用的关键指标信息（包括 PV 、首屏耗时、JS 错误数/率、API 和静态资源请求成功率、失败率、请求耗时）和应用总体评分。
   您可以将鼠标移动至评分左侧，进行按分数或者按 PV 排序。单击面板中的收藏键收藏该应用面板， 开启收藏后可一键查看所有收藏面板。



数据分析大屏

您可以在数据总览页面，单击各应用模块中的折线图按钮 ，即可查看数据分析大屏。



计分规则

不同指标评分规则和占比不同，说明如下：

评分指标	评分规则	占比
页面报错率（页面报错次数 / 页面打开次数）	1. 当报错率 ≤ 0.5%，得分100 2. 当0.5% < 报错率 < 10%，得分：100 − 10 × 报错率 3. 当报错率 ≥ 10%，得分 = 0	30%
页面打开平均耗时	1. 小于等于 1000ms，100分 2. 大于 1000ms，每增加 N 个100ms，得分：100 − 10 × N，最低0分	10%
接口成功率（接口访问成功次数 / 访问总次数）	同页面报错率	30%
接口访问平均耗时	同页面打开平均耗时	5%
静态资源请求成功率（静态资源成功次数 / 请求总次数）	同页面报错率	20%
静态资源平均耗时	同页面打开平均耗时	5%

各应用关键指标色块含义

指标名称	绿色	橙色	红色	灰色
首屏耗时	耗时 ≤ 1000ms	1000ms ≤ 耗时 ≤ 3000ms	耗时 > 3000ms	信息缺失
JS 错误	JS 错误率 ≤ 0.5%	0.5% < JS 错误率 < 10%	JS 错误率 ≥ 10%	信息缺失
API	成功率 > 99.5%	90% ≤ 成功率 ≤ 99.5%	成功率< 90%	信息缺失
静态资源	成功率 > 99.5%	90% ≤ 成功率 ≤ 99.5%	成功率 < 90%	信息缺失

页面性能

最近更新时间：2024-07-11 15:22:51

页面性能模块支持多维度分析页面性能情况，您可以通过性能变化趋势图、页面加载瀑布图、地区视图等维度分析首屏时间、请求响应等页面性能关键指标。本文将为您介绍如何查看页面性能。

前提条件

在您开始执行操作前，请确认已在前端性能监控接入应用。如何接入应用，请参 应用接入 。

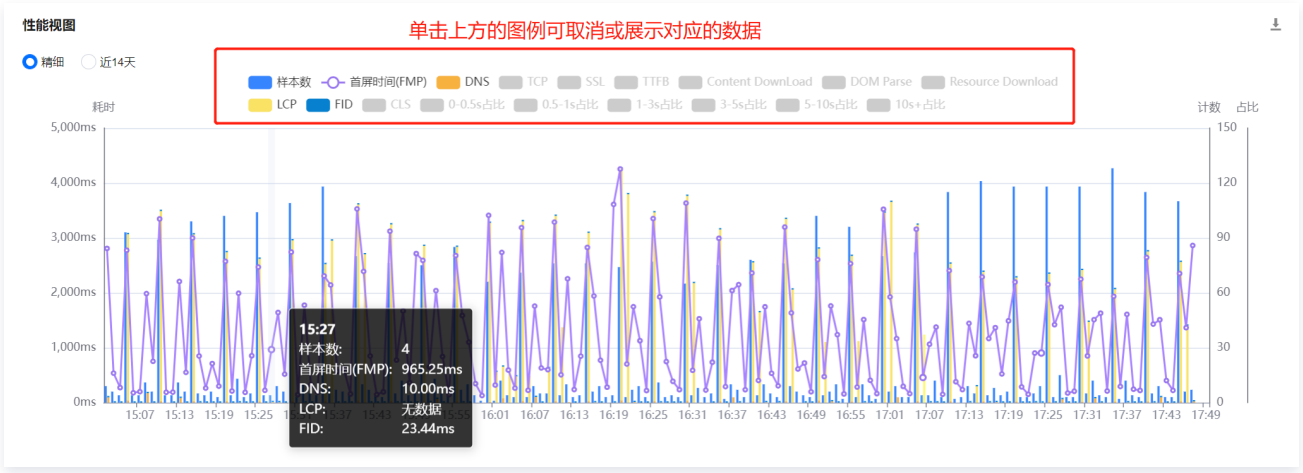
操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击**前端性能监控 > 页面性能**。
3. 进入页面性能页面即可查看页面性能情况。

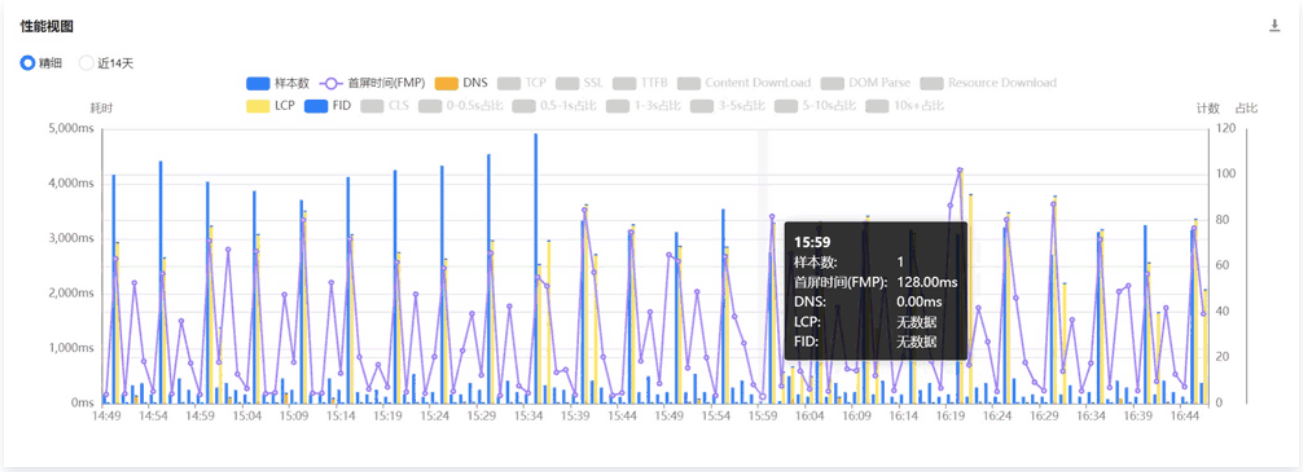
性能视图

展示页面性能关键指标变化趋势图。

- 单击上方的图例可取消或展示对应的数据。

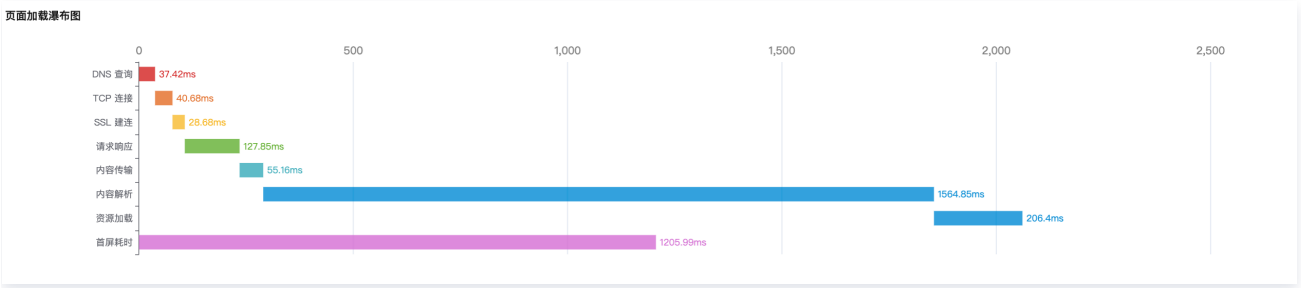


- 支持按时间段或近14天粒度展示变化趋势图。
- 在曲线中拖动鼠标可展示某一时刻首屏渲染时间。
- 把鼠标移动到图表中，并向上滑动鼠标可缩小图表时间跨度，向下滑动鼠标可放大图表时间跨度。



页面加载瀑布图

通过页面加载瀑布图可查看各阶段的耗时情况，您可以根据各阶段耗时情况优化页面性能。
页面完全加载时间为 TCP、DNS、SSL、TTFB、DOM 解析和资源加载的时间之和。



Core Web Vitals

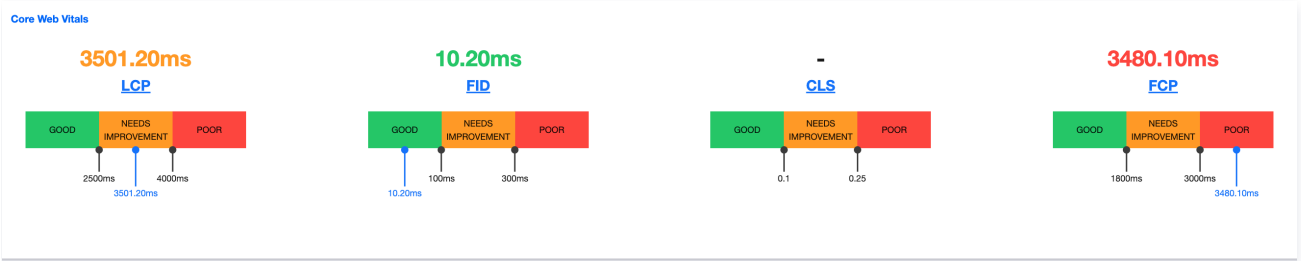
Core Web Vitals 通过不同的角度加载速度，交互性和视觉稳定性反映了用户的体验，并根据良好，需要优化，较差三个等级的进行评分，协助您进行用户体验优化。
Core Web Vitals 包含四个核心基础指标 LCP、FID、CLS 和 FCP，含义如下：

指标名称	英文名称	含义
LCP	Largest Contentful Paint	从用户请求网址到在可视窗口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
FID	First Input Delay	从用户首次与您的网页互动（单击链接、单击按钮等）到浏览器响应此次互动之间的用时。这种衡量方案的对象是被用户首次点击的任何互动式元素
CLS	Cumulative Layout Shift	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
FCP	First Contentful Paint	测量页面从开始加载到页面内容的任何部分在屏幕上完成渲染的时间。

指标等级：

指标等级	含义
GOOD	良好
NEEDS IMPROVEMENT	需要优化
POOR	较差

如下图，表示您的 LCP 性能处于建议优化等级，可从渲染最大可见内容元素（图片、视频）方向进行优化。



其它视图

视图名称	说明

页面性能 TOP 视图	展示页面性能最差的 5个页面。包括首屏时间、页面完全加载时间、与前一天的对比情况等数据。帮助您了解性能较差的页面情况，并快速定位性能瓶颈。
网络/平台视图	用饼图形式展示来自各地域网络或平台的异常数量、占比及首屏时间 。其中网络包括3G、4G、WIFI 等，平台包括 Macos、Windows、IOS 等
ISP 视图	用饼图形式展示来自各运营商的异常数量、占比及首屏时间。运营商包括移动，电信、联通等。
地区视图	用饼图形式展示来自各区域的异常数量、占比及首屏时间。
品牌/机型视图	用饼图形式展示来自手机品牌/机型的异常数量、占比及首屏时间。
浏览器视图	用饼图形式展示来自各浏览器的异常数量、占比及首屏时间。
Version 视图	用饼图形式展示来自各应用版本的异常数量、占比及首屏时间。您可以通过在接入应用时的 new Aegis 传入 version 来自定义研发相关的版本信息，默认使用 SDK 的版本。
Ext1 视图、Ext2 视图、Ext3 视图	自定义视图，由您自定义上报时传入参数。详情请参见 接入指南 。

注意：

视图都需要勾选自定义视图选择才可生效；如下所示：

- **运营商：ISP 视图。**
- **网络：网络视图。**
- **平台：平台视图。**
- **浏览器：浏览器视图。**
- **操作系统：操作系统视图。**
- **版本：Version 视图。**
- **地域：地区视图。**

指标说明

指标名称	单位	含义
首屏渲染时间（FMP）	ms	从用户请求打开新网页到浏览器呈现出首屏内容所用的时间
页面完全加载	ms	从用户请求打开新网页到浏览器完全呈现出相应网页所用的时间。页面完全加载时间 = TCP + DNS + SSL + TTFB + DOM 解析 + 资源加载
首字节（TTFB）	ms	发出页面请求到接收到应答数据第一个字节所花费的毫秒数
DOM 解析（DOM Parse）	ms	DOM 解析耗时
DOM Ready	ms	初始的 HTML 文档被完全加载和解析完成时间

DNS	ms	DNS 查询耗时
TCP	ms	TCP 连接耗时
SSL	ms	SSL 安全连接耗时
响应请求	ms	响应请求耗时
Resource Download	ms	资源加载耗时
Content Download	ms	表示浏览器从接收到第一个字节到最后一个字节所花费的时间

性能数据分析

最近更新时间：2024-09-09 18:18:21

前端性能监控支持您对下列数据进行分析：

类型	功能说明
页面性能	包括首屏渲染时间、LCP、DNS 查询耗时、 TCP 连接耗时、请求响应耗时、内容传输耗时、DOM 解析耗时、资源加载耗时 、SSL 建连耗时等页面性能指标。
日志查询	包括历史日志、页面访问日志和自定义事件日志查询，详情可参见 日志查询 。
异常分析	JS 错误、Promise 错误、Ajax 请求异常、JS 加载异常、接口返回码异常等异常情况。
页面访问	页面访问用于展示页面访问量情况（UV、PV ），并支持多维度分析页面访问情况。
API 监控	可监控 HTTP Code 成功率、Retcode 成功率、API 请求耗时等调用情况。 • HTTP Code 成功率：HTTP Code 是指 HTTP 协议的状态码，默认返回码100 – 30x是正常情况，其余为异常情况。Code 为 0 表示 HTTP 请求没有到达服务端或者服务端没有返回。HTTP Code 成功率 = (HTTP 请求返回100 – 30x 的数量) / (总 HTTP 请求的数量)。 • Retcode 成功率：Retcode 是服务侧定义的业务返回码，默认 Retcode 为 0 表示接口正常，否则为接口异常。Retcode 成功率 = (Retocde 成功个数) / (接口总个数)
静态资源	前端 HTML 页面中主要包含的静态资源有：JS 文件、CSS 文件和图片文件，若这些文件加载耗时较长、失败等，将直接对页面造成影响甚至瘫痪。
自定义测速	自定义上报，参见 接入指南 选择接入应用方式，进入 实例方法 说明文档 > 使用 reportTime 或 time、timeEnd 方式自定义埋点上报数据。
自定义事件	自定义上报，参见 接入指南 选择接入应用方式，进入 实例方法 说明文档 > 使用 reportEvent 方式自定义上报事件。

说明

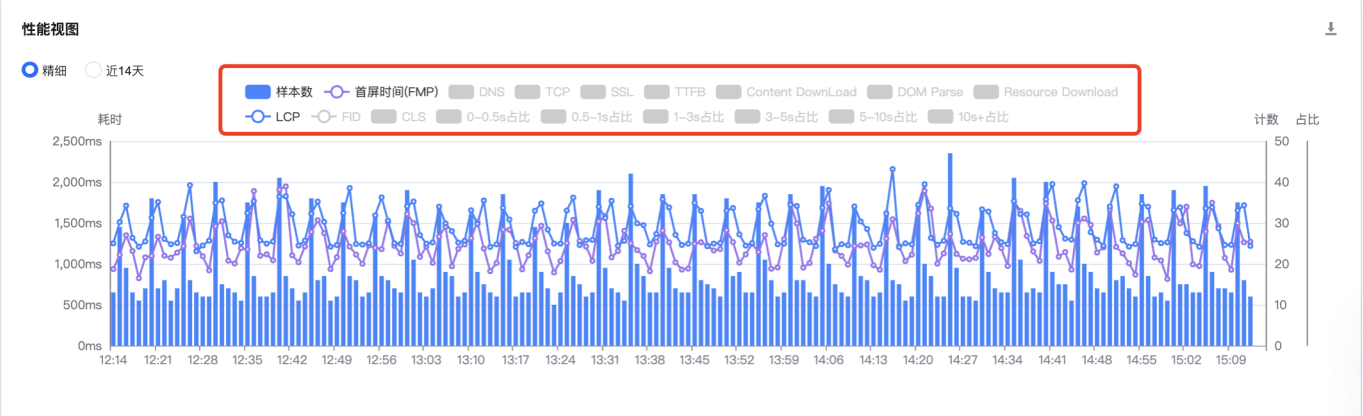
LCP：从用户请求网址到在可视窗口中渲染最大可见内容元素所需的时间。

操作说明

总览视图分析

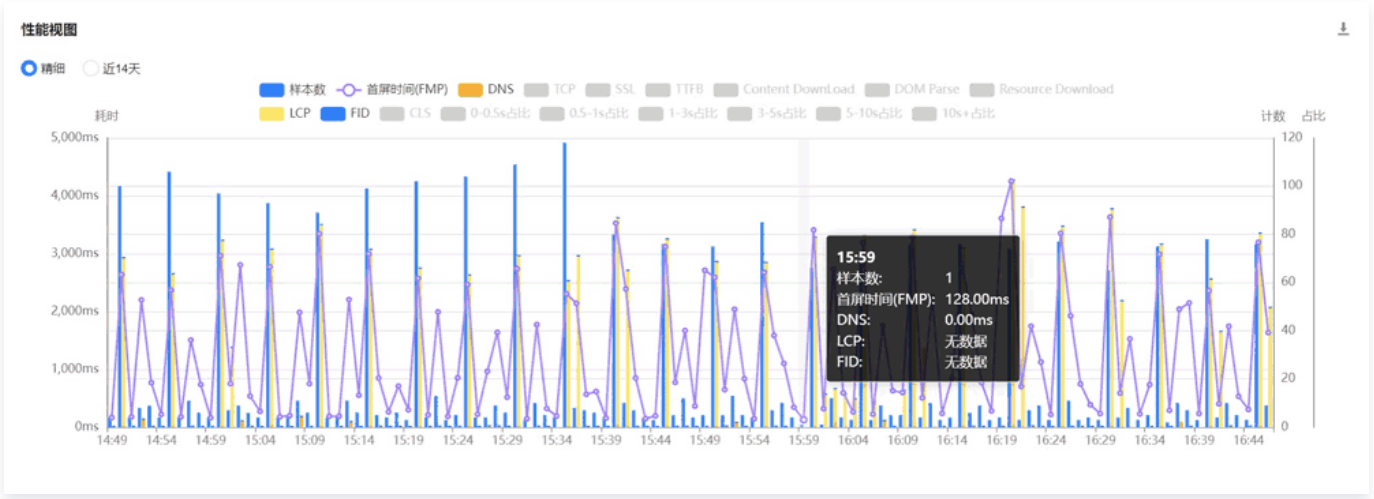
展示 页面性能 关键指标变化趋势图。

- 单击上方的图例可取消或展示对应的数据。



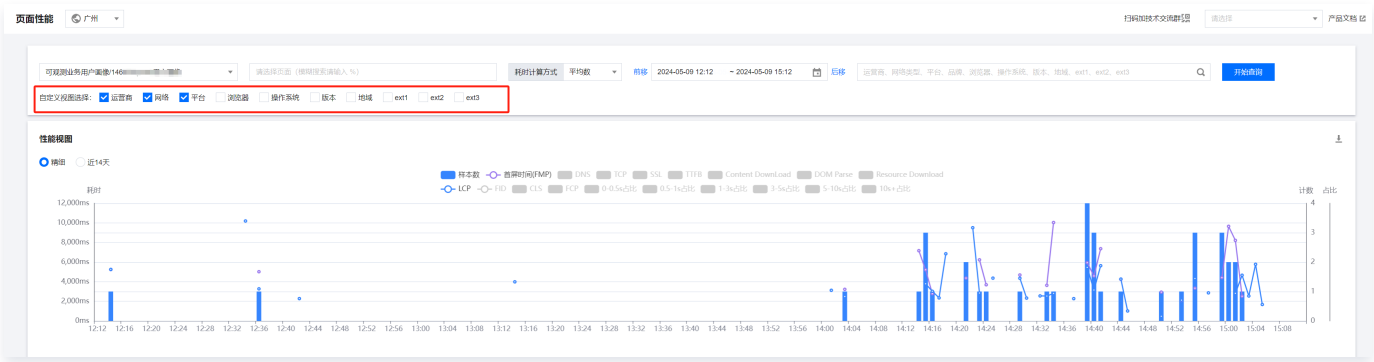
- 支持按时间段或近14天粒度展示变化趋势图。
- 在曲线中拖动鼠标可展示某一时刻首屏渲染时间。

