link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=229 sentences=527, words=3626

1.为什么要做前端监控

- 更快发现问题和解决问题
- 做产品的决策依据
- 提升前端工程师的技术深度和广度,打造简历亮点
- 为业务扩展提供了更多可能性

2.前端监控目标

2.1 稳定性(stability)

错误名称 备注 JS错误 JS执行错误或者promise异常 资源异常 script、link等资源加载异常 接口错误 ajax或fetch请求接口异常 白屏 页面空白

2.2 用户体验(experience)

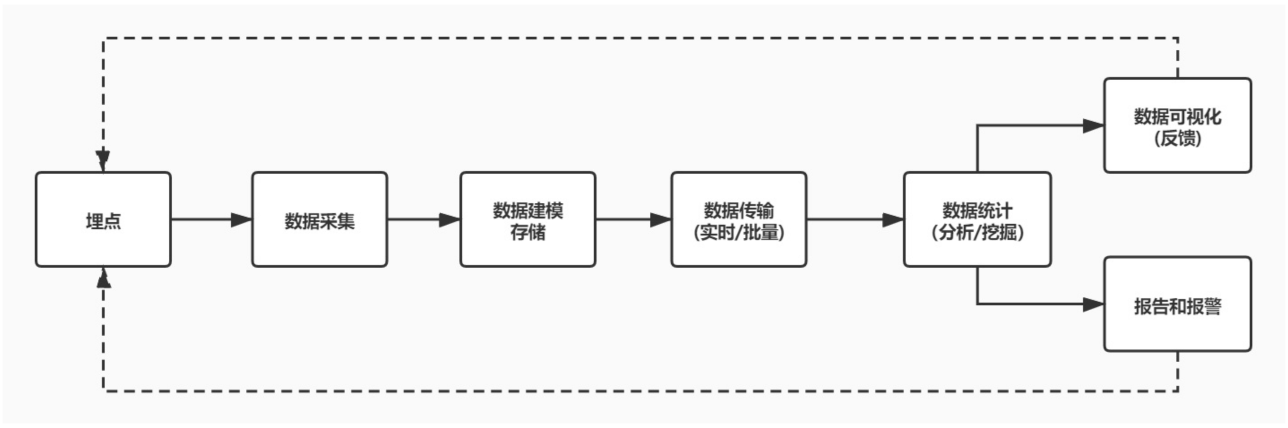
错误名称 备注 加载时间 各个阶段的加载时间 TTFB(time to first byte)(首字节时间) 是指浏览器发起第一个请求到数据返回第一个字节所消耗的时间，这个时间包含了网络请求时间、后端处理时间 FP(First Paint)(首次绘制) 首次绘制包括了任何用户自定义的背景绘制，它是将第一个像素点绘制到屏幕的时刻 FCP(First Content Paint)(首次内容绘制) 首次内容绘制是浏览器将第一个DOM渲染到屏幕的时间,可以是任何文本、图像、SVG等的时间 FMP(First Meaningful paint)(首次有意义绘制) 首次有意义绘制是页面可用性的量度标准 FID(First Input Delay)(首次输入延迟) 用户首次和页面交互到页面响应交互的时间 卡顿 超过50ms的长任务

2.3 业务(business)

错误名称 备注 PV page view 即页面浏览量或点击量 UV 指访问某个站点的不同IP地址的人数 页面的停留时间 用户在每一个页面的停留时间

3.前端监控流程

- 前端埋点
- 数据上报
- 分析和计算 将采集到的数据进行加工汇总
- 可视化展示 将数据按各种维度进行展示
- 监控报警 发现问题后按一定的条件触发报警



3.1 常见的埋点方案

3.1.1 代码埋点

- 代码埋点，就是以嵌入代码的形式进行埋点,比如需要监控用户的点击事件,会选择在用户点击时,插入一段代码，保存这个监听行为或者直接将监听行为以某一种数据格式直接传递给服务器端
- 优点是可以在任意时刻，精确的发送或保存所需要的数据信息
- 缺点是工作量较大

3.1.2 可视化埋点

- 通过可视化交互的手段，代替代码埋点
- 将业务代码和埋点代码分离，提供一个可视化交互的页面，输入为业务代码，通过这个可视化系统，可以在业务代码中自定义的增加埋点事件等等,最后输出的代码耦合了业务代码和埋点代码
- 可视化埋点其实是用系统来代替手工插入埋点代码

3.1.3 无痕埋点

- 前端的任意一个事件都被绑定一个标识，所有的事件都别记录下来
- 通过定期上传记录文件,配合文件解析，解析出来我们想要的数据，并生成可视化报告供专业人员分析
- 无痕埋点的优点是采集全量数据,不会出现漏埋和误埋等现象
- 缺点是给数据传输和服务器增加压力，也无法灵活定制数据结构

4.编写监控采集脚本

4.1 开通日志服务

- 日志服务(Log Service,简称 SLS) (https://sls.console.aliyun.com/lognext/profile)是针对日志类数据一站式服务，用户无需开发就能快捷完成数据采集、消费、投递以及查询分析等功能，帮助提升运维、运营效率，建立 DT 时代海量日志处理能力
- 日志服务帮助文档 (https://help.aliyun.com/product/28958.html)
- Web Tracking (https://help.aliyun.com/document_detail/31752.html)

4.2 监控错误

4.2.1 错误分类

- JS错误
  - JS错误
  - Promise异常

- 资源异常
  - 监听error

4.2.2 数据结构设计

1. jsError

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590815288710",
  "userAgent": "Chrome",
  "kind": "stability",
  "type": "error",
  "errorType": "jsError",
  "message": "Uncaught TypeError: Cannot set property 'error' of undefined",
  "filename": "http://localhost:8080/",
  "position": "0:0",
  "stack": "btnClick (http://localhost:8080/:20:39)^HTMLInputElement.onclick (http://localhost:8080/:14:72)",
  "selector": "HTML BODY #container .content INPUT"
}
```

2. promiseError

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590815290600",
  "userAgent": "Chrome",
  "kind": "stability",
  "type": "error",
  "errorType": "promiseError",
  "message": "someVar is not defined",
  "filename": "http://localhost:8080/",
  "position": "24:29",
  "stack": "http://localhost:8080/:24:29^new Promise ()^btnPromiseClick (http://localhost:8080/:23:13)^HTMLInputElement.onclick (http://localhost:8080/:15:86)",
  "selector": "HTML BODY #container .content INPUT"
}
```

