

# 前端代码异常监控方案window.onerror

我是开发微信图文页一名普通的码农。近期加班加点上线非常重要的广告功能：



appmsg.js

```
//其他功能
```

```
//广告功能
```

```
addContact.onclick = function(){
```

```
  //关注
```

```
}
```

[rapheal.sinaapp.com](http://rapheal.sinaapp.com)

底部的广告区域有关注公众号的按钮，用户点击之后就会给广告主带来粉丝，给文章所有者带来广告收入。某天，码农心血来潮，了解一下每篇文章的图片都来自什么域名，于是加了一段统计脚本....



appmsg.js

//其他功能

```
//加入图片域名识别
var imgs = document.getElementsByTagName("img");
for (var i = 0, len = imgs.length; i < len; ++i){
    //imgs[i].getAttribute("src") balblabla
}
```

//广告功能

```
addContact.onclick = function(){
    //关注
}
```

[rapheal.sinaapp.com](http://rapheal.sinaapp.com)

如此简单的for循环能难得了我，测试啥，直接上线！

## 投诉来了

下午15:00上完线，下班后突然收到一堆同事电话：我们这边发现广告的关注点不动了，用户好多投诉进来了，看到你15:00上了线，快看看有什么问题！在家VPN简单看了看代码，知道真相后简直无法直视：



appmsg.js

//其他功能

```
//加入图片域名识别
var imgs = document.getElementsByTagName("img");
for (var i = 0, len = imgs.length; i < l; ++i){
    //imgs[i].getAttribute("src") balblabla
}
```