- 把鼠标移动到图表中，并向上滑动鼠标可缩小图表时间跨度，向下滑动鼠标可放大图表时间跨度。

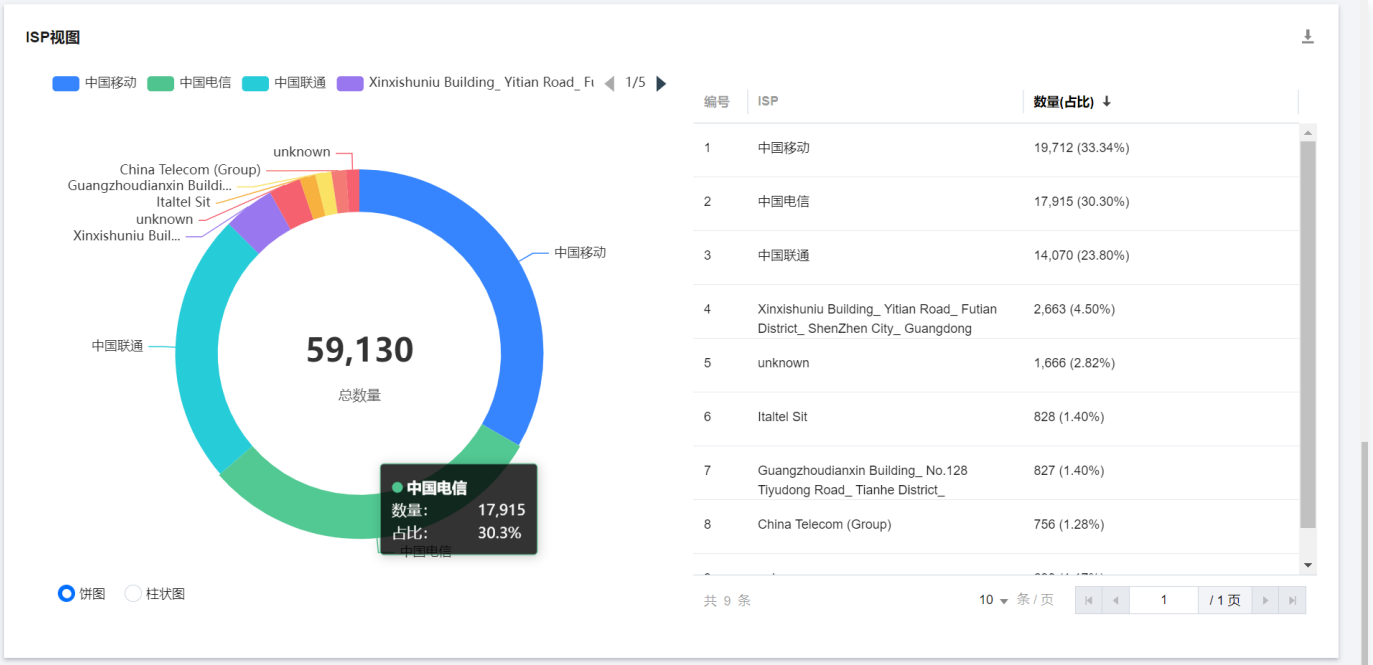


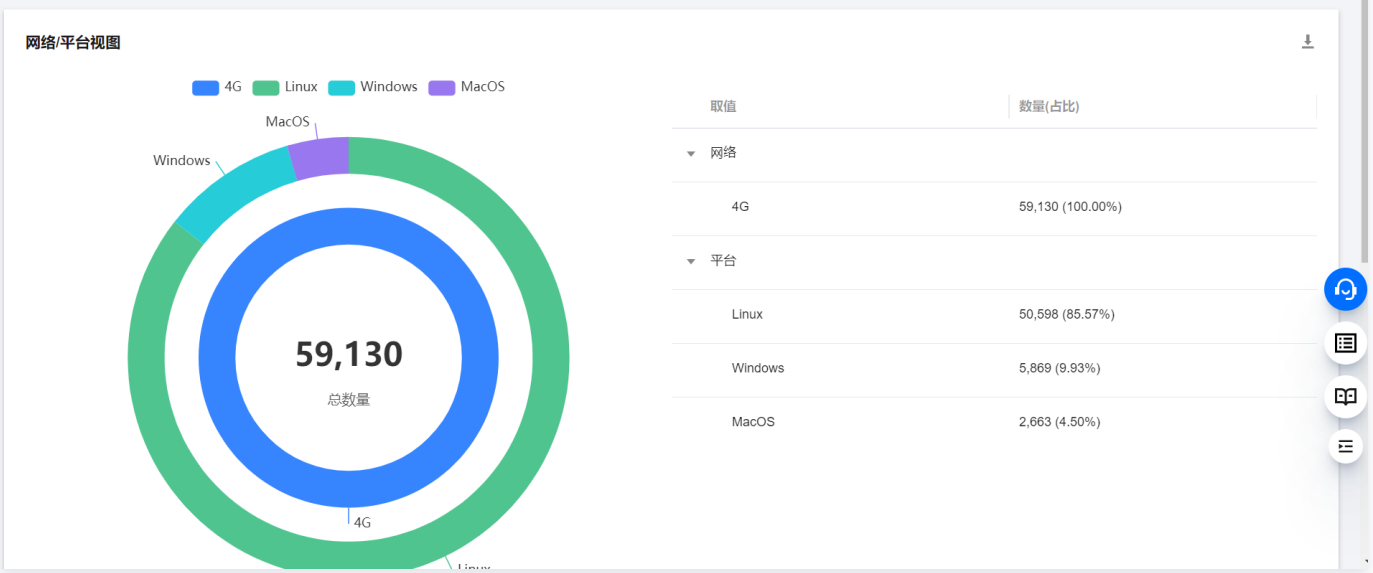
自定义视图选择

- 您可以在页面顶部，勾选需要展示的视图。



- 您还可以勾选后页面底部将会根据各维度展示前端性能数据情况，方便您从各维度分析前端性能数据。





其它视图

视图名称	说明
网络/平台视图	用饼图形式展示来自各区域的异常数量、占比及首屏时间 。其中网络包括3G、4G、WIFI 等，平台包括 Macos、Windows、IOS 等
ISP 视图	用饼图形式展示来自各运营商的异常数量、占比及首屏时间。运营商包括移动、电信、联通等。
地区视图	用饼图形式展示来自各区域的异常数量、占比及首屏时间。
品牌/机型视图	用饼图形式展示来自手机品牌/机型的异常数量、占比及首屏时间。
浏览器视图	用饼图形式展示来自各浏览器的异常数量、占比及首屏时间。
Version 视图	用饼图形式展示来自各应用版本的异常数量、占比及首屏时间。您可以通过在接入应用时的 new Aegis 传入 version 来自定义研发相关的版本信息，默认使用 SDK 的版本。
Ext1 视图、Ext2 视图、Ext3 视图	自定义视图，由您自定义上报时传入参数。详情请参见 接入指南 。

日志查询

最近更新时间：2024-08-09 14:51:01

前端性能监控提供查询与展示接口请求日志、JS 错误日志、Promise 错误、JS 加载异常等。本文将为您介绍如何使用日志查询。

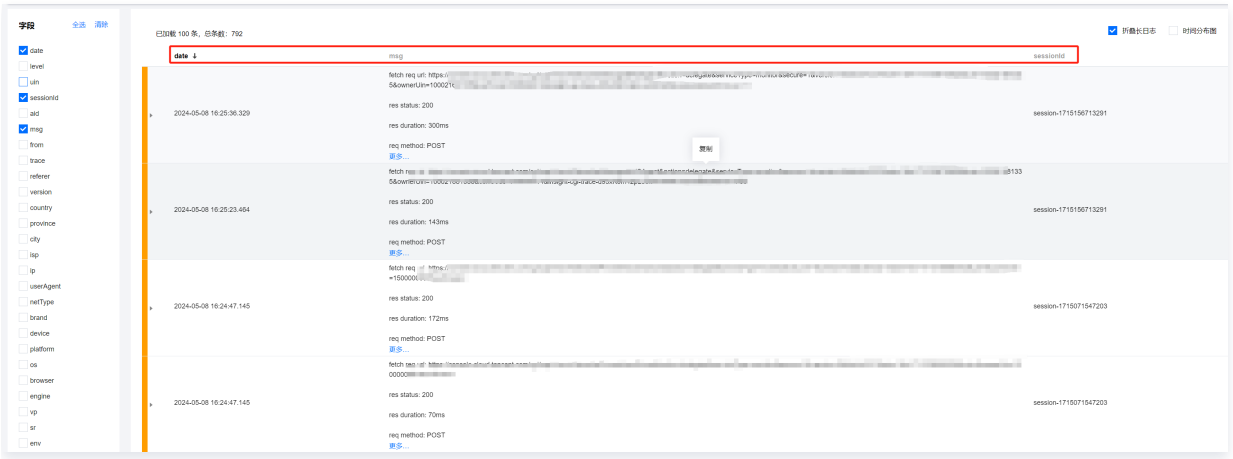
操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击**前端性能监控 > 日志查询**。
3. 进入日志查询页面。

操作说明

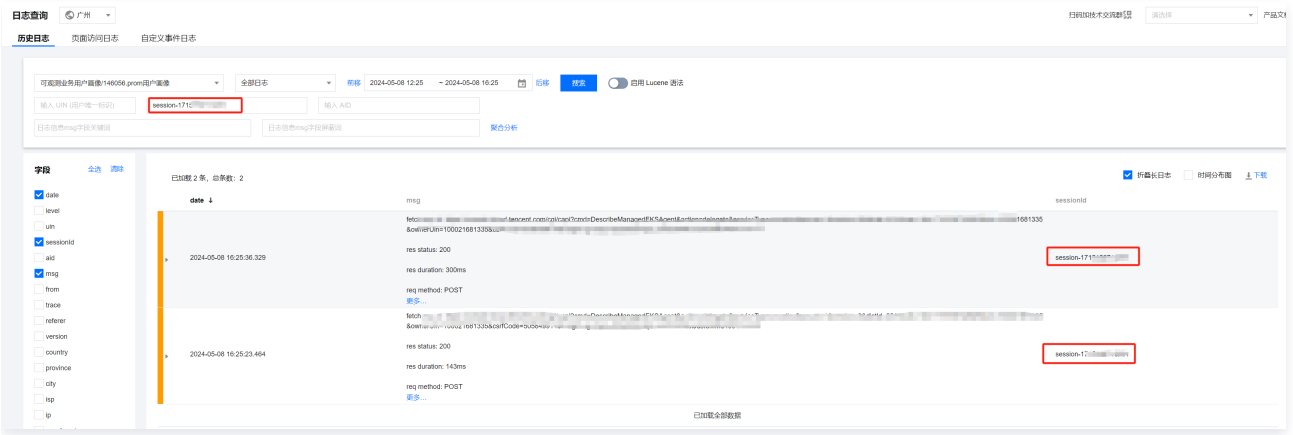
字段

勾选日志列表需要展示的字段，其中字段 **date** 表示服务端接收的时间。



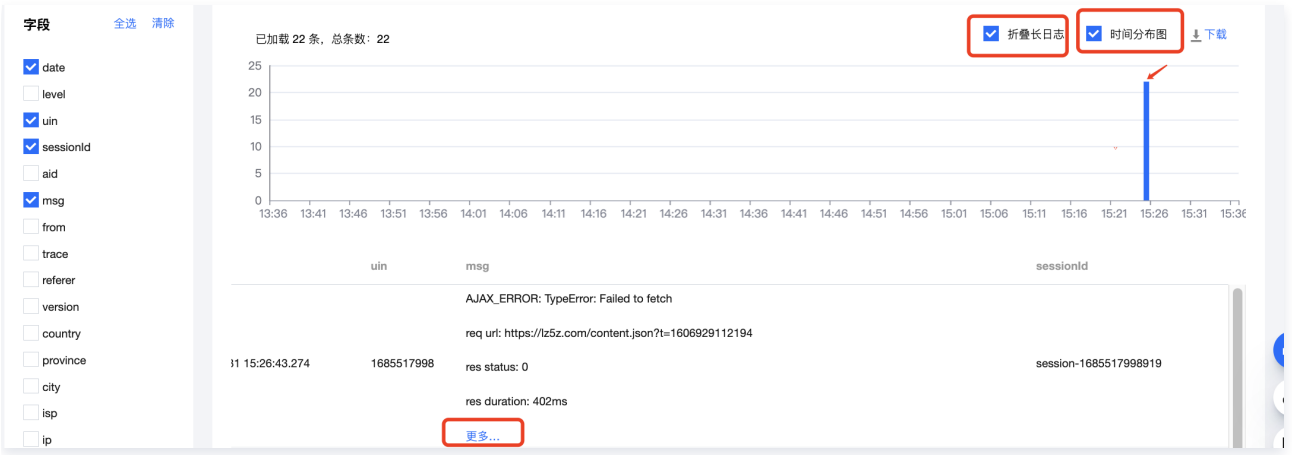
高级搜索

- 支持筛选应用、日志类型、时间等筛选。
- 支持根据用户唯一标识、sessionid、AID、关键词和屏蔽词进行日志筛选。
- 启用 Lucene 语法功能：定义是否开启全文检索功能。



日志列表

- 折叠长日志：不折叠一行内将展示所有日志信息，折叠后最多展示8行长日志信息。
- 时间分布图：是否展示某段时间内日志数量变化趋势图。


字段
全选
清除
date
level
uin
sessionId
aid
msg
from
trace
referer
version
country
province
city
isp
ip
已加载 22 条，总条数：22
折叠长日志
时间分布图
下载
25
20
15
10
5
0
13:36
13:41
13:46
13:51
13:56
14:01
14:06
14:11
14:16
14:21
14:26
14:31
14:36
14:41
14:46
14:51
14:56
15:01
15:06
15:11
15:16
15:21
15:26
15:31
uin
msg
sessionId
AJAX_ERROR: TypeError: Failed to fetch
req url: https://lz5z.com/content.json?t=1606929112194
15:26:43.274
1685517998
res status: 0
session-1685517998919
res duration: 402ms
更多...

应用管理
应用设置

最近更新时间：2024-07-10 17:56:21

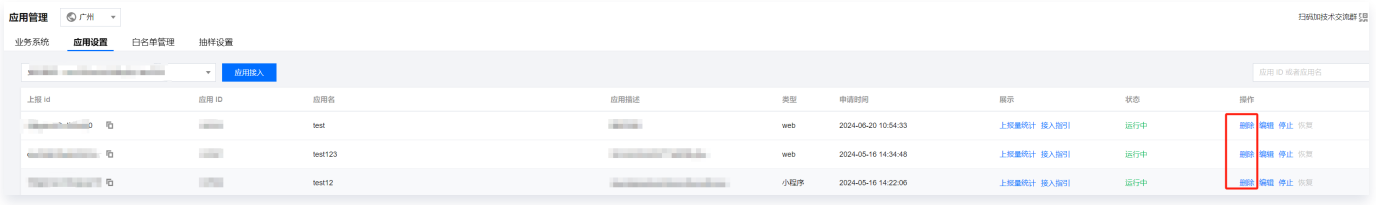
应用设置页面用于管理页面接入应用，包括创建、删除和编辑应用等。您还可以在应用设置页面应用上报量变化趋势和上报请求总量，还可以针对应用系统设置抽样率，避免上报量过高产生不必要的上报流量费用。本文将为您介绍如何使用应用管理功能。

操作步骤

> **说明：**
> 如需接入应用请参见 应用接入 。

删除应用

1. 登录 腾讯云可观测平台 。
2. 单击左侧菜单栏的**前端性能监控** > **应用管理** > **应用设置**，进入应用设置页面。
3. 找到对应的应用，在操作列单击**删除**，在弹框中确认删除即可。

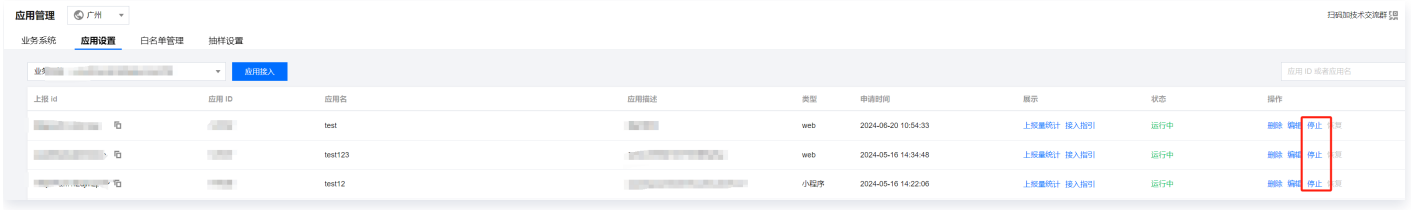


编辑应用

1. 登录 腾讯云可观测平台 。
2. 单击左侧菜单栏的 **前端性能监控** > **应用管理** > **应用设置**，进入应用设置页面。
3. 在操作列单击**编辑**，在弹框中编辑相关信息，并单击**提交**即可。