3. resourceError

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590816168643",
  "userAgent": "Chrome",
  "kind": "stability",
  "type": "error",
  "errorType": "resourceError",
  "filename": "http://localhost:8080/error.js",
  "tagName": "SCRIPT",
  "timeStamp": "76",
  "selector": "HTML BODY SCRIPT"
}
```

4.2.3 报表

- 查询语句 (https://help.aliyun.com/document_detail/29060.html?spm=5176.2020520112.0.0.636c34c07HzVho)

```
* | SELECT kind,count(*) as number GROUP BY kind
```

4.2.4 实现

1. webpack.config.js

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    mode: 'development',
    context: process.cwd(),
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'monitor.js'
    },
    devServer: {
        contentBase: path.resolve(__dirname, 'dist')
    },
    module: {},
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            inject: 'head'
        })
    ]
}
```

2. index.html

src\index.html

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>monitortitle>
head>

<body>
    <div id="container">
        <div class="content">
            <input type="button" value="点击抛出错误" onclick="btnClick()" />
            <input type="button" value="点击抛出promise错误" onclick="btnPromiseClick()" />
        div>
    div>

    <script>
        function btnClick() {
            window.someVariable.error = 'someVariable';
        }
        function btnPromiseClick() {
            new Promise(function (resolve, reject) {
                console.log(someVar.some);
            });
        }
    script>
    [illegible]
[illegible]
[illegible]
```

3. src\index.js

src\index.js

```
import './monitor'
```

4. monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
injectJsError();
```

5. jsError.js

src\monitor\lib\jsError.js

```
import tracker from '../util/tracker';
import getLastEvent from '../util/getLastEvent';
import getSelector from '../util/getSelector';
import formatTime from '../util/formatTime';
export function injectJsError() {

    window.addEventListener('error', function (event) {
        let lastEvent = getLastEvent();
        if (event.target && (event.target.src || event.target.href)) {
            tracker.send({
                kind: 'stability',
                type: 'error',
                errorType: 'resourceError',
                filename: event.target.src || event.target.href,
                tagName: event.target.tagName,
                timeStamp: formatTime(event.timeStamp),
                selector: getSelector(event.path || event.target),
            })
        } else {
            tracker.send({
                kind: 'stability',
                type: 'error',
                errorType: 'jsError',
                message: event.message,
                filename: event.filename,
                position: (event.lineNo || 0) + ":" + (event.columnNo || 0),
                stack: getLines(event.error.stack),
                selector: lastEvent ? getSelector(lastEvent.path || lastEvent.target) : ''
            })
        }
    }, true);

    window.addEventListener('unhandledrejection', function (event) {
        let lastEvent = getLastEvent();
        let message = '';
        let line = 0;
        let column = 0;
        let file = '';
        let stack = '';
        if (typeof event.reason === 'string') {
            message = event.reason;
        } else if (typeof event.reason === 'object') {
            message = event.reason.message;
        }
        let reason = event.reason;
        if (typeof reason === 'object') {
            if (reason.stack) {
                var matchResult = reason.stack.match(/at\s+(.+):(\d+):(\d+)/);
                if (matchResult) {
                    file = matchResult[1];
                    line = matchResult[2];
                    column = matchResult[3];
                }
                stack = getLines(reason.stack);
            }
        }
        tracker.send({
            kind: 'stability',
            type: 'error',
            errorType: 'promiseError',
            message: message,
            filename: file,
            position: line + ':' + column,
            stack,
            selector: lastEvent ? getSelector(lastEvent.path || lastEvent.target) : ''
        })
    }, true);
}
function getLines(stack) {
    if (!stack) {
        return '';
    }
    return stack.split('\n').slice(1).map(item => item.replace(/^\s+at\s+/g, '')).join('^');
}
```

6. formatTime.js

src\monitor\util\formatTime.js

```
export default (time) => {
    return `${time}`.split(".")[0]
}
```

7. getLastEvent.js

src\monitor\util\getLastEvent.js

```
let lastEvent;
['click','pointerdown', 'touchstart', 'mousedown', 'keydown', 'mouseover'].forEach(event => {
    document.addEventListener(event, (event) => {
        lastEvent = event;
    }, {
        capture: true,
        passive: true
    });
});
export default function () {
    return lastEvent;
};
```

8. getSelector.js

src\monitor\util\getSelector.js

```
const getSelector = function (path) {
    return path.reverse().filter(function (element) {
        return element !== window && element !== document;
    }).map(function (element) {
        var selector;
        if (element.id) {
            selector = `#${element.id}`;
        } else if (element.className && typeof element.className === 'string') {
            selector = '.' + element.className.split(' ').filter(function (item) { return !!item }).join('.');
        } else {
            selector = element.nodeName;
        }
        return selector;
    }).join(' ');
}
export default function (pathsOrTarget) {
    if (Array.isArray(pathsOrTarget)) {
        return getSelector(pathsOrTarget);
    } else {
        var paths = [];
        var element = pathsOrTarget;
        while (element) {
            paths.push(element);
            element = element.parentNode;
        }
        return getSelector(paths);
    }
}
```

9. tracker.js

src\monitor\util\tracker.js

- PutWebtracking (https://help.aliyun.com/document_detail/120218.html)

```
let host = 'cn-beijing.log.aliyuncs.com';
let project = 'zhufengmonitor';
let logstore = 'zhufengmonitor-store';
var userAgent = require('user-agent')
function getExtraData() {
    return {
        title: document.title,
        url: location.href,
        timestamp: Date.now(),
        userAgent: userAgent.parse(navigator.userAgent).name
    };
}

class SendTracker {
    constructor() {
        this.url = `http://${project}.${host}/logstores/${logstore}/track`;
        this.xhr = new XMLHttpRequest();
    }
    send(data = {}, callback) {
        let extraData = getExtraData();
        let logs = { ...extraData, ...data };
        for (let key in logs) {
            if (typeof logs[key] === 'number') {
                logs[key] = "" + logs[key];
            }
        }
        console.log(logs);
        console.log(JSON.stringify(logs, null, 2));
        let body = JSON.stringify({
            __logs__: [logs]
        });
        this.xhr.open("POST", this.url, true);
        this.xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
        this.xhr.setRequestHeader('x-log-apiversion', '0.6.0');
        this.xhr.setRequestHeader('x-log-bodyrawsize', body.length);
        this.xhr.onload = function () {
            if ((this.status >= 200 && this.status 300) || this.status == 304) {
                callback && callback();
            }
        }
        this.xhr.onerror = function (error) {
            console.log('error', error);
        }
        this.xhr.send(body);
    }
}

export default new SendTracker();
```

