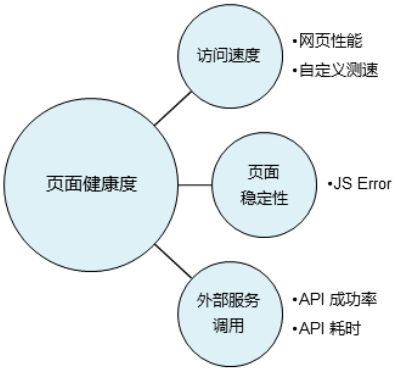
# 业务实时监控服务 ARMS

## 前端监控

# 前端监控

# 前端监控概述

ARMS 前端监控专注于 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。

**注意**：ARMS 前端监控不监控页面内容以及 API 内容。

## 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。

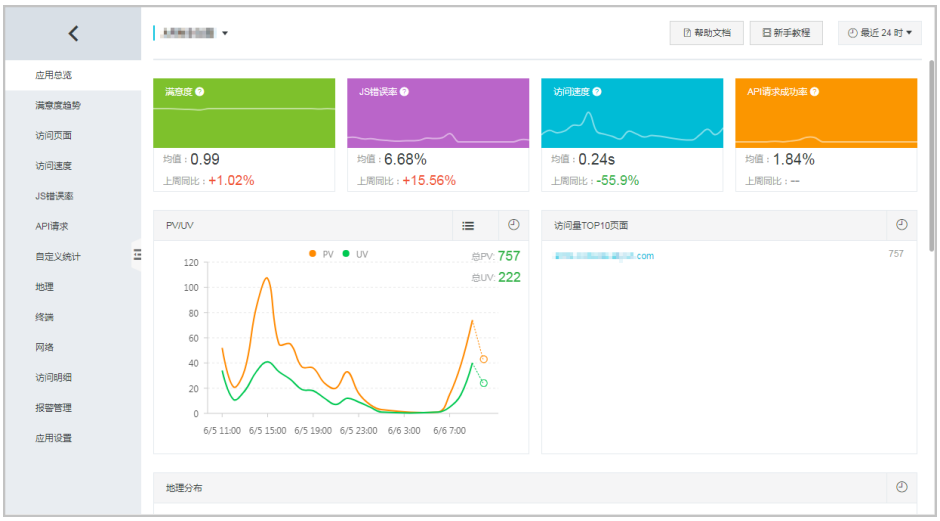
为了保证线上业务稳定运行，我们会在 Server 端对业务的运行状态进行各种监控。现有的 Server 端监控系统相对已经很成熟了，而页面加载和页面运行时的状态监控一直比较欠缺。例如：

- 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。




在新窗口中查看大图

# 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+ | | |
| Safari for iOS | Safari for iOS 9.2+ | | |
| Android Browser | android_webkit 4.4.2+ | | |
| Weex | Weex 0.16.0 + | | |

# 接入

# 快速接入前端监控

本文介绍了如何快速接入 ARMS 前端监控平台。

## 新建应用站点

登录 ARMS 控制台，在左侧导航栏中选择**前端监控**。

在**前端监控**页面单击**新建应用站点**。

在**新建应用站点**对话框中填写站点名称，并单击**确定**。

## 按照提示嵌入代码

在**应用设置**页面上按提示复制代码，并粘贴在<body>第一行，然后重启应用即可。

## 其他接入方式

如果您的前端页面需要 npm 或 weex 接入，则需要站点 pid。

要获取该值，请在**应用设置**页面的**复制/粘贴BI探针**标签页上，复制**站点ID** 文本框中的内容，并填写到对应的接入方式的 pid 配置项中。

- Npm 接入说明
- Weex 接入配置

# Npm 接入配置

本文介绍了 Npm 仓库的接入配置。

## 安装

在 npm 仓库中安装 alife-logger。

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

## 使用

### 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',
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

**注意：**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 | 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 BrowserLogger.singleton 获取实例。

