ALIBABA CLOUD

阿里云

# 应用实时监控服务 ARMS

## 前端监控

文档版本：20220609

阿里云

# 法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| `Courier字体` | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid`<br>*Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

# 1.什么是ARMS前端监控？

ARMS前端监控专注于对Web场景、Weex场景和小程序场景的监控，从页面打开速度（测速）、页面稳定性（JS诊断错误）和外部服务调用成功率（API）这三个方面监测Web和小程序页面的健康度。

## 为什么要有前端监控？

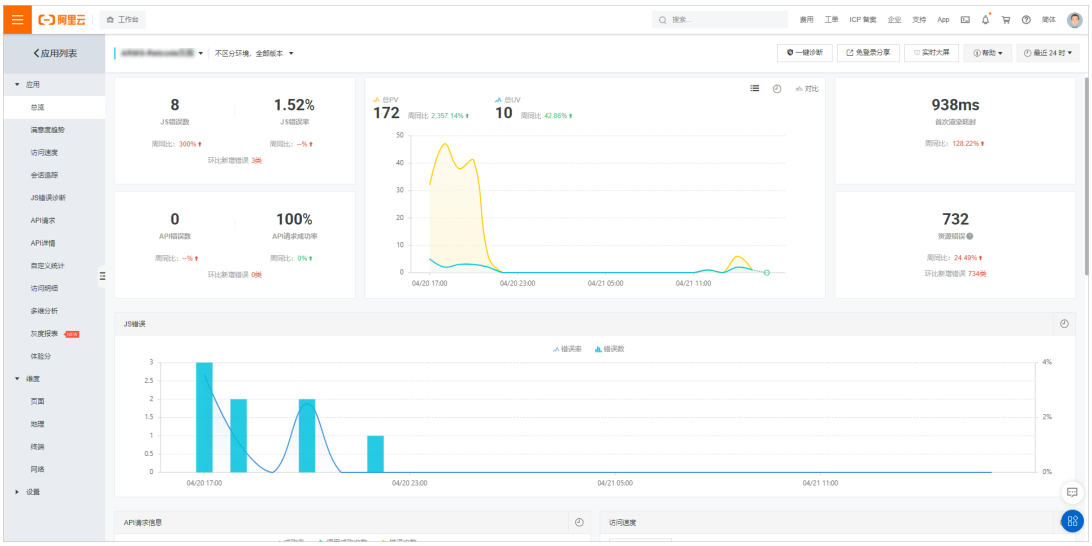
用户访问您的业务时，整个访问过程大致可以分为三个阶段：页面生产时（服务器端状态）、页面加载时和页面运行时。

为了保证线上业务稳定运行，我们会在服务器端对业务的运行状态进行各种监控。现有的服务器端监控系统相对已经很成熟，而页面加载和页面运行时的状态监控一直比较欠缺。例如：

- 无法第一时间获知用户访问您的站点时遇到的错误。
- 各个国家、各个地区的用户访问您的站点的真实速度未知。
- 每个应用内有大量的异步数据调用，而它们的性能、成功率都是未知的。

## 我们的解决方案

ARMS前端监控重点监控页面的加载过程和运行时状态，同时将页面加载性能、运行时异常以及API调用状态和耗时等数据，上报到日志服务器。之后借助ARMS提供的海量实时日志分析和处理服务，对当前线上所有真实用户的访问情况进行监控。最后通过直观的报表展示，帮助您及时发现并诊断问题。




## 支持多种场景的前端监控

ARMS前端监控支持以下场景的前端监控。单击图标即可跳转至相应文档。

## ARMS前端监控能力概览

ARMS前端监控具备丰富的前端监控能力，以下是一些示例。单击图标即可跳转至相应文档。

## 浏览器和平台兼容性

| 浏览器/平台 | 支持版本 | SDK自动上报 | 用户自主上报 |
| --- | --- | --- | --- |
| Safari | Safari 9+ | ✔ | ✔ |
| Chrome | Chrome 49+ | ✔ | ✔ |
| IE | IE 9+ | ✔ | ✔ |

| 浏览器/平台 | 支持版本 | SDK自动上报 | 用户自主上报 |
| --- | --- | --- | --- |
| Edge | Edge 12+ | ✔ | ✔ |
| Firefox | Firefox 36+ | ✔ | ✔ |
| Opera | Opera 43+ | ✔ | ✔ |
| Safari for iOS | Safari for iOS 9.2+ | ✔ | ✔ |
| Android Browser | android_webkit 4.4.2+ | ✔ | ✔ |
| Weex | Weex 0.16.0+ | ✖ | ✔ |

# 2.前端监控文档导读

本文介绍ARMS前端监控文档的总体框架，单击标题即可跳转至相应文档。

# 3.接入前端监控

## 3.1. 前端监控接入概述

ARMS 前端监控支持针对 Web 场景、Weex 场景和小程序场景的监控，请按照对应的文档开始使用前端监控。

### Web场景

**Web场景**

- 以CDN方式安装探针

**Web场景**

- 以npm方式安装探针

### Weex场景

**Weex场景**

- Weex接入配置

### 小程序场景

**小程序场景**

- 开始监控钉钉小程序

**小程序场景**

- 开始监控支付宝小程序

小程序场景

- 开始监控微信小程序

小程序场景

- 开始监控其他类别小程序

# 3.2. Web场景

## 3.2.1. 以npm方式接入前端监控

要使用ARMS前端监控子产品监控Web应用，必须先以CDN或npm方式安装探针。本文介绍如何以npm方式为Web应用安装ARMS前端监控探针。

### 安装

在npm仓库中安装 `alife-logger` 。

```
npm install alife-logger --save
```

### 初始化

SDK以 `BrowserLogger.singleton` 方式初始化。

```
const BrowserLogger = require('alife-logger');
                // BrowserLogger.singleton(conf) conf传入config配置。
                const __bl = BrowserLogger.singleton({
                pid: 'your-project-id',
                // 设定日志上传地址：
                // 部署新加坡地域可设为`https://arms-retcode-sg.aliyuncs.com/r.png?`。
                // 部署美西地域可设为`http://arms-us-west-1.console.aliyun.com/r.png?`。
                imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
                // 其他config配置。
                });
```

使用npm方式接入ARMS前端监控时，Web端SDK会自动生成UID来统计UV等信息。自动生成的UID可以用来区分用户的标识，但不具有业务属性，如需自定义UID，请在上述代码中加入以下内容：

```
uid: 'xxx', // 该值用于区分用户的标识，根据业务设置。
```

示例：

```
const BrowserLogger = require('alife-logger');
                // BrowserLogger.singleton(conf) conf传入config配置。
                const __bl = BrowserLogger.singleton({
                    pid: 'your-project-id',
                        // 设定日志上传地址：
                        // 部署新加坡地域可设为`https://arms-retcode-sg.aliyuncs.com/r.png?`。
                        // 部署美西地域可设为`http://arms-us-west-1.console.aliyun.com/r.png?
`。
                    uid: 'xxx', // 该值用于区分用户的标识，根据业务设置。
                        imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
                    // 其他config配置。
                });
```

## API说明

> **说明** 该方法只适用于npm引入。

调用参数说明：`BrowserLogger.singleton(config,prePipe)`

静态方法，返回一个单例对象，传入的config、prePipe只在第一次调用时生效，此后调用只返回已经生成的实例。

| 参数 | 类型 | 描述 | 是否必须 | 默认值 |
| --- | --- | --- | --- | --- |
| config | Object | 站点配置，其他配置查看config配置项，请参见SDK参考。 | 是 | 无 |
| prePipe | Array | 预上报内容 | 否 | 无 |

此方法可以用于在应用入口初始化SDK，也可以在每次调用时获取实例。

通过 `BrowserLogger.singleton` 获取实例。

```
const __bl = BrowserLogger.singleton();
```

关于 `__bl` 的其他API使用方式，请参见API参考。

Config配置与CDN引入配置相同。请参见SDK参考。

如果在调用 `BrowserLogger.singleton()` 之前执行的部分逻辑需要上报一些数据，则需要使用数据预上报。具体操作，请参见数据预上报。

```
const BrowserLogger = require('alife-logger');
                // 与CDN的Pipe结构一致。
                const pipe = [
                // 将当前页面的HTML也作为一个API上报。
                ['api', '/index.html', true, performance.now, 'SUCCESS'], //相当于__bl.api(api, success, time, code, msg)。
                // SDK初始化完成后即开启SPA自动解析。
                ['setConfig', {enableSPA: true}]
                ];
                const __bl = BrowserLogger.singleton({pid:'站点唯一ID'},pipe);
```

## @static singleton() 获取单例对象

## 其他上报API

## Config配置

## 预上报

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `true` |

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| enableSPA | Boolean | 监听页面的hashchange事件并重新上报PV，适用于单页面应用场景。 | 否 | `false` |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

## 相关文档

- 前端监控接入概述
- SDK参考
- 以CDN方式接入前端监控
- 在Weex环境接入前端监控

# 3.2.2. 以CDN方式接入前端监控

如果需要使用ARMS前端监控来监控Web应用，必须先通过CDN或npm方式安装探针。本文介绍如何通过CDN方式为Web应用安装ARMS前端监控探针。

## 安装前端监控探针

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**，并在顶部菜单栏选择目标地域。
3. 在**前端列表**页面右上角单击**创建应用站点**。
4. 在**接入中心**面板的**前端、移动端和用户端应用**区域，单击**Web & H5**。


5. 在**接入Web & H5**面板的**创建应用**区域输入自定义的应用名称。
6. 在**接入Web & H5**面板的**SDK扩展配置项**区域选中需要的选项，便于快捷生成待插入页面的BI探针代码。
   - **关闭API自动上报**：选中此项后，需手动调用`__bl.api()`方法上报API成功率。
   - **开启SPA自动解析**：选中此项后，ARMS会监听页面的`hashchange`事件并自动上报PV，适用于SPA（Single-Page Application，单页面应用）场景。
   - **开启首屏FMP采集**：选中此项后，ARMS将采集首屏FMP（First Meaningful Paint，首次有效渲染）数据。
   - **开启页面资源上报**：选中此项后，在页面onload事件触发时会上报页面加载的静态资源。
   - **与应用监控关联**：选中此项后，API请求会与后端应用监控进行端到端关联。
   - **开启用户行为回溯**：选中此项后，JS错误诊断可提供用户行为回溯。
   - **开启Console追踪**：开启此项后，用户行为回溯会追踪Console内容，包括`error`、`warn`、`log`、`info`等。

> **注意** 此功能会影响Console的路径。

7. 选择其中一种方法安装前端监控探针。
   - 异步加载：复制提供的代码并粘贴至页面HTML中 `<body>` 元素内部的第一行，然后重启应用。


完整的HTML示例：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="ie=edge,chrome=1" />
    <meta name="viewport" content="width=device-width" />
    <title>ARMS</title>
  </head>
  <body>
    <script>
      !(function (c, b, d, a) {
        c[a] || (c[a] = {})
        c[a].config = {
          pid: 'xxx',
          appType: 'web',
          imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
          sendResource: true,
          enableLinkTrace: true,
          behavior: true,
          useFmp: true,
          enableSPA: true,
        }
        with (b) with (body) with (insertBefore(createElement('script'), firstChild))
setAttribute('crossorigin', '', (src = d))
      })(window, document, 'https://retcode.alicdn.com/retcode/bl.js', '__bl')
    </script>
    <div id="app"></div>
  </body>
</html>
```

- 同步加载：复制提供的代码并粘贴至页面HTML中 `<body>` 元素内部的第一行，然后重启应用。


- npm包：
  a. 执行以下命令以引入npm包。

```
npm install alife-logger --save
```

  b. 从控制台上复制以下初始化命令并执行。

```
const BrowserLogger = require('alife-logger');
const __bl = BrowserLogger.singleton({pid:"b590lhguqs@8cc3f63543d****",appType:"w
eb",imgUrl:"https://arms-retcode.aliyuncs.com/r.png?",sendResource:true,behavior:
true,enableLinkTrace:true,enableConsole:true});
```

## 异步加载和同步加载方式的区别

- 异步加载：又称为非阻塞加载，表示浏览器在下载执行JS之后还会继续处理后续页面。若对页面性能的要求非常高，建议使用此方式。

> 注意 由于是异步加载，ARMS无法捕捉到监控SDK加载初始化完成之前的JS错误和资源加载错误。

- 同步加载：又称为阻塞加载，表示当前JS加载完毕后才会进行后续处理。如需捕捉从页面打开到关闭的整个过程中的JS错误和资源加载错误，建议使用此方式。

## 自定义UID

使用同步加载或异步加载方式安装ARMS前端监控探针时，Web端SDK会自动生成UID来统计UV等信息。自动生成的UID可以用来区分用户的标识，但不具有业务属性，如需自定义UID，请在上述代码 `config` 中加入以下内容：

```
uid: "xxx" // 该值用于区分用户的标识，根据业务设置。
```

例如：

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxx",appType:undefined,imgUrl:"https:
//arms-retcode.aliyuncs.com/r.png?", uid: "xxxx"};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("crosso
rigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

**注意** 如果更改了上一步的SDK扩展配置项，代码将会发生变化，请务必重新复制粘贴。

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `true` |
| enableSPA | Boolean | 监听页面的hashchange事件并重新上报PV，适用于单页面应用场景。 | 否 | `false` |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求，具体操作，请参见SDK参考。

### 相关文档

- 前端监控接入概述
- SDK参考
- 以npm方式接入前端监控
- 在Weex环境接入前端监控

# 3.3. Weex场景

## 3.3.1. 在Weex环境接入前端监控

本文介绍如何在Weex环境中接入ARMS前端监控。

### 导入npm包

如需在Weex环境中使用ARMS前端监控，则首先需要在项目中执行以下命令导入alife-logger npm包，以使用专门的WeexLogger模块来上报日志。

```
npm install alife-logger --save
```

### 初始化

在*/utils*目录下创建monitor.js文件，并按照以下示例代码完成SDK初始化。

> **说明** 在Weex应用入口调用 `singleton(props)` 静态方法获取实例时，需要在传入的props中设定相关配置，请参见：
>
> - 通用API：@static singleton()
> - 通用API：setPage()
> - 通用API：setConfig()

```
// 在monitor.js文件中增加以下内容。
import WeexLogger from 'alife-logger/weex';
const fetch = weex.requireModule('stream').fetch;
const serialize = (data) = >{
    data = data || {};
    var arr = [];
    for (var k in data) {
        if (Object.prototype.hasOwnProperty.call(data, k) && data[k] !== undefined) {
            arr.push(k + '=' + encodeURIComponent(data[k]).replace(/\(/g, '%28').replace(/\
)/g, '%29'));
        }
    }
    return arr.join('&');
}
// 初始化SDK。
const wxLogger = WeexLogger.singleton({
    pid: 'your-project-id',
    uid: 'zhangsan',
    // 设置uid，用于生成UV报告。
    page: 'Lazada | Home',
    // 设置初始页面名称。SDK将在初始化完成后发送PV日志。
    imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
    // 设定日志上传地址。如需部署至新加坡地域，则改为'https://arms-retcode-sg.aliyuncs.com/r.png?
'。
    // 设置GET上报方法，示例如下：
    sendRequest: (data, imgUrl) = >{
        const url = imgUrl + serialize(data);
        fetch({
            method: 'GET',
            url
        });
    },
    // 设置POST上报方法，示例如下：
    postRequest: (data, imgUrl) = >{
        fetch({
            method: 'POST',
            type: 'json',
            url: imgUrl,
            body: JSON.stringify(data)
        });
    }
});
export
default wxLogger;
```

## 上报

通过实例调用相应的上报方法进行上报。

```
// in some biz module
import wxLogger from '/utils/monitor';
wxLogger.api('/search.do', true, 233, 'SUCCESS');
```

## 通用API：`@static singleton()`

`@static singleton()` 为静态方法，作用是返回一个单例对象。props只在第一次调用时生效，用法如下表所示。

此方法可用于在应用入口初始化SDK，请参见初始化。

`WeexLogger.singleton(props)` 调用参数说明

| 属性 | 类型 | 描述 | 是否必需 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 站点ID | 是 | 无 |
| page | String | 初始化的Page Name | 否 | 无 |
| uid | String | 用户ID | 是 | 无 |
| imgUrl | String | 日志上传地址，以英文问号（?）结尾。 | 否 | 无 |

## 通用API：`setPage()`

`setPage()` 的作用是设置当前页面的Page Name，并且默认上报一次PV日志。

```
import wxLogger from '/utils/monitor';
// ...
wxLogger.setPage(nextPage);
```

`wxLogger.setPage(nextPage)` 调用参数说明

| 参数 | 类型 | 描述 | 是否必需 | 默认值 |
| --- | --- | --- | --- | --- |
| nextPage | String | Page Name | 是 | 无 |

## 通用API：`setConfig()`

`setConfig()` 的作用是在SDK初始化完成后修改部分配置项，具体配置与 `singleton()` 方法相同。配置项详情，请参见setConfig()。

```
import wxLogger from '/utils/monitor';
// ...
wxLogger.setConfig(config);
```

`wxLogger.setConfig(config)` 调用参数说明

| 参数 | 类型 | 描述 | 是否必需 | 默认值 |
| --- | --- | --- | --- | --- |
| config | Object | 需要修改的配置项以及值。 | 是 | 无 |
| uid | String | 用户ID，用于统计UV。 | 是 | Storage设置 |

### 日志上报API

请参见API参考中的日志上报接口。

### 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 是 | 无 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

# 3.4. 小程序场景

## 3.4.1. 开始监控钉钉小程序

本文介绍如何使用ARMS前端监控开始监控钉钉小程序，以及相关的通用配置、API方法和进阶场景。

### 背景信息

关于钉钉小程序的背景信息，请参见钉钉小程序。

### 操作步骤

操作步骤包括引入npm包并初始化、上报和设置安全域名。

1. 引入npm包并初始化。
   i. 在钉钉小程序的项目中引入alife-logger npm包，以便使用该模块来上报日志。

```
npm install alife-logger
```

ii. 将以下内容添加至*/utils*目录下的monitor.js文件中以完成初始化。

> **说明** 您可以自定义JS文件的名称和存放位置。

```
import EAppLogger from 'alife-logger/eapp';
const Monitor = EAppLogger.init({
    pid: 'xxx',
    region: "cn", // 指定应用部署的地域：中国设为cn，海外地区靠近新加坡的设为sg，靠近美国的设为us。
});
export default Monitor;
```

> **说明** 关于参数的详细配置，请参见通用SDK配置项。

2. 使用以下方法静默采集和上报PV、Error、API、性能及Health数据。

i. 在app.js中，使用 `Monitor.hookApp(options)` 方法静默捕获Error类日志。其中的options即为App层相应的Object配置。

```
import Monitor from '/util/monitor';
App(Monitor.hookApp({
  onError(err) {
      console.log('进入onError:', err);
  },
  onLaunch() {
    console.log('进入onLaunch');
  },
  onShow(options) {
  },
  onHide() {
  }
}));
```

ii. 在Page的JS文件中通过 `Monitor.hookPage(options)` 方法静默上报API请求、PV、Health数据。

```
import Monitor from '/util/monitor';
// 使用hookPage后，生命周期的API会自动打点。
Page(Monitor.hookPage({
   data: {},
    onLoad(query) {
    },
    onReady() {
    // 页面加载完成。
    },
    onShow() {
    },
    onLoad(query) {
    },
    onHide() {
    },
    onUnload() {
    }
}));
```

3. 设置安全域名。
   - 如果region设为 `cn` ，则将arms-retcode.aliyuncs.com添加到HTTP安全域名。
   - 如果region设为 `sg` ，则将arms-retcode-sg.aliyuncs.com添加到HTTP安全域名。
   - 如果region设为 `us` ，则将arms-retcode-us.aliyuncs.com添加到HTTP安全域名。

## API方法：静默打点基础API

| 方法 | 参数 | 备注 | 示例使用场景 |
| --- | --- | --- | --- |
| hookApp | {} | 请传入原有的App参数。 | 在App的生命周期中自动打点。 |
| hookPage | {} | 请传入原有的Page参数。 | 在Page的生命周期中自动打点。 |

> **说明** 小程序监控项目如需使用hookApp、hookPage嵌入生命周期打点，必须符合标准小程序关于App和Page的规范，即App层有onError，Page层有onShow、onHide、onUnload。使用方法示例，请参见操作步骤。

## API方法：其他设置API

| 方法 | 参数 | 备注 |
| --- | --- | --- |
| setCommonInfo | {[key: string]: string;} | 设置日志基础字段，可用于灰度发布等场景。 |
| setConfig | {[key: string]: string;} | 设置config字段，具体操作，请参见SDK配置项参数。 |
| pageShow | {} | 触发Page Show，发送PV数据。 |
| pageHide | {} | 触发Page Hide，发送Health数据。 |
| error | String/Object | 错误日志打点。 |
| api | 请参见API参考 | API类日志上报。 |
| sum/avg | String | 自定义求和、求均值日志上报。 |

## 高级使用方法

当基础使用方法无法满足需求时，请参见以下进阶场景。

- 手动上报API相关信息（不采用静默上报方式）
  i. 将disableHook设为 `true` ，不静默上报dd.httpRequest请求的日志。
  ii. 手动调用 `api()` 方法上报API相关信息。
- 取消静默上报并改为手动打点
  i. 在App和Page对应的JS文件中不再使用 `hookApp()` 、 `hookPage()` 方法。
  ii. 如需发送当前页面的PV数据，则在Page的 `onShow()` 方法下调用 `pageShow()` 方法。

> **说明** 请勿与 `hookPage()` 方法同时使用此方法，否则会造成PV类日志重复上报。

```
import Monitor from '/util/monitor';
Page({
    onShow: function() {
        Monitor.pageShow();
    }
})
```

iii. 如需发送当前页面的Health类数据，统计当前页面的健康度和页面停留时间，则在Page的 `onHide()` 和 `onUnload()` 方法下调用 `pageHide()` 方法。

> **说明** 请勿与 `hookPage()` 方法同时使用此方法，否则会造成日志重复上报。

```
import Monitor from '/util/monitor';
  Page({
      onHide: function() {
          Monitor.pageHide();
      },
      onUnload: function() {
          Monitor.pageHide();
      }
      ...
  })
```

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `false` |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

## 相关文档

- 什么是ARMS前端监控?
- 开始监控支付宝小程序
- 开始监控微信小程序
- 开始监控其他类别小程序

# 3.4.2. 开始监控支付宝小程序

本文介绍如何使用ARMS前端监控开始监控支付宝小程序，以及相关的通用配置、API方法和进阶场景。

## 背景信息

关于支付宝小程序的背景信息，请参见支付宝小程序。

## 基础使用方法

基础方法包含引入npm包并初始化、上报和设置安全域名这三个步骤：

1. 引入npm包并初始化：
   i. 在支付宝小程序的项目中引入 `alife-logger` npm包，以便使用该模块来上报日志。

```
npm install alife-logger
```

   ii. 将以下内容添加至 /utils目录下的monitor.js文件中以完成初始化。

> **说明** 您可以自定义JS文件的名称和存放位置。

```
import AlipayLogger from 'alife-logger/alipay';
const Monitor = AlipayLogger.init({
    pid: 'xxx',
    region: "cn", // 指定应用部署的地域：中国设为cn，中国以外地域设为sg。
});
export default Monitor;
```

> **说明** 关于参数的详细配置，请参见通用SDK配置项。

2. 使用以下方法静默采集PV、Error、API、性能及Health数据：

i. 在app.js中，使用**Monitor.hookApp(options)**方法静默捕获Error类日志。其中的**options**即为App层相应的Object配置。

```
import Monitor from '/util/monitor';
App(Monitor.hookApp({
  onError(err) {
      console.log('进入onError:', err);
  },
  onLaunch() {
    console.log('进入onLaunch');
  },
  onShow(options) {
  },
  onHide() {
  }
}));
```

ii. 在page的JS文件中通过**Monitor.hookPage(options)**方法静默上报API请求、PV、Health数据。

```
import Monitor from '/util/monitor';
// 使用hookPage后，生命周期的API会自动打点。
Page(Monitor.hookPage({
   data: {},
    onLoad(query) {
    },
    onReady() {
    // 页面加载完成。
    },
    onShow() {
    },
    onLoad(query) {
    },
    onHide() {
    },
    onUnload() {
    }
}));
```

3. 设置安全域名：
   - 如果**region**设为 `cn` ，则将arms-retcode.aliyuncs.com添加到HTTP安全域名。
   - 如果**region**设为 `sg` ，则将arms-retcode-sg.aliyuncs.com添加到HTTP安全域名。

## API方法：静默打点基础API

| 方法 | 参数 | 备注 | 示例使用场景 |
| --- | --- | --- | --- |
| hookApp | {} | 请传入原有的App参数。 | 在App的生命周期中自动打点。 |
| hookPage | {} | 请传入原有的Page参数。 | 在Page的生命周期中自动打点。 |

> **说明** 小程序监控项目如需使用hookApp、hookPage嵌入生命周期打点，必须符合标准小程序关于App和Page的规范，即App层有**onError**，Page层有**onShow**、**onHide**、**onUnload**。使用方法示例，请参见基础使用方法。

## API方法：其他设置API

| 方法 | 参数 | 备注 |
| --- | --- | --- |
| setCommonInfo | {[key: string]: string;} | 设置日志基础字段，可用于灰度发布等场景。 |
| setConfig | {[key: string]: string;} | 设置config字段，具体操作，请参见SDK配置项参数。 |
| pageShow | {} | Page Show打点，发送PV数据。 |
| pageHide | {} | Page Hide打点，发送Health数据。 |
| error | String/Object | 错误日志打点。 |
| api | 请参见API参考 | API类日志上报。 |
| sum/avg | String | 自定义求和、求均值日志上报。 |

## 进阶场景

当基础使用方法无法满足需求时，请参见以下进阶场景：

- 手动上报API相关信息（不采用静默上报方式）：
  i. 将**disableHook**设为 `true` ，不静默上报**my.httpRequest**请求的日志。
  ii. 手动调用**api()**方法上报API相关信息。
- 取消静默上报并改为手动打点：
  i. 在App和Page对应的JS文件中不再使用hookApp、hookPage方法。
  ii. 如需发送当前页面的PV数据，则在Page的**onShow**方法下调用**pageShow()**方法。

  > **说明** 请勿与**hookPage()**方法同时使用此方法，否则会造成PV类日志重复上报。

  ```
  import Monitor from '/util/monitor';
  Page({
      onShow: function() {
          Monitor.pageShow();
      }
  })
  ```

  iii. 如需发送当前页面的Health类数据，统计当前页面的健康度和页面停留时间，则在Page的**onHide**和**onUnload**方法下调用**pageHide()**方法。

  > **说明** 请勿与**hookPage()**方法同时使用此方法，否则会造成日志重复上报。

```
import Monitor from '/util/monitor';
  Page({
      onHide: function() {
          Monitor.pageHide();
      },
      onUnload: function() {
          Monitor.pageHide();
      }
      ...
  })
```

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `false` |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

## 相关文档

- 前端监控接入概述
- 开始监控钉钉小程序

- 开始监控微信小程序
- 开始监控其他类别小程序

# 3.4.3. 开始监控微信小程序

本文介绍如何使用ARMS前端监控开始监控微信小程序，以及相关的通用配置、API方法和进阶场景。

## 背景信息

关于微信小程序的背景信息，请参见微信小程序。

## 基础使用方法

基础方法包含获取微信小程序监控SDK并初始化、上报和设置安全域名这三个步骤：

1. 获取微信小程序监控SDK并初始化：
   i. 在微信小程序*/utils*目录下新建*wxLogger.js*文件，并将JS文件的内容复制并粘贴至新建的*wxLogger.js*文件中。
   ii. 在*/utils*目录下新建*monitor.js*文件，并将以下内容添加至新建的*monitor.js*文件中以完成初始化。

   > **说明** 您可以自定义JS文件的名称和存放位置。

   - 如果项目使用node module (require) 方式集成，则添加以下内容：

   ```
   const WXLogger = require('./wxLogger.js');
       const Monitor = WXLogger.init({
           pid: 'xxx',
           region: 'cn',     // 指定应用部署的地域：中国设为cn，海外地区靠近新加坡的设为sg。
       });
       export default Monitor;
   ```

   - 如果项目使用ES module (import) 方式集成，则添加以下内容：

   ```
   import WXLogger from './wxLogger.js';
       const Monitor = WXLogger.init({
           pid: 'xxx',
           region: 'cn',     // 指定应用部署的地域：中国设为cn，海外地区靠近新加坡的设为sg。
       });
       export default Monitor;
   ```

   > **说明** 关于参数的详细配置，请参见通用SDK配置项。

2. 使用以下方法静默采集PV、Error、API、性能及Health数据：

i. 在app.js中，使用**Monitor.hookApp(options)**方法静默捕获Error类日志。其中的**options**即为App层相应的Object配置。

```
import Monitor from '/util/monitor';
App(Monitor.hookApp({
  onError(err) {
      console.log('进入onError:', err);
  },
  onLaunch() {
    console.log('进入onLaunch');
  },
  onShow(options) {
  },
  onHide() {
  }
}));
```

ii. 在page的JS文件中通过**Monitor.hookPage(options)**方法静默上报API请求、PV、Health数据。

