



前端稳定性监控

演讲者：周传森



为什么需要前端监控

前端监控主要用于跟踪和理解用户在使用应用时的体验和问题

- 提升用户体验 - 可通过监控用户交互、页面渲染时间、页面加载速度等，来发现和优化瓶颈，进一步提升用户体验。
- 故障排查 - 前端监控可以帮助我们实时收集和上报网页错误、性能问题等，当问题发生时可以尽快发现和定位问题，降低故障恢复时间。
- 用户行为分析 - 通过前端监控，我们可以收集用户的行为数据，如点击事件、页面停留时间、路径分析等，帮助我们更好地理解用户使用习惯，优化产品设计。
- 业务数据监控 - 可以监控关键业务数据的变化，例如购物车转化率、注册量等，及时发现潜在问题。

为什么不用第三方监控

- 方便团队做自定义的UV用户识别，比如通过登录账号ID或者通过设备信息；甚至从设备信息转入登录态后的继承
- 方便接入自己团队的各种告警业务等
- 方便做各维度数据的联合分析，比如发生错误可以联动查询用户行为追溯数据等
- 方便做业务需求上的拓展，比如自定义埋点、特殊的数据分析维度
- 方便前后端全链路的一个API请求链路分析；
- 私有化部署需要付费且价格不低，不易定制化。

前端监控做了那些

前端搭建监控体系，可以概括为为了做两件事：

- 如何及时发现问题
- 如何快速定位问题

可以拆分为

- 页面的性能情况 - 如何监控页面的性能情况
- 用户的行为情况 - 包括PV、UV、访问来路，路由跳转等
- 接口的调用情况 - 通过http访问的外部接口的成功率、耗时情况等
- 页面的稳定情况 - 各种前端异常等
- 数据上报及优化 - 如何将监控捕获到的数据优雅的上报

前端监控系统目录

1. 前端监控演讲
2. 为什么需要前端监控
3. 为什么不用第三方监控
4. 前端监控做了那些
5. 前端监控系统目录
6. 错误监控
 1. 整体错误上报流程
 2. TransportData 上报流程
 3. sourceMap 上传流程
7. 性能监控
 1. Core Web Vitals (核心网页指标)
8. 行为监控

错误监控

- IS运行异常
- 静态资源加载异常
- Promise异常
- HTTP请求异常
- 跨域脚本错误
- Vue2、Vue3 错误捕获

```
const handler = (event: ErrorEvent) => {
    // 阻止向上抛出控制台报错
    event.preventDefault();
    // 处理错误
    console.log(event);
    // HandleEvents.handleError(event);
};

window.addEventListener('error', (event) => handler(event), true)

function throwError() {
    setTimeout(() => {
        throw new Error('error');
    }, 1000);
}

throwError();
```