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

关于 __bl 的其他 API 使用方式，请参考 API 使用指南。

## Config 配置

Config 配置与 cdn 引入配置相同。请参考 SDK 配置项。

## 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 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'],
// 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);
```

Pipe 的数据结构与 cdn 引入相同。

# Weex 接入配置

本文介绍了 weex 环境中云前端监控的接入配置。

## WeexLogger 的使用

### 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) 静态方法来初始化，需要在传入的 props 中设定相关配置，详情见通用 API。

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

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

在 WeexLogger 已经初始化的前提下，业务模块中使用 singleton() 方法来获取实例，再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';

const wxLogger = WeexLogger.singleton();

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 | - |
| 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，也可以在每次调用时获取实例。示例见**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

请参考 API 使用指南的“日志上报接口”。

# 控制台功能

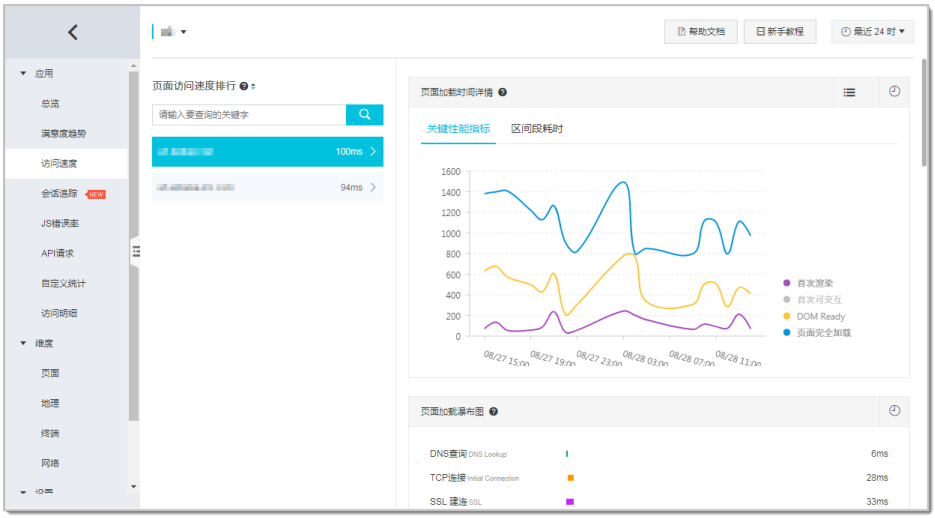
# 页面访问速度

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：

- 页面加载时间详情
- 页面加载瀑布图
- 性能样本分布
- 慢页面会话追踪
- 地理分布和终端分布

左侧的**页面访问速度排行**，是按照**页面首次渲染时间**指标排序的，单击上箭头或下箭头可改变排序顺序。




## 页面加载时间详情

此模块展示以下指标：

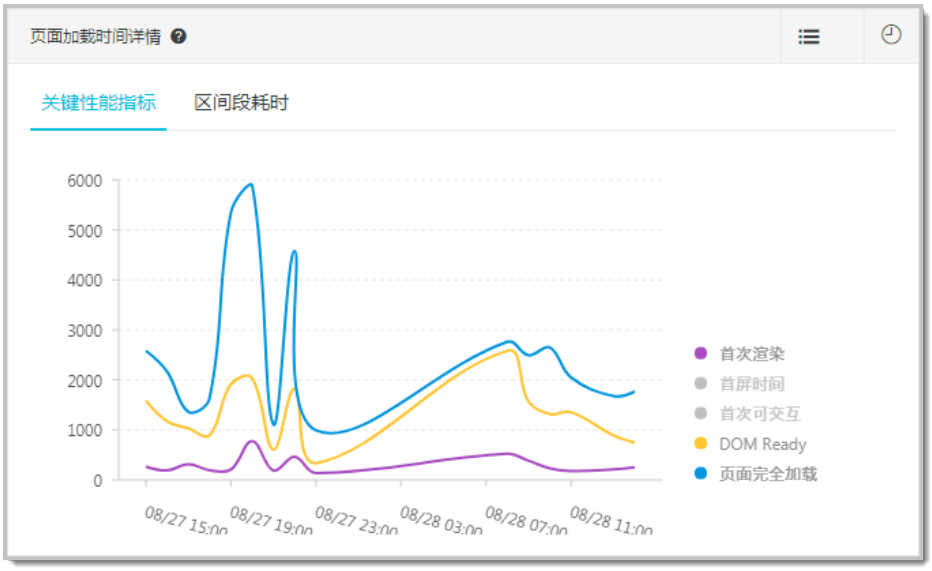
**关键性能指标**：

- 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

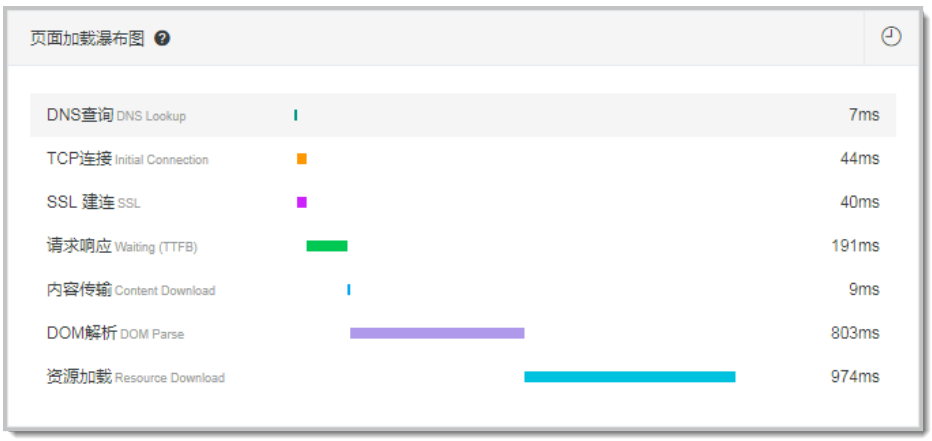
关于这些指标的计算方式，请参考**统计指标说明**。




**注意**：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。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。

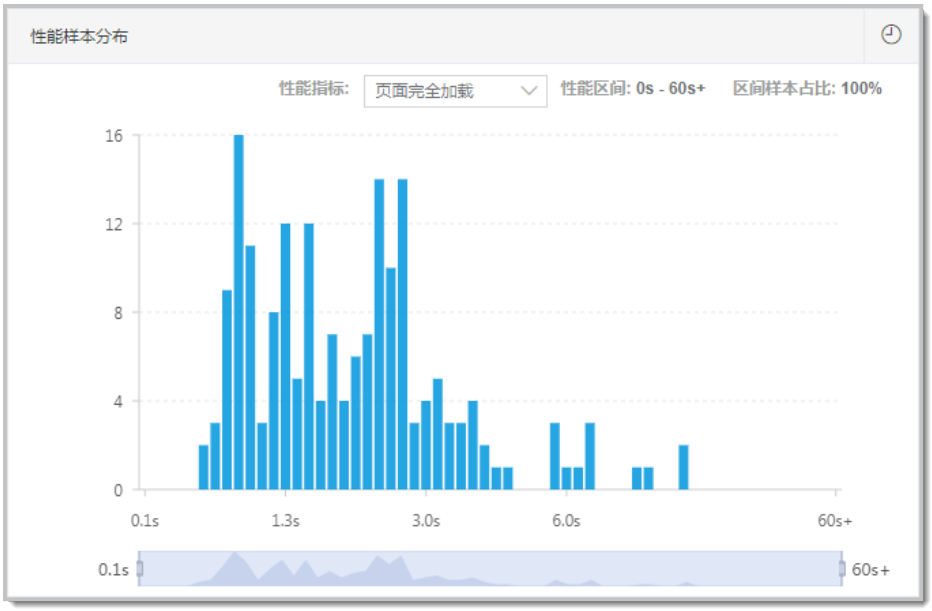
# 页面加载瀑布图

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。




# 性能样本分布

此模块非常直观地展示了样本性能分布情况。结合性能指标选项，您可以快速知道页面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，即秒开率是多少，或者长尾访问用户的样本占比。

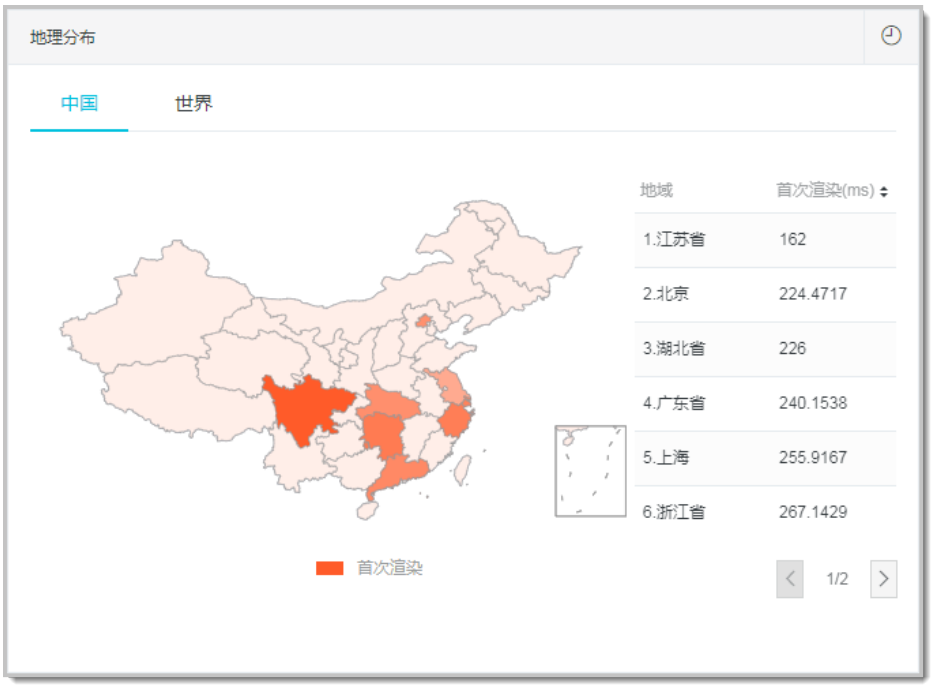



# 慢页面会话追踪

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。详情请参考慢会话追踪。

# 地理分布和终端分布

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和终端分布统计数据，以帮助您更好地定位性能瓶颈。



# 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，请在 SDK 的 config 部分将 sendResource 配置为 true。重新部署应用后，页面 onload 时会上报当前页面加载的静态资源信息，此后即可在阿里云前端监控平台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@8cc3f63543da641",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>
```