4.3.接口异常采集脚本

4.3.1 数据设计

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590817024490",
  "userAgent": "Chrome",
  "kind": "stability",
  "type": "xhr",
  "eventType": "load",
  "pathname": "/success",
  "status": "200-OK",
  "duration": "7",
  "response": "{\"id\":1}",
  "params": ""
}
```

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590817025617",
  "userAgent": "Chrome",
  "kind": "stability",
  "type": "xhr",
  "eventType": "load",
  "pathname": "/error",
  "status": "500-Internal Server Error",
  "duration": "7",
  "response": "",
  "params": "name=zhufeng"
}
```

4.3.2 实现

1. src\index.html

src\index.html

```

    monitor

+
+


<span class="hljs-addition">+        function sendAjaxSuccess() {</span>
<span class="hljs-addition">+            let xhr = new XMLHttpRequest;</span>
<span class="hljs-addition">+            xhr.open('GET', '/success', true);</span>
<span class="hljs-addition">+            xhr.responseType = 'json';</span>
<span class="hljs-addition">+            xhr.onload = function () {</span>
<span class="hljs-addition">+                console.log(xhr.response);</span>
<span class="hljs-addition">+            }</span>
<span class="hljs-addition">+            xhr.send();</span>
<span class="hljs-addition">+        }</span>
<span class="hljs-addition">+        function sendAjaxError() {</span>
<span class="hljs-addition">+            let xhr = new XMLHttpRequest;</span>
<span class="hljs-addition">+            xhr.open('POST', '/error', true);</span>
<span class="hljs-addition">+            xhr.responseType = 'json';</span>
<span class="hljs-addition">+            xhr.onload = function () {</span>
<span class="hljs-addition">+                console.log(xhr.response);</span>
<span class="hljs-addition">+            }</span>
<span class="hljs-addition">+            xhr.onerror = function (error) {</span>
<span class="hljs-addition">+                console.log(error);</span>
<span class="hljs-addition">+            }</span>
<span class="hljs-addition">+            xhr.send("name=zhufeng");</span>
        }

```

2. monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
+import { injectXHR } from './lib/xhr';
injectJsError();
+injectXHR();
```

3. webpack.config.js

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    mode: 'development',
    context: process.cwd(),
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'monitor.js'
    },
    devServer: {
        contentBase: path.resolve(__dirname, 'dist'),
+        before(router) {
+            router.get('/success', function (req, res) {
+                res.json({ id: 1 });
+            });
+            router.post('/error', function (req, res) {
+                res.sendStatus(500);
+            });
+        }
    },
    module: {},
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            inject: 'head'
        })
    ],

}
```

4. xhr.js

src\monitor\lib\xhr.js

```
import tracker from '../util/tracker';
export function injectXHR() {
    let XMLHttpRequest = window.XMLHttpRequest;
    let oldOpen = XMLHttpRequest.prototype.open;
    XMLHttpRequest.prototype.open = function (method, url, async, username, password) {
        if (!url.match(/logstores/) && !url.match(/sockjs/)) {
            this.logData = {
                method, url, async, username, password
            }
        }
        return oldOpen.apply(this, arguments);
    }
    let oldSend = XMLHttpRequest.prototype.send;
    let start;
    XMLHttpRequest.prototype.send = function (body) {
        if (this.logData) {
            start = Date.now();
            let handler = (type) => (event) => {
                let duration = Date.now() - start;
                let status = this.status;
                let statusText = this.statusText;
                tracker.send({
                    kind: 'stability',
                    type: 'xhr',
                    eventType: type,
                    pathname: this.logData.url,
                    status: status + "-" + statusText,
                    duration: "" + duration,
                    response: this.response ? JSON.stringify(this.response) : "",
                    params: body || ''
                })
            }
            this.addEventListener('load', handler('load'), false);
            this.addEventListener('error', handler('error'), false);
            this.addEventListener('abort', handler('abort'), false);
        }
        oldSend.apply(this, arguments);
    };
}
```

4.4 白屏

- 白屏就是页面上什么都没有

4.4.1 数据设计

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590822618759",
  "userAgent": "chrome",
  "kind": "stability",
  "type": "blank",
  "emptyPoints": "0",
  "screen": "2049x1152",
  "viewPoint": "2048x994",
  "selector": "HTML BODY #container"
}
```

4.4.2 实现

- screen (https://developer.mozilla.org/zh-CN/docs/Web/API/Window/screen) 返回当前window的screen对象,返回当前渲染窗口中和屏幕有关的属性
- innerWidth (https://developer.mozilla.org/zh-CN/docs/Web/API/Window/innerWidth) 只读的 Window 属性 innerWidth 返回以像素为单位的窗口的内部宽度
- innerHeight (https://developer.mozilla.org/zh-CN/docs/Web/API/Window/innerHeight) 窗口的内部高度(布局视口)的高度
- layout_viewport (https://developer.mozilla.org/en-US/docs/Glossary/layout_viewport)
- elementsFromPoint (https://developer.mozilla.org/zh-CN/docs/Web/API/Document/elementsFromPoint)方法可以获取到当前视口内指定坐标处，由里到外排列的所有元素

1. src\index.html

```

    monitor

<span class="hljs-addition">+        let content = document.getElementsByClassName('content')[0];</span>
<span class="hljs-addition">+        content.innerHTML = '@'.repeat(10000);</span>
```

2. monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
import { injectXHR } from './lib/xhr';
+import { blankScreen } from './lib/blankScreen';
injectJsError();
injectXHR();
+blankScreen();
```

3. onload.js

src\monitor\util\onload.js

```
export default function (callback) {
    if (document.readyState === 'complete') {
        callback();
    } else {
        window.addEventListener('load', callback);
    }
};
```