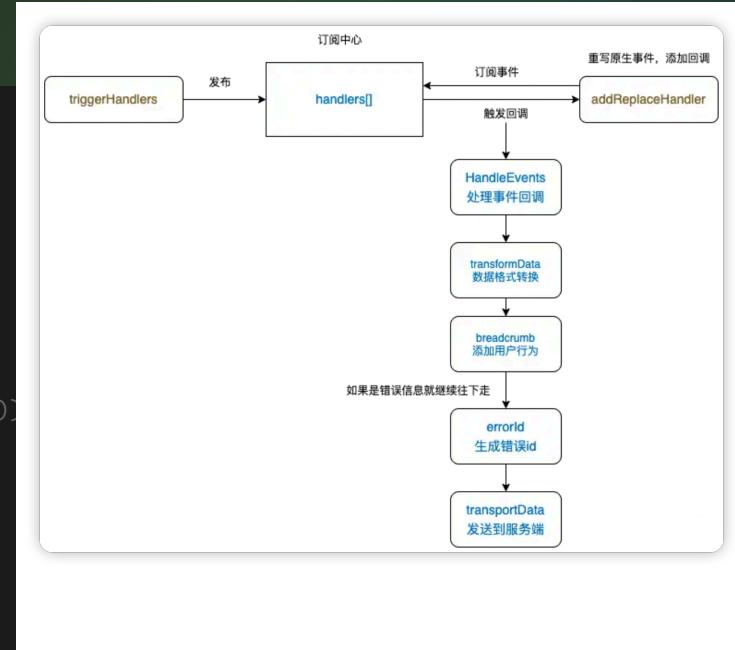
以及最后sourceMap上传到私服的过程

整体错误上报流程

以Js运行异常为例，整体上报流程如下：

```
1 // HandleEvents 是个map对象，这里就对应handleError函数
2 handleError(errorEvent: ErrorEvent) {
3     const target = errorEvent.target as ResourceErrorTarget;
4     // code error
5     const { message, filename, lineno, colno, error } = errorEvent;
6     let result: ReportDataType;
7
8     // 处理SyntaxError, stack没有lineno、colno
9     result || (result = transformData(message, filename, lineno, colno));
10    result.type = ErrorTypes.JAVASCRIPT_ERROR;
11    breadcrumb.push({
12        type: BreadCrumbTypes.CODE_ERROR,
13        category: breadcrumb.getCategory(BreadCrumbTypes.CODE_ERROR),
14        data: { ...result },
15        level: Severity.Error,
16    });
17    transportData.send(result);
18},
```

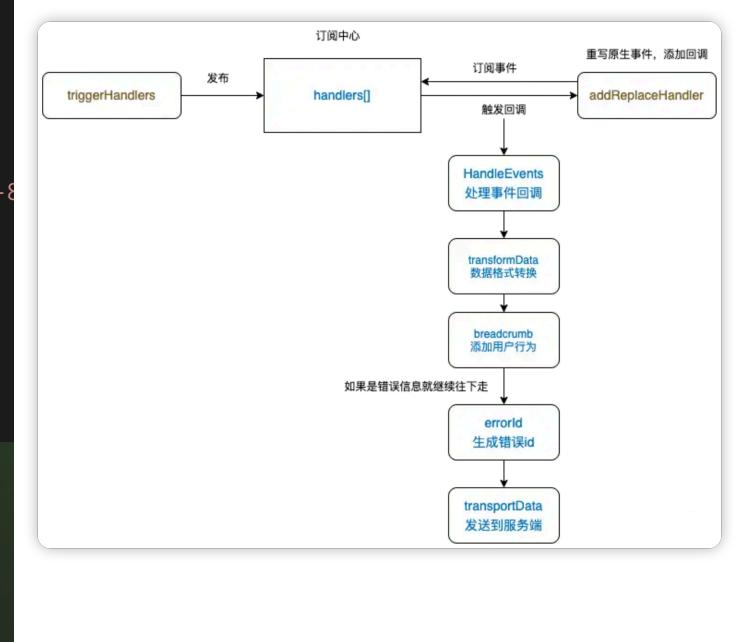
后面就是生成 errorId 和 上报方式 的流程



TransportData上报流程

```
1 // XMLHttpRequest 形式上报
2 async xhrPost(data: any, url: string) {
3   const requestFun = (): void => {
4     const xhr = new XMLHttpRequest();
5     xhr.open(EMethods.Post, url);
6     xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
7     xhr.withCredentials = true;
8     xhr.send(JSON.stringify(data));
9   };
10  // this.queue.addFn(requestFun);
11  requestFun;
12 }
```

这就是使用 `addEventListener('error')` 进行监听的全部流程



sourceMap上传流程

自定义vite-plugin-sourcemap-xk插件，将sourceMap上传到私服

```
1 import path from 'path';
2 import fs from 'fs';
3 import request from 'request';
4
5 const TAG = '[vite-plugin-sourcemap-xk]: ';
6
7 export default function vitePluginSourcemapXk(pluginOptions) {
8     return {
9         name: 'sourcemap-xk',
10        writeBundle(options: any, bundle: any) {
11            // ...获取sourcemap文件
12        },
13        async closeBundle() {
14            // ...上传sourcemap
15        },
16    };
17 }
```