停止和恢复上报

在应用设置列表中，您可以单击操作列中的**停止**或**恢复**即可停止或恢复应用上报数据。



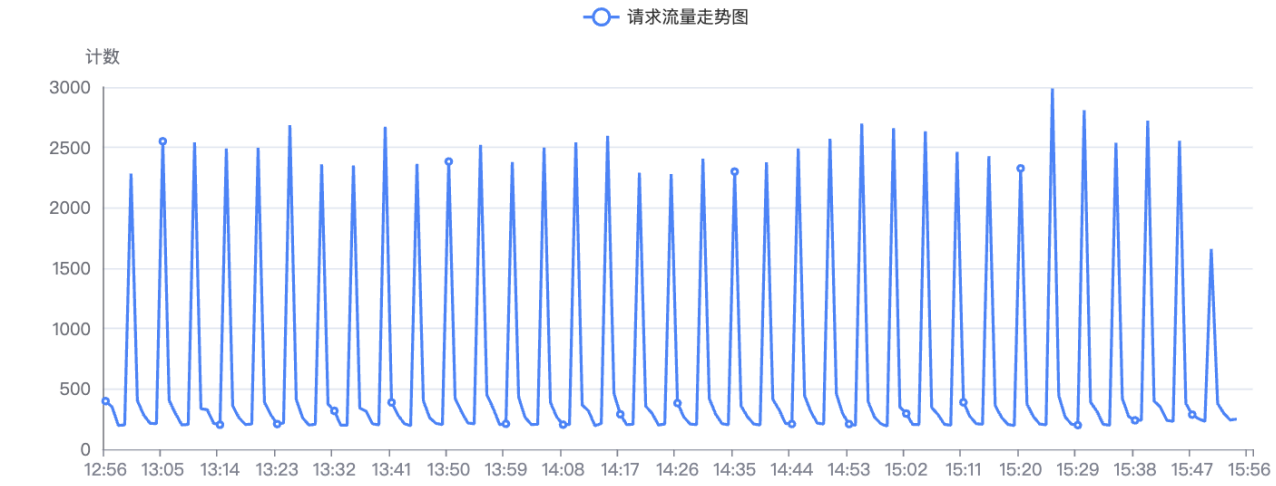
上报数据量统计

在应用设置列表中，您可以单击操作列中的**上报量统计使用**，查看各应用上报量变化趋势和上报请求总量。当数据量过大时可做相应的处理，避免上报量过高产生不必要的上报流量费用。

120000.云demo 上报量统计

全部 前移 2023-05-31 12:56 ~ 2023-05-31 15:56 后移

数据汇总-请求量: 126,787



业务系统

最近更新时间：2024-11-06 21:01:41

业务系统用于分组管理您接入的应用。本文将为您介绍如何使用业务系统功能。

> **说明：**
> 目前已支持中国大陆、新加坡以及硅谷三个地域，应用的页面性能等数据可能会因地域不同而受到影响，境外的应用建议选择境外地域进行上报。

操作步骤

创建业务系统

1. 登录 腾讯云可观测平台 。
2. 单击左侧菜单栏的**前端性能监控** > **应用管理** > 业务系统。
3. 进入业务系统页面，单击**创建业务系统**。
4. 在弹框中填写业务名称和勾选协议即可。

编辑业务系统

在操作列中单击**编辑**，即可编辑业务系统的名称以及描述。

标签

用于管理控制子账号用户是否能访问前端性能监控所接入的业务系统。详细操作请参见 资源标签 。

白名单管理

最近更新时间：2025-02-05 11:27:02

白名单管理适用于开发者针对特定的用户上报更多的信息，例如：上报更多日志信息、API 信息等。本文将为您介绍如何使用白名单管理功能。白名单详细逻辑可参见 白名单说明 。

操作步骤

添加白名单

1. 登录 前端性能监控 控制台。
2. 在左侧导航栏选择**应用管理**，然后在应用管理页面选择**白名单管理**，进入白名单管理页面。
3. 单击**添加白名单**，在弹框中填写用户 **uin** 和**备注**，然后单击**提交**。

删除白名单

在白名单列表中找到对应用户的 **uin**，在操作列中单击**删除**，并在弹框中确认删除即可。

抽样设置

最近更新时间：2024-07-03 16:59:42

抽样设置用于客户根据自身需求设置各指标是否上报、上报率或每分钟上报量，从而降低无效成本。例如：日志信息、页面测速等。本文将为您介绍如何使用抽样设置功能。

操作步骤

1. **登录** 腾讯云可观测平台 **。**
2. **单击左侧菜单栏的前端性能监控 > 应用管理 > 抽样设置，进入抽样设置页面。**
3. **选择需要修改上报的应用，根据自身需要在下方的信息中修改指标是否上报、上报率或每分钟上报量，确认之后点击下方的修改即可。**

访问管理

概述

最近更新时间：2024-11-05 21:01:41

如果您在腾讯云中使用到了前端性能监控，该服务由不同的人管理，但都共享您的云账号密钥，将存在以下问题：

- 您的密钥由多人共享，泄密风险高。
- 您无法限制其它人的访问权限，易产生误操作造成安全风险。

此时，您就可以通过子账号实现不同的人员管理不同的服务，来规避以上的问题。默认情况下，子账号无使用前端性能监控权限。因此，我们需要创建策略来允许子账号使用他们所需要资源的权限。

简介

访问管理（Cloud Access Management，CAM）是腾讯云提供的一套 Web 服务，它主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。通过 CAM，您可以创建、管理和销毁用户（组），并通过身份管理和策略管理控制哪些人可以使用哪些腾讯云资源。
当您使用 CAM 时，可以将策略与一个用户或一组用户关联起来，策略能够授权或者拒绝用户使用指定资源完成指定任务。有关 CAM 策略的更多相关基本信息，请参见 策略语法 。有关 CAM 策略的更多相关使用信息，请参见 策略 。

授权方式

前端性能监控支持资源级授权和按标签授权两种方式。

- 资源级授权：您可以通过策略语法或默认策略给子账号单个资源的管理权限，详细请参见 策略语法 和 策略授予 。
- 按标签授权：您可以通过给资源标记标签，实现给子账号对应的标签下资源的管理权限，详细请参见 资源标签 。

若您无需对子账号进行前端性能监控相关资源的访问管理，您可以跳过此章节。跳过这些部分不会影响您对文档中其余部分的理解和使用。

策略语法

最近更新时间：2024-11-05 18:43:11

概述

访问策略可用于授予访问前端性能监控相关的权限。访问策略使用基于 JSON 的访问策略语言。您可以通过访问策略语言授权指定委托人（principal）对指定的前端性能监控资源执行指定的操作。
访问策略语言描述了策略的基本元素和用法，有关策略语言的说明可参见 CAM 策略管理 。

策略语法

CAM 策略：

```
{
        "version":"2.0",
        "statement":
        [
           {
              "effect":"effect",
              "action":["action"],
              "resource":["resource"],
               "condition": {"key":{"value"}}
           }
       ]
}
```

元素用法

- 版本 version 是必填项，目前仅允许值为"2.0"。
- 语句 statement 是用来描述一条或多条权限的详细信息。该元素包括 effect、action、resource，condition 等多个其他元素的权限或权限集合。一条策略有且仅有一个 statement 元素。
  ⅰ. 影响 effect 描述声明产生的结果是“允许”还是“显式拒绝”。包括 allow (允许)和 deny (显式拒绝)两种情况。该元素是必填项。
  ⅱ. 操作 action 用来描述允许或拒绝的操作。操作可以是 API（以 name 前缀描述）或者功能集（一组特定的 API，以 permid 前缀描述）。该元素是必填项。
  ⅲ. 资源 resource 描述授权的具体数据。资源是用六段式描述。每款产品的资源定义详情会有所区别。有关如何指定资源的信息，请参阅您编写的资源声明所对应的产品文档。该元素是必填项。
  ⅳ. 生效条件 condition 描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、IP 地址等信息。有些服务允许您在条件中指定其他值。该元素是非必填项。

指定效力

如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。同时，也可以显式拒绝（deny）对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也无法访问。下面是指定允许效力的示例：

```
"effect" : "allow"
```

指定操作

前端性能监控定义了可在策略中指定一类控制台的操作，指定的操作按照操作性质分为读取部分接口（apm:Describe*）和全部接口（apm:*）。
指定允许操作的示例如下：

```
"action": [
  "name/apm:Describe*"
```

```
]
```

指定资源

资源（resource）元素描述一个或多个操作对象，如前端性能监控资源等。所有资源均可采用下述的六段式描述方式。

```
qcs:project_id:service_type:region:account:resource
```

参数说明如下：

参数	描述	是否必选
qcs	是 qcloud service 的简称，表示是腾讯云的云服务	是
project_id	描述应用信息，仅为了兼容 CAM 早期逻辑，一般不填	否
service_type	产品简称，这里为 rum	是
region	描述地域信息	是
account	描述资源拥有者的主账号信息，即主账号的 ID，表示为 `uin/${OwnerUin}`，如 `uin/100000000001`	是
resource	描述具体资源详情，前缀为 instance	是

下面是前端性能监控的六段式示例：

```
"resource":["qcs::rum::uin/1250000000:Instance/rum-vpasY123"]
```

实际案例

基于资源 ID，分配指定资源的读写权限，主账号ID 为 1250000000。
示例：为子用户分配查询业务系统（ID：rum-vpasY123）权限。

```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action": [
                "rum:DescribeTawInstances"
            ],
            "resource": [
                "qcs::rum::uin/1250000000:Instance/rum-vpasY123"
            ]
        }
    ]
}
```

支持资源级授权的 API 列表

API 操作名称	API 描述
DescribeData	获取 QueryData
DescribeError	获取首页错误信息
DescribeLogList	获取 CLS 日志列表

DescribeProjects	获取应用列表
DescribePvList	获取 PV 列表
DescribeScores	获取首页分数列表
DescribeTawAreas	查询片区信息
DescribeTawInstances	查询业务系统
DescribeUvList	获取 UV 列表
DescribeWhitelists	获取白名单列表
CreateProject	创建应用
CreateStarProject	添加星标应用
CreateTawInstance	创建业务系统
CreateWhitelist	创建白名单

策略授予

最近更新时间：2024-11-06 21:01:42

子账号默认没有前端性能监控任何权限。需要主账号授予子账号相关权限，子账号才能正常访问前端性能监控资源。

操作前提

使用主账号或拥有 QcloudCamFullAccess 权限的子账号登录腾讯云控制台，并参考 新建子用户 操作步骤创建子账户。

自定义策略

1. 使用主账号或拥有 QcloudCamFullAccess 权限的子账号进入**访问控制** > **策略**。
2. 单击**新建自定义策略** > **按策略语法创建**，选择空白模板。根据 策略语法说明 完成策略编辑。

策略授权

说明：
前端性能监控为您创建默认策略 QcloudRUMFullAccess（前端性能监控（RUM）全读写访问权限）和 QcloudRUMReadOnlyAccess（前端性能监控（RUM）只读访问权限），您可以通过搜索策略名称快速进行默认策略授权。也可以对自定义策略进行授权。授权成功后，子账号才能正常访问相关资源。

1. 使用主账号或拥有 QcloudCamFullAccess 权限的子账号进入**访问控制** > **策略**。
2. 进入策略管理页，在策略名称搜索框中输入对应的策略名称。

3. 选择只读访问或全读写访问权限，在操作列中单击**关联用户/组**。

4. 在弹框中勾选对应的用户，单击**确定**即可。

资源标签

最近更新时间：2024-05-16 17:14:01

前端性能监控结合腾讯云资源标签功能，为您提供按标签授予子账号权限和按标签分账功能。
资源标签是腾讯云提供的管理资源工具。资源标签分为标签键和标签值，一个标签键可对应多个标签值，您可以参见以下步骤进行按标签授权和账单分账。

使用场景

某公司有多个业务系统接入了前端性能监控。这些系统分别由 A、B 两个部门独立研发、运营。现需要对 A、B 部门创建标签、绑定资源并授予权限，说明如下：

- 创建 A 标签：绑定 A 部门所有业务系统。
- 创建 B 标签：绑定 B 部门所有业务系统。

按标签授权

用户 A 为 A 部门开发人员，负责 A 部门所有业务系统开发。需要授予该开发人员 A 标签权限。

按标签分账

用户 B 为公司的财务人员，负责对 A、B 部门财务支出进行独立核算。需要授予该财务人员 A、B 标签权限，并按标签进行分账核算。

准备工作

步骤1：创建标签

参见下列步骤，分布创建 A、B 标签。

1. 进入 标签列表页 。
2. 单击**新建标签**，进入添加标签页面，填写标签键和对应的标签值。

3. 单击**确定**，完成标签创建。

步骤2：为资源分配标签

参见下列步骤，为 A 标签绑定A部门下的所有业务系统，为 B 标签绑定 B 部门下的所有业务系统。

1. 进入 前端性能监控 > 应用管理 > 业务系统 。

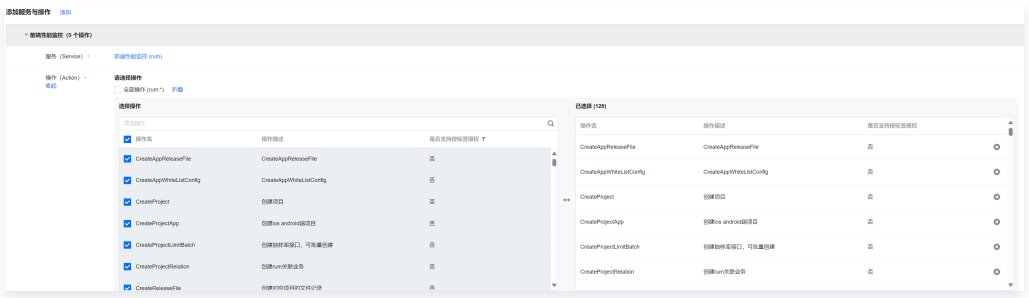
2. 单击**创建业务系统**，在弹框中填写信息并绑定标签。也可以在列表中找到已创建的业务系统，在操作列单击**编辑标签**，选择对应的标签。

按标签授权

按标签授权的策略，根据下列步骤给用户 A 授予 A 标签权限，用户 B 授予 A、B 标签权限。

1. 进入 策略管理页，单击左上角的**新建自定义策略**。
2. 在弹出的选择创建方式窗口中，单击**按标签授权**，进入按标签授权页面。

3. 在编辑策略页面填写以下信息后单击**下一步**。



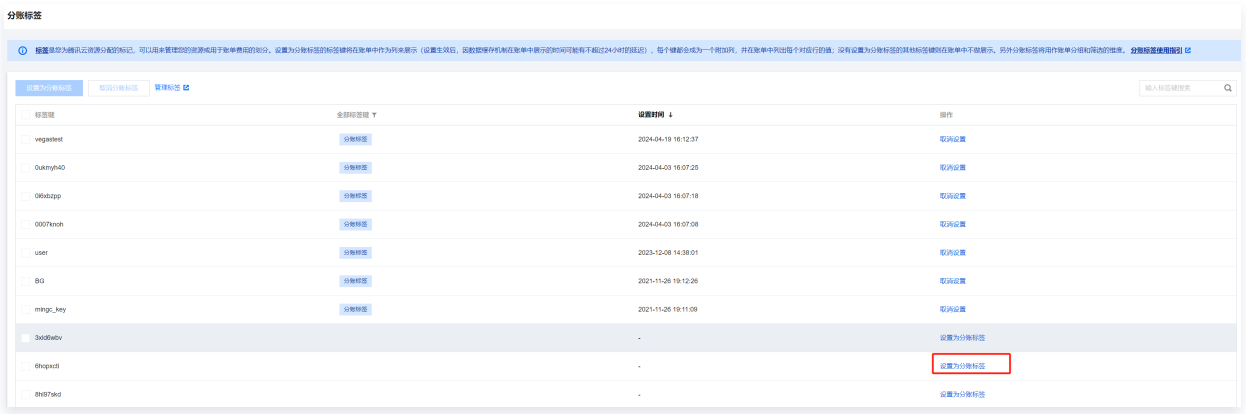
4. 进入关联用户/用户组/角色页面，选择需要关联的用户/用户组/角色，检查并完成策略（可修改策略名称）并单击**完成**。

按标签分账

步骤1：设置分账标签

根据下列步骤设置 A、B 标签为分账标签：

1. 若要在账单中使用标签功能，您需要进入 费用中心控制台，选择在左侧菜单**分账管理 > 分账标签**。被设置为分账标签的标签键会作为账单的单独一列展示，您可根据此标签键来对账单进行筛选和分类展示。
2. 在此页面您可看到已创建的标签键列表，选择需要展示的标签键，单击**设置为分账标签**，即可将该标签键设置为账单中的分账标签。



步骤2：按标签展示账单

您可在 账单概览 页，查看并单击新的选项**按标签汇总**，通过选择具体的**标签键**，可查看根据该标签键汇总的相关资源的柱状图和列表。

告警策略
新建告警

最近更新时间：2024-11-06 15:44:01

本文将为您介绍如何为前端性能关键指标设置告警，在指标发生异常时及时通知您。
RUM 告警功能使用腾讯云可观测平台告警实现，更多文档可以参考 告警概述 。

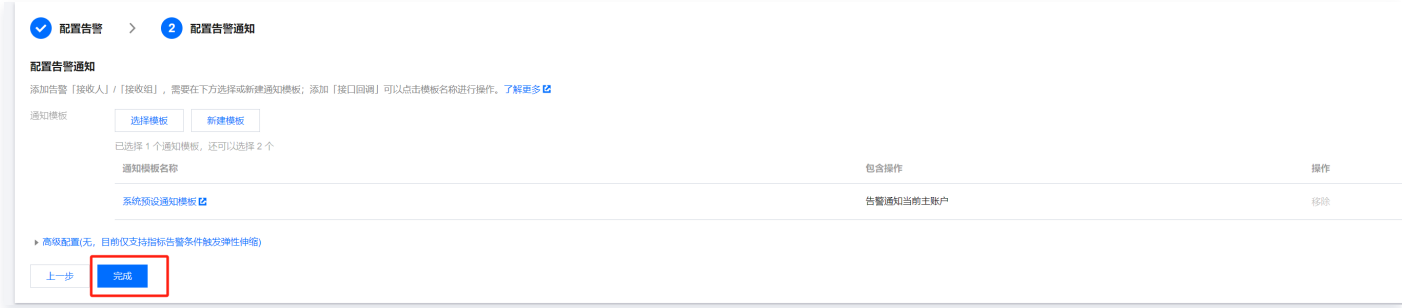
> **注意：**
> 暂不支持新加坡地区的日志告警，目前正在开发中，敬请期待。

操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击**告警管理 >策略管理**，进入策略管理页面。
3. 单击**新建策略**，配置告警策略，配置说明如下：