4. blankScreen.js

src\monitor\lib\blankScreen.js

```
import tracker from '../util/tracker';
import onload from '../util/onload';
function getSelector(element) {
    var selector;
    if (element.id) {
        selector = `#${element.id}`;
    } else if (element.className && typeof element.className === 'string') {
        selector = '.' + element.className.split(' ').filter(function (item) { return !!item }).join('.');
    } else {
        selector = element.nodeName.toLowerCase();
    }
    return selector;
}
export function blankScreen() {
    const wrapperSelectors = ['body', 'html', '#container', '.content'];
    let emptyPoints = 0;
    function isWrapper(element) {
        let selector = getSelector(element);
        if (wrapperSelectors.indexOf(selector) >= 0) {
            emptyPoints++;
        }
    }
    onload(function () {
        let xElements, yElements;
        debugger
        for (let i = 1; i 9; i++) {
            xElements = document.elementsFromPoint(window.innerWidth * i / 10, window.innerHeight / 2)
            yElements = document.elementsFromPoint(window.innerWidth / 2, window.innerHeight * i / 10)
            isWrapper(xElements[0]);
            isWrapper(yElements[0]);
        }
        if (emptyPoints >= 0) {
            let centerElements = document.elementsFromPoint(window.innerWidth / 2, window.innerHeight / 2)
            tracker.send({
                kind: 'stability',
                type: 'blank',
                emptyPoints: "" + emptyPoints,
                screen: window.screen.width + "x" + window.screen.height,
                viewPoint: window.innerWidth + 'x' + window.innerHeight,
                selector: getSelector(centerElements[0]),
            })
        }
    });
}
```

4.5 加载时间

- PerformanceTiming (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceTiming)
- DOMContentLoaded (https://developer.mozilla.org/zh-CN/docs/Web/Events/DOMContentLoaded)
- FMP (https://docs.google.com/document/d/1BR94tJdZLsin5poeet0XoTW60M0SjvOJQttKT-JK8Hl/view#)

4.5.1 阶段含义

字段 含义 navigationStart 初始化页面，在同一个浏览器上下文中前一个页面unload的时间戳，如果没有前一个页面的unload,则与fetchStart值相等 redirectStart 第一个HTTP重定向发生的时间,有跳转且是同域的重定向,否则为0 redirectEnd 最后一个重定向完成时的时间,否则为0 fetchStart 浏览器准备好使用http请求获取文档的时间,这发生在检查缓存之前 domainLookupStart DNS域名开始查询的时间,如果有本地的缓存或keep-alive则时间为0 domainLookupEnd DNS域名结束查询的时间 connectStart TCP开始建立连接的时间,如果是持久连接,则与 `fetchStart`

值相等 secureConnectionStart https 连接开始的时间,如果不是安全连接则为0 connectEnd TCP完成握手的时间，如果是持久连接则与 `fetchStart`

值相等 requestStart HTTP请求读取真实文档开始的时间,包括从本地缓存读取 requestEnd HTTP请求读取真实文档结束的时间,包括从本地缓存读取 responseStart 返回浏览器从服务器收到（或从本地缓存读取）第一个字节时的Unix毫秒时间戳 responseEnd 返回浏览器从服务器收到（或从本地缓存读取，或从本地资源读取）最后一个字节时的Unix毫秒时间戳 unloadEventStart 前一个页面的unload的时间戳 如果没有则为0 unloadEventEnd 与 `unloadEventStart`

相对应，返回的是 `unload`

函数执行完成的时间戳 domLoading 返回当前网页DOM结构开始解析时的时间戳,此时 `document.readyState`

变成loading,并将抛出 `readyStateChange`

事件 domInteractive 返回当前网页DOM结构结束解析、开始加载内嵌资源时时间戳, `document.readyState`

变成 `interactive`

，并将抛出 `readyStateChange`

事件(注意只是DOM树解析完成,这时候并没有开始加载网页内的资源) domContentLoadedEventStart 网页domContentLoaded事件发生的时间 domContentLoadedEventEnd 网页domContentLoaded事件脚本执行完毕的时间,domReady的时间 domComplete DOM树解析完成,且资源也准备就绪的时间, `document.readyState`

