阿里云
应用实时监控服务

前端监控

文档版本：20190916

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。

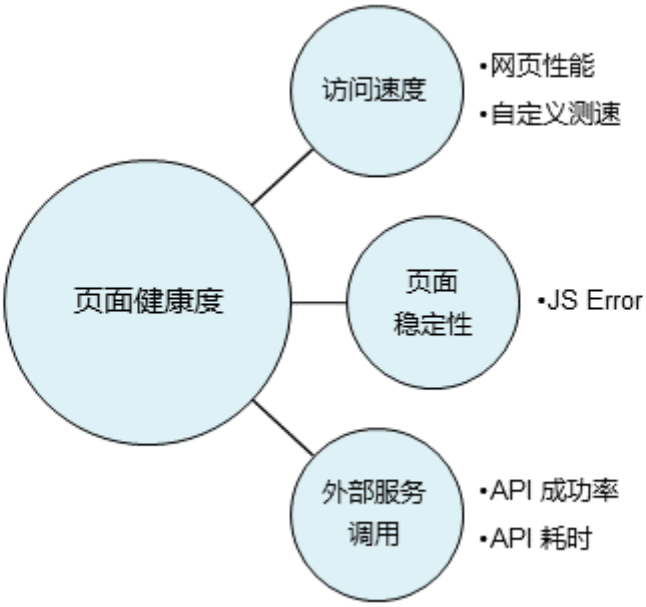
通用约定

格式	说明	样例
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	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:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
##	表示参数、变量。	`bae log list --instanceid` *`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。	`swich` *`{stand | slave}`*

目录

1 前端监控概述

ARMS 前端监控平台专注于 Web 端体验数据监控，从页面打开速度（测速）、页面稳定性（JS Error）和外部服务调用成功率（API）这三个方面监测 Web 页面的健康度。



同时，基于应用性能指标算法（APDEX），ARMS 对应用站点及页面进行了满意度评分，可以很直观地了解用户对站点或页面的体感。

为什么要有前端监控?

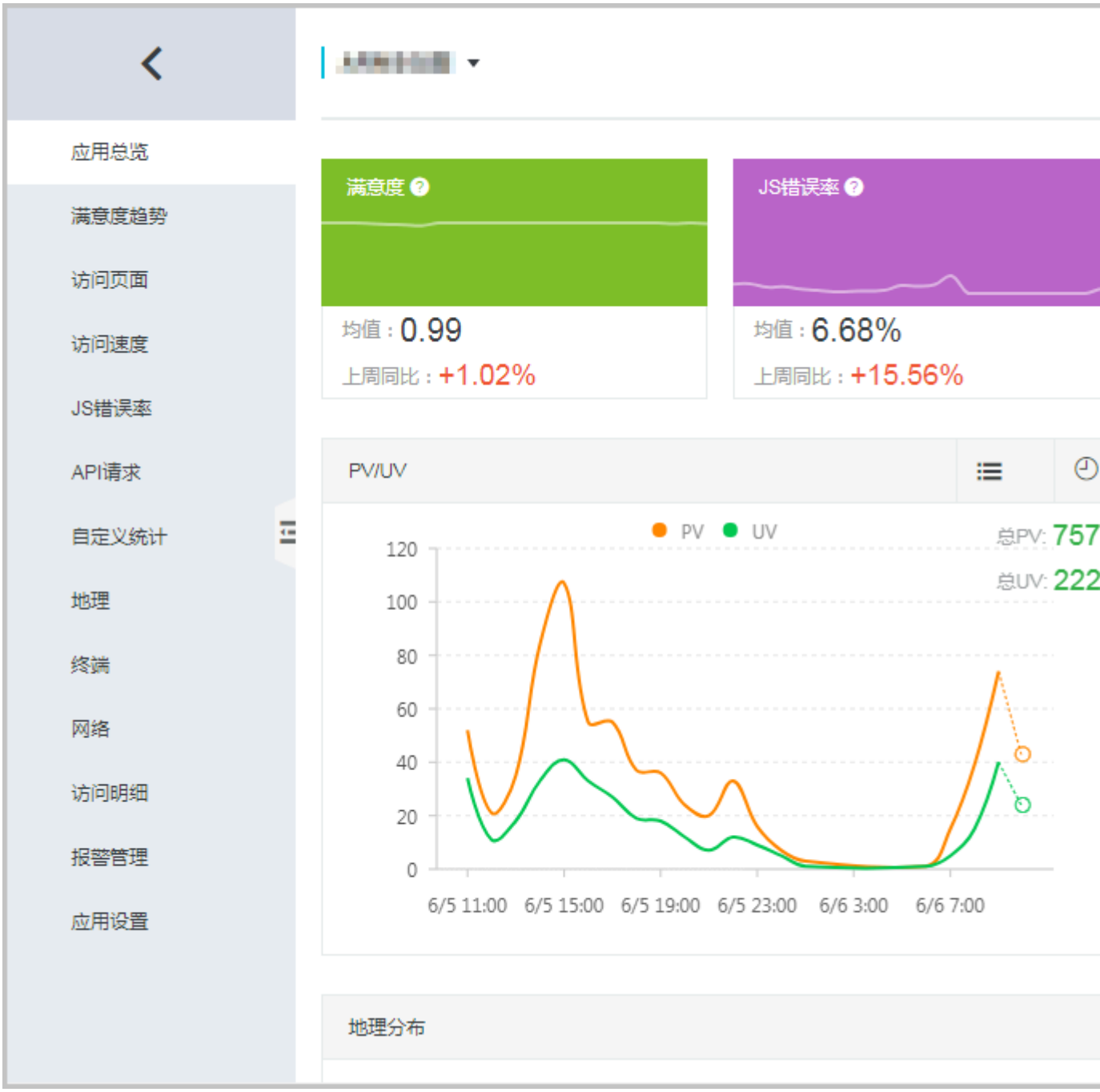
用户访问我们的业务时，整个访问过程大致可以分为三个阶段：页面生产时（Server 端状态）、页面加载时和页面运行时。

为了保证线上业务稳定运行，我们会在 Server 端对业务的运行状态进行各种监控。现有的 Server 端监控系统相对已经很成熟了，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我们的站点时遇到的错误；
- 各个国家、各个地区的用户访问我们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 前端监控平台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。之后借助阿里云中间件平台 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。



浏览器/平台兼容性

浏览器/平台	支持版本	自动上报	主动上报
Safari	Safari 9+	##	##
Chrome	Chrome 49+	##	##
IE	IE 9+	##	##
Edge	Edge 12+	##	##
Firefox	Firefox 36+	##	##
Opera	Opera 43+	##	##

浏览器/平台	支持版本	自动上报	主动上报
Safari for iOS	Safari for iOS 9.2+	##	##
Android Browser	android_webkit 4.4.2+	##	##
Weex	Weex 0.16.0 +		##

2 开始使用前端监控

2.1 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

Weex 场景

小程序场景

2.2 Web 场景

2.2.1 以 npm 方式安装探针

要使用 ARMS 前端监控监控 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
```

使用

初始化

SDK 以 `BrowerLogger.singleton` 方式初始化。

```
const BrowerLogger = require('alife-logger');
// BrowserLogger.singleton(conf) conf传入config配置
const __bl = BrowerLogger.singleton({
    pid: 'your-project-id',
    imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?', // 设定日志上传地址,新加坡部署可选`https://arms-retcode-sg.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，若您想自定义 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
```

例如：

```
const BrowserLogger = require('alife-logger');
// BrowserLogger.singleton(conf) conf传入config配置
const __bl = BrowserLogger.singleton({
    pid: 'your-project-id',
    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?', // 设定日志上传地址,新加坡部署可选`https://arms-retcode-sg.aliyuncs.com/r.png?`
    // 其他config配置
});
```

API 说明

@static singleton() 获取单例对象

 说明：

该方法只适用于 npm 引入。

调用参数说明：`BrowerLogger.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	是	-
prePipe	Array	预上报内容	否	-

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。

其他上报 API

通过 `BrowerLogger.singleton` 获取实例。

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

关于 `__bl` 的其他 API 使用方式，请参考 #unique_8。

Config 配置

Config 配置与 CDN 引入配置相同。请参考#unique_9。

预上报

场景：在调用 `BrowserLogger.singleton()` 之前执行的部分逻辑需要上报一些数据。

```
const BrowerLogger = 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'],
    // 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);
```

2.2.2 以 CDN 方式安装探针

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

操作步骤

1. **ARMS 控制台，在左侧导航栏中选择应用监控 > 应用列表。**
2. **在左侧导航栏中选择前端监控，在前端监控页面右上角单击新建应用站点。**

3. 在新建应用站点对话框中，选择站点类型 Web，输入站点名称，并单击确定。

4. 在应用的设置页面的 SDK 扩展配置项区域勾选需要的选项。

   - 关闭 API 自动上报：勾选此项后，需手动调用 `__bl.api()` 方法上报 API 成功率。
   - 开启 SPA 自动解析：勾选此项后，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：勾选此项后，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：勾选此项后，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：勾选此项后，API 请求会与后端应用监控进行端到端关联。

5. 在复制/粘贴BI探针区域，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。

使用 CDN 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，若您想自定义 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("crossorigin","",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。

后续步骤

ARMS 前端监控还提供了更多 SDK 配置项，可满足进一步的需求，请参考#unique_9。

#unique_11
#unique_9
#unique_12
#unique_13

2.3 Weex 场景

2.3.1 Weex 接入配置

本文介绍了 weex 环境中云前端监控的接入配置。

导入 npm 包

在 weex 环境中，使用专门的 WeexLogger 模块来上报日志，接入时需要在项目中导入 `alife-logger` npm 包。

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

初始化

在 weex 应用入口调用 `singleton(props)` 静态方法获取实例，需要在传入的 `props` 中设定相关配置，详情参见通用 API。

```
// in app.js
import WeexLogger from 'alife-logger/weex';

const wxLogger = WeexLogger.singleton({
    pid: 'your-project-id',
    uid: 'zhangsan', // Login uid, for UV report
    page: 'Lazada | Home', // Initial page name, if passed, SDK will
send a PV log after Initialization completed
    imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?' // 设定日志上传地
址,新加坡部署可选`https://arms-retcode-sg.aliyuncs.com/r.png?`
});
```

上报

通过实例调用相应的上报方法进行上报。

```
// in some biz module
import WeexLogger from 'alife-logger/weex';

wxLogger.api('/search.do', true, 233, 'SUCCESS');
```

通用 API

@static singleton() 获取单例对象

静态方法，返回一个单例对象。`props` 用法如下（只在第一次调用时生效）。

调用参数说明：`WeexLogger.singleton(props)`

属性	类型	描述	是否必须	默认值
pid	String	站点 ID	是	-
page	String	初始化的 Page Name	否	-
uid	String	用户 ID	是	-

属性	类型	描述	是否必须	默认值
imgUrl	String	日志上传地址，以“?”结尾	否	-

此方法可用于在应用入口初始化 SDK，示例参见初始化。

setPage() 设置当前页面的 Page Name

设置 Page Name，并且上报一次 PV 日志（默认）。

调用参数说明：

```
const wxLogger = WeexLogger.singleton();
// ...
wxLogger.setPage(nextPage);
```

参数	类型	描述	是否必须	默认值
nextPage	String	Page Name	是	-

setConfig() 修改配置项

用于在 SDK 初始化完成后修改部分配置项，具体配置同 `singleton()` 方法。

调用参数说明：

```
const wxLogger = WeexLogger.singleton();
// ...
wxLogger.setConfig(next);
```

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

日志上报 API

请参见 API 使用指南的“日志上报接口”。

2.4 小程序场景

2.4.1 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。

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。

   a. 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
   ```

   b. 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，请参考通用参数。

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

   a. 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() {
     }
   ```

```
}));
```

b. 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。

通用参数

通用的初始化参数如下表所示。

参数	类型	描述	是否必需	默认值
pid	String	站点 ID。	是	null
uid	String	用户 ID，用于统计 UV。	否	Storage 设置
tag	String	传入的标记。每条日志都会携带该标记。	否	无
disabled	Boolean	是否禁用日志上报功能。	否	false
sample	Integer	日志采样率，值为 1、10 或 100。性能和成功 API 日志按照 1/sample 的比例上报。	否	1
enableLinkTrace	Boolean	是否支持前后端链路追踪。	否	false

参数	类型	描述	是否必需	默认值
disableHook	Boolean	是否禁用 dd.httpRequest 请求监听。默认会监听并用于上报 API 调用成功率。	否	false
sendRequest	Function	发送日志的方法。如果不配置，则采用默认值 dd.httpRequest。	否	dd.httpRequest
getCurrentPage	Function	获取当前页面的方法。	否	getCurrentPage

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。使用方法示例请参考操作步骤。

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。
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	String	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，请参考以下进阶场景。

- 手动上报 API 相关信息（不采用静默上报方式）
  1. 将 disableHook 设为 true，不静默上报 dd.httpRequest 请求的日志。
  2. 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
  1. 在 App 和 Page 对应的 JS 文件中不再使用 hookApp、hookPage 方法。
  2. 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();
         }
     })
     ```

  3. 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();
           }
           ...
       })
     ```

更多信息

- #unique_18
- #unique_19
- #unique_20
- #unique_21

2.4.2 接入支付宝小程序

本文介绍将支付宝小程序接入 ARMS 前端监控的基础使用方法、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。

   a. 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
   ```

   b. 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，请参考通用配置。

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

   a. 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() {
     }
   }));
   ```

b. 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。

通用配置

通用的初始化参数如下表所示。

参数	类型	描述	是否必需	默认值
pid	String	站点 ID。	是	null
uid	String	用户 ID，用于统计 UV。	否	Storage 设置
tag	String	传入的标记。每条日志都会携带该标记。	否	无
disabled	Boolean	是否禁用日志上报功能。	否	false

参数	类型	描述	是否必需	默认值
sample	Integer	日志采样率，值为1、10或100。性能和成功API日志按照1/sample的比例上报。	否	1
enableLinkTrace	Boolean	是否支持前后端链路追踪。	否	false
disableHook	Boolean	是否禁用my.httpRequest请求监听。默认会监听并用于上报API调用成功率。	否	false
sendRequest	Function	发送日志的方法。如果不配置，则采用默认值my.httpRequest。	否	my.httpRequest
getCurrentPage	Function	获取当前页面的方法。	否	getCurrentPage

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。使用方法示例请参考基础使用方法。

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。

方法	参数	备注
setConfig	{[key: string]: string;}	设置 config 字段。
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	String	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，请参考以下进阶场景。

- 手动上报 API 相关信息（不采用静默上报方式）
  1. 将 disableHook 设为 `true`，不静默上报 my.httpRequest 请求的日志。
  2. 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
  1. 在 App 和 Page 对应的 JS 文件中不再使用 hookApp、hookPage 方法。
  2. 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();
         }
     })
     ```

  3. 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();
    }
    ...
})
```

