### 溯源中心前端监控方案

前端监控包含以下这个方面的监控

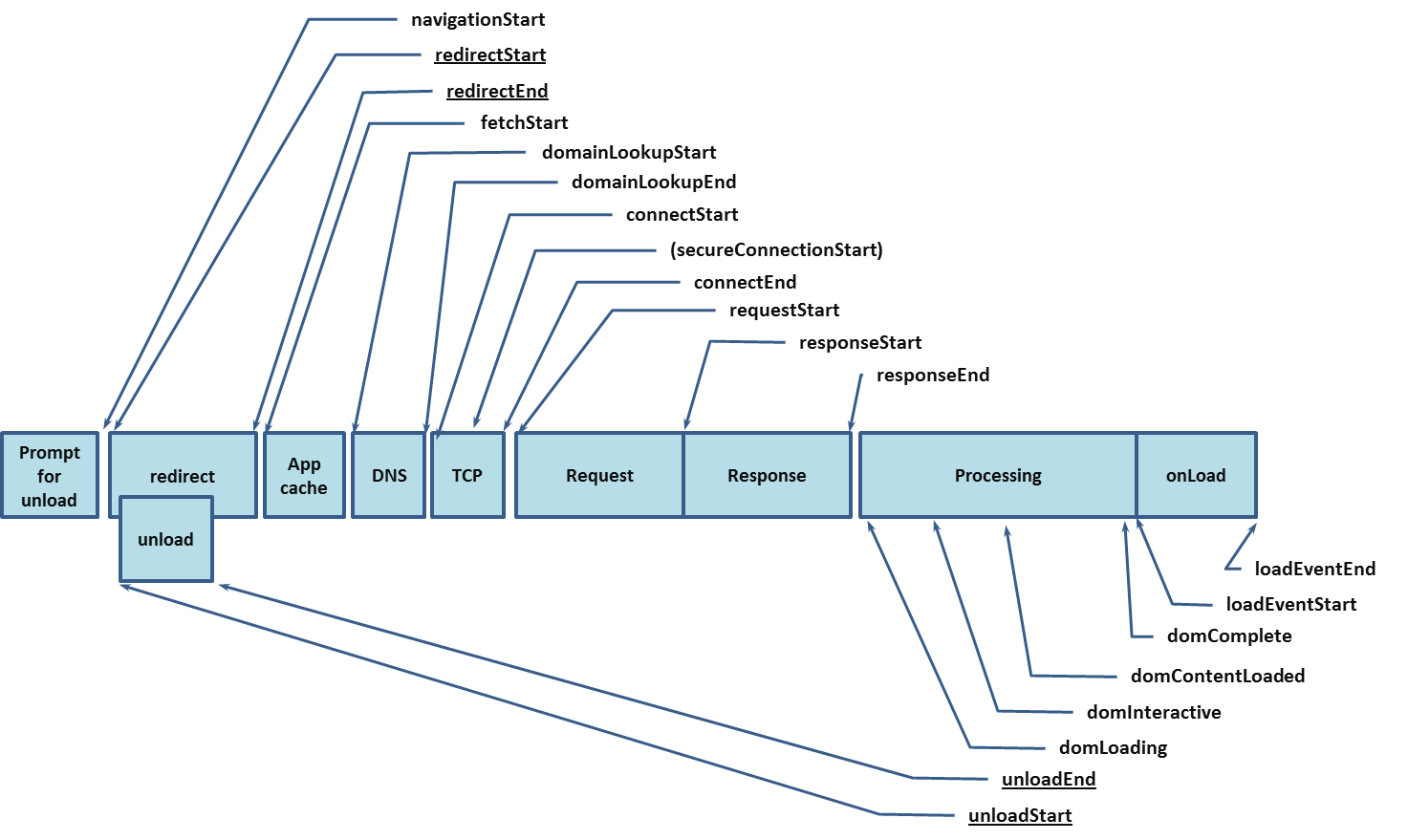
1. 页面性能的监控
2. 静态资源性能监控
3. 错误监控
4. 接口性能监控
5. 用户行为监控

分别从性能角度和用户角度进行分析监控，以下是对每种监控的原理和实现手段进行说明：

### 页面性能的监控

这个监控的主要目的是要页面从请求开始，到加载完成的各个环节耗时，一般来说会用到Performance API。Performance是前端性能监控的API。它可以检测页面中的性能，它可以检测到白屏时间、首屏时间、用户可操作的时间节点，页面总下载的时间、DNS查询的时间、TCP链接的时间等关键性能指标，页面性能监控会用到这个api的timing属性。