配置类型	配置项	说明
基本信息	策略名称	自定义策略名称。
	备注	自定义策略备注。
配置告警规则	监控类型	选择**前端性能监控**类型。
	策略类型	支持**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**策略类型 。
	筛选条件（与）	筛选出符合条件的对象进行告警检测，各筛选条件之间为 AND 关系。筛选条件仅展示有上报数据的对象。您可以根据需求选择需要检测的告警对象。
	告警对象维度	支持自定义告警通知内容中的告警对象，假设您选择了业务系统、应用和日志类型，则告警对象显示业务系统=xxx，应用=xxx，日志类型=xxx。
	触发条件	支持满足任意条件或满足所有条件。
配置告警通知	通知模板	系统为您默认配置通知模板，如需创建通知模板请参见 新建通知模板 。
高级配置	弹性伸缩	启用并配置成功后，达到告警条件可触发弹性伸缩策略并进行缩容或扩容。

4. 配置完以上信息后单击**完成**，即成功创建告警策略。在指标发生异常时，将会通过您配置的告警渠道发送告警通知。



查看告警

最近更新时间：2024-05-22 09:23:51

本文将为您介绍如何查看前端性能监控告警历史。

查看告警历史

1. 登录 腾讯云可观测平台 。
2. 单击左侧栏**告警管理>告警历史**，进入 告警历史 页面。
3. **监控类型**选择前端性能监控，然后单击**确定**。
4. 您还可以单击左上角的**时间范围**和**筛选**，进行告警历史的类别筛选。

实践教程
Aegis SDK 支持获取请求头和返回头

最近更新时间：2024-09-11 21:53:52

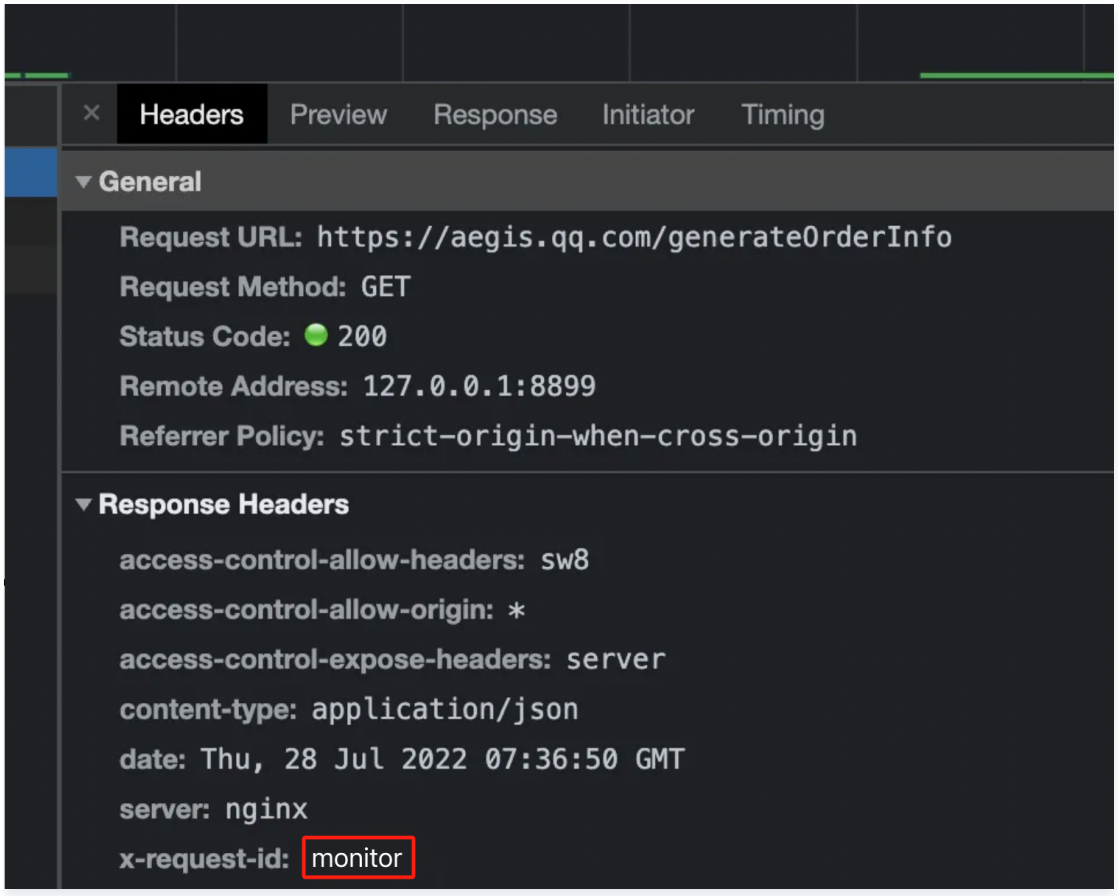
通过记录自定义请求头和响应头的方式，打通前后端链路。

> **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 Header `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：

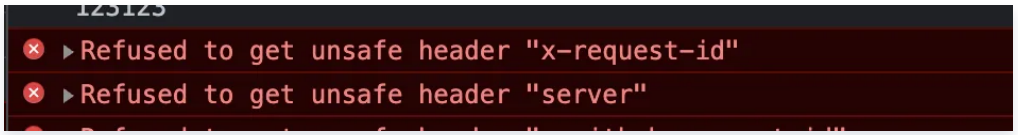
1. 将 SDK 升级到最新版本。
2. 修改初始化 SDK 的方式。

```
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。

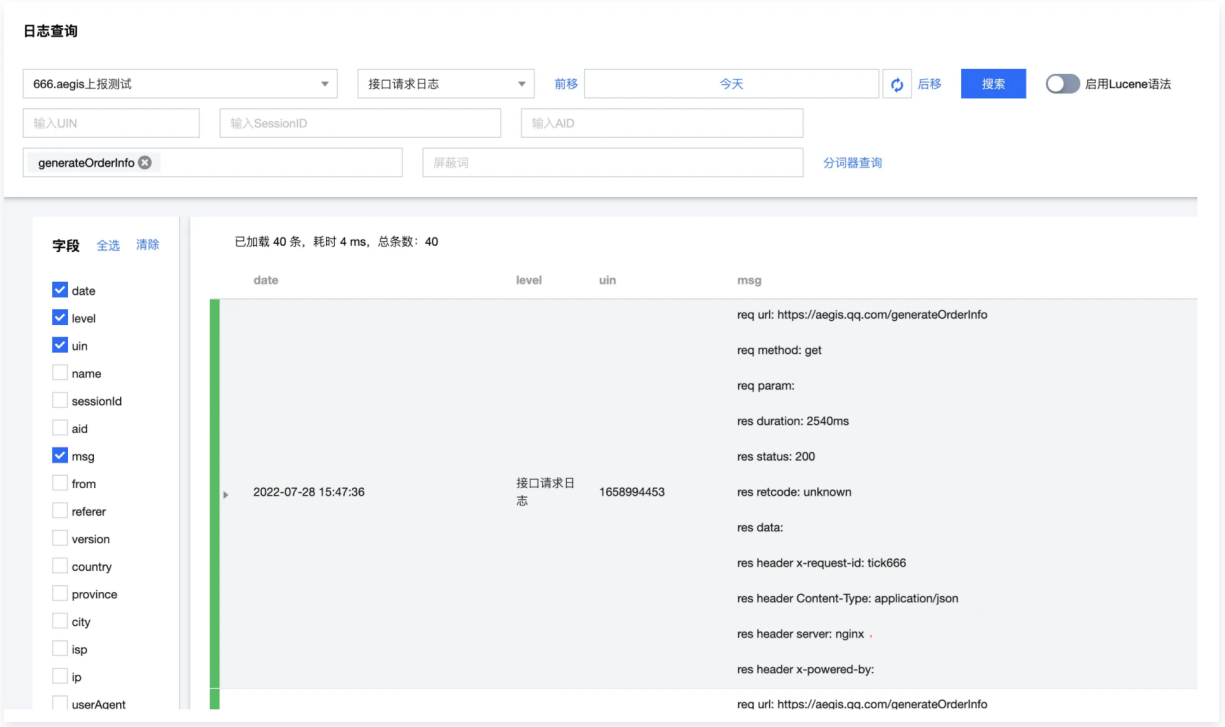


> **说明：**
> 具体报错原因可参见 相关文档 。

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：



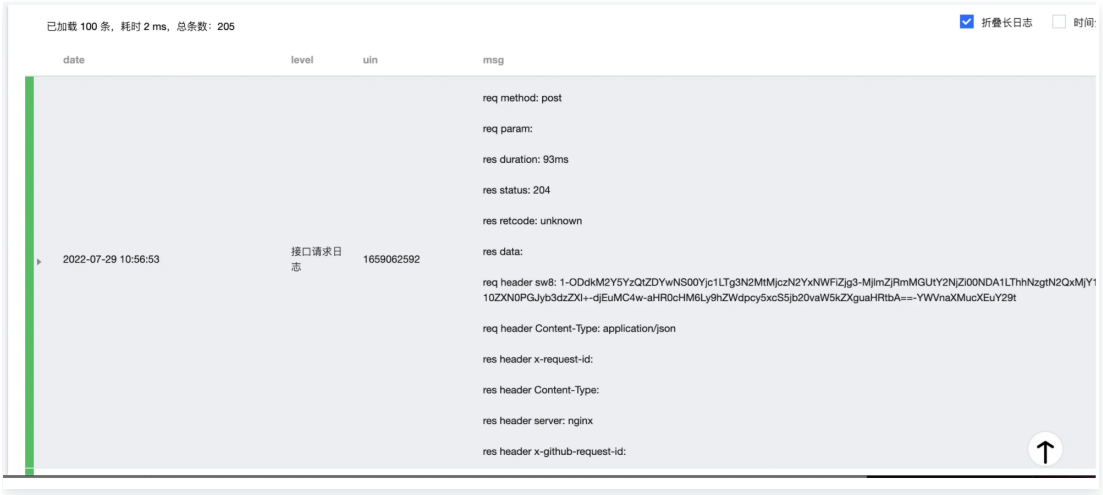
获取 Request Headers

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({
```

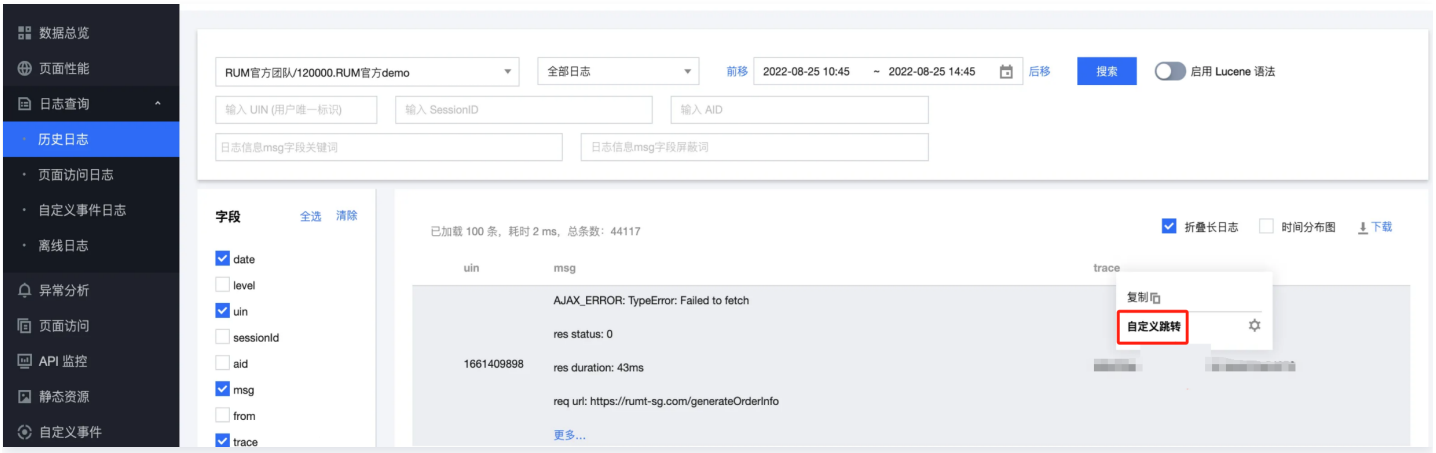
```
        id: 'xxx',
        reportApiSpeed: true,
        api: {
            apiDetail: true, // 上报接口请求参数和返回值
            reportRequest: true, // 全量接口上报
            reqHeaders: ['sw8'], // 记录 request header
        },
    });
```

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 14:48:11

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。如下为您介绍如何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阅。

操作步骤

接入应用

步骤1：创建业务系统

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏单击**前端性能监控 > 应用管理 > 业务系统**。
3. 在业务系统页单击**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：参见下列步骤新建一个 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'
});
```

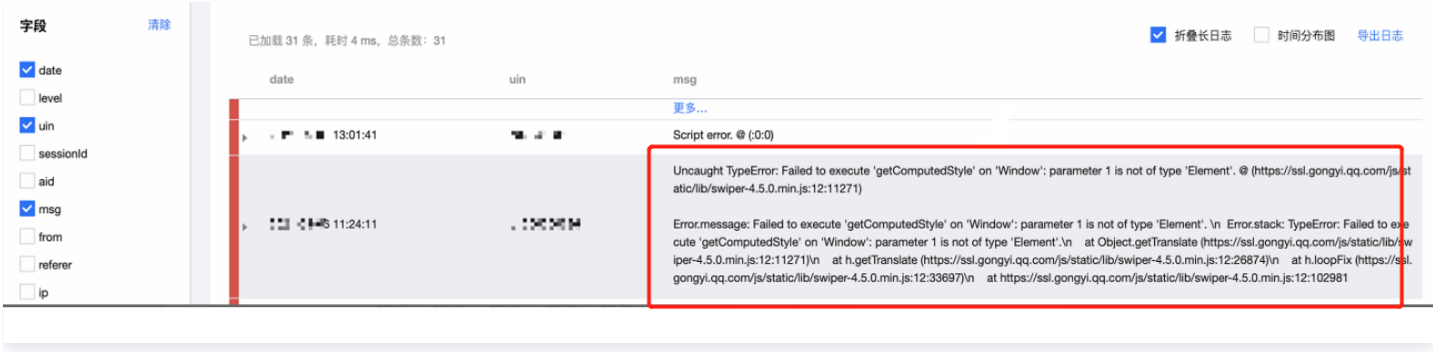
步骤4：验证是否接入成功

进入数据总览，若有数据则证明接入成功。

小程序优化

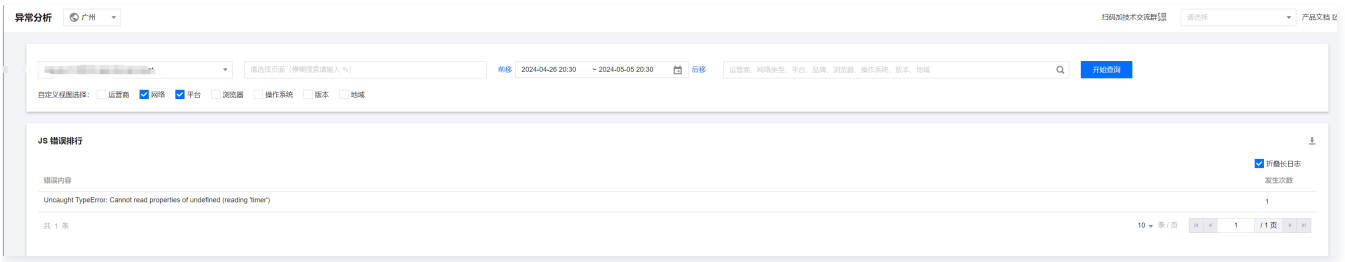
白屏问题处理

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。

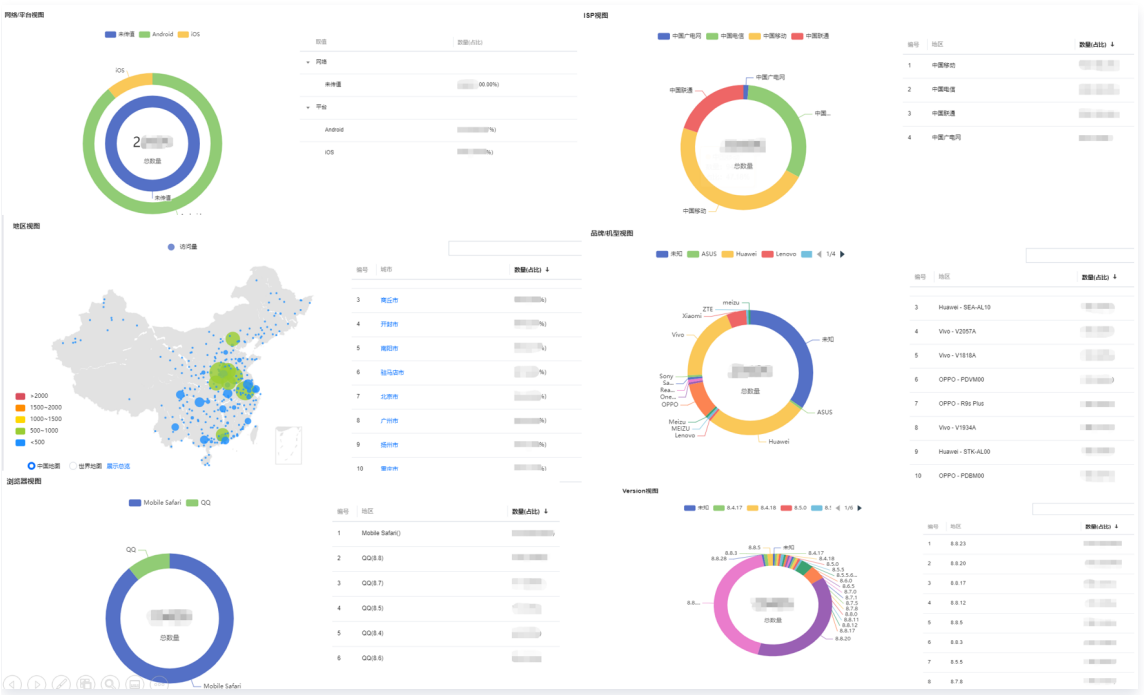


JS/ Ajax 等问题处理

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，更加精准的知道异常分布情况（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。

如何根据性能数据优化应用

最近更新时间：2024-05-08 17:33:01

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口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（单击链接、单击按钮等）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。

- LCP（Largest Contentful Paint – 最大内容绘制）：LCP 度量从用户请求网址到在视口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（单击链接、单击按钮等）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。

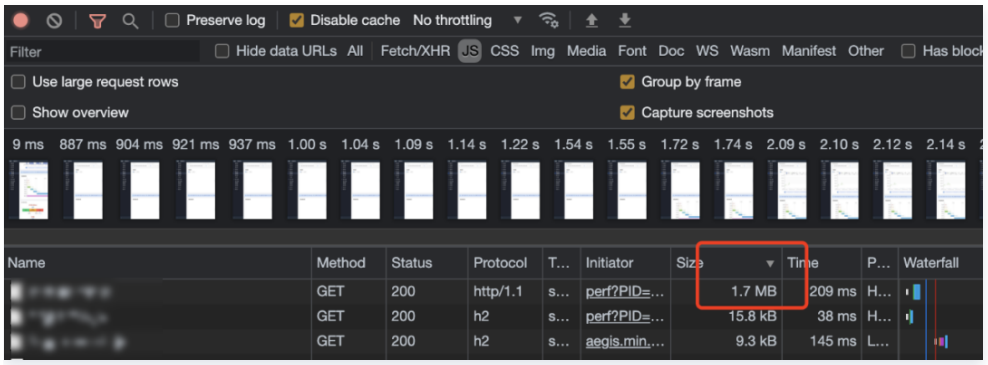
按网络和区域分析性能数据

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。



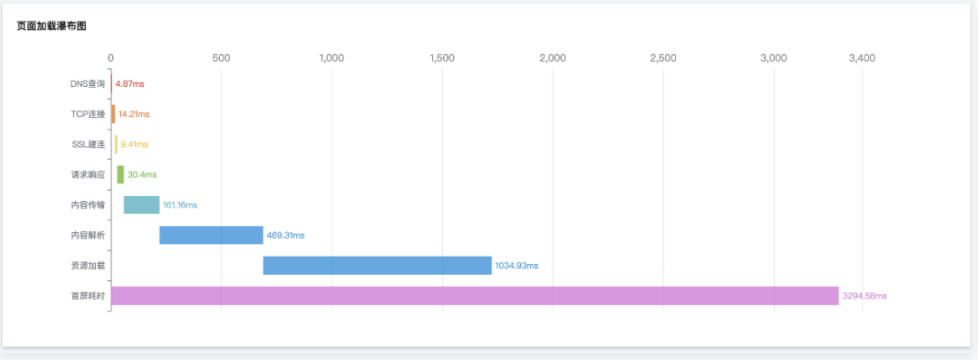
再进一步分析，该测试应用使用的是 React 框架，在没有服务端渲染的情况下，页面是会在加载主 JS 后才渲染的。而用户大部分 JS 文件都打包成一个 bundle ，导致产生了一个超大的 JS 文件，这个 JS 文件就成为了用户页面渲染的瓶颈。除此之外还发现了该 JS 文件没有支持 HTTP2 协议。那我们该如何优化?

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。

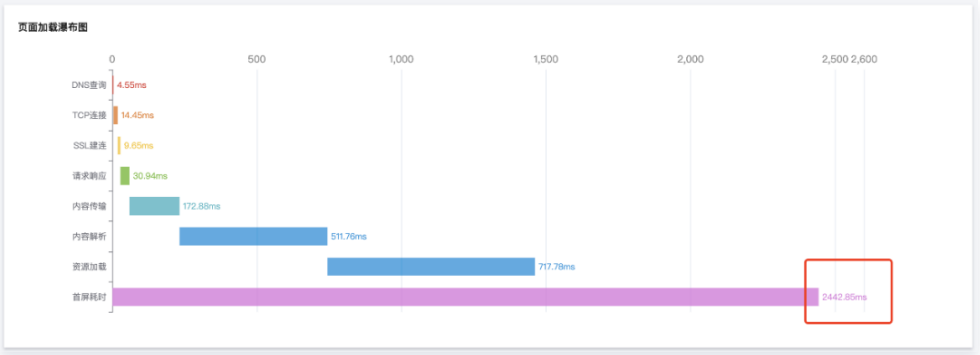
通过上述步骤优化后首屏耗时从 4.8s 优化到 3.2s，最重要的是 “资源加载” 耗时直接下降50%。



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。

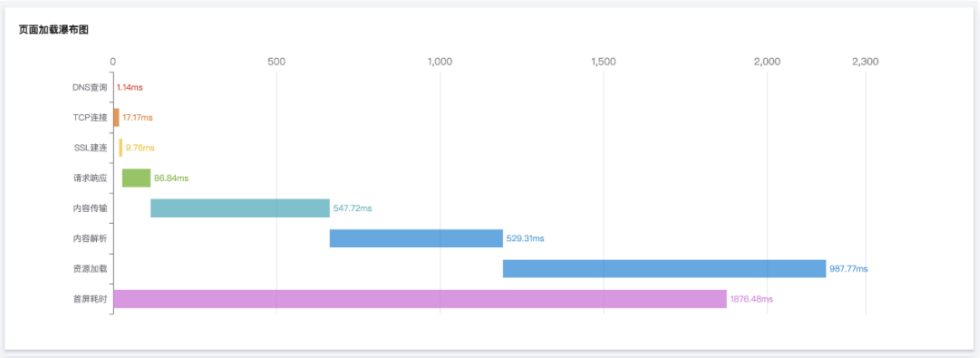
没有分页带来了列表无法在渲染之初就确定长度，导致获取数据后渲染列表的时候页面发生较大的偏移，同时也带来了超多的 DOM 变化。

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：2024-06-05 16:19:41

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：

步骤一：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面过调用 API 对账号以及账号空钥进行鉴权。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。

步骤二：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，建议使用
rum.internal.tencentcloudapi.com，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,
```