//广告功能

```
addContact.onclick = function()
    //关注
}
```

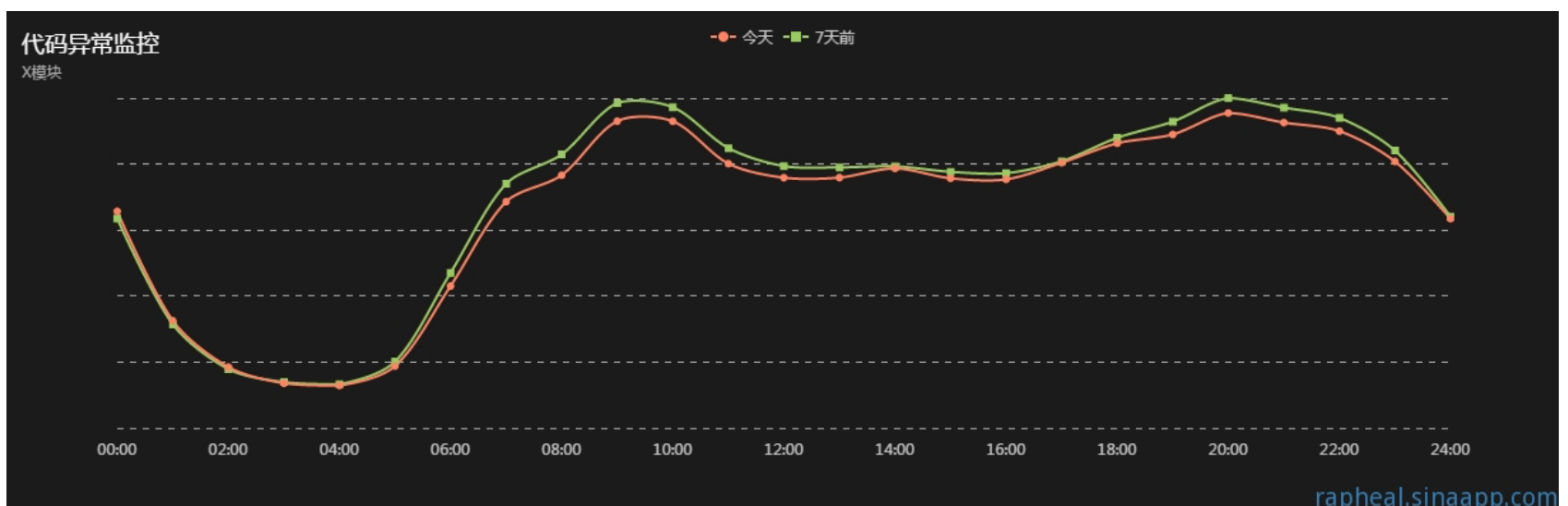


rapheal.sinaapp.com

把变量len误写成l，导致下边的js脚本不执行了！

## 反思

对于写代码这件事来说，我们几乎不能避免自己出bug。那我们如果能够在用户侧部署监控，看着用户在我们面前“出错”的话，我们就能很快发现问题并及时处理。假如我们能监控到图文页一天异常发生量，我们在发送异常的时候往服务器上报异常信息，服务器就统计出每一分钟异常的总数：