**Performance.time属性介绍**

  
  
**属性说明：**

navigationStart：浏览器处理当前网页的启动时间

fetchStart：浏览器发起http请求读取文档的毫秒时间戳。

domainLookupStart：域名查询开始时的时间戳。

domainLookupEnd：域名查询结束时的时间戳。

connectStart：http请求开始向服务器发送的时间戳。

connectEnd：浏览器与服务器连接建立（握手和认证过程结束）的毫秒时间戳。

requestStart：浏览器向服务器发出http请求时的时间戳。或者开始读取本地缓存时。

responseStart：浏览器从服务器（或读取本地缓存）收到第一个字节时的时间戳。

responseEnd：浏览器从服务器收到最后一个字节时的毫秒时间戳。

domLoading：浏览器开始解析网页DOM结构的时间。

domInteractive：网页dom树创建完成，开始加载内嵌资源的时间。

domContentLoadedEventStart：网页DOMContentLoaded事件发生时的时间戳。

domContentLoadedEventEnd：网页所有需要执行的脚本执行完成时的时间，domReady的时间。

domComplete：网页dom结构生成时的时间戳。

loadEventStart：当前网页load事件的回调函数开始执行的时间戳。

loadEventEnd：当前网页load事件的回调函数结束运行时的时间戳。

根据performance.timing api的相关含义，可以解读出页面加载各个环节的性能数据：

1.pervPage: t.fetchStart - t.navigationStart,表示重定向情况下，上一页面时间

2.redirect: t.responseEnd - t.redirectStart,表示页面重定向到服务器返回总耗时

3.dns: t.domainLookupEnd - t.domainLookupStart,表示DNS查找时间

4.connect: t.connectEnd - t.connectStart, 表示TCP建连时间

5.network: t.connectEnd - t.navigationStart,表示网络总耗时

6.send: t.responseStart - t.requestStart,表示浏览器从发送到接收到后端第一个返回

7.receive: t.responseEnd - t.responseStart,后端数据传输时间

8.request: t.responseEnd - t.requestStart,浏览器从请求开始到服务器完全返回数据时间

9.dom: t.domComplete - t.domLoading, 表示dom解析时间

10.loadEvent:t.loadEventEnd - t.loadEventStart,表示 loadEvent时间

11.frontend: t.loadEventEnd - t.domLoading, 表示前端解析总时间

12.load: t.loadEventEnd - t.navigationStart, 表示页面完全加载总时间

13.domReady:t.domContentLoadedEventStart - t.navigationStart, 表示domready时间

14.interactive: t.domInteractive - t.navigationStart, 表示用户可以操作界面时间

15.ttfb: t.responseStart - t.navigationStart, 表示从开始请求到浏览器收到首字节时间

**前端实现原理：**

监听 window.addEventListener('load', ()=>{}); 时间，这个时候页面加载完成，在回调函数中读取

window.performance.timing 的各个属性值，用相关的属性相减得到网页各个阶段的用时。

**页面性能监控表设计：**

字段：全部为字符串提交，红色字段为重点关注字段

1.system：所属系统

2.userName：登录的用户名

3.pageUrl:当前页面的路径

4.pervPage:表示重定向情况下，上一页面时间

5.redirect:表示页面重定向到服务器返回总耗时

6.dns: 表示DNS查找时间

7.connect: 表示TCP建连时间

8.network: 表示网络总耗时

9.send:表示浏览器从发送到接收到后端第一个返回

10.receive: 后端数据传输时间

11.request:浏览器从请求开始到服务器完全返回数据时间

12.dom: 表示dom解析时间