变成 `complete`

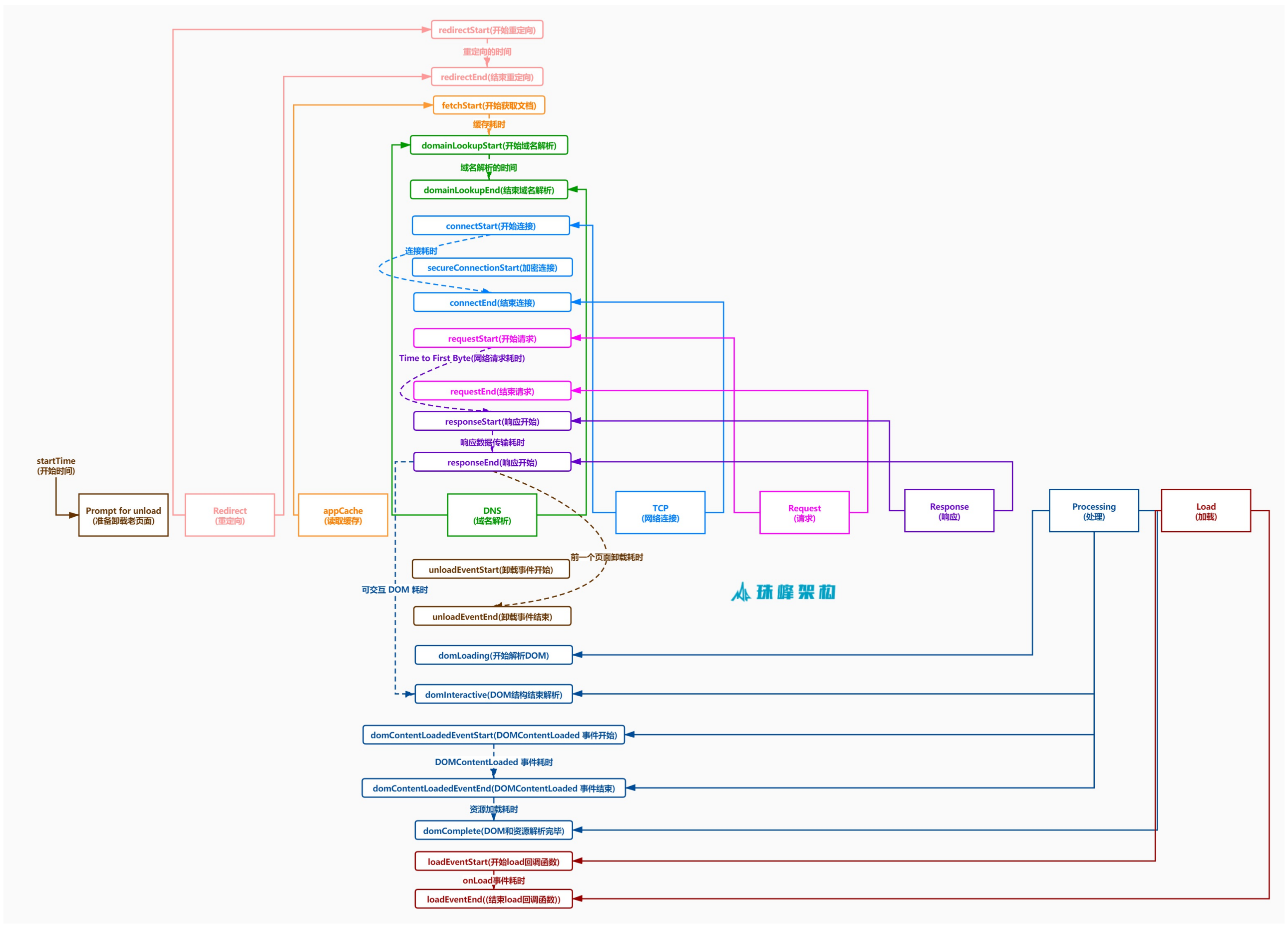
.并将抛出 `readystatechange`

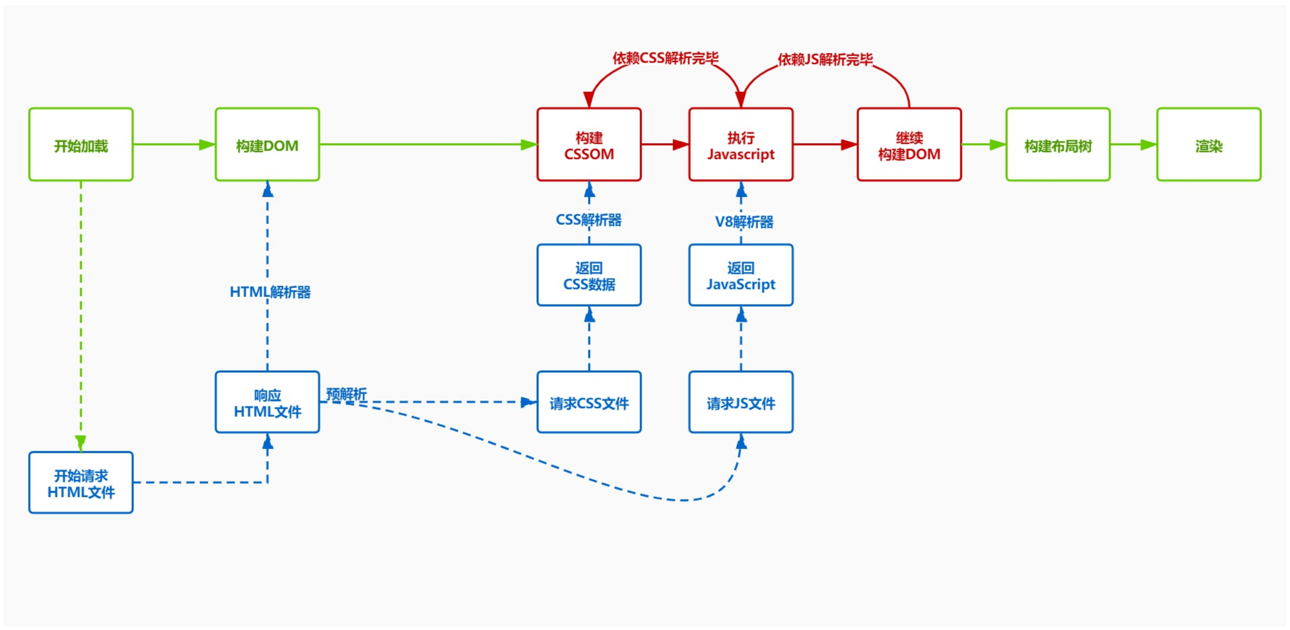
事件 loadEventStart load 事件发送给文档，也即load回调函数开始执行的时间 loadEventEnd load回调函数执行完成的时间

4.5.2 阶段计算

字段 描述 计算方式 意义 unload 前一个页面卸载耗时 unloadEventEnd – unloadEventStart - redirect 重定向耗时 redirectEnd – redirectStart 重定向的时间 appCache 缓存耗时 domainLookupStart – fetchStart 读取缓存的时间 dns DNS 解析耗时 domainLookupEnd – domainLookupStart 可观察域名解析服务是否正常 tcp TCP 连接耗时 connectEnd – connectStart 建立连接的耗时 ssl SSL 安全连接耗时 connectEnd – secureConnectionStart 反映数据安全连接建立耗时 ttfb Time to First Byte(TTFB)网络请求耗时 responseStart – requestStart TTFB是发出页面请求到接收到应答数据第一个字节所花费的毫秒数 response 响应数据传输耗时 responseEnd – responseStart 观察网络是否正常 dom DOM解析耗时 domInteractive – responseEnd 观察DOM结构是否合理，是否有JS阻塞页面解析 dcl DOMContentLoaded 事件耗时 domContentLoadedEventEnd – domContentLoadedEventStart 当 HTML 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发，无需等待样式表、图像和子框架的完成加载 resources 资源加载耗时 domComplete – domContentLoadedEventEnd 可观察文档流是否过大 domReady DOM阶段渲染耗时 domContentLoadedEventEnd – fetchStart DOM树和页面资源加载完成时间，会触发 `domContentLoaded`

