监控

前端监控一般分为三种，分别为页面埋点、性能监控以及异常监控。

这一章节我们将来学习这些监控相关的内容，但是基本不会涉及到代码，只是让大家了解下前端监控该用什么方式实现。毕竟大部分公司都只是使用别了第三方的监控工具，而不是选择自己造轮子。

页面埋点

页面埋点应该是大家最常写的监控了，一般起码会监控以下几个数据：

- PV / UV
- 停留时长
- 流量来源
- 用户交互

对于这几类统计，一般的实现思路大致可以分为两种，分别为手写埋点和无埋点的方式。

相信第一种方式也是大家最常用的方式，可以自主选择需要监控的数据然后在相应的地方写入代码。这种方式的灵活性很大，但是唯一的缺点就是工作量较大，每个需要监控的地方都得插入代码。

另一种无埋点的方式基本不需要开发者手写埋点了，而是统计所有的事件并且定时上报。这种方式虽然没有前一种方式繁琐了，但是因为统计的是所有事件，所以还需要后期过滤出需要的数据。

性能监控

性能监控可以很好的帮助开发者了解在各种真实环境下，页面的性能情况是如何的。

对于性能监控来说，我们可以直接使用浏览器自带的 Performance API 来实现这个功能。

对于性能监控来说，其实我们只需要调用 `performance.getEntriesByType('navigation')` 这行代码就行了。对，你没看错，一行代码我们就可以获得页面中各种详细的性能相关信息。

```
> performance.getEntriesByType("navigation")
< ▼[PerformanceNavigationTiming] ⓘ
  ▼0: PerformanceNavigationTiming
     connectEnd: 14.159999991534278
     connectStart: 14.159999991534278
     decodedBodySize: 17503
     domComplete: 3008.5649999964517
     domContentLoadedEventEnd: 1379.350000002887
     domContentLoadedEventStart: 1035.8849999982304
     domInteractive: 1035.8499999855676
     domainLookupEnd: 14.159999991534278
     domainLookupStart: 14.159999991534278
     duration: 3010.3399999788962
     encodedBodySize: 11548
     entryType: "navigation"
     fetchStart: 14.159999991534278
     initiatorType: "navigation"
     loadEventEnd: 3010.3399999788962
     loadEventStart: 3008.585000003224
     name: "https://juejin.im/"
     nextHopProtocol: ""
     redirectCount: 0
     redirectEnd: 0
     redirectStart: 0
     requestStart: 25.439999997615814
     responseEnd: 182.87499998230487
     responseStart: 54.64499999652617
     secureConnectionStart: 0
   ▶serverTiming: []
     startTime: 0
     transferSize: 0
     type: "navigate"
     unloadEventEnd: 0
     unloadEventStart: 0
     workerStart: 0
```


我们可以发现这行代码返回了一个数组，内部包含了相当多的信息，从数据开始在网络中传输到页面加载完成都提供了相应的数据。

异常监控

对于异常监控来说，以下两种监控是必不可少的，分别是代码报错以及接口异常上报。

对于代码运行错误，通常的办法是使用 `window.onerror` 拦截报错，该方法能拦截到大部分的详细报错信息，但是也有例外

- 对于跨域的代码运行错误会显示 `Script error.` 对于这种情况我们需要给 `script` 标签添加 `crossorigin` 属性
- 对于某些浏览器可能不会显示调用栈信息，这种情况可以通过 `arguments.callee.caller` 来做栈递归

对于异步代码来说，可以使用 `catch` 的方式捕获错误，比如 `Promise` 可以直接使用 `catch` 函数，`async await` 可以使用 `try catch` 。

但是要注意线上运行的代码都是压缩过的，需要在打包时生成 sourceMap 文件便于 debug。

对于捕获的错误需要上传给服务器，通常可以通过 `img` 标签的 `src` 发起一个请求。

另外接口异常就相对来说简单了，可以列举出出错的状态码。一旦出现此类的状态码就可以立即上报出错。接口异常上报可以让开发人员迅速知道有哪些接口出现了大面积的报错，以便迅速修复问题。

小结

这一章节内容虽然不多，但是这类监控的知识网上的资料确实不多，相信能给大家一个不错的思路。