如何精確統計頁面停留時長

今日頭條技術 前端開發 今天

> 作者：今日頭條技術
> 鏈接：https://techblog.toutiao.com/2018/06/05/ru-he-jing-que-tong-ji-ye-mian-ting-liu-shi-chang/

1、背景

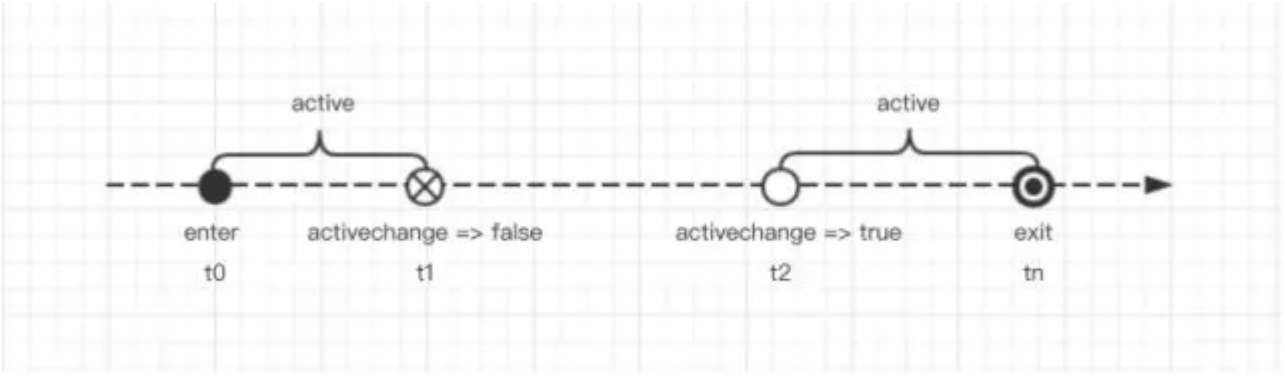
頁面停留時間（Time on Page）簡稱Tp，是網站分析中很常見的一個指標，用於反映用戶在某些頁面上停留時間的長短，傳統的Tp統計方法會存在一定的統計盲區，比如無法監控單頁應用，沒有考慮用戶切換Tab、最小化窗口等操作場景。基於上述背景，重新調研和實現了精確統計頁面停留時長的方案，需要兼容單頁應用和多頁應用，並且不耦合或入侵業務代碼。

2、分析

我們可以把一個頁面生命週期抽象為三個動作：「進入」、「活躍狀態切換」、「離開」

动作	触发行为
进入	首次加载、页面跳转、刷新、浏览器前进后退
活跃状态切换	页面失去焦点/获得焦点、切换窗口最小化、切换浏览器tab、电脑睡眠和唤醒
离开	关闭窗口、页面跳转、刷新、浏览器前进后退

如下圖，計算頁面停留時長既如何監控這三個動作，然後在對應觸發的事件中記錄時間戳，比如要統計活躍停留時長就把active 區間相加即可，要統計總時長既tn -t0 。



2.1 如何監聽頁面的進入和離開?

對於常規頁面的首次加載、頁面關閉、刷新等操作都可以通過window.onload 和 window.onbeforeunload 事件來監聽頁面進入和離開，瀏覽器前進後退可以通過pageshow 和 pagehide 處理。

- load / beforeunload

- pageshow / pagehide

对于单页应用内部的跳转可以转化为两个问题：

- 监听路由变化
- 判断变化的URL是否为不同页面 。

2.1.1 监听路由变化

目前主流的单页应用大部分都是基于 browserHistory (history api) 或者 hashHistory 来做路由处理，我们可以通过监听路由变化来判断页面是否有可能切换。注意是有可能切换，因为URL发生变化不代表页面一定切换，具体的路由配置是由业务决定的（既URL和页面的匹配规则）。

browserHistory

路由的变化本质都会调用 History.pushState() 或 History.replaceState()，能监听到这两个事件就能知道。通过 popstate 事件能解决一半问题，因为 popstate 只会在浏览器前进后退的时候触发，当调用 history.pushState() or history.replaceState() 的时候并不会触发。

> The popstate event is fired when the active history entry changes. If the history entry being activated was created by a call to history.pushState() or was affected by a call to history.replaceState(), the popstate event's state property contains a copy of the history entry's state object.
>
> Note that just calling history.pushState() or history.replaceState() won't trigger apopstateevent. The popstate event will be triggered by doing a browser action such as a click on the back or forward button (or calling。history.back() or history.forward() in JavaScript).

