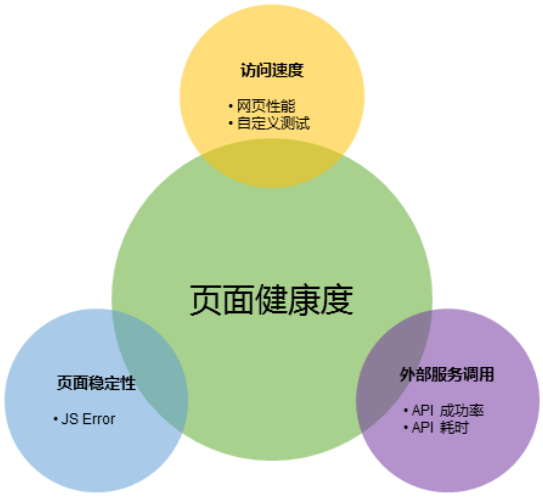
业务实时监控服务 ARMS

前端监控

前端监控

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。

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。

在左侧菜单栏中选择**前端监控**。

单击**新建应用站点**，在弹出的对话框中填写站点名称后单击**确认**。

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。

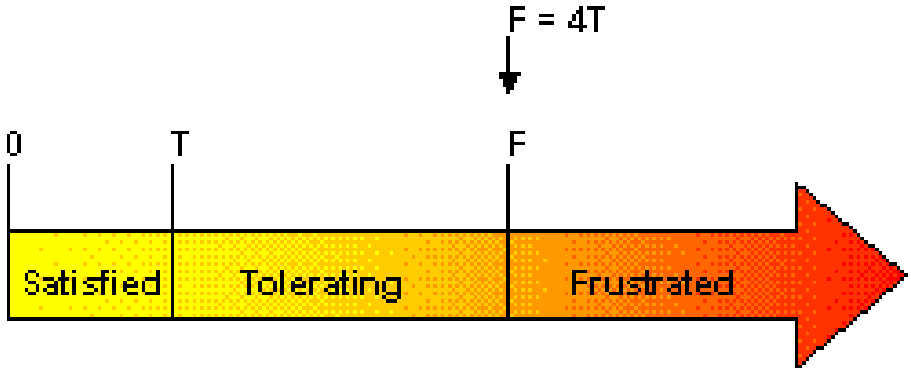
满意度

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**。

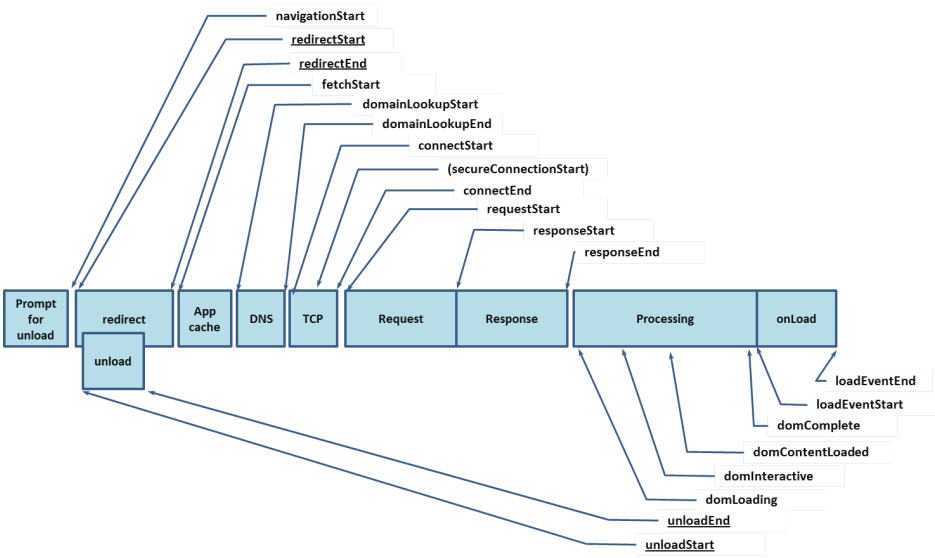
错误率 = 错误样本量 / 总样本量

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 - domLoading	
res	资源加载耗时	loadEventStart - domContentLoaded EventEnd	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	domLoading - navigationStart	如果浏览器支持，则会取 chrome.loadTimes().firstPaintTime 计算
tti	Time to Interact，首次可交互时间	domInteractive - navigationStart	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 - navigationStart	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 - navigationStart	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。

高级选项

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
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
autoSendPv	Boolean	是否初始化后自动发送 PV，默认会自动发送	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

部分设置项详细说明

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(/\?.*$/, '') : '';
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';
}
});
});
```

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**：

```
[
{rule: /(\w+)\/\d{4,}/g, target: '$1'},
{rule: /([^\/]\/[^\/]+)(\/[^\/]+)\/.*$/, target: '$1$2/'}
]
```

此设置项的默认值会过滤掉类似xxxx/123456后面的数字，以及超过 3 层/的 URL path。例如xxx.com/aaa/bbb/ccc.html只会取xxx.com/aaa/bbb。

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。

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{4,}/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。

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);
```

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);
});
```

对于页面初始化完成后的第一次 PV 上报，如果也希望手动控制，则可在配置项中设置autoSendPv为false，然后在应用初始化完成后调用setPage。

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'];
```