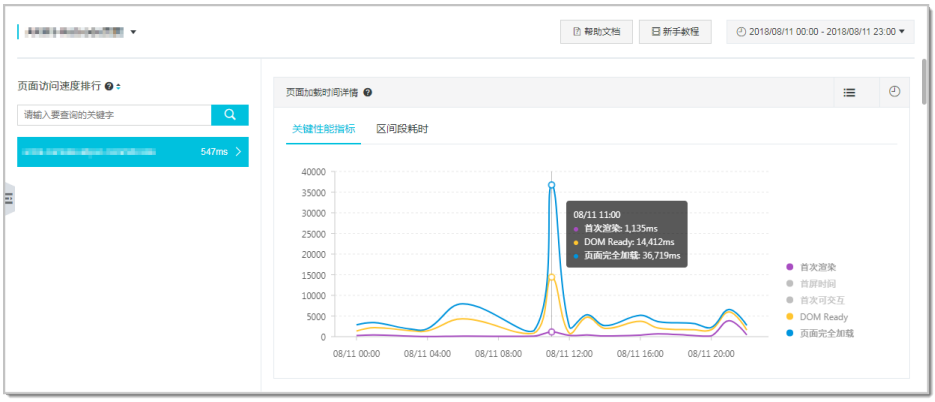
**注意**：静态资源加载信息的上报是在页面 onload 时触发的，上报信息量较大。如果应用对页面性能要求很高，则可以不开启该配置。

## 排查性能瓶颈问题

接下来以一个示例介绍如何排查性能瓶颈问题。

# 1. 发现问题

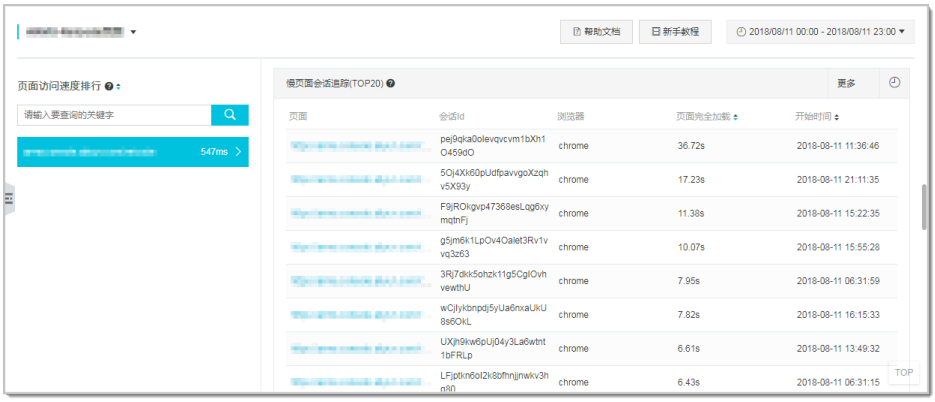
进入前端监控的**访问速度**页面后，发现页面性能较差。11:00 时的页面完全加载时间达到 36.7 秒。



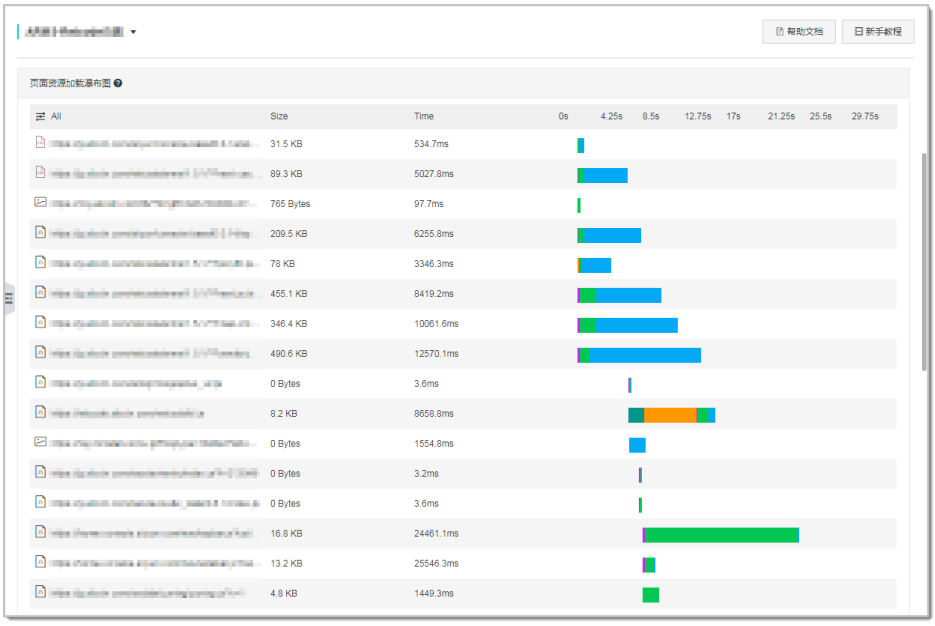

# 2. 慢页面会话追踪

在**访问速度**页面上，通过右侧的页面滚动条，可向下滚动至**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。

下图中 11:36 左右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。




在**慢页面会话追踪（TOP20）**模块的**页面**栏中单击页面名称，即可进入**会话详情**页面。借助该页面的**页面资源加载瀑布图**，您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。




在**会话详情**页面顶部的**页面信息**模块，您可以查看本次访问的客户端 IP 地址、浏览器、操作系统等 UA 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。

## 3. 其他发现问题的入口

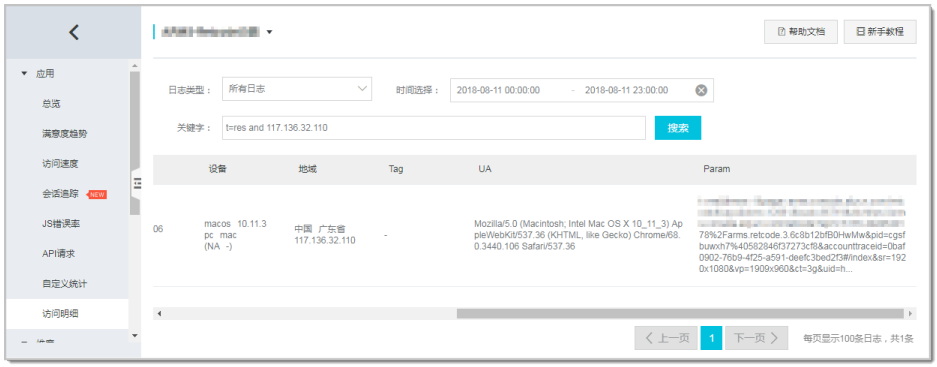
**会话追踪**页面

您也可以进入**会话追踪**页面，以查看该应用下的会话列表。会话列表会按从慢到快的顺序，列出页面完全加载时间最慢的前 100 个页面，帮助您快速发现耗时较长的会话。另外，您也可以按照页面、会话 ID、浏览器、浏览器版本号过滤相关会话信息。单击**操作**栏中的**详情**，即可查看该会话的页面资源加载详情。

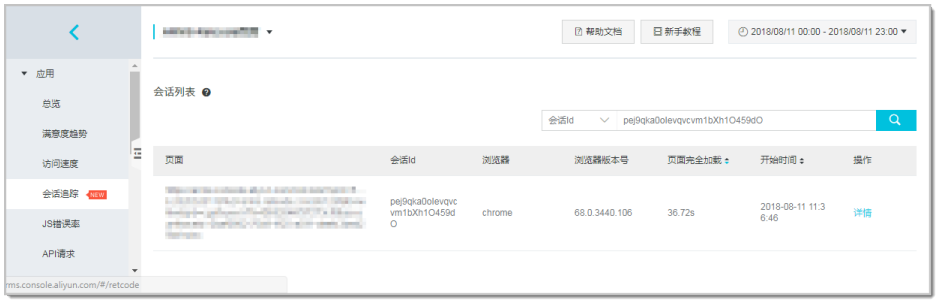
**访问明细**页面

如果当前会话列表中找不到您要排查的会话信息，可以通过**访问明细**页面查找到相应的日志详细信息，并在 param 中找到对应的 sid（即会话 ID）。然后在会话列表中查找此会话 ID，即可定位到要排查的会话信息。