参考

- #unique_11
- #unique_23
- #unique_20
- #unique_21

2.4.3 接入微信小程序

本文介绍将微信小程序接入 ARMS 前端监控的基础使用方法、通用配置、API 方法和进阶场景。

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。
   a. 将 https://retcode.alicdn.com/retcode/wl.js 的内容复制并粘贴至微信小程序 /utils 目录下的 wxLogger.js 文件中。
   b. 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。

   - 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'
       });
       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'
       });
       export default Monitor;
   ```

说明:

关于参数的详细配置，请参考通用配置。

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

   a. 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() {
  }
}));
```

   b. 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。

通用配置

通用的初始化参数如下表所示。

参数	类型	描述	是否必需	默认值
pid	String	站点 ID。	是	null
uid	String	用户 ID，用于统计 UV。	否	Storage 设置
tag	String	传入的标记。每条日志都会携带该标记。	否	无
disabled	Boolean	是否禁用日志上报功能。	否	false
sample	Integer	日志采样率，值为 1、10 或 100。性能和成功 API 日志按照 1/sample 的比例上报。	否	1
enableLinkTrace	Boolean	是否支持前后端链路追踪。	否	false
disableHook	Boolean	是否禁用 request 请求监听。默认会监听并用于上报 API 调用成功率。	否	false
sendRequest	Function	发送日志的方法。如果不配置，则采用默认值 wx.request。	否	wx.request
getCurrentPage	Function	获取当前页面的方法。	否	getCurrentPage

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。使用方法示例请参考基础使用方法。

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。
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	String	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，请参考以下进阶场景。

- 手动上报 API 相关信息（不采用静默上报方式）

  1. 将 disableHook 设为 `true`，不静默上报 wx.request 请求的日志。
  2. 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

  1. 在 App 和 Page 对应的 JS 文件中不再使用 hookApp、hookPage 方法。
  2. 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();
         }
     })
     ```

  3. 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();
           }
           ...
       })
     ```

参考

- #unique_11
- #unique_23
- #unique_19
- #unique_21

2.4.4 接入其他类别小程序

本文介绍将其他类别的小程序接入 ARMS 前端监控的基础使用方法、通用配置、API 方法和进阶场景。本文适用于除钉钉 E 应用、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。

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。

   a. 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
   ```

   b. 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，请参考通用配置。

2. 上报日志。

   a. 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() {
      }
    });
```

b. 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，请参考 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。

通用配置

通用的初始化参数如下表所示。

参数	类型	描述	是否必需	默认值
pid	String	站点 ID。	是	null
uid	String	用户 ID，用于统计 UV。	否	Storage 设置
tag	String	传入的标记。每条日志都会携带该标记。	否	无
disabled	Boolean	是否禁用日志上报功能。	否	false
sample	Integer	日志采样率，值为 1、10 或 100。性能和成功 API 日志按照 1/sample 的比例上报。	否	1
disableHook	Boolean	是否禁用 my.httpRequest 请求监听。默认会监听并用于上报 API 调用成功率。	否	false
sendRequest	Function	发送日志的方法。如果不配置，则无法发送日志。	是	无

参数	类型	描述	是否必需	默认值
getCurrentPage	Function	获取当前页面的方法。	是	无

`sendRequest` 参数说明

此参数用于发送日志，需要支持 Get/Post 方法。上报 Error 类的日志时会采用 Post 上报方式。方法的参数说明如下：

参数	类型	描述
url	String	日志上报的地址。
resData	Object	需要 Post 上报的内容。如果该参数有值则需要用 Post 方法上报，否则用 Get 方法上报。

`sendRequest` 配置示例

```
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) {
                //...
            }
        });
    }
```

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。

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。
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	String	API 类日志上报。
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。使用方法示例请参考基础使用方法。

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，请参考以下进阶场景。

- 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。

参考

- #unique_11
- #unique_23
- #unique_19
- #unique_20

3 控制台功能

3.1 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：

- 页面加载详情
- 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

左侧的页面访问速度排行，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。

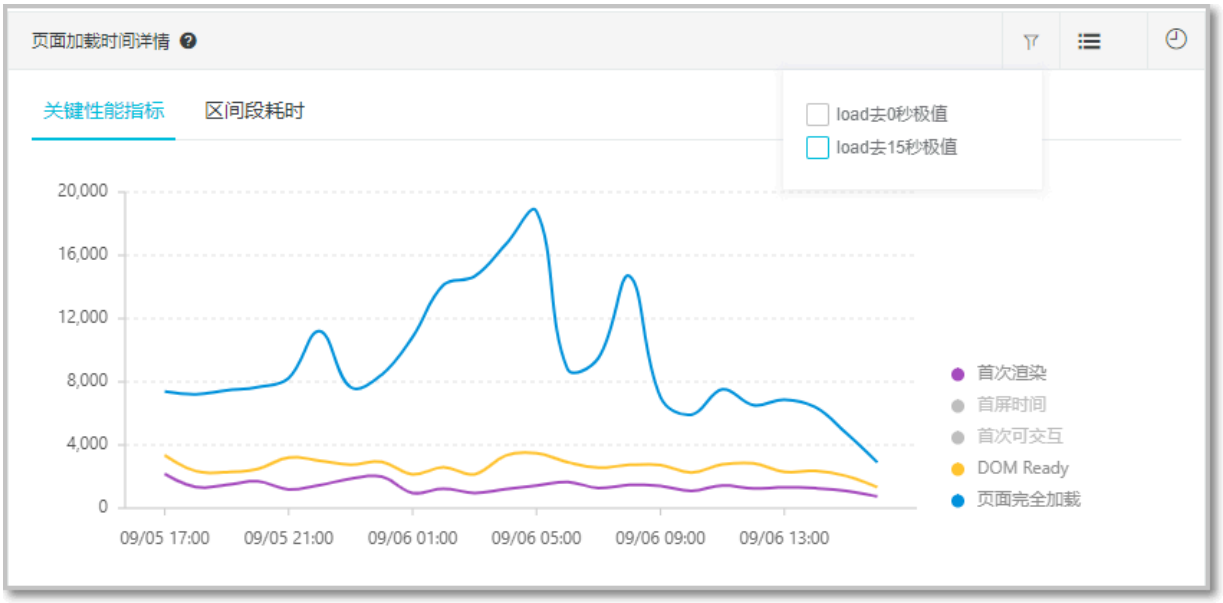
页面加载详情

此模块展示以下指标：

- 关键性能指标：
  - 首次渲染
  - 首屏时间
  - 首次可交互
  - DOM Ready
  - 页面完全加载时间

- 区间段耗时：
  - DNS 查询
  - TCP 连接
  - 请求响应
  - 内容传输
  - DOM 解析
  - 资源加载

关于这些指标的计算方式，请参考#unique_28。

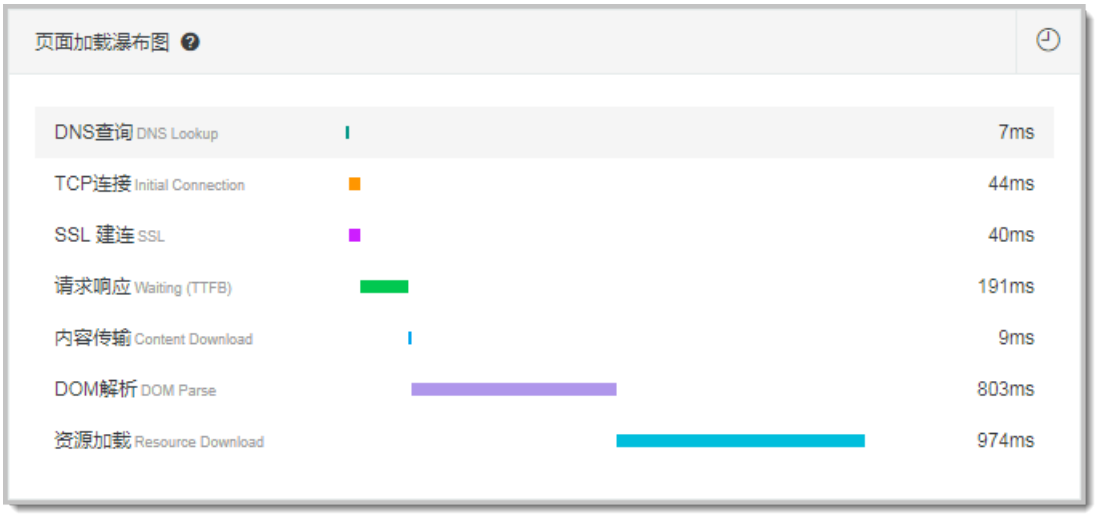


说明：

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，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。

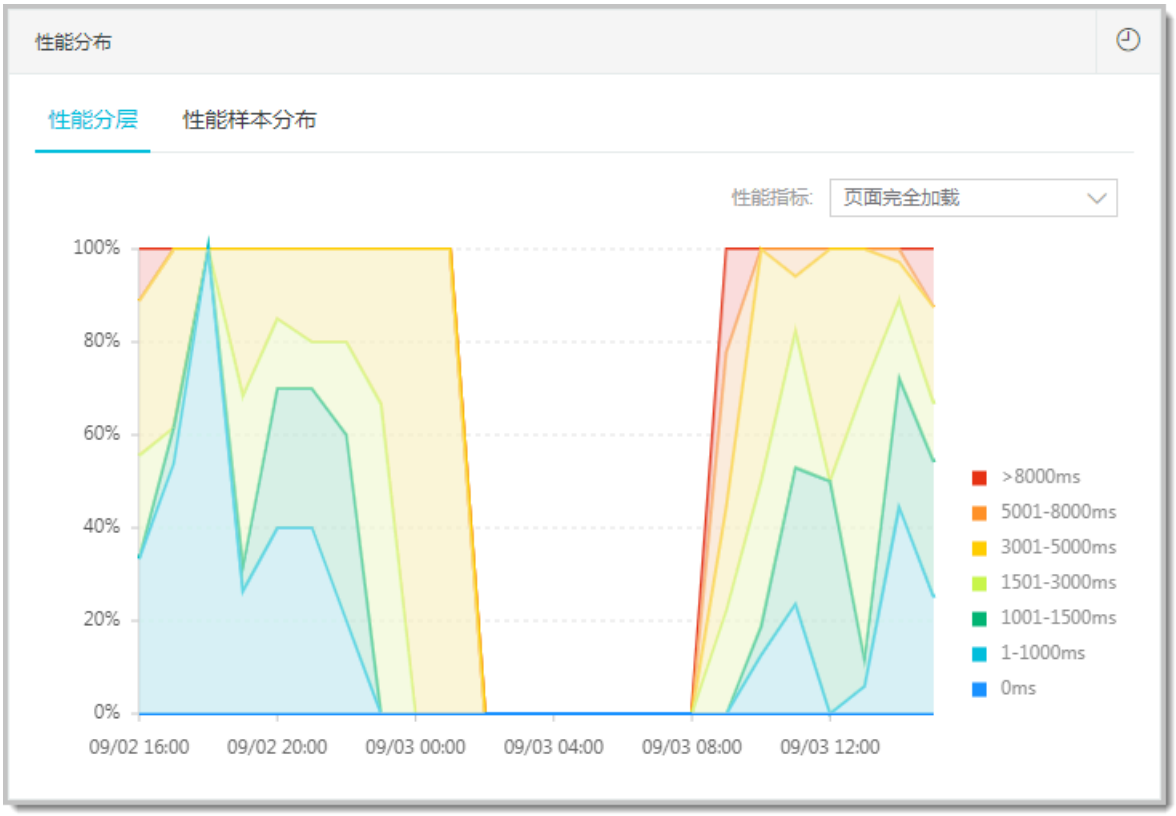


性能分布

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。

图 3-1: 性能分层



在性能样本分布上，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。

图 3-2: 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。详情请参考#unique_29。

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，所以我们提供地理分布、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。

图 3-3: 地理分布

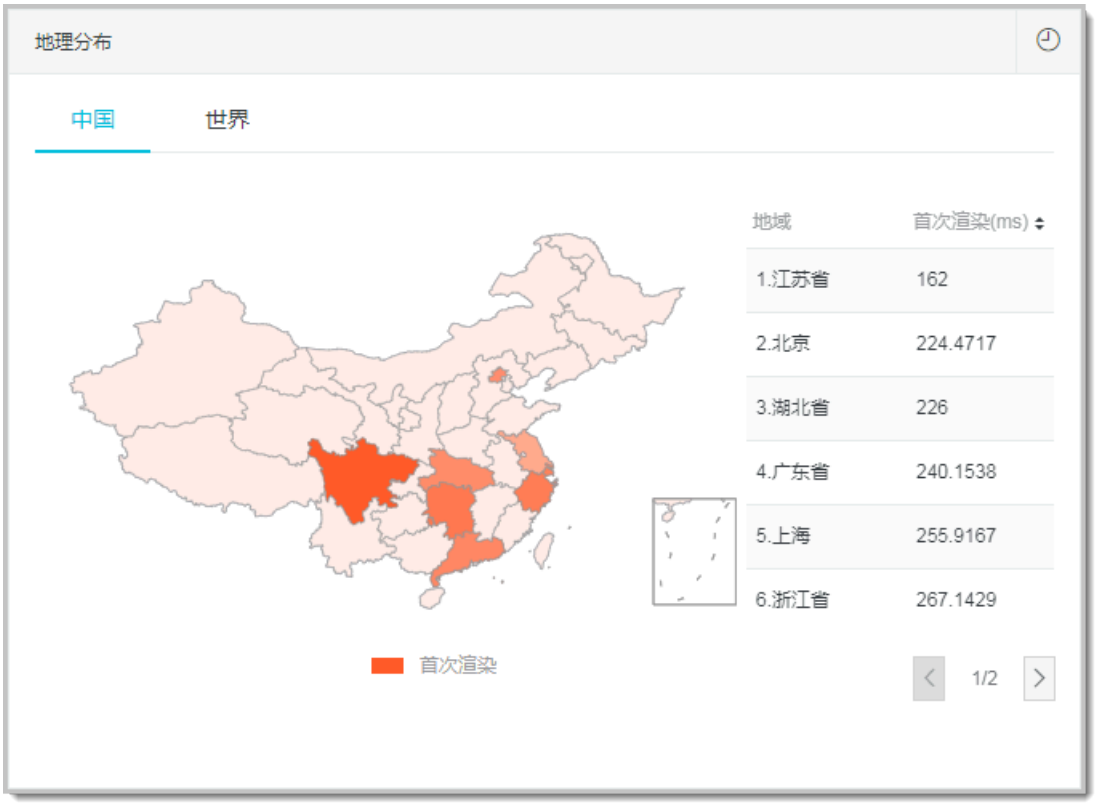
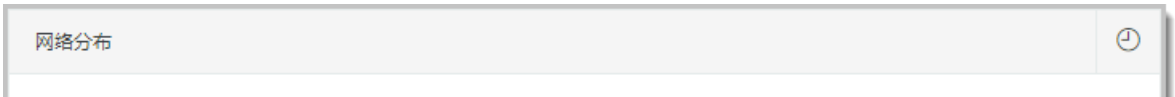