事件 首次渲染耗时 首次渲染耗时 responseEnd-fetchStart 加载文档到看到第一帧非空图像的时间，也叫白屏时间 首次可交互时间 首次可交互时间 domInteractive-fetchStart DOM树解析完成时间，此时document.readyState为interactive 首包时间耗时 首包时间 responseStart-domainLookupStart DNS解析到响应返回给浏览器第一个字节的时间 页面完全加载时间 页面完全加载时间 loadEventStart - fetchStart - onLoad onLoad事件耗时 loadEventEnd – loadEventStart





4.5.3 数据结构

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590828364183",
  "userAgent": "chrome",
  "kind": "experience",
  "type": "timing",
  "connectTime": "0",
  "ttfbTime": "1",
  "responseTime": "1",
  "parseDOMTime": "80",
  "domContentLoadedTime": "0",
  "timeToInteractive": "88",
  "loadTime": "89"
}
```

4.5.4 实现

1. src\index.html

src\index.html

```
    monitor

        let content = document.getElementsByClassName('content')[0];
        //content.innerHTML = '@'.repeat(10000);
        document.addEventListener('DOMContentLoaded', function () {
<span class="hljs-addition">+            let start = Date.now();</span>
<span class="hljs-addition">+            while ((Date.now() - start) < 1000) {}</span>
<span class="hljs-addition">+        });</span>
```

2. monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
import { injectXHR } from './lib/xhr';
import { blankScreen } from './lib/blankScreen';
+import { timing } from './lib/timing';
injectJsError();
injectXHR();
blankScreen();
+timing();
```

3. timing.js

src\monitor\lib\timing.js

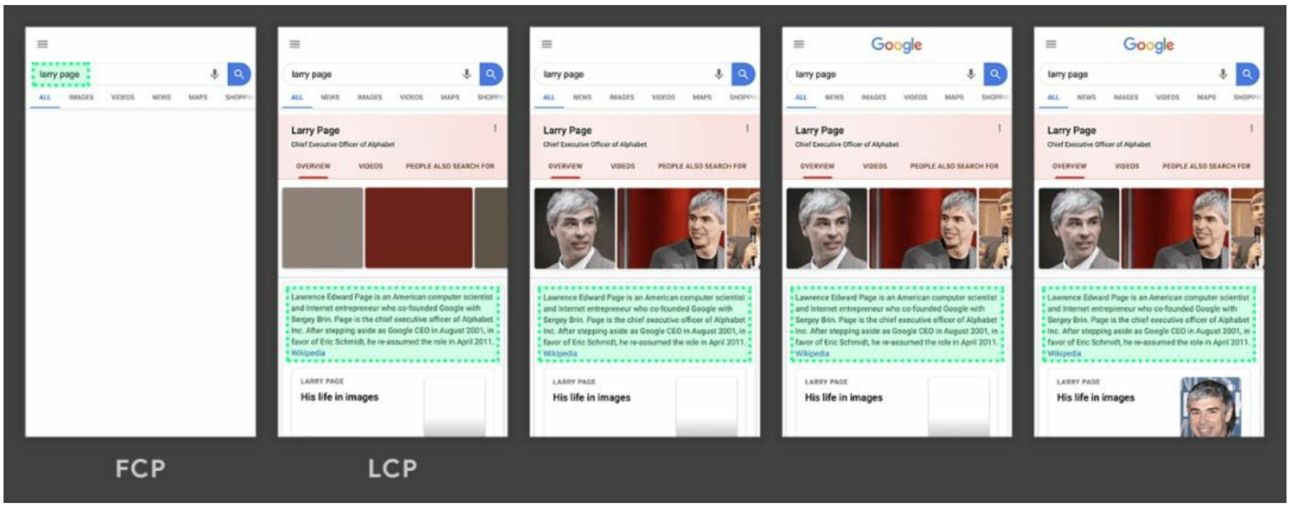
```
import onload from '../util/onload';
import tracker from '../util/tracker';
import formatTime from '../util/formatTime';
import getLastEvent from '../util/getLastEvent';
import getSelector from '../util/getSelector';
export function timing() {
    onload(function () {
        setTimeout(() => {
            const {
                fetchStart,
                connectStart,
                connectEnd,
                requestStart,
                responseStart,
                responseEnd,
                domLoading,
                domInteractive,
                domContentLoadedEventStart,
                domContentLoadedEventEnd,
                loadEventStart } = performance.timing;
            tracker.send({
                kind: 'experience',
                type: 'timing',
                connectTime: connectEnd - connectStart,
                ttfbTime: responseStart - requestStart,
                responseTime: responseEnd - responseStart,
                parseDOMTime: loadEventStart - domLoading,
                domContentLoadedTime: domContentLoadedEventEnd - domContentLoadedEventStart,
                timeToInteractive: domInteractive - fetchStart,
                loadTime: loadEventStart - fetchStart
            });

        }, 3000);
    });
}
```

4.6 性能指标

- PerformanceObserver.observe (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceObserver/observe)方法用于观察传入的参数中指定的性能条目类型的集合。当记录一个指定类型的性能条目时，性能监测对象的回调函数将会被调用
- entryType (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceEntry/entryType)
- paint-timing (https://w3c.github.io/paint-timing/)
- event-timing (https://wicg.github.io/event-timing/)
- LCP (https://wicg.github.io/largest-contentful-paint/)
- FMP (https://docs.google.com/document/d/1BR94tJdZLsin5poeet0XoTW60M0SjvOJQttKT-JK8HI/view)
- time-to-interactive (https://github.com/WICG/time-to-interactive)

字段 描述 备注 计算方式 FP First Paint(首次绘制) 包括了任何用户自定义的背景绘制，它是首先将像素绘制到屏幕的时刻 FCP First Content Paint(首次内容绘制) 是浏览器将第一个 DOM 渲染到屏幕的时间,可能是文本、图像、SVG等,这其实就是白屏时间 FMP First Meaningful Paint(首次有意义绘制) 页面有意义的内容渲染的时间 LCP (Largest Contentful Paint)(最大内容渲染) 代表在viewport中最大的页面元素加载的时间 DCL (DomContentLoaded)(DOM加载完成) 当 HTML 文档被完全加载和解析完成之后, `DOMContentLoaded`