例如：在已知用户的客户端 IP 的情况下，如需定位相应的会话信息，则可以在**访问明细**页面上，通过搜索关键字 t=res and 117.136.32.110，即可找到对应的会话 ID。




根据查找到的会话 ID，就可以在会话列表中通过过滤定位到具体的会话内容。

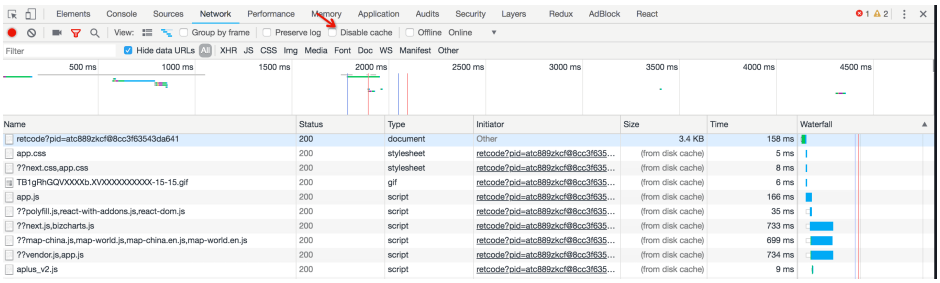



# 常见问题 1：资源加载瀑布图中 size 为 0

这里的 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）。但我们会发现该属性返回的内容有时会为 0，原因是：如果是从本地缓存获取资源，或者如果是跨源资源，则该属性返回零。

在 Chrome 浏览器中按 **F12** 打开控制台，当 **Disable cache** 选项未启用时，可以看到 transferSize 为 0：

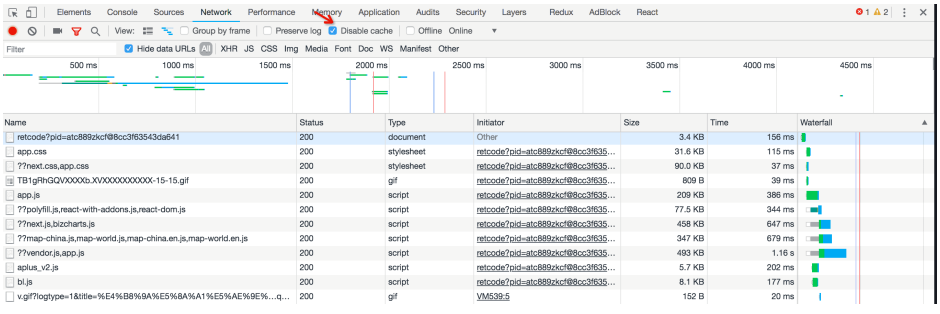


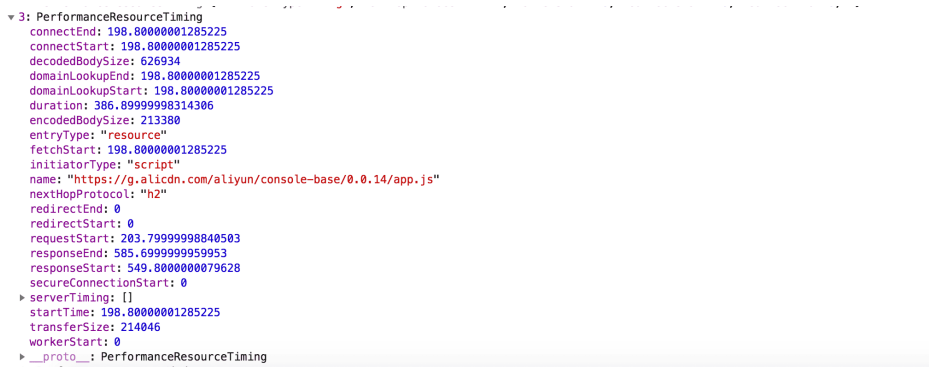


```
4: PerformanceResourceTiming
  connectEnd: 192.9999999993015
  connectStart: 192.9999999993015
  decodedBodySize: 626934
  domainLookupEnd: 192.9999999993015
  domainLookupStart: 192.9999999993015
  duration: 141.70000000740401
  encodedBodySize: 214180
  entryType: "resource"
  fetchStart: 192.9999999993015
  initiatorType: "script"
  name: "https://g.alicdn.com/aliyun/console-base/0.0.14/app.js"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  requestStart: 195.9000000206288
  responseEnd: 334.7000000067055
  responseStart: 198.00000000395812
  secureConnectionStart: 0
  serverTiming: []
  startTime: 192.9999999993015
  transferSize: 0
  workerStart: 0
  __proto__: PerformanceResourceTiming
```


# 解决方法

启用 **Disable cache** 选项后，transferSize 内容正常：






# 常见问题 2：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，部分情况下看到的时间为 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** 打开控制台，在 **Network** 标签页上禁用缓存（**Disable cache**）后，刷新页面就可以看到经过网络过程所耗的时间了。

# 常见问题 3：跨域问题

查看 API 返回的数据时，如果发现很多时间数据返回为 0，是因为受同源策略的影响，跨域资源获取到的时间点会为 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：*。

# 常见问题 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。

## 常见问题 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 耗时略多几毫秒，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统计从 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您排查性能瓶颈。