13.loadEvent:表示 loadEvent时间

14.frontend: 表示前端解析总时间

15.load: 表示页面完全加载总时间

16.domReady: 表示domready时间

17.interactive: 表示用户可以操作界面时间

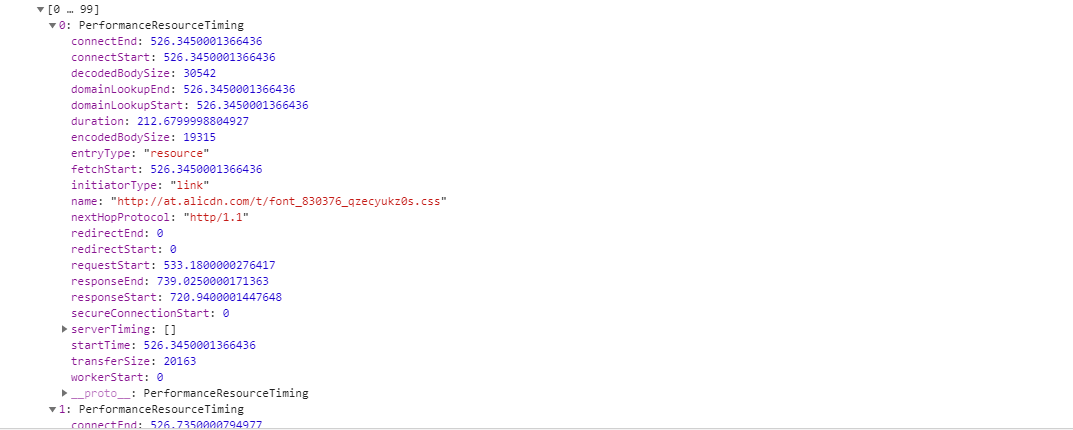
18.ttfb: 表示从开始请求到浏览器收到首字节时间

15.createtime //创建时间

### 二．静态资源监控

有时候需要单独的对静态资源进行监控，例如script，img，link标签等，这个时候需要

用到performance.getEntriesByType api，调用performance.getEntriesByType('resource')将会返回当前页面的资源数组对象，如下图形式：



通过具体时间属性相减形式得出关键时间节点：

1.initiatorType: t.initiatorType,表文件类型：img,script,link等

2.name: t.name,表示文件名字

3.duration: t.duration,表示加载总时间

4.redirect: t.redirectEnd - t.redirectStart, 表示重定向时间

5.dns: t.domainLookupEnd - t.domainLookupStart,表示DNS解析时间

6.connect: t.connectEnd - t.connectStart, 表示TCP建连时间

7.network: t.connectEnd - t.startTime, 表示网络总耗时间

8.send: t.responseStart - t.requestStart, 表示发送开始到接受第一个返回时间

9.receive: t.responseEnd - t.responseStart, 表示接收总时间，服务器传输数据时间

10.request: t.responseEnd - t.requestStart, 表示请求总时间

11.ttfb: t.responseStart - t.requestStart, 表示首字节时间

**前端实现原理**

监听 window.addEventListener('load', ()=>{}); 时间，这个时候页面加载完成，在回调函数中调用

performance.getEntriesByType('resource')，返回数组，对数组进行遍历加工，最终得到需要的数据

**静态资源监控表设计：**

字段：红色字段为重点关注字段

1.system：所属系统

2.userName：登录的用户名

3.pageUrl:当前页面的路径

4.initiatorType: t.initiatorType,表文件类型：img,script,link等

5.name: t.name,表示文件名字

6.duration: t.duration,表示加载总时间

7.redirect: t.redirectEnd - t.redirectStart, 表示重定向时间

8.dns: t.domainLookupEnd - t.domainLookupStart,表示DNS解析时间

9.connect: t.connectEnd - t.connectStart, 表示TCP建连时间

10.network: t.connectEnd - t.startTime, 表示网络总耗时间

11.send: t.responseStart - t.requestStart, 表示发送开始到接受第一个返回时间

12.receive: t.responseEnd - t.responseStart, 表示接收总时间，服务器传输数据时间

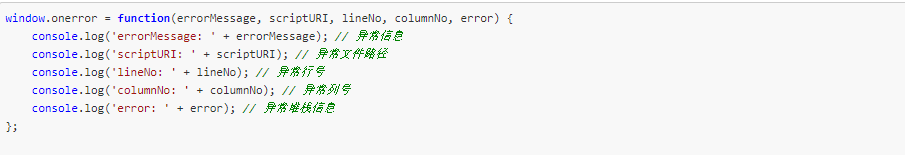
13.request: t.responseEnd - t.requestStart, 表示请求总时间

14.ttfb: t.responseStart - t.requestStart, 表示首字节时间

15.createtime //创建时间

### 三．错误信息捕获

经过了大量测试及联调的项目在有些时候还是会有十分隐蔽的bug存在，这种复杂而又不可预见性的问题唯有通过完善的监控机制才能有效的减少其带来的损失，因此对于直面用户的前端而言，异常捕获与上报是至关重要，window.onerror提供了全局监听异常的能力，通常情况下监听其回调事件，即可获得有用的参数：



**前端实现**

通过监听全局错误对象window.onerror 在回调函数中，取得错误的信息，位置（列号，行好），错误文件名字进行保存。在进一步的实现中，最好的能把错误位置打源代码上下6行，记录下来。