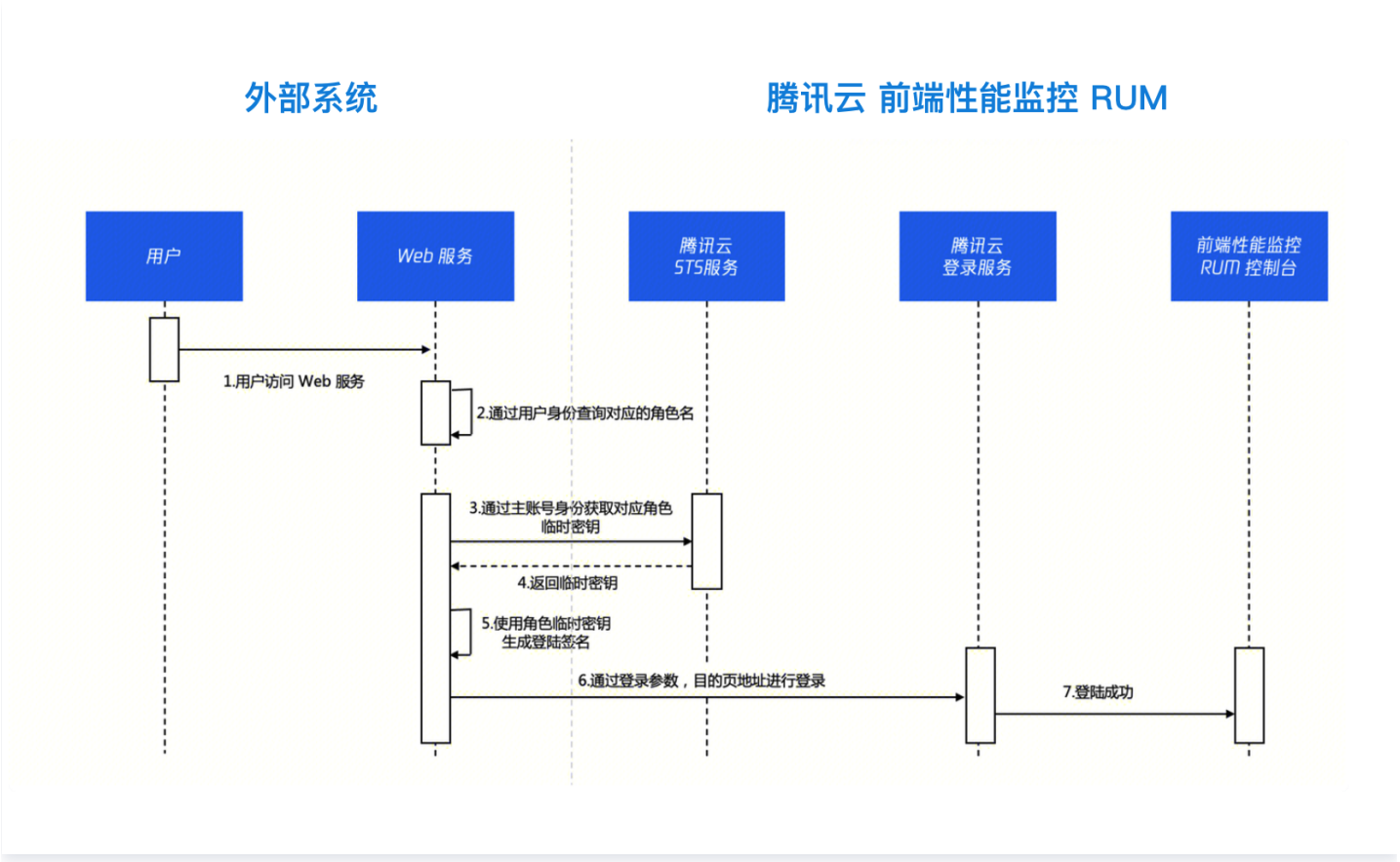
```
                    ExpiredTime : data.ExpiredTime,
                });
            },
            (err) => {
                console.error("error", err);
            }
        ).catch(err => {
            console.log(err);
        });
    },
});
const timestamp = +new Date();
const fileKey = `${projectID}-${version}-${timestamp}-${fileName}`;
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：2024-07-29 14:11:01

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

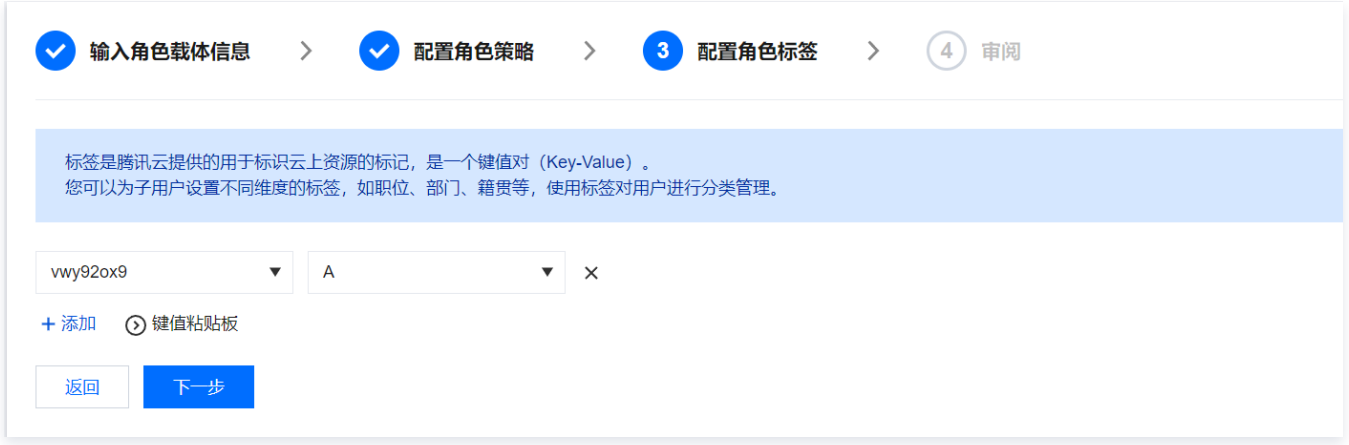
企业用户（运维或开发人员）根据业务需求申请对应的权限，用户身份对应腾讯云账号角色，可以通过 控制台 或 创建角色API 创建对应的角色：

通过控制台创建 CAM 角色

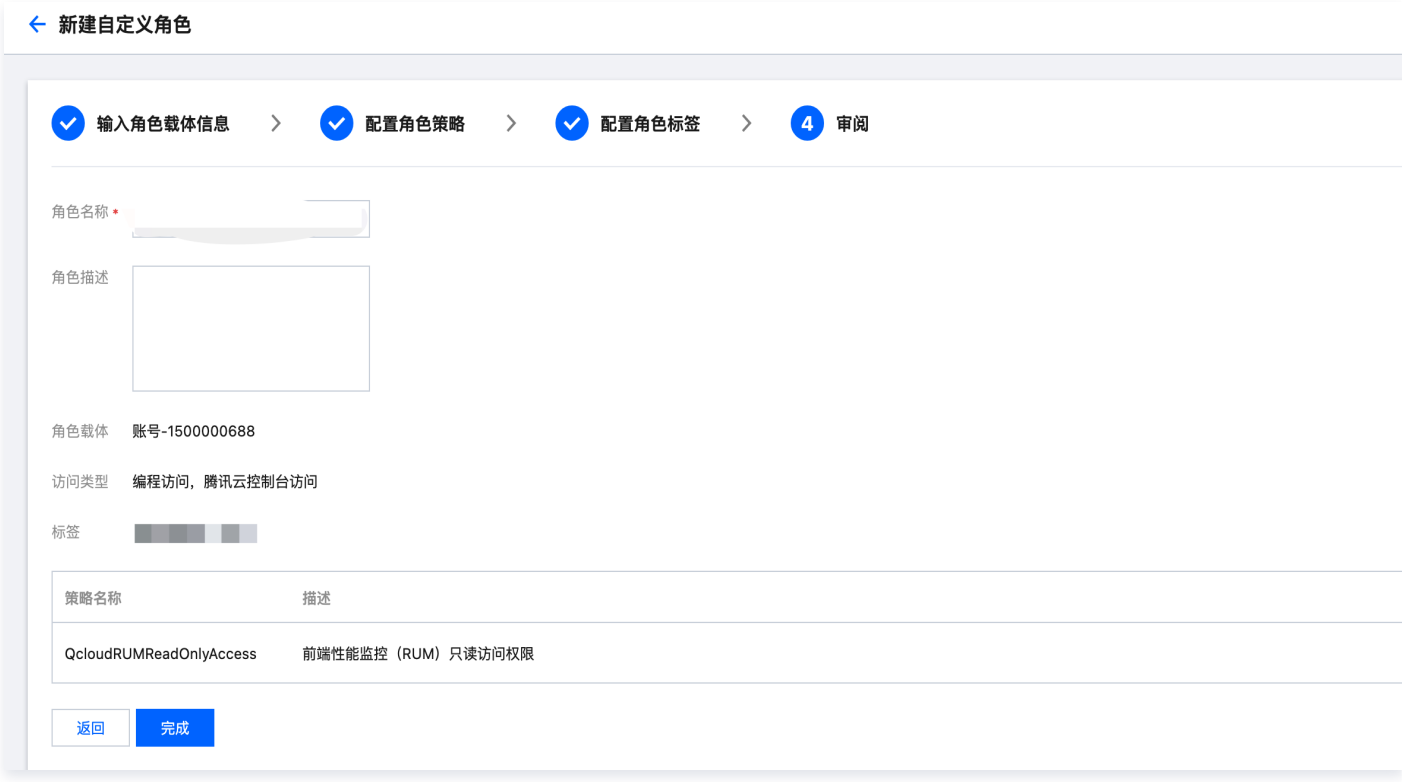
1. 登录 访问管理 CAM 控制台 。
2. 单击左侧菜单栏中的**角色**，进入角色页面。
3. 选择**新建角色 > 腾讯云账户**，开始新建自定义角色。
4. 选择**当前主账号**并勾选**允许当前角色服务控制台**，单击**下一步**。

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。



7. 输入角色名，完成创建。



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，依此类推。

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_52PCpauvYykeiSpVZ7w5g2qOvV1Azs&nonce=6743
9&timestamp=1484793352&signature=AJ***3D&s_url=https%3A//console.cloud.tencent.com/monitor/rum%3
FhideWidget%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)
```

常见问题
产品相关问题

最近更新时间：2024-10-18 18:41:21

RUM 是什么?

前端性能监控（Real User Monitoring，RUM）是一站式前端监控解决方案，专注于 Web 和小程序等大前端领域，主要关注用户页面性能（页面测速、接口测速、CDN 测速等）和质量（JS 错误、Ajax 错误等），并且联动腾讯云应用性能监控实现前后端一体化监控。用户只需要安装 SDK 到自己的项目中，通过简单配置化，即可实现对用户页面质量的全方位守护，真正做到了低成本使用和无侵入监控。可参考文档 快速入门 。

Aegis SDK 的作用是什么?

Aegis SDK 是嵌入到用户页面或者使用 npm 安装到用户代码中的上报 SDK，主要负责采集用户侧性能和质量数据。

RUM 和 Aegis SDK 有什么关系?

- RUM 是前端性能监控平台，Aegis 是前端监控 SDK，负责数据采集和上报。
- Aegis 是前端性能监控（RUM）提供的监控 SDK，开发者将其代码嵌入到页面中，即可实现对页面性能和质量的监控。
  - aegis-web-sdk：web页面监控 sdk
  - aegis-mp-sdk：小程序页面监控 sdk

RUM 采集了哪些用户信息?

RUM 采集了哪些用户信息可参考 前端性能监控 SDK 隐私保护协议 、前端性能监控服务协议 。

接入前端性能监控，用户的流量可能会很大，或者某个时间点要发活动，应该怎么做?

如果页面 QPS 超过 1w 或者上报 QPS 超过 2w，请提前联系 腾讯云助手 进行报备。

接入前端性能监控，对于境外用户使用和上报，性能是否有影响? CDN 和 上报是就近接入的吗?

CDN 目前在以下国家均有 OC 节点：英国、越南、日本、美国、新加坡、中国香港、韩国、泰国。上报数据的节点目前还没有在境外部署，会走中国香港的节点，然后把数据转存到国内。

> **说明：**
> - RUM 2022年6月正式上线新加坡节点，数据上报和存储节点都在新加坡。
> - RUM 2022年10月正式上线硅谷节点，数据上报和存储节点都在北美硅谷。

具体使用方式：
在 RUM 平台创建境外的业务系统，地域选择新加坡或者硅谷，然后在该业务系统下创建应用。应用接入的时候与国内上报域名不同，需要开发者改下 hostUrl 参数修改上报域名。具体上报域名如下：

- 国内可以选择选择使用 https://rumt-zh.com/ 作为上报域名。
- 新加坡地区可以选择 https://rumt-sg.com/ 作为上报域名。
- 硅谷地区可以选择 https://rumt-us.com/ 作为上报域名。

> ⚠ **注意：**
> 不同上报域名对应不同数据存储和服务器部署区域，所以不能混报，例如使用国内节点的 RUM 应用上报 ID，向新加坡节点上报数据，就无法正常上报。

RUM 支持私有化部署吗?

私有化方案目前还在开发中，敬请期待。

用户的日志会保存多久?

- 用户上报的原始日志，包括错误，自定义上报，页面访问，保留 30 天。
- 性能相关的指标数据，如页面性能，API 监控、静态资源监控等，保留 30 天。
- RUM 每天定时计算得出来的数据，例如：每天的应用评分，每天的 PV/UV 汇总数据等，保留 90 天。

前端性能监控目前支持了哪些平台? 提供了什么平台的 SDK?