## 常见问题 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。通过该时间轴，您可以看到页面加载过程中 API 请求发起的时间点和耗时。

# JS 错误统计

本文介绍了 ARMS 前端监控的 JS 错误统计功能。

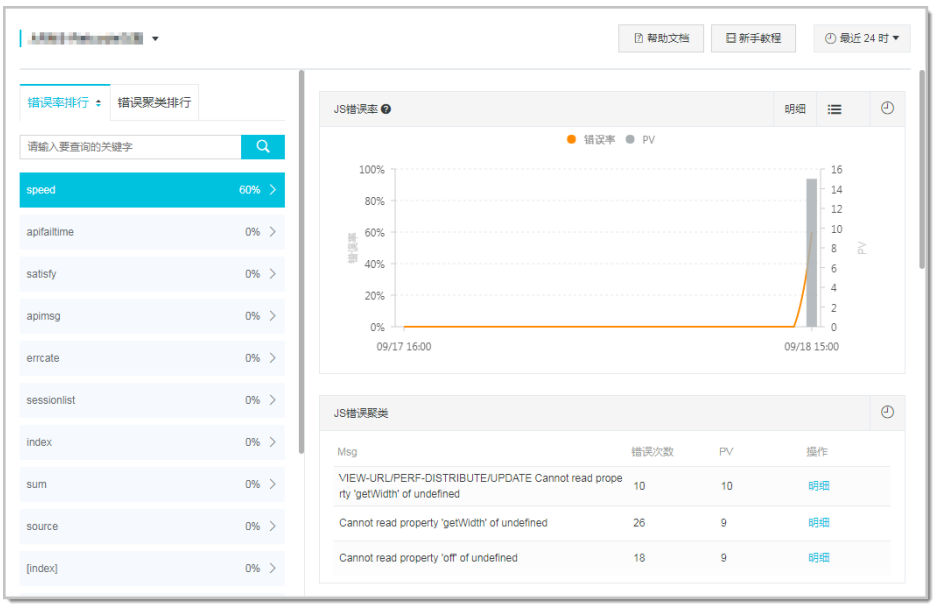
## JS 错误率的定义是什么？

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。



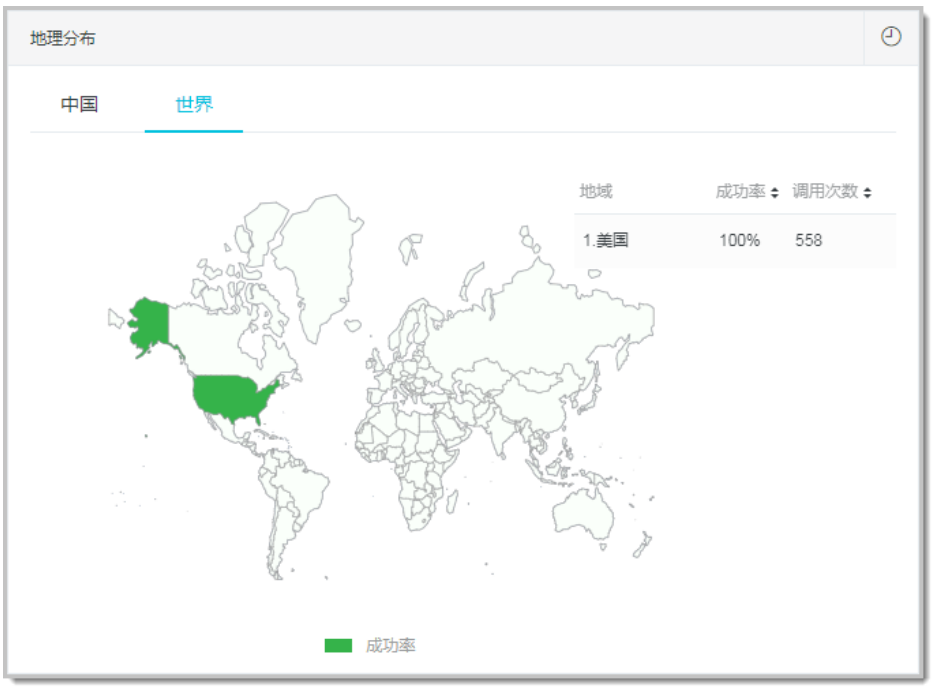

**注意：** 由于错误率排行榜仅会展示错误率最高或最低的前 100 个页面，当站点的页面总数超过 200 个时，错误率不属于这两个区间的页面始终不会显示在排行中。例如，假设站点共有 220 个页面，那么无论选择按错误率升序还是降序排列，都会有 20 个页面不会显示在排行中。