图 3-4: 终端分布

图 3-5: 网络分布



3.2 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

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`。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。

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("crossorigin","",src=d)
})(window,document,"https://retcode.alicdn.com/retcode/bl.js","__bl");
</script>
```

说明:

静态资源加载信息的上报是在页面 onload 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，建议不开启该配置。

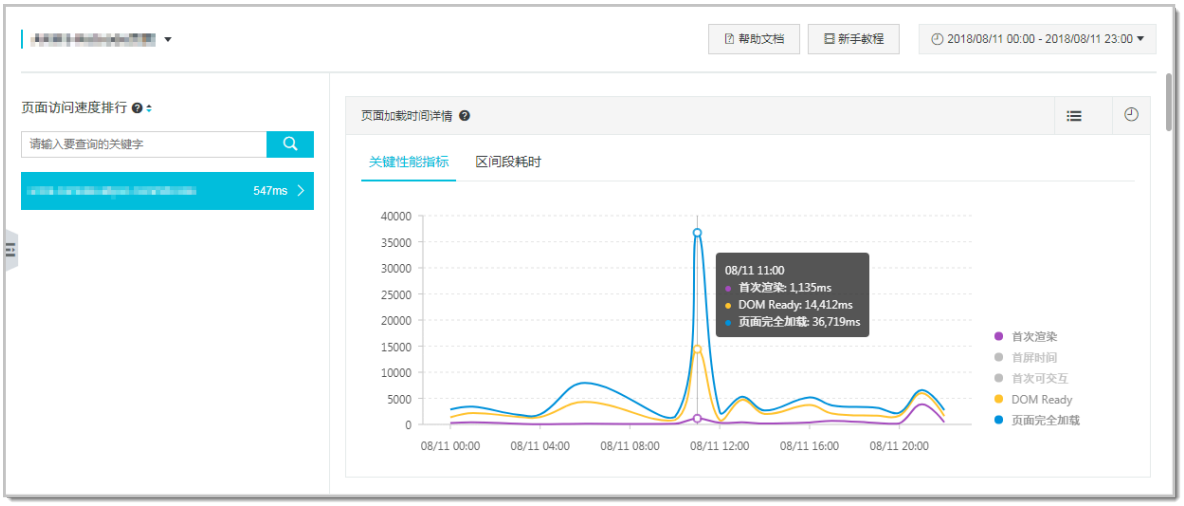
功能入口

1. 登录 ARMS 控制台。
2. 在左侧导航栏中单击前端监控，在前端监控页面上单击应用名称。
3. 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

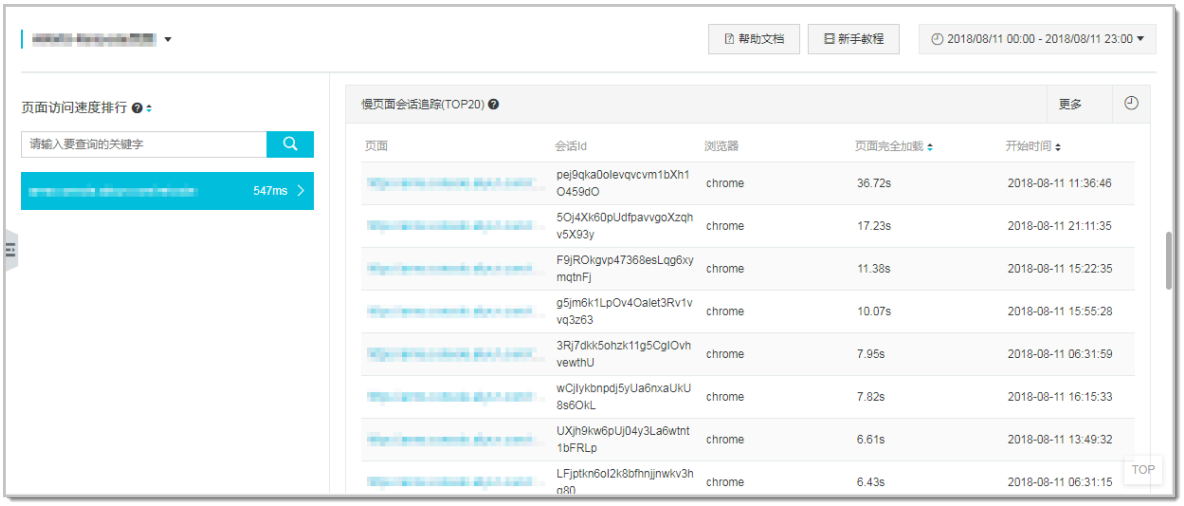
接下来以一个示例介绍如何定位页面性能瓶颈。

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。

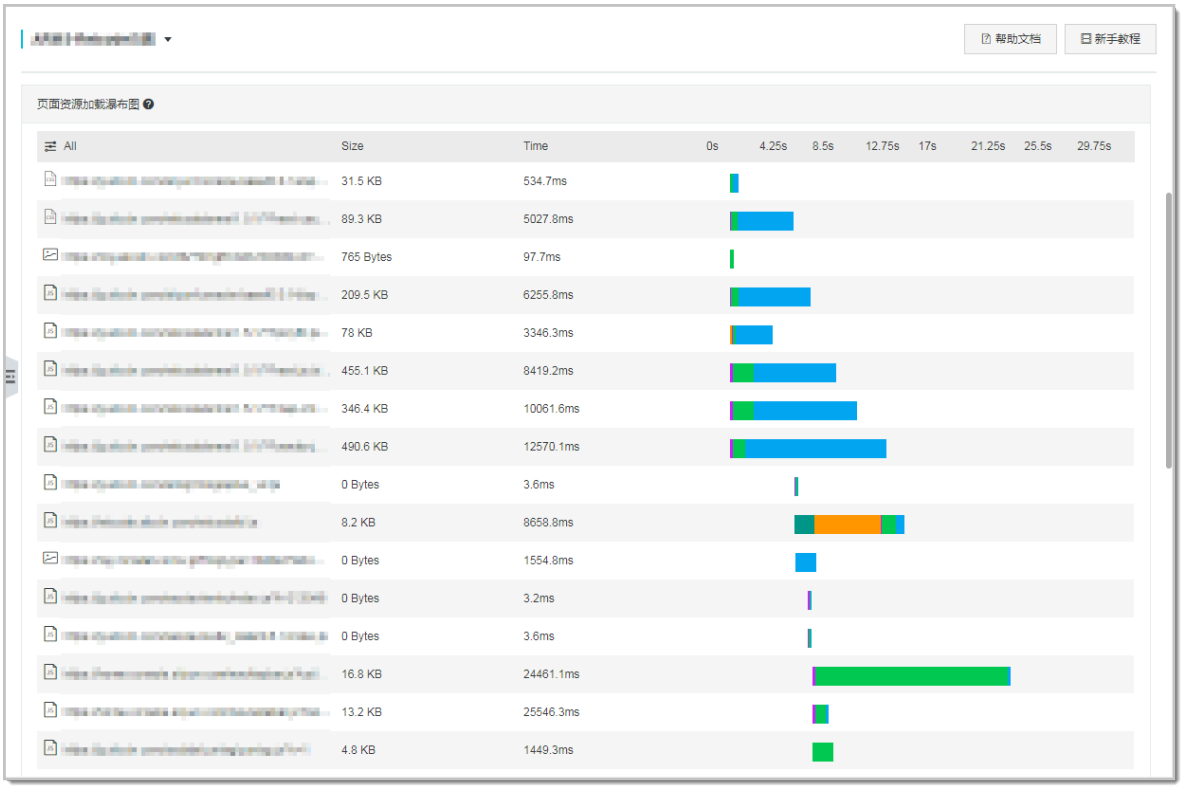


2. 在访问速度页面上，使用滚动条向下滚动至慢页面会话追踪（TOP20）区域框。该区域框会列出该页面在指定时间段内加载最慢的 20 个会话。

   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直接原因。



3. 在慢页面会话追踪（TOP20）区域框的页面列中单击页面名称，即可进入会话追踪 > 会话详情页面。借助该页面的页面资源加载瀑布图，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. 在会话追踪 > 会话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。

在左侧导航栏中单击会话追踪，即可查看该应用下的会话列表。您也可以按照页面地址、会话 ID、浏览器、浏览器版本号等条件筛选会话。

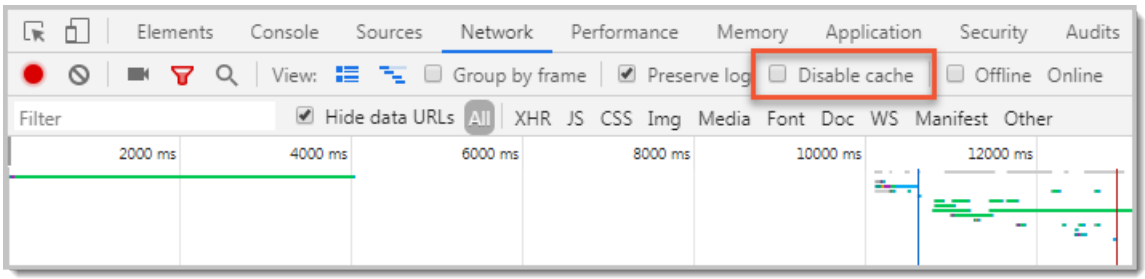
单击操作列中的详情，即可打开会话详情页面，查看该会话的基本页面信息、页面资源加载瀑布图和 API 加载瀑布图。

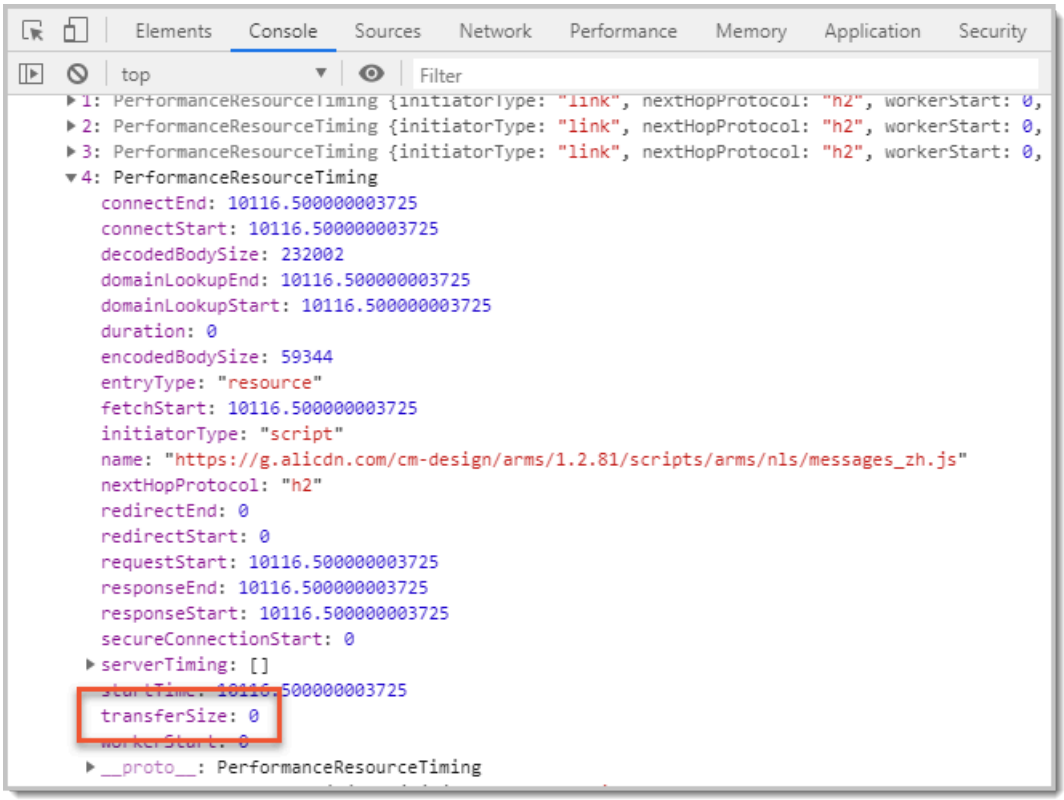
常见问题

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。

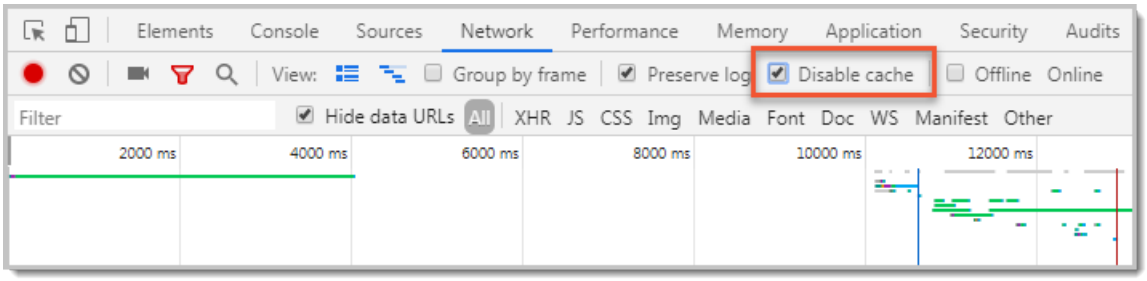
在 Chrome 浏览器中按 F12 打开开发者工具面板，当 Network 页签上的 Disable cache 未勾选时，transferSize 为 0。

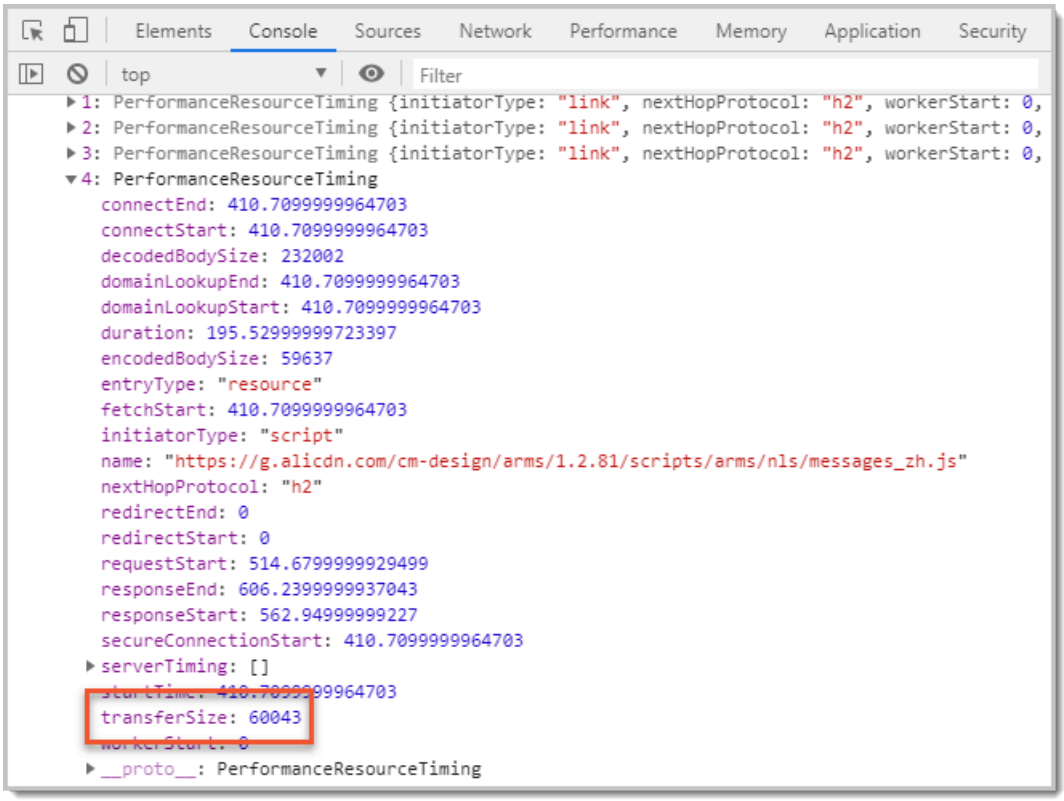




解决方法

勾选 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，取消勾选 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。

   请放心，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

- #unique_31

- #unique_32

3.3 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。

快速入门

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，单击顶部的实时大屏。
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。
  - 高错误率 Top 5：JavaScript 错误率最高的 5 个服务。
- 今日告警信息
  - 告警数量：告警的数量。
  - 最近告警信息：今日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 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。
  - 低成功率 Top 5：API 请求成功率最低的 5 个服务。
- 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。

更多信息

- #unique_31
- #unique_34
- #unique_35
- #unique_36

3.4 JS 错误统计

本文介绍了 ARMS 前端监控的 JS 错误统计功能。

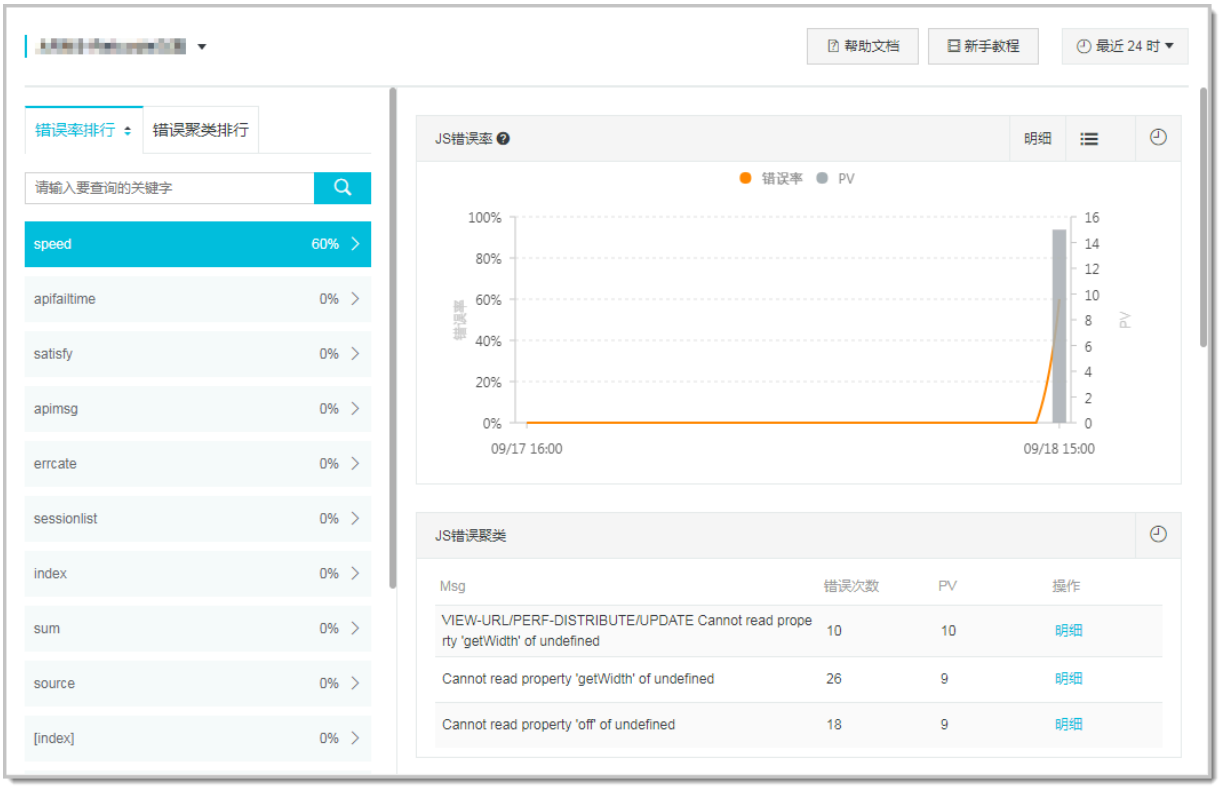
JS 错误率的定义是什么?

在 ARMS 前端监控中，JS 错误率的计算公式为：