```
import Monitor from '/util/monitor';
// 使用hookPage后，生命周期的API会自动打点。
Page(Monitor.hookPage({
   data: {},
    onLoad(query) {
    },
    onReady() {
    // 页面加载完成。
    },
    onShow() {
    },
    onLoad(query) {
    },
    onHide() {
    },
    onUnload() {
    }
}));
```

3. 设置安全域名：
   - 如果**region**设为 `cn` ，则将 `https://arms-retcode.aliyuncs.com` 添加到Request合法域名。
   - 如果**region**设为 `sg` ，则将 `https://arms-retcode-sg.aliyuncs.com` 添加到Request合法域名。

## API方法：静默打点基础API

| 方法 | 参数 | 备注 | 示例使用场景 |
| --- | --- | --- | --- |
| hookApp | {} | 请传入原有的App参数。 | 在App的生命周期中自动打点。 |
| hookPage | {} | 请传入原有的Page参数。 | 在Page的生命周期中自动打点。 |

> **说明** 小程序监控项目如需使用hookApp、hookPage嵌入生命周期打点，必须符合标准小程序关于App和Page的规范，即App层有**onError**，Page层有**onShow**、**onHide**、**onUnload**。使用方法示例，请参见基础使用方法。

## API方法：其他设置API

| 方法 | 参数 | 备注 |
| --- | --- | --- |
| setCommonInfo | {[key: string]: string;} | 设置日志基础字段，可用于灰度发布等场景。 |
| setConfig | {[key: string]: string;} | 设置config字段，具体操作，请参见SDK参考。<br>**说明** 对于setConfig方法，小程序场景不支持配置uid，您可以使用setUsername代替uid标识用户。 |
| pageShow | {} | Page Show打点，发送PV数据。 |
| pageHide | {} | Page Hide打点，发送Health数据。 |
| error | String/Object | 错误日志打点。 |
| api | 请参见API参考 | API类日志上报。 |
| sum/avg | String | 自定义求和、求均值日志上报。 |

## 进阶场景

当基础使用方法无法满足需求时，请参见以下进阶场景：

- 手动上报API相关信息（不采用静默上报方式）：
  i. 将**disableHook**设为 `true` ，不静默上报**wx.request**请求的日志。
  ii. 手动调用**api()**方法上报API相关信息。
- 取消静默上报并改为手动打点：
  i. 在App和Page对应的JS文件中不再使用hookApp、hookPage方法。
  ii. 如需发送当前页面的PV数据，则在Page的**onShow**方法下调用**pageShow()**方法。

> **说明** 请勿与**hookPage()**方法同时使用此方法，否则会造成PV类日志重复上报。

```
import Monitor from '/util/monitor';
Page({
    onShow: function() {
        Monitor.pageShow();
    }
})
```

iii. 如需发送当前页面的Health类数据，统计当前页面的健康度和页面停留时间，则在Page的onHide和onUnload方法下调用pageHide()方法。

> **说明** 请勿与 hookPage()方法同时使用此方法，否则会造成日志重复上报。

```
import Monitor from '/util/monitor';
  Page({
      onHide: function() {
          Monitor.pageHide();
      },
      onUnload: function() {
          Monitor.pageHide();
      }
      ...
  })
```

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `false` |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

## 相关文档

- 前端监控接入概述
- 开始监控钉钉小程序
- 开始监控支付宝小程序
- 开始监控其他类别小程序

# 3.4.4. 开始监控其他类别小程序

本文介绍如何使用ARMS前端监控开始监控其他类别的小程序，即除钉钉小程序、支付宝小程序和微信小程序之外的各类符合标准规范的小程序，以及相关的通用配置、API方法和进阶场景。

## 基础使用方法

基础方法包含引入npm包并初始化、上报和设置安全域名这三个步骤：

1. 引入npm包并初始化：
   i. 在小程序的项目中引入 `alife-logger` npm包，以便使用该模块来上报日志。

   ```
   npm install alife-logger
   ```

   ii. 将以下内容添加至/utils目录下的monitor.js文件中以完成初始化。

   > **说明** 您可以自定义JS文件的名称和存放位置。

```
import MiniProgramLogger from 'alife-logger/miniprogram';
const Monitor = MiniProgramLogger.init({
    pid: 'xxx',
    uid: 'userxxx', // 设置用户uid，用于统计UV信息。
    region: 'cn', // 指定应用部署的地域：中国设为cn，中国以外地域设为sg。默认值为cn。
    // 基础小程序监控需要手动传入RPC。请按照实际业务写实现方法，以下示例为钉钉E应用中的调用方式。
    sendRequest: (url, resData) => {
          // 此部分由业务方配置，支持Get/Post上报。
        // demo in dingding
        var method = 'GET';
        var data;
        if (resData) {
            method = 'POST';
            data = JSON.stringify(resData);
        }
        dd.httpRequest({
            url: url,
            method: method,
            data: data,
            fail: function (error) {
                //...
            }
        });
    },
     // 手动传入获取当前页面路径的方法。请按照实际业务写实现方法，以下示例为钉钉E应用中的调用方式。
     getCurrentPage: () => {
          // 此部分由业务方配置。
        if (typeof getCurrentPages !== 'undefined' && typeof getCurrentPages === 'function') {
            var pages = (getCurrentPages() || []);
            var pageLength = pages.length;
            var currPage = pages[pageLength - 1];
            return (currPage && currPage.route) || null;
        }
     }
});
export default Monitor;
```

> **说明** 关于参数的详细配置，请参见通用SDK配置项。

2. 上报日志：

i. 在app.js中，使用以下两种方法之一上报日志：

- 使用**Monitor.hookApp(options)**方法静默捕获Error类日志。其中的**options**即为App层相应的Object配置。

```
import Monitor from '/utils/monitor';
    App(Monitor.hookApp({
      onError(err) {
        console.log('进入onError:', err);
      },
      onLaunch() {
        console.log('进入onLaunch');
      }
      onShow(options) {
      },
      onHide() {
      }
    }));
```

- 使用**Monitor.error(err)** 方法手动上报Error类日志。

```
import Monitor from '/utils/monitor';
    App({
      onError(err) {
          Monitor.error(err);
        console.log('进入onError:', err);
      },
      onLaunch() {
        console.log('进入onLaunch');
      }
      onShow(options) {
      },
      onHide() {
      }
    });
```

ii. 在page的JS文件中，使用以下两种方法之一上报日志：

- 使用Monitor.hookPage(options)方法静默上报PV、Health数据。

> **说明** 此方法不支持静默上报API请求。

```
import Monitor from '/utils/monitor';
    Page(Monitor.hookPage({
       data: {},
        onLoad(query) {
        },
        onReady() {
            // 页面加载完成。
        },
        onShow() {
        },
        onLoad(query) {
        },
        onHide() {
        },
        onUnload() {
        },
        onTitleClick() {
            /**
             * 统计打点数据，自定义打点。
             * @desc
             */
            Monitor.sum('titleClick');
        }
    }));
```

- 调用API方法主动打点。

> **说明** 关于API方法的详细信息，请参见API方法。

```
import Monitor from './util/monitor';
    Page({
       data: {},
        onShow() {
            Monitor.pageShow();
        },
        onHide() {
            Monitor.pageHide();
        },
        onUnload() {
            Monitor.pageHide();
        },
        onTitleClick() {
            /**
             * 统计打点数据，自定义打点。
             * @desc
             */
            Monitor.sum('titleClick');
        }
    });
```

3. 设置安全域名：
   - 如果region设为 `cn` ，则将 `https://arms-retcode.aliyuncs.com` 添加到合法域名。
   - 如果region设为 `sg` ，则将 `https://arms-retcode-sg.aliyuncs.com` 添加到合法域名。

## API方法

| 方法 | 参数 | 备注 |
| --- | --- | --- |
| hookApp | {} | 请传入原有的App参数。可用于在App的生命周期中自动打点。 |
| hookPage | {} | 请传入原有的Page参数。可用于在Page的生命周期中自动打点。 |
| setCommonInfo | {[key: string]: string;} | 设置日志基础字段，可用于灰度发布等场景。 |
| setConfig | {[key: string]: string;} | 设置config字段，具体操作，请参见SDK配置项参数。 |
| pageShow | {} | Page Show打点，发送PV数据。 |
| pageHide | {} | Page Hide打点，发送Health数据。 |
| error | String/Object | 错误日志打点。 |
| api | 请参见API参考 | API类日志上报。 |

| 方法 | 参数 | 备注 |
| --- | --- | --- |
| sum/avg | String | 自定义求和、求均值日志上报。 |

> **说明** 小程序监控项目如需使用hookApp、hookPage嵌入生命周期打点，必须符合标准小程序关于App和Page的规范，即App层有**onError**，Page层有**onShow**、**onHide**、**onUnload**。使用方法示例，请参见基础使用方法。

大部分日志上报API与Web端监控SDK一致，其他API的使用方法如下：

- 如需发送当前页面的PV数据，则在Page的**onShow**方法下调用**pageShow()** 方法。

  > **说明** 请勿与**hookPage()**方法同时使用此方法，否则会造成PV类日志重复上报。

  ```
  import Monitor from '/util/monitor';
  Page({
      onShow: function() {
          Monitor.pageShow();
      }
  })
  ```

- 如需发送当前页面的Health类数据，统计当前页面的健康度和页面停留时间，则在Page的**onHide**和**onUnload**方法下调用**pageHide()**方法。

  > **说明** 请勿与**hookPage()**方法同时使用此方法，否则会造成日志重复上报。

