前端监控搭建之数据收集

@被删

代码发布后就结束了吗?

通常前端建立搭建监控体系，主要是为了解决两个问题：

1. 如何及时发现问题。
2. 如何快速定位并解决问题。

等用户反馈有问题的时候，就晚（完）啦！

前端监控搭建主要做些什么?

一般来说，结合开发和产品的角度来看，前端监控体系需要做的事情包括：

1. 页面的整体访问情况，包括常见的 PV、UV、用户行为上报。
2. 页面的性能情况，包括加载耗时、接口耗时统计。
3. 灰度发布与有效的监控能力，方便及时发现问题。
4. 用户反馈问题，需要足够的日志定位问题。

总体来说分成三点：数据收集 + 数据上报 + 数据监控

1

前端系统质量评估

如何评估系统质量

要进行有效地监控，首先我们需要将监控数据进行上报。传统的页面开发过程中，系统的质量通常从三方面来评估：

- 页面访问速度 → HTTP 请求速度、页面加载速度
- 页面稳定性/异常 → HTTP 异常、JavaScript 执行异常、业务逻辑异常
- 外部服务调用情况 → CGI 接口异常、服务调用异常

对前端来说，一般需要采集这些数据：HTTP 请求信息、页面加载情况、系统运行情况。

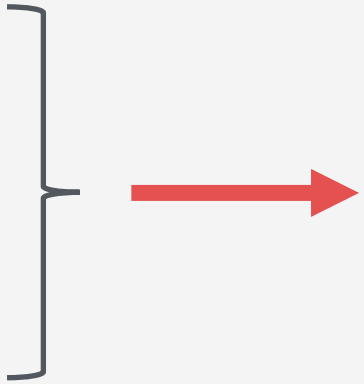
2

前端监控数据采集

将前端采集的数据进行归类

对前端来说，一般需要采集这些数据：HTTP 请求信息、页面加载情况、系统运行情况。

- HTTP 请求信息：请求耗时、请求异常
- 页面加载情况：生命周期数据、PV/UV、用户行为数据
- 系统运行情况：异常数据、用户行为数据



- 生命周期数据
- HTTP 测速数据
- 异常数据
- 用户行为数据

除此之外，我们还需要获取用户日志，用于进行反馈和问题排查。

生命周期数据--PerformanceTiming

生命周期包括页面加载的关键时间点，常常包括页面打开、更新、关闭等耗时数据。可以通过 PerformanceTiming 属性获取到相关数据：

PerformanceTiming 属性	描述
PerformanceTiming.navigationStart	当前浏览器窗口的前一个网页关闭，发生 unload 事件时的时间戳
PerformanceTiming.domLoading	返回当前网页 DOM 结构开始解析时（即 Document.readyState 属性变为“loading”、相应的readystatechange 事件触发时）的时间戳
PerformanceTiming.domInteractive	返回当前网页 DOM 结构结束解析、开始加载内嵌资源时（即 Document.readyState 属性变为“interactive”、相应的 readystatechange 事件触发时）的时间戳
PerformanceTiming.domComplete	返回当前文档解析完成（即 Document.readyState 变为“complete”且相对应的 readystatechange）被触发时的时间戳
PerformanceTiming.loadEventStart	返回该文档下，load 事件被发送时的时间戳
PerformanceTiming.loadEventEnd	返回当 load 事件结束，即加载事件完成时的时间戳

生命周期数据--MutationObserver

当初始的 HTML 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发。

由于前端框架的出现，很多时候页面的渲染交给框架来控制，可以在框架本身提供的生命周期函数中进行数据的收集。

可以使用 MutationObserver 接口：该接口提供了监听页面 DOM 树变化的能力，结合 performance 获取到具体的时间。

```
// 注册监听函数
const observer = new MutationObserver((mutations) => {
  console.log(`时间：${performance.now()}，DOM树发生了变化！有以下变化类型：`);
  for (let i = 0; i < mutations.length; i++) {
    console.log(mutations[0].type);
  }
});
// 开始监听document的节点变化
observer.observe(document, {
  childList: true,
  subtree: true,
});
```

HTTP 测速数据

请求相关的数据，我们同样可以通过 PerformanceTiming 属性获取：

PerformanceTiming 属性	描述
PerformanceTiming.connectStart	返回 HTTP 请求开始向服务器发送时的时间戳
PerformanceTiming.connectEnd	返回浏览器与服务器之间的连接建立时的时间戳，连接建立指的是所有握手和认证过程全部结束
PerformanceTiming.secureConnectionStart	返回浏览器与服务器开始安全链接的握手时的时间戳
PerformanceTiming.requestStart	返回浏览器向服务器发出 HTTP 请求时（或开始读取本地缓存时）的时间戳
PerformanceTiming.responseStart	返回浏览器从服务器收到（或从本地缓存读取）第一个字节时的时间戳