```
JS 错误率 = 指定时间内发生 JS 错误的 PV / 总 PV
```

错误率排行

在左侧的错误率排行标签页上，列出的是站点内错误率最高或最低的前 100 个页面，可以按照错误率升序或降序排列。右侧的 JS 错误率图表展示的是左侧列表选中页面在指定时间范围内的 JS 错误率曲线和 PV。



说明:

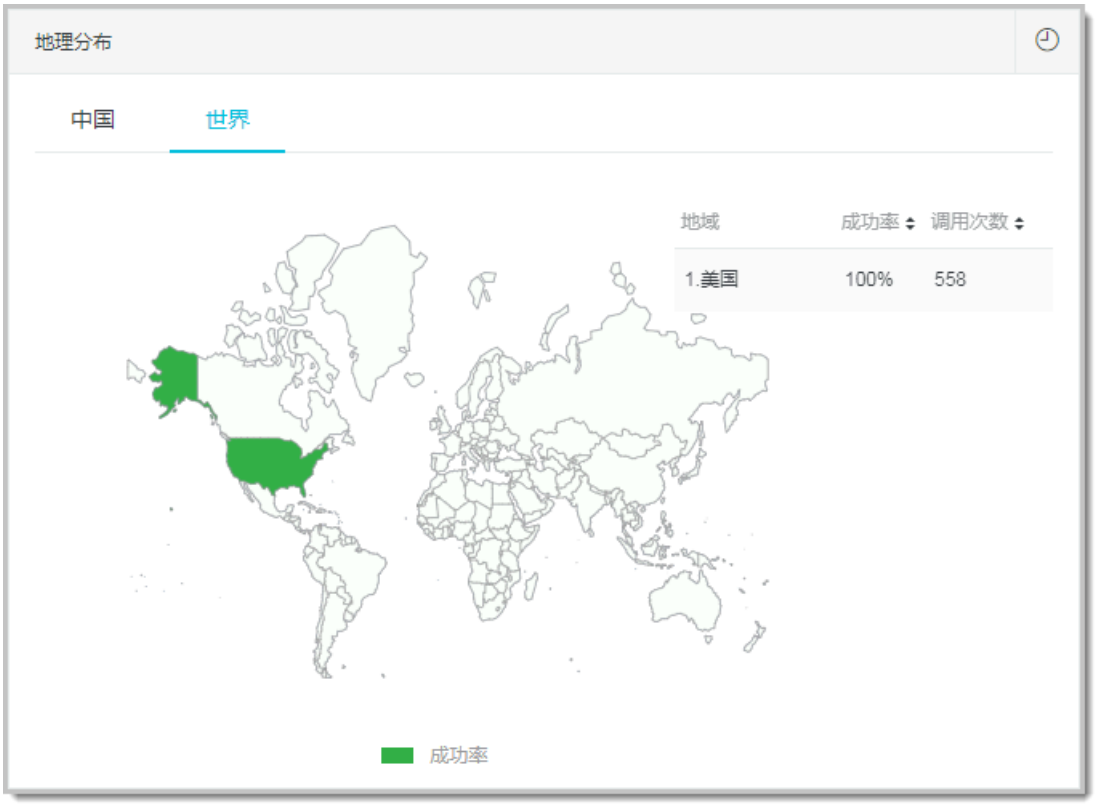
由于错误率排行榜仅会展示错误率最高或最低的前 100 个页面，当站点的页面总数超过 200 个时，错误率不属于这两个区间的页面始终不会显示在排行中。例如，假设站点共有 220 个页面，那么无论选择按错误率升序还是降序排列，都会有 20 个页面不会显示在排行中。

错误聚类排行

在左侧的错误聚类排行标签页上，列出的是每种错误信息的发生次数排行。右侧的 JS 错误调用页面展示的是出现左侧列表选中错误的页面，按错误次数降序排列。

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。

通用操作

在 JS 错误统计模块中，您可以执行以下通用操作。

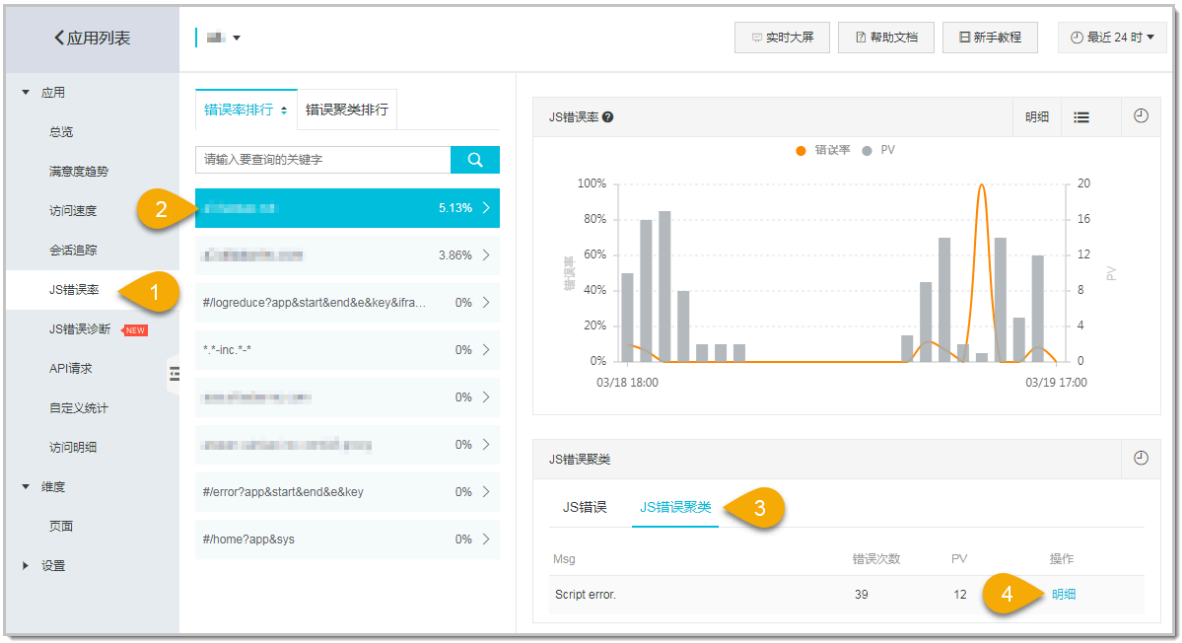
- 在左侧标签页上，单击标签上的上箭头或下箭头来改变列表的排列顺序。上箭头表示升序，下箭头表示降序。
- 在右侧的详情显示区域中，单击右上角的 和 图标，可在图表和表格视图间切换。
- 在右侧的详情显示区域中，单击右上角的 图标，可指定时间范围。

如何进行 JS 错误排查?

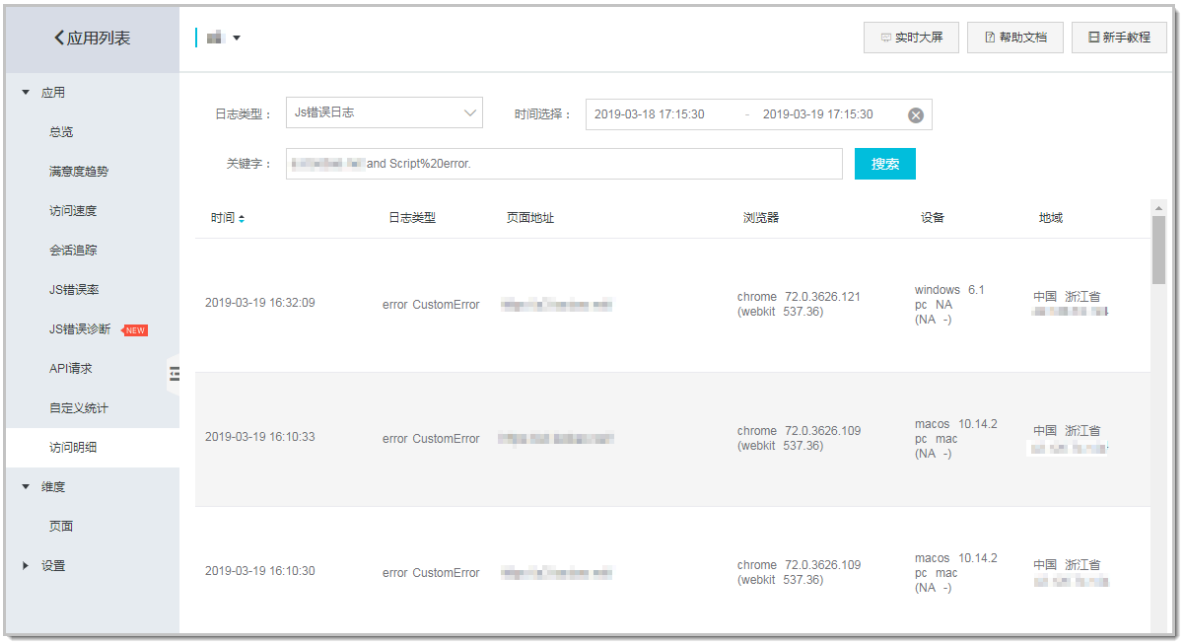
您可通过以下操作进入 JS 错误详情页面。

1. 登录 ARMS 控制台。
2. 在左侧导航栏中单击前端监控，在前端监控页面上单击应用名称。
3. 在应用列表的左侧导航栏中选择应用 > JS 错误率。

4. 在 JS 错误率标签页选择需排查的对象，并单击 JS 错误类聚区域的明细。



JS 错误详情页面打开。



说明:

若信息过长显示不完整，将鼠标移到错误信息外即可显示全部信息。

在 JS 错误详情页可查看错误明细，根据每条错误上的关键信息，即可定位到 JS 错误所在的文件和信息，帮助前端工程师快速定位问题。

每条错误可显示的关键信息：

- 上报时间
- 日志类型
- 页面地址
- 浏览器
- 设备
- 地域
- Tag
- UA（User Agent）
- Param 参数
- Message（信息）
- Stack（错误栈信息）
- File（错误文件）
- Line/Col（错误位置）

JS 错误详情页还提供了错误搜索功能，搜索条件包括：

- 日志类型：默认是 JS 错误日志
- 时间选择：错误产生的时间

> 说明:
> 为提高搜索效率，时间跨度不宜太短。

- 关键字：根据 Message 的关键词进行搜索

> 说明:
> 目前只支持全 Message 匹配搜索，暂不支持模糊搜索。

如何设置 JS 错误率报警?

您可以设置针对 JS 错误率的报警。在以下示例中，触发报警的条件是最近 10 分钟的错误率平均值超过 20%，且最近 10 分钟的错误总数超过 20 个。

1. 在应用列表的左侧导航栏中选择设置 > 报警管理。
2. 在设置报警规则区域单击新建报警。
3. 在新建报警对话框中，按照下图输入各项参数。

如何上报资源加载失败的情况（例如 404）？

SDK 监控的 JS 错误仅限脚本相关错误，不包括资源加载错误（例如 404）。为防止阻塞业务代码，SDK 会延后加载，因此一般无法捕捉页面加载失败的错误。

如果仍需监控资源失败情况并上报数据，请遵循以下步骤。

1. 在页面 Head 最前面监听 addEventListener onerror 。