  ```
  import Monitor from '/util/monitor';
    Page({
        onHide: function() {
            Monitor.pageHide();
        },
        onUnload: function() {
            Monitor.pageHide();
        }
        ...
    })
  ```

## 进阶场景

当基础使用方法无法满足需求时，请参见以下进阶场景：

- 设置**uid**（主要用于统计UV信息）
  - 如果在监控SDK初始化之前，能够获得与用户有关的信息，则可以直接设置**uid**。
  - 如果在监控SDK初始化之前，无法获得与用户有关的信息，则可以在应用onShow前获取用户信息，然后通过**setCommonInfo({uid: 'xxx'})**； 设置**uid**。
- 设置小程序相关的公共信息
  请使用**setCommonInfo**方法设置小程序相关的公共信息。ARMS前端监控会对以下字段进行统计分析：
  - sr：屏幕尺寸
  - vp：浏览器窗口可见区域

- dpr：屏幕像素比
- ul：文档语言
- dr：文档Refer
- ct：网络连接类型（例如WiFi或3G）

> **警告** 切勿使用setCommonInfo方法设置过多字段，否则可能超出请求的限制长度，从而导致请求失败。

## 通用SDK配置项

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求。以下是适用于本文场景的通用配置项。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | 否 | 由SDK自动生成 |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否 | `false` |

ARMS前端监控还提供了更多SDK配置项，可满足进一步的需求。更多信息，请参见SDK参考。

## 相关文档

- 前端监控接入概述
- 开始监控钉钉小程序
- 开始监控微信小程序
- 开始监控支付宝小程序

# 4.控制台功能

## 4.1. 前端监控实时大屏

通过ARMS前端监控实时大屏，您可以一次性查看被监控应用的所有关键实时监控数据。

### 功能入口

1. 登录ARMS控制台，并在左侧导航栏中单击**前端监控**。
2. 在**前端监控**页面上，单击应用名称。
3. 在应用的**总览**页面上，单击右上角的**实时大屏**。
4. 在监控大屏页面上，查看实时更新的应用前端监控数据，例如JS错误率、API请求成功率等。

### 功能介绍

ARMS前端监控实时大屏上包含被监控应用的所有关键实时监控数据，适合用于在大屏幕上展示。

> **说明** 实时大屏上的监控数据最快每分钟更新一次。

### 可用操作

- 查看实时大屏上的各项监控数据。
- 将鼠标悬停在统计图的曲线上，可显示各时间点对应的统计结果。
- 单击右上角的**全屏显示**，以全屏模式观看实时大屏。

### 字段说明

ARMS前端监控实时大盘所包含的字段含义说明如下。

- JS错误率
  - JS错误率：JavaScript出错的比例。
  - 相比昨日均值：与昨日JavaScript平均错误率相比上升或下降的比例。
  - 统计图：最近1小时的JavaScript错误率曲线。
  - 页面错误率排行：JavaScript错误率排行。
- 最近24小时告警
  - 告警数量：告警的数量。
  - 最近告警信息：最近24小时来自前端监控报警的告警信息。
- PV/UV
  - 今日PV：被监控应用的今日PV值。
  - 今日UV：被监控应用的今日UV值。
  - 同比昨日：与昨日PV/UV相比上升或下降的比例。
  - 统计图：最近1小时的PV/UV曲线。
  - 统计表：PV/UV排名前5的地域及对应PV/UV值。
  - 高访问量Top 5：访问量最多的5个服务。
- API请求成功率
  - API请求成功率：API请求成功的比例。

- 相比昨日均值：与昨日API请求成功率相比上升或下降的比例。
- 统计图：最近1小时的API请求成功率曲线。
- API成功率排行：API请求成功率排行。

- 访问速度
  - 访问速度：首次渲染耗时，单位为毫秒（ms）。
  - 相比昨日均值：与昨日访问速度均值相比上升或下降的比例。
  - 统计图：最近1小时的访问速度曲线。
  - 低访问速度Top 5：访问速度最低的5个服务。

### 相关文档

- 页面访问速度
- JS错误诊断
- API请求
- 快速创建ARMS报警（旧版）

# 4.2. 页面访问速度

本文介绍ARMS前端监控的访问速度页面所包含的功能。
将应用成功接入ARMS前端监控后，您可以在**访问速度**页面上查看应用的以下性能数据：

> **说明** 您可以手动上报自定义的性能指标，例如自定义首次渲染时间、首次可交互时间或其他性能指标。具体操作，请参考API参考。

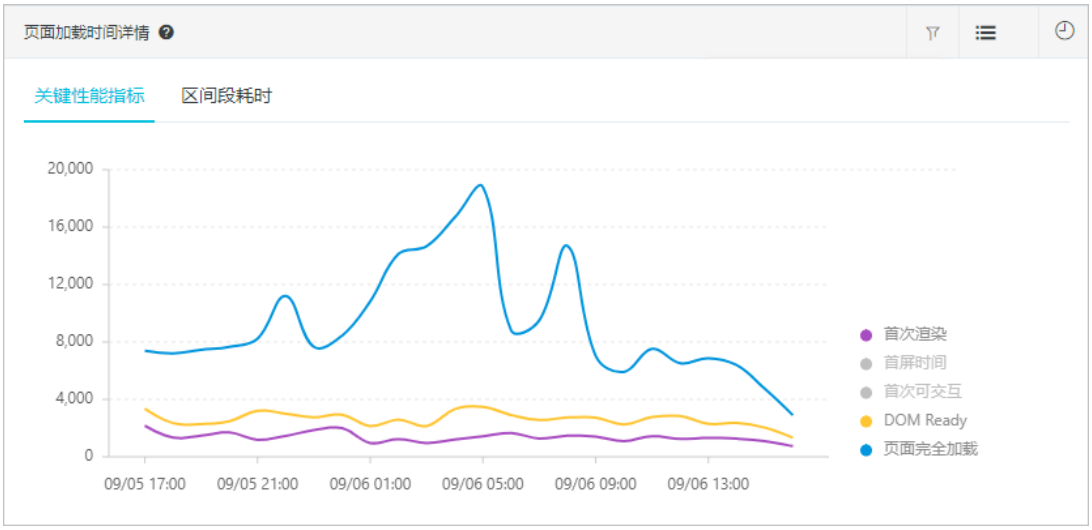
- 页面加载时间详情
- 页面加载瀑布图
- 性能分布
- 慢页面会话追踪
- 页面加载分布情况
- 手动上报自定义的性能指标

### 功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击**前端监控**，在**前端监控**页面上单击目标应用名称。
3. 在左侧导航栏选择**应用 > 访问速度**。

在访问速度页面左侧的**页面访问速度排行**区域，可按页面首次渲染时间指标排序和访问量指标排序，单击上箭头或下箭头可改变排序顺序。

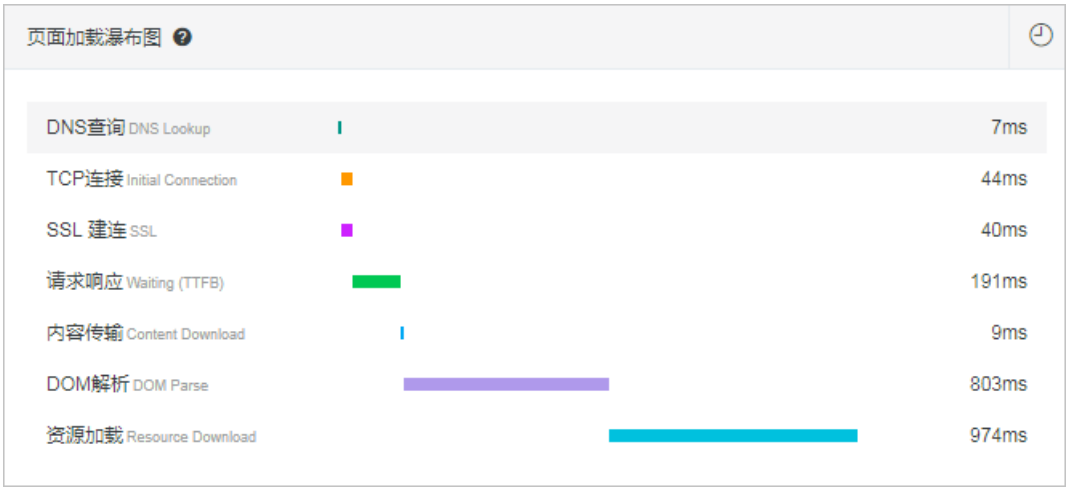

## 页面加载时间详情




> **说明**
>
> - 折线图中数据是按照指定时间段内该指标数据的平均值展示的。平均值可以体现一段时间内性能的均值情况，但容易受到极值的影响，例如某次用户访问网络很差，导致整体页面加载非常慢，就会直接拉高平均值的数据。您可以单击右上角的图标，使用去极值功能去除极值，避免极值影响性能的整体趋势。
> - 如果折线图中数据骤增，您可以通过性能样本分布、慢页面会话追踪模块定位问题。

## 页面加载瀑布图

此瀑布图按照页面加载的顺序，直观地展示了各阶段的耗时情况。图中的数据是指定时间段内指定指标的平均值数据。如需优化性能，可结合具体阶段采取针对性的方法。

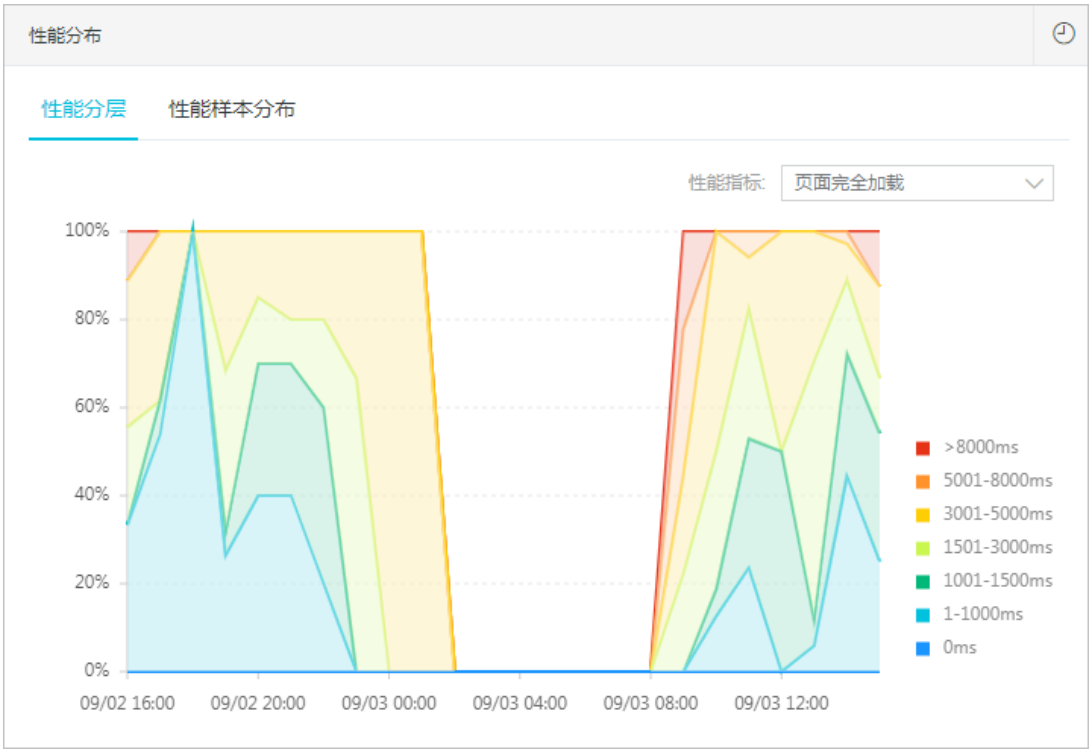



## 性能分布

此模块非常直观地展示了页面性能分布情况。

在**性能分层**页签上，您可以看到以时间为横轴的堆积折线图，了解各时间点上的性能分布情况。

性能分层




在**性能样本分布**页签上，您可以看到页面加载时间在指定时间区间内的样本占比。例如，有多少比例的页面能够在1秒内打开，或者长尾访问用户的样本占比。

性能样本分布


## 慢页面会话追踪

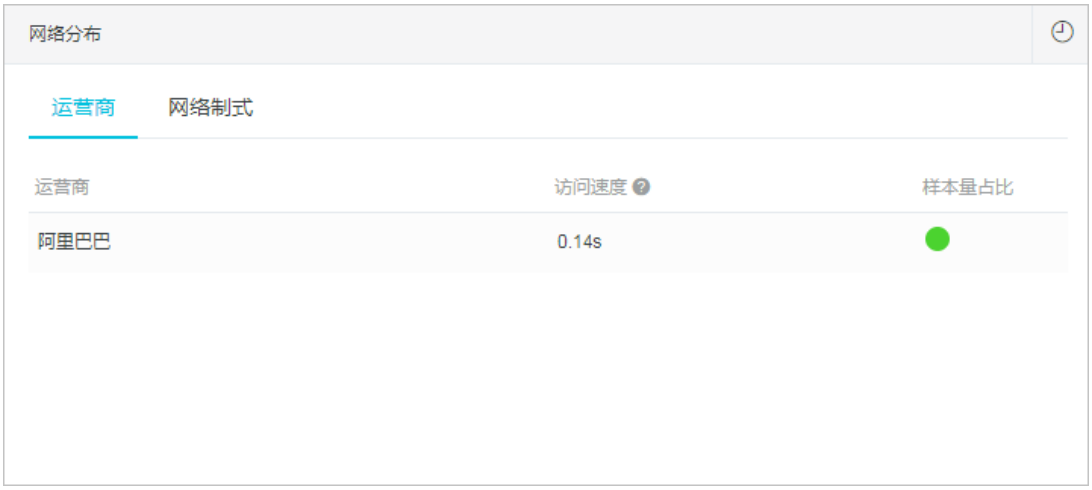
慢会话追踪功能可提供页面加载过程中静态资源加载的性能瀑布图，帮助您根据页面性能数据详细了解页面资源加载情况，并快速定位性能瓶颈。更多信息，请参见会话追踪。


## 页面加载分布情况

页面的加载是在用户端的浏览器上进行的，加载耗时与地理位置、网络状况、浏览器或运营商等因素有关，所以前端监控提供地理分布、终端分布、网络分布、版本分布等统计数据，以帮助您更好地定位性能瓶颈。

终端分布


网络分布




版本分布


## 性能指标含义

Web端关键性能指标

| 上报字段 | 描述 | 计算公式 | 备注 |
| --- | --- | --- | --- |
| FMP（First Meaningful Paint） | 首屏时间 | 参见FMP技术实现方案 | 无 |

| 上报字段 | 描述 | 计算公式 | 备注 |
| --- | --- | --- | --- |
| FPT（First Paint Time） | 首次渲染时间（白屏时间） | responseEnd - fetchStart | 从请求开始到浏览器开始解析第一批HTML文档字节的时间差。 |
| TTI（Time to Interact） | 首次可交互时间 | domInteractive - fetchStart | 浏览器完成所有HTML解析并且完成DOM构建，此时浏览器开始加载资源。 |
| Ready | HTML加载完成时间， 即DOM Ready时间。 | domContentLoadEventEnd - fetchStart | 如果页面有同步执行的JS，则同步JS执行时间=Ready-TTI。 |
| Load | 页面完全加载时间 | loadEventStart - fetchStart | Load=首次渲染时间+DOM解析耗时+同步JS执行+资源加载耗时。 |
| FirstByte | 首包时间 | responseStart - domainLookupStart | 无 |

区间段耗时字段含义

| 上报字段 | 描述 | 计算公式 | 备注 |
| --- | --- | --- | --- |
| DNS | DNS查询耗时 | domainLookupEnd - domainLookupStart | 无 |
| TCP | TCP连接耗时 | connectEnd - connectStart | 无 |
| TTFB（Time to First Byte） | 请求响应耗时 | responseStart - requestStart | TTFB有多种计算方式，ARMS采用的标准，请参见Google Development定义。 |
| Trans | 内容传输耗时 | responseEnd - responseStart | 无 |
| DOM | DOM解析耗时 | domInteractive - responseEnd | 无 |
| Res | 资源加载耗时 | loadEventStart - domContentLoadedEventEnd | 表示页面中的同步加载资源。 |
| SSL | SSL安全连接耗时 | connectEnd - secureConnectionStart | 只在HTTPS下有效。 |

# 4.3. 会话追踪

当您定位问题需要问题上下文以便还原出错现场时，可以使用ARMS前端监控的会话追踪功能。此功能根据用户名称或用户ID实现全链路追踪，复现用户访问的行为轨迹，包括页面加载、接口请求、JS错误和用户操作等，帮助有效定位和分析出错原因。

## 功能入口

1. 登录ARMS控制台。

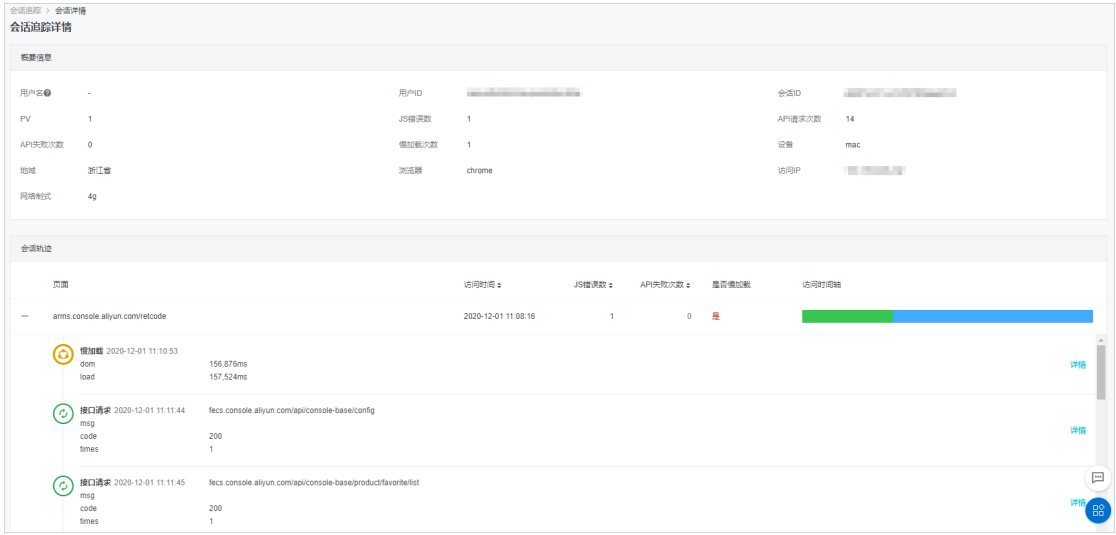
2. 在左侧导航栏选择**前端监控 > 前端列表**，然后单击目标应用名称。
3. 在左侧导航栏选择**应用 > 会话追踪**。

## 查看会话详情

1. （可选）在**会话追踪**页面，输入**用户名**或**用户ID**，然后单击**搜索**查找到对应会话。

   > 说明
   > - 用户名称的设置方法，请参见SDK参考。
   > - 单击**展开**可以添加会话ID、访问IP、页面地址、浏览器、浏览器版本号、网络制式和地域搜索条件。

2. 在会话列表中单击目标会话的**会话ID**，查看**会话追踪详情**页面。
3. 在**概要信息**区域，查看该会话的用户名、用户ID、会话ID、PV、JS错误数、API请求次数、API失败次数、慢加载次数、设备、地域、浏览器、访问IP和网络制式等基础信息。
4. 在**会话轨迹**区域，查看用户的访问路径。
   i. 单击目标页面左侧的+图标，可以展开该页面下的用户行为轨迹。




   ii. 单击目标用户行为右侧的**详情**，查看对应的行为详情信息，例如：API详情、慢加载详情和JS错误详情。

## 其他会话追踪入口

- 在左侧导航栏选择**应用 > 访问速度**，在**慢页面会话追踪（Top 20）**区域单击**会话Id**。
- 在左侧导航栏选择**应用 > JS错误诊断**，在**高频错误**页签单击目标错误信息右侧**操作**列**诊断**，在**用户行为回溯**区域单击右侧的**查看会话**。
- 在左侧导航栏选择**应用 > API详情**，在**API请求列表**区域单击**错误次数**，在**网络请求信息**区域单击右侧的**查看会话**。
- 在左侧导航栏选择**应用 > 访问明细**，在**日志列表**页签单击**操作**列的**查看会话**。

## 相关文档

- 慢会话追踪

# 4.4. JS错误诊断

ARMS前端监控的JS错误诊断功能可展示JS错误的基本信息和分布情况，以及回溯用户行为，帮助您快速定位错误位置。

## 功能入口

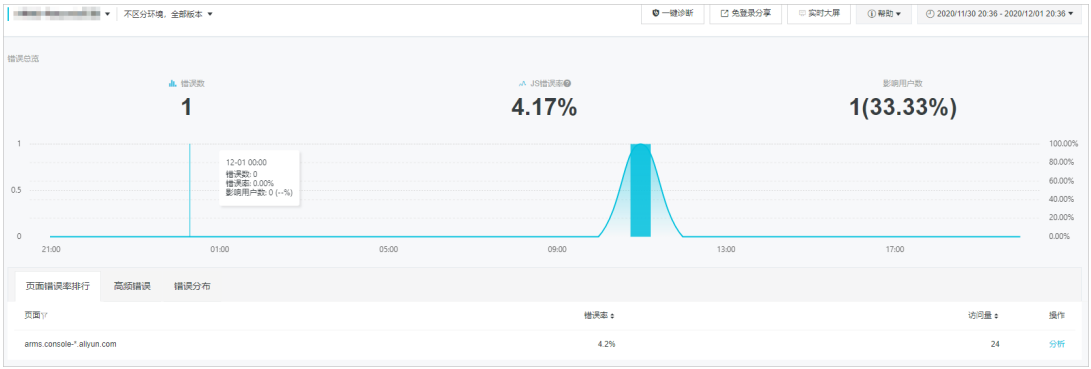
1. 登录ARMS控制台。
2. 在左侧导航栏单击**前端监控**，在**前端监控**页面上单击应用名称。
3. 在左侧导航栏选择**应用 > JS错误诊断**。
   在**JS错误诊断**页面右上角，可以设置需要查看的时间段。

## 查看应用的错误总览

**错误总览**区域可展示选定时间段内的JS错误基本统计信息和趋势，包括以下指标：

- **错误数**：选定时间段内的JS错误总数。
- **JS错误率**：选定时间段内发生过错误的PV占总PV的比例。
- **影响用户数**：JS错误影响到的用户数量和比例。

应用的错误总览




在**错误总览**区域可执行以下操作：

- 将鼠标悬浮于曲线上，曲线拐点所对应时间点的错误数、错误率、影响用户数将显示在浮层中。
- 将鼠标悬浮于曲线拐点上，当鼠标显示为手形指针时单击拐点，可打开该时间点的**异常洞察**对话框。更多信息，请参见查看异常洞察。
- 在曲线图区域内按住鼠标左键并拖动鼠标来框选其中一段，即可放大查看该段曲线。单击右上角的**重置缩放**即可还原视图。

> **说明** 在JS错误诊断页面上，默认情况下**错误总览**区域显示的是应用层面的总览信息。在**页面错误率排行**区域或**异常洞察**对话框中**页面错误率Top 5**页签单击**分析**后，展示的是对应页面的总览信息。

## 查看异常洞察

**异常洞察**对话框可显示具体时间点的JS错误情况，包括以下指标：

- **错误数**：对应时间点的JS错误总数。
- **JS错误率**：对应时间点发生过错误的PV占总PV的比例。
- **影响用户数**：JS错误影响到的用户数量和比例。
- **高频错误Top 5**：对应时间点出现次数最多的前5种JS错误，包括ARMS捕捉到的JS错误内容、错误出现次

数和影响用户数。

- **页面错误率Top 5**：对应时间点JS错误率最高的前5个页面，包括出现过JS错误的页面名称、页面的JS错误率和页面访问量。


在**异常洞察**对话框可执行以下操作：

- 单击**高频错误Top 5**页签，然后单击**操作**列中的**诊断**，进入**错误详情**页面。更多信息，请参见查看错误详情。
- 单击**页面错误率Top 5**页签，然后单击目标页面**操作**列中的**分析**，可查看该页面的错误总览视图。

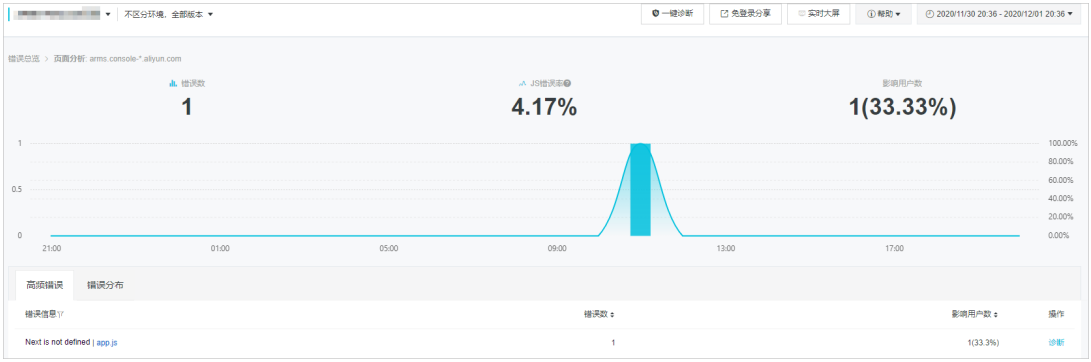
## 查看页面错误率排行

**页面错误率排行**页签可按JS错误率从高到低的顺序展示选定时间段内出现JS错误的页面，包括以下指标：

- **页面**：出现过JS错误的页面。
- **错误率**：选定时间段内在该页面发生过错误的PV占总PV的比例。
- **访问量**：页面的访问量。

单击**操作**列中的**分析**，可查看该页面的错误总览视图。

页面的错误总览




## 查看高频错误

**高频错误**页签可按出现次数从多到少的顺序展示选定时间段内的JS错误，包括以下指标：

- **错误信息**：ARMS捕捉到的JS错误内容。
- **页面**：JS错误出现的页面。
- **错误数**：JS错误出现的次数。
- **影响用户数**：JS错误影响到的用户数量和比例。

单击**操作**列中的**诊断**，进入**错误详情**页签。更多信息，请参见查看错误详情。

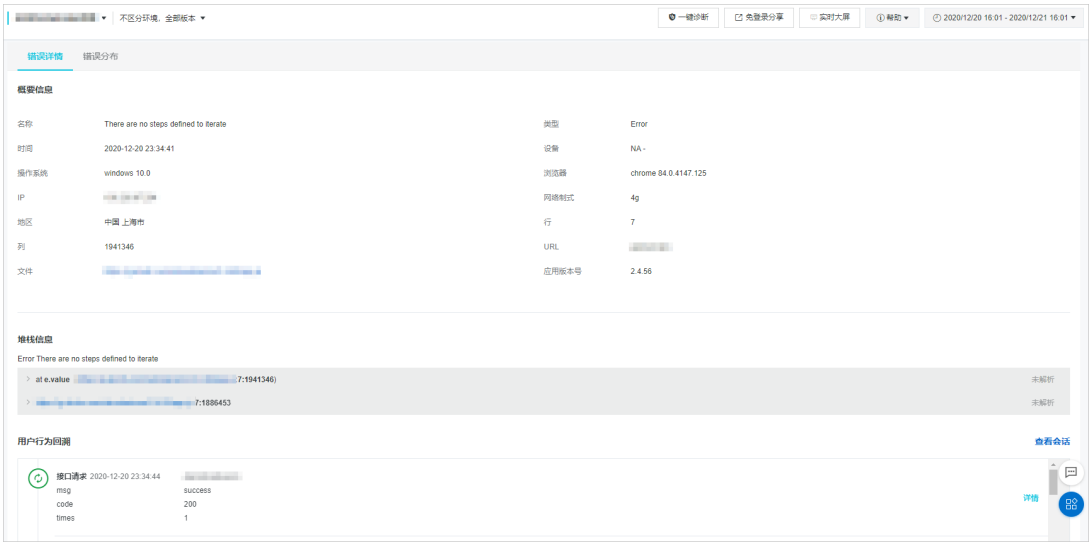
> **说明** 在JS错误诊断页面上，默认情况下**高频错误**页签显示的是应用层面的JS错误。在**页面错误率排行**区域或**异常洞察**对话框中**页面错误率Top 5**页签单击**分析**后，展示的是对应页面上的JS错误。

## 查看错误详情

**错误详情**页签可展示以下信息：

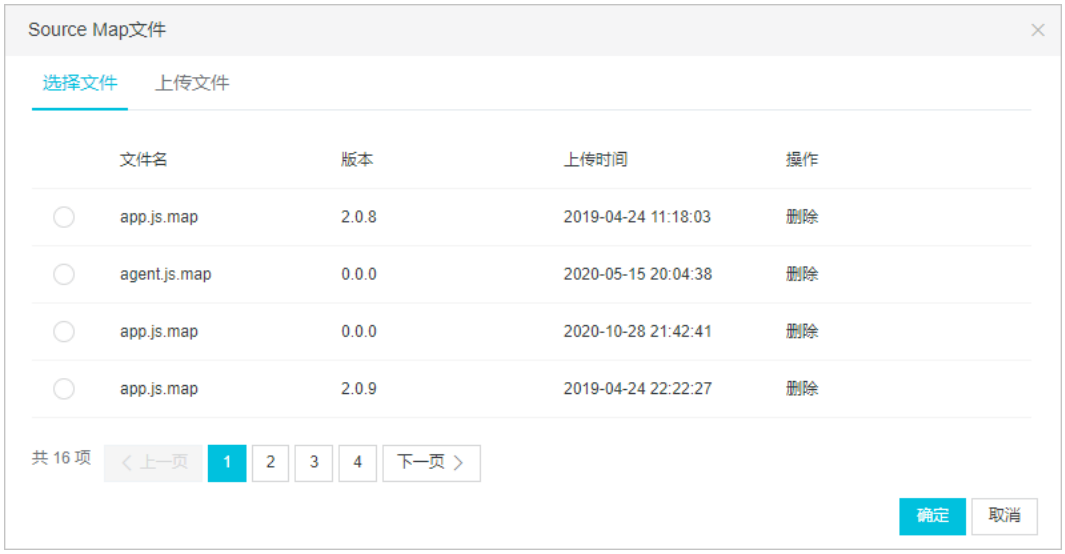
- **概要信息**
  - **名称**
  - **类型**
  - **时间**（JS错误的发现时间）
  - **设备**
  - **操作系统**
  - **浏览器**
  - **IP**
  - **网络制式**
  - **地区**
  - **行**
  - **列**
  - **URL**
  - **文件**（出现JS错误的文件路径）
  - **应用版本号**
- **堆栈信息**：与JS错误出现位置有关的信息。
- **用户行为回溯**：回溯的用户行为信息，用于还原报错现场。

JS错误详情页面

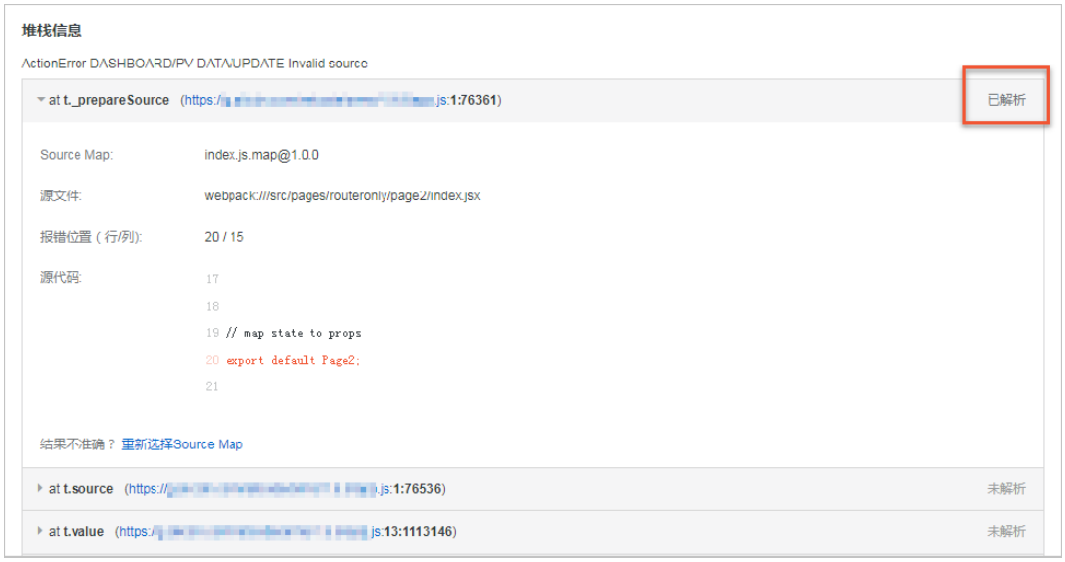



在**错误详情**页签上可执行以下操作：

- 如需确定JS错误的准确出错位置，请在**堆栈信息**区域单击一条堆栈信息左侧的三角形图标展开该行，单击**选择Source Map**，然后在**Source Map文件**对话框中选择现有的Source Map文件或上传新的Source Map文件，最后单击**确定**。




ARMS将利用Source Map文件还原准确的JS错误位置。




- 如需查看用户行为轨迹，请查看回溯用户行为区域。
- 如需查看该错误的分布情况，请单击**错误分布**页签。

## 回溯用户行为

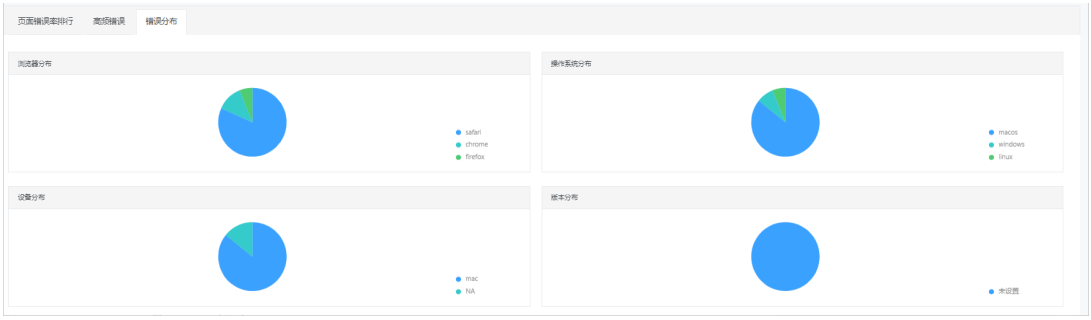
**错误详情**页签上的**用户行为回溯**区域展示用户的行为轨迹，辅助还原报错现场。


## 查看错误分布

JS错误诊断页面的**错误分布**页签可展示具体JS错误的分布情况，统计维度包括：

- **时间分布**：仅页面层面的错误分布支持。
- **浏览器分布**
- **操作系统分布**
- **设备分布**
- **版本分布**
- **地理分布**：在中国维度下按省、直辖市、自治区统计，在世界维度下按国家/地区统计。

JS错误分布页面

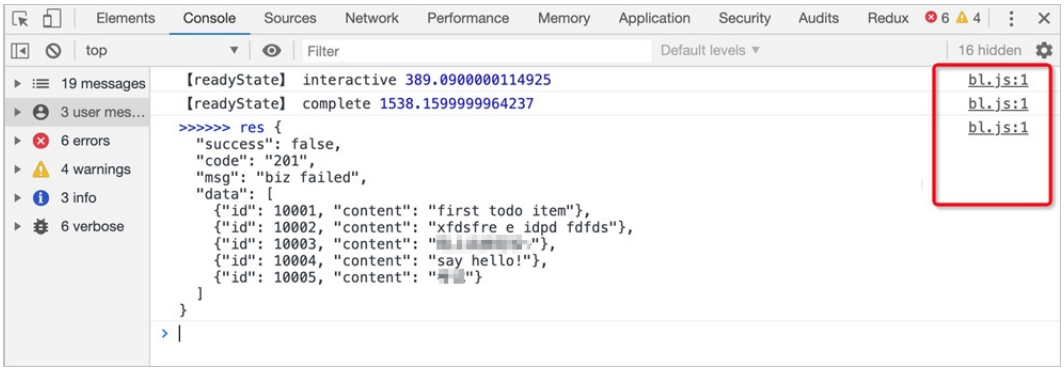



在**错误分布**页签上可执行以下操作：

- 在**时间分布**区域，将鼠标悬浮于分布图上，可以展示具体的错误数。
- 在**浏览器分布**、**操作系统分布**、**设备分布**和**版本分布**区域，将鼠标悬浮于分布图上，可以展示具体的错误数和占比情况。
- 在**地理分布**区域的**中国**或**世界**页签上，单击右侧表格中的**错误数**箭头，可切换表格的排序顺序（从正序切换为倒序，或从倒序切换为正序）。

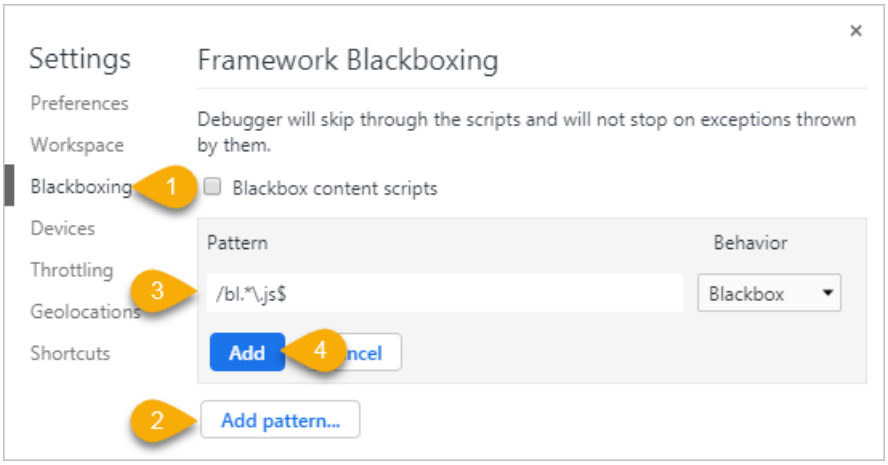
## 常见问题

- 如何开启或关闭用户行为回溯功能？
  该功能默认开启。如需关闭，请在config配置中添加SDK配置项behavior: false。SDK配置项的详细信息，请参见SDK参考。
- 开启用户行为回溯后，调试过程中通过console.log打印出的信息会定位到ARMS的SDK代码bl.js中，而不是源代码中的位置，如何解决？




造成这种现象的原因是ARMS通过重写console对象的log等方法来监控浏览器控制台打印的内容。解决方法为：

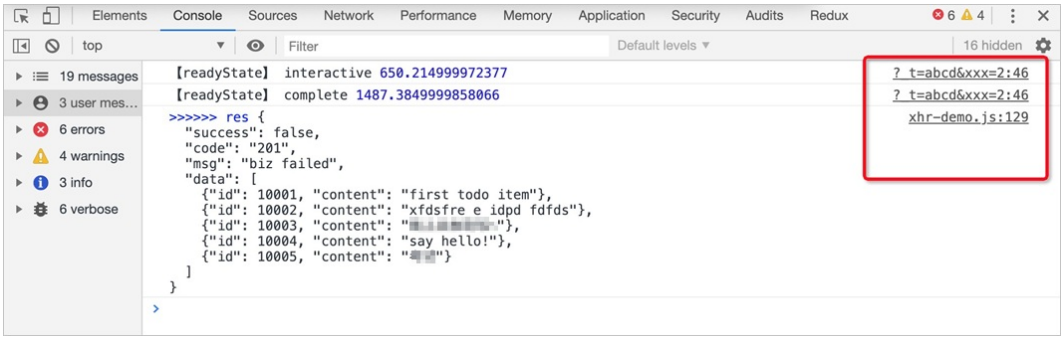
- 方法一（推荐）：设置Chrome浏览器的黑盒（Blackboxing）。
  a. 打开Chrome浏览器，按Ctrl+Shift+I打开开发者工具面板，然后单击设置图标。
  b. 在**Settings**面板左侧单击**Blackboxing**，单击**Add pattern**，在**Pattern**文本框中输入*/bl.*\.js$*，并单击**Add**。




- 方法二：使用SDK配置项behavior: false关闭用户行为回溯。