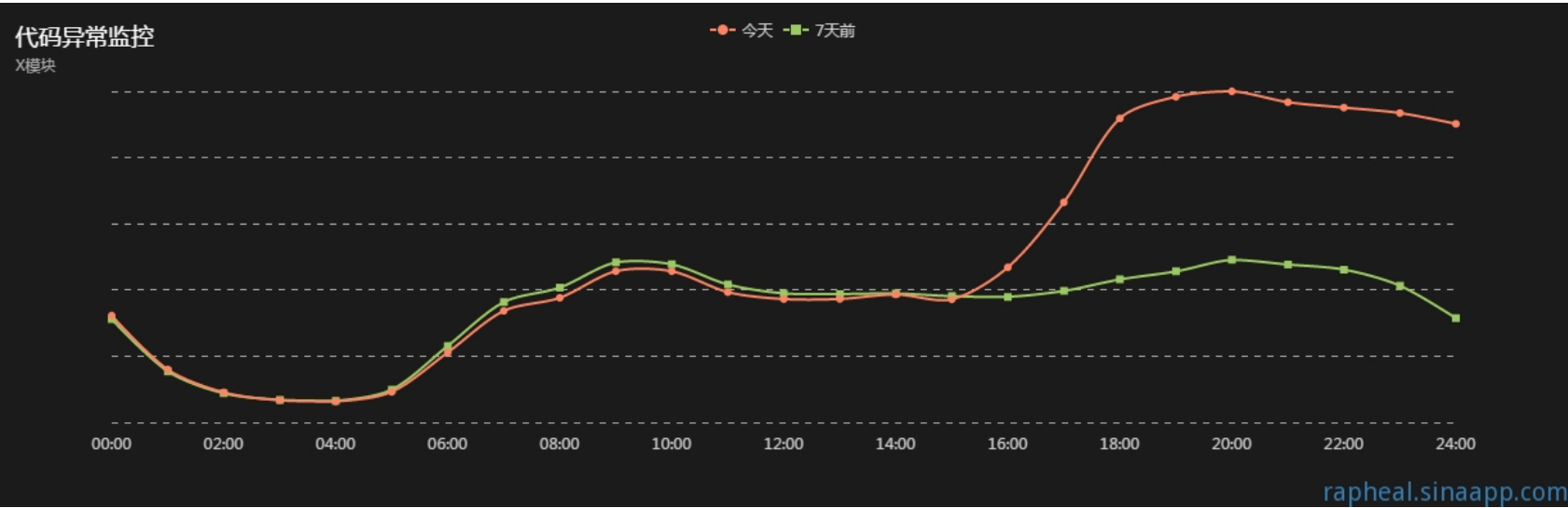
rapheal.sinaapp.com

那我们可以非常直观地看到发生异常的量，例如上图的例子就可以看出，今天跟上周的异常量是吻合的，没有特别的突增或者突减（当然异常减少是我

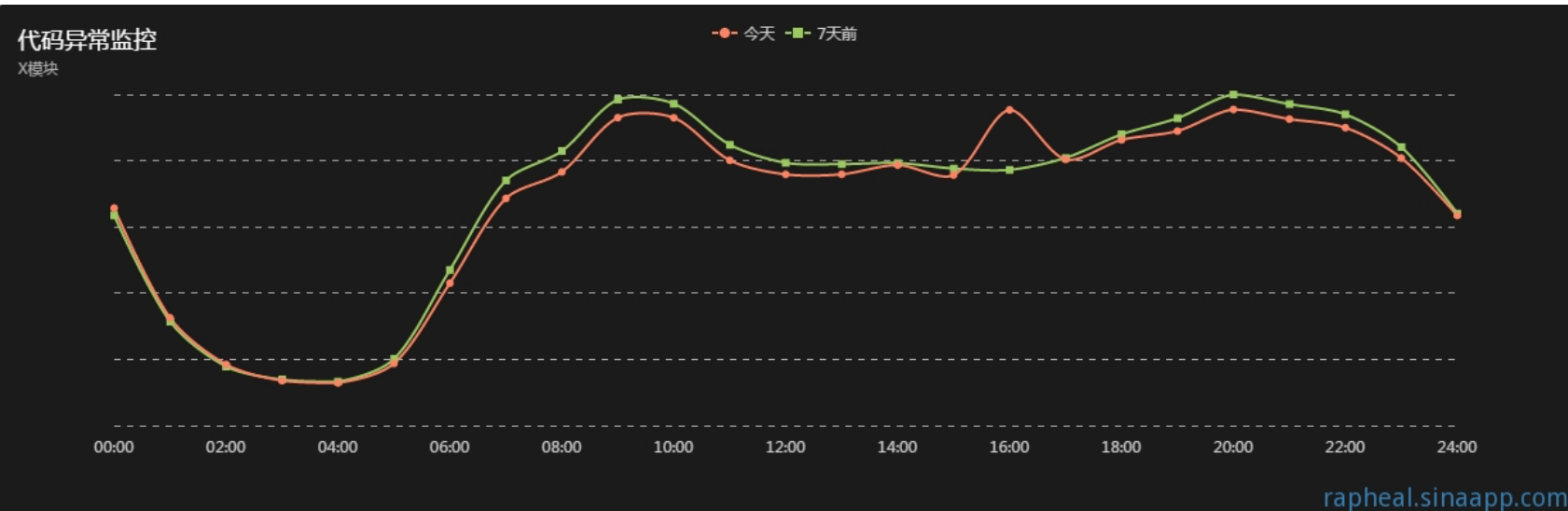
们需要去做到的！但是后边会讲到如用户浏览器的Javascript插件如果运行时出错，也会被我们捕捉到，所以实际上很难把异常数清零）。