```
window.addEventListener("error", function (e) {
var elem = e.target;
if(/img|script/.test(elem.tagName.toLowerCase())){
        window.__sourceError = window.sourceError || [];
        window.__sourceError.push(e);
    }
}, true); /*指定事件处理函数在捕获阶段执行*/
```

说明:

第三个参数一定要设为 true。

2. 当 DOM Ready、window.__bl 实例产生后，通过手动方式从 window.__sourceError 解析错误信息。

```
window.__bl && __bl.error(new Error('发生了一个资源加载的错误'), {
  filename: '',       })
// 如果需要方便后台报警设置,可调用 __bl.sum('error-404', 1); 表示某些事件发生的次数。
```

关于因跨域资源共享导致的 Script Error，请参考 #unique_38。

3.5 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。

快速入门

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，在前端监控页面上单击应用名称。
3. 在左侧导航栏中选择应用 > JS 错误诊断。

查看错误总览

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。

图 3-7: 应用层面的错误总览

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。详见Source Map文件。

- 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。在曲线图上双击即可还原视图。

> **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。在页面错误率排行或页面错误率Top 5 页签单击分析后，展示的是对应页面的总览信息。

图 3-8: 页面层面的错误总览

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。

图 3-9: 页面错误率排行

在页面错误率排行页签可执行以下操作：

分析：单击操作列中的分析，查看该页面的错误总览视图。

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 JS 错误诊断页面。
- 单击页面错误率Top 5 页签，然后单击操作列中的分析，进入该页面的错误总览页面。

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。

在高频错误页签可执行以下操作：

诊断：单击操作列中的诊断，进入错误详情页签。

> 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。在页面错误率排行或页面错误率 Top 5 页签单击分析后，高频错误页签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
  - 地区
  - 行
  - 列
  - URL
  - 文件（出现 JS 错误的文件路径）
- 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。

图 3-10: JS 错误详情页面

在错误详情页签上可执行以下操作：

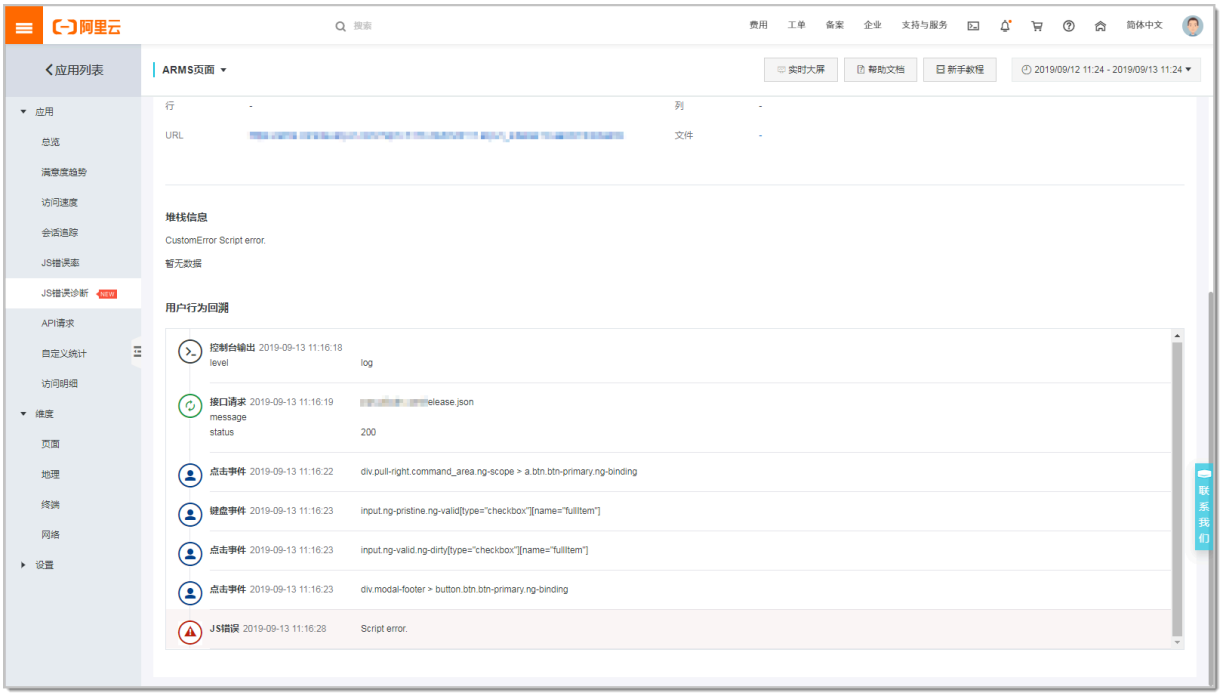
- 如需确定 JS 错误的准确出错位置，请在堆栈信息区域单击一条堆栈信息左侧的三角形图标展开该行，单击选择 Source Map，然后在 Source Map文件对话框中选择现有的 Source Map 文件或上传新的 Source Map 文件，最后单击确定。

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：最近 24 小时和最近 30 天。
- 浏览器。
- 操作系统。
- 设备。

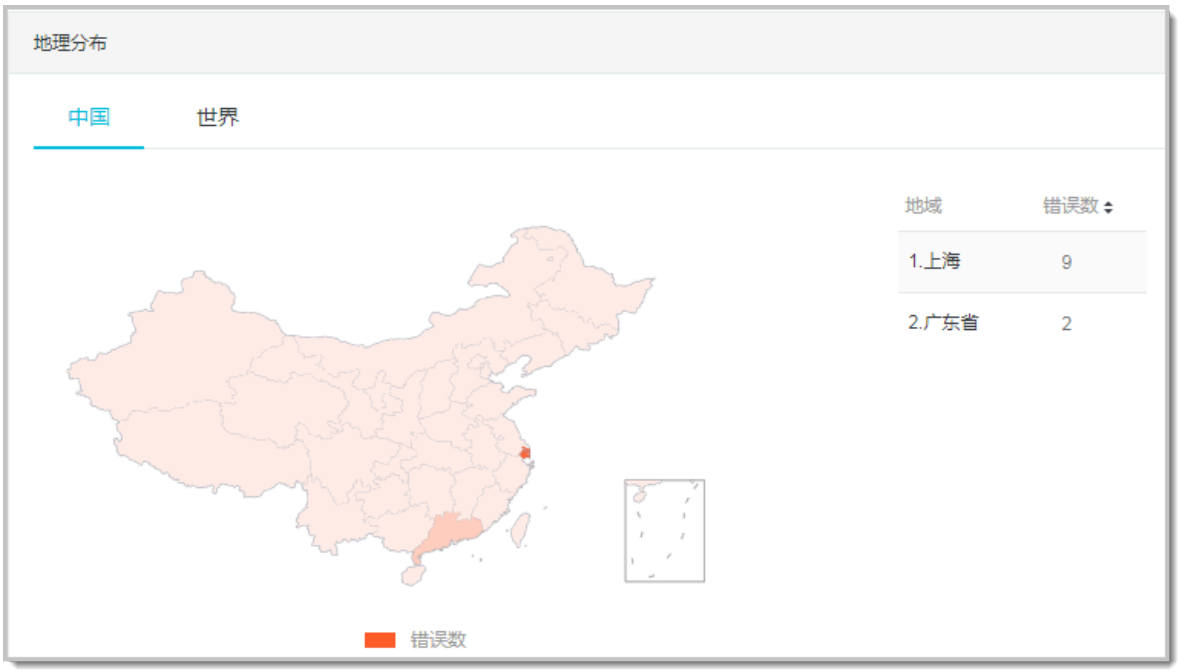
· 地理：在中国维度下按省、直辖市、自治区统计，在世界维度下按国家/地区统计。

图 3-11: 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，将鼠标悬浮于曲线上，曲线拐点所对应时间点的错误数将显示在浮层中。

- 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列标题，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）。

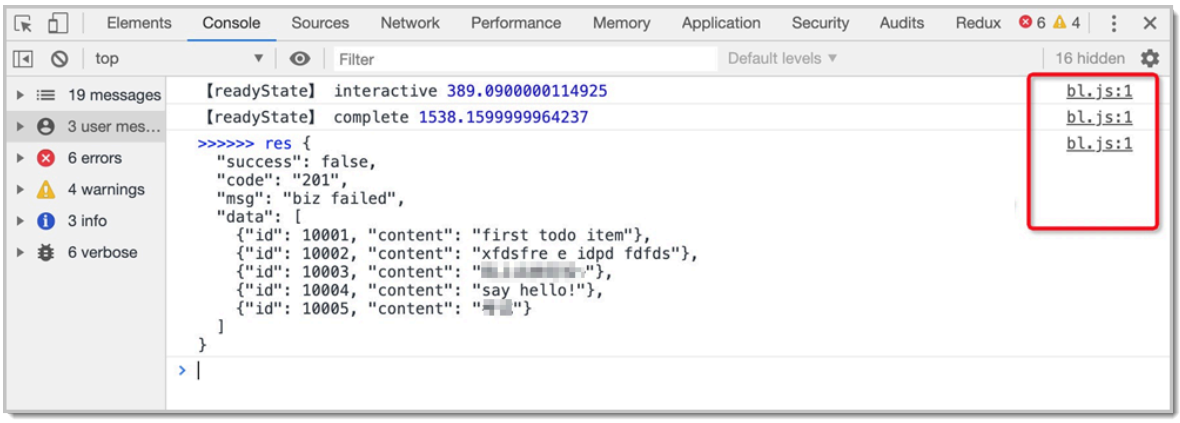


常见问题

- 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 配置项的详细信息参见#unique_9。