```
<script>
    ! (function ( c , b, d, a ) {
        c [a] || ( c[a] = {});
        c [a].config = {
            pid: "xxxxx",
            imgUrl: "https://arms-retcode.aliyuncs.com/r.png?",
            sendResource: true,
            enableLinkTrace: true,
            behavior: false
        };
        with(b) with(body) with(insertBefore(createElement("script"), firstChild)) setA
ttribute("crossorigin", "", src = d)
    })(window, document, "https://retcode.alicdn.com/retcode/bl.js", "__bl");
</script>
```

按照上述方法处理后，console.log打印出的信息即可定位到源代码中的位置。




## 相关文档

- SDK参考

# 4.5. API请求

API请求提供应用中每个API的调用情况，包括调用成功率、返回信息、调用成功或失败的平均耗时等。

## 功能介绍

阿里云ARMS前端监控的API请求模块，可清晰展示以下信息：

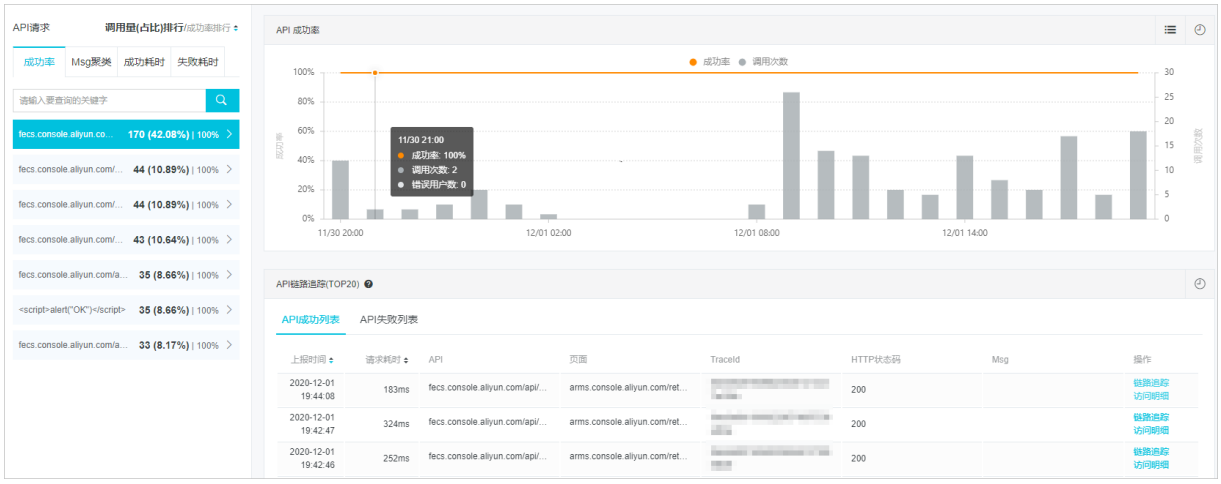
- 每个API的成功率
- API返回信息
- API接口的调用成功平均耗时
- API接口的调用失败平均耗时

此外，该模块还会展示上述统计数据在以下维度上的分布情况：

- 地理
- 浏览器
- 操作系统
- 设备
- 分辨率
- 版本

## API成功率

左侧的**成功率**页签展示的是API调用成功率排行。右侧**API成功率**区域展示的是左侧列表选中的API在指定时间范围内的调用量和调用成功率曲线，**API链路追踪（TOP 20）**区域展示的是该API调用成功或失败的链路TOP 20，单击**操作**列的**链路追踪**和**访问明细**，可以查看对应链路的调用链路详情和访问明细。

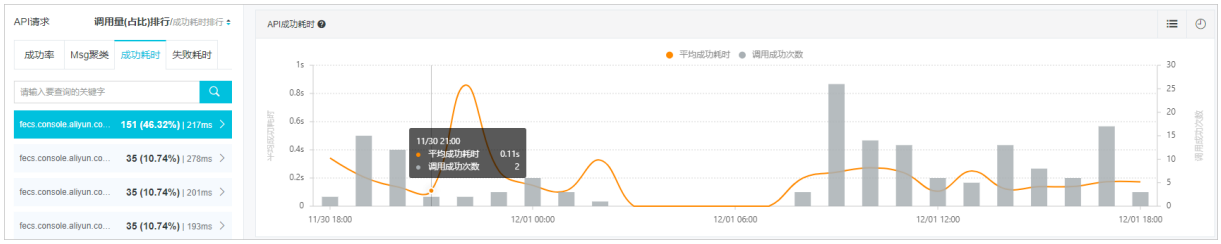



## API返回信息

左侧的**Msg聚类**页签展示的是API返回信息排行。右侧的**Msg调用详情**区域展示的是左侧列表选中的返回信息的API调用列表，按调用量降序排列。

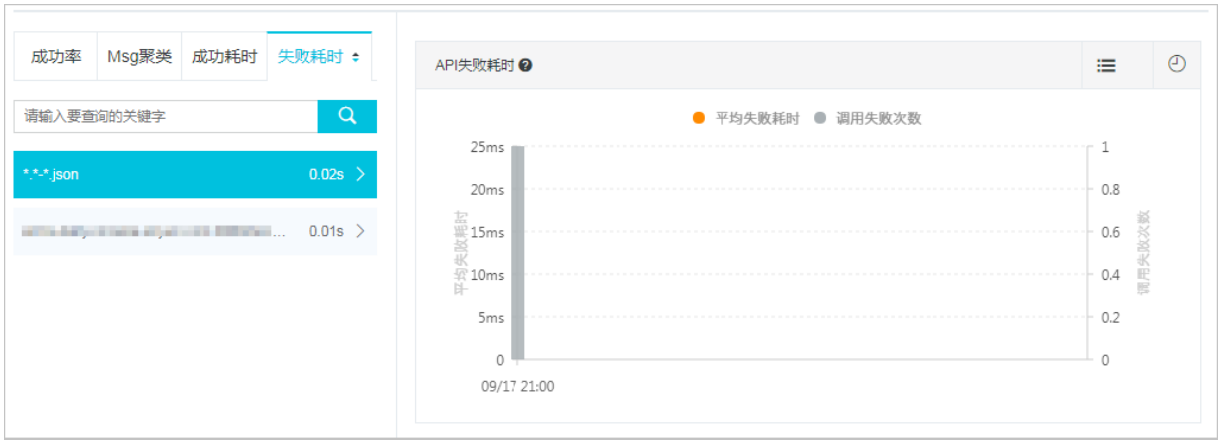

## API成功耗时

左侧的**成功耗时**页签展示的是API调用成功时的平均耗时。右侧的**API成功耗时**区域展示的是左侧列表选中的API的调用成功平均耗时曲线和调用成功次数柱形图。




## API失败耗时

左侧的**失败耗时**页签展示的是API调用失败时的平均耗时。右侧的**API失败耗时**区域展示的是左侧列表选中的API的调用失败平均耗时曲线和调用失败次数柱形图。




## 地理分布

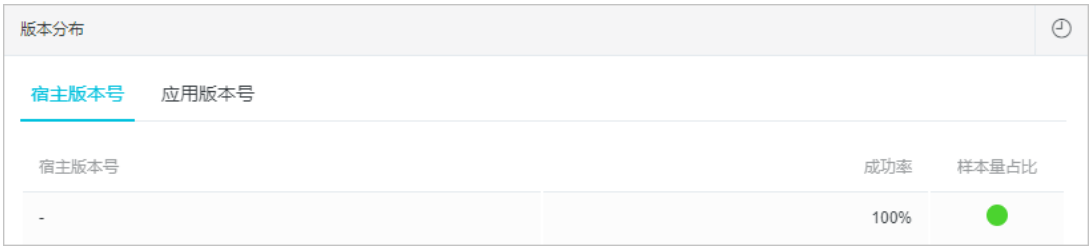
在**地理分布**模块，您可以查看上述统计信息的地理分布情况。地理分布又分为中国和世界两个维度，中国维度的单位为省，世界维度的单位为国家/地区。

## 终端分布

在**终端分布**模块中，您可以查看上述统计信息的终端分布情况。终端分布又细分为浏览器、操作系统、设备、分辨率等维度。


## 版本分布

在**版本分布**模块中，您可以查看上述统计信息的宿主版本号和应用版本号。

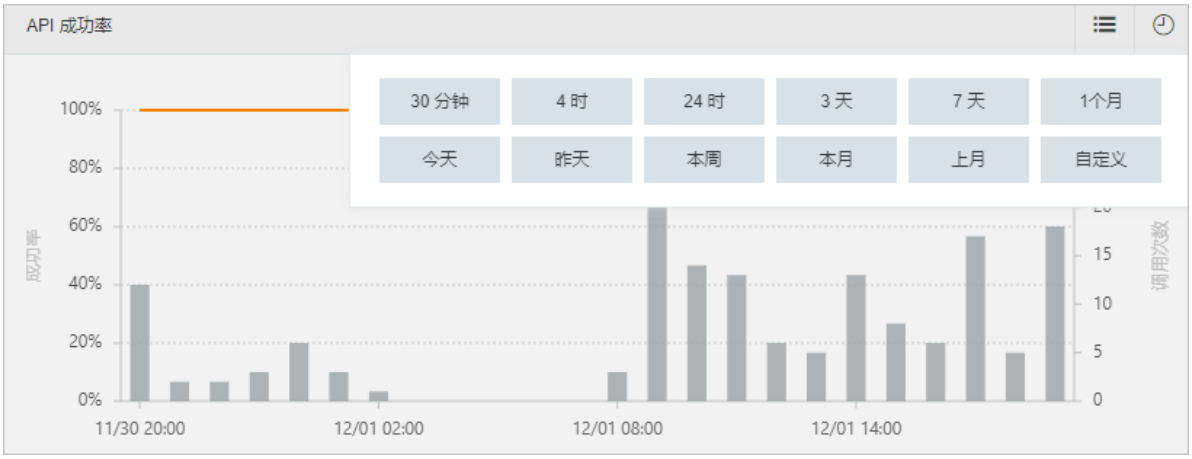



## 通用操作

在API请求模块中，您可以执行以下通用操作。

- 在左侧页签上，单击页签上的上箭头或下箭头来改变列表的排列顺序。上箭头表示升序，下箭头表示降序。
- 在右侧的详情显示区域中，单击右上角的列表图标，可在图表和表格视图间切换。


- 在右侧的详情显示区域中，单击右上角的时钟图标，可指定时间范围。

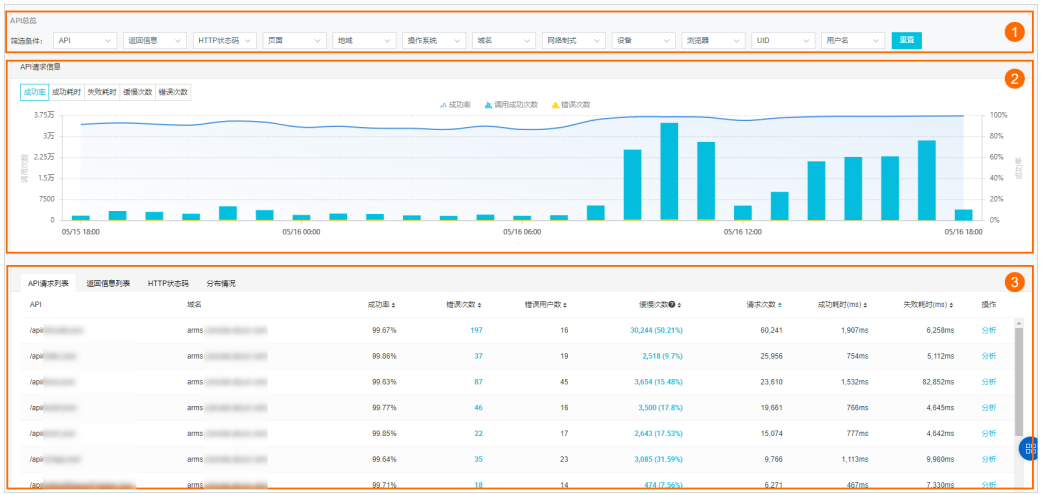



# 4.6. API详情

API详情页提供指定时间段内应用中所有API请求的成功率、平均成功耗时、平均失败耗时、缓慢次数、错误次数，并以API请求发起端的地域、域名、网络制式等维度展示统计数据。

## 功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**。
3. 在**前端列表**页面的应用列表中单击目标应用。
4. 在左侧导航栏中单击**应用 > API详情**。
   本文将页面划分为3个区域进行说明。



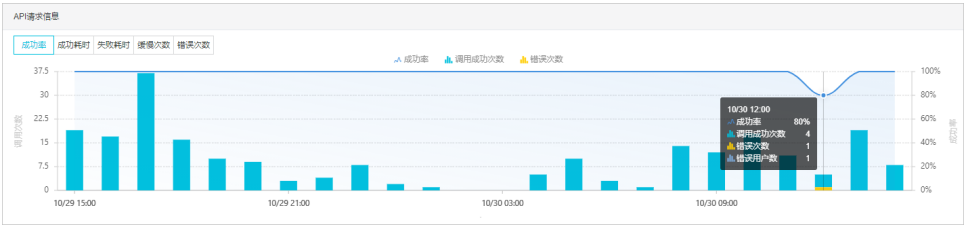
## 筛选条件

通过设置不同维度的筛选条件，查看相应的API信息。可筛选的维度包括**API**、**返回信息**、**HTTP状态码**、**页面**、**地域**、**操作系统**、**域名**、**网络制式**、**设备**、**浏览器**、**UID**和**用户名**。

将鼠标悬浮于一个筛选框中，单击右侧的×，可删除该条件的选择。单击最右侧的重置，可清除所有筛选条件。

## 成功率

在**API请求信息**区域选择**成功率**，可以查看成功率、调用成功次数、错误次数和错误用户数。

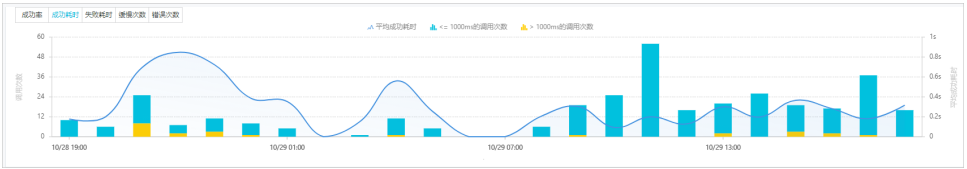


- 成功率：以折线图显示指定时间段内应用中所有API请求的成功率，对应右侧纵坐标。
- 调用成功次数：以蓝色柱状图显示指定时间段内每小时的调用次数，对应左侧纵坐标。
- 错误次数：以黄色柱状图显示指定时间段内每小时的调用错误次数，对应左侧纵坐标。

将鼠标悬浮于柱状图上，可查看指定时间段内的具体的成功率、调用成功次数、错误次数和错误用户数。

## 成功耗时或失败耗时

在**API请求信息**区域选择**成功耗时**或**失败耗时**，可以查看平均成功耗时或平均失败耗时、≤1000ms的调用次数和>1000ms的调用次数。



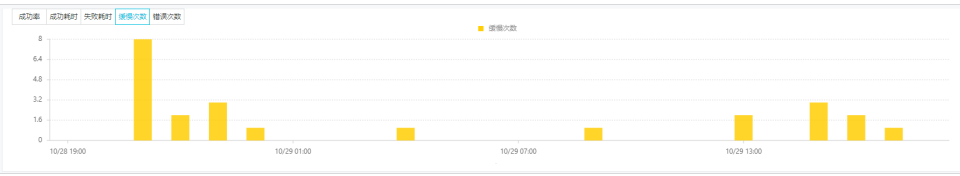
- 平均成功耗时或平均失败耗时：以折线图显示指定时间段内应用中所有API的平均成功耗时或平均失败耗时，对应右侧纵坐标。
- ≤1000ms的调用次数：以蓝色柱状图显示每小时≤1000ms的成功调用总次数或失败调用总次数，对应左侧纵坐标。
- >1000ms的调用次数：以黄色柱状图显示每小时>1000ms的成功调用总次数或失败调用总次数，对应左侧

纵坐标。

## 缓慢次数

> **说明** 缓慢次数是指耗时>1000ms的调用次数。

在**API请求信息**区域选择**缓慢次数**，可以查看缓慢次数。



缓慢次数：以柱状图显示指定时间段内应用中所有API的缓慢请求次数，对应左侧纵坐标。

## 错误次数

在**API请求信息**区域选择**错误次数**，可以查看错误次数。


错误次数：以柱状图显示指定时间段内应用中所有API的错误请求次数，对应左侧纵坐标。

## API请求列表

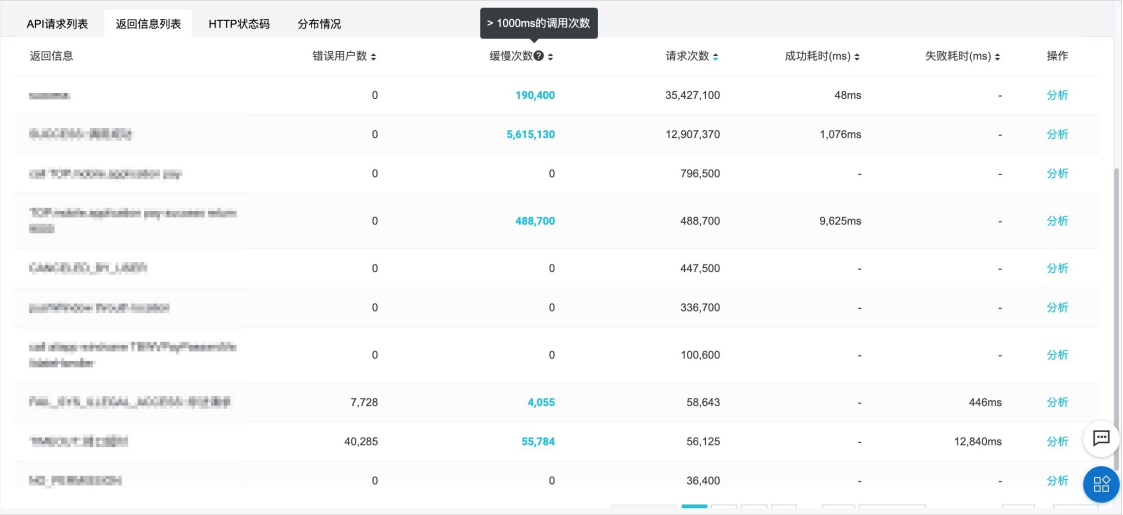
在区域③中的**API请求列表**页签，可以查看各API在指定时间段内的请求情况。


- 单击列表首行中的属性右侧的图标，可对列表进行排序。
- 将鼠标悬浮于**API**列的API别名上，单击图标，可以修改API别名，修改后，所有API将会以别名展示。单击**设置为搜索值**，可以将该API设置为筛选项。
- 单击**错误次数**列的数字，可以查看指定时间段内错误详情和错误分布情况。 在**请求详情**区域，单击**查看调用链**，可以查看错误请求的调用链路和业务轨迹。单击**查看会话**，可以查看错误请求的会话追踪详情。
- 单击**缓慢次数**列的数字，显示指定时间段内缓慢请求详情和缓慢请求分布情况。 在**请求详情**区域，单击**查看调用链**，可以查看缓慢请求的调用链路和业务轨迹。单击**查看会话**，可以查看缓慢请求的会话追踪详情。
- 单击右侧**操作**列的**分析**，可以查看API详情、API错误详情、API缓慢详情和分布情况。

- 在**API详情**页面，可以查看API请求成功率和请求详情。在**请求详情**区域，单击**查看调用链**，可以查看该API的调用链路和业务轨迹。单击**查看会话**，可以查看该API的会话追踪详情。
- 在**API错误详情**页面，可以查看错误次数分布和请求详情。
- 在**API缓慢详情**页面，可以查看响应时间分布和网络请求信息。
- 在**分布**页面，可以查看返回信息、HTTP状态码、页面、域名和地理分布信息，并可以查看操作系统、浏览器、设备和网络制式各维度的占比。

## 返回信息列表

在区域③中选择**返回信息列表**，可以查看各API的所有返回信息。




- 单击列表首行中的属性右侧的图标，可对列表进行排序。
- 将鼠标悬浮于**返回信息**列的返回信息上，单击**设置为搜索值**，可以将该返回信息设置为筛选项。
- 单击**缓慢次数**列的数字，显示指定时间段内缓慢请求详情和缓慢请求分布情况。 在**请求详情**区域，单击**查看调用链**，可以查看缓慢请求的调用链路和业务轨迹。单击**查看会话**，可以查看缓慢请求的会话追踪详情。
- 单击右侧**操作**列的**分析**，可以查看API详情、API错误详情、API缓慢详情和分布情况。
  - 在**API详情**页面，可以查看API请求成功率和请求详情。在**请求详情**区域，单击**查看调用链**，可以查看该API的调用链路和业务轨迹。单击**查看会话**，可以查看该API的会话追踪详情。
  - 在**API错误详情**页面，可以查看错误次数分布和请求详情。
  - 在**API缓慢详情**页面，可以查看响应时间分布和网络请求信息。
  - 在**分布**页面，可以查看返回信息、HTTP状态码、页面、域名和地理分布信息，并可以查看操作系统、浏览器、设备和网络制式各维度的占比。

## HTTP状态码

在区域③中选择**HTTP状态码**，可以查看所有的HTTP状态码。


- 单击列表首行中的属性右侧的图标，可对列表进行排序。
- 将鼠标悬浮于**HTTP状态码**列的HTTP状态码上，单击**设置为搜索值**，可以将该HTTP状态码设置为筛选

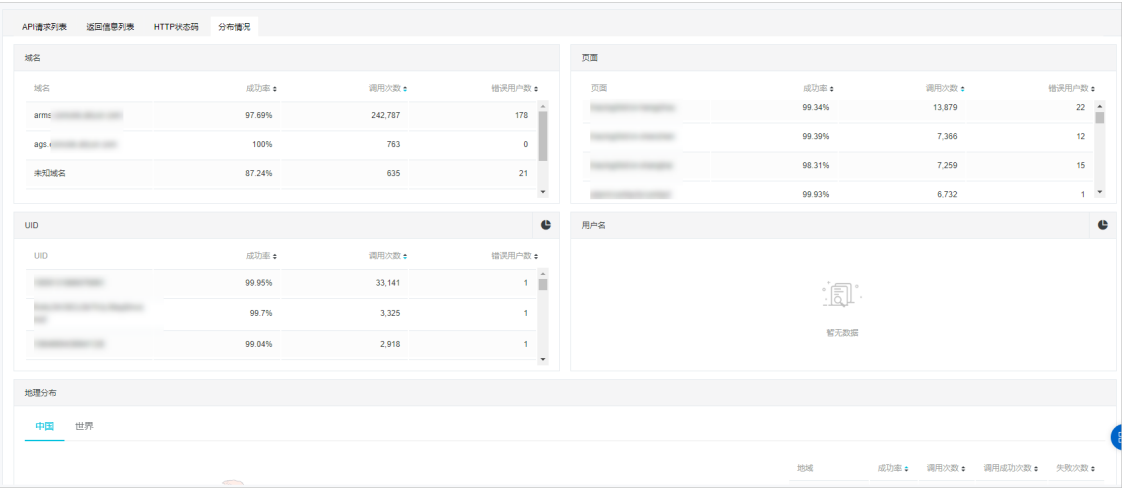
项。

- 单击**缓慢次数**列的数字，显示指定时间段内缓慢请求详情和缓慢请求分布情况。在**请求详情**区域，单击**查看调用链**，可以查看缓慢请求的调用链路和业务轨迹。单击**查看会话**，可以查看缓慢请求的会话追踪详情。
- 单击右侧**操作**列的**分析**，可以查看API详情、API错误详情、API缓慢详情和分布情况。
  - 在**API详情**页面，可以查看API请求成功率和请求详情。在**请求详情**区域，单击**查看调用链**，可以查看该API的调用链路和业务轨迹。单击**查看会话**，可以查看该API的会话追踪详情。
  - 在**API错误详情**页面，可以查看错误次数分布和请求详情。
  - 在**API缓慢详情**页面，可以查看响应时间分布和网络请求信息。
  - 在**分布**页面，可以查看返回信息、HTTP状态码、页面、域名和地理分布信息，并可以查看操作系统、浏览器、设备和网络制式各维度的占比。

### 分布情况

在区域③中选择**分布情况**，可以查看域名、页面、UID、用户名和地理分布信息，并可以查看操作系统、浏览器、设备和网络制式各维度的占比。将鼠标悬浮于**域名**或**页面**列上，单击**设置为搜索值**，可以将该域名或页面设置为筛选项。

> **说明**
>
> - 只有操作系统和浏览器支持版本号分布。
> - 操作系统、浏览器、设备和网络制式饼图只展示Top 5的维度分布，单击各区域右上角的切换视图，可查看该维度所有类别占比情况。




# 4.7. 自定义统计

本文介绍了前端监控中的自定义统计功能。

为帮助您监控和统计轻量级的业务交互行为，ARMS前端监控提供了以下两类自定义统计功能：

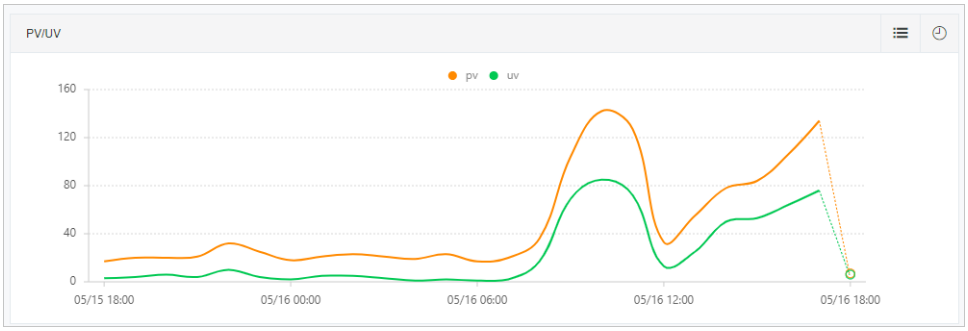
- 求和统计：用于统计业务中某些事件发生的次数总和，例如某个按钮被点击的次数、某个模块被加载的次数等。
- 均值统计：用于统计业务中某些事件发生的平均值，例如某个模块加载的平均耗时等。

在上述两类自定义统计功能中，ARMS都提供了以下几个维度的统计数据（以均值统计为例）：

- 统计详情

在统计详情折线图中可以看到指定时间段内该事件的均值数据及样本量趋势。假设统计的是某个模块的耗时数据，那么在统计详情中，可以看到对应时间区间的平均耗时数据和发送的样本数量。


- PV/UV
  在PV/UV折线图中可以看到指定时间段内该事件的PV和UV统计。




- 地理分布
  根据中国省市、世界国家的维度统计相应区域内该事件的上报情况。ARMS前端监控提供区域的上报量、均值及UV数据，帮助业务方快速了解该事件在不同区域的差别，从而辅助业务方进行决策。
- 终端分布
  浏览器、设备、操作系统、分辨率都可能会影响前端页面的性能、兼容性及展示问题，因此ARMS前端监控提供这几个维度的均值及样本量情况，让业务方了解到该事件在不同浏览器、设备、操作系统及分辨率上的分布情况。


## 求和统计API

在页面中引入前端监控SDK后，在业务JavaScript文件中使用以下日志上报API进行求和统计。

调用参数：`__bl.sum(key, value)`

调用参数说明：

| 参数 | 类型 | 描述 | 是否必需 | 默认值 |
| --- | --- | --- | --- | --- |
| key | String | 事件名 | 是 | 无 |
| value | Number | 单次累加上报量，默认1 | 否 | 1 |

示例：

```
__bl.sum('event-a');
__bl.sum('event-b', 3);
```

### 求均值统计API

在页面中引入前端监控SDK后，在业务JavaScript文件中使用以下日志上报API进行求均值统计。

调用参数：`__bl.avg(key, value)`

调用参数说明：

| 参数 | 类型 | 描述 | 是否必需 | 默认值 |
| --- | --- | --- | --- | --- |
| key | String | 事件名 | 是 | 无 |
| value | Number | 统计上报量，默认0 | 否 | 0 |

示例：