## 上线

如果有了上边的监控，我们能做什么呢？回到15:00，我上线后，后台收到了非常多的异常上报，于是：



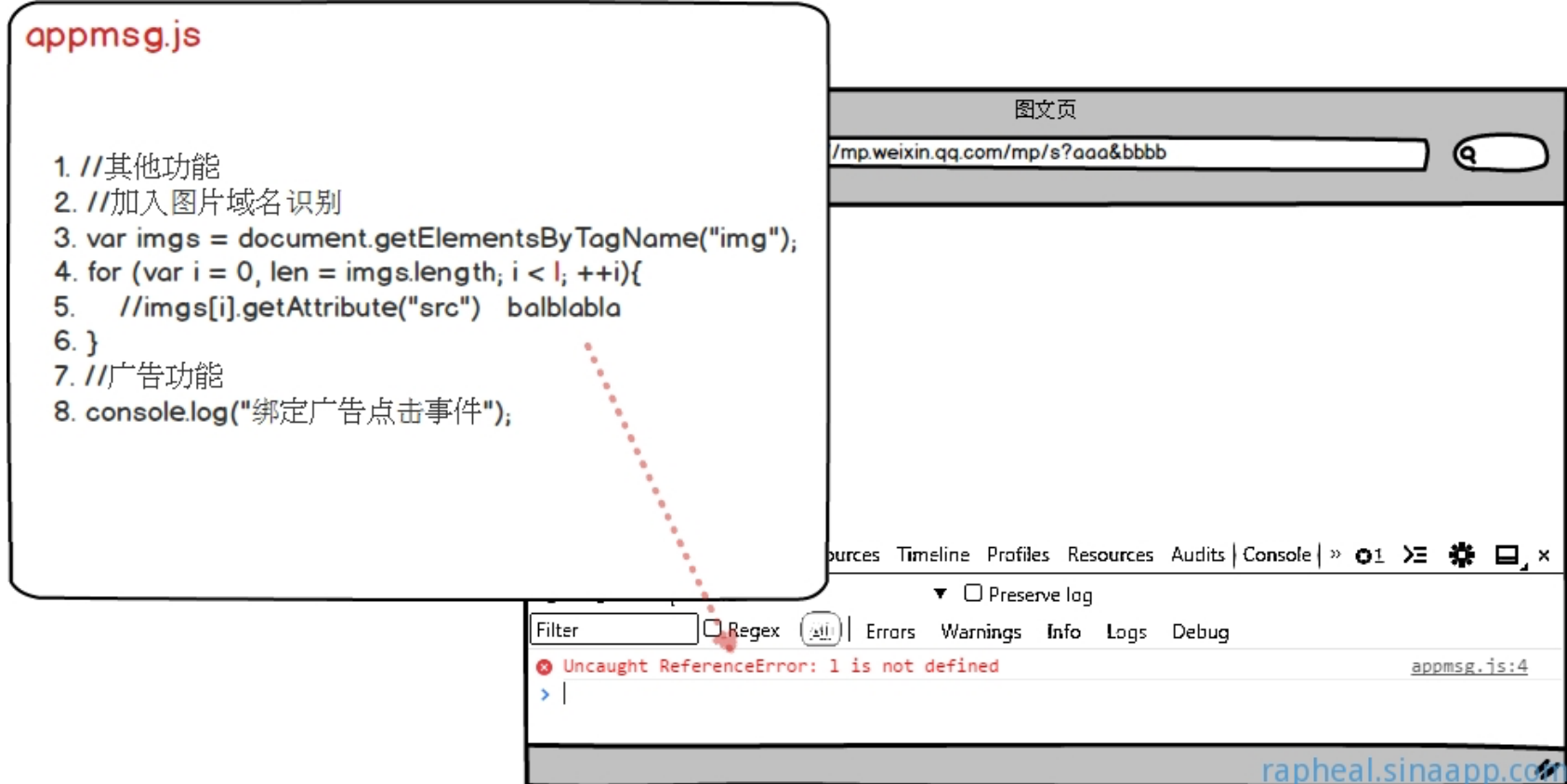
这时候后台发现监控曲线已经偏离7天前（或者昨天）超过一定的百分比（或者数量），立马发短信发微信给负责人，在用户投诉之前，负责人已经知道这个异常现象的发生，只要及时做修复，就可以恢复正常：