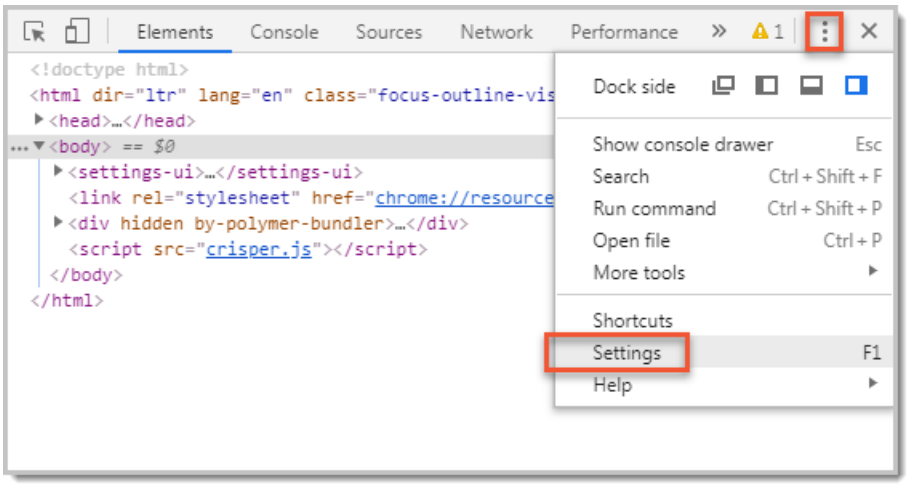
- 开启用户行为回溯后，调试过程中通过 `console.log` 打印出的信息会定位到 ARMS 的 SDK 代码 `bl.js` 中，而不是原代码中的位置，如何解决?



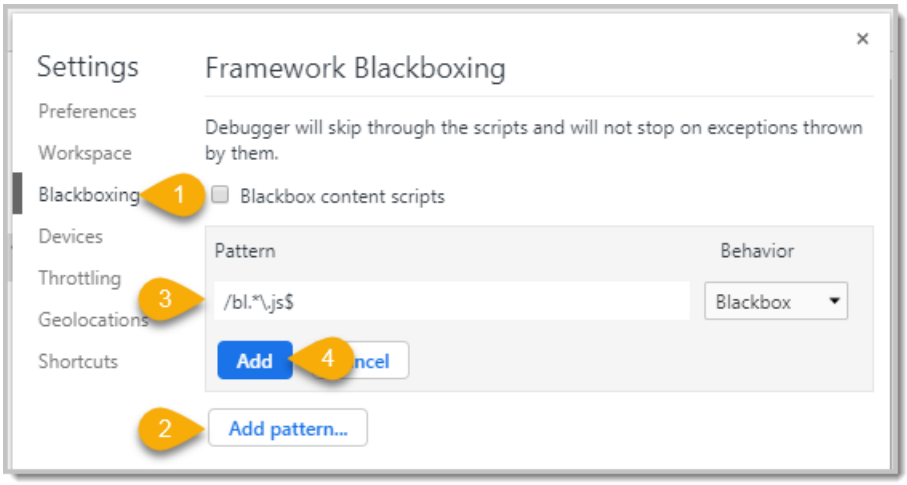
造成这种现象的原因是 ARMS 通过重写 `console` 对象的 `log` 等方法来监控浏览器控制台打印的内容。解决方法为:

- 方法一（推荐）：设置 Chrome 浏览器的黑盒（Blackboxing）。

  1. 打开 Chrome 浏览器，按 Ctrl + Shift + I 打开开发者工具面板，在设置菜单中单击 Settings。



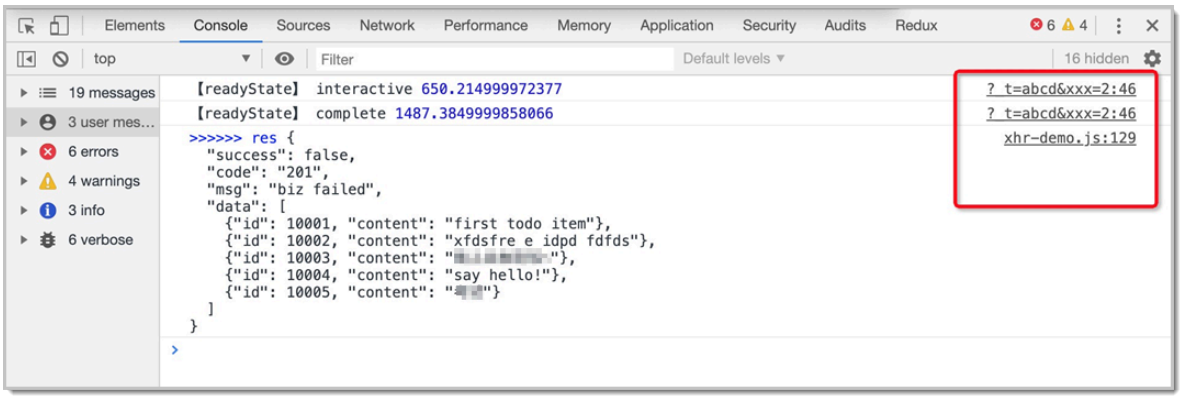
  2. 在 Settings 面板左侧单击 Blackboxing，单击 Add pattern，在 Pattern 文本框中输入 `/bl.*\.js$`，并单击 Add。



- **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)) setAttribute("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原代码中的位置。



相关文档

#unique_9

3.6 API 请求监控

本文介绍了前端监控中的 API 请求监控功能。

功能介绍

阿里云 ARMS 前端监控的 API 请求监控模块，可清晰展示以下信息：

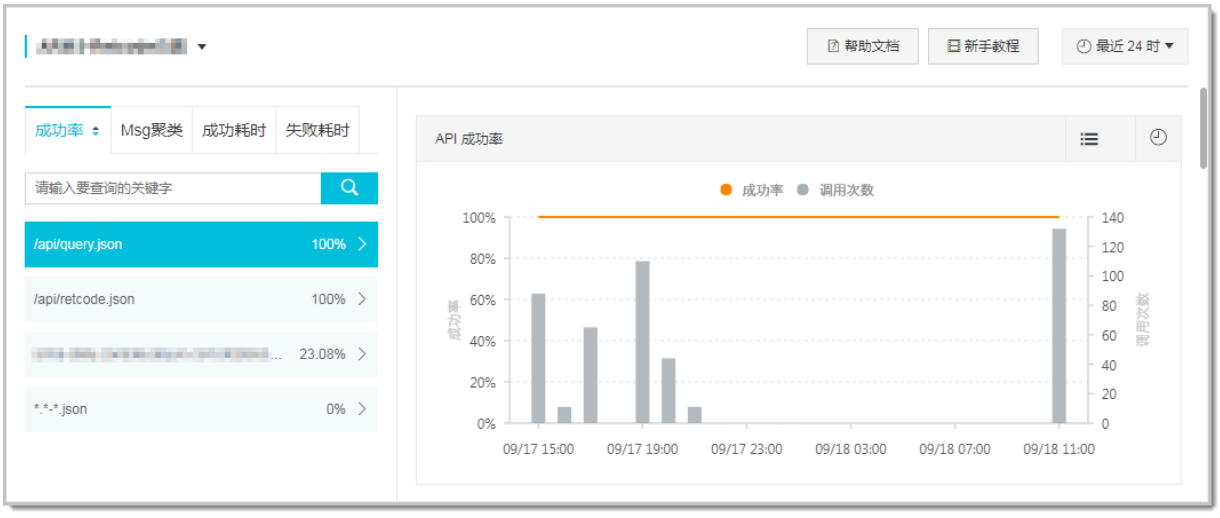
- 每个 API 的成功率
- API 返回信息
- API 接口的调用成功平均耗时
- API 接口的调用失败平均耗时

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：

- 地理位置
- 浏览器
- 操作系统
- 设备
- 分辨率

API 成功率

左侧的成功率标签页展示的是 API 成功率排行。右侧图表展示的是左侧列表选中 API 在指定时间范围内的 API 调用量和成功率曲线。



API 返回信息

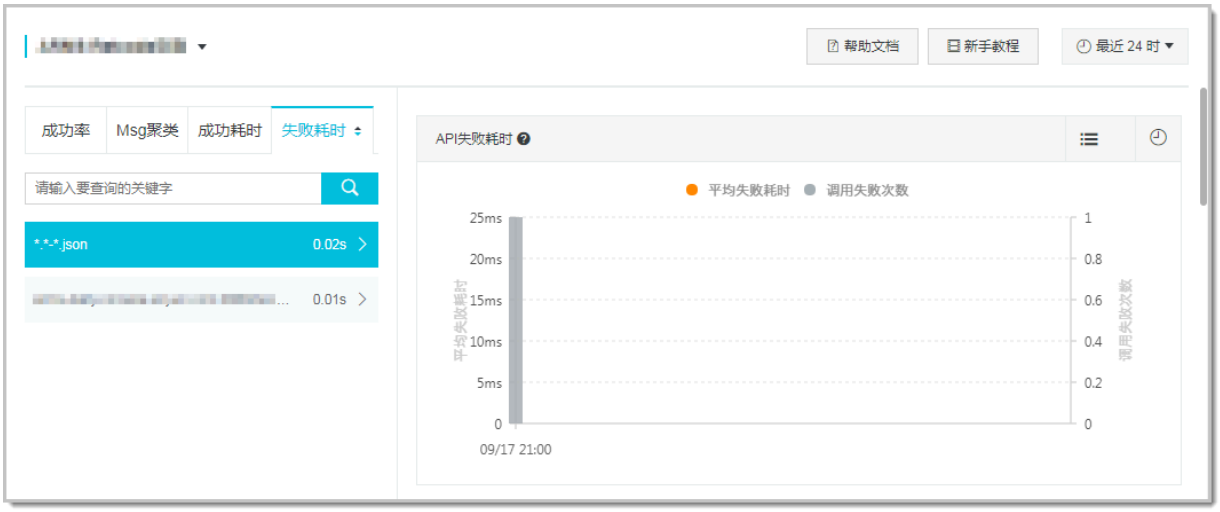
左侧的 Msg 聚类标签页展示的是 API 返回信息排行。右侧的 Msg 调用详情区域展示的是左侧列表选中返回信息的 API 调用列表，按调用量降序排列。

API 成功耗时

左侧的成功耗时标签页展示的是 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 API 的调用成功平均耗时曲线和调用成功次数柱形图。

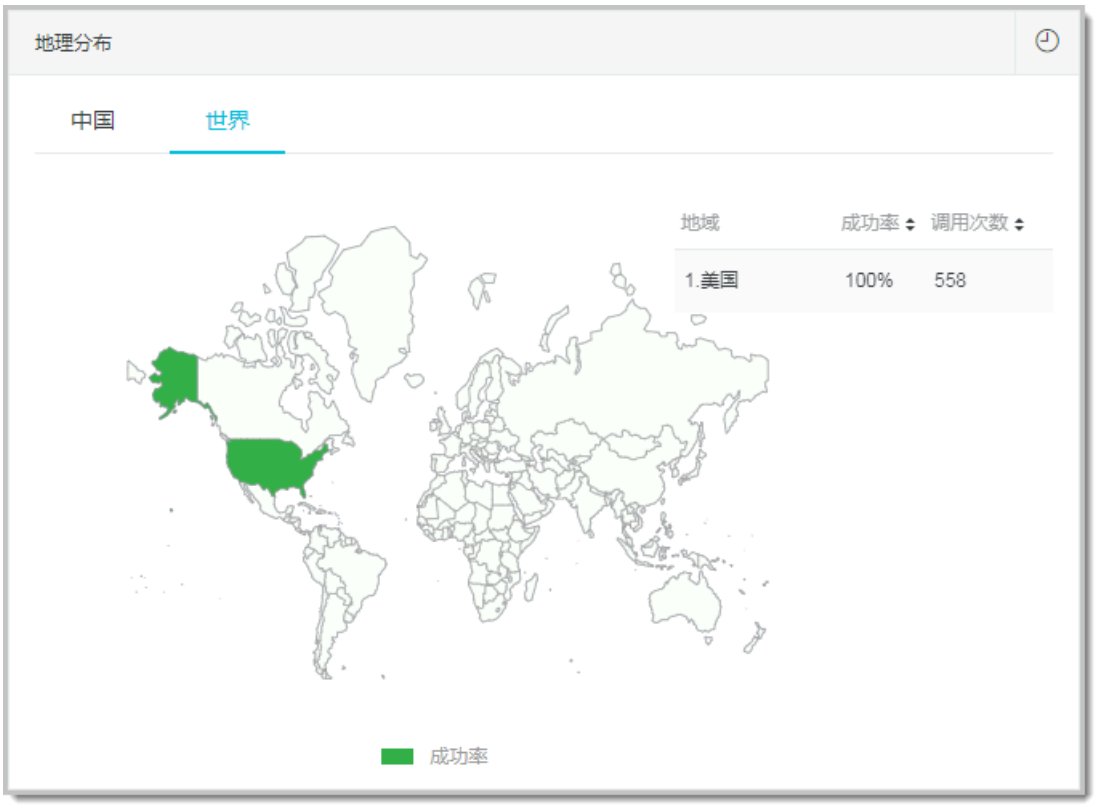
API 失败耗时

左侧的失败耗时标签页展示的是 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 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

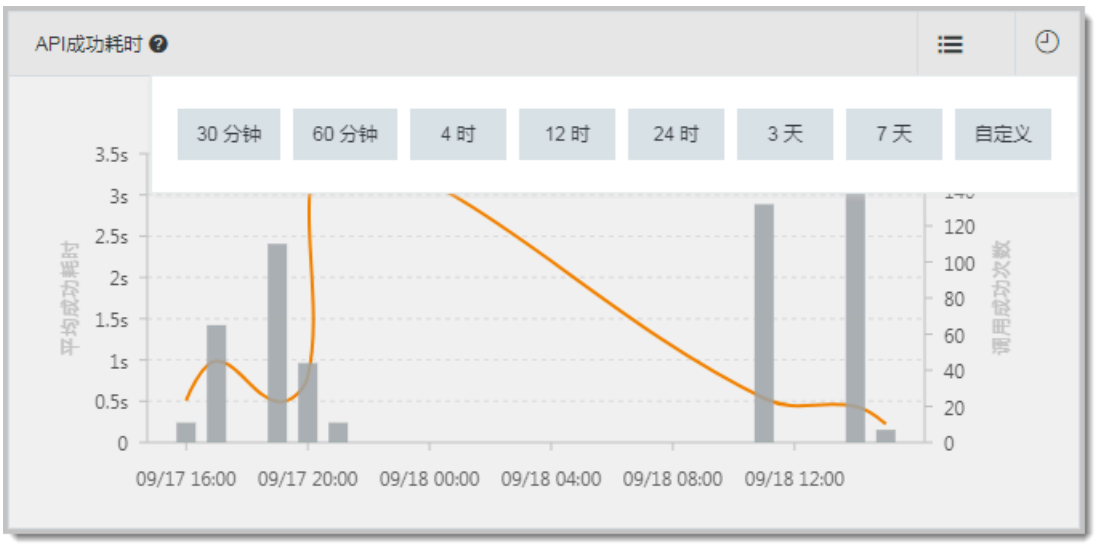
在终端分布模块中，您可以查看上述统计信息的终端分布情况。终端分布又细分为浏览器、操作系统、设备、分辨率等维度。

通用操作

在 JS 错误统计模块中，您可以执行以下通用操作。

- 在左侧标签页上，单击标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。



3.7 自定义统计

本文介绍了前端监控中的自定义统计功能。

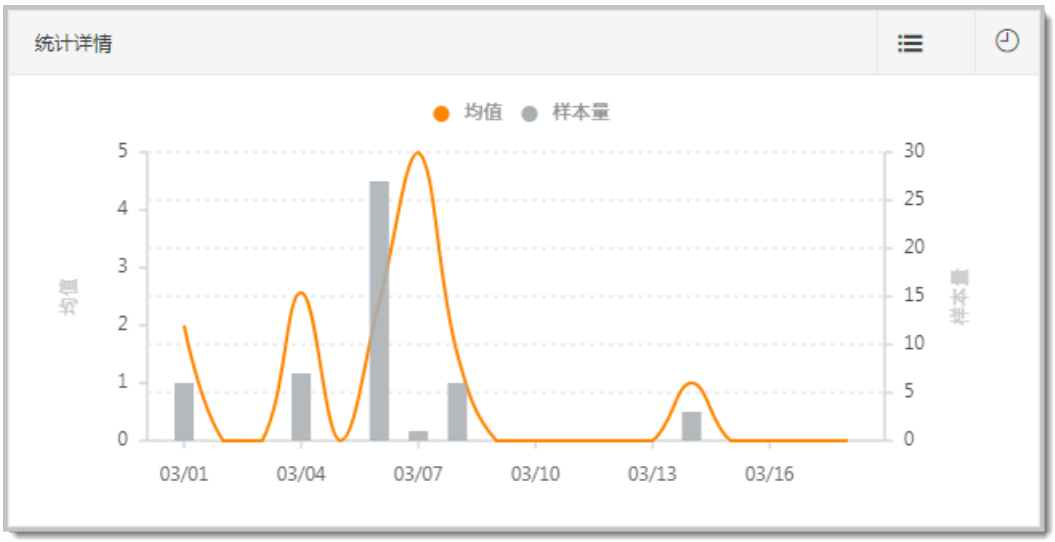
为帮助您监控和统计轻量级的业务交互行为，ARMS 前端监控提供了以下两类自定义统计功能：

- 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 都提供了以下三个维度的统计数据（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。



- 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	-
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	-
value	Number	统计上报量，默认 0	否	0

示例：

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

3.8 前后端链路追踪

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。

必须开通阿里云 ARMS 前端监控和应用监控服务并搭配使用二者，才能获得前后端的完整调用链路。

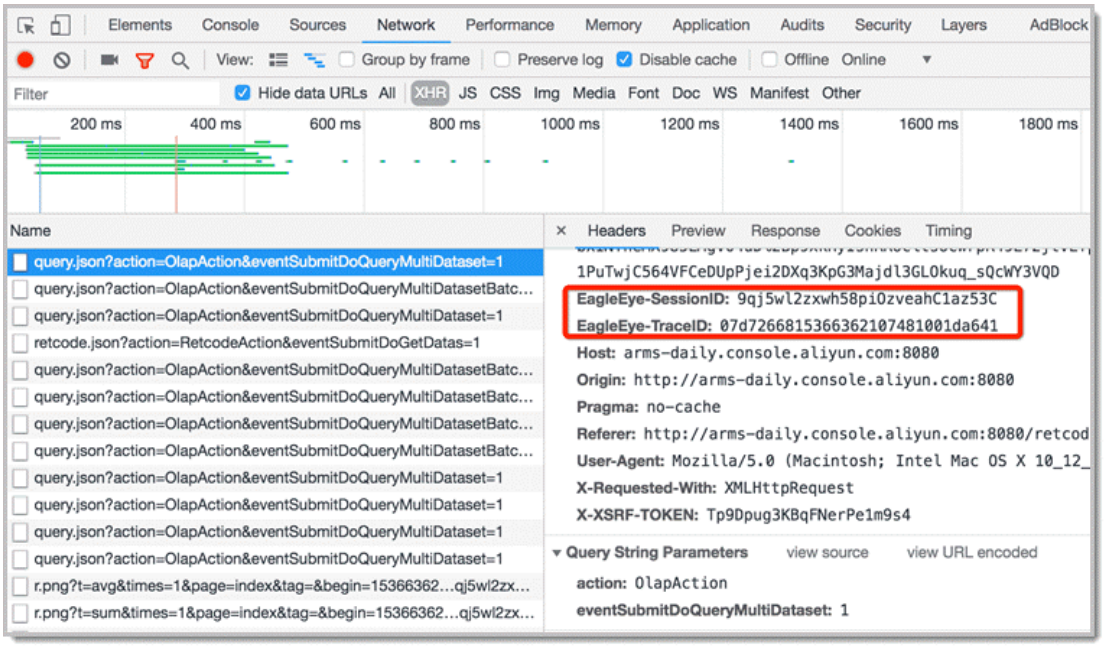
- 阿里云 ARMS 前端监控
  - SDK 配置
    - 允许 API 自动上报，即不关闭 API 自动上报。
    - 允许前后端链路追踪，即配置 enableLinkTrace 为 `true`。具体配置为：

