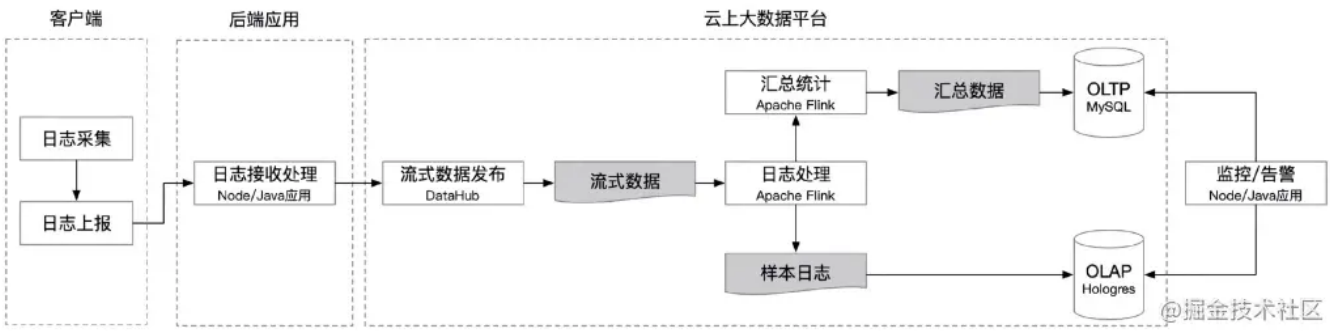
羽晋 Lv2
2021年03月06日 阅读 2014

关注

前端监控体系及实现技术详解

1.前端监控体系概况

一个完整的前端监控体系包括了采集、日志存储、日志切分&计算、数据分析、告警等流程，对于一名前端开发工程师来说，也就意味着工作不再局限于前端业务的开发工作，需要有Nginx服务运维能力、实时/离线分析能力、Node应用开发运维能力等等。



目前主流的前端监控系统主要有 阿里ARMS 、FUNdebug、sentry 等，但不论哪种系统的体系都基本是符合上文所讲到的体系，下面就体系中客户端部分的功能的实现进行探究。

2.采集部分

2.1稳定性

2.1.1脚本错误

脚本错误主要有两类：语法错误、运行时错误。监控的方式主要有：

复制代码

```
try-catch
window.onerror
```

控特定错误，两种形式结合使用更加高效。使用window.onerror捕获JS运行时错误,使用window.addEventListener('unhandledrejection')捕获未处理的promise reject错误，window.addEventListener('error')捕获资源加载错误。但它也能捕获js运行时错误，为避免重复上报js运行时错误，此时只有event.srcElement inatanceof HTMLScriptElement或HTMLLinkElement或HTMLImageElement时才进行数据采集。

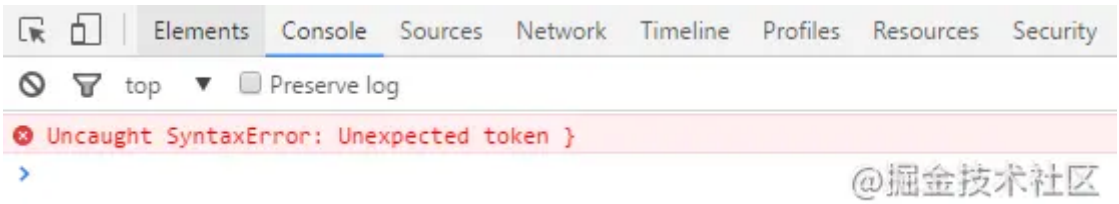
另外，当try-catch中代码发生语法错误或异步错误时，则无法正常捕捉。

示例 · try-catch (语法报错)

复制代码

```
try {
    function empty()   // <-  throw error 语法错误
} catch(e){
    console.log('语法错误信息 ↙');
    console.log(e);
}
```