通过这些数据，我们可以观察后端服务是否稳定、是否还有优化空间。

异常数据

一般来说，脚本执行异常大多数情况下会直接导致功能不可用，因此首先需要关注系统异常的数据。

常见的前端异常包括：

1. 逻辑错误，开发实现功能的时候，逻辑梳理不符合预期；
2. 代码健壮性，代码边界情况考虑不周，异常逻辑执行出错；
3. 网络错误，用户网络情况异常、后台服务异常等错误；
4. 系统错误，代码运行环境兼容性问题导致出错；
5. 页面内容异常，缺少内容、绑定事件异常、样式异常等。

可以使用以下方法来进行拦截：

- window.onerror
- document.addEventlistener(error)
- XMLHttpRequest status

通过监听 window.onerror 事件，我们可以获取项目中的错误和分析堆栈，
将错误信息自动上报到后台服务中。

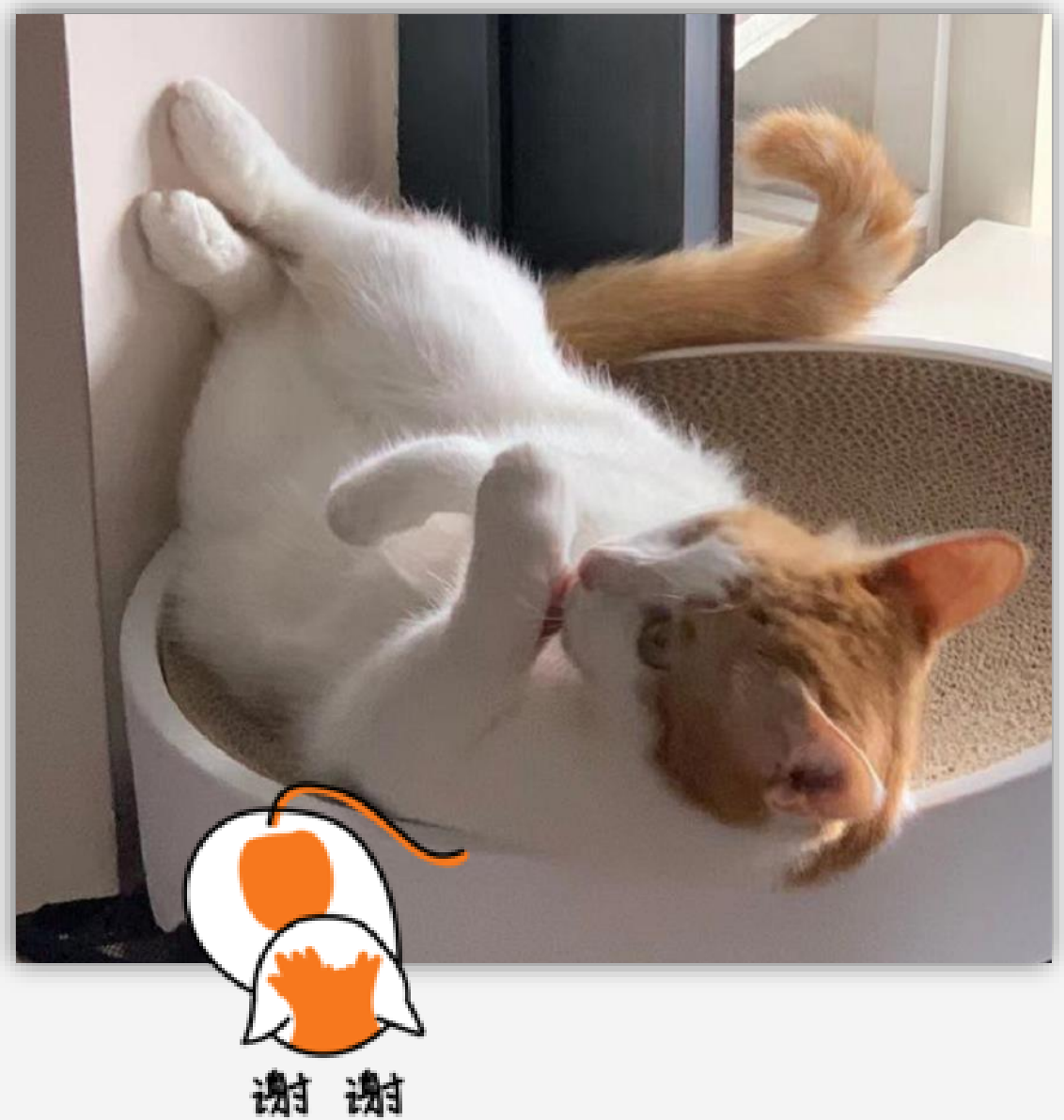
用户相关数据

用户相关数据通常用来统计分析用户行为，来针对性调整页面功能、更好地发挥页面的作用。同时，我们还可以通过一些用户交互数据，来观测系统功能是否正常。

1. 用户行为数据：
    - 页面浏览量或点击量
    - 用户在每一个页面的停留时间
    - 用户通过什么入口来访问该页面
    - 用户在页面中的一些操作行为
2. 用户日志：
    - 当系统出现异常的时候，要使用日志进行定位

数据采集之后要做些什么？

敬请期待~



Github: godbasin
@被删