```
<script>
!(function(c,b,d,a){c[a]||(c[a]={});c[a].config={pid:"xxx",
imgUrl:"https://arms-retcode.aliyuncs.com/r.png?", enableLink
Trace: 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>
```

  - 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（如图所示）。

警告：

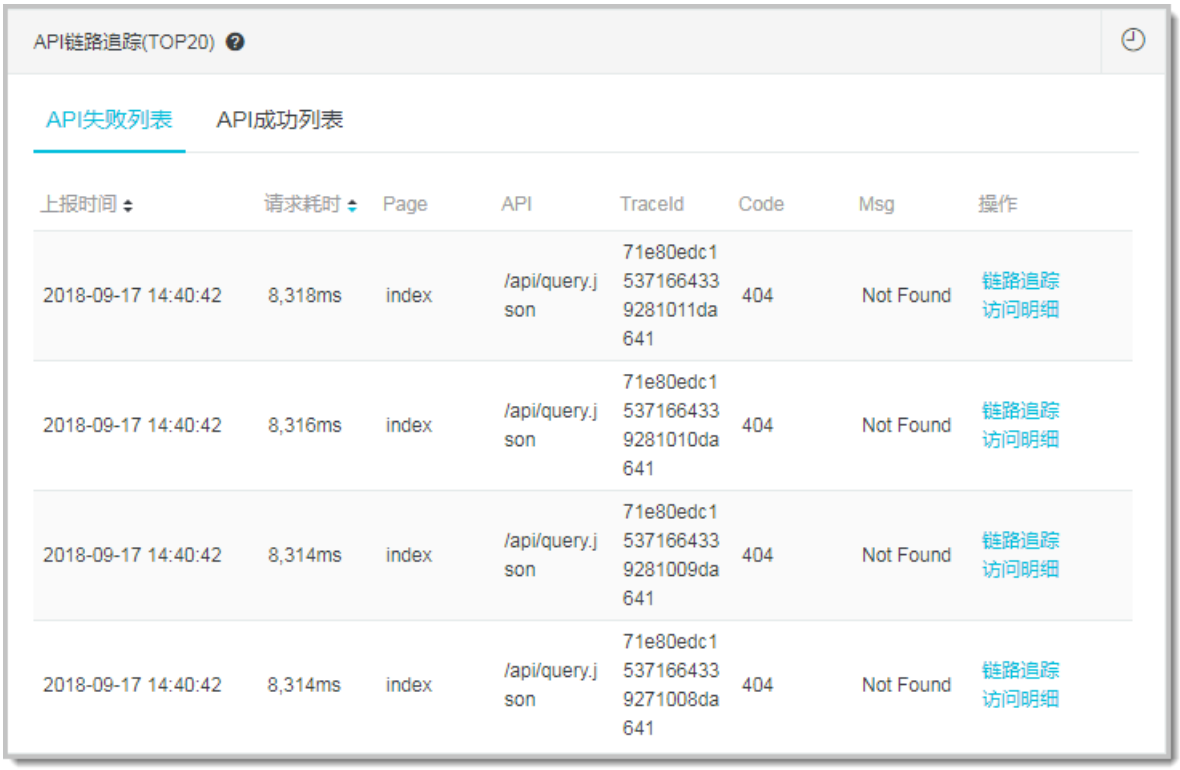
EagleEye-TraceID 和 EagleEye-SessionID 的值有相应的含义，请勿自行生成。



- 阿里云 ARMS 应用监控
  - 关于应用监控的配置，请参考#unique_43。
  - 仅 2.4.5 或更高版本的应用监控探针支持该功能。

使用方法

在阿里云 ARMS 前端监控中，总览、API 请求和页面这三个页面均提供了 API 链路追踪模块。



您可以在此模块中执行以下操作：

- 按照上报时间或请求耗时对 API 进行升序或降序排列。
- 单击链路追踪，即可进入调用链路标签页，查看前端监控的整体耗时和后端应用的调用时序图。
- 单击访问明细，即可进入查看详情页面，查看 API 请求的详细信息。

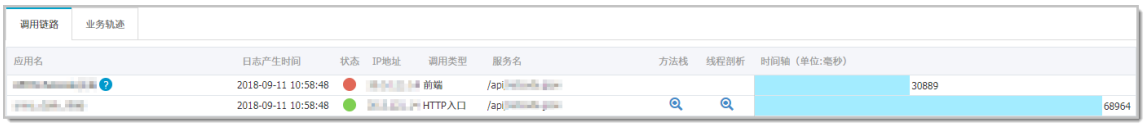
使用场景和案例

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。

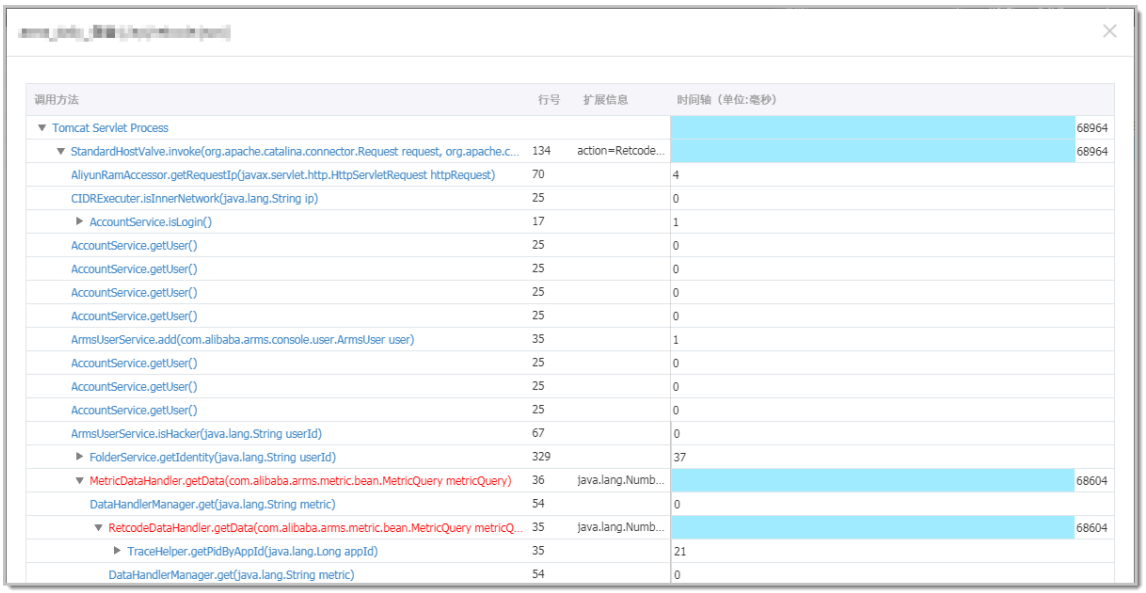
通过调用的时间轴，可以知道是网络传输还是后端调用的时间长，同时点击后端应用中的线程剖析，可以看到本次请求后端的完整调用链路。从而根据业务定位是什么原因导致 API 错误。

- 当 API 返回错误码或者业务逻辑错误时定位问题

1. 在 API 链路追踪模块的 API 失败列表中，找到相关的 API 或者 TraceID，并单击相应的链路追踪按钮，以查看前端监控的整体耗时和后端应用的调用时序图。



2. 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。
3. 对后端应用单击方法栈栏中的放大镜图标，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

  1. 在 API 链路追踪模块中，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。
  2. 在耗时较长的记录所在行上，单击相应的链路追踪按钮，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。

4 高级选项

4.1 前端监控 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，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。

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("crossorigin","",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，修改方法参见 #unique_8。

SDK 配置项参数

参数	类型	描述	是否必需	默认值
`pid`	String	项目唯一 ID，由 ARMS 在创建站点时自动生成。	是	无
`tag`	String	传入的标记，每条日志都会携带该标记。	否	无
`page`	String	页面名称。	否	默认取当前页面 URL 的关键部分：`host + pathname`。

参数	类型	描述	是否必需	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
parseHash	Function	与 enableSPA 搭配使用，参见 parseHash：将 URL Hash 解析为 Page。	否	参见 parseHash：将 URL Hash 解析为 Page。
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。
ignoreUrlCase	Boolean	忽略 Page URL 大小写。	否	true：默认忽略。
urlHelper	*	代替旧参数 ignoreUrlPath，用于配置 URL 过滤规则，参见 urlHelper：URL 过滤。	否	参见 urlHelper：URL 过滤。
apiHelper	*	代替旧参数 ignoreApiPath，用于配置 API 过滤规则，参见 apiHelper：API 过滤。	否	参见 apiHelper：API 过滤。
parseResponse	Function	用于解析自动上报 API 时返回的数据，参见 config.parseResponse：Response 解析方法。	否	参见 config.parseResponse：Response 解析方法。
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 和 ignoreErrors，参见 ignore：忽略指定 URL、API、JS 错误。	否	参见 ignore：忽略指定 URL、API、JS 错误。
disabled	Boolean	禁用日志上报功能。	否	false
sample	Integer	日志采样配置，值为 1、10 或 100。性能和成功 API 日志按照 1/sample 的比例采样，参见 config.sample：日志采样上报配置。	否	1
sendResource	Boolean	上报页面静态资源，参见 sendResource：上报静态资源。	否	false

参数	类型	描述	是否必需	默认值
useFmp	Boolean	采集首屏 FMP（First Meaningful Paint，首次有效渲染）数据。	否	false
enableLinkTrace	Boolean	进行前后端链路追踪，参见#unique_46。	否	false
release	String	应用版本号。	否	undefined
environment	String	进行前后端链路追踪的环境。	否	production

parseHash：将 URL Hash 解析为 Page

单页面应用场景中（参见#unique_47），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';
        }
    });
```

```
});
```

urlHelper：URL 过滤

当页面 URL 类似于 http://xxx.com/projects/123456（projects 后面的数字是项目 ID）时，如果将xxx.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。

说明：

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#unique_8）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。

apiHelper：API 过滤

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：URL 过滤。

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。

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。

config.parseResponse：Response 解析方法

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.errorResponse || '';
    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。

ignore：忽略指定 URL、API、JS 错误

ignore 的值是一个对象，对象中包含 3 个属性：ignoreUrls、ignoreApis 和 ignoreErrors。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: []
```

```
    },
```

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;
                        }
                        return false;
                    }]
                }
            });
```

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;
                        return false;
                    }]
                }
            });
```

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;
                        return false;
                    }]
                }
```

```
            });
```

config.sample：日志采样上报配置

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 错误和失败 API 等其他类型日志的上报。

采样值 sample 默认为 1，值可以是 1、10 或 100，对应的采样率为 1/sample，即 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。

sendResource：上报静态资源

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("crossorigin","",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。

更多信息

- #unique_47
- #unique_8

4.2 API 使用指南

本文介绍了前端监控 SDK 的一些接口及其使用场景。

SDK 开放了部分数据上报接口，用户可在页面中自行调用来上报更多数据。

api() 接口调用成功率上报

此接口用于上报页面的 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`disabledHook`设置为`true`，具体配置参见 #unique_9。

调用参数说明：`__bl.api(api, success, time, code, msg)`

参数	类型	描述	是否必须	默认值
api	String	接口名	是	-
success	Boolean	是否调用成功	是	-
time	Number	接口耗时	是	-
code	String/Number	返回码	否	''
msg	String	返回信息	否	''