```
__bl.avg('event-a', 1);
__bl.avg('event-b', 3);
```

# 4.8. 一键诊断

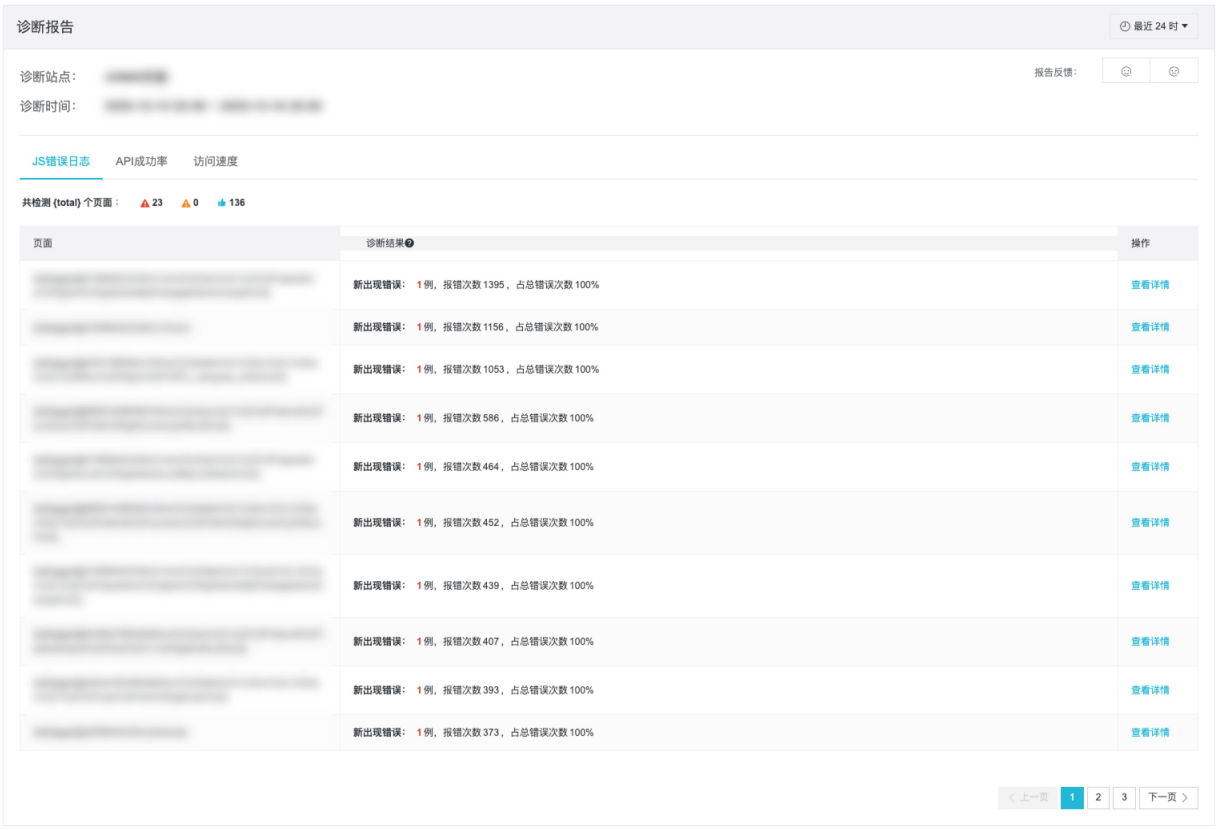
诊断报告是对过去一段时间内前端监控所采集的数据的总结，通过JS错误日志、API成功率、访问速度三个部分进行分析和报告。帮助您快速了解站点主要的监控指标，发现站点存在的问题。

### 功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击**前端监控**，在**前端监控**页面上单击应用名称。
3. 在页面右上角，单击**一键诊断**。
4. 在**诊断报告**页面，查看诊断报告。

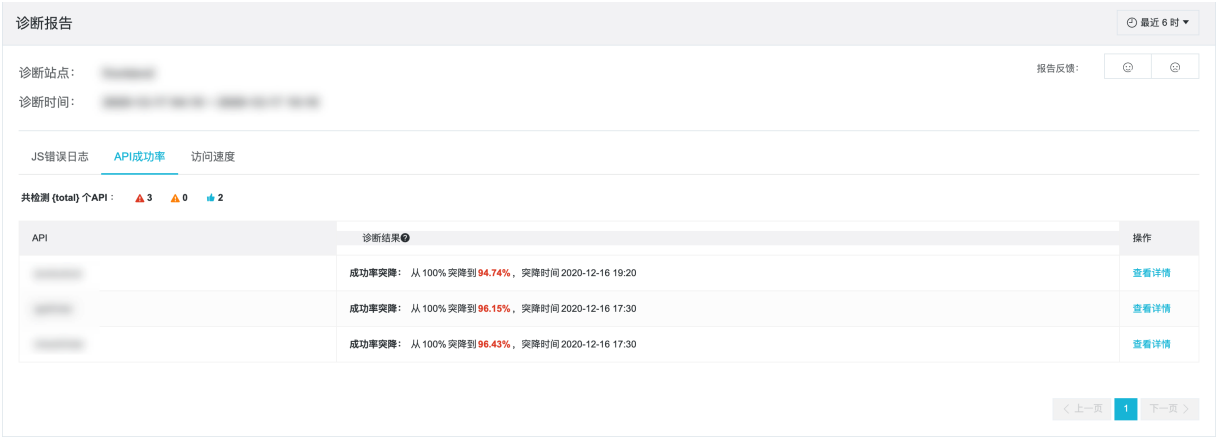
### JS错误日志

JS错误日志以JS错误数量为指标，统计错误、异常、正常三种情况的PV。以页面为聚合维度，列出诊断结果（JS错误率=错误PV/总PV），诊断结果显示了报错次数，占总错误数的比例，以及新出现错误的数量等信息。



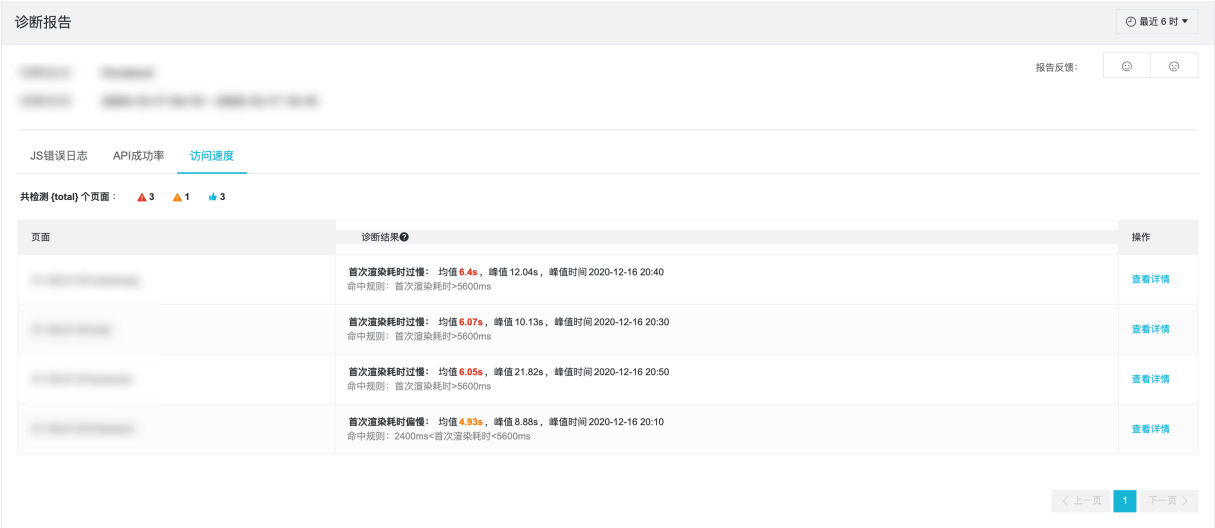

## API成功率

API成功率以API调用次数为指标，统计调用失败、异常、成功三种情况的数量。以API为聚合维度，列出诊断结果（API成功率=API调用成功次数/API总调用次数），诊断结果显示了API调用情况（例如，成功率过低、成功率突降等）、均值、谷值等信息。




## 访问速度

访问速度以页面为指标，统计了失败、异常、成功三种情况的数量。以页面为聚合维度，列出诊断结果（首次渲染耗时=responseEnd - fetchStart），诊断结果显示了页面渲染情况（例如，首次渲染耗时偏慢等）、均值、峰值、峰值时间等信息。




# 4.9. 前端监控告警规则（新版）

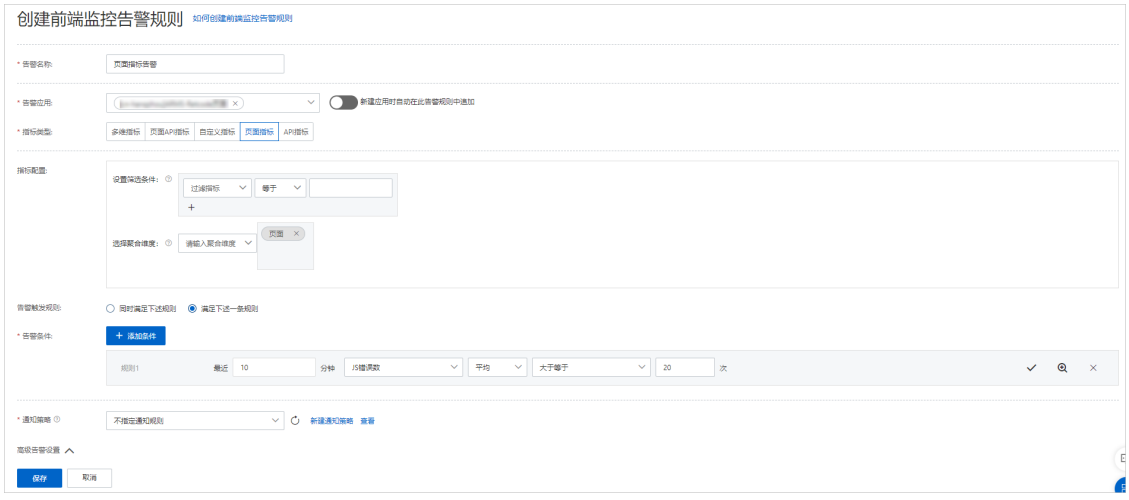
通过创建前端监控告警规则，您可以制定针对特定前端监控的告警规则。当规则被触发时，系统会以您指定的通知方式向告警联系人或钉群发送告警信息，以提醒您采取必要的问题解决措施。

## 前提条件

已成功接入前端监控，请参见前端监控接入概述。

## 操作步骤

1. 登录ARMS控制台。
2. 在左侧导航栏中选择**前端监控 > 前端监控告警规则**。
3. 在**前端监控告警规则**页面的右上角单击**创建前端监控告警规则**。
4. 在**创建前端监控告警规则**页面输入所有必填信息，完成后单击**保存**。




| 参数 | 说明 |
| --- | --- |
| 告警名称 | 告警的名称。例如：页面指标报警。 |
| 告警应用 | 选择需要设置告警的前端监控。可以选择多个前端监控。 |

| 参数 | 说明 |
| --- | --- |
| 新建应用时自动在此告警规则中追加 | 是否将之后新建的前端监控自动接入当前告警。 |
| 指标类型 | 选择监控指标的维度：<br>○ 多维指标：可以选择多个维度的指标。<br>○ 页面API指标：指定指标维度为页面和API。<br>○ 自定义指标：指定指标维度为自定义统计Key。<br>○ 页面指标：指定指标维度为页面。<br>○ API指标：指定指标维度为API。<br>**说明** 不同指标维度，告警规则的指标条件不同。 |
| 指标配置 | ○ 设置筛选条件：当前告警的筛选条件的表达式。<br>**说明**<br>■ 如果一个过滤指标的值需要*或*的关系，表达式可以使用*属于*或*不属于*，值用半角逗号（,）隔开。例如：`api`属于`api1,api2,api3`。<br>■ 如果需要同时满足多条筛选条件，单击+图标，即可编辑第二条筛选条件。<br>○ 选择聚合维度：指标类型。通过设置指标类型，可以预设聚合维度。当指标类型设置为*多维指标*时，此处需要选择聚合维度。<br>○ 选择指标：当指标类型为多维指标时，可以选择指标。此处选择的指标用于告警规则中添加的指标条件。 |
| 告警触发规则 | ○ 同时满足下述规则：需满足所有告警条件才会触发告警。<br>○ 满足下述一条规则：满足任意一条告警条件就会触发告警。 |

| 参数 | 说明 |
| --- | --- |
| 告警条件 | 单击**+添加条件**，设置告警规则表达式。例如：最近10分钟JS错误率平均大于等于20%。然后单击右侧✓图标。<br>**说明**<br>○ 选择的聚合维度不同，可以选择的条件不同。<br>○ 单击规则右侧图标，可以修改告警规则表达式。<br>○ 单击规则右侧图标，可以预览前端监控在当前规则下的指标走势图。<br>○ 单击规则右侧×图标，可以删除该告警规则表达式。<br>○ 若需设置多条告警规则，单击**+添加条件**，即可编辑第二条告警规则。 |
| 通知策略 | ○ 不指定通知规则：告警被触发时不会发送告警，仅当通知策略的匹配告警事件规则被触发时才会发送告警。<br>○ 指定通知规则发送告警：告警被触发时，ARMS通过指定通知策略的通知方式发送告警信息。您可以选择已有的通知策略，也可以新建一个通知策略。更多信息，请参见通知策略。<br>**说明** 单击查看，可以查看选中的通知策略详情。 |
| 高级告警设置 | |
| 告警数据修订策略 | 用于无数据、复合指标和环比同比等异常数据的修复。当告警指标没有达到设置的条件时，告警数据补0、补1或不补充数据。<br>○ 补零：将被判断的数值修复为0。<br>○ 补一：将被判断的数值修复为1。<br>○ 补Null：不会触发报警。<br>更多详细信息，请参见告警管理名词解释。 |

## 管理告警

创建的应用监控告警规则在**告警规则**页面上，您可以对告警规则执行启动、停止、编辑、删除、查看告警详情等操作。

1. 登录ARMS控制台。
2. 在左侧导航栏中选择**前端监控 > 前端监控告警规则**。
3. （可选）在**前端监控告警规则**页面的搜索框中输入告警名称，并单击搜索图标。

**说明** 您可以输入告警名称的一部分内容进行模糊搜索。

4. 在搜索结果列表的**操作**列中，按需对目标告警规则采取以下操作：


- 如需编辑告警规则，请单击**编辑**，在**编辑告警**页面中编辑告警规则，并单击**保存**。
- 如需删除告警规则，请单击**删除**，并在**提示**对话框中单击**确认**。
- 如需启动已停止的告警规则，请单击**启动**，并在**提示**对话框中单击**确认**。
- 如需停止已启动的告警规则，请单击**停止**，并在**提示**对话框中单击**确认**。
- 如需查看告警事件历史，请单击**告警历史**，然后在**告警事件历史**页面上查看相关记录。

> **说明** **告警事件历史**页面可以查看告警规则被触发之后生成的事件，如果告警事件不满足任何通知策略的匹配告警事件规则，则不会发送告警通知。在左侧导航栏选择**告警管理 > 告警发送历史**，可以查看满足通知策略匹配告警事件规则的告警事件被发送告警通知的记录。

## 相关文档

- 查看告警发送历史
- 查看告警事件历史

# 5.使用教程

## 5.1. 用ARMS前端监控诊断页面缓慢问题

页面性能极其影响用户体验，而用户体验很大程度地决定了用户去留，因此前端性能监控和分析尤为重要。本文主要介绍如何使用ARMS前端监控为您诊断页面缓慢的问题。

### 前提条件

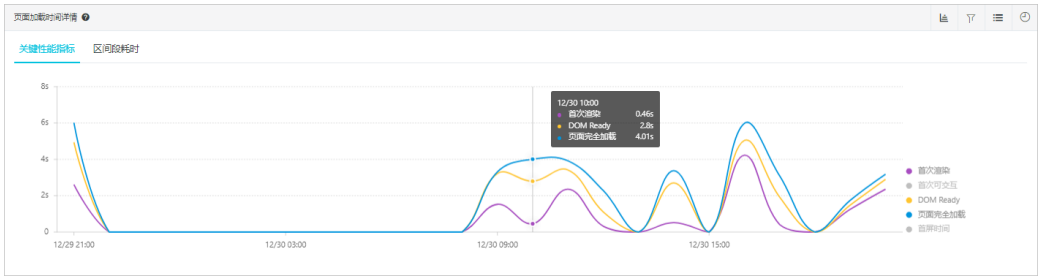
您的前端应用已接入ARMS，具体操作，请参见前端监控接入概述。

### 诊断页面缓慢的问题

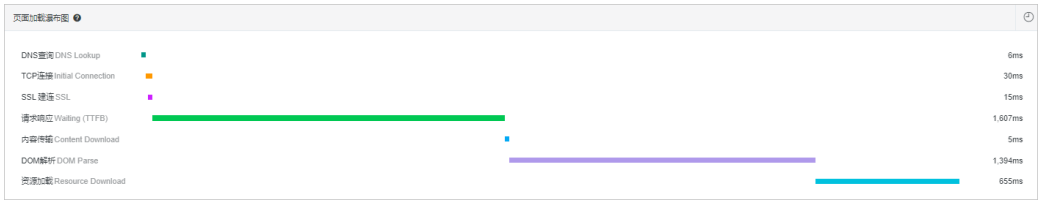
以下步骤演示了一个排查策略示例的具体步骤：

1. 登录ARMS控制台。
2. 在左侧导航栏单击**前端监控**。
3. 在**前端监控**页面上单击目标应用名称。
4. 在左侧导航栏选择**应用 > 访问速度**。
5. 在**页面加载时间详情**区域和**页面加载瀑布图**区域查看关键性能指标是否有异常：

   页面加载时间详情

   页面加载瀑布图

   - 如果**页面加载详情**区域的首次渲染时间偏高，或**页面加载瀑布图**区域的DNS查询、TCP连接和SSL建连耗时偏高，表示页面打开缓慢可能是由网络原因造成的，那么您需要检查网络问题。
   - 如果**页面加载详情**区域的DOM Ready耗时偏高，或**页面加载瀑布图**区域的请求响应和内容传输耗时偏高，表示页面打开缓慢的原因可能是API请求缓慢，那么您需要执行步骤4进行排查。
   - 如果**页面加载详情**区域的页面完全加载时间偏高，或**页面加载瀑布图**区域的DOM解析和资源加载耗时偏高，表示页面打开缓慢的原因可能是前端资源加载缓慢，那么您需要执行步骤5进行排查。

6. 在左侧导航栏选择**应用 > API详情**。

   i. 在**API请求列表**页签，单击目标API右侧**操作**列的**分析**。

ii. 在**API详情**页签，单击**请求详情**区域右上角的**查看调用链**，查看前端监控的整体耗时和后端应用的调用时序图：

- 如果后端应用处理时间较短，而整体耗时较长，则说明API请求从发送到服务端以及从服务端返回数据到浏览器端的网络传输耗时较长。在此情况下，可以在**API详情**页面，单击**请求详情**区域右上角的**查看会话**，查看本次访问的网络制式、地域、浏览器、设备和操作系统等信息。
- 如果后端应用处理时间较长，则说明后端处理的性能较差。此时可在**方法栈**栏中单击放大镜图标，然后在**方法栈**对话框中查看后端链路上哪部分内容耗时较长，继而定位问题。

7. 在左侧导航栏选择**应用 > 访问速度**。
8. 在**访问速度**页面的**慢页面会话追踪（TOP20）**区域，单击目标慢会话的页面名称。


在**会话追踪**页面的**页面资源加载瀑布图**区域，您可以查看导致页面打开缓慢的资源详情。

## 更多信息

- 页面访问速度
- 用ARMS前端监控诊断JS错误

# 5.2. 用ARMS前端监控诊断JS错误

对于前端应用来说，JS错误的发生直接影响前端应用的质量，因此对于JS错误的定位及诊断显得尤为重要。ARMS前端监控提供的JS错误诊断功能可以辅助排查JS错误，能够做到精准定位、快速诊断。

## 前提条件

前端开发人员已使用构建工具生成Source Map。

## 背景信息

实际情况中，JS错误诊断经常会遇到以下困难。

- 复现困难。
  假设您的一位用户是A，当A访问某网页时，该页面会加载在A本地的浏览器上。由于页面的加载耗时受地域、网络情况、浏览器或者运营商等因素影响，排查问题时无法复现A在访问页面时的具体情况。
- 缺少监控信息，无法深入排查。
  大部分前端监控会通过*PerformanceTiming*对象来获取完整的页面加载耗时信息，这将缺失页面静态资源的加载情况，导致无法深入定位性能瓶颈。

ARMS前端监控可利用Source Map还原代码真正的错误位置，还可以利用用户行为回溯功能还原JS错误现场。这样使得开发者能够迅速定位出错的源代码位置以及相应的代码块。

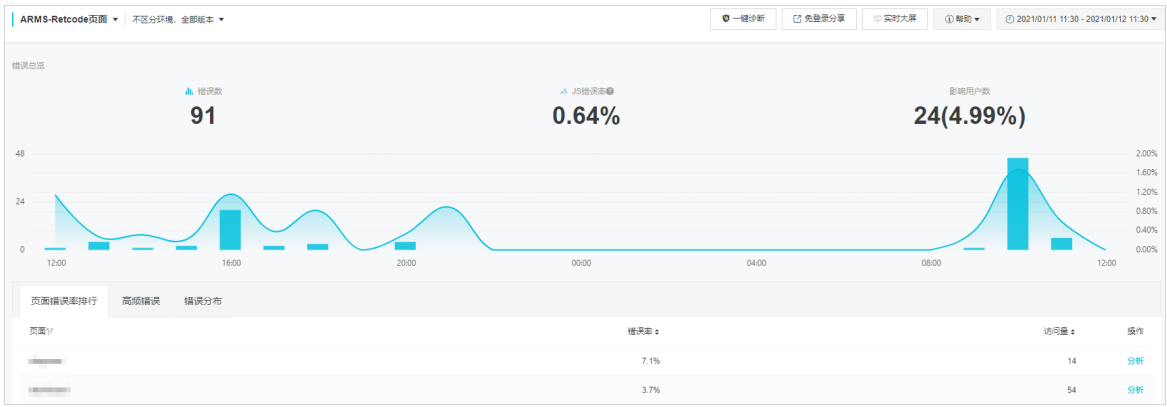
## 步骤一：安装探针

如需全方位监控前端应用，那么您首先需要为您的前端应用安装ARMS探针。请根据实际需求选择一种方式来安装探针，具体操作，请参见前端监控接入概述。

## 步骤二：查看错误总览

1. 登录ARMS控制台。

2. 在左侧导航栏，单击**前端监控**。
3. 在**前端监控**页面，单击您的应用名称。
4. 在左侧导航栏，单击**JS错误诊断**。




- 通过**错误总览**查看总的错误数、JS错误率、影响用户数及占比。
- 通过曲线图判断JS的错误趋势。
- 通过**高频错误**筛选出频繁出现的错误。
- 通过**页面错误率排行**、**错误分布**等信息对JS错误有一个全局的判断。

## 步骤三：诊断具体错误

对全局错误信息有一个初步的判断后，还需要排查具体的错误。进入ARMS前端监控中JS错误诊断的异常诊断有以下两种方式：

- 单击**高频错误**页签的**诊断**直接进入异常诊断。
- 在错误曲线图中下钻到某个时刻的**异常洞察**弹出框，再进入到异常诊断。

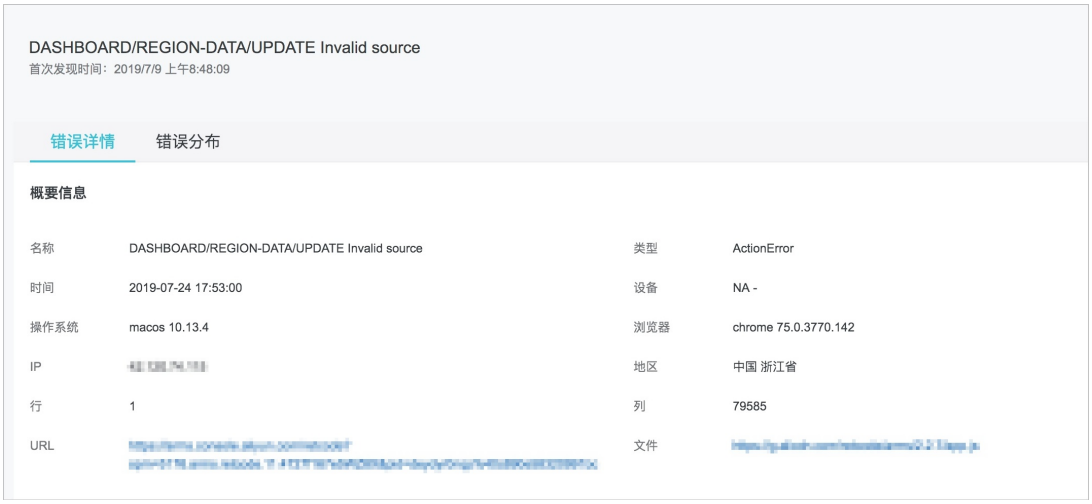
本文以第二种方式为例。

1. 观察到错误曲线图中某个时刻的错误率突然变高，将鼠标悬浮于该曲线拐点上，当鼠标显示为手形指针时单击拐点，可打开该时间点的**异常洞察**对话框。更多信息，请参见查看异常洞察。


2. 单击**高频错误Top 5**页签，选择其中一条错误，然后单击**操作**列中的**诊断**。
   进入**错误详情**页面。

## 步骤四：查看错误详情

JS错误详情信息包括首次发现时间、首次发现版本（可选上报指标）、错误名称、错误类型、错误时间、设备、操作系统、浏览器、IP、网络制式、地区、错误URL、错误所在文件及行列信息、应用版本号等。如下图所示，可以从错误详情信息中看出是实时大屏页面的地图模块在更新时报了一个数据非法的异常，错误发生在第1行第79585列。

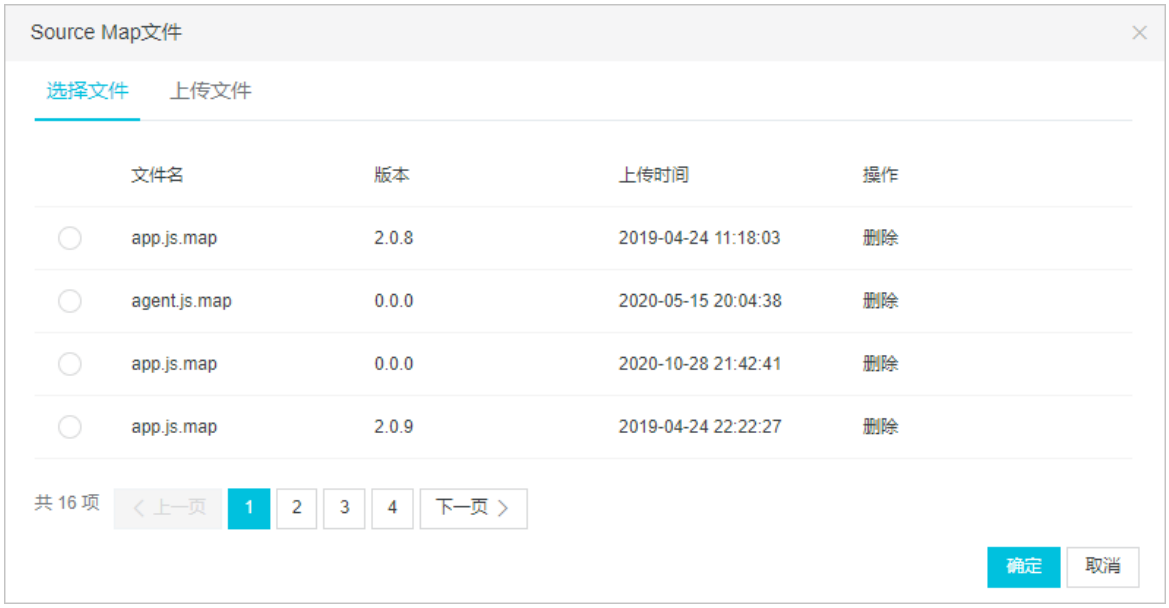



## 步骤五：定位错误代码

通过查看JS错误详情得到的信息不足以诊断问题，因为线上代码一般是经过编译、压缩、混淆等处理，因此您看到的JS代码不具备可读性。虽然知道是第1行第79585列发生了错误，但这并不代表是源码的实际位置。此时，就需要用到Source Map进行源码映射。

1. 在**堆栈信息**区域，单击一条堆栈信息左侧的>图标展开该行，并单击**选择Source Map**。
2. 在**Source Map文件**对话框中选择已有Source Map文件，或上传新的Source Map文件，并单击**确定**。

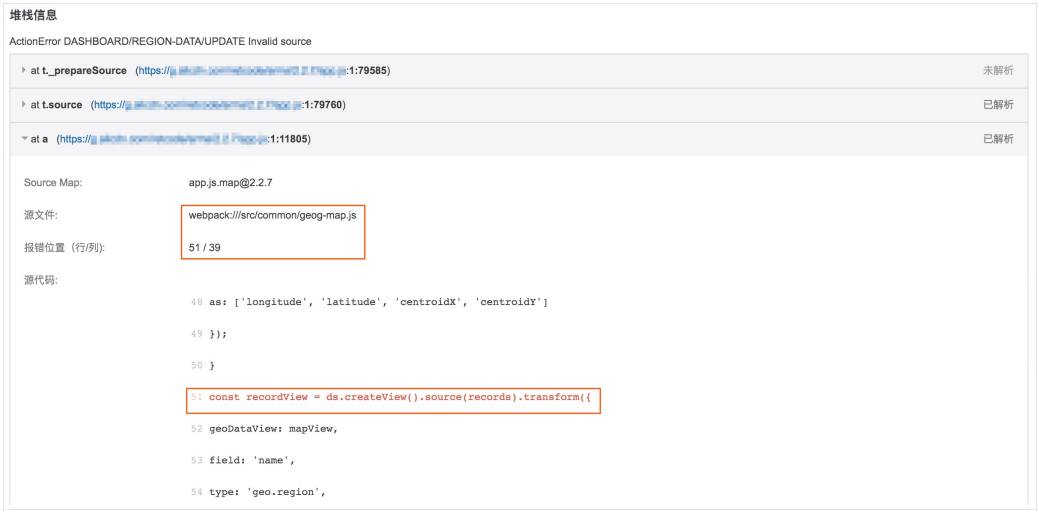
> **说明** 一次最多可以上传5个。




ARMS前端监控将利用Source Map文件还原准确的JS错误位置，如果选择的Source Map能够匹配出源码的错误，则原始错误位置将以红色字体标注于**源代码**区域中。如下图所示，经过解析后就可以一目了然地看出是哪个文件里面的哪一行出现错误。除此之外，错误堆栈的每一行都可以使用Source Map做源码

映射。

Source Map源码映射



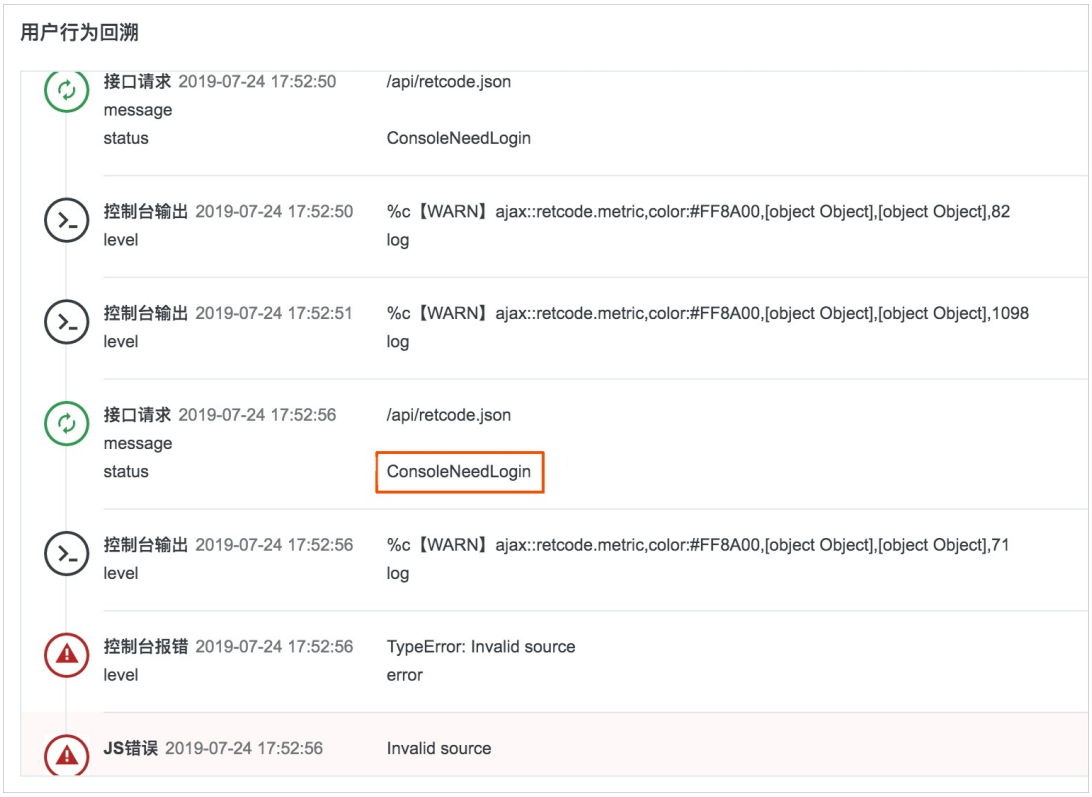

## 步骤六：查看用户行为回溯

虽然通过Source Map源码映射，可以看到源码报错是创建地图组件时，传入了非法的数据导致的。但非法的数据可能性有很多，上述代码中实际已经对数据做了判空容错处理，但依然触发了数据非法异常。如果能对报错时的数据做一个快照，就能更准确地定位问题。实际上在全局异常捕获中，现场数据一般无法获取到，只能依赖用户主动上报时附带数据信息。

最好的辅助排查方式就是复现，ARMS前端监控把页面上发生的各个事件节点定义为用户行为，包括页面加载、路由跳转、页面单击、API请求、控制台输出等信息，按照时间顺序将用户行为串联起来就是用户的行为链路。通过出错时的行为回溯可以帮助开发者复现问题。

如用户行为回溯所示，通过出错时用户行为回溯可以看到出错前有一个API请求，这个API请求试图去请求数据实时更新地图模块，但是返回的数据是ConsoleNeedLogin，说明此时页面已经退出登录状态了，这就是导致非法数据的根源。

用户行为回溯




## 相关文档

- JS错误诊断
- “Script error.”的产生原因和解决办法

# 5.3. 诊断网页加载过慢的问题

定位、排查网页加载过慢问题的原因有诸多难点。针对这类问题，ARMS前端监控的慢会话追踪功能提供页面静态资源加载的性能瀑布图，可深入定位页面资源加载情况，全方位地诊断故障根源，从而快速排除故障。

## 问题描述

网页加载较慢是经常出现且前端非常关注的问题之一。定位、排查解决这类问题的难点如下：

- 复现困难
  假设您的一位用户是A，当A访问某网页时，该页面会加载在A本地的浏览器上。由于页面的加载耗时受地域、网络情况、浏览器或者运营商等因素影响，排查问题时无法复现A在访问页面时的具体情况。
- 监控信息缺少，无法深入排查
  大部分前端监控会通过PerformanceTiming对象来获取完整的页面加载耗时信息，这将缺失页面静态资源的加载情况，导致无法深入定位性能瓶颈。

## 解决方案

只需将Web端应用接入ARMS前端监控，并且在接入时开启页面资源上报功能，然后利用ARMS前端监控的慢会话追踪功能，即可帮助您快速定位性能瓶颈。

## 步骤一：接入ARMS前端监控

ARMS前端监控SDK默认不上报页面加载的静态资源信息。若使用慢会话追踪功能对慢页面加载问题快速定位，则需获取页面加载的静态资源信息。

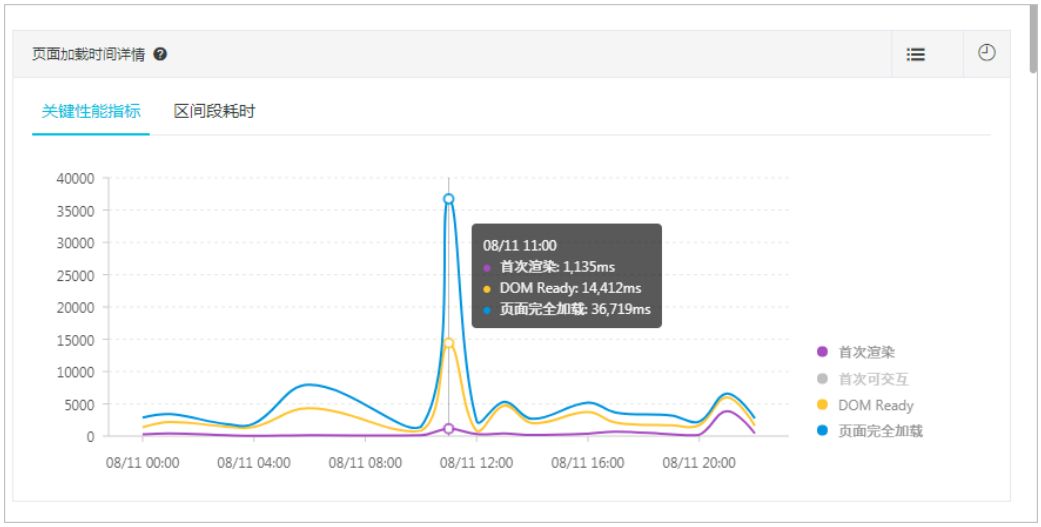
将您的Web端应用接入ARMS前端监控，请参见以CDN方式接入前端监控。

> **注意** 在接入时选中**开启页面资源上报**项，才能使用慢会话追踪功能。

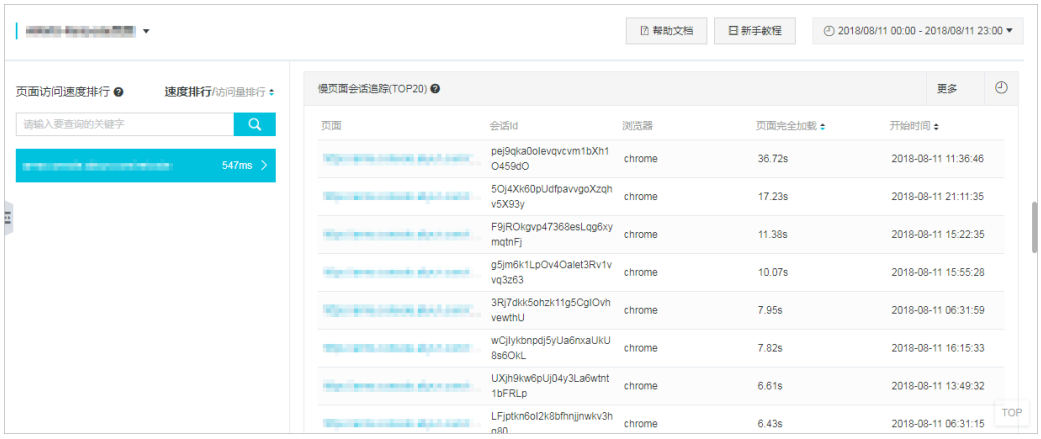
## 步骤二：定位故障

您可以通过两种不同入口的方式来定位故障，两种方式均能达到使用慢会话追踪功能诊断网页性能问题的目的。

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**前端监控**。
3. 在**前端监控**页面上，单击您的应用名称。
4. 在左侧导航栏中单击**访问速度**。
   **访问速度**页面的详情介绍，请参见页面访问速度。在本示例中，该页面性能较差，在11:00时的页面完全加载时间达到36.7 s。



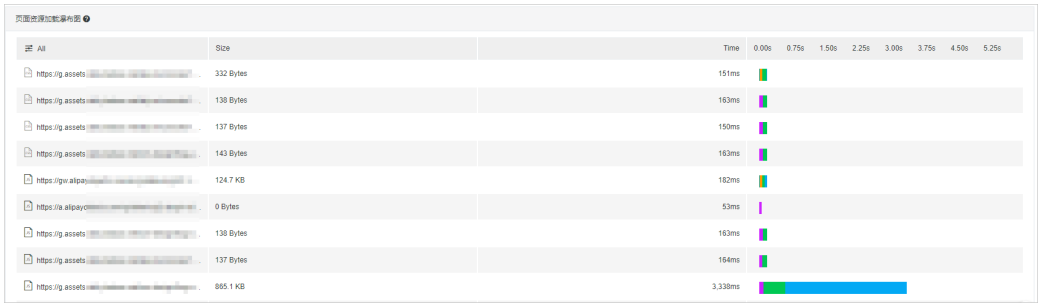

5. 在**访问速度**页面上，拖动右侧的滚动条至**慢页面会话追踪（TOP20）**区域。
   该区域会列出该页面在指定时间段内加载最慢的20个会话。
   在本示例中，您可以看到在11:36:46的页面加载时间为36.72 s，可以判断这次访问是导致页面加载时间骤增的原因。




6. 在**慢页面会话追踪（TOP20）**区域的**页面**列中单击目标页面名称。
   进入**慢加载详情**页面。

7. 根据**慢加载详情**页面的信息定位故障原因，进而排除故障。
   **慢加载详情**页面顶部的**页面信息**区域展示了本次访问的客户端IP地址、浏览器、操作系统等信息，帮助您确认故障原因。


   **慢加载详情**页面的**页面资源加载瀑布图**区域展示了页面静态资源加载的瀑布图，帮助您快速定位资源加载的性能瓶颈。




   **慢加载详情**页面的详细信息，请参见慢会话追踪。

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**前端监控**。
3. 在**前端监控**页面上，单击您的应用名称。
4. 在左侧导航栏中单击**会话追踪**。
   **会话列表**区域展示了该应用下的所有会话，您可以按照用户名、用户ID、会话ID等条件进行过滤，快速发现耗时较长的会话信息。
5. 在**会话列表**区域，单击目标会话名称。然后根据**会话追踪详情**页面的信息定位故障原因，进而排除故障。
   **会话追踪**页面的详细信息，请参见慢会话追踪。

操作至此，已使用慢会话追踪功能完成问题排查，该功能可以帮助你复现用户在访问页面时的页面资源加载情况，快速定位性能瓶颈问题。

## 方式二：从会话追踪开始排查

## 相关操作

为避免在出现问题后被动诊断错误原因，您还可以使用ARMS的报警功能针对一个接口或全部接口创建报警，即可在出现问题的第一时间向运维团队发送通知。

创建报警操作步骤，请参见快速创建ARMS报警（旧版）。

## 更多信息

- 页面访问速度
- 慢会话追踪
- 快速创建ARMS报警（旧版）

# 5.4. 使用用户行为回溯诊断JS错误

在JS错误诊断过程中，ARMS前端监控提供用户行为回溯功能，全面还原错误发生时的用户行为，能够辅助您快速定位解决问题。

## 背景信息

ARMS前端监控将页面上发生的各个事件节点定义为用户行为，包括控制台行为、页面跳转、用户点击、用户输入、接口请求等行为。按照时间顺序将用户行为串联起来就构成用户的行为链路，通过错误出现时的行为回溯分析用户的行为链路，可以辅助您复现错误场景。

## 步骤一：安装探针

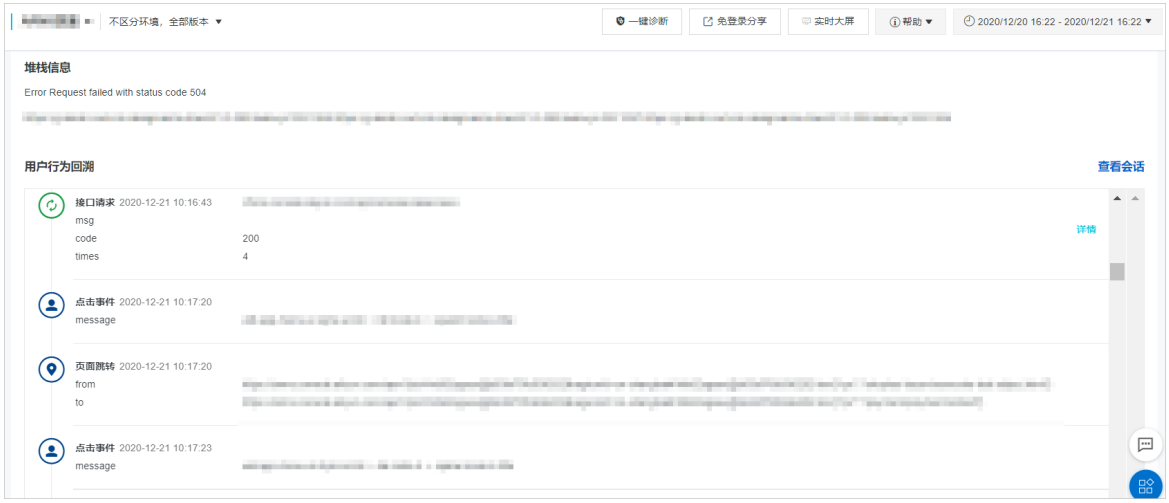
为您的前端应用安装ARMS探针后，ARMS将对前端应用进行全方位监控。请根据实际需求选择一种方式来安装探针，具体操作，请参见前端监控接入概述。

## 步骤二：诊断错误

通过用户行为回溯功能复现JS错误场景，诊断错误出现的具体原因。

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**前端监控**。
3. 在**前端监控**页面上，单击目标应用名称。
4. 在左侧导航栏单击**JS错误诊断**。
5. 在**JS错误诊断**页面，单击页面中部的**高频错误**页签。


6. 单击错误右侧**操作**列的**诊断**。
7. 在**错误详情**页面，分析**用户行为回溯**区域的用户行为，从而判断出是哪部分的操作引发的错误。




## 相关文档

- “Script error.”的产生原因和解决办法

# 5.5. 慢会话追踪

慢会话追踪功能可提供页面加载过程中静态资源加载的性能瀑布图，帮助您根据页面性能数据详细了解页面资源加载情况，并快速定位性能瓶颈。

## 前提条件

> **注意** 静态资源加载信息的上报是在页面加载时触发的，上报信息量较大。加载耗时大于8秒时会全量上报，介于2~8秒时会采样5%，小于2秒时不上报。如果应用对页面性能要求很高，不建议进行该配置。

- 接入前端监控，具体操作，请参见前端监控接入概述。
- 修改SDK配置。
  阿里云ARMS前端监控SDK默认不上报页面加载的静态资源信息。如需获取页面加载的静态资源信息并使用慢会话追踪功能，请在SDK的 `config` 部分将sendResource配置为*true*。SDK的具体配置如下所示。

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"atc889zkcf@8cc3f6354******",imgUrl:
"https://arms-retcode.aliyuncs.com/r.png?",sendResource:true};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("cros
sorigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

- 重新部署应用。
  重新部署应用后，页面的onload事件触发时就会上报当前页面加载的静态资源信息，继而可在阿里云ARMS前端监控中对慢页面加载问题进行快速定位。

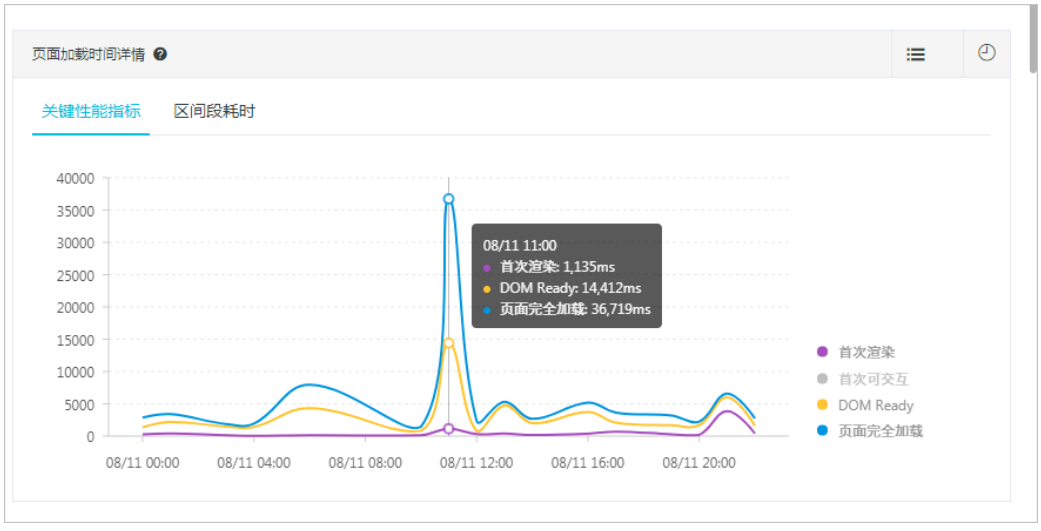
## 功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**前端监控**。
3. 在**前端监控**页面上单击目标应用名称。
4. 在左侧导航栏中选择**应用 > 会话追踪**。

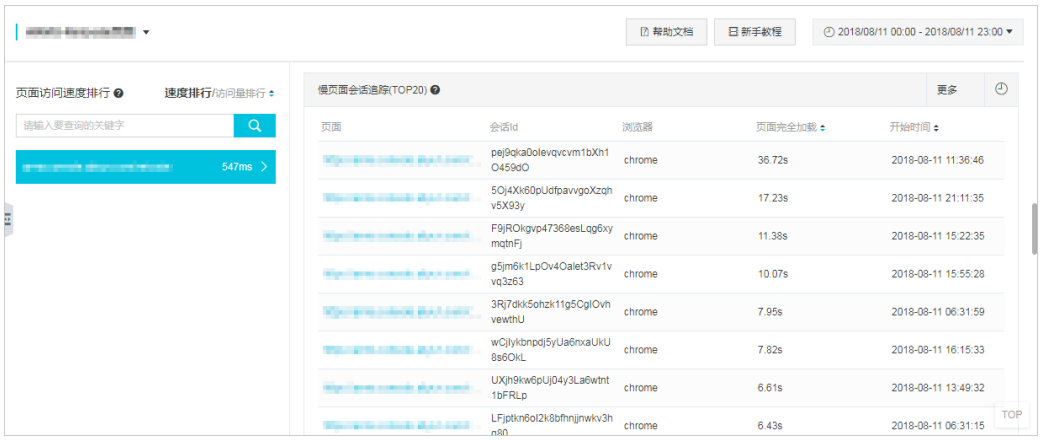
## 使用案例：定位页面性能瓶颈

接下来以一个示例介绍如何定位页面性能瓶颈。

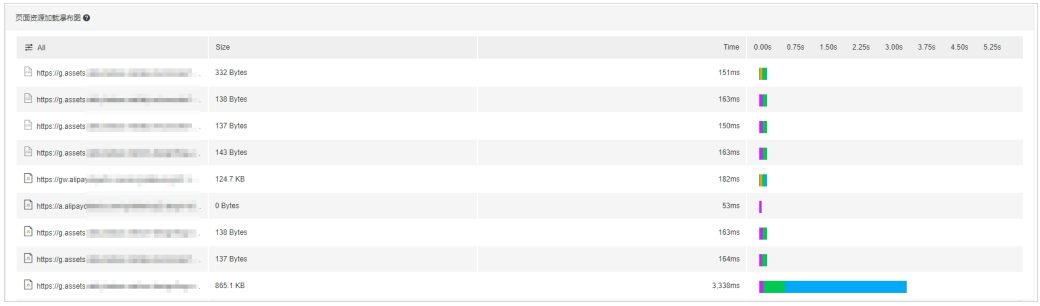
1. 在左侧导航栏中选择**应用 > 访问速度**，结果如下图所示。可见11:00时的页面完全加载时间长达36.7秒。



2. 在**访问速度**页面上，拖动右侧的滚动条至**慢页面会话追踪（TOP20）**区域。该区域会列出该页面在指定时间段内加载最慢的20个会话。
   更改指定时间段，可单击页面右上角显示的时间区间，在弹出的列表中进行选择。
   下图中11:36:46有一次会话的页面加载时间为36.72秒，可以判断这次访问应该是导致页面加载时间骤增的原因。




3. 在**慢页面会话追踪（TOP20）**区域的**页面**列中单击页面名称。
   进入**慢加载详情**页面，可以直观地查看页面静态资源加载的瀑布图，并借此快速定位资源加载的性能瓶颈。

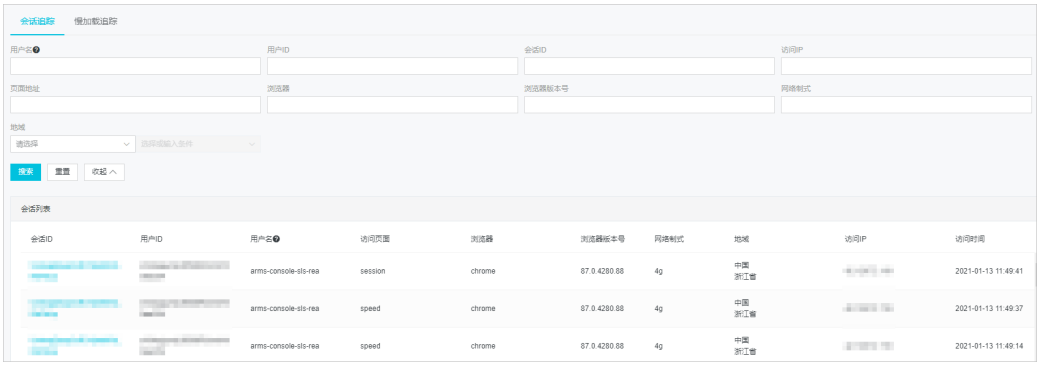



4. 在**慢加载详情**页面顶部的**页面信息**区域框，可以查看本次访问的客户端IP地址、浏览器、操作系统等信息，从而进一步确认问题是由网络原因还是其他原因导致的，并进行针对性的优化。


## 发现性能问题的其他渠道

除了**访问速度**页面外，您也可以通过**会话追踪**页面发现性能问题。

1. 在左侧导航栏中单击**会话追踪**，即可在**会话追踪**页面查看该应用下的会话列表。您也可以按照用户名、用户ID、会话ID、访问IP、页面地址、浏览器、浏览器版本号、网络制式、地域等条件筛选会话。



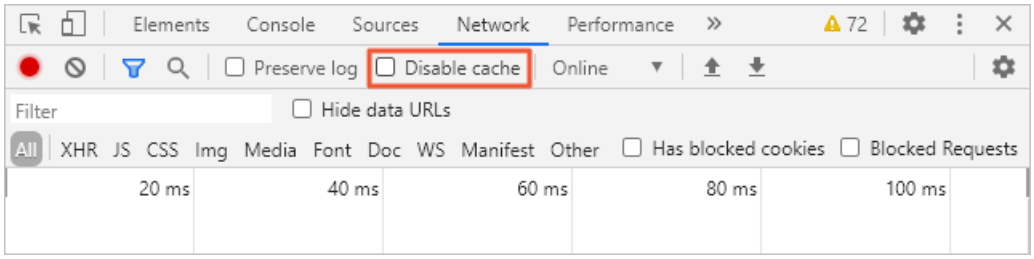
2. 单击**会话ID**列中的ID，即可打开**会话详情**页面，并查看该会话的概要信息和会话轨迹。更多信息，请参见会话追踪。

## 常见问题

1. **为什么资源加载瀑布图中 `Size` 为0？**

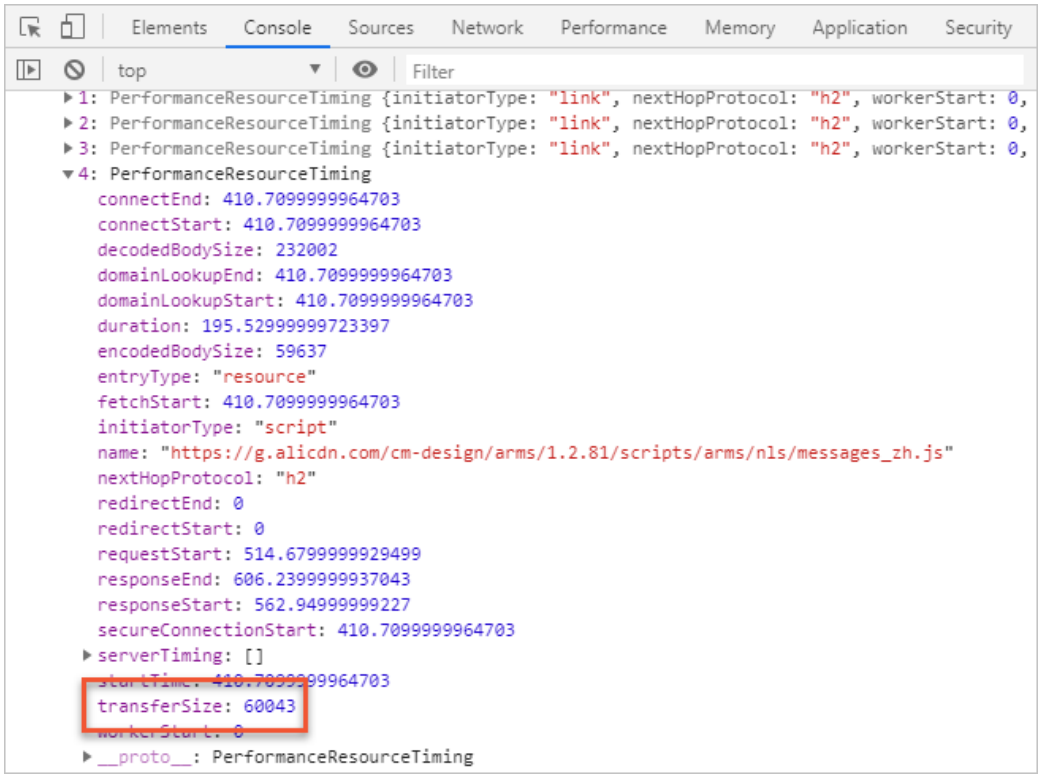
   `Size` 数据是通过PerformanceResourceTiming.transferSize获取的。transferSize只读属性表示所提取资源的大小（以八位字节表示）。如果是从本地缓存获取资源，或者如果是跨源资源，则该属性返回的值为0。

   在Chrome浏览器中按**F12**打开开发者工具面板，当**Network**页签上的**Disable cache**未选中时，transferSize为0。




   **解决方法**

   选中**Disable cache**后，`transferSize` 即恢复正常。




2. **为什么资源加载瀑布图中 `Time` 为0？**

   `Time` 数据是通过PerformanceResourceTiming.duration获取的。在瀑布图中查看静态资源加载情况时，部分情况下 `Time` 为0，是由于该请求命中了缓存，并且是通过 `max-age` 控制的长缓存。

   **解决方法**

   在Chrome浏览器中按**F12**打开开发者工具面板，取消选中**Network**页签上的**Disable cache**，刷新页面后即可看到经过网络过程所耗的时间。

3. **为什么很多返回的时间数据为0？**

   查看API返回的数据时，如果发现很多返回的时间数据为0，是因为受同源策略的影响，跨域资源获取的时间点会为0，主要包括以下属性：

   - redirectStart
   - redirectEnd
   - domainLookupStart
   - domainLookupEnd
   - connectStart
   - connectEnd
   - secureConnectionStart
   - requestStart
   - responseStart

   **解决方法**

   在资源响应头中添加 `Timing-Allow-Origin` 配置，例如： `Timing-Allow-Origin: *` 。

4. **API加载瀑布图反映哪个时间段内的API加载情况？**

   API加载瀑布图对应的时间段为：

   - 开始时间：页面开始加载时间

- 结束时间：页面完全加载时间+1分钟

API加载瀑布图的作用是更直观地展现页面加载过程中所请求API的整体情况。

5. **为什么API加载瀑布图中的耗时与页面资源加载瀑布图中的耗时不一致？**
   API加载瀑布图中的耗时会比页面资源加载瀑布图中的API耗时多几毫秒，原因在于二者的获取方式不同。具体而言，API加载瀑布图中的耗时是通过计算从API发送请求到API数据返回所花费的时间获取的，而页面资源加载瀑布图中的API耗时是通过浏览器提供的API `performance.getEntriesByType('resource')` 获取的。
   耗时统计数据的几毫秒差异不会影响排查性能瓶颈。
6. **API加载瀑布图中时间轴的起点时间是什么？**
   API加载瀑布图中的时间轴的起点时间是API发起请求的时间与页面fetchStart时间的差值。该时间轴展示页面加载过程中API请求发起的时间点和耗时。

## 更多信息

- 页面访问速度
- 前端监控常见问题

# 5.6. 使用前后端链路追踪诊断API错误原因

在前端监控中，即便已知API的请求耗时，也无从知晓准确的网络传输性能、后端服务的调用链路及性能，因而无法快速准确地排查应用API问题。前后端链路追踪功能可以解决此类问题，它会将API请求从前端发出到后端调用的链路串联起来，真实还原代码执行的完整现场。

## 前提条件

您已经开通ARMS前端监控和ARMS应用监控，请参见开通和升级ARMS。阿里云ARMS应用监控需要2.4.5或更高版本，关于其配置，请参见应用监控概述。

## 背景信息

应用监控可提供API在后端的处理性能及调用链路，但这些数据未必能准确反映用户的真实体验。前端监控只能监控到API从发送到返回的整体耗时及状态，无法提供后端服务的调用链路及性能数据。在这种情况下，前后端链路追踪功能可将前端与后端串联起来，给您一站式的问题排查体验。

## 配置ARMS前端监控

1. 确认前端站点和应用之间是否存在对应关系。
2. 不关闭API自动上报，允许API自动上报。
3. 配置enableLinkTrace为 `true` ，允许前后端链路追踪。具体配置为：

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxx",imgUrl:"https://arms-retcode
.aliyuncs.com/r.png?", enableLinkTrace: true};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("cr
ossorigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

1. 确认前端站点和应用之间是否存在对应关系。
2. 配置enableLinkTrace和enableApiCors为 `true` 。

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxx",imgUrl:"https://arms-retcode
.aliyuncs.com/r.png?",
enableLinkTrace: true, enableApiCors: true};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("cr
ossorigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

**注意** 配置enableApiCors为 `true` ，后端服务也需要支持跨域请求及自定义header值，请确认所有请求都配合联调正常，否则会出现请求失败的问题。Nginx配置参考如下：