无法捕捉错误

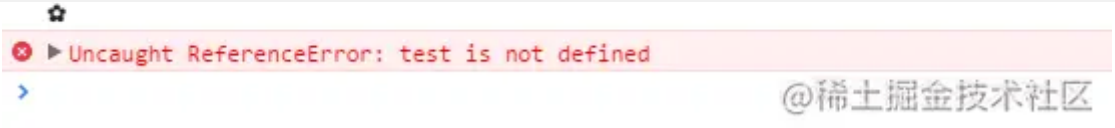


示例 · try-catch (异步错误)

复制代码

```
try {
    setTimeout(function() {
        test // <- throw error 异步错误
    },0)
} catch(e){
    console.log('异步错误信息 ↙');
    console.log(e);
}
```

无法捕捉错误



语法错误无法在 try-catch 中进行捕抓、而异步报错则可以通过为异步函数块再包装一层 try-catch，增加标识信息来配合定位，可以用工具来进行处理，这里不展开。

2.1.1.1解决跨域资源Script error

还有一点要特别注意，要注意跨域资源的脚本的报错

> developer.mozilla.org...
>
> 当加载自不同域的脚本中发生语法错误时，为避免信息泄露（参见bug 363897），语法错误的细节将不会报告，而代之简单的"Script error."。在某些浏览器中，通过在
>
> 例子: sandbox.runjs.cn/show/... 请打开页面打开控制台。该页面分别加载了两个不同域的js脚本，配置了crossorigin的window.onerror可以报出详细的错误，没有配置crossorigin只能报出'script error'，并且没有错误信息

解决Script error 来自同源策略的影响的具体步骤如下：

1. 为页面上script标签添加 `crossorigin` 属性。

```
<script src="http://127.0.0.1:8077/main.js" crossorigin></script>
```

增加 `crossorigin` 属性后，浏览器将自动在请求头中添加一个 `Origin` 字段，发起一个 跨来源资源共享 请求。 `Origin` 向服务端表明了请求来源，服务端将根据来源判断是否正常响应。

2. 响应头中增加 `Access-Control-Allow-Origin` 来支持跨域资源共享。

`Access-Control-Allow-Origin: *` 表示通过该跨域请求，且该资源可以被任意站点跨站访问。而当该资源仅允许来自 http://127.0.0.1:8066 的跨站请求，其它站点都不能跨站访问时，将可以返回：

Vary 字段的作用在于为缓存服务器提供缓存规则及缓存命运的依据。当增加 `Vary:Origin` 响应头后，缓存服务器将会按照 Origin 字段的内容，缓存不同版本，在请求响应时根据请求头中的 Origin 决定是否能够使用缓存响应。

举例 · 不加 Vary将存在错误命中缓存的问题

上图中，第一个请求（Origin: 127.0.0.1:8066）响应被浏览器缓存了，当第二个请求（Origin: 127.0.0.1:8888）发起，被错误命中了前一个请求的缓存，收到了 Access-Control-Allow-Origin:http://127.0.0.1:8066 的响应时，将导致资源加载失败。

所以当 Access-Control-Allow-Origin 不是返回为 * 时，需要加上 Vary 返回头来避免引缓存导致的权限问题。

2.1.2 接口异常

通过重写 `XMLHttpRequest` 和 `fetch` 的原生方法来实现

封装xmlHttpRequest

复制代码

```
if(!window.XMLHttpRequest) return;
var xmlhttp = window.XMLHttpRequest;
var _oldSend = xmlhttp.prototype.send;
var _handleEvent = function (event) {
    if (event && event.currentTarget && event.currentTarget.status !== 200) {
          // 自定义错误上报 }
}
xmlhttp.prototype.send = function () {
    if (this['addEventListener']) {
        this['addEventListener']('error', _handleEvent);
        this['addEventListener']('load', _handleEvent);
        this['addEventListener']('abort', _handleEvent);
    } else {
        var _oldStateChange = this['onreadystatechange'];
        this['onreadystatechange'] = function (event) {
            if (this.readyState === 4) {
                _handleEvent(event);
```

```
}
```