前端性能监控目前支持 Web、小程序（微信、QQ）、Hippy、Weex、React Native、 Flutter 和 Cocos 等平台的数据上报。

RUM 的 uv 是否支持高维度数据，例如页面地址、ext、省市等?

aegis sdk 没有根据用户设置的 uin 来计算 uv，因为如果当前用户没有登录，则无法计算，而且 uin 值由开发者设置，并不完全可靠，还有无法将用户登录前后状态统一起来，因此 aegis sdk 为每个用户独立生成一个 aid 作为用户（设备）的唯一标识，并且存储在 localstorage 里面，不受登录状态影响。使用 aid 的缺点是如果用户清理了设备的 localstorage，aid 会重新生成一个，导致 uv 计算比实际高一点，但是清理设备 localstorage 是低频操作，因此我们认为可以接受。还有多个账号重复登录同一个设备的问题，按照正常的 uv 计算逻辑，应该算多个值的，但是我们只计算了一个值。aid 的计算算法如下：

```
async getAid(callback: Function) {
// 某些情况下操作 localStorage 会报错.
  try {
  let aid = await localStorage.getItem('AEGIS_ID');
  if (!aid) {
  aid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
    const r = (Math.random() * 16) | 0;
    const v = c === 'x' ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  });
  localStorage.setItem('AEGIS_ID', aid);
  }
  callback?.(aid || '');
  } catch (e) {
  callback?.('');
  }
}
```

RUM 目前 uv 数据暂时不支持高维数据，原因在于 uv 计算是一个比较消耗资源的事情，而且需要提前预计算好，例如一个项目有 N 个页面，就需要提前把每个页面的 uv 计算出来，但是考虑到很多项目的页面都是高维的，所以会导致我们计算 uv 的压力非常大。如果再加上其他维度，例如省市、ext，相当于计算这些所有维度组合的笛卡尔积个数的计算任务，压力就更大了。

为什么有一段时间的 pv 有数据，uv 却没数据?

uv 会按天去重。假如有一个用户当天的早些时间访问过您的项目，uv 统计过了，当天再次访问就不计算 uv 了。

RUM 告警支持哪些渠道? 如何设置?

此问题可参考 告警通知渠道 。

RUM 日报功能在哪里?

可以在 报表管理 中新增日报。

技术排查相关文档

最近更新时间：2025-06-09 17:12:22

代码中 `throw new Error` Aegis SDK 没有捕获到，如何处理?

在 VUE 框架中，Vue.config.errorHandler 会主动捕获错误，可以通过主动捕获再调用 aegis.error() 进行上报。

接入 SDK 后没有数据是怎么回事?

没有数据可以从5个方向查问题：

1. 查看上报接口（aegis.qq.com）返回是否正常，正常返回200和204，如果接口返回403可以看下一个问题。
2. 看日志查询里面页面访问和历史是否有数据，如果没有数据，有可能是上报延迟导致（1-2min）。
3. 数据总览里面的数据是每整数小时计算一次，如果没有数据可以看下是否上个小时没有页面访问，或者新接入还没开始计算。
4. 如果是页面性能没有数据，或者缺少数据，可以看下 network 中上报接口 performance 中的 firstScreenTiming 的值，如果这个值为0，或者大于15s，就会导致页面性能缺少数据。

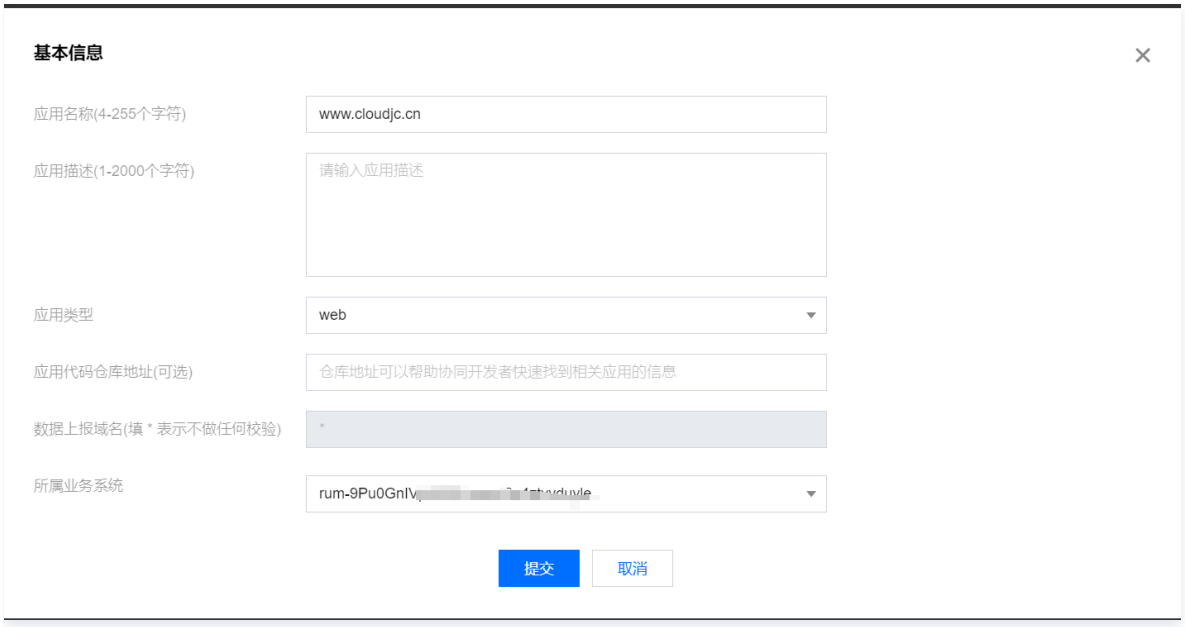
   解决办法：把 sdk 初始化尽可能放在代码最前面，可以考虑用 cdn 并且在 head 中初始化代码，尤其对于一些 SSR 的页面，因为页面首屏时间较小，初始化 sdk 过晚会导致无法采集到页面首屏信息。
5. 对于日志中没有数据，尤其刚刚接入的应用，因为 Aegis SDK 默认只采集错误日志，例如接口报错、js 错误、promise 错误等，所以刚刚接入会看不到日志。开发者可以主动上报一些日志（aegis.infoAll、aegis.report等），看是否有数据。如果有需要，也可以开启全量接口上报（api.reportRequest），查看上报的接口的信息。

前端监控里面统计到的接口耗时（静态资源耗时）跟后端统计的相差很远，可能是什么原因导致的?

1. 用户网络出问题，请求没有发生到后端，耗时比较久后失败了。
2. 用户请求发出了，但是中间网络层耗时比较久，后端处理很快，过了很久才返回给用户。
3. 用户看的是平均值，可能由于某个或者某几个用户出现比较高的异常值导致整体值偏低，用户可以看下中位数是否依然异常。
4. 用户请求接口（静态资源）受多方面因素影响，网络、地域、设备都会对最终值产生影响，cdn 没有预热，dns 第一次解析耗时比较久等原因都会导致前后端统计不一致，前端性能监控通过浏览器的 performance 对象获取这些耗时并且进行上报。
5. 前端性能监控只是把这个耗时记录下来，并不对这个耗时负责。

接口请求报错403要怎么处理?

接口403一般是因为页面域名校验失败导致的，您可以在 前端性能监控 > 应用管理 > 应用设置 中检查应用创建时候设置的域名跟实际上报的域名是否一致。如果应用不需要校验，域名可以填 `*` 。



> **说明：**
> 如果不确定您的上报域名是什么的话，可以在浏览器控制台输入 `location.host` 查看。对于非 web 应用，默认填了 `*` 表示不需要校验。

用户在 RUM 上创建应用后，会得到一个长度为18位的字符串，这个字符串为上报 ID，new Aegis 的时候传入的 ID 就是这个上报 ID。如果这个 ID 不正确，也会报403。

Aegis 初始化

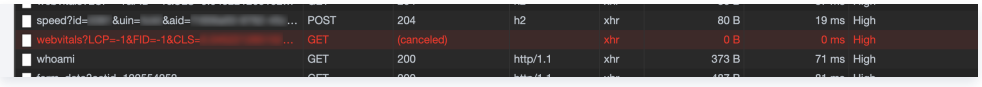
```
new Aegis({id: 'pGUVFTCZyewhxxxxxx'})
```

> **说明：**
> 如果应用刚创建，RUM 数据同步需要一定的时间，大概在1-2min。

日志里面的“图片加载失败”、“JS 加载失败”、“CSS 加载失败” 的问题如何查?

1. 自行访问一下这个资源看是否有异常情况。
2. 若访问正常，但是日志里面还是有比较多的异常情况。
3. 再查看一下这些异常是否有聚集的情况，例如区域聚集，运营商聚集等。
4. 若没有发现有聚集的情况，那大概率是因为用户的网络问题导致的资源加载异常，可以尝试应用中接入 PWA 或者离线包来减少这类情况。本质上，前端没有办法完全避免这种情况。

应用接入 SDK 后，为什么会有一个 `aegis.qq.com/speed/webvitals` 的接口在页面刷新的时候 cancel？



webvitals 中 CLS 的值必须是页面可见性改变的时候才计算的，用户刷新页面的时候，如果选中了 Preserve log，就会有一条请求处理发送中，但恰好被页面刷新给 cancel 掉，就出现这种情况。对用户和数据没有影响，可以忽略。

有不少错误日志是 “Script error. @ (:0:0) 没啥信息” 这种是第三方 JS 异常吗? 可以通过配置过滤掉吗?

Script error. 也被称为跨域错误，当网站请求并且执行一个非本域名下的脚本的时候，如果跨域脚本发生错误，就有可能抛出这个错误。由于应用中，我们的脚本都是放在 CDN 上的，因此这种错误最为常见。

其实这并不是一个 JavaScript Bug。但出于安全考虑，浏览器会刻意隐藏其他域的 JS 文件抛出的具体错误信息，这样做可以有效避免敏感信息无意中被不受控制的第三方脚本捕获。因此浏览器只允许同域下的脚本捕获具体错误信息，而其他脚本只知道发生了一个错误，但无法获知错误的具体内容。更多信息，请参见 Webkit 源码 。

```
289 bool ScriptExecutionContext::sanitizeScriptError(String& errorMessage, int& lineNumber, String& sourceURL)
290 {
291     KURL targetURL = completeURL(sourceURL);
292     if (securityOrigin()->canRequest(targetURL))
293         return false;
294     errorMessage = "Script error.";
295     sourceURL = String();
296     lineNumber = 0;
297     return true;
298 }
```

具体解决方式：

解决办法1：CORS

步骤1：资源添加 crossorigin 属性。

```
<script src="http://another-domain.com/app.js" crossorigin="anonymous"></script>
```

步骤2：CDN 添加 cors 响应头，这个基本是 cdn 默认的，所以实际上我们并不需要做什么。

```
Access-Control-Allow-Origin: *
```

解决办法2：try catch

window.onerror 中只能捕获 `Script error.`，但是 try catch 中却能打印详细的错误栈。

```
<!doctype html>
<html>
<body>
    <script src="http://another-domain.com/app.js"></script>
    <script>
        window.onerror = function (message, url, line, column, error) {
            console.log(message, url, line, column, error);
        }
        try {
            foo(); // 调用app.js中定义的foo方法
        } catch (e) {
            console.log(e);
            throw e; // 主动抛出的错误捕获后不是 Script error
        }
    </script>
</body>
</html>
```

如果不想解决，只想直接屏蔽，可以参见下一个问题，不上报特殊日志。

对于一些特殊的日志，不想上报到 RUM，可以怎处理?

SDK 提供一个 hook 来帮助用户在日志上报前对日志进行屏蔽和修改，可以使用 beforeRequest，返回 false 就可以不上报该条日志，返回修改过后的 data 就可以实现对上报数据的修改。

beforeRequest

```
new Aegis({
  id: 'pGUVFTCZyewhxxxxxx',
  beforeRequest(data) {
    // 入参data的数据结构：{logs: {…}, logType: "log"}
    if (data.logType === 'log' && data.logs.msg.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 log，且内容包含 otheve.beacon.qq.com 的请求
      return false;
    }
    // 入参data数据结构：{logs: {}, logType: "speed"}
    if (data.logType === 'speed' && data.logs.url.indexOf('otheve.beacon.qq.com') > -1) {
      // 拦截：日志类型为 speed，并且接口 url 包含 otheve.beacon.qq.com 的请求
      return false;
    }
    if (data.logType === 'performance') {
      // 修改：将性能数据的首屏渲染时间改为2s
      data.logs.firstScreenTiming = 2000;
    }
    return data;
  }
})
```

SDK 如何获取网络类型？为什么我的网络类型不正确?

若 UA 里面有 NetType ，则 UA 是从 NetType 获取的。若没有，您用的是 `navigator.connection.effectiveType || navigator.connection.type` ， `effectiveType` ，则会根据您的网速推算出的一个类型，并不是实际类型。

为什么 SDK 上报的接口请求耗时跟 network 里面的时间不一致?

SDK 通过劫持 fetch 和 xhr 的方式实现对接口的测速，在请求前和请求后分别进行打点测速，因为 JS 计算时间差的逻辑依赖 js 主线程执行，如果 end 时间执行的时候有线程阻塞，会导致实际计算的时长要大于浏览器 network 时长。

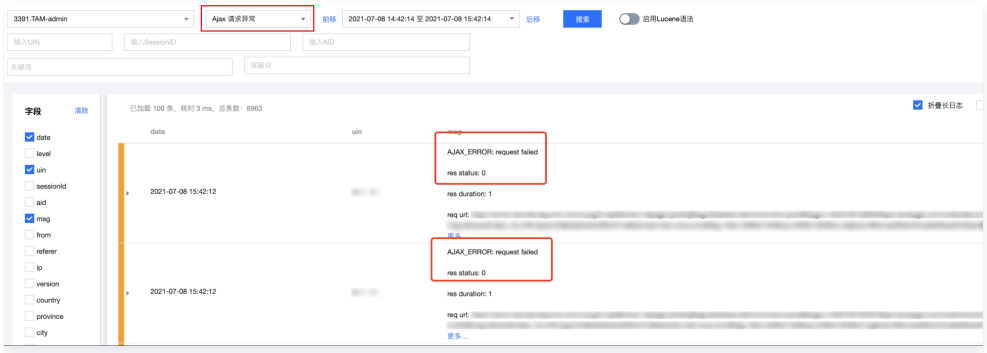
为什么上报的网络类型跟实际网络类型不符合，例如当前用户使用的是 wifi，上报的却是4G?

我们目前通过 UA 里面的 NetType 和 `navigator.connection.effectiveType` 获取网络类型，前者依赖浏览器注入网络信息到 UA，后者是浏览器提供的 “等效网络类型”，并不代表真实的网络类型。 `navigator.connection.effectiveType` 这个 API 目前还存在兼容性问题，IOS 暂不支持，所以会被识别为未知。

我开启了 SPA 参数，但是页面之间跳转的时候，为什么没有上报页面性能呢?

SPA 页面之间的跳转，本质上只是一段 JS 代码的执行，首屏探测的是从用户浏览器发起请求到页面可见元素渲染完成的时间，因此没有办法计算首屏。

日志里面上报了非常多 AJAX 异常，状态码是0，这个可能是什么原因导致的?



HTTP 接口的 status 是0，有以下几种可能：超时、abort、cancel、跨域。
开发者本地调试可能没有发现这种错误，但是在用户侧却能经常发生，这个是什么原因呢? 因为用户侧可能随时离开页面，或者因为网络问题把某次 HTTP 请求中断了，这种情况就会发生 status 为0的情况。
虽然 status 为0有多种情况，但是我们也可以根据日志里面的信息看出一些端倪。例如：上述截图里面的耗时都非常小，而且发生的用户比较集中，因此可以判断该接口是浏览器插件屏蔽掉了，这种情况常见于一些数据上报的接口。如果接口出现比较随机，耗时也不高，整体量也不算大，这种情况，abort 和 cancel 的概率比较大，经常发生于用户离开页面，刷新页面，接口没有请求完成的情况。如果看到 duration 非常长，那超时的可能性就比较大了，开发者也可以查看自己接口的超时时间来进行对比。至于跨域的情况，可以检查一下后台的业务逻辑是否正常，而 cancel 的情况，多半是前端业务控制的，也可以看下具体的业务逻辑。

开发者本地调试可能没有发现这种错误，但是在用户侧却能经常发生，这个是什么原因呢?

因为用户侧可能随时离开页面，或者因为网络问题把某次 HTTP 请求中断了，这种情况就会发生 status 为0的情况。
虽然 status 为0有多种情况，但我们也可以根据日志里面的信息看出一些问题。例如上述截图里面的耗时都非常小，而且发生的用户比较集中，因此可以判断该接口是浏览器插件屏蔽掉了，这种情况常见于一些数据上报的接口。
如果看到 duration 非常长，那超时的可能性就比较大了，开发者也可以查看自己接口的超时时间来进行对比。至于跨域的情况，可以检查一下后台的业务逻辑是否正常，而 cancel 的情况，多半是前端业务控制的，也可以看下具体的业务逻辑。

我API监控页面看到了非常多接口的 retcode 成功率都是0，或者很低，这个是因为什么?

SDK 通过从用户接口的返回值中的第一层和第二层中获取 [code, ret, retcode, errcode] 这几个参数中的任意一个，作为接口业务的返回码（区别于 HTTP 的状态码），默认这个返回码等于0接口为正常。
业务对于返回码有不同的定义，开发者可以通过 api 参数下的 retCodeHandler 函数对其进行矫正，retCodeHandler 函数返回 isErr 和 code 两个值。isErr 用来计算接口返回值的成功率，code 用来统计接口的返回码占比。当该函数返回 isErr 为 true 的时候，这个接口的信息会同时上报到历史日志里面的“接口返回码异常”，方便开发者定位接口问题。除此之外，还有很多非业务的接口，如一些数据上报，是不满足这些规范的，建议可以通过 beforeRequest 进行修正。

为什么我接入 Aegis 后没有首屏数据?

Aegis sdk 根据 DOM 变化记录首屏，如果您的 sdk 引入较晚，或者初始化较晚，可能会出现无法获取首屏的情况。如果出现这种情况的话，建议可以试着把 sdk 引入和初始化放在更前面，如 head 里，然后再观察一下数据。

为什么有些接口的测速数据没有上报?

有可能是因为初始化 Aegis 的时候这个接口已经发出去了。Aegis web sdk 目前只劫持了 fetch 和 xhr 两种发送请求的方式，如果您的请求是其他的方式，例如 beacon，就无法监控到。Aegis 小程序 sdk 也是通过劫持 wx.request 实现的，如果在引入 Aegis 之前，wx.request 已经被修改了，也可能监控不到，建议尽早引入和初始化 Aegis sdk。

页面首屏和页面完全加载时间有什么关系?

总体来说，页面首屏和页面完全加载时间是正相关的。大多数情况下，用户的首屏时间是小于页面完全加载的。Aegis SDK 根据用户页面 DOM 变化计算首屏时间，如果用户页面完全加载后，还继续发生 DOM 变化，就有可能发生首屏时间晚于页面完全加载的情况。在服务端直出场景，瀑布图会出现首屏时间大于 DOM 解析的情况，这是由于移动端设备兼容性问题，有些设备无法获取到 DNS 查询、TCP 连接、SSL 建连时间，这三个指标汇总后的平均值偏小，导致除了首屏时间外的其他指标都往左偏移。

我想要全量上报所有接口的信息需要怎么办?

```
new Aegis({
  id: '',
  api: {
    reportRequest: true, // 接口全量上报，info 对全部用户生效
    apiDetail: true, // 上报接口的同时，也会上报接口请求参数和返回值
  }
})
```

对于 SSR 页面，接入 aegis sdk 需要注意哪些问题?