# 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了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。

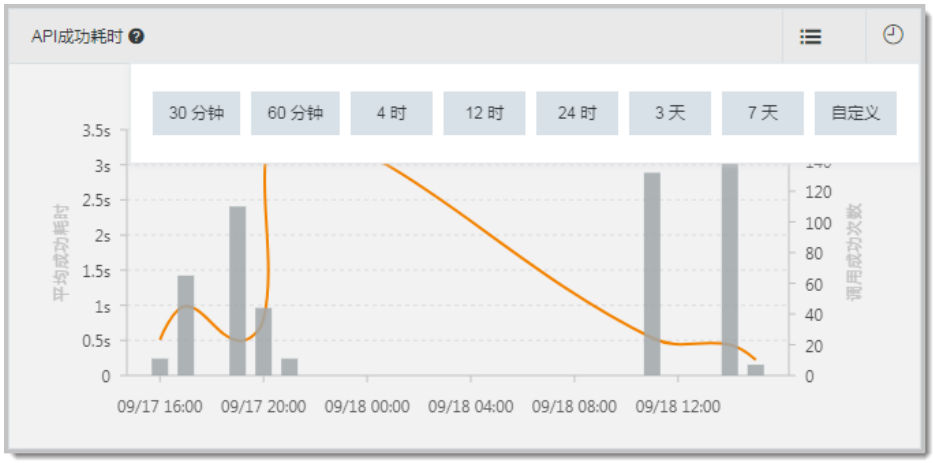
# 通用操作

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。



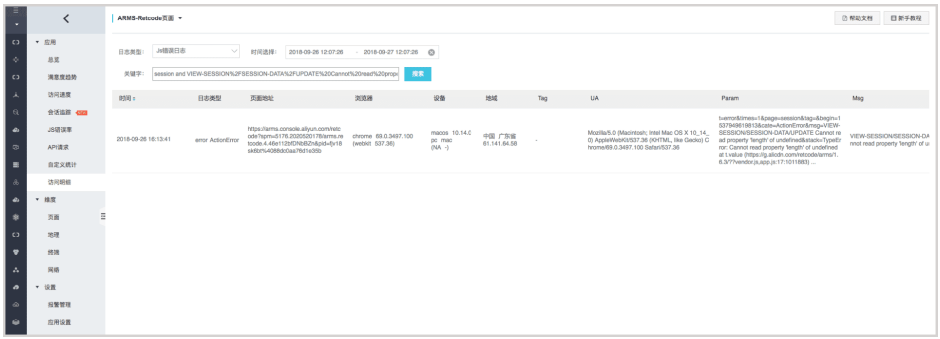

# 如何进行 JS 错误排查？

您可通过以下操作进入 **JS 错误详情**页面。

单击左侧导航栏中的 JS 错误率标签页。

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。



**注意：** 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

**注意：** 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

**注意：** 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。

在左侧导航栏中选择**设置** > **报警设置**。

在**设置报警规则**区域单击**新建规则**。

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。

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*/
```

**注意：** 第三个参数一定要设为 true。

当 DOM Rready、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 Script Error 的产生原因和解决方法。

## 相关文档

- Script Error 的产生原因和解决方法

# 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：

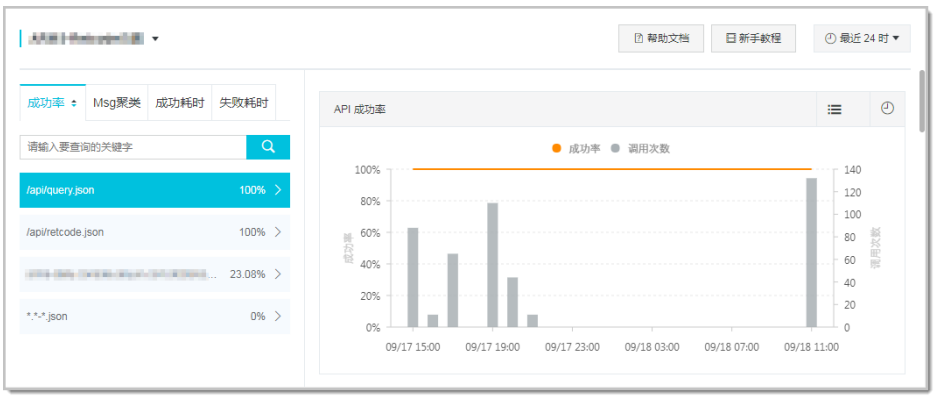
- 每个 API 的成功率
- API 返回信息
- API 接口的调用成功平均耗时
- API 接口的调用失败平均耗时

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

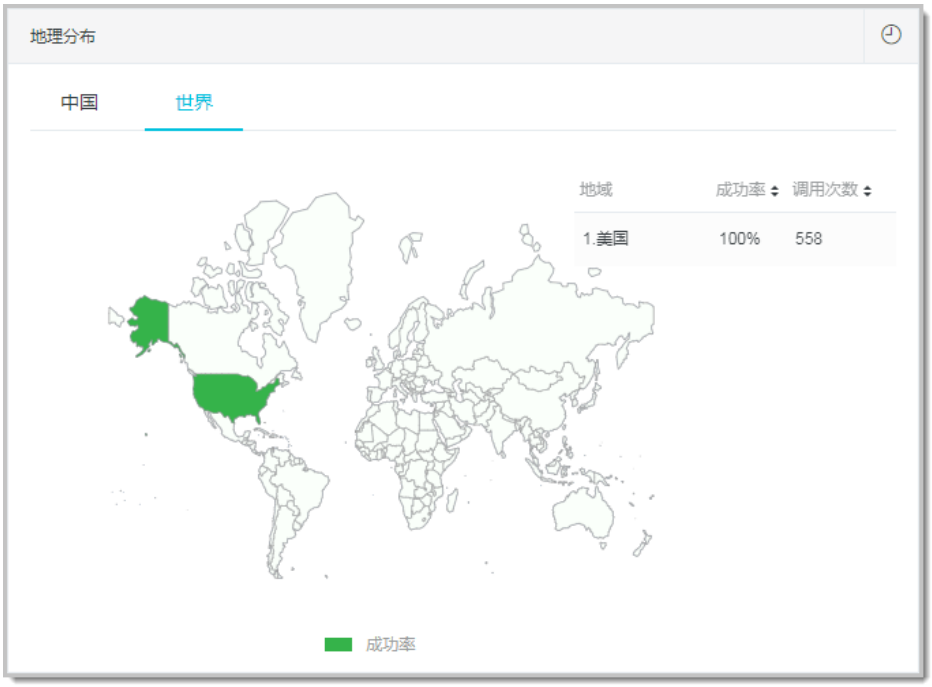
左侧的成功耗时标签页展示的是 API 调用成功时的平均耗时。右侧的 API 成功耗时区域展示的是左侧列表选中 API 的调用成功平均耗时曲线和调用成功次数柱形图。

# 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。

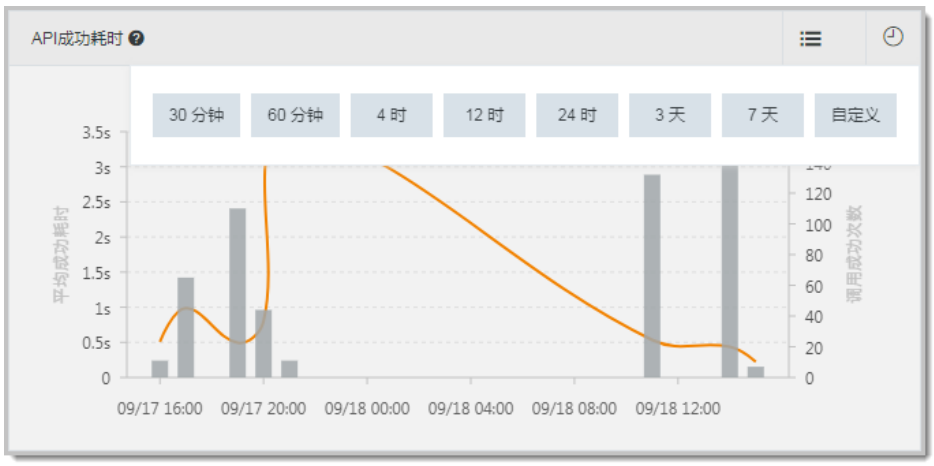
# 通用操作

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。




# 自定义统计

本文介绍了前端监控中的自定义统计功能。

## 功能介绍

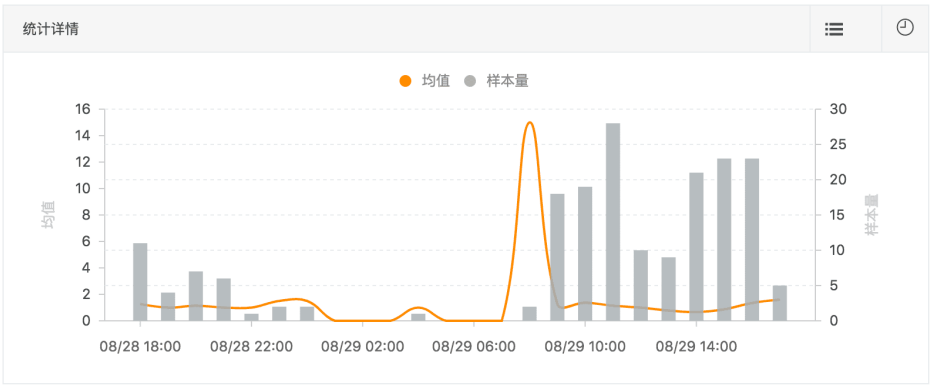
为帮助您监控和统计轻量级的业务交互行为，ARMS 前端监控提供了以下两类自定义统计功能：

- 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）：

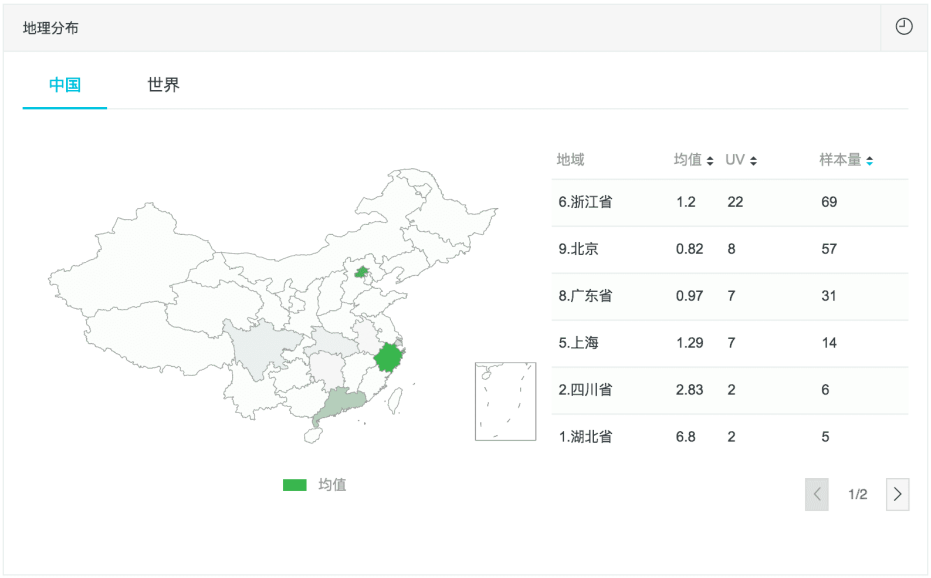
统计详情

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。

# 使用方法

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 进行求和与求均值的统计。

## 求和统计 API

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

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 | 否 | 0 |

| | | | | |
|---|---|---|---|---|
| | | 认 0 | | |

示例：

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

# 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。

## 前提条件

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?", enableLinkTrace: 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

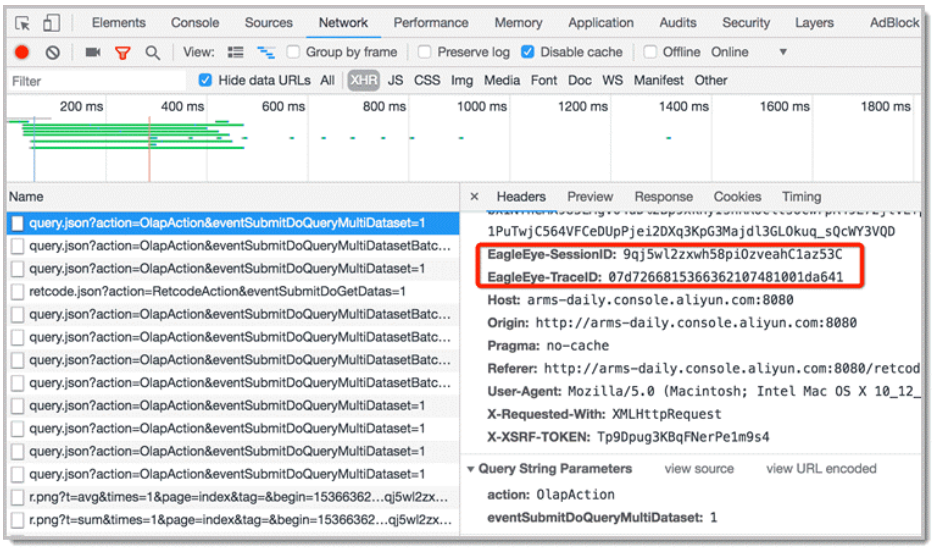
- 在允许 API 自动上报的前提下，如果 API 与当前应用的域名同源，则会在 API 请求头（Request Header）加入两个自定义 Header：EagleEye-TraceID 和 EagleEye-SessionID。EagleEye-TraceID 即串联前后端链路的标识。

- 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）。

  CAUTION:

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



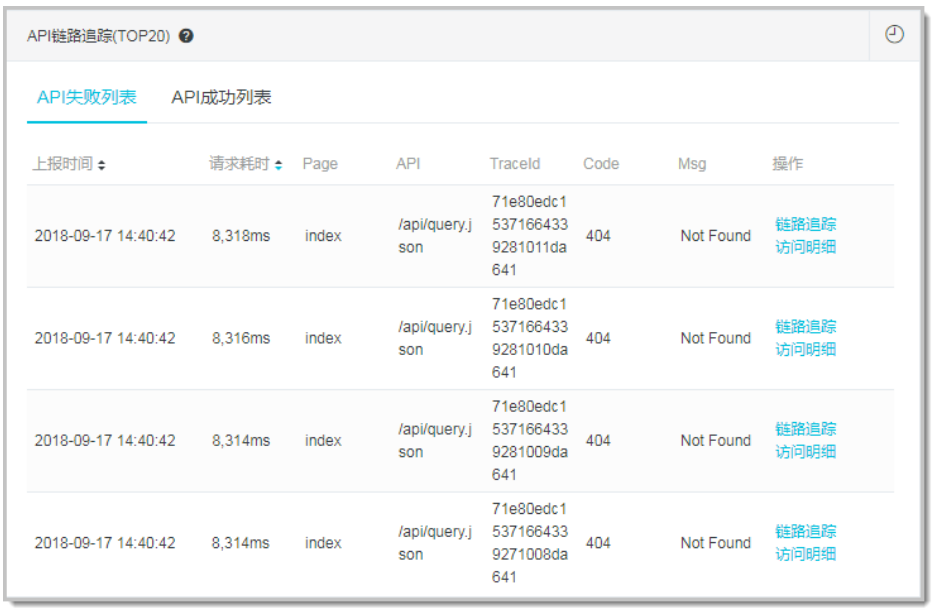

阿里云 ARMS 应用监控

关于应用监控的配置，请参考应用监控概述。

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。

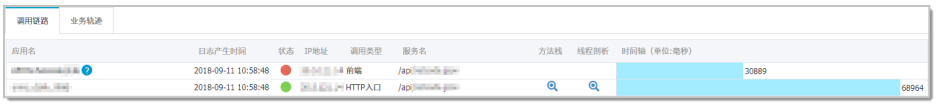
# 使用场景和案例

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。

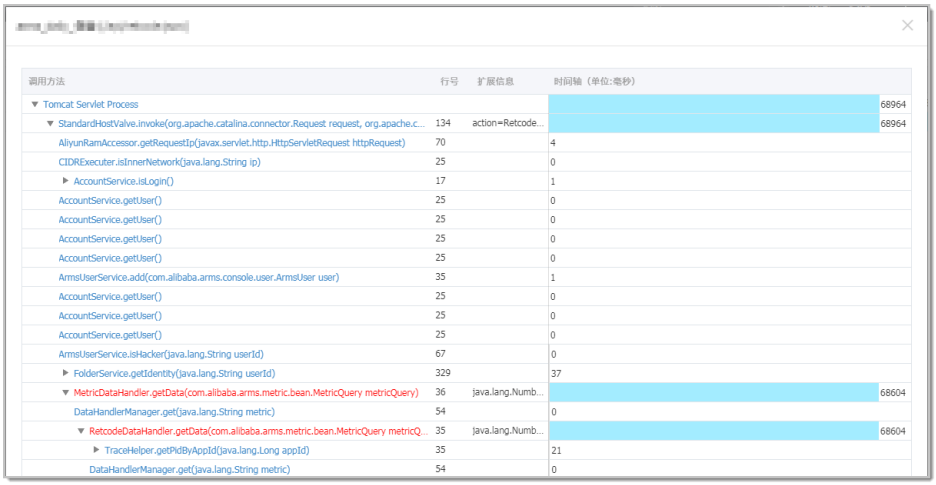
当 API 返回错误码或者业务逻辑错误时定位问题

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。




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。

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

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。

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。

# 实时监控大屏

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

请查看以下交互式文档，了解 ARMS 前端监控的实时大屏功能。为获得最佳体验，请在新窗口中打开查看。

浏览器不支持 iframe，请在新窗口中查看。

# 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。

**提示：** 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。

## 参考

- 页面访问速度
- JS 错误统计
- API 请求监控
- 快速创建 ARMS 报警

# 高级选项

# SDK 配置项

本文介绍了 ARMS 前端监控 SDK 的初始化配置项，说明了如何通过设置不同的参数来应对不同应用场景。

## 使用方式

在向页面埋入 BI 探针时，可以向 config 中添加额外参数。或者在页面初始化完成之后，在前端 JS 代码中调用 setConfig方法（参见 API 使用指南）来修改。

## 示例