封装fetch

复制代码

```
if(!window.fetch) return;
    let _oldFetch = window.fetch;
    window.fetch = function () {
        return _oldFetch.apply(this, arguments)
        .then(res => {
            if (!res.ok) { // True if status is HTTP 2xx
                // 上报错误
            }
            return res;
        })
        .catch(error => {
            // 上报错误
            throw error;
        })
}
```

2.1.3资源异常

资源异常:页面内的图片、css、JS等Assets资源加载失败在捕获类型的error事件里可以拿到资源加载失败回调:

复制代码

```
window.addEventListener( 'error ', function(e){
//排除JSError
    if( ! (e instanceof ErrorEvent)){
//资源路径
        e.target.src || e.target.href
        //资源类型
        e.target.tagName
    }
}, true)
```

tips：addEventListener第三个参数设置为true就在捕获过程中执行，反之就在冒泡过程中执行处理函数。

2.1.5Crash

Crash:页面因为内存溢出、死循环等原因导致崩溃的异常

Sw侵入性强，风险高指的是一个页面只能有一个Sw，若页面上需要使用，则需要进行嵌入，浏览器进入后台或切换页面后会进入Pause状态

2.2流畅性

2.2.1页面加载速度

以上统计的都是一些通用的性能表示指标，但其实他们有时候并不能很好的表示页面的性能，比如页面的onload时间，有可能页面是在加载完成过后再进行一些页面的渲染工作，这时候就需要一些自定义指标来衡量

下面则是用户在使用过程中与流畅性/操作流畅性相关的四个指标评价标准

2.2.1响应速度

响应速度:从用户操作到页面响应的耗时，通常要求小于100ms

基于PerformanceEventTiming监听用户的任意输入（例如click、touchmove、wheel等）到浏览器给出响应的延迟时间:

复制代码

```
observer.observe({type: "event", buffered: true});
```

2.2.3动画流畅性

动画流畅性:页面上任意动画的帧率、帧数是否稳定

在动画运行期间监听每次requestAnimationFrame的执行，计算:

1.帧率:动画运行帧数/动画运行时间

2.掉帧率:(以60FPS标准应该运行的帧数–实际运行帧数)/以60FPS标准应该运行的帧数

复制代码

```
//以60帧每秒的标准，每一帧之间的间隔msIn0neFrame = 1000/60;
//期望帧数
expectedFrames = Math.floor(e.elapsedTime*1000 / msInoneFrame ) ;
//掉帧率
error_rate = (expectedFrames –运行帧数)/expectedFrames
```

2.2.4滚动流畅性

滚动流畅性:从用户手指滑动到页面真正开始滚动的延迟时间，通常要求小于100ms

chrome在《Speed Launch Metrics Survey》提到了SSL和SUL2个指标，它们分别用来衡量首次和非首次手指滑动到画面开始滚动起来的延时

复制代码

```
//目前只有集成了UC内核的阿里系APP（如手淘、猫客）支持。
var observer = new Performanceobserver ( ( list)=>{
list .getEntries ( ) .forEach ( entry =>{
if( entry –duration > 100 ) {
// send
}) ;
}) ;
observer.observe ( {entryTypes: [ "touchscrolllatency" ] } ) ;
//entry对象包含核心字段:
{
" name " : "ssl" // "ssl" or "sul"
```

2.2.5卡顿

卡顿:页面整个生命周期中，主线程持续执行某一个任务的耗时大于50ms

浏览器的事件队列机制决定，要实现小于100毫秒的响应，应用必须在每50毫秒内将控制返回给主线程:

PerformanceLongTask提供了检测卡顿的能力，可以检测到浏览器内核主线程卡顿时间超过50ms的异常:

复制代码

```
var observer = new Performanceobserver(function(list) {
list.getEntries().forEach(entry => {
//开时时间: entry.startTime
//持续时间: entry.duration
}
});
observer.observe({type: "longtask", buffered: true});
```