接直出的页面需要注意两件事情:

1. aegis-web-sdk 只能运行在浏览器环境，所以记得在服务端不要执行初始化操作。
2. SSR 页面初始化完成比较早，所以如果把 aegis-sdk 初始化放在比较靠后的地方（body 里面或者最后）可能会导致 sdk 无法获取到页面首屏的情况。建议 SSR 页面使用 script 标签引入 aegis sdk，并且放在 head 中初始化，解决上述两个问题。

我的接口名称是写在接口的参数里面的，如何进行上报?

aegis sdk 上报数据的时候，默认只上报了接口的 url，没有上报参数，对于这种情况，我们提供 reportApiSpeed.urlHandler 钩子函数对上报内容进行修正，用户可以根据 urlHandler 参数里面的内容进行优化。
例如：腾讯云控制台上的接口，名称统一都是 `console.cloud.tencent.com/cgi/capi`，接口真实名称通过参数里面的 cmd 确定，这样的情况，优化方式如下:

```
new Aegis({
  id: '',
  reportAssetSpeed: true, // 静态资源测速
  reportApiSpeed: {
    urlHandler(url) {
      if (url.indexOf('/cgi/capi') > -1) {
        const urls = url.split(/[=|&|?]/);
        if (urls.length > 6) {
          url = `${urls[0]}/${urls[6]}/${urls[2]}`;
        }
      }
      return url;
    },
  },
});
```

使用相关问题

最近更新时间：2025-01-06 15:21:32

接入前端性能监控，小程序接入，正式环境需要将上报域名添加到安全域名中。安全域名要怎么添加?

Aegis SDK 使用 `https://aegis.qq.com` 进行数据上报，如果不希望使用 qq 域名，您也可以修改为 `https://rumt-zh.com` 。您可登录 微信公众平台 > **开发管理 > 服务器域名**，完成配置。

初始化 SDK 的时候传入的 UIN 是什么?

User Identification Number 用户唯一标识，简称 UIN，是 Aegis SDK 提供给业务侧填写的用户唯一标识，RUM 不进行校验 UIN 是否是一个真实标识，只把它作为查询日志的索引。开发中可以把业务提供的用户 ID 写入，也可以写入 open-id、uuid 等信息，只需方便您查询即可。

自定义测速耗时为什么仅支持0 - 60000之间的数据?

自定义测速的逻辑是用户上报任意数据，我们对用户数据进行求均值和中位数，并把它们作为展示的数据。由于无法判断服务端的脏数据，若产生少量脏数据，将会对用户实际数据产生非常大的影响。因此我们目前在服务端对用户数据进行了限制，目前只支持0 - 60000内的数据。

RUM 根据哪个字段来计算 UV?

Aegis SDK 为每个用户独立生成一个 aid 作为用户（设备）的唯一标识，存储在浏览器的 localStorage，用于计算 UV。aid 不受登录状态影响，只有用户清理浏览器缓存 aid 才会更新。

RUM 为什么不根据用户 UIN 来计算 UV?

如果当前用户没有登录，则没有 UIN 值，并且 UIN 值由开发者设置，可靠性比较低，也无法将用户登录前后的状态统一。

日志类型里面的接口请求日志是什么?

有两种情况用户会把接口请求信息发送服务端：

- 白名单用户将接口请求日志发送上来。
- 对于一般用户，如果发生 js 错误，我们会同步把错误发生之前的一些接口信息也发上来，用于帮助用户查找分析问题。

接口请求日志是用户接口的请求和返回内容。一般只有白名单用户才会上报接口请求日志，或者如果开发者在 sdk 里面开启全量接口上报，也会上报接口请求日志。

首屏时间（FirstScreenTiming）是怎么计算的?

监听页面打开3s内的首屏 DOM 变化，并认为 DOM 变化数量最多的那一刻为首屏框架渲染完成时间（SDK 初始化后 setTimeout 3s收集首屏元素，由于 JS 是在单线程环境下执行，收集时间点可能大于3s）。

性能里面的 DOM 解析（DOM parse）时间是如何计算的?

PerformanceTiming 接口中的 domInteractive 到 domLoading 所消耗的时间。

RUM 采集数据时用的时间是客户侧（如浏览器）还是服务侧? 中间的延迟大概会有多久?

RUM 采集数据时用的服务侧的时间，时间显示会比实际上报延迟1s – 2s，日志搜索可能有1分钟 – 2分钟延迟。

接入前端性能监控后，没有数据是怎么回事?

可以从以下几个方向查找问题：

- 查看上报接口（rumt-zh.com 或者 aegis.qq.com）是否正常，正常返回200和204，如果有一些接口返回403可以看 技术排查相关问题 。
- 看日志查询里面页面访问和历史是否有数据，如果没有数据，有可能是上报延迟导致（1分钟 – 2分钟）。
- 数据总览里面的数据是每整数小时计算一次，如果没有数据可以看下是否上个小时没有页面访问，或者新接入还没开始计算。
- 如果是页面性能没有数据，或者缺少数据，可以看下 network 中上报接口 performance 中 firstScreenTiming 的值，如果这个值为0或者大于15s，就会导致页面性能缺少数据。解决办法：把 SDK 初始化尽可能放在代码最前面，可以考虑用 CDN 并且在 head 中初始化代码，尤其对于一些 SSR 的页面，因为页面首屏时间较小，初始化 SDK 过晚会导致无法采集到页面首屏信息。

> **注意：**
> 目前只有 web 和小程序支持首屏算法。

接入前端性能监控后，如何查看当前项目上报量?

业务使用 RUM，可以在 前端性能监控 > 应用管理 的**业务系统**或者**应用设置**页面查看上报量。

- 业务系统页面

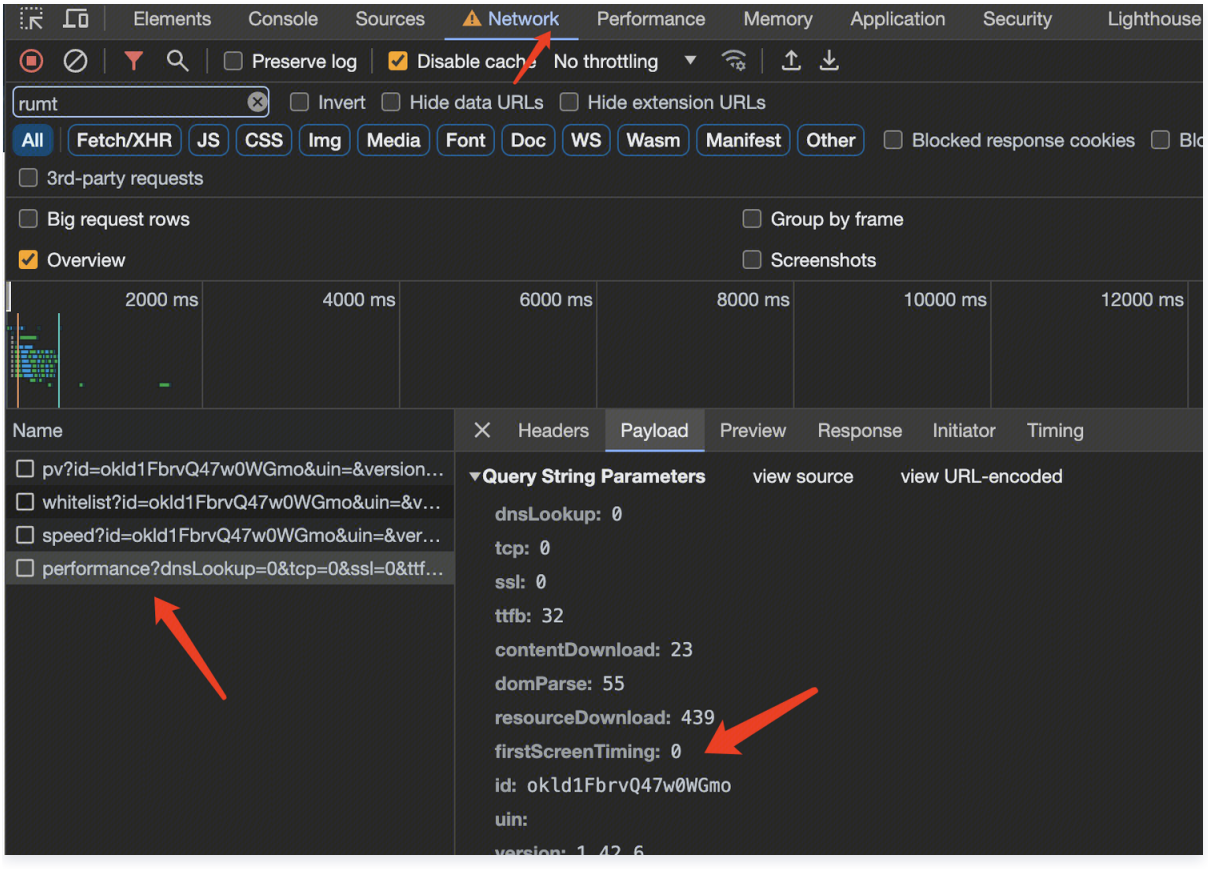
- 应用设置页面

接入前端性能监控后，日志中的“图片加载失败”、“JS 加载失败”、“CSS 加载失败” 问题如何处理?

首先自己访问一下这个资源是否存在异常情况。如果访问正常，但是日志里面还是有比较多的异常情况，再查看一下这些异常是否有聚集的情况，例如区域聚集、运营商聚集等。如果没有发现有聚集的情况，可能是因为用户的网络问题导致资源加载异常，可以尝试项目中接入 PWA 或者离线包来减少这类情况。本质上，前端没有办法完全避免这种情况。

为什么接入 Aegis 后没有首屏数据? 为什么上报接口中首屏数据 firstScreenTiming 为0? 为什么页面性能数据那么少? 为什么首屏数据那么大?

Aegis sdk 根据 DOM 变化记录首屏，如果 SDK 引入较晚，或者初始化较晚，可能会出现无法获取首屏的情况。当出现这种情况时，建议可以试着把 SDK 引入和初始化放在更前面，例如 head 里，然后再观察一下数据。
尤其针对服务端直出的项目（SSR），页面加载完成即首屏完成，如果 Aegis sdk 初始化较晚，就会上报一个首屏时间 **firstScreenTiming** 为0的数据，该数据不是有效数据，因此不参与计算，导致样本量减少，而少数有值的数据上报，综合计算又导致整体首屏值计算偏高。



除此之外，少数非常大的首屏值（异常用户）也会影响平均值，用户可以看下中位数的值是否正常。

为什么页面上报了很多重复数据，例如页面切换时重复上报多个 pv?

可能是重复初始化了多个 Aegis 的实例，建议检查业务逻辑。

对于 SSR 页面，接入 Aegis SDK 需要注意哪些问题?

1. aegis-web-sdk 只能运行在浏览器环境，所以记得在服务端不要执行初始化操作。
2. SSR 页面初始化完成较早，所以如果把 aegis-sdk 初始化放在比较靠后的地方（body 里面或者最后）可能会导致 SDK 无法获取到页面首屏的情况。建议 SSR 页面使用 script 标签引入 Aegis SDK，并且放在 head 中初始化，解决上述两个问题。

日志搜索为什么没办法搜模糊匹配?

例如页面地址为 `https://test.abcdefg.qq.com/`，希望通过 `from:test.*.com` 对这个页面进行搜索，结果发现没办法搜到数据。
建议用户修改搜索关键词，使用多个关键词进行搜索，例如 from: "test" AND from “腾讯网”

为什么接口错误率上报失败率接近100%，但实际接口返回正确?

正常情况下，是由于业务或者第三方框架手动调用 xhr.abort() 导致的。此时 aegis 会获取到 xhr.readyState=0，并且 xhr.status=0，此时会认为接口请求失败。需要重点排查下业务或者第三方框架（特别是 angular），重点搜索下 abort。

前端监控里面统计到的接口耗时（静态资源耗时）跟后端统计的相差很远，可能是什么原因导致的?

1. 用户网络问题，请求没有发生到后端，耗时比较久后失败了。
2. 用户请求发出了，但是中间网络层耗时比较久，后端处理很快，但是过了很久才返回给用户。
3. 用户看的是平均值，可能由于某个或者某几个用户出现比较高的异常值导致整体值偏低，用户可以看下中位数是否依然异常。
4. 用户请求接口(静态资源)受多方面因素影响：网络、地域、设备都会对最终值产生影响。
5. cdn 没有预热，dns 第一次解析耗时比较久等原因都会导致前后端统计不一致。
6. 前端性能监控只是把这个耗时记录下来，并不对这个耗时负责。

前端监控统计到 cdn 资源失败，但是实际自己访问没有出现问题，可能是什么原因导致的?

首先看下对应 cdn 的成功率是多少，如果这个 cdn 大概率都是成功的，可能是因为前端访问这个资源时超时、abort 等因素导致的，一般情况可以忽略。
如果 cdn 整体成功率不高，可能就是 cdn 确实有问题了，您可以从以下几个方面去查：

- 本地访问这个 cdn 是否有问题，如果有问题，找具体人排查。
- 看下区域分布是否有问题，可以看前端监控的数据，也可以在 腾讯云诊断服务平台 查看资源。
- 同样可以看下运营商分布，也可能出现局部运营商把 cdn 封禁的情况。

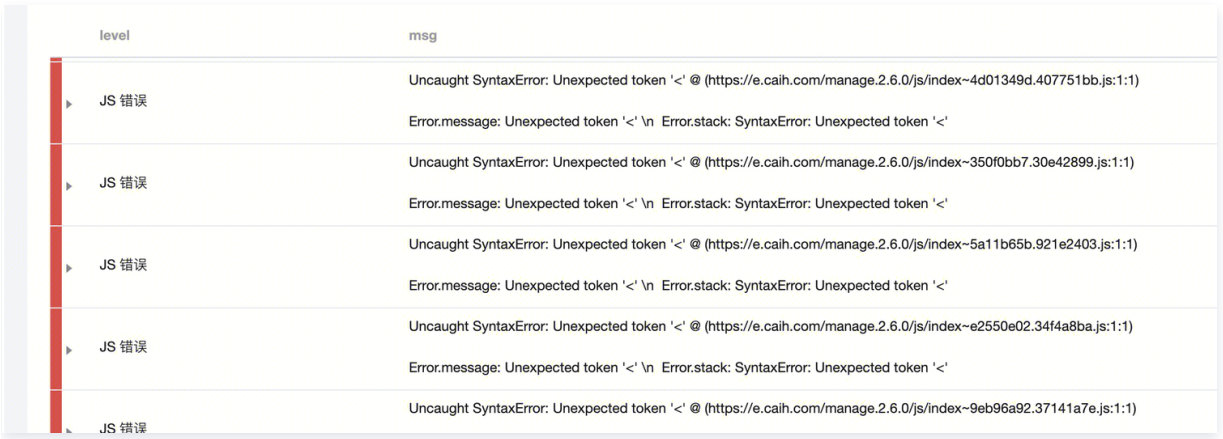
为什么不能看某个具体用户的性能数据，例如接口耗时、静态资源耗时、首屏耗时等数据的原始数据?

对于性能类的数据，我们只保留了计算后的数据，例如平均值、错误率、中位数、90分位数等，所以无法根据用户信息进行查找。如果用户对此有诉求，可以用以下几个方式进行查询：

1. 开启全量接口耗时上报，具体参考 api.reportRequest 。
2. 如果业务数据量比较少，可以把用户信息写到 ext 中，然后通过 ext 进行查询。
3. 可以在 beforeRequest 中把性能数据，例如接口耗时、静态资源耗时等信息通过 aegis.infoAll 上报到历史日志中，然后去历史日志中进行查询。

在日志里面的 js 错误日志里面看到一些莫名其妙的错误，代码里面找不到相关的问题，可能是哪些原因导致的?

有些用户在历史日志中，发现了一些奇怪的报错，本地不会复现，并且搜源码也没搜到相关的问题。这种情况大概率由于用户侧执行的代码跟开发者执行的代码不一致导致的。



可以从两个方面排查这类问题。首先，如果报错的用户比较集中（可以根据 aid、IP 等信息来看），可能是由于用户浏览器插件导致的。如果用户有某些聚类特征，例如主要集中在某个地区、某个运营商等，也有可能是 cdn 劫持导致的。

- 如果是 cdn 劫持，开发者可以通过使用 sri 等方案来解决。
- 如果是浏览器插件导致的，其实对整个业务没什么影响，可忽略。如果不希望出现在报错信息中，也可以通过 beforeRequest 钩子函数进行屏蔽，具体参考 处理方法 。
- 如果用户页面运行在某些终端中，或者小程序中，也有可能终端或者小程序注入代码导致 js 报错，如常见的“ReferenceError: Can't find variable: WeixinJSBridge”

开启了 SPA 参数，但是页面之间跳转的时候，为什么没有上报页面性能呢?

SPA 页面之间的跳转，本质上只是一段 js 代码的执行，首屏探测的是从用户浏览器发起请求到页面可见元素渲染完成的时间，因此，没有办法计算首屏。

抽样是如何抽的?

目前抽样分为服务端抽样和客户端抽样，客户端抽样通过参数 random 来控制。

```
new Aegis({
   id: 'pGaaFTC******VTKBe',
   random: 0.5 // random 0~1
});
```

无论服务端抽样还是客户端抽样，都是基于用户级别的抽样（即如果一个用户命中黑名单，所有 aegis 上报接口都不会上报）。

> ⚠ 注意:
>
> - 如果同时设置服务端抽样和用户抽样，则实际抽样率为：客户端抽样率 * 服务端抽样率。
> - 白名单用户不受抽样率影响。
> - 服务端抽样通过 whitelist 接口下发抽样率控制，实际上报量会比设置的抽样率高，因为有些接口在 whitelist 接口返回之前就上报了。
> - RUM 中可以设置精准的服务端采样。

为什么使用 aegis-first-screen-timing 标记后的元素不是最终 aegis.firstScreenInfo 里面的元素?

aegis.firstScreenInfo 默认为关闭状态，如果需要该值，需要主动开启。

```
new Aegis({
  id: '',
  pagePerformance: {
    firstScreenInfo: true,
  }
})
```

因为标记的元素有可能是跟其父元素同时被 MutationObserver 收集进来的，此时 aegis.firstScreenInfo 里面的 element 和 markDoms 对应的是标记元素的父元素。如果 aegis.firstScreenInfo 为 undefined，请注意设置 aegis-first-screen-timing 的元素必须得有宽高，同时 window.innerWidth、window.innerHeight 也需有值。

1. 可以尝试把 aegis 使用 cdn 的方式引入，并且在 head 中进行初始化，看下是否初始化太晚导致的。
2. 在渲染后的 html 中看下 dom 中标记的元素是否存在，标记的自定义属性是否存在，有些构建工具会自动删除这类自定义属性。

第三方 SDK、插件、组件内部可以使用 Aegis SDK 吗?