在上线的时候会有一个异常数的飙升，在修复后，异常数恢复到正常范围内。

## 如何检测前端异常

那剩下的问题就是如何检测前端的异常，先看看刚刚那段代码在浏览器的出错展现：



一般语法错误以及运行时错误，浏览器都会在console里边体现出错误信息，以及出错的文件，行号，堆栈信息。

来到这里，我们要定义一下本文说到的前端代码异常是什么意思。前端代码异常指的是以下两种情况：

1. JS脚本里边存着语法错误；
2. JS脚本在运行时发生错误。

有什么方法可以抓到这个错误，有两个方案：

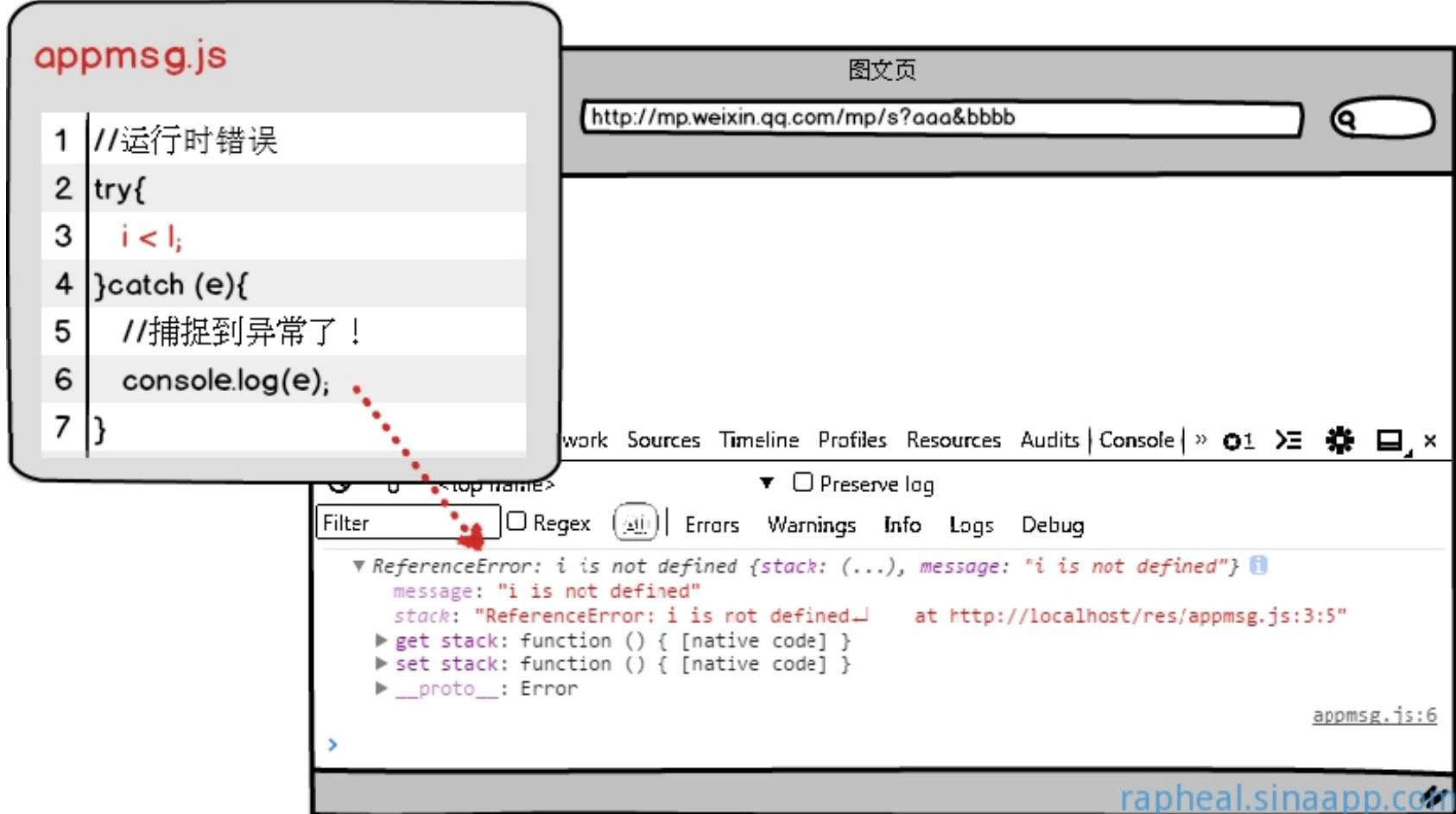
1. try, catch方案。你可以针对某个代码块使用try,catch包装，这个代码块运行时出错时能在catch块里边捕捉到。
2. window.onerror方案。也可以通过window.addEventListener("error", function(evt){}), 这个方法能捕捉到语法错误跟运行时错误，同时还能知道出错的信息，以及出错的文件，行号，列号。