```
<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>
```

以上代码的 config 中除了默认的pid参数外，还添加了enableSPA参数，用于单页面应用（Single Page Application）的场景。

# 配置项参数

| 参数名 | 类型 | 描述 | 是否必须 | 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，默认取当前页面 URL 的关键部分 | 否 | host + pathname |
| enableSPA | Boolean | 是否监听页面的 hashchange 事件并重新上报 PV，适用于单页面应用场景 | 否 | false |
| parseHash | Function | 配合 enableSPA 使用，详情见下文 | 否 | 见下文 |
| disableHook | Boolean | 是否禁用 AJAX 请求监听，默认会监听并用于 API 调用成功率上报 | 否 | false |
| ignoreUrlCase | Boolean | 是否忽略 Page URL 大小写，默认为忽略 | 否 | true |
| ignoreUrlPath | * | URL 过滤规则，详情见下文 | 否 | 见下文 |
| ignoreApiPath | * | API 过滤规则，详情见下文 | 否 | 见下文 |
| disabled | Boolean | 禁用日志上报功能 | 否 | false |
| parseResponse | Function | API 自动上报时返回值的解析方法 | 否 | 见下文 |
| 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。详情见下文 | 否 | 见下文 |
| sendResource | Boolean | 是否上报页面静态资源, 详情见下文 | 否 | false |

# 部分设置项详细说明

## 1. parseHash 将 URL hash 解析为 page 的方法

单页面应用场景中（参见进阶场景介绍），在enableSPA设为true的前提下，页面触发 hashchange 事件时，此参数用于将 URL hash 解析为 page 字段的方法。

默认值是一个简单的字符串处理方法：