```
upstream test {
        server 192.168.220.123:9099;
        server 192.168.220.123:58080;
    }
    server {
        listen    5800;
        server_name  192.168.220.123;
        root         /usr/share/nginx/html;
        include /etc/nginx/default.d/*.conf;
        location / {
            proxy_pass http://test;
            proxy_set_header Host $host:$server_port;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Real-PORT $remote_port;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   EagleEye-TraceID $eagleeye_traceid;
            proxy_set_header   EagleEye-SessionID $eagleEye_sessionid;
            proxy_set_header   EagleEye-pAppName $eagleeye_pappname;
        }
```

3. 配置ignore，请参见ignore。 具体配置为：

```
let whitelist = ['api.xxx','source3']; // 白名单。
let blacklist = ['source2','source6']; // 黑名单。
// 可根据需求，选择黑白名单（在方法中return true或者false去控制）。
ignore: {
            ignoreApis: [
                function(str) {   // 方法。
                    if (whitelist.includes(str)) {
                        return false;
                    }
                    return true; // 返回true则忽略。
                }]
        }
```

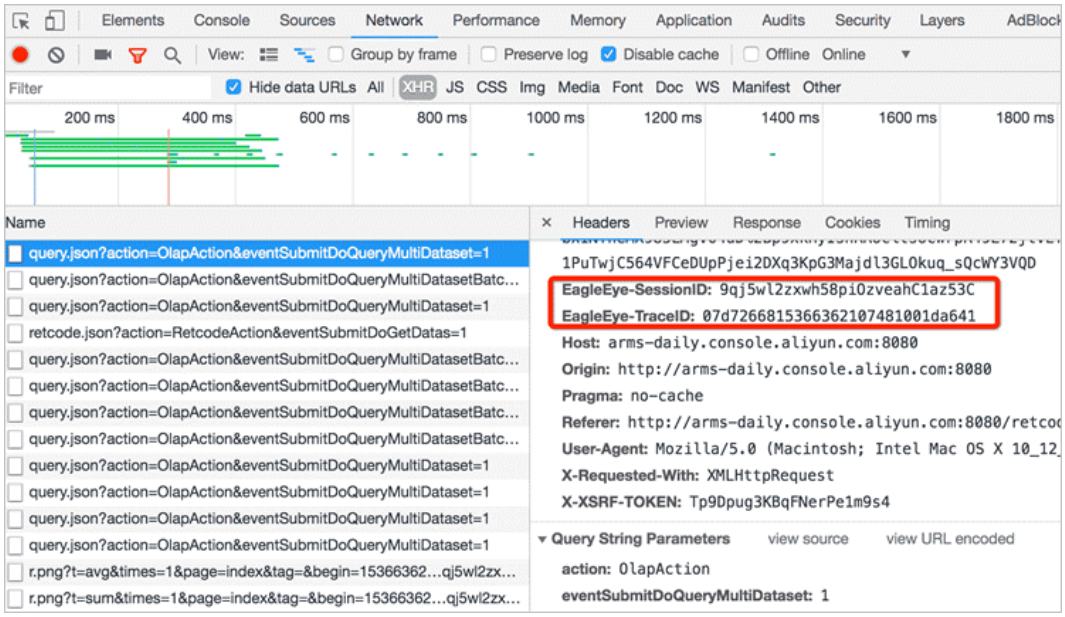
**说明** ignore类似于黑白名单，可以避免部分第三方资源请求的header值被修改，从而防止资源请求出现错误。

# API与当前应用域名非同源

## 工作原理

- 在允许API自动上报的前提下，如果API与当前应用的域名同源，则会在API请求头（Request Header）加入两个自定义Header：EagleEye-TraceID和EagleEye-SessionID。EagleEye-TraceID即串联前后端链路的标识。
- 如果API与当前应用的域名不同源，则不会在请求头添加自定义Header，以保证应用请求可以正常发送。
- 如需验证前后端链路追踪配置是否生效，可以打开控制台查看对应API请求的Request Headers中是否有EagleEye-TraceID和EagleEye-SessionID这两个标识。

> 警告 EagleEye-TraceID和EagleEye-SessionID的值有相应的含义，请勿自行生成。

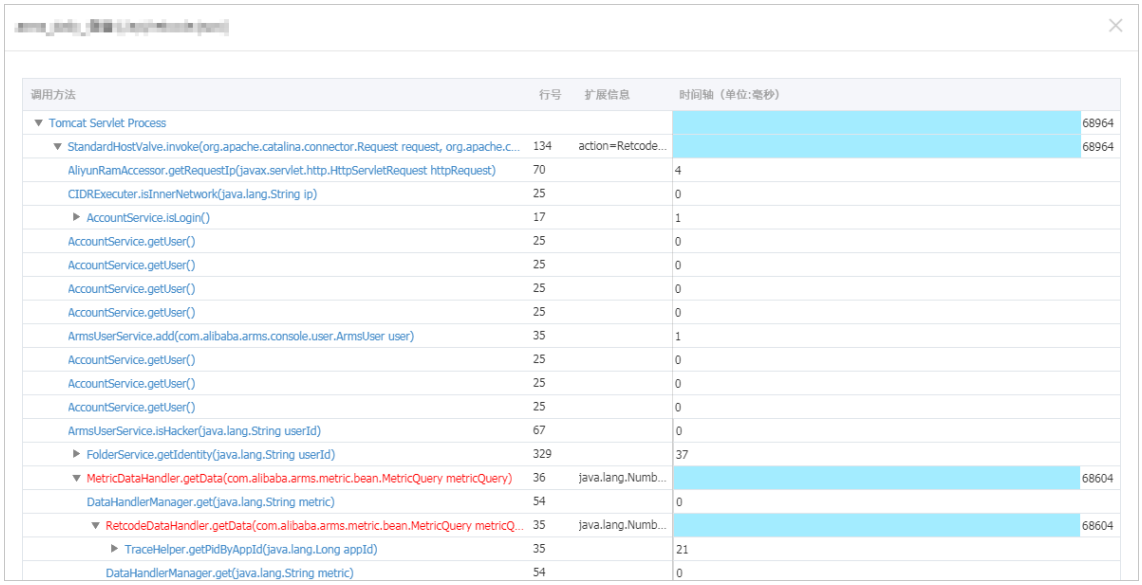



## 使用场景和案例

通过调用的时间轴，可以知道是网络传输还是后端调用导致请求耗时时间过长，同时单击后端应用中的线程剖析，可以看到本次请求后端的完整调用链路。从而根据业务定位是什么原因导致API错误：

- 当API返回错误码或者业务逻辑错误时，定位问题操作如下：
  i. 登录ARMS控制台。
  ii. 在左侧导航栏单击**前端监控**，然后单击目标应用名称。
  iii. 在左侧导航栏单击**API请求**。
  iv. 在右侧的**API链路追踪（TOP 20）**区域的**API失败列表**中，找到相关的API或者TraceID，并单击右侧的**链路追踪**，以查看前端监控的整体耗时和后端应用的调用时序图。


  v. 根据调用的时间轴判断，耗时长的是网络传输还是后端调用。
  vi. 对后端应用单击**方法栈**列中的放大镜图标，可以查看本次请求的完整后端调用链，并根据业务定位导致API错误的原因。




- 当API耗时较长时，定位问题操作如下：
  i. 登录ARMS控制台。
  ii. 在左侧导航栏单击**前端监控**，然后单击目标应用名称。
  iii. 在左侧导航栏单击**API请求**。
  iv. 在右侧的**API链路追踪（TOP20）**区域中，按照请求耗时对API进行降序排列，找到耗时较长的API或者TraceID。
  v. 单击右侧的**链路追踪**链接，以查看前端监控的整体耗时和后端应用的调用时序图。
    - 如果后端应用处理时间较短，而整体耗时较长，则说明API请求从发送到服务端以及从服务端返回数据到浏览器端的网络传输耗时较长。此时可以单击**访问明细**，并在查看详情页面上查看本次访问的网络、地域、浏览器、设备、操作系统等信息。
    - 如果后端应用处理时间较长，则说明后端处理的性能较差。此时可以单击**方法栈**栏中的放大镜图标，并在本地方法栈对话框中查看后端链路上哪部分内容耗时较长，继而定位问题。

# 5.7. 与链路追踪Tracing Analysis前后端打通

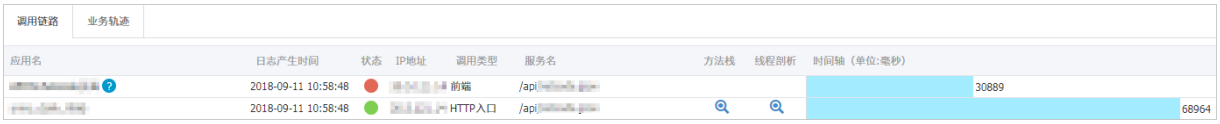
在前端监控中，即便已知API的请求耗时，也无从知晓准确的网络传输性能、后端服务的调用链路及性能，因而无法快速准确地排查应用API问题。前后端链路追踪功能可以解决此类问题，它会将API请求从前端发出到后端调用的链路串联起来，真实还原代码执行的完整现场。

## 前提条件

您需要先通过npm方式安装探针，具体操作，请参见以npm方式接入前端监控。

## 背景信息

默认前端监控和应用监控都是自动打通的，通过前端监控和会话追踪都能查看端到端的请求追踪数据。当您使用OpenTracing协议（开源Trace ID，即uber-trace-id）的情况下，需要您手动将前端监控的JavaScript配置与链路追踪Tracing Analysis前后端打通。

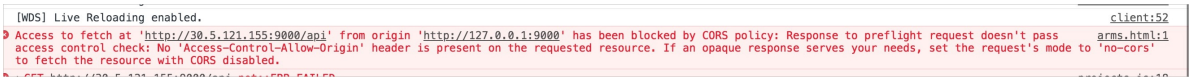


## 配置参数

当您通过npm方式安装探针时，需要完成以下参数配置。