事件被触发，无需等待样式表、图像和子框架的完成加载 L (onLoad) 当依赖的资源全部加载完毕之后才会触发 TTI (Time to Interactive) 可交互时间 用于标记应用已进行视觉渲染并能可靠响应用户输入的时间点 FID First Input Delay(首次输入延迟) 用户首次和页面交互(单击链接，点击按钮等)到页面响应交互的时间



4.6.1 数据结构设计

1. paint

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590828364186",
  "userAgent": "chrome",
  "kind": "experience",
  "type": "paint",
  "firstPaint": "102",
  "firstContentPaint": "2130",
  "firstMeaningfulPaint": "2130",
  "largestContentfulPaint": "2130"
}
```

2. firstInputDelay#

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590828477284",
  "userAgent": "chrome",
  "kind": "experience",
  "type": "firstInputDelay",
  "inputDelay": "3",
  "duration": "8",
  "startTime": "4812.344999983907",
  "selector": "HTML BODY #container .content H1"
}
```

4.6.2 实现

1. src\index.html

src\index.html

```

    monitor

        let content = document.getElementsByClassName('content')[0];
        //content.innerHTML = '@'.repeat(10000);
<span class="hljs-addition">+        setTimeout(() => {</span>
<span class="hljs-addition">+            let h1 = document.createElement('h1');</span>
<span class="hljs-addition">+            h1.innerHTML = '我是最有重要的内容';</span>
<span class="hljs-addition">+            h1.setAttribute('elementtiming', 'meaningful');</span>
<span class="hljs-addition">+            content.appendChild(h1);</span>
<span class="hljs-addition">+        }, 2000);</span>

```

2. timing.js

src\monitor\lib\timing.js

```
import onload from '../util/onload';
import tracker from '../util/tracker';
import formatTime from '../util/formatTime';
import getLastEvent from '../util/getLastEvent';
import getSelector from '../util/getSelector';
export function timing() {
+    let FMP, LCP;
+    new PerformanceObserver((entryList, observer) => {
+        let perfEntries = entryList.getEntries();
+        FMP = perfEntries[0];
+        observer.disconnect();
+    }).observe({ entryTypes: ['element'] });

+    new PerformanceObserver((entryList, observer) => {
+        const perfEntries = entryList.getEntries();
+        const lastEntry = perfEntries[perfEntries.length - 1];
+        LCP = lastEntry;
+        observer.disconnect();
+    }).observe({ entryTypes: ['largest-contentful-paint'] });

+    new PerformanceObserver(function (entryList, observer) {
+        let lastEvent = getLastEvent();
+        const firstInput = entryList.getEntries()[0];
+        if (firstInput) {
+            let inputDelay = firstInput.processingStart - firstInput.startTime;//处理延迟
+            let duration = firstInput.duration;//处理耗时
+            if (firstInput > 0 || duration > 0) {
+                tracker.send({
+                    kind: 'experience',
+                    type: 'firstInputDelay',
+                    inputDelay: inputDelay ? formatTime(inputDelay) : 0,
+                    duration: duration ? formatTime(duration) : 0,
+                    startTime: firstInput.startTime,
+                    selector: lastEvent ? getSelector(lastEvent.path || lastEvent.target) : ''
+                });
+            }
+        }
+        observer.disconnect();
+    }).observe({ type: 'first-input', buffered: true });

    onload(function () {
        setTimeout(() => {
            const {
                fetchStart,
                connectStart,
                connectEnd,
                requestStart,
                responseStart,
                responseEnd,
                domLoading,
                domInteractive,
                domContentLoadedEventStart,
                domContentLoadedEventEnd,
                loadEventStart } = performance.timing;
            tracker.send({
                kind: 'experience',
                type: 'timing',
                connectTime: connectEnd - connectStart,//TCP连接耗时
                ttfbTime: responseStart - requestStart,//ttfb
                responseTime: responseEnd - responseStart,//Response响应耗时
                parseDOMTime: loadEventStart - domLoading,//DOM解析渲染耗时
                domContentLoadedTime: domContentLoadedEventEnd - domContentLoadedEventStart,//DOMContentLoaded事件回调耗时
                timeToInteractive: domInteractive - fetchStart,//首次可交互时间
                loadTime: loadEventStart - fetchStart//完整的加载时间
            });
+            const FP = performance.getEntriesByName('first-paint')[0];
+            const FCP = performance.getEntriesByName('first-contentful-paint')[0];
+            console.log('FP', FP);
+            console.log('FCP', FCP);
+            console.log('FMP', FMP);
+            console.log('LCP', LCP);
+            tracker.send({
+                kind: 'experience',
+                type: 'paint',
+                firstPaint: FP ? formatTime(FP.startTime) : 0,
+                firstContentPaint: FCP ? formatTime(FCP.startTime) : 0,
+                firstMeaningfulPaint: FMP ? formatTime(FMP.startTime) : 0,
+                largestContentfulPaint: LCP ? formatTime(LCP.renderTime || LCP.loadTime) : 0
+            });
        }, 3000);
    });
}
```

4.7 卡顿

- 响应用户交互的响应时间如果大于100ms,用户就会感觉卡顿

4.7.1 数据设计

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590828656781",
  "userAgent": "chrome",
  "kind": "experience",
  "type": "longTask",
  "eventType": "mouseover",
  "startTime": "9331",
  "duration": "200",
  "selector": "HTML BODY #container .content"
}
```

4.7.2 实现

1. src\index.html

src\index.html

```
    monitor

+             执行longTask

        let content = document.getElementsByClassName('content')[0];
<span class="hljs-addition">+        let longTaskBtn = document.getElementById('longTaskBtn');</span>
<span class="hljs-addition">+        longTaskBtn.addEventListener('click', longTask);</span>
<span class="hljs-addition">+        function longTask() {</span>
<span class="hljs-addition">+            let start = Date.now();</span>
<span class="hljs-addition">+            console.log('longTask开始 start', start);</span>
<span class="hljs-addition">+            while (Date.now() < (200 + start)) { }</span>
<span class="hljs-addition">+            console.log('longTask结束 end', (Date.now() - start));</span>
<span class="hljs-addition">+        }</span>
```

2. monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
import { injectXHR } from './lib/xhr';
import { blankScreen } from './lib/blankScreen';
import { timing } from './lib/timing';
+import { longTask } from './lib/longTask';
injectJsError();
injectXHR();
blankScreen();
timing();
+longTask();
```