```
function (hash) {
var page = hash ? hash.replace(/^#\/?/, '').replace(/^(https?:)?\/\//, '').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。

**示例：**

```
// 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onload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';
}
});
});
```

## 2. ignoreUrlPath URL 过滤规则

在页面 URL 类似于http://xxx.com/projects/123456的场景中（projects 后面紧跟的是项目 ID），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，所以需要过滤掉这些非关键字符。

**生效场景**

此设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（参见 API 使用指南）修改过 page，或者enableSPA已设为true，则此设置项无效。

默认值是一个数组，**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类似于xxxx/123456后面的数字，例如xxxx/00001和xxxx/00002都会变成xxxx/**。

ignoreUrlPath 的值可以是以下多种类型，用法分别为：

- String或RegExp：将匹配到的字符串去掉。关于RegExp请参考正则表达式的语法；
- Object<rule, target>：对象包含两个 key，分别是 rule 和 target，作为 JS 字符串的replace方法的入参，请参考 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，每条子规则都可以是上述类型之一。

## 3. config.ignoreApiPath API 过滤规则

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

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。

## 4. config.parseResponse Response 解析方法

该方法用于解析自动上报 API 时返回的数据。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。

## 5. config.sample 抽样上报配置

该配置通过随机抽样上报来减小用户上报量并降低负载。

该配置会对**性能**和**成功 API** 日志随机抽样上报，后台日志处理会根据对应的抽样配置进行还原，**不会影响 JS Error 和失败 API 等其他类型日志的上报。**

抽样值 sample 默认为 1，可选 1、10 或 100，对应的抽样率为 1/sample ，即 1 表示 100% 采样，10 表示 10% 采样，100 表示 1% 采样。

由于随机抽样会与实际监控结果有一定误差，建议日均 PV 在 100 万以上的站点采用抽样。

**注意**：如果用户原上报量较小，该配置可能会造成较大的统计结果误差。

### 6. sendResource 上报页面加载的静态资源

在页面load时会根据该项配置判断是否要上报页面加载的静态资源，所以请直接使用在config配置sendResource的方式。而setConfig的方式可能在页面load后才会触发，从而配置的sendResource会无效。使用方式如下：

```
<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>
```

sendResource配置为true后，在页面onload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，点击查看本次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。

# 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。

## 日志上报接口

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 SDK 配置项。

**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 SDK 配置项。

**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，更多信息请参考进阶场景介绍。

**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);
```

# 进阶场景

本文介绍了一些更复杂场景中的 SDK 用法。

## 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聚类到对应的子页面。

SDK 提供了 SPA 页面的两种处理方式：

### 1. 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相配套的还有<Function>parseHash（参见 SDK 配置项）。

### 2. 完全手动上报

此方法可用于所有的单页面应用场景。如果第一种方法无法满足，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。

**示例**

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

## 数据预上报

场景一：在页面刚刚加载时，有一些数据需要上报，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）。

场景二：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，就会将预先挂载到 window.__bl.pipe 上的方法及参数依次调用。

**注意：**在 SDK 初始化完成前，如果多次设置 __bl.pipe 的值，只会以最后一次为准。

pipe也可以用于 SDK 初始化完成后调用（支持 IE9 及以上）。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，则可以使用此方式。

**示例：**

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'];
```

# 参考信息

# 统计指标说明

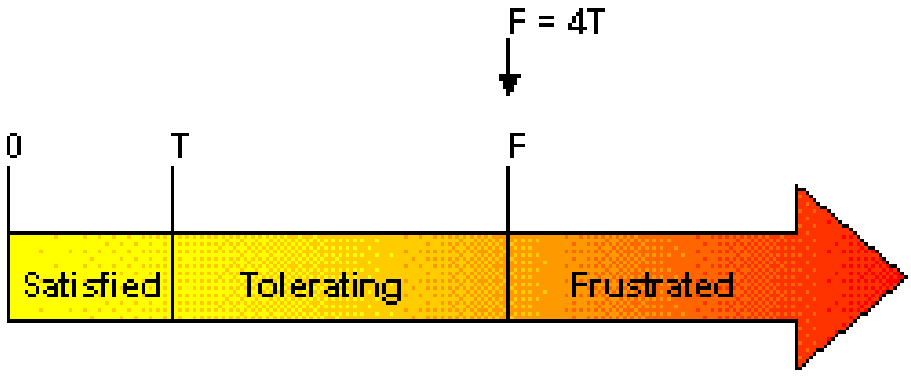
# 满意度

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**。

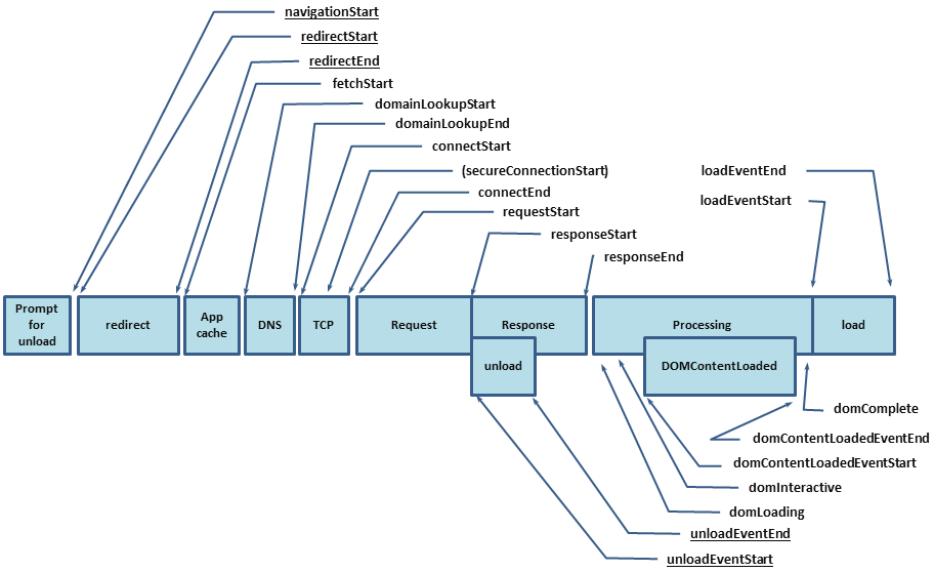
错误率 = 错误样本量 / 总样本量

页面异常除了自动上报的 JS Error 外，也包括手动调用 error 方法上报的错误。

# 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

**阶段耗时**

| 上报字段 | 描述 | 计算方式 | 备注 |
| --- | --- | --- | --- |
| dns | DNS 解析耗时 | domainLookupEnd - domainLookupStart | |
| tcp | TCP 连接耗时 | connectEnd - connectStart | |
| 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 | |
| dom | DOM 解析耗时 | domInteractive - responseEnd | |
| res | 资源加载耗时 | loadEventStart - domContentLoadedEventEnd | 表示页面中的同步加载资源 |

**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 | |

| | | | |
|---|---|---|---|
| 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 API 方法上报的数据。

# 统计 PV、UV 的方法

ARMS 采用业界标准方法进行 PV 和 UV 的统计，具体如下：

- 统计 PV：每次页面触发 1 次 load 事件计 1 次 PV。
- 统计 UV：每当有新的用户访问应用时，我们会在 cookie 中计入 1 个 UID，该 UID 有效期为 6 个月，有效期内该 UID 的所有访问记为 1 次 UV。