示例

```
var begin = Date.now(),
    url = '/data/getTodoList.json';

$.ajax({
    url: url,
    data: {id: 123456}
}).done(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);
    // do something ....
}).fail(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);
    // do something ...
});
```

error() 错误信息上报

此接口用于上报页面中的 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。

调用参数说明：`__bl.error(error, pos)`

参数	类型	描述	是否必须	默认值
error	Error	JS 的 Error 对象	是	-
pos	Object	错误发生的位置，包含以下3个属性	否	-
pos.filename	String	错误发生的文件名	否	-
pos.lineno	Number	错误发生的行数	否	-
pos.colno	Number	错误发生的列数	否	-

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);
});
```

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
});
```

sum() 求和统计

此接口用于统计业务中某些事件发生的次数。

调用参数说明： `__bl.sum(key, value)`

参数	类型	描述	是否必须	默认值
key	String	事件名	是	-
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);

__bl.sum('group-x::event-c', 2);
```

avg() 求平均统计

此接口用于统计业务场景中某些事件发生的平均次数或平均值。

调用参数说明：`__bl.avg(key, value)`

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

示例

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

__bl.avg('events::event-c', 10);
__bl.avg('speed::event-d', 142.42);
```

其它接口

非日志上报接口，一般用于修改 SDK 的部分设置项。

setConfig() 修改配置项

用于在 SDK 初始完成后重新修改部分配置项，具体配置参见 #unique_9。

调用参数说明：`__bl.setConfig(next)`

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

示例：修改 disableHook 禁用 API 自动上报

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

setPage() 设置当前页面的 page name

用于重新设置页面的 page name（默认会触发重新上报 PV）。此接口一般用于单页面应用，更多信息请参考#unique_47。

调用参数说明：`__bl.setPage(next, sendPv)`

参数	类型	描述	是否必须	默认值
page	String	新的 page name	是	-
sendPv	Boolean	是否上报 PV，默认会上报	否	`true`

示例 1：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);
```

示例 2：仅设置当前页面的 page 为'homepage'，但不触发 PV 上报

```
__bl.setPage('homepage', false);
```

4.3 前端监控进阶场景

ARMS 前端监控 SDK 提供了若干高级选项，可满足进阶场景下的前端监控需求，例如 SPA 页面上报、数据预上报和自定义性能指标上报。

SPA 页面上报

SPA（Single Page Application）即单页面应用。在此类型的应用中，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。

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

  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 `<Function>parseHash`（参见#unique_9）。

- 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。

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

数据预上报

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'],

    // 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'];
```

自定义性能指标上报

您可以使用 __bl.pipe 方法上报以下自定义性能指标：

- 自定义首屏时间 cfpt
- 自定义首次可交互时间 ctti
- 其他自定义性能指标 t1 ~ t10（共计 10 个）

上述自定义性能指标是在页面 Load 事件触发前上报的，且这些指标是同时上报的，即不会逐一上报。

以 CDN 方式接入时，上报自定义性能指标的示例如下：

```
__bl.pipe=[ 'performance',
```

```
   {
      cfpt: 100,
      ctti: 20,
      t1: 300,
      t2: 300,
      t3: 60,
      t4: 1300,
      t5: 300,
      t6: 1300,
      t7: 40,
      t8: 300,
      t9: 300,
      t10: 4100
    }];
```

以 Weex 方式接入时，上报自定义性能指标的示例如下：

```
const BrowserLogger = require('alife-logger');
// 与 CDN 的 Pipe 结构一致
const pipe = ['performance', {cfpt:100, ctti:20, t1:300, t2:300, t3:60
, t4:1300, t5:300, t6:1300, t7:40, t8:300, t9:300, t10:4100}];
const __bl = BrowserLogger.singleton({pid:'站点唯一ID'},pipe);
```

5 使用教程

5.1 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，请参见文档#unique_52来进行接入。

> 注意:
> 在接入时勾选开启页面资源上报项，才能使用慢会话追踪功能。

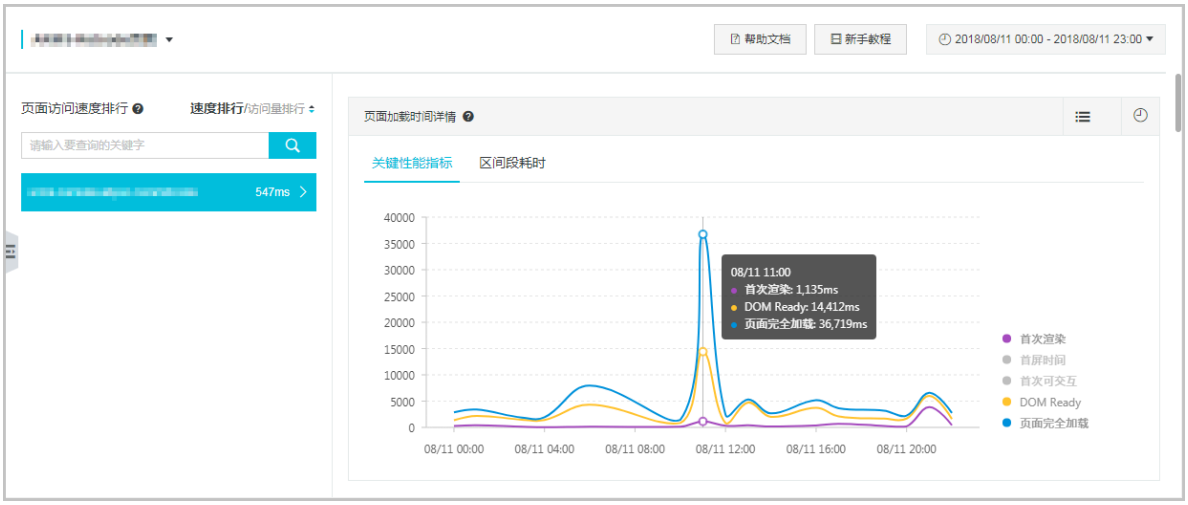
步骤二：定位故障

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。

方式一：从访问速度开始排查

1. ARMS 控制台，在左侧导航栏中单击前端监控。
2. 在前端监控页面上，单击您的应用名称。
3. 在左侧导航栏中单击访问速度。

   访问速度页面详情介绍请参见#unique_53。在本示例中，该页面性能较差，在 11:00 时左右的页面完全加载时间达到 36.7 s。



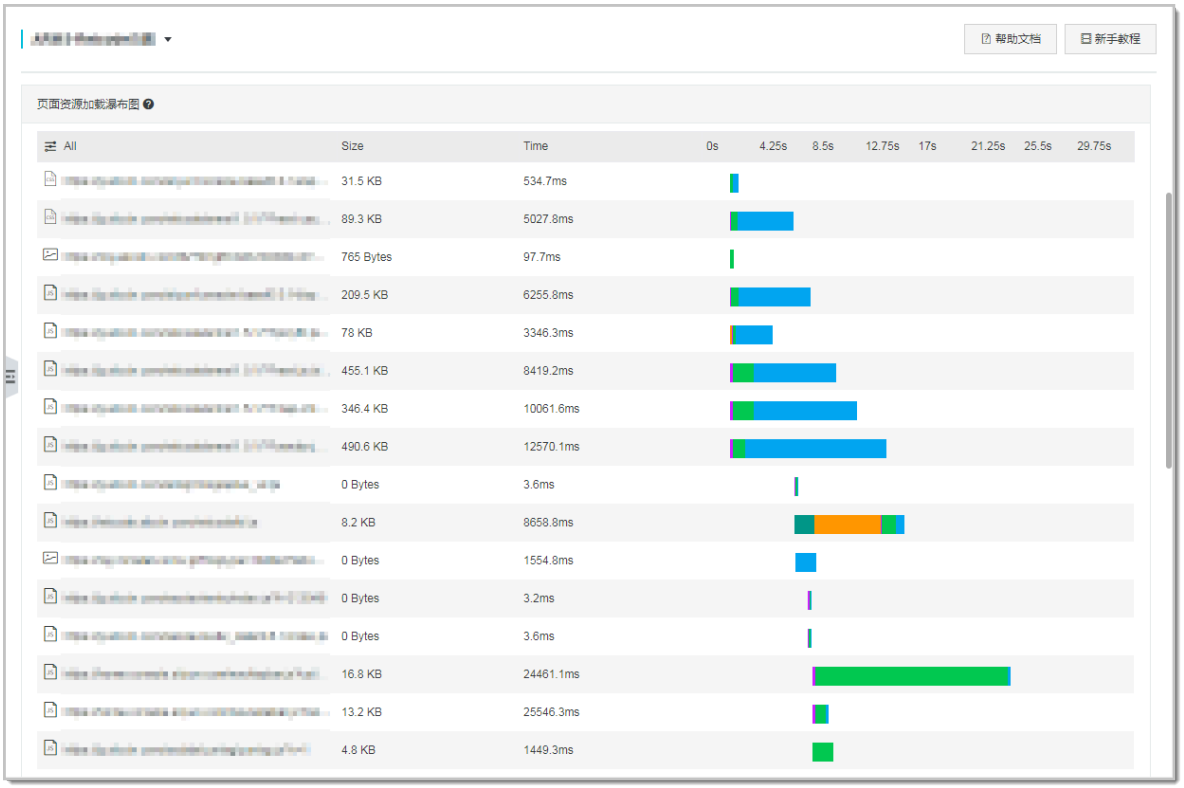
4. 在访问速度页面上，拖动右侧的滚动条至慢页面会话追踪（TOP20）模块。该模块会列出该页面在指定时间段内加载最慢的 20 个会话。

   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。

5. 在慢页面会话追踪（TOP20）模块的页面列中单击目标页面名称，进入会话详情页面。然后根据会话详情页的信息定位故障原因，进而排除故障。

   会话详情页面顶部的页面信息模块展示了本次访问的客户端 IP 地址、浏览器、操作系统等信息，帮助您进一步确认故障原因。

   会话详情页面的页面资源加载瀑布图模块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。



   更多慢会话追踪页面详情请参见#unique_54。

方式二：从会话追踪开始排查

1. ARMS 控制台，在左侧导航栏中单击前端监控。
2. 在前端监控页面上，单击您的应用名称。

3. 在左侧导航栏中单击会话追踪，进入会话列表页。

   会话列表页展示了该应用下加载时长 Top 100 的会话，可以按照页面、会话 ID、浏览器、浏览器版本号进行过滤，帮助用户可以快速发现耗时较长的会话信息。

4. 在会话列表页单击目标页面操作列的详情。然后根据会话详情页的信息定位故障原因，进而排除故障。

   慢会话追踪页面详情请参见#unique_54。

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。

后续操作

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#unique_55。

更多信息

- #unique_53
- #unique_54
- #unique_55

6 参考信息

6.1 统计指标说明

本文说明了 ARMS 应用监控各页面的关键统计指标的含义。

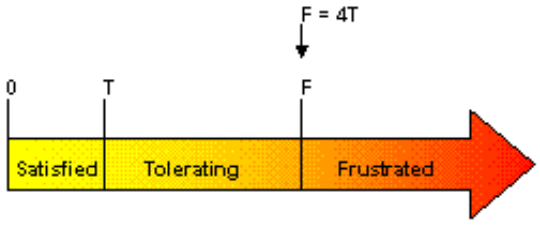
满意度

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感定义为三个等级：

- 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。

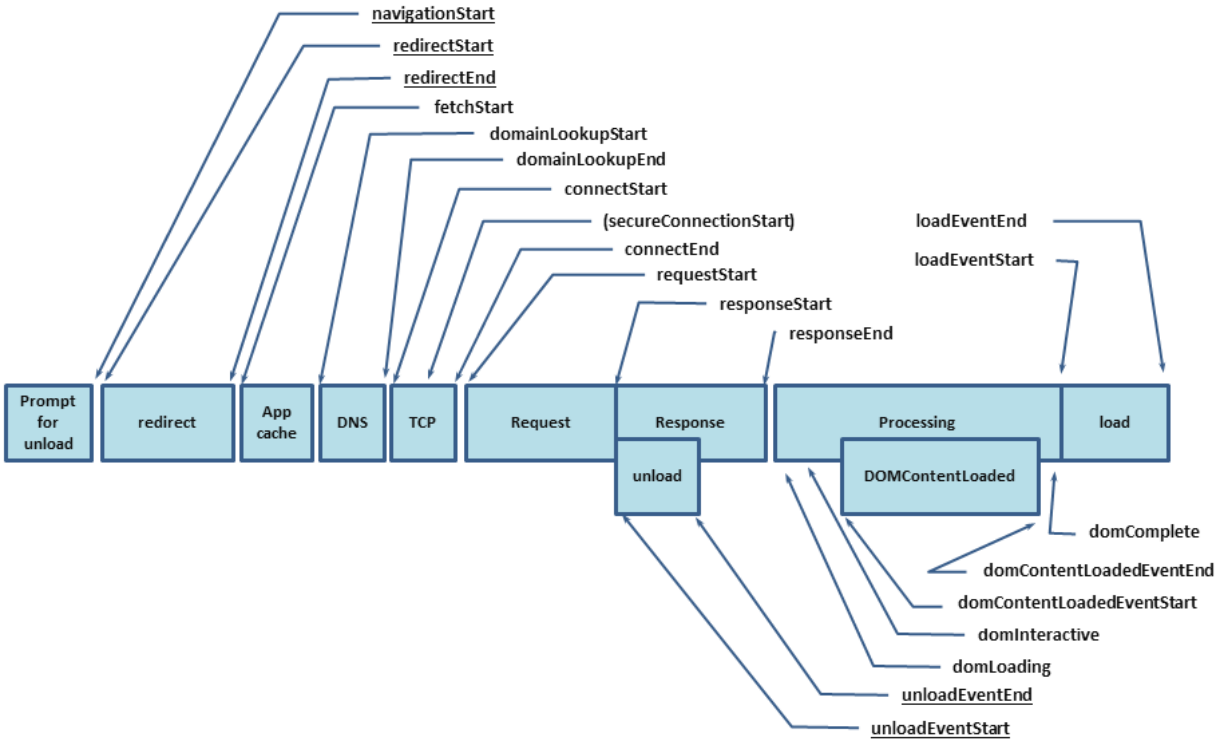
```
错误率 = 错误样本量 / 总样本量
```

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

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

字段含义

表 6-1: 阶段耗时

上报字段	描述	计算方式	备注
dns	DNS 解析耗时	domainLookupEnd - domainLookupStart	无
tcp	TCP 连接耗时	connectEnd - connectStart	无
ssl	SSL 安全连接耗时	connectEnd - secureConnectionStart	只在 HTTPS 下有效
ttfb	Time to First Byte（TTFB），网络请求耗时	responseStart - requestStart	TTFB 有多种计算方式，ARMS 以 Google Development 定义为准
trans	数据传输耗时	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

表 6-2: 关键性能指标

上报字段	描述	计算方式	备注
firstbyte	首包时间	responseStart - domainLookupStart	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

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 #unique_8上报的数据。