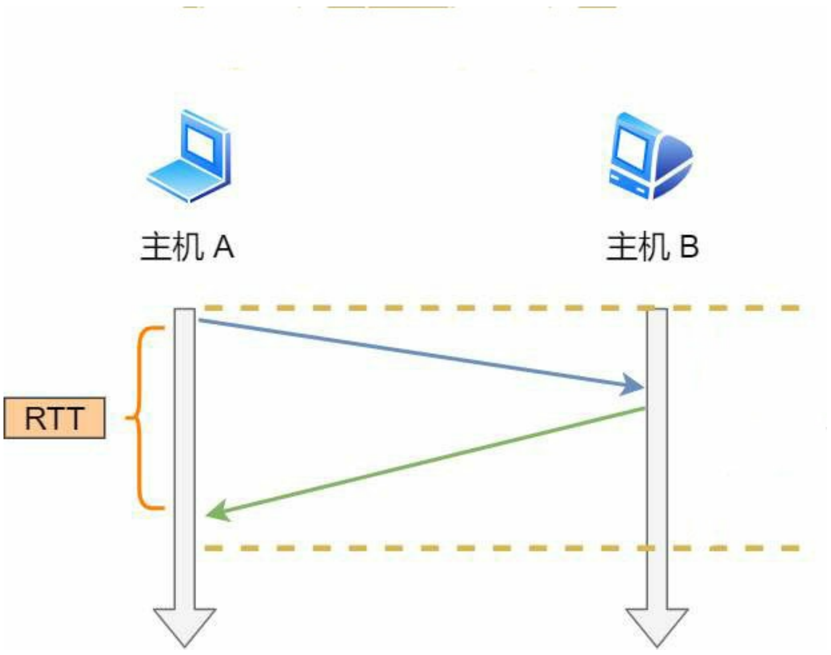
3. longTask.js

src\monitor\lib\longTask.js

```
import tracker from '../util/tracker';
import formatTime from '../util/formatTime';
import getLastEvent from '../util/getLastEvent';
import getSelector from '../util/getSelector';
export function longTask() {
    new PerformanceObserver((list) => {
        list.getEntries().forEach(entry => {
            if (entry.duration > 100) {
                let lastEvent = getLastEvent();
                requestIdleCallback(() => {
                    tracker.send({
                        kind: 'experience',
                        type: 'longTask',
                        eventType: lastEvent.type,
                        startTime: formatTime(entry.startTime),
                        duration: formatTime(entry.duration),
                        selector: lastEvent ? getSelector(lastEvent.path || lastEvent.target) : ''
                    });
                });
            }
        });
    }).observe({ entryTypes: ["longtask"] });
}
```

4.8 pv

- netinfo (http://wicg.github.io/netinfo)
- RTT(Round Trip Time)一个连接的往返时间，即数据发送时刻到接收到确认的时刻的差值
- navigator.sendBeacon() (https://developer.mozilla.org/zh-CN/docs/Web/API/Navigator/sendBeacon) 方法可用于通过HTTP将少量数据异步传输到Web服务器。



4.8.1 数据结构

```
{
  "title": "前端监控系统",
  "url": "http://localhost:8080/",
  "timestamp": "1590829304423",
  "userAgent": "chrome",
  "kind": "business",
  "type": "pv",
  "effectiveType": "4g",
  "rtt": "50",
  "screen": "2049x1152"
}
```

4.8.2 实现

1. src\index.html

src\index.html

```
    前端监控SDK
```

2. src\monitor\index.js

src\monitor\index.js

```
import { injectJsError } from './lib/jsError';
import { injectXHR } from './lib/xhr';
import { blankScreen } from './lib/blankScreen';
import { timing } from './lib/timing';
import { longTask } from './lib/longTask';
+import { pv } from './lib/pv';
injectJsError();
injectXHR();
blankScreen();
timing();
longTask();
+pv();
```

3. pv.js

src\monitor\lib\pv.js

```
import tracker from '../util/tracker';
export function pv() {
    var connection = navigator.connection;
    tracker.send({
        kind: 'business',
        type: 'pv',
        effectiveType: connection.effectiveType,
        rtt: connection.rtt,
        screen: `${window.screen.width}x${window.screen.height}`
    });
    let startTime = Date.now();
    window.addEventListener('unload', () => {
        let stayTime = Date.now() - startTime;
        tracker.send({
            kind: 'business',
            type: 'stayTime',
            stayTime
        });
    }, false);

}
```

5.查询报表

- 图表说明 (https://help.aliyun.com/document_detail/69313.html)
- 查询日志数据
- 聚类分析
- 数据可视化
    - 趋势 柱状图、拆线图、曲线图
    - 比例 饼图、环状图、面积图
- 仪表盘

5.1 监控项分布

```
* | SELECT type, COUNT(*) as number GROUP BY type LIMIT 10
```

5.2 浏览器分布

```
* | SELECT userAgent, COUNT(*) as number GROUP BY userAgent LIMIT 10
```

5.3 页面分辨率分布

```
* | SELECT screen, COUNT(*) as number GROUP BY screen LIMIT 10
```

6.参考

6.1 第三方

6.1.1 商业产品

- 神策数据 (https://www.sensorsdata.cn/)
- GrowingIO (https://www.growingio.com/)

6.1.2 开源产品

- fundebug (https://www.fundebug.com/)
- BetterJS (https://github.com/BetterJS/doc)
- Sentry (https://sentry.io)
- @sentry/browser (https://sentry.io/onboarding/zhufeng/get-started/)

6.2 defer 和 async

- defer异步下载JavaScript文件,会在HTML解析完成之后 `DOMContentLoaded`事件调用前执行,不会阻塞页面渲染
  - 如果script标签设置了该属性，则浏览器会异步的下载该文件并且不会影响到后续DOM的渲染
  - 如果有多个设置了defer的script标签存在，则会按照顺序执行所有的script
  - defer脚本会在文档渲染完毕后， `DOMContentLoaded`事件调用前执行
- async异步下载JavaScript文件，下载完成之后会立即执行，有可能会阻塞页面渲染
  - async的设置,会使得script脚本异步的加载并在允许的情况下执行
  - async的执行,并不会按着script在页面中的顺序来执行，而是谁先加载完谁执行