不建议这样使用，因为我们的 SDK 是由 for 页面设计，默认采集页面的 pv、错误、全局的接口信息等，如果您在自己的 SDK 内部使用了 Aegis SDK，会导致页面数据重复采集、xhr 重复劫持等问题。而且 Aegis SDK 默认采集数据上报，用户页面并没有授权第三方的 SDK 采集其他页面的数据，因此可能还存在合规的问题。

上报数据中的 eval 为什么会变成 eval?

js 里面 eval 是一个敏感操作，之前会被腾讯的网关屏蔽报错，所以被改成了 eval。

```
220         xhr.setRequestHeader('Content-Type', options.contentType);
221       }
222       if (typeof options.data === 'string') {
223         options.data = options.data.replace(/eval/gi, 'evaI');      nateji, 21个月前 •
224       }
225
226       xhr.send(options.data);
227     }
```

小程序上报的 referer 的值如何理解?

小程序的 referer 指的是小程序的场景值，具体值可以参考 小程序文档 。

小程序 SDK 初始化之后报错 “unexpected loadSdkSubPackage”。

可点击 此处 参考解决。可以更新微信开发工具，或者在微信开发者工具的**详情**页面，在**本地设置**中勾选**启用独立域进行调试**。

上报统计的接口耗时跟浏览器中 network 中接口耗时不一致可能是什么原因导致的?

Aegis SDK 中默认采用劫持 xhr 和 fetch 请求的方式统计接口耗时，即在接口请求之前打一个点，接口请求之后打一个点，然后计算这两个点之间的时间差值，为接口耗时。这样统计接口耗时可能跟 network 中接口真实耗时有所不同，可能看起来明显大于真实接口耗时。主要有以下几个原因导致：

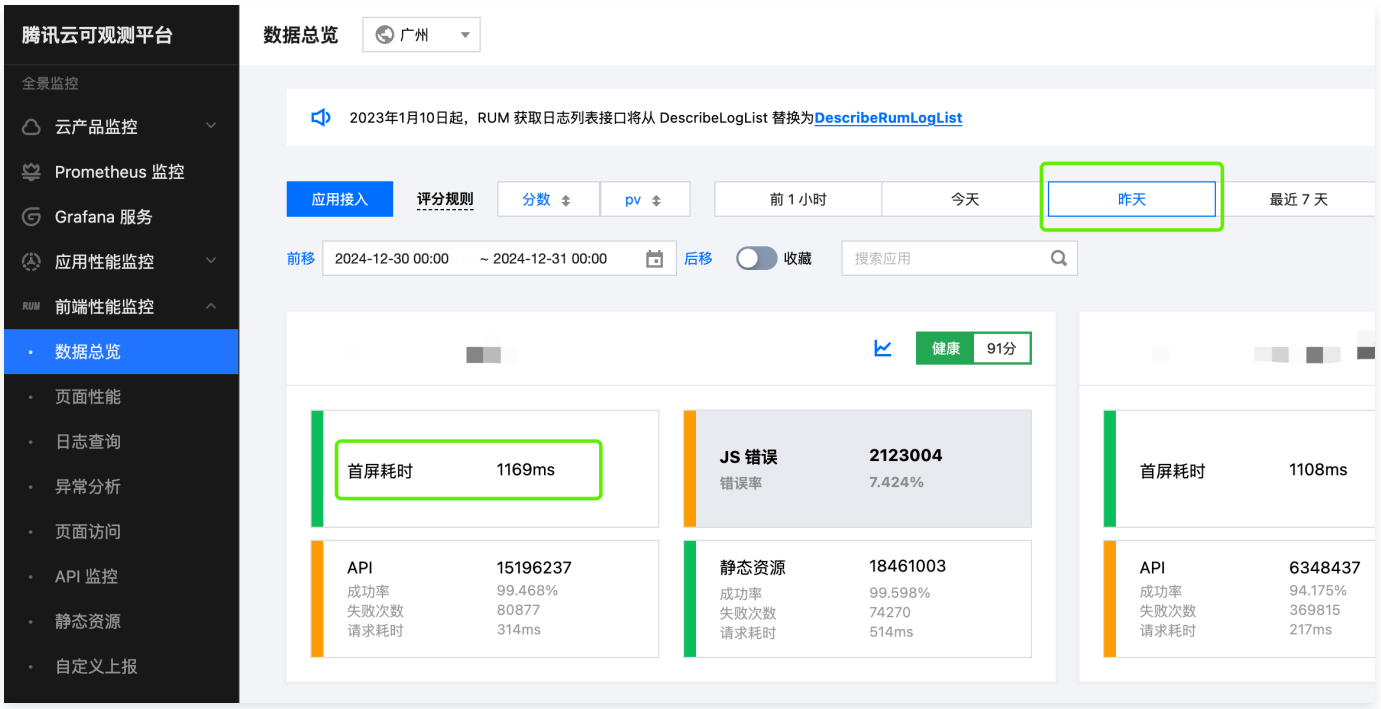
- 浏览器有请求并发限制，js 代码把请求事件发送给浏览器之后，浏览器并不是立即执行的，而且把所有接口排队然后执行，如果接口使用 http/1.x 协议并且进入到浏览器并发限制，就会导致这个问题。因为我们统计的是 js 的耗时，浏览器阻塞时间会默认加上，导致统计时间大于真实接口耗时。
- js 打点的方式也会遇到单线程阻塞的问题，两次记时的代码并非真实时间，而且两次时间循环的时间，就像遇到的 setTimeout 和 setInterval 的问题一样。

解决方案：
可以开启 api.usePerformanceTiming 参数。开启后，Aegis SDK 会从 performance 中重新获取接口耗时，让接口耗时统计更为准确。需要注意的是，开启该参数的前提是用户业务接口请求 url 是独立且唯一的，用户可以在接口请求 url 中添加时间戳等方式来保证 url 唯一性。如果 url 不唯一，可能导致同 url 接口耗时获取错误，业务本身无重复 url 的情况则可以直接开启。

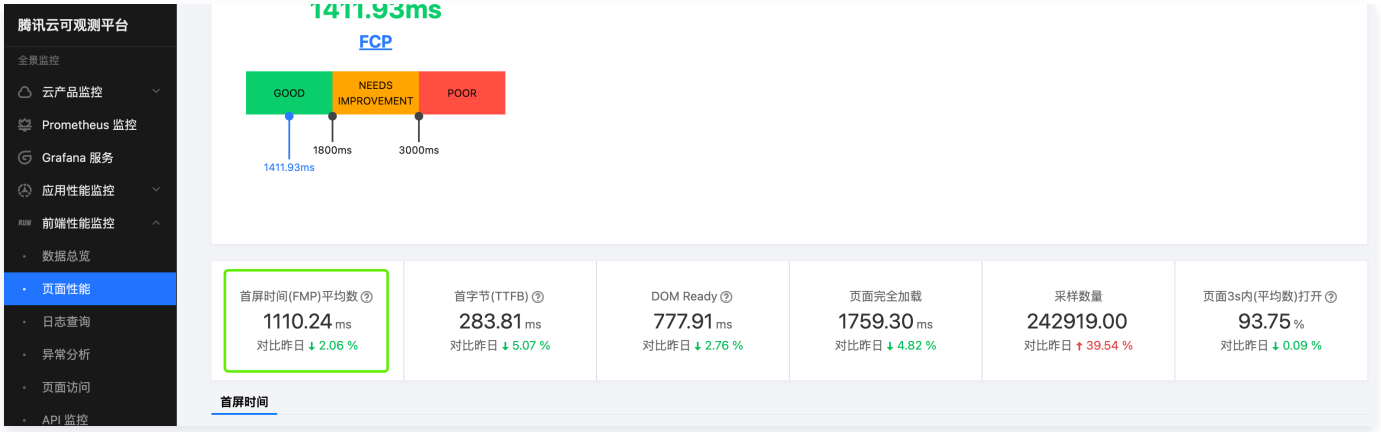
数据总览里面的首屏耗时或者请求耗时为什么与详情页面里面查到的数据不一致?

数据总览里面的值，是通过定时任务计算出来的，每个小时计算一次，如果选了多个小时，相当于多个小时的计算平均值的平均值，而数据详情里面的值，是真实的平均值，所以这两个略有区别。

- 数据总览 的值



- 数据详情的值，在 页面性能 中查看。



Aegis SDK 中是否有限制日志长度?

Aegis SDK 中有限制每条日志最大长度为100k，超过时日志会被截断。同时，数据上报的接入层也有限制单条日志最大长度为100k，并且网关有限流每次上报数据不能超过500k。

首屏数据能把 url 后面的 query 参数带上吗?

RUM 对于性能数据的页面地址，只取了 path，不取 query ，核心是为了降低 query 对数据聚合的影响。但是有些用户的页面核心信息是写在 query 里面的，导致无法区分不同页面的性能。对此，我们提供一个 钩子函数 。用于修改上报性能数据的地址。

[3] pagePerformance.urlHandler，假如您的页面url是restful风格的，如：
www.example.com/user/1000
www.example.com/user/1001

在上报页面测速时需要将这些页面地址聚合：

```js
new Aegis({
  // xxx
  pagePerformance: {
    urlHandler() {
      if ((/www\.example\.com\/user\/\d*/).test(window.location.href)) {
        return 'www.example.com/user/:id';
      }
    }
  }
  // xxx
})
```

[4] pagePerformance.firstScreenInfo，默认值为 false，如果需要在内存中保留 aegis.firstScreenInfo

只需要在 urlHandler 中修改返回值就行，该返回值就是上报页面性能数据中页面地址的值。

> ⚠ **注意：**
> 在修改时，可以把需要的 query 关键词放在 path 里面，例如原 url 是 `https://a.qq.com/?name=tick`，可以修改为 `a.qq.com_tick`。

webvitals 里面统计的 FCP 和 LCP 是否包含了浏览器滚动条滚动后页面的渲染？如果用户有操作，例如 spa 页面跳转、页面重新渲染，时间是否有影响？

webvitals 中的数据是从浏览器 API 中获取的，所以这部分实现对于我们来说是黑盒的，但是我们也可以根据官方给出的描述和自己实验给出结论。
首先拿 LCP 来说，看官方的定义：
Largest Contentful Paint is the metric that measures the time a website takes to show the user the largest content on the screen, complete and ready for interaction. Google defines that this metric considers only the content above the page's fold, meaning everything that appears without scrolling.
由此可见，页面滚动后的内容渲染不在 LCP 的计算范围内，对于“页面渲染时触发浏览器操作是否有影响”这个点，我们可使用 Chrome 浏览器放慢网速，模拟相关操作，发现也是没有计算在内。再重新理解这个定义，可以看到 “complete and ready for interaction” 表示的是在浏览器可交互之前，且浏览器未发生滚动的时候，页面最大元素渲染完成时间。

Aegis SDK 采集到错误 Failed to execute 'trAansaction' on 'IDBDatabase' 如何解决？

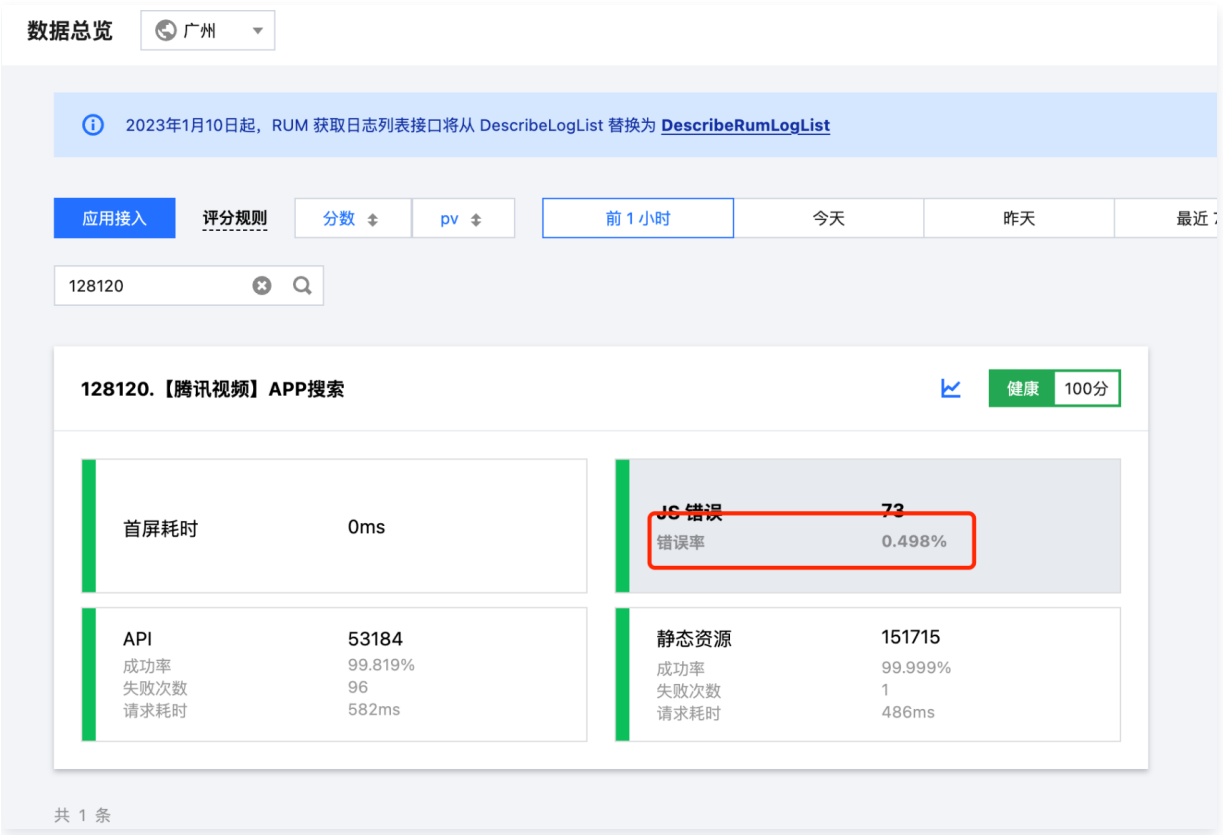
Aegis SDK 中，如果开启了离线日志，会将用户日志写到 indexedDB 中，如果浏览器 indexedDB 满容或者支持性有问题，就会报这个错误，目前对用户侧无影响。由于离线日志已经下线，建议用户升级 SDK，或者关闭离线日志（删除初始化中 offlineLog: true 这行代码）。

Aegis SDK 可以自动捕获 console.error 的错误吗？

不能，因为可能导致重复重写的问题，例如用户自己已经重写过 console.error 了，捕获后进行上报；如果用户对这个逻辑不了解，可能会导致一些敏感数据上报，因此 Aegis SDK 默认没有捕获 console.error 中抛出的日志。用户有需求的话可以自行重写，并且使用 aegis.report 或者 aegis.error 进行上报。

JS 错误率的分母是怎么来的？

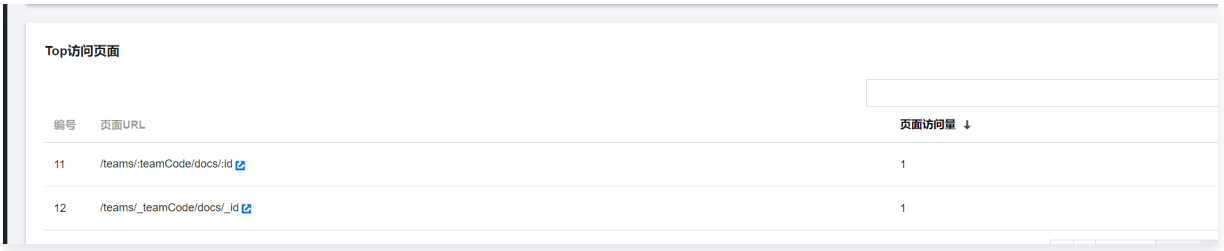
分母数据来源于 PV 总条数。



在使用前端性能监控的 aegis-web-sdk 时，由于 sdk 自带 flog: "https://cdn.go.cn/vasdev/web webpersistance y2/v1.8.2/flog.core.min.js"这一段远程代码，导致这的插件无法通过审核，怎么解决?

升级最新的 SDK 即可，目前最新版本为1.39.1。

上传一种 URL，却出现两种 URL 格式?



目前后台为两部分：node 和 go。node 里面上报指标用的是纯文本协议，不能有特殊字符，因此我们就在 node 代码里面将 : 替换成 _ 。后期会将全部流量都迁移至 go，此问题会被解决。

RUM 告警不支持蓝鲸?

只有上报域名 `aegis.qq.com` 才支持蓝鲸。

performance 页面测速接口上报两次?

客户自己设置某个元素为首屏元素 AEGIS-FIRST-SCREEN-TIMING，当 aegis 识别到该元素后认为首屏已经完成，会立马上报。第二次上报为正常上报。

RUM 报表的每日邮件推送未收到?

检查 通知模板 的账户邮箱是否已经验证。

RUM 上报接口报错400?

本地检查上报参数，一般400参数是上报参数异常。

每条上报日志长度大小为多少?

每条日志的字符最多只能有200 * 1024个。

RUM 会上报过滤敏感词日志吗? 可以修改过滤配置吗?

会的，一般涉及公司内部的敏感词汇日志会被过滤。例如 facebook、baidu、zhiyan 等；不能修改过滤配置，涉及安全风险。

RUM 支持 traceId 查询全链路信息吗? 如何使用?

支持。具体参考腾讯云可观测平台 OpenTelemetry 前后链路打通 。

DAU、WAU、MAU 分别表示日，周，月的什么指标?

数据根据 aid 进行计算，aid 是 sdk 为用户独立生成的唯一标识。

- DAU：日活跃用户数
- WAU：周活跃用户数
- MAU：月活跃用户数

首屏耗时的定义

首屏耗时表示监听页面打开3s内的首屏 DOM 变化，并认为 DOM 变化数量最多的那一刻为首屏框架渲染完成时间（SDK 初始化后 setTimeout 3s 收集首屏元素，由于 JS 是在单线程环境下执行，收集时间点可能大于3s）。

没找到 lcp\fcp 指标的上报? 上报时机是怎样的?

部分指标依赖页面隐藏，时机是在切换页面上报，本地可以切换 tab 看看。

flv 数据格式导致请求不会断开

升级最新版 sdk 即可，已经兼容 flv。

不引入 aegis-sdk，直接使用接口上报日志可以吗?

可以支持，但不建议。

直接使用接口上报日志，有上报量，但没有详细日志?

请求中要携带 logBody 参数，未携带 logBody 则只会统计上报量，没有日志。着重注意 content-type 的内容是 json 还是 urlParams 格式。