

# 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 - 包括各阶段加载耗时，一些关键性的用户体验指标等
- 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. 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

- JS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

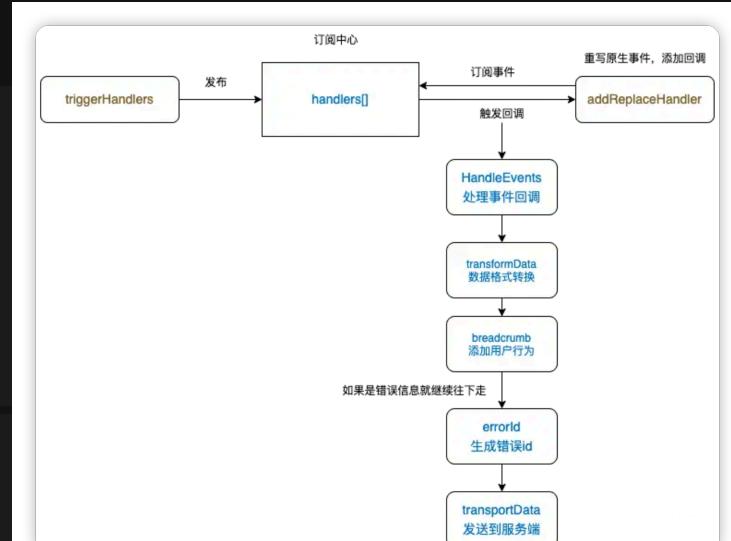
# 整体错误上报流程

以Js运行异常为例，整体上报流程如下：

```
1 // setupReplace 中添加 addReplaceHandler
2 // 替代处理的回调函数和类型
3 addReplaceHandler({
4   callback: (error) => {
5     HandleEvents.handleError(error);
6   },
7   type: EventTypes.ERROR,
8 });

// addReplaceHandler 添加订阅和函数替代
export function addReplaceHandler(handler: ReplaceHandler) {
  if (!subscribeEvent(handler)) return; // 添加订阅
  replace(handler.type as EventTypes); // 替换函数
}

// subscribeEvent 添加订阅，存在handlers订阅中心
export function subscribeEvent(handler: ReplaceHandler): boolean {
  if (!handler || getFlag(handler.type)) return false;
  setFlag(handler.type, true);
  handlers[handler.type] = handlers[handler.type] || [];
  handlers[handler.type].push(handler.callback);
}
```



# 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 }
```

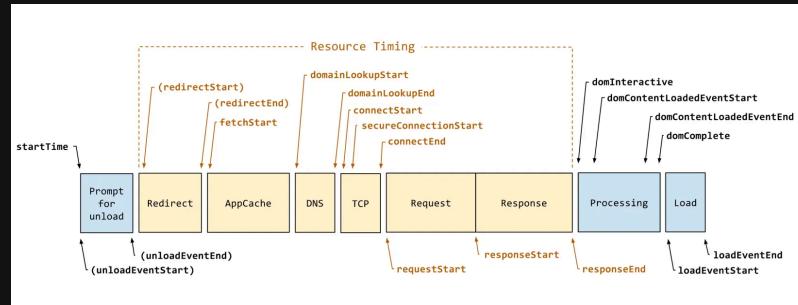
这就是使用 sourceMap 上传流程

# 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



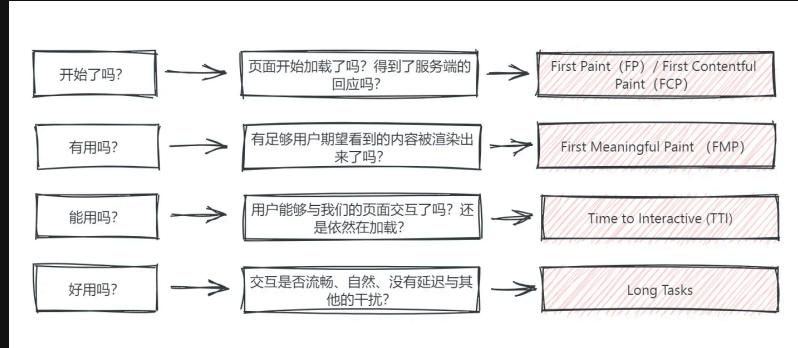
```
//直接往 PerformanceObserver() 入参匿名回调函数，成功 new 了一个 PerformanceObserver 类的，名为 observer 的对象
var observer = new PerformanceObserver(function (list, obj) {
  var entries = list.getEntries();
  for (var i = 0; i < entries.length; i++) {
    //处理“navigation”和“resource”事件
  }
});
//调用 observer 对象的 observe()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

# Core Web Vitals (核心网页指标)

## Core Web Vitals



- **Largest Contentful Paint (LCP):** 衡量加载性能。为了提供良好的用户体验，LCP 必须在网页首次开始加载后的 **2.5 秒** 内发生。
- **Interaction to Next Paint (INP):** 衡量互动。为了提供良好的用户体验，网页的 INP 不得超过 **200 毫秒**。
- **Cumulative Layout Shift (CLS):** 衡量视觉稳定性。为了提供良好的用户体验，必须将 CLS 保持在 **0.1** 或更低。

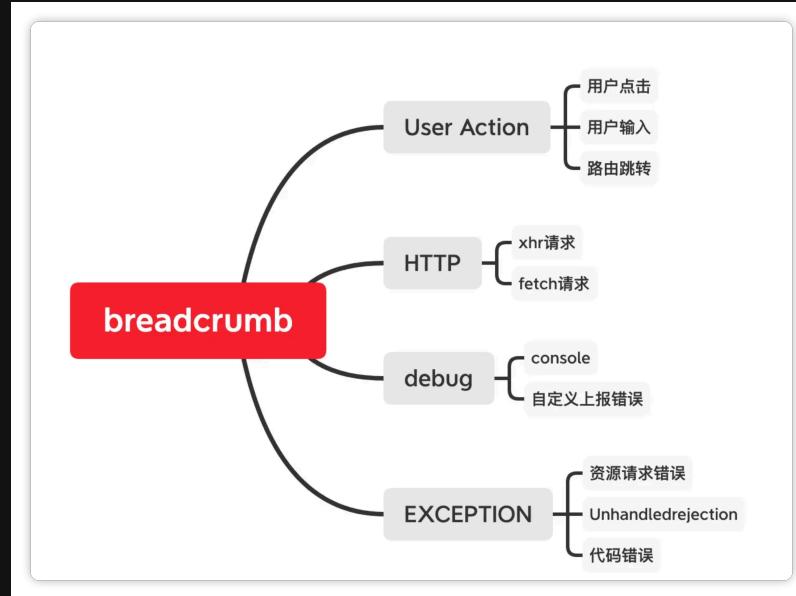


## 性能监控工具

- 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!

..