这里需要通过猴子补丁(Monkeypatch)解决，运行时重写 history.pushState 和 history.replaceState 方法：

```
let _wr =  function (type) {
  let orig = window.history[type]
  return  function () {
    let rv = orig.apply(this, arguments)
    let e = new Event(type.toLowerCase())
    e.arguments = arguments
    window.dispatchEvent(e)
    return rv
  }}
window.history.pushState = _wr('pushState')
window.history.replaceState = _wr('replaceState')
window.addEventListener('pushstate',  function (event) {})
window.addEventListener('replacestate',  function (event) {})
```

hashHistory

hashHistory 的实现是基于 hash 的变化，hash 的变化可以通过 hashchange 来监听

2.1.2 判断URL是否为不同页面

方案1: 客户端定义

通过业务方在初始化的时候配置页面规则，然后JS通过URL匹配不同的规则来区分不同的页面，这种方案在客户端数据上报的时候就已经明确了不同的页面，伪代码：

```
new Tracer({
  rules: [
    { path: '/index' },
    { path: '/detail/:id' },
    { path: '/user', query: {tab: 'profile'} }
  ])
```

方案2: 数据分析平台定义

假设我们最终上报后有一个数据分析平台来展现，我们可以在类似数据平台来配置页面规则，这样在客户端实现的代码逻辑就不需要区分页面，而是每次URL发生变化就将数据上报，最终通过数据平台配置的页面URL规则来求和、过滤数据等。

当数据展现平台不支持配置URL规则来区分页面的时候，可以采用方案1；当有数据平台支持的时候采用方案2更合理；

2.1.3 对于页面进入和离开相关事件整理

	首次加载	关闭窗口	刷新	页面跳转	浏览器前进后退
单页（browserHistory）	load	beforeunload	load / beforeunload	pushstate / replacestate	popstate
单页（hashHistory）	load	beforeunload	load / beforeunload	hashchange	hashchange
多页	load	beforeunload	load / beforeunload	load / beforeunload	pageshow / pagehide

2.2 如何监听页面活跃状态切换?

可以通过 Page Visibility API 以及在 window 上声明 onblur/onfocus 事件来处理。

2.2.1 Page Visibility API

一个网页的可见状态可以通过 Page Visibility API 获取，比如当用户 切换浏览器Tab、最小化窗口、电脑睡眠 的时候，系统API会派发一个当前页面可见状态变化的 visibilitychange 事件，然后在事件绑定函数中通过 document.hidden 或者 document.visibilityState 读取当前状态。

```
document.addEventListener('visibilitychange',  function (event) {
  console.log(document.hidden, document.visibilityState)})
```

2.2.2 onblur/onfocus

可以通过 Page Visibility API 以及在 window 上声明 onblur/onfocus 事件来处理。对于PC端来说，除了监听上述相关事件外，还可以考虑监听鼠标行为，比如当一定时间内鼠标没有操作则认为用户处于非活跃状态。

2.3 什么时机上报数据?