上边只是简单的说了一下方案，我在接下来的2个小节来讨论一下两个方案实现细节。

## try,catch

我们可以通过对代码块加入一个try,catch块来抓出错信息：





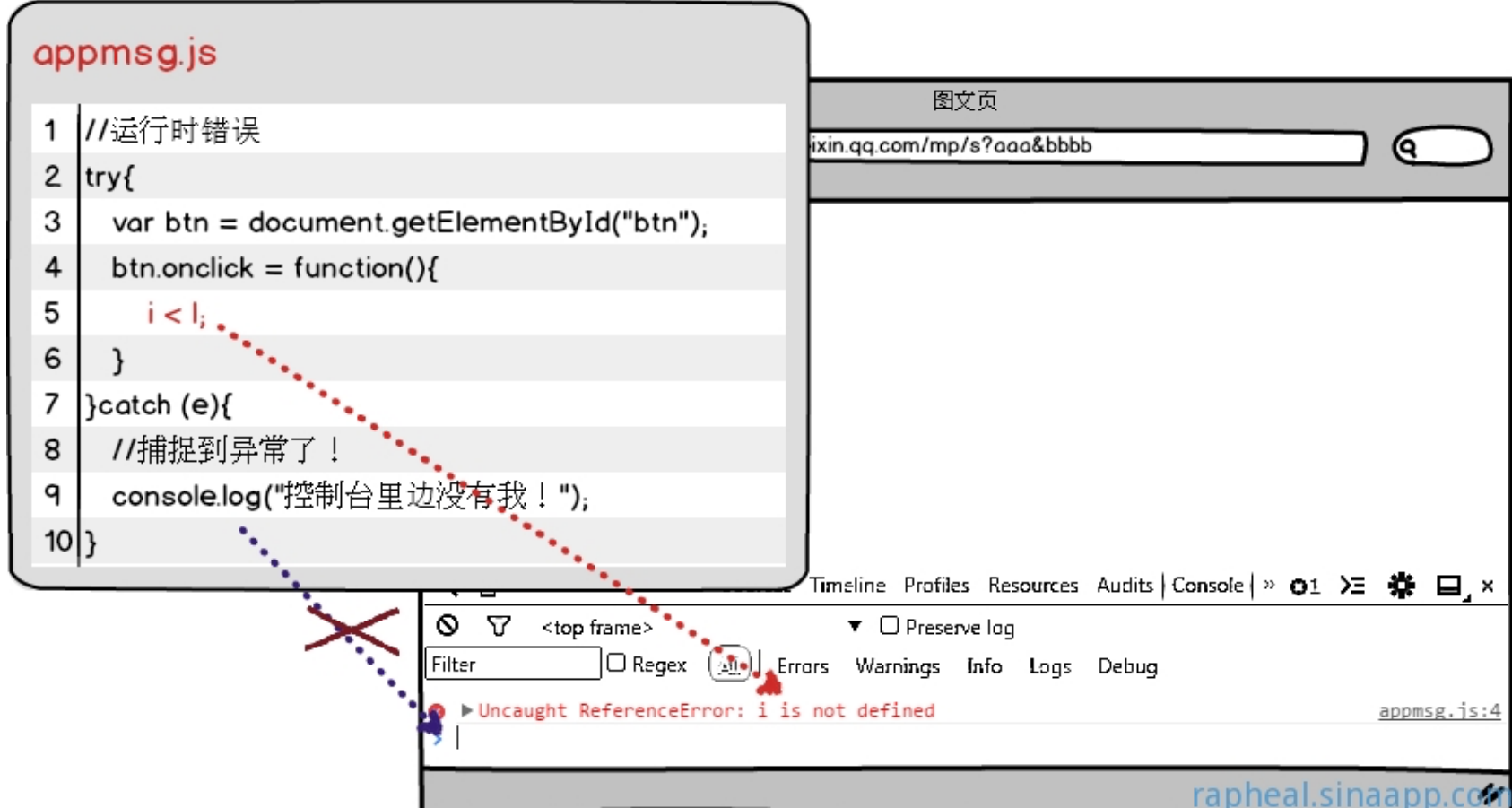
从console可以看到，try,catch能够知道出错的信息，并且也有堆栈信息可以知道在哪个文件第几行第几列发生错误。