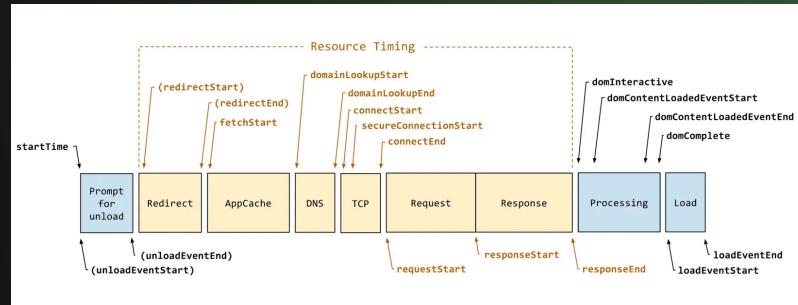
这就是使用 sourceMap 上传流程

性能监控

- BBC页面加载时长每增加1秒，用户流失10%
- Pinterest减少页面加载时长40%，提高了搜索和注册数15%
- DoubleClick发现如果移动网站加载时长超过3秒，53%的用户会放弃

网页性能指标及影响因素

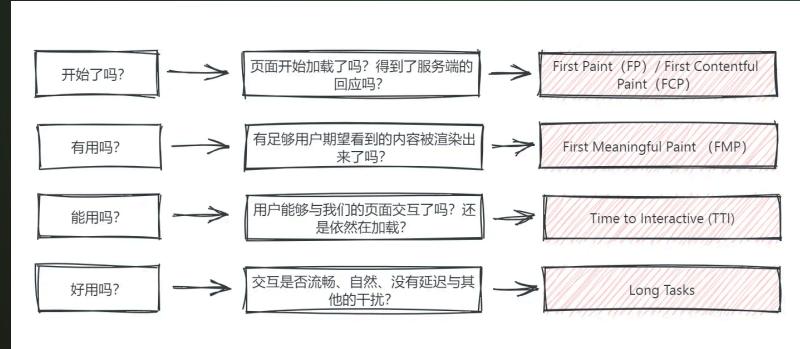
1. performance.timing
2. performance.getEntries()
3. PerformanceObserver



```
//直接往 PerformanceObserver() 入参匿名回调函数, 成功 new 了一个 PerformanceObserver 类的, 名为 observer 的对象
var observer = new PerformanceObserver(function (list, obj) {
  var entries = list.getEntries();
  for (var i = 0; i < entries.length; i++) {
    //处理“navigation”和“resource”事件
  }
});
//调用 observer 对象的 observe() 方法
observer.observe({ entryTypes: ['navigation', 'resource'] });
```

Core Web Vitals (核心网页指标)

Core Web Vitals

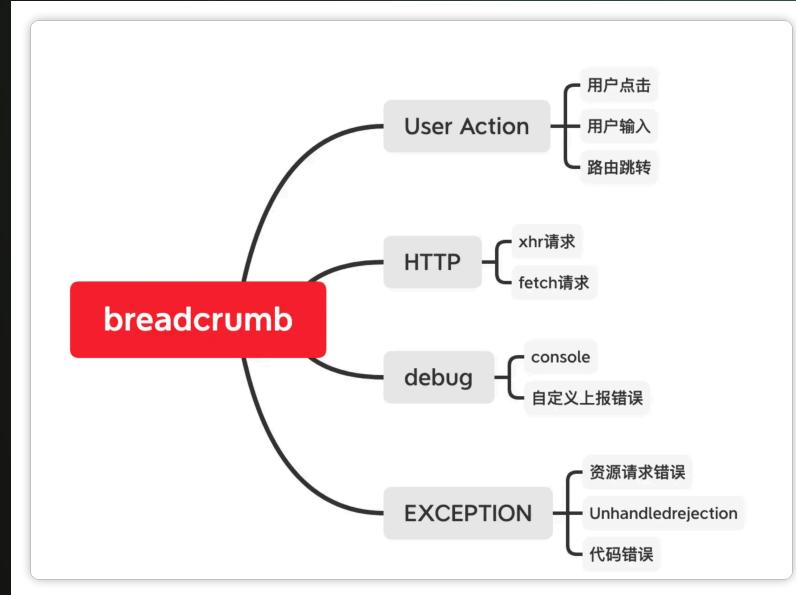


性能监控工具

- Lighthouse
- WebPageTest
- GTmetrix
- PageSpeed Insights
- Calibre

行为监控

- PV、UV
- 路由跳转
 - Hash 路由
 - History 路由
- 用户点击事件
- 用户自定义埋点
- HTTP 请求捕获
- 页面停留时间
- 访客来路
- User Agent 解析



Thanks for Listening!