**错误监控表设计：**

字段：红色字段为重点关注字段  
1.system：系统名称

2.userName：用户名称

3.pageUrl,表示所在页面的url

4.fileName //错误的文件名字

5.message //错误信息

6.row //错误文件的行号

7.col //错误文件的列号

8.fileCode //错误文件源代码

9.createtime //创建时间

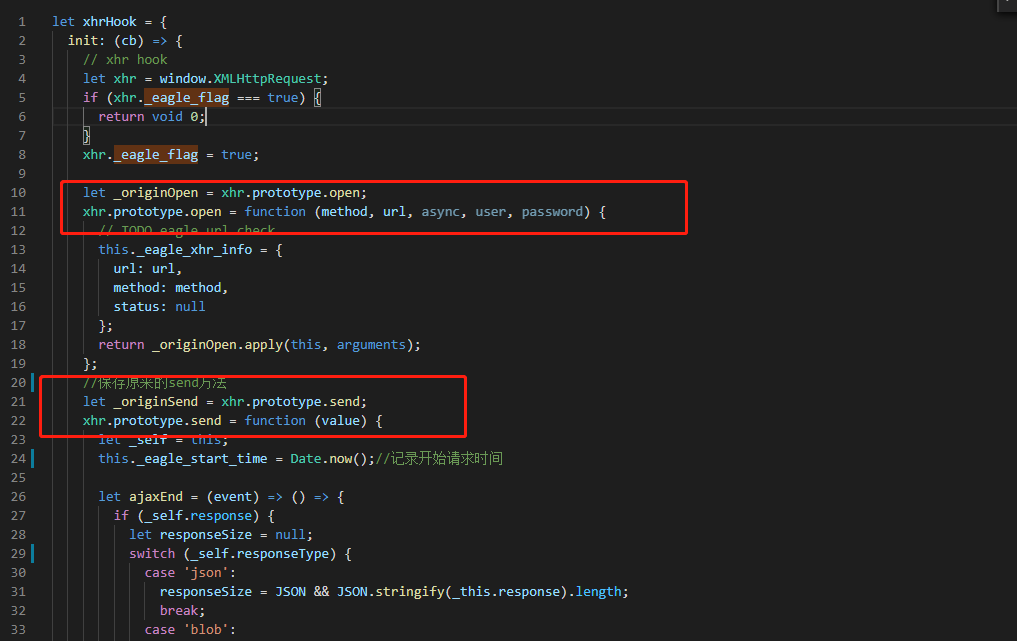
### 接口请求监控

前端对应接口的监控有非常重要的意义，现在行业流行的单页应用非常依赖接口的速度

对接口的监控可以定位系统流程问题，定位到具体哪一个接口有问题。前端现在还没有对接口监控专门的api处理，需要重写送接口的api，在重写的api上加上发送的时间，在成功回调的时候减去发送的时间，得到整个接口的请求时间

**前端实现**

通过装饰器模式，缓存原来的方法，记录开始发送时间，监听完成时刻，最终算出接口用时。



**接口监控表设计：**

1.system //所属系统

2.userName //登录的用户

3.pageUrl //所在的页面url

4.method //请求用的方法 get，post，delete

5.url //请求接口的路径

6.duration //请求接口的耗时

7.status //请求接口的状态

8.event //接口回调event 对象

9.createtime //创建时间

10.param //入参

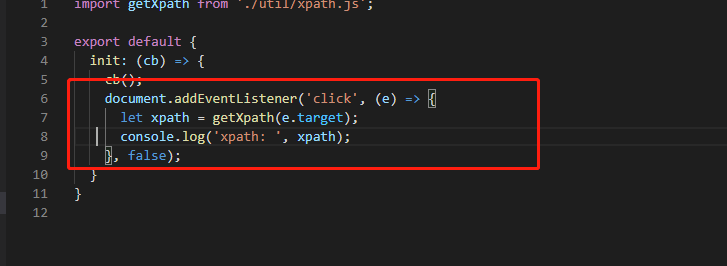
### 用户行为监控

记录用户行为路径，对于产品优化具有重要意义，前端通过监控用户点击的元素

记录用户行为路径，用户点击按钮的位置可以用xpath来描述。

**前端实现:**

点击每个元素记录其在页面的xpath



**用户路径表设计：**

1.system //所属系统

2.userName //登录的用户

3.pageUrl //所在的页面url

4.xpath //页面元素xpath

5.classname //页面元素classnamae

6.tagName//页面元素tagname

7.innerText //页面元素innerText

8.createtime //创建时间