但是try,catch的方案有2个缺点：

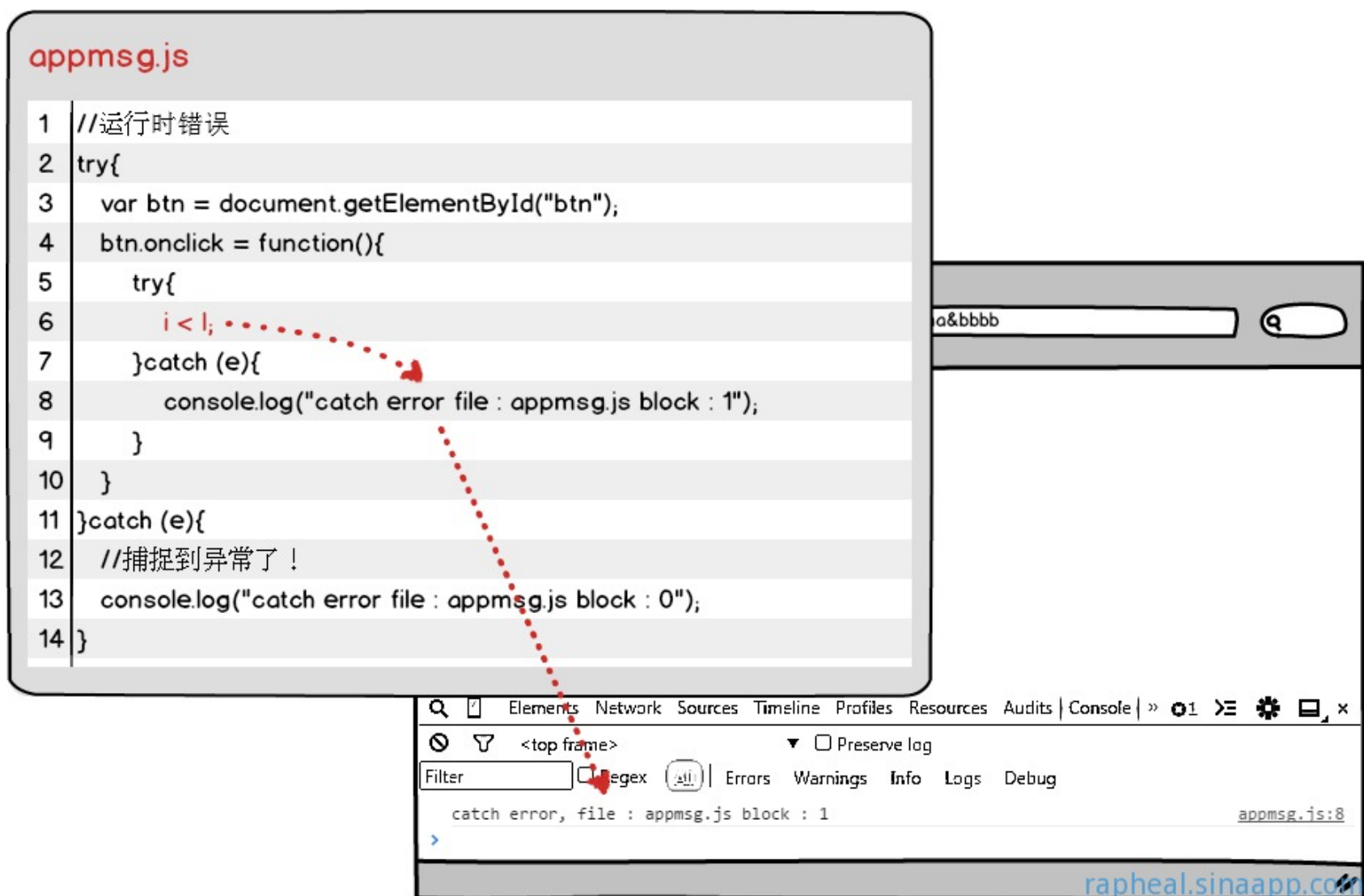
1. 没法捕捉try,catch块，当前代码块有语法错误，JS解释器压根都不会执行当前这个代码块，所以也就没办法被catch住；
2. 没法捕捉到全局的错误事件，也即是只有try,catch的块里边运行出错才会被你捕捉到，这里的块你要理解成一个函数块。

关于第一个缺点，我们没有任何解决办法，原因上边说了，但是一般语法阶段我们是能在开发阶段/或者用工具检测到的，于是乎它就被忽略了。

第二个缺点应该怎么理解呢？try, catch只能捕捉到当前执行流里边的运行错误，对于异步回调来说，是不属于这个try,catch块的：



我们可以怎么去改进这个方案，我利用了uglifyjs的词法语法分析，再uglifyjs最后输出压缩文件的时候往文件块以及function块加入了try,catch，同时为每个块加入编号，这样出错的时候我们在日志里边就能看到是哪个块有问题。



