前端监控脚本

# JSniffer

嗅探JavaScript 异常，捕获异常信息：

* 行号
* 出错文件
* 错误消息
* 错误上下文（函数体，调用过程）

# DOMLint

处理浏览器解析好的DOM结构，检测异常情况：

* DOCTYPE
* 文档编码
* 文档标题
* 外部JavaScript, CSS 的编码设置
* HTTPS 页面中是否有引用 HTTP 资源
* 重复 ID
* 元素异常嵌套情况（inline内嵌block，不允许自我嵌套的元素）
* Lable 标签是否有正确设置 for 属性
* Img 元素是否有设置 alt, width, height 等属性

统计页面资源引用情况：

* Img, JavaScript, CSS, flash 的使用情况
* 文档大小

# HTMLint

在DOMReady后再请求一次当前地址，获得当前页面的HTML源码。然后解析源码：

* DOCTYPE
* Charset 是否在head中第一个设置
* Title 是否有设置
* 重复ID检测
* 标签的闭合情况
* …

通过再发一次当前页面地址的GET请求，获取HTML源码的方式有以下风险：

1. PV 倍增，服务器压力倍增
2. 无法获取POST提交的页面的源码
3. 服务端设置的token导致重复请求，而使获取的源码不一致
4. 重复请求对业务可能的影响

所以这种获取源码的方式只适用于使用GET请求访问、仅用于信息展示作用的页面。

另一种方法与DOM类似，取出的内容是被解析过的结果：

document.getElementsByTagName('html')[0].outerHTML

document.documentElement.outerHTML

终极方法是实现客户端插件，直接读取用户的浏览器缓存。

<http://www.mnot.net/cache_docs/> Caching Tutorial

<http://www.mnot.net/javascript/xmlhttprequest/cache.html> XMLHttpRequest Caching Tests

<http://www.web-caching.com/mnot_tutorial/how.html>

<http://discuss.joelonsoftware.com/default.asp?joel.3.193583.5> Get html source in javascript

# CSSniffer

基于浏览器DOM嗅探整个文档使用背景图的情况，而无需重新请求加载源码。

# CSSLint

有两种检测机制：

* 基于字符串简单解析源码，找出背景图的使用情况
* CSS Parse 详细解析源码，找出：
  + 背景图
  + 硬编码
  + CSS hacks
  + 基于 Alice 规范的检测。

# JSLint

对于前端监控系统来说，目前实用价值不大，可作为兴趣研究项目。

# See Also

<https://damnit.jupiterit.com/home/learn>