2.3在VUE应用中脚本错误的捕获

前段时间在我准备自己的一个前端监控sdk项目时候碰巧就遇到了有人在思否上提了一个问题，这个问题答案其实很简单，因为 Vue 全局捕获错误 Vue.config.errorHandler = function(error, vm, info){},这个方法中的 error 和 window.onerror 吐出来的格式是不一样的，直接通过 source-map 包处理是搞不成的，映射关系映射不上去，一堆问题，需要用 TraceKit这个包来转换下就可以用了。

sentry目前也提供了专门给Vue项目使用的监控SDK,其中就用到了 Vue.config.errorHandler 方法，不过错误转换则是使用的 raven-js，是收集浏览器错误的, 已经集成了 TraceKit，下面是sentry中的关键部分源码

复制代码

```
function vuePlugin(Raven, Vue) {
  Vue = Vue || window.Vue;
```

```
  Vue.config.errorHandler = function VueErrorHandler(error, vm, info) {
      const metaData = {};
    if (Object.prototype.toString.call(vm) === '[object Object]') {
      metaData.componentName = formatComponentName(vm);
      metaData.propsData = vm.$options.propsData;
    }

    if (typeof info !== 'undefined') {
      metaData.lifecycleHook = info;
    }
    // ...

    // 上报
    Raven.captureException(error, {
      extra: metaData
    });
    if (typeof _oldOnError === 'function') {
      // 为什么这样做?
      _oldOnError.call(this, error, vm, info);
    }
  };
}
module.exports = vuePlugin;
```

以上“为什么这样做？”的答案是:

这里其实是用到了AOP（面向切面编程）的思想，它的主要作用是把一些跟核心业务逻辑模块无关的功能抽离出来，其实就是给原函数增加一层，不用管原函数内部实现。

另外如果不这样写的话也需要单独将错误抛出，因为当开发者配置了 Vue.config.errorHandler过后，错误信息是不会再打印在控制台的，最简单就是再使用console.error()将信息打印在控制台。

Vue2.0中重写数组的8个方法就是使用的这个思想，详情请参考zhuanlan.zhihu.com/p/166676919

2.4用户行为回放

首先是要确定哪些行为需要采集？我们站在用户的立场去考虑一个单页应用的浏览周期内的可能流程：进入应用首页——加载页面内容——浏览页面内容——用户交互（鼠标交互/键盘交互等）——跳转到新页面.....

在以上代码中，Api请求，error 在前面已经讲了监控的方法，所以下面整理一下图标、键盘以及路由跳转事件的监控记录

我们可以在顶层的document上全面监听各类用户交互事件，如click,keypress,mousemove,scroll等等。但是这种方法也有一个明显的缺陷，假设用户监听了某个dom上的click事件，并且设置了event.stopPropagation()，这种情况的点击事件是无法被document监听到的，而往往这类行为对于错误诊断和业务分析都尤为重要。解决方法是在addEventListener中埋入钩子。

复制代码

```
var bhEventHandler = function(){
  //记录用户行为
}
document.addEventListener('click', bhEventHandler, false);
var types = ['EventTarget', 'Node'];
for (var i = 0; i < types.length; i++) {
    var type = types[i];
    var proto = window[type] && window[type].prototype;
    reWrite(proto.addEventListener, function (orig) {
        //重写addEventListener，记录用户行为
    });
}
```

路由跳转无疑会触发浏览器历史记录的改变，每当处于激活状态的历史记录条目发生变化时，window的popstate事件会触发，但是调用history.pushState()或者history.replaceState()不会触发popstate事件。因此路由跳转的监控可以分为两个方面，一方面在window. onpopstate中埋入钩子，另一方面在history.pushState和history.replaceState中埋入钩子。