```
const BrowserLogger = require('alife-logger');
const __bl = BrowserLogger.singleton({
    pid:"xxx",      // 站点ID
    appType:"web",
    enableLinkTrace:true,
    linkType: 'tracing', // 链路追踪类型：tracing与链路追踪的Tracing产品做前后端链路打通
    enableApiCors: true  // 是否允许请求跨域及自定义header，默认值为false，设置为true后则需要后端服务支持
});
```

- enableApiCors:true： 需要后端服务支持请求跨域及自定义header值，否则请求会失败，如下图所示。




若请求失败，您可以参考以下配置处理，以在Node.js应用中配置为例：

```
//用于设置允许跨域访问该服务.
app.all('*', function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Content-Type,uber-trace-id'); // linkType: 'tracing' 时，header值需要允许 uber-trace-id；linkType: 'arms'时，header值需要支持：EagleEye-TraceID、EagleEye-SessionID、EagleEye-pAppName
  res.header('Access-Control-Allow-Methods', '*');
  next();
});
```

- enableLinkTrace:true： 开启后
  - 如果API请求与页面域名同源时，则默认会在header中增加traceId的透传，实现前后端链路追踪。
  - 如果API请求与页面域名非同源时，要实现前后端链路追踪，则需要同时配置enableApiCors:true，并且需要后端服务支持请求跨域及自定义header值，

> **注意** 请确认所有请求都配合联调正常，否则会出现请求失败的问题。若请求失败，请参见上述在Node.js应用中配置的举例来处理。

- linkType: 'tracing'： 指枚举值。
  - 默认为arms，可与arms后端监控做前后端链路追踪，header需要允许EagleEye-TraceID、EagleEye-SessionID、EagleEye-pAppName加入。
  - tracing：可与链路追踪产品做前后端链路追踪，header需要允许uber-trace-id。

# 6.前端监控特殊使用场景

## 6.1. SPA页面上报

在SPA（Single Page Application）单页面应用中，页面只会刷新一次。传统的方式只会在页面加载完成后上报一次PV，而无法统计到各个子页面的PV，也无法让其他类型的日志按子页面聚合。本文介绍如何使用ARMS前端监控SDK解决SPA页面上报问题。
ARMS前端监控SDK提供了针对SPA页面的两种处理方式：

- 开启SPA自动解析
- 完全手动上报

### 开启SPA自动解析

此方法适用于大部分以URL Hash作为路由的单页面应用场景。

在初始化的配置项中，设置enableSPA为`true`，即会开启页面的Hashchange事件监听（触发重新上报PV），并将URL Hash作为其他数据上报中的page字段。

与enableSPA配套的还有parseHash，更多信息，请参见enableSPA和parseHash。

### 完全手动上报

此方法可用于所有的单页面应用场景。如果第一种方法无效，则可用此方法。

SDK提供了setPage方法来手动更新数据上报时的page name。调用此方法时，默认会重新上报页面PV。更多信息，请参见setPage()。

```
// 监听应用路由变更事件。
app.on('routeChange', function (next) {
    __bl.setPage(next.name);
});
```

### 相关文档

- SDK参考
- API参考

## 6.2. 数据预上报

在某些情况下，例如SDK尚未完成初始化时，会导致数据上报出现问题。本文介绍如何使用ARMS前端监控SDK实现数据预上报。

### 会导致数据上报出现问题的情形

以下情形会导致数据上报出现问题：

- 在页面刚刚加载时，有一些数据需要上报，但此时SDK可能还未完成初始化，或者不确定是否已完成初始化。
- 在应用的初始化逻辑中调用setConfig方法，但由于SDK是异步加载的，此时可能还未加载完成。

### 解决办法

SDK在`__bl`对象上增加了一个pipe属性，用于将预调用的信息缓存到此变量中。例如：

```
__bl.pipe = [
    // 将当前页面的HTML也作为一个API上报。
    ['api', '/index.html', true, performance.now, 'SUCCESS'], //相当于__bl.api(api, success,
time, code, msg)。
    // SDK初始化完成后即开启SPA自动解析。
    ['setConfig', {enableSPA: true}]
];
```

如果只上报单条数据，也可以直接写成：

```
__bl.pipe = ['msg', '我是另一个普通的消息'];
```

其中数组的第0个表示方法名，后面依次是入参。SDK初始化完成后，就会依次调用预先挂载到 `window.__bl.pipe` 上的方法及参数。

> **说明** 在SDK初始化完成前，如果多次设置 `__bl.pipe` 的值，则以最后设置的为准。

如果不能确定SDK是否初始化完成，又不想添加太多判断逻辑，也可以在SDK初始化完成后调用pipe（支持IE9及以上）。

例如，单页面应用中，设置 `autoSend: false` 后，在应用初始化后上报第一次PV，此时并不确定SDK是否初始化完成。

```
// 设置页面name为 'homepage'，并且上报PV。
__bl.pipe = ['setPage', 'homepage'];
```

## 相关文档

- SDK参考
- API参考

# 7.SDK参考

ARMS前端监控提供一系列SDK配置项，让您能够通过设置参数来满足额外需求，例如忽略指定URL、API、JS错误的上报、通过过滤URL中的非关键字符使页面聚类、通过随机采样上报来减小上报量并降低负载等。

## 本页索引



## 如何使用前端监控SDK配置项

可通过以下两种方式使用前端监控SDK配置项：

- 向页面插入BI探针时在config中添加额外参数。
  例如，以下示例代码的config中，除了默认的pid参数外，还添加了用于单页面应用（Single Page Application）场景的enableSPA参数。

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxxxxx",enableSPA:true};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("cros
sorigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

- 页面初始化完成后，在前端JS代码中调用setConfig方法来修改配置项。
  `__bl.setConfig(next)` 调用参数说明：

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| next | Object | 需要修改的配置项以及值 | 是 | 无 |

  示例：修改disableHook禁用API自动上报。

```
__bl.setConfig({
    disableHook: true
});
```

## pid

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| pid | String | 项目唯一ID，由ARMS在创建站点时自动生成。 | 是 | 无 |

[回到顶部]

## uid

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| uid | String | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK随机自动生成且每半年更新一次。 | • Weex场景：必需<br>• 其他场景：非必需 | • Weex场景：无<br>• 其他场景：由SDK自动生成 |

示例：调用setConfig方法修改SDK配置项。

```
__bl.setConfig({
    uid: 12345
});
```

**说明** 对于setConfig方法，小程序场景不支持配置uid，您可以使用setUsername代替uid标识用户。

[回到顶部]

## tag

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| tag | String | 传入的标记，每条日志都会携带该标记。 | 否 | 无 |

[回到顶部]

## page

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| page | String | 页面名称。 | 否 | 默认取当前页面URL的关键部分：`host + pathname`。 |

[回到顶部]

## setUsername

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| setUsername | Function | 用于设置一个方法，该方法需要返回类型为String的用户名称。 | 否 | 无 |

本配置项用于设置一个方法，该方法需要返回类型为String的用户名称。配置后可以根据用户名称进行全链路追踪（会话追踪）、查询用户会话轨迹，便于排查问题。

> **说明** 若页面加载初期暂时无法获取用户名，则可设置返回Null，但不要设置返回一个临时用户名。因为SDK发送日志时会再次调用setUsername方法获取用户名，如果之前设置了返回临时用户名，则会一直沿用最初设置的用户名，不再重新获取。

示例：调用setConfig方法修改SDK配置项。

```
__bl.setConfig({
    setUsername: function () {
        return "username_xxx";
    }
});
```

[回到顶部]

## enableSPA

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| enableSPA | Boolean | 监听页面的hashchange事件并重新上报PV，适用于单页面应用场景。 | 否（仅Web场景支持） | `false` |

[回到顶部]

## parseHash

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| parseHash | Function | 与enableSPA搭配使用。 | 否 | 见下文 |

单页面应用场景中（参见SPA页面上报），在enableSPA设为 `true` 的前提下，页面触发hashchange事件时，parseHash参数用于将URL Hash解析为Page字段。

**默认值**

其默认值由一个简单的字符串处理方法获得：

```
function (hash) {
    var page = hash ? hash.replace(/^#/, '').replace(/\?.*$/, '') : '';
    return page || '[index]';
}
```

此项一般情况下不需要修改。如果需要在上报时使用自定义的页面名，或者URL的Hash比较复杂，则需要修改此配置项。例如：

```
// 定义页面Hash和Page的映射关系。
var PAGE_MAP = {
    '/': '首页',
    '/contact': '联系我们',
    '/list': '数据列表',
    // ...
};
// 页面load后调用SDK方法。
window.addEventListener('load', function (e) {
    // 调用setConfig方法修改SDK配置项。
    __bl.setConfig({
        parseHash: function (hash) {
            key = hash.replace(/\?.*$/, '');
            return PAGE_MAP[key] || '未知页面';
        }
    });
});
```

[回到顶部]

## disableHook

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| disableHook | Boolean | 禁用AJAX请求监听。 | 否 | `false`：默认会监听并用于API调用成功率上报。 |

[回到顶部]

## ignoreUrlCase

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| ignoreUrlCase | Boolean | 忽略Page URL大小写。 | 否 | `true`：默认忽略。 |

[回到顶部]

## urlHelper

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| urlHelper | * | 代替旧参数ignoreUrlPath，用于配置URL过滤规则。 | 否 | 见下文 |

当页面URL类似于 `http://example.com/projects/123456` （projects后面的数字是项目ID）时，如果将 `example.com/projects/123456` 作为page上报，会导致在数据查看时页面无法聚成一类。这种情况下，为了使同类页面聚类，可以使用urlHelper参数过滤掉非关键字符，例如此例中的项目ID。

> **注意**
> - 用于URL过滤规则的旧参数ignoreUrlPath现已废弃，请使用新参数urlHelper。若继续使用旧参数且不使用新参数，配置仍将生效。若同时使用新旧参数，则新参数的配置生效。
> - 此配置项只在自动获取页面URL作为Page时才会生效。如果手动调用setPage或setConfig方法（参考API参考）修改过Page，或者如果enableSPA已设为 `true` ，则此设置项无效。

**默认值**

此配置项的默认值是以下数组，一般情况下不需要修改。

```
[
    // 将所有Path中的数字变成*。
    {rule: /\/([a-z\-_]+)?\d{2,20}/g, target: '/$1**'},
    // 去掉URL末尾的'/'。
    /\/$/
]
```

此设置项的默认值会过滤掉 `xxxx/123456` 后面的数字，例如 `xxxx/00001` 和 `xxxx/00002` 都会变成 `xxxx/**` 。

**值类型**

urlHelper的值可以是以下类型：

- `String` 或 `RegExp` （正则表达式）：将匹配到的字符串去掉。
- `Object<rule, target>` ：对象包含两个Key（rule和target）作为JS字符串的replace方法的入参。使用方法参见JS相关教程中的String::replace方法。
- `Function` ：将原字符串作为入参执行方法，将执行结果作为Page。
- `Array` ：用于设置多条规则，每条规则都可以是上述类型之一。

[回到顶部]

## apiHelper

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| apiHelper | * | 代替旧参数ignoreApiPath，用于配置API过滤规则。 | 否 | 见下文 |

用于在自动上报API时过滤接口URL中的非关键字符，用法及含义同urlHelper。

> **注意** 用于API过滤规则的旧参数ignoreApiPath现已废弃，请使用新参数apiHelper。若继续使用旧参数且不使用新参数，配置仍将生效。若同时使用新旧参数，则新参数的配置生效，旧参数的配置不生效。

**默认值**

默认值是一个对象，一般情况下不需要修改：

```
{rule: /(\w+)\/\d{2,}/g, target: '$1'}
```

此设置项的默认值会过滤掉接口URL中类似 `xxxx/123456` 后面的数字。

如果需要在API中上报 `?` 后面的参数，例如： `https://arms.console.aliyun.com/apm?pid=fr6fbgbeot` 中的pid参数，需要按照以下方法手动上报API数据：

- 使用ignore方法，关闭自动上报。具体操作，请参见ignore。
- 使用api()自行上报API数据。具体操作，请参见api()。

[回到顶部]

## parseResponse

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| parseResponse | Function | 用于解析自动上报API时返回的数据。 | 否 | 见下文 |

该配置项用于解析自动上报API时返回的数据。

**默认值**

该配置项的默认值为：

```
function (res) {
    if (!res || typeof res !== 'object') return {};
    var code = res.code;
    var msg = res.msg || res.message || res.subMsg || res.errorMsg || res.ret || res.errorR
esponse || '';
    if (typeof msg === 'object') {
        code = code || msg.code;
        msg = msg.msg || msg.message || msg.info || msg.ret || JSON.stringify(msg);
    }
    return {msg: msg, code: code, success: true};
}
```

以上代码会解析返回的数据，尝试提取出 `msg` 和 `code` 。对于一般应用来说，此设置项不需要修改。如果默认值无法满足业务需求，则可以重新设置。

[回到顶部]

## ignore

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| ignore | Object | 忽略指定URL/API/JS错误。符合规则的日志将被忽略且不会被上报，包含子配置项ignoreUrls、ignoreApis、ignoreErrors和ignoreResErrors。 | 否 | 见下文 |

ignore的值是一个对象，对象中包含4个属性：ignoreUrls、ignoreApis、ignoreErrors和ignoreResErrors。可单独设置其中的1个或多个属性。

**默认值**

该配置性的默认值为：

```
ignore: {
        ignoreUrls: [],
        ignoreApis: [],
        ignoreErrors: [],
        ignoreResErrors: []
    },
```

### ignoreUrls

ignoreUrls表示忽略某些URL，符合规则的URL下的日志都不会被上报。值可以是 `String` 、 `RegExp` 、 `Function` 或者以上三种类型组成的数组。示例：

```
__bl.setConfig({
                ignore: {
                    ignoreUrls: [
                    'http://host1/',  // 字符串
                    /.+?host2.+/,     // 正则表达式
                    function(str) {   // 方法
                        if (str && str.indexOf('host3') >= 0) {
                            return true;   // 不上报
                        }
                        return false;      // 上报
                    }]
                }
            });
```

### ignoreApis

ignoreApis表示忽略某些API，符合规则的API将不会被监控。值可以是 `String` 、 `RegExp` 、 `Function` 或者以上三种类型组成的数组。示例：

```
__bl.setConfig({
                ignore: {
                    ignoreApis: [
                    'api1','api2','api3', // 字符串
                    /^random/,  // 正则表达式
                    function(str) { // 方法
                        if (str && str.indexOf('api3') >= 0) return true;   // 不上报
                        return false;   // 上报
                    }]
                }
            });
```

### ignoreErrors

ignoreErrors表示忽略某些JS错误，符合规则的JS错误不会被上报。值可以是 `String` 、 `RegExp` 、 `Function` 或者以上三种类型组成的数组。示例：

```
__bl.setConfig({
                ignore: {
                    ignoreErrors: [
                    'test error', // 字符串
                    /^Script error\.?$/, // 正则表达式
                    function(str) { // 方法
                        if (str && str.indexOf('Unkown error') >= 0) return true;   // 不上报
                        return false;   // 上报
                    }]
                }
            });
```

ignoreResErrors

ignoreResErrors表示忽略指定的资源错误，符合规则的资源错误不会被上报。值可以是 `String` 、 `RegExp` 、 `Function` 或者以上三种类型组成的数组。示例：

```
__bl.setConfig({
                ignore: {
                    ignoreResErrors: [
                    'http://xx/picture.jpg', // 字符串
                    /jpg$/, // 正则表达式
                    function(str) { // 方法
                        if (str && str.indexOf('xx.jpg') >= 0) return true;   // 不上报
                        return false;   // 上报
                    }]
                }
            });
```

[回到顶部]

## disabled

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| disabled | Boolean | 禁用日志上报功能。 | 否 | `false` |

[回到顶部]

## sample

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| sample | Integer | 日志采样配置，值为1~100的整数。对性能日志和成功API日志按照 `1/sample` 的比例采样，关于性能日志和成功API日志的指标说明，请参见统计指标说明。 | 否 | `1` |

sample配置项作用：

- 减小API的数据上报量（降低部分费用），更精确的方案建议使用ignore参数过滤掉某些不需要监控的API

数据，更多信息，请参考ignore。

- 降低采集数据的性能开销。

sample配置项说明：

- 为了减小上报量和降低负载，该配置项会对性能和成功API日志进行随机采样上报，后台处理日志时会根据对应的采样配置进行还原，因此不会影响JS错误率和失败API率等指标的计算。但是会导致在查询API详情或者相关数据明细的时候，无法保证百分之百可以查看明细。
- 采样值 `sample` 默认为 `1` ，值可以是1~100的整数，对应的采样率为 `1/sample` ，例如： `1` 表示100% 采样， `10` 表示10% 采样， `100` 表示1% 采样。

**警告** 由于是随机采样，如果原上报量较小，该配置项可能会造成较大的统计结果误差，建议仅日均PV在100万以上的站点使用该配置项。

[回到顶部]

## sendResource

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| sendResource | Boolean | 上报页面静态资源。 | 否 | `false` |

如果sendResource配置为 `true` ，则页面load事件触发时会上报当前页面加载的静态资源信息。如果发现页面加载较慢，可以在会话追踪页面查看页面加载的静态资源瀑布图，判断页面加载较慢的具体原因。

sendResource示例代码：

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxxxxx",sendResource:true};
with(b)with(body)with(insertBefore(createElement("script"),firstChild))setAttribute("crosso
rigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

**说明** 由于是在页面load事件触发时判断，所以请在config中配置sendResource（如以上示例所示），不要使用setConfig的方式，因为这种方式可能在页面load事件完成后才会触发，从而使该配置项会无效。

[回到顶部]

## useFmp

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| useFmp | Boolean | 采集首屏FMP（First Meaningful Paint，首次有效渲染）数据。 | 否 | `false` |

[回到顶部]

## enableLinkTrace

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| enableLinkTrace | Boolean | 进行前后端链路追踪，请参见使用前后端链路追踪诊断API错误原因。 | 否（仅Web场景、支付宝小程序、微信小程序、钉钉小程序支持） | `false` |

[回到顶部]

## release

> **注意** 请在初始化时的config中配置release，不要使用setConfig的方式。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| release | String | 应用版本号。建议您配置，便于查看不同版本的上报信息。 | 否 | `undefined` |

[回到顶部]

## environment

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| environment | String | 环境字段，取值为：prod、gray、pre、daily和local，其中：<br>• prod表示线上环境。<br>• gray表示灰度环境。<br>• pre表示预发环境。<br>• daily表示日常环境。<br>• local表示本地环境。 | 否 | `prod` |

[回到顶部]

## behavior

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| behavior | Boolean | 是否为了便于排查错误而记录报错的用户行为。 | 否（仅Web场景和小程序场景支持） | Browser默认为`true`，小程序默认为`false`。 |

[回到顶部]

## autoSendPerf

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| autoSendPerf | Boolean | 是否允许自动发送性能日志。 | 否 | `true` |

[回到顶部]

## c1\c2\c3

除了以上列出的配置项，您可能还需要更多的附加信息辅助处理业务问题，因此ARMS SDK提供了3个可自定义字段的配置项，配置后字段内容会包含在每一条上报的日志中。

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| c1 | String | 业务自定义字段，每条日志都会携带该标记。 | 否 | 无 |
| c2 | String | 业务自定义字段，每条日志都会携带该标记。 | 否 | 无 |
| c3 | String | 业务自定义字段，每条日志都会携带该标记。 | 否 | 无 |

[回到顶部]

## 相关文档

- API参考
- SPA页面上报
- 数据预上报

# 8.API参考

前端监控SDK开放了部分接口，包括用于上报数据的数据上报类接口和用于修改SDK配置项的方法类接口。

## 本页索引



### `api()`

调用 `api()` 接口来上报页面的API调用成功率。

SDK默认会监听页面的AJAX请求并调用此接口上报。 如果页面的数据请求方式是JSONP或者其他自定义方法（例如客户端SDK等），可以在数据请求方法中调用 `api()` 方法手动上报。

> **说明** 如果要调用此接口，建议在SDK配置项中将disableHook设置为*true*。更多信息，请参见disableHook。

`api()` 语法：

```
__bl.api(api, success, time, code, msg, begin, traceId, sid)
```

或者

```
__bl.api({api: xxx, success: xxx, time: xxx, code: xx, msg: xx, begin: xx, traceId: xx, sid
: xx})
```

`api()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| api | String | 接口名 | 是 | 无 |
| success | Boolean | 是否调用成功 | 是 | 无 |
| time | Number | 接口耗时 | 是 | 无 |
| code | String/Number | 返回码 | 否 | '' |
| msg | String | 返回信息 | 否 | '' |
| begin | Number | API请求开始时间戳 | 否 | '' |
| traceId | String | EagleEye-TraceID的值 | 否 | '' |
| sid | String | EagleEye-SessionID的值 | 否 | '' |

`api()` 使用示例：

```
var begin = Date.now(),
    url = '/data/getTodoList.json',
    traceId = window.__bl && __bl.getTraceId('EagleEye-TraceID'),
    sid = window.__bl && __bl.getSessionId('EagleEye-SessionID');
// 备注：请求的header部分请加入EagleEye-TraceID、EagleEye-SessionID
fetch(url, {
    headers: {
        'EagleEye-TraceID': traceId,
        'EagleEye-SessionID': sid
    }
}).then(function (result) {
    var time = Date.now() - begin;
    // 上报接口调用成功
    window.__bl && __bl.api(url, true, time, result.code, result.msg, begin, traceId, sid);
    // do something ....
}).catch(function (error) {
    var time = Date.now() - begin;
    // 上报接口调用失败
    window.__bl && __bl.api(url, false, time, 'ERROR', error.message, begin, traceId, sid);
    // do something ...
});
```

[回到顶部]

**error()**

调用 `error()` 接口来上报页面中的JS错误或使用者想关注的异常。

一般情况下，SDK会监听页面全局的Error并调用此接口上报异常信息，但由于浏览器的同源策略往往无法获取错误的具体信息，此时就需要使用者手动上报。

`error()` 语法：

```
__bl.error(error, pos)
```

`error()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| error | Error | JS的Error对象 | 是 | 无 |
| pos | Object | 错误发生的位置，包含pos.filename、pos.lineno和pos.colno属性。 | 否 | 无 |
| pos.filename | String | 错误发生的文件名 | 否 | 无 |
| pos.lineno | Number | 错误发生的行数 | 否 | 无 |
| pos.colno | Number | 错误发生的列数 | 否 | 无 |

`error()` 使用示例1：监听页面的JS Error并上报。

```
window.addEventListener('error', function (ex) {
    // 一般事件的参数中会包含pos信息。
    window.__bl && __bl.error(ex.error, ex);
});
```

`error()` 使用示例2：上报一个自定义的错误信息。

```
window.__bl && __bl.error(new Error('发生了一个自定义的错误'), {
    filename: 'app.js',
    lineno: 10,
    colno: 15
});
```

`error()` 使用示例3：上报一个自定义类型的错误信息。

```
__bl.error({name:'CustomErrorLog',message:'this is an error'}, {
    filename: 'app.js',
    lineno: 10,
    colno: 15
});
```

[回到顶部]

**sum()**

调用 `sum()` 方法，可以自定义上报的日志，它通常被用于统计业务中特定事件发生的次数。通过 `sum()` 方法上报的数据，您可以在**自定义统计**页面查看：

- 自定义事件发生的趋势图。
- 发生该事件的PV或UV统计。
- 维度分布信息。

`sum()` 语法：

```
__bl.sum(key, value)
```

`sum()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| key | String | 事件名 | 是 | 无 |
| value | Number | 单次累加上报量 | 否 | 1 |

`sum()` 使用示例：

```
__bl.sum('event-a');
__bl.sum('event-b', 3);
```

[回到顶部]

**avg()**

调用 `avg()` 可以自定义上报的日志，它通常被用于计算业务中特定事件发生的平均次数或平均值。通过 `avg()` 方法上报的数据，您可以在**自定义统计**页面查看：

- 自定义事件发生的趋势图。
- 发生该事件的PV或UV统计。
- 维度分布信息。

`avg()` 语法：

```
__bl.avg(key, value)
```

`avg()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| key | String | 事件名 | 是 | 无 |
| value | Number | 统计上报量 | 否 | 0 |

`avg()` 使用示例：

```
__bl.avg('event-a', 1);
__bl.avg('event-b', 3);
```

[回到顶部]

**`reportBehavior()`**

调用 `reportBehavior()` 接口即刻上报当前行为队列。

若不手动调用此接口，发生JS Error会自动触发上报当前行为队列，最大为100条，若超出100条，队列头的行为将被舍弃。

> **说明** 您需要在SDK配置中将behavior配置为*true*，才可调用此方法。

`reportBehavior()` 语法：

```
__bl.reportBehavior()
```

`reportBehavior()` 调用参数：无请求参数。

[回到顶部]

**`performance()`**

> **注意** 仅支持在Web端使用。

在页面onLoad之后调用 `performance()` 接口来上报除默认性能指标外的自定义性能指标。

> **说明** 调用此方法的时间必须是在页面onLoad之后，否则会因尚未完成默认性能指标采集而导致调用无效。一次PV期间只能有效调用一次。

`performance()` 使用方法：

1. 将SDK配置项autoSendPerf设置为*false*，从而关闭自动上报性能指标，并等待手动触发上报。
2. 调用 `__bl.performance(Object)` 方法来手动上报自定义指标，此过程中也会自动上报默认性能指标。

`performance()` 使用示例1（以CDN方式接入时）：

```
window.onload = () => {
  setTimeout(()=>{
    __bl.performance({cfpt:100, ctti:200, t1:300, …});
  }, 1000); //设置一定的延时，确保默认性能数据采集完成。
};
```

`performance()` 使用示例2（以npm包方式接入时）：