2.3.1 页面离开时上报

对于页面刷新或者关闭窗口触发的操作可能会造成数据丢失

2.3.2 下次打开页面时上报

会丢失历史访问记录中的最后一个页面数据

目前采用的方案2，对于单页内部跳转是即时上报，对于单页/多页应用触发 window.onbeforeunload 事件的时候会把当前页面数据暂存在 localStorage 中，当用户下次进入页面的时候会把暂存数据上报。有个细节问题，如果用户下次打开页面是在第二天，对于统计当天的活跃时长会有一定的误差，所以在数据上报的同时会把该条数据的页面进入时间/离开时间带上。

3、设计

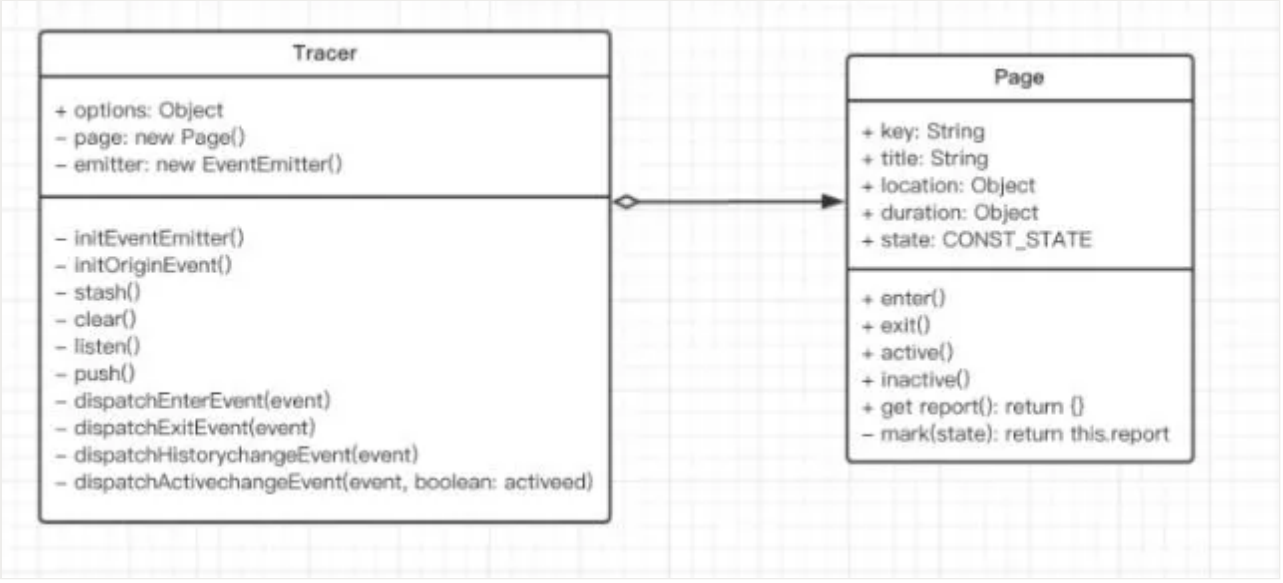
3.1 UML类关系图

Tracer

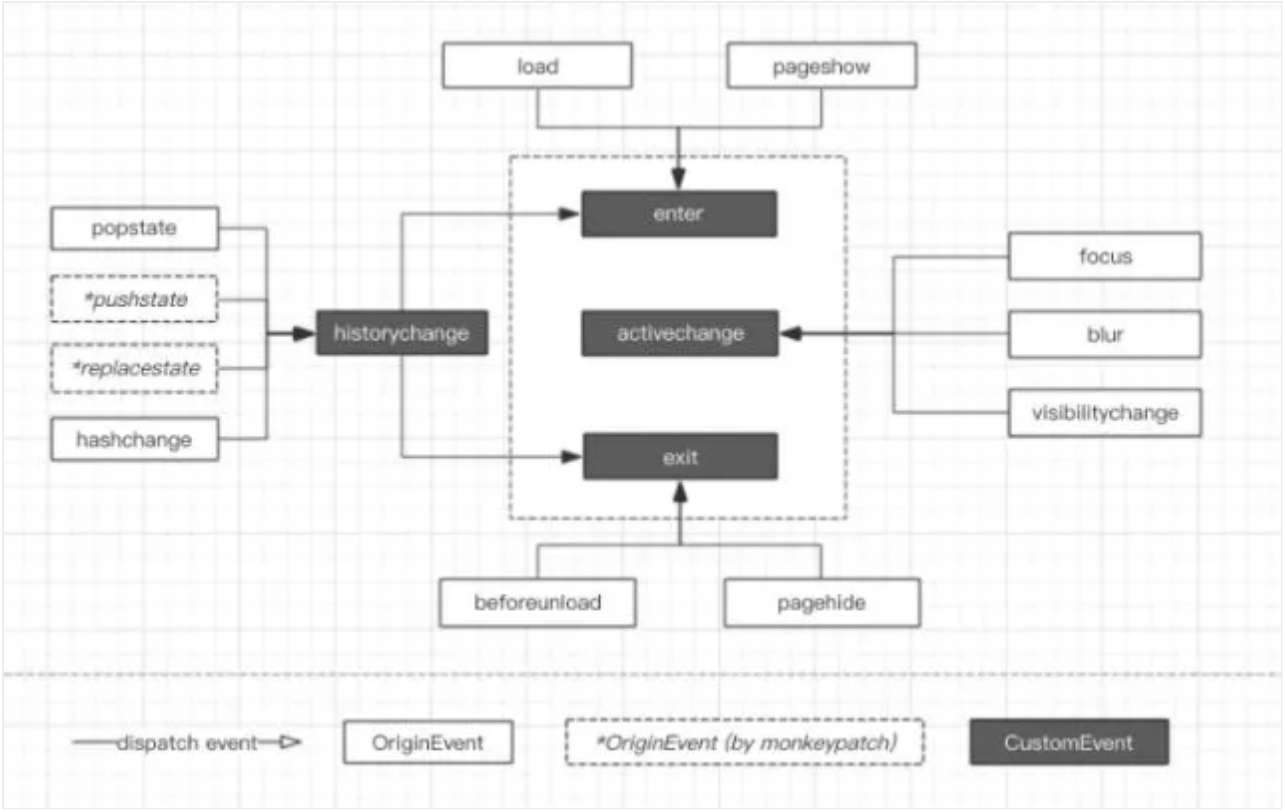
核心类，用来实例化一个监控，对原生事件和自定义事件的封装，监听 enter activechange exit 事件来操作当前 Page 实例。

> P.S. 取名来自暴雪旗下游戏守望先锋英雄猎空(Tracer)，直译为：追踪者。

Page
页面的抽象类，用来实例化一个页面，封装了 enter exit active inactive 等操作，内部通过 state 属性来维护当前页面状态。



3.2 事件派发关系图



4、兼容性

Desktop

Chrome	Edge	Firefox	IE	Opera	Safari
33	Yes	18	11	15	7

Mobile

Android	iOS
Android 4+	iOS 8+

5、思考

对于页面停留时长的定义可能在不同场景会有差异，比如内部业务系统或者OA系统，产品可能更关心用户在页面的活跃时长；而对于资讯类型的产品，页面可见时长会更有价值。单一的数据对业务分析是有限的，所以在具体的代码实过程中我们会把停留时长分三个指标，这样能更好的帮助产品/运营分析。

- active 页面活跃时长
- visible 页面可见时长 //仅支持Desktop
- duration 页面总停留时长

6、参考

- https://developer.mozilla.org/en-US/docs/Web/API/WindowEventHandlers/onhashchange
- https://developer.mozilla.org/en-US/docs/Web/Events/popstate
- https://developer.mozilla.org/en-US/docs/Web/API/PageVisibilityAPI
- https://stackoverflow.com/questions/4570093/how-to-get-notified-about-changes-of-the-history-via-history-pushstate

推荐↓↓↓

Web开发

阅读原文

喜欢此内容的人还喜欢

一行能装逼的JavaScript代码，我终于忍不住风骚了...

Web开发

Vue超好玩的新特性：在CSS中使用JS变量

Web开发

2021年React学习路线图

前端之巅