复制代码

```
var origPopstate = window.onpopstate;
window.onpopstate = function () {
    //记录路由行为
    if (origPopstate) {
        return origPopstate.apply(this, args);
    }
};
var dosomething = function (orig) {
    return function (...args) {
        //记录路由行为
        return orig.apply(this, args);
    };
};
reWrite(window.history.pushState, dosomething);
reWrite(window.history.replaceState, dosomething);
```

dom 树来实现用户行为的回放，就跟看视频是一个效果，之前 vue 的教学站实现点击视频，直接编辑 demo 那种效果，实际上就是这个原理，我们看到的不是视频，是在绘制 dom 树 社区有这个库 rrweb，就是实现这个功能的。

但是这个接口有个缺点就是需要收集上报的信息很多，在用户基数很大的应用中并不合适，比较适合用在B端用户的应用上。

3.信息上报

采集收集好信息过后接下来就是数据上报的工作了，目前主要由以下3种上报信息方式。

丢点：在浏览器点击跳转时，跳转前的点击上报请求都会进行一个三次握手，如果此时，网络较慢、服务器运行缓慢或者上报请求还在处理阶段，这时，如果页面被卸载了，浏览器都会自动对当前的请求进行abort。这样，这个http的请求就没有建立，导致上报没有真正发出。

下面的例子展示了一个理论上的统计代码——在卸载事件处理器中尝试通过一个同步的 XMLHttpRequest 向服务器发送数据。这导致了页面卸载被延迟。

复制代码

```
window.addEventListener('unload', logData, false);
function logData() {
    var client = new XMLHttpRequest();
    client.open("POST", "/log", false); // 第三个参数表明是同步的 xhr
    client.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
    client.send(analyticsData);
}
```

这就是 sendBeacon() 方法存在的意义。使用 sendBeacon() 方法会使用户代理在有机会时异步地向服务器发送数据，同时不会延迟页面的卸载或影响下一导航的载入性能。这就解决了提交分析数据时的所有的问题：数据可靠，传输异步并且不会影响下一页面的加载。此外，代码实际上还要比其他技术简单许多！

下面的例子展示了一个理论上的统计代码模式——通过使用 sendBeacon() 方法向服务器发送数据。

复制代码

```
window.addEventListener('unload', logData, false);
```

所以这里的推荐是优先检查浏览器是否支持Navigator.sendBeacon()，不支持的话再使用其他上报方式。

4.日志的解析、处理、存储，到分析、告警功能实现

1.要做的事情

- 日志接收处理;建设后端应用,提供日志上报接口给到采集SDK
- 数据发布:后端应用接收到的日志后处理成可被实时流计算处理的流式数据，如DataHub、SLS、Kafaka等
- 日志处理:流计算平台对流式数据进行实时处理，可基于Flink、Spark、Storm等
- 1.基于样本的实时分析:基于OLAP数据库进行实时分析，如阿里云Hologres、Hive、Kylin等
- 2实时汇总计算:将原始日志实时汇总计算成指标（如错误率）后存入OLTP数据库，如阿里云RDS、GaussDB、TBase等
- 监控告警:建设前台应用，实现监控、告警

2.需要的系统服务:

应用服务器＋DataHub + Flink + Hologres (RDS)

4.1SourceMap定位报错位置

以上绝大多数都是后端同学的工作了，但前端部分还有个较为重要的任务便是实现根据sourceMap+线上Js代码报错信息定位到真实逻辑代码，以快速定位异常

一般真正上线运行的js代码都是要经过压缩加密的，若直接上线源码将会出现的问题有两个

1. 源代码泄漏

缩代码的报错时，可以通过错误信息的行列数与对应的 SourceMap 文件，处理后得到源文件的具体错误信息。

SourceMap 文件中的 sourcesContent 字段对应源代码内容，不希望将 SourceMap 文件发布到外网上，而是将其存储到脚本错误处理平台上，只用在处理脚本错误中。

通过 SourceMap 文件可以得到源文件的具体错误信息，结合 sourcesContent 上源文件的内容进行可视化展示，让报错信息一目了然！

基于 SourceMap 快速定位脚本报错方案

使用sourceMap文件定位真实代码的流程参考：

将会用到source-map和stacktracey这两个库，其中source-map用于解析sourcemap文件、还原源码，stacktracey用于解析错误的stack信息。