```
const BrowserLogger = require('alife-logger');
const __bl = BrowserLogger.singleton({pid:'站点唯一ID'});
window.onload = () => {
  setTimeout(()=>{
    __bl.performance({cfpt:100, ctti:200, t1:300, …});
  }, 1000); //设置一定的延时，确保默认性能数据采集完成。
};
```

> **说明** 自定义性能指标含义：
>
> - cfpt：自定义首次渲染时间
> - ctti：自定义首次可交互时间
> - t1～t10：其他自定义性能指标（共计10个）

[回到顶部]

### `setConfig()`

在SDK初始完成后调用 `setConfig()` 接口来重新修改部分配置项，关于SDK配置项的详细信息，请参见SDK配置项。

`setConfig()` 语法：

```
__bl.setConfig(next)
```

`setConfig()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| next | Object | 需要修改的配置项以及值 | 是 | 无 |

`setConfig()` 使用示例：修改disableHook禁用API自动上报。

```
__bl.setConfig({
    disableHook: true
});
```

[回到顶部]

**setPage()**

调用 `setPage()` 接口来重新设置页面的page name（默认会触发重新上报PV）。此接口一般用于单页面应用，更多信息，请参见SPA页面上报。

`setPage()` 语法：

```
__bl.setPage(page, sendPv)
```

`setPage()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| page | String | 新的page name。 | 是 | 无 |
| sendPv | Boolean | 是否上报PV，默认会上报。 | 否 | true |

`setPage()` 使用示例：

```
// 设置当前页面的page name为当前的URL hash，并重新上报PV
__bl.setPage(location.hash);
// 仅设置当前页面的page为'homepage'，但不触发PV上报
__bl.setPage('homepage', false);
```

[回到顶部]

**setCommonInfo()**

调用 `setCommonInfo()` 用于设置公共字段。

`setCommonInfo()` 语法：

```
__bl.setCommonInfo(obj)
```

`setCommonInfo()` 其中入参是object，示例如下：

```
__bl.setCommonInfo({
  name: 'xxx',
  common: 'xxx'
});
```

> **说明** 该方法建议object信息不要太大，否则容易导致get请求超长，从而导致请求失败的问题。

[回到顶部]

**addBehavior()**

调用 `addBehavior()` 接口在当前行为队列末尾添加一条自定义用户行为。

SDK维护一条最大长度为100条的用户行为队列，调用 `addBehavior()` 接口可在当前行为队列末尾添加一条自定义用户行为，当发生JS error时上报当前的行为队列，并将队列清空。

您可以在**JS错误诊断**页面查看用户行为回溯，具体操作，请参见使用用户行为回溯诊断JS错误。

> **说明** 您需要在SDK配置中将behavior配置为*true*，才可调用此方法。

`addBehavior()` 语法：

```
__bl.addBehavior(behavior)
```

`addBehavior()` 调用参数

| 参数 | 类型 | 描述 | 是否必选 | 默认值 |
| --- | --- | --- | --- | --- |
| data | Object | 行为数据。包含：<br>• name：行为名称，String类型，必填，最大长度20字符。<br>• message：行为内容，String类型，必填，最大长度200字符。 | 是 | 无 |
| page | String | 行为发生的页面 | 否 | location.pathname的值 |

`addBehavior()` 使用示例：

```
_bl.addBehavior({
  data:{name:'string',message:'sting'},
  page:'string'
})
```

[回到顶部]

## 创建多实例方法

创建多实例请使用NPM包方式，现在正式版NPM包已发布，NPM包：alife-logger 。

- 使用方法：

```
const BrowerLogger = require('alife-logger');
const bl2 = BrowerLogger.createExtraInstance(props); // 通过createExtraInstance 创建实例
bl2.custom({
  key: 'biz',
  msg: 'msg info'
});
```

> **说明** 其中props为Object，具体的参数与SDK的config基本保持一致。

- 如果新实例只上报自定义信息：

```
var props = {
  pid: 'xxxx', //新实例需要上报的站点
  region: 'cn',
  page: '',
  uid: ''
}
```

- 使用方法：

```
const WeexLogger = require('alife-logger/weex');
const wl2 = WeexLogger.createExtraInstance(props); // 通过createExtraInstance 创建实例
wl2.custom({
  key: 'biz',
  msg: 'msg info'
});
```

> **说明** 其中props为Object，具体的参数与SDK的config基本保持一致。

- 如果新实例只上报自定义信息：

```
var props = {
  pid: 'xxxx', //新实例需要上报的站点
  region: 'cn',
  sendRequest: function(data, imgUrl) {
    // 发送Get日志方案
  },
  postRequest: function(data, imgUrl) {
    // 发送POST日志方案
  }
}
```

使用方法：

```
import MiniProgramLogger from 'alife-logger/miniprogram'; // 具体路径根据是钉钉小程序、支付宝小程序及其他类小程序来选择对应的路径
const MiniInstance = MiniProgramLogger.createExtraInstance({
  pid: 'xxxinstance',
  uid: 'userxxx', // 用来设置用户uid，统计UV信息
  region: 'cn', // region为cn、sg分别表示是日志上报到中国服务器、还是新加坡服务器，如不配置默认是中国服务器
  // 基础小程序监控需要手动传入rpc(方法实现按照实际业务来写)
  sendRequest: (url, resData) => {
    // 发送方法
  }
});
```

## Web页面

## Weex页面

## 小程序页面

## 相关文档

- SDK参考
- SPA页面上报

# 9.统计指标说明

本文说明ARMS前端监控各页面的关键统计指标含义以及日志字段的含义。

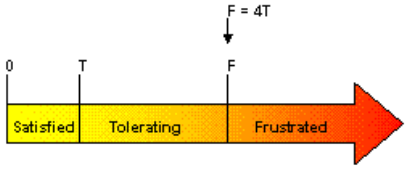
## 满意度

性能指数APDEX（全称Application Performance Index）是一个国际通用的应用性能计算标准。该标准将用户对应用的体验定义为三个等级：

- 满意（0~T）
- 可容忍（T~4T）
- 不满意（大于4T）

计算公式为：

```
Apdex=(满意数+可容忍数/2)/总样本量
```




图片来源：apdex.org

ARMS取页面首次渲染时间（First Paint Time）作为计算指标，默认定义T为2秒。

## JS稳定性

JS稳定性在ARMS中是指页面的JS错误率。

在一个PV周期内，如果发生过错误（JS Error），则此PV周期为错误样本。

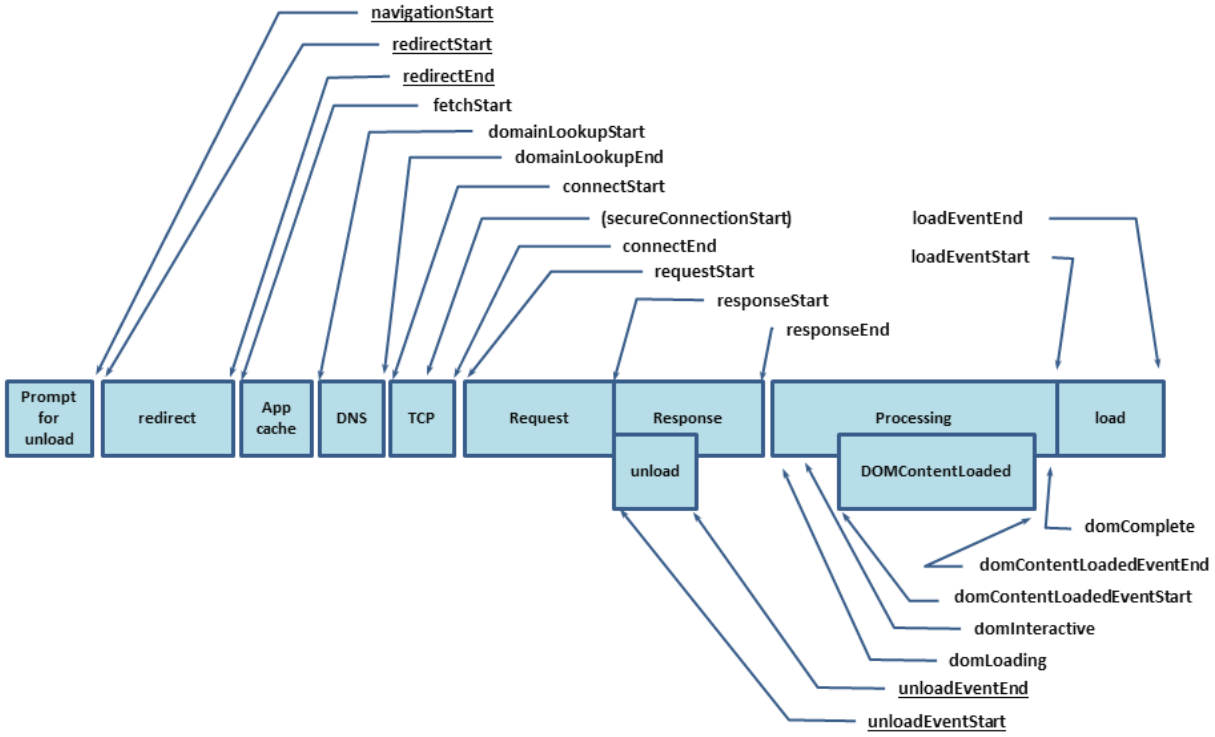
```
错误率=错误样本量/总样本量
```

页面异常除了自动上报的JS Error外，也包括手动调用API参考上报的错误。

## 访问速度

在ARMS中，访问速度是指页面的*首次渲染时间*。

在性能测速统计中，所有数据都是根据W3C规范中定义的Navigation Timing API计算出来的。




图片来源：www.w3.org

Web端关键性能指标

| 上报字段 | 描述 | 计算公式 | 备注 |
| --- | --- | --- | --- |
| FMP（First Meaningful Paint） | 首屏时间 | 参见FMP技术实现方案 | 无 |
| FPT（First Paint Time） | 首次渲染时间（白屏时间） | responseEnd - fetchStart | 从请求开始到浏览器开始解析第一批HTML文档字节的时间差。 |
| TTI（Time to Interact） | 首次可交互时间 | domInteractive - fetchStart | 浏览器完成所有HTML解析并且完成DOM构建，此时浏览器开始加载资源。 |
| Ready | HTML加载完成时间， 即DOM Ready时间。 | domContentLoadEventEnd - fetchStart | 如果页面有同步执行的JS，则同步JS执行时间=Ready-TTI。 |
| Load | 页面完全加载时间 | loadEventStart - fetchStart | Load=首次渲染时间+DOM解析耗时+同步JS执行+资源加载耗时。 |
| FirstByte | 首包时间 | responseStart - domainLookupStart | 无 |

区间段耗时字段含义

| 上报字段 | 描述 | 计算公式 | 备注 |
| --- | --- | --- | --- |
| DNS | DNS查询耗时 | domainLookupEnd - domainLookupStart | 无 |
| TCP | TCP连接耗时 | connectEnd - connectStart | 无 |
| TTFB（Time to First Byte） | 请求响应耗时 | responseStart - requestStart | TTFB有多种计算方式，ARMS采用的标准，请参见Google Development定义。 |
| Trans | 内容传输耗时 | responseEnd - responseStart | 无 |
| DOM | DOM解析耗时 | domInteractive - responseEnd | 无 |
| Res | 资源加载耗时 | loadEventStart - domContentLoadedEventEnd | 表示页面中的同步加载资源。 |
| SSL | SSL安全连接耗时 | connectEnd - secureConnectionStart | 只在HTTPS下有效。 |

小程序关键性能指标

| 上报字段 | 描述 | 计算方式 | 备注 |
| --- | --- | --- | --- |
| FPT（First Paint Time） | 首次渲染时间 | onShow (first page) - onLaunch (app) | 小程序从onLaunch到第一个页面onShow之间的时间。 |

## API成功率

`API`**成功率=接口调用成功的样本量/总样本量**

统计API成功率的样本除了自动上报的AJAX请求，还包括手动调用API参考上报的数据。

## 日志字段

下列表格说明了日志中的字段含义。

通用字段

| 字段 | 含义 |
| --- | --- |
| uid | 用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK自动生成且每半年更新一次。 |
| username | 用户名称，需要通过SDK主动上报，否则内容为空。 |
| release | 应用版本号 |
| environment | 生产环境 |
| page | 页面 |

| 字段 | 含义 |
| --- | --- |
| sampling | 采样率 |
| tag | 用户自定义Tag |

API

| 字段 | 含义 |
| --- | --- |
| api | API请求地址，不带参数。 |
| msg | responseText：字符串形式的响应数据。 |
| code | 状态码 |
| time | API耗时 |
| success | API成功与否 |

JS错误

| 字段 | 含义 |
| --- | --- |
| msg | 报错内容 |
| stack | 错误堆栈 |
| cate | 错误类型 |
| file | 出错文件 |
| line | 出错行 |
| col | 出错列 |
| times | 出错次数 |

## 日志说明

多维分析

| 日志类型 | Type | 查询字段<br>（通用指标字段：所有日志皆可查询过滤） |
| --- | --- | --- |
| PV日志 | PV | 在页面Onload的时候上传，用于计算PV和UV。<br>PV、UV计算方式：<br>• PV：PV日志的条数。<br>• UV：对于PV日志，按照UID去重。 |
| 性能日志 | Perf | 性能指标 |

| 日志类型 | Type | 查询字段<br>（通用指标字段：所有日志皆可查询过滤） |
| --- | --- | --- |
| 慢加载日志<br>（>8s的性能日志） | RES | 性能指标 |
| JS错误日志 | Error | • JS错误消息<br>• JS文件URL<br>• JS错误类型 |
| API日志 | API | • API名称<br>• API消息<br>• HTTP状态码<br>• API耗时<br>• 域名<br>• API是否报错<br>• TraceID |
| SUM日志 | SUM | 自定义Key：事件名（如scroll-count） |
| AVG日志 | AVG | 自定义Key：事件名（如scroll-time） |
| 资源错误日志 | ResourceError | 资源错误SRC |
| 无 | Custom | 无 |

# 10.前端监控常见问题

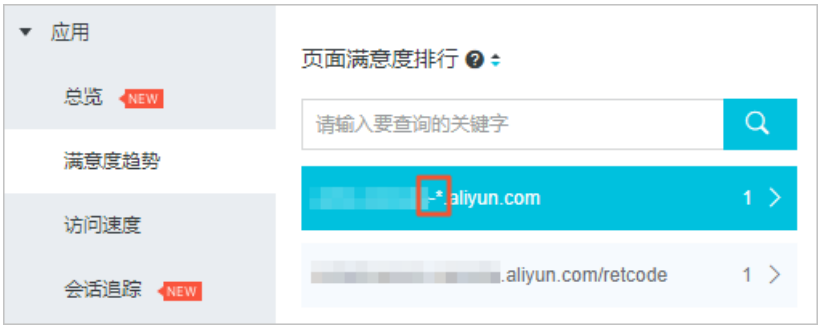
本文解答了关于前端监控的常见问题。

## 本页目录



## 为什么有些监控页面或API名称中出现了星号（*）？

ARMS前端监控的页面统计是以实际打开的页面URL为基础，对各个维度进行统计。监控页面或API名称中的星号（*）并不是真实页面URL的一部分，而是表示这是经过URL收敛后的结果。换言之，它表示这不是一个具体的URL，而是一些相似URL的集合。




URL收敛算法说明

- 问题：“变量”会导致同类URL发散成多个，因而难以监控和分析。
- 目标：合并同类URL，用星号（*）代替不断变化的“变量”。
- 方案：采用我们自研的URL收敛算法，在尽可能保留语义信息的前提下，合并同类URL，减少URL总数。主要分为以下两步。
  - 聚类：将相似URL归纳为一组。

- “变量”识别：提取同组URL中不断变化的“变量”，并以星号（*）代替。

收敛过程如下图所示。


解决方案

关闭URL收敛的具体操作，请参考urlHelper。

[回到顶部]

## 为什么页面访问量列表和页面访问速度列表不一致？

这是因为您的应用是SPA（ Single-Page Application，单页面应用），且开启了SPA自动解析。在SPA的应用场景下，页面访问量和页面访问速度的统计方法如下。

- 页面访问量：触发hashchange事件后会自动上报PV，以统计该Hash值对应页面的PV情况。所以在查看SPA应用的页面访问量列表时，可以查看对应的Hash页面的具体PV。
- 页面访问速度：因为SPA应用切换Hash值后，页面的访问速度不会变化，所以访问速度的统计不以Hash值为维度，这样不仅可以减少不必要的上报量，而且还可以清晰了解页面的性能情况。

[回到顶部]

## 为什么API日志中没有生成TraceId，导致无法跳转至对应的应用监控？

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**，然后单击目标应用名称。
3. 在左侧导航栏中选择**设置 > 应用设置**。
4. 请在**接入步骤**页签检查您的ARMS探针配置项是否选中**与应用监控关联**。若未选中，请选中后，再重新将ARMS探针接入前端应用。


检查API日志中是否生成TraceId。若仍未生成TraceId，则执行步骤。

5. 检查您的页面请求和API请求的域名是否一致。为防止因跨域名验证导致API请求失败，因此若存在跨域名访问的情况，则不能正常生成TraceId。

   解决方案，请参考API与当前应用域名非同源。

[回到顶部]

## 为什么在诊断JS错误时，会显示Source Map文件错误？

1. 请确保文件后缀名为*.js.map*。
2. 请确保账号有ARMS的写入权限，如果账号没有权限，请联系账号管理员。

[回到顶部]

## 用户控制台的设置和setConfig的关系是什么？

控制台的设置仅可以帮助您快速生成相应的配置代码，生成的代码仅在发布后生效，而修改setConfig是直接生效的。


此外，控制台的设置只能在接入时使用，接入完成后，您需要通过setConfig来修改配置。

[回到顶部]

## 资源消耗图展示的资源消耗和计费流量的关系是什么？

资源消耗图展示的是真实的资源消耗，您在设置消费限制中设置的也是真实消耗的上限额度，更多信息，请参见设置消费限制。真实的资源消耗中，只有部分流量是计费的。

[回到顶部]

## SDK中应该如何配置环境和版本号？

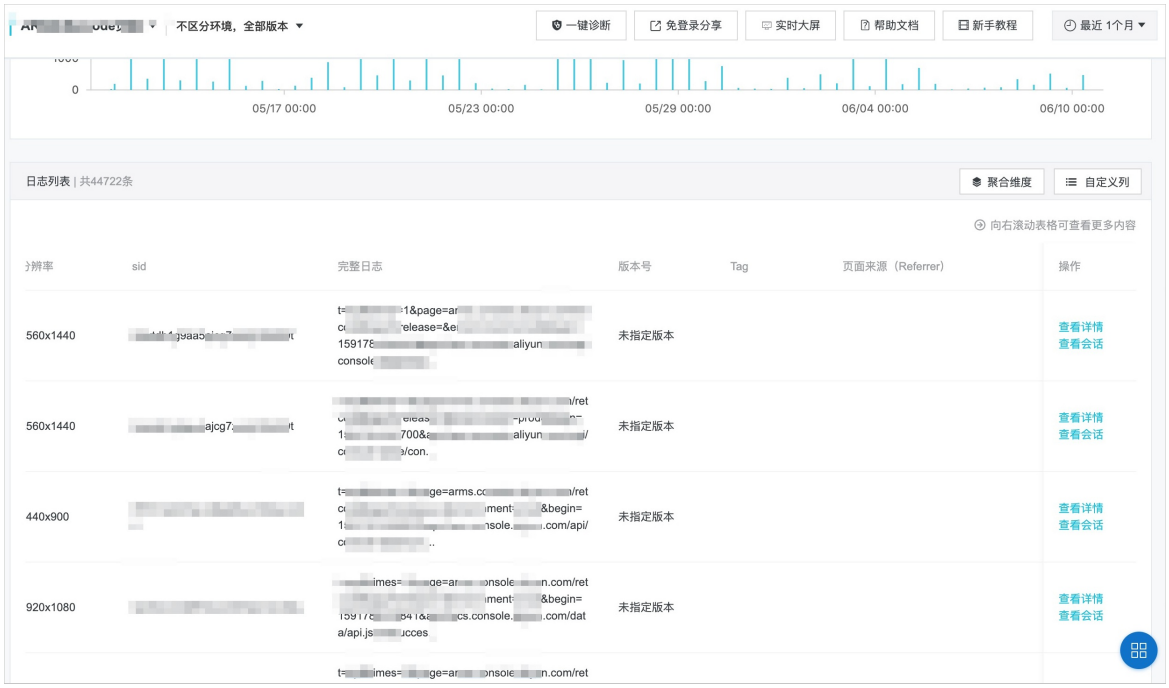
您可以通过配置release参数来区分版本号，更多信息，请参见release。您也可以通过配置environment来区分不同环境，更多信息，请参见environment。

- prod表示线上环境。
- gray表示灰度环境。
- pre表示预发环境。
- daily表示日常环境。
- local表示本地环境。

[回到顶部]

## 如何查看配置的版本号？

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**，然后单击目标应用名称。
3. 在左侧导航栏中选择**应用 > 访问明细**。
   日志的版本号显示在**日志列表**区域的**版本号**列内。




4. 您也可以在**菜单栏**内按照环境和版本筛选日志。只有在PV日志设置完成版本号后，才可以使用按照版本号筛选日志功能。


[回到顶部]

## 如何查看用户的页面停留时间？

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**，然后单击目标应用名称。
3. 在左侧导航栏选择**应用 > 会话追踪**。
4. 在会话列表中单击目标会话的**会话ID**，查看**会话追踪详情**页面。
5. 将光标放置在访问时间轴列内的**时间轴**区域上，即可查看页面停留时间。

[回到顶部]

## 如何查看ARMS的前端自定义性能指标？

1. 登录ARMS控制台。
2. 在左侧导航栏选择**前端监控 > 前端列表**，然后单击目标应用名称。
3. 在左侧导航栏中选择**应用 > 访问速度**。
4. 自定义性能指标展示在**页面访问速度排行**区域中。

[回到顶部]

## ARMS配置未生效怎么处理？

原因可能是浏览器缓存未更新。您可以先在导航栏内选择访问明细后，切换版本号为最新配置的版本号并查看趋势图。如果未配置版本号，可以在SDK中配置release参数，请参见release。发布完成后，请观察最新版本是否符合预期。

[回到顶部]

## 为什么小程序JS Error没有上报？

小程序JS Error没有上报可能是因为在async方法下，信息被小程序底层的try catch捕获，导致错误信息上传失败。您可以尝试手动上报错误信息，请参见API参考。

[回到顶部]

## 可以监听console.error的JS Error吗？

- 在浏览器中，针对满足JS Error格式的错误信息，浏览器会静默上报。
- 在小程序端，您可尝试手动上报错误信息，请参见API参考。

[回到顶部]

## 在Weex环境中，小程序设置的UID为什么没生效？

- 如果未调用setConfig方法，请检查初始化配置时是否配置UID，如果缺失请补全。
- 如果调用了setConfig方法，请在setConfig中重新配置UID。

[回到顶部]

## 日志的保存时效是多长？

- 基础版的日志的保存时效为7天。
- 专家版的日志的保存时效为30天。

[回到顶部]

## 试用期创建的站点可以在开通专家版之后继续使用吗？

- 在试用期到期之后的15天内，站点将因为欠费被停用，您可以尝试重新启动应用。
- 在试用期到期之后的15天后，出于节约计算资源和存储资源的考虑，超出试用时间未开通专业版的站点将被删除，相关的资源也会释放，这部分数据是无法找回的。

[回到顶部]

## ARMS应用版本和宿主版本分别是什么？

- 应用版本是当前线上项目的版本号，可以通过配置SDK的release字段来配置，请参见release。
- 宿主版本是SDK自动获取的，是当前项目所依附的App的版本号，目前因为没有对应宿主App版本的解析方案，只能解析到淘宝、支付宝或微信。

[回到顶部]

## 为什么控制台中不同模块的同一个页面的访问量差别大？

访问速度页面中：访问量=性能日志*Sample

维度页面中：访问量=PV日志

静默上报的性能日志只有在页面Onload之后才上报一次，即每次刷新页面，只会有一次性能日志上报。

开启单页应用模式后，每次切换路由都会有PV日志上报，即单页应用中，性能日志在量级上远远小于PV日志，导致对应的访问量差异大。

[回到顶部]

## 为什么duration会小于connect download?

ARMS资源加载的性能数据是从 `performance.getEntriesByType('resource')` 中获取的，当获取页面资源时间详情有跨域限制时，默认情况下，跨域资源的性能数据中以下字段从 `performance.getEntriesByType('resource')` 中获取的值为0。

```
redirectStart
redirectEnd
domainLookupStart
domainLookupEnd
connectStart
connectEnd
secureConnectionStart
requestStart
responseStart
```

部分时间属性由于计算方式中涉及以上字段，导致不准或者异常，例如：`connect download: responseEnd - responseStart`，由于responseStart的时间戳是0，导致connect download大于duration。

- 对于您自用的CDN资源，设置响应头Timing-Allow-Origin允许获取资源时间，可以解决上述问题。
- 对于第三方资源，建议此时以duration的值为主参考。

[回到顶部]

## 如果调用__bl.performance()方法时不确定SDK是否已加载完成怎么办?

请参见SPA页面上报。

[回到顶部]

# 11.“Script error.”的产生原因和解决办法

“Script error.”是一个常见错误，但由于该错误不提供完整的报错信息（错误堆栈），问题排查往往无从下手。本文提出了该错误的产生原因和解决办法，以及在ARMS中忽略该错误的方法。

## “Script error.”的产生原因

“Script error.”有时也被称为跨域错误。当网站请求并执行一个托管在第三方域名下的脚本时，就可能遇到该错误。最常见的情形是使用CDN托管JS资源。

为了更好地理解，假设以下HTML页面部署在*http://test.com*域名下：

```
<!doctype html>
<html>
<head>
  <title>Test page in http://test.com</title>
</head>
<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);
  }
  foo(); // 调用app.js中定义的foo方法。
  </script>
</body>
</html>
```

假设 `foo` 方法调用了一个未定义的 `bar` 方法：

```
// another-domain.com/app.js
function foo() {
  bar(); // ReferenceError: bar is not a function
}
```

页面运行之后，捕获到的异常信息如下：

```
"Script error.", "", 0, 0, undefined
```

其实这并不是一个JavaScript Bug。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。

```
bool ScriptExecutionContext::sanitizeScriptError(String& errorMessage, int& lineNumber, Str
ing& sourceURL)
    {
        KURL targetURL = completeURL(sourceURL);
        if (securityOrigin()->canRequest(targetURL))
            return false;
        errorMessage = "Script error.";
        sourceURL = String();
        lineNumber = 0;
        return true;
    }
```

了解了 “Script error.”的产生原因之后，接下来看看如何解决这个问题。

## 解法1：开启跨域资源共享CORS（Cross Origin Resource Sharing）

为了跨域捕获JavaScript异常，可执行以下两个步骤：

1. 添加 `crossorigin="anonymous"` 属性。

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

   此步骤的作用是告知浏览器以匿名方式获取目标脚本。这意味着请求脚本时不会向服务端发送潜在的用户身份信息（例如Cookies、HTTP证书等）。

2. 添加跨域HTTP响应头。

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

   或者

   ```
   Access-Control-Allow-Origin: http://test.com
   ```

   > **说明** 大部分主流CDN默认添加了 `Access-Control-Allow-Origin` 属性。以下是阿里CDN的示例：

   ```
   $ curl --head https://retcode.alicdn.com/retcode/bl.js | grep -i "access-control-allow-
   origin"
   => access-control-allow-origin: *
   ```

完成上述两步之后，即可通过 `window.onerror` 捕获跨域脚本的报错信息。回到之前的案例，页面重新运行后，捕获到的结果是：

```
=> "ReferenceError: bar is not defined", "http://another-domain.com/app.js", 2, 1, [Object
Error]
```

## （可选）解法2：`try catch`

难以在HTTP请求响应头中添加跨域属性时，还可以考虑 `try catch` 这个备选方案。

在之前的示例HTML页面中加入 `try catch` ：

```
<!doctype html>
<html>
<head>
  <title>Test page in http://test.com</title>
</head>
<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;
  }
  </script>
</body>
</html>
```

再次运行，输出结果如下：

```
=> ReferenceError: bar is not defined
     at foo (http://another-domain.com/app.js:2:3)
     at http://test.com/:15:3
=> "Script error.", "", 0, 0, undefined
```

可见 `try catch` 中的Console语句输出了完整的信息，但 `window.onerror` 中只能捕获“Script error”。根据这个特点，可以在catch语句中手动上报捕获的异常，更多信息，请参见API参考。

```
__bl.error(error, pos);
```

> **说明** 尽管 `try catch` 方法可以捕获部分异常，但推荐采用解法1。

## 如何在ARMS中忽略“Script error.”？

“Script error.”往往来源于第三域名下的脚本，如果不影响应用的运行，您可以选择通过ARMS前端监控的SDK参数ignore忽略该错误。

`ignore` 可忽略指定JS错误的上报，其配置项的值是一个对象，包含3个属性：ignoreUrls、ignoreApis和ignoreErrors。默认值如下：

```
ignore: {
    ignoreUrls: [],
    ignoreApis: [],
    ignoreErrors: []
},
```

针对“Script error.”，使用ignoreErrors即可。ignoreErrors表示忽略某些JS错误，符合该规则的JS错误不会上报。值可以是*String*、*RegExp*、*Function*或者以上三种类型组成的数组。示例：

```
__bl.setConfig({
    ignore: {
        ignoreErrors: /^Script error\.?$/
    }
});
```

## 相关链接

- API参考
- SDK参考