我们收集到错误信息的stack举个例子大概如下：

复制代码

```
Error: test at App.render (webpack:///./src/app.tsx?:47:13) at finishClassComponent
(webpack:///./node_modules/react-dom/cjs/react-dom.development.js?:18470:31) at updateClass(
(webpack:///./node_modules/react-dom/cjs/react-dom.development.js?:18423:24) at beginWork$1
...
```

以stack的第一行来说明

1. 括号中的webpack:///./src/app.tsx表示源码文件的名字。（这里文件名字比较奇怪暂时先不深究
2. 括号中的:47:13表明错误发生在47行13列

手动解析stack太麻烦，可以直接调用stacktracey帮助我们解析，简单方便。

复制代码

```
const tracey = new Stacktracey(errorStack); // 解析错误信息
 for (const frame of tracy) {
  // 这里的frame就是stack中的一行所解析出来的内容
  console.log(frame.line, frame.column);
```

我们已经知道错误发生的行和列了，那么在通过source-map获得源码，就可以知道具体是哪里出错了。

下面是一个小小的例子结合了source-map和stacktracey。

其中有一点需要说明的是，stacktracey所解析出的行列信息是在压缩后代码中的位置，为了获得对应在源码中的行列信息，需要调用consumer.originalPositionFor来获取。关于更多source-map的api还是得看看文档。

复制代码

```
const sourceMap = require('source-map');
const SourceMapConsumer = sourceMap.SourceMapConsumer;
const Stacktracey = require('stacktracey');
const errorStack = '...'; // 错误信息
const sourceMapFileContent = '...'; // sourcemap文件内容
const tracey = new Stacktracey(errorStack); // 解析错误信息
const sourceMapContent = JSON.parse(sourceMapFileContent);
const consumer = await new SourceMapConsumer(sourceMapContent);
for (const frame of tracey) { // 这里的frame就是stack中的一行所解析出来的内容
// originalPosition不仅仅是行列信息，还有错误发生的文件originalPosition.source
 const originalPosition = consumer.originalPositionFor({ line: frame.line, column: frame.co
 const sourceContent = consumer.sourceContentFor(originalPosition.source);
 console.log(sourceContent);
 }
```

这里是是一个利用source-map库实现问题定位的demo项目github.com/joeyguo/noe...

作者的写的项目启动文档不全，我这里补充一下，运行前要先分别进入主目录以及example目录下执行npm install安装相关依赖后再执行启动命令

可以看到，在报错页面的产生的报错脚本信息经过处理后，可在另一个页面查看其对应的真正源码

参考文章：

developer.aliyun.com/article/714...

github.com/cshover/fra...

zhuanlan.zhihu.com/p/136840107

zhuanlan.zhihu.com/p/64033141

zhuanlan.zhihu.com/p/136840107

www.zhihu.com/question/28...

www.jianshu.com/p/80b559be0...

由校： 实现一个监控系统 —— 阿里巴巴前端练习生计划课程

文章分类 前端 文章标签 监控

羽晋 Lv2 前端实习 @ 阿里
获得点赞 80 · 获得阅读 4,742

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（2） 最新 最热

mvtac Lv1 独立开发 @ 独立开发 7月前

润住 Lv2 前端工程师 @ 阿里巴巴... 7月前

待我慢慢研究~

点赞 回复

相关推荐

mogii 4月前 前端

基于Sentry搭建前端异常监控系统

虽然在我们的项目上线前会有很多的测试流程，但是测试流程肯定无法保证 100%覆盖所有...

1395 28 7

好学习吧丶 1年前 前端

去大厂，你就应该了解前端监控和埋点!

在现今用户就是上帝的年代，互联网竞争如此之大的时代，有针对性的对每个用户的喜好定制不同的内容，按照用户...

1.6w 369 41

爱秀的演员 5月前 架构 前端

搭建前端异常监控系统

1. 收集错误 2. 上报错误 3. 代码上线打包将sourcemap文件上传至错误监控服务器 4. 发生错误时监控服务器接收错误...

1082 22 2

K_ON 1年前 前端

前端性能监控方案（首屏、白屏时间等）

总下载时间：window.onload的触发节点。 白屏时间节点指的是从用户进入网站（输入url、刷新、跳转等方式）的时...

1.1w 56 9

前端早早聊 8月前 前端 监控

今天聊：60 天急速自研-前端埋点监控跟踪系统

前端早早聊大会，前端成长的新起点，与掘金联合举办。 加 Scott 微信 codingdreamer 进大会周边技术群，前端页...

2350 43 3

前端刊物 3年前

如何设计一个前端监控系统

一般来说，产品做出的原型多多少少会带有“个人”倾向，UI设计的交互也会上所不同，再当公司生存下来了后，数据

navigator.sendBeacon() 方法可用于通过HTTP将少量数据异步传输到Web服务器。

2094 29 评论

前端早早聊 1年前 前端 监控

小公司自建前端监控埋点体系，证明可行

前端早早聊大会，前端成长的新起点，与掘金联合举办。 加微信 codingdreamer 进大会专属内推群，赢在新的起跑...

3.2w 664 41

杭城小刘 11月前 监控 性能优化

APM 监控系统：卡顿监控、启动时间监控、CPU 使用率监控

App 的性能问题是影响用户体验的重要因素之一。性能问题主要包含：Crash、网络请求错...

1536 17 2

得物技术 7月前 监控 前端

【得物技术】前端性能监控实践

对于前端来说，最重要是的体验，而在前端体验中，最为核心的就是性能。秒开率、流畅程度等一系列指标都直接影...

2267 32 评论

korbinzhao 3年前 JavaScript 前端

前端数据监控到底在监控什么?

前端数据监控一般分为性能数据监控和线上异常监控。本文对这两块数据的监控原理和方法进行整理说明。 将监控代...

6514 157 2

程序员秋风 3年前 Node.js Redis

搭建一个前端监控系统,不再错过BUG

还记得在我上一家公司中，某一大佬做了一个监控系统，牛逼哄哄，挺想研究他到底是怎么搞出来的。当然我们也不...

1.2w 708 47

我的前端杂货铺 2年前 前端 监控

如何监控网页崩溃?

本文是如何监控网页的卡顿？的下篇。今天我们把话题聚焦在如何监控网页的崩溃上。 卡顿也就是网页暂时响应比较...

5267 108 3

荒山 2年前 前端 团队管理

if 我是前端团队 Leader，怎么制定前端协作规范?

笔者长期单枪匹马在前端领域厮杀(言外之意就是团队就一个人)，自己就是规范。随着公司...

14.0w 3868 220

不如吃茶去 2月前 JavaScript 前端

前端监控系统之接口监控

前言 为什么要对接口进行监控 接口的性能监控在系统上我们可以在后端进行统计，但是在浏览器上发送请求时，可...

794 6 2

爱创课堂前端技术分享 12月前 前端

前端监控和前端埋点

前端监控和前端埋点，一个是目的，一个是实现方式，做前端埋点，是为了做前端监控，那为什么要做前端监控呢？...

9459 169 8

有赞技术 2年前 前端 有赞

有赞前端质量保障体系

最近一年多一直在做前端的一些测试，从小程序到店铺装修，基本都是纯前端的工作，刚开始从后端测试转为前端测...

2.6w 698 28

前端新能源 3年前

把前端监控做到极致

说到监控，大家第一时间想到的肯定是 Zabbix、Nagios 等各种强大的后端监控服务。诚然...

2.1w 752 12

yeyan1996 2年前 JavaScript 前端

一个合格的中级前端工程师需要掌握的 28 个 JavaScript 技巧

2. 循环实现数组 map 方法 3. 使用 reduce 实现数组 map 方法 4. 循环实现数组 filter 方法 5. 使用 reduce 实现数组...

21.6w 4933 291