## window.onerror

window.onerror一样可以拿到出错的信息以及文件名、行号、列号，还可以

在window.onerror最后return true让浏览器不输出错误信息到控制台。

The image shows a code editor with a file named `appmsg.js`. The code is as follows:

```
1 //运行时错误
2 window.onerror = function(msg,url,line,col,error){
3   console.log(msg);
4   console.log(url);
5   console.log(line);
6   console.log(col);
7   console.log(error);
8   //return true;
9 }
10 i < 1;
```

A bracket groups lines 3 through 7. A red dotted line connects the closing brace of the `onerror` function to the console. A text box points to line 8 with the text: "return true不输出这条错误信息到console".

The browser's developer console shows the following error logs:

- Uncaught ReferenceError: i is not defined `appmsg.js:3`
- `http://localhost/res/appmsg.js` `appmsg.js:4`
- `10` `appmsg.js:5`
- `1` `appmsg.js:6`
- `ReferenceError: i is not defined {stack: (...), message: "i is not defined"}` `appmsg.js:7`
- `Uncaught ReferenceError: i is not defined` `appmsg.js:10`

window.onerror能捕捉到语法错误，但是语法出错的代码块不能跟window.onerror在同一个块（语法都没过，更别提window.onerror会被执行了）

The image shows a code editor with a file named `appmsg.js`. The code is as follows:

```
1 //语法错误
2 window.onerror = function(msg,url,line,col,error){
3   console.log(msg);
4   console.log(url);
5   console.log(line);
6   console.log(col);
7   console.log(error);
8   //return true;
9 }
10 if= 1;
```

A red dotted line connects the `if= 1;` line to the console.

The browser's developer console shows the following error log:

- `Uncaught SyntaxError: Unexpected token =` `appmsg.js:10`



只要把window.onerror这个代码块分离出去，并且比其他脚本先执行（注意这个前提！）即可捕捉到语法错误。

The diagram illustrates the setup for capturing JavaScript errors using `window.onerror`. It shows two code snippets:

```
error.js  
1 window.onerror = function(msg,url,line,col,error){  
2   console.log(msg);  
3   console.log(url);  
4   console.log(line);  
5   console.log(col);  
6   console.log(error);  
7   //return true;  
8 }
```

```
appmsg.js  
1 //语法错误  
2 if = 1;
```

Red dotted arrows indicate the flow of error information from the `appmsg.js` script to the `error.js` handler and then to the browser's console. The browser window shows the address bar with `/mp/s?aaa&bbbb` and the console displaying the following error:

```
Uncaught SyntaxError: Unexpected token =  
http://localhost/res/appmsg.js  
2  
4  
▶ SyntaxError: Unexpected token = {stack: (...), message: "Unexpected token ="}  
✖ Uncaught SyntaxError: Unexpected token =
```

对于跨域的JS资源，window.onerror拿不到详细的信息，需要往资源的请求添加额外的头部。

The diagram illustrates the setup for capturing JavaScript errors across domains. It shows the `appmsg.html` file with the following script tag:

```
<script src="http://res.wx.qq.com/appmsg.js" crossorigin></script>
```

The browser window shows the address bar with `mp.weixin.qq.com`. A callout box provides details about the static resource request:

```
静态资源请求/响应 res.wx.qq.com  
Request URL : http://res.wx.qq.com/appmsg.js  
Response Header:  
Access-Control-Allow-Origin:http://mp.weixin.qq.com
```

Red dotted arrows show the request from the browser to the static resource and the error information being passed to the console. The browser console displays the same error as in the previous diagram:

```
Uncaught SyntaxError: Unexpected token =  
http://localhost/res/appmsg.js  
2  
4  
▶ SyntaxError: Unexpected token = {stack: (...), message: "Unexpected token ="}  
✖ Uncaught SyntaxError: Unexpected token =
```

静态资源请求需要加多一个Access-Control-Allow-Origin头部，同时script引入外链的标签需要加多一个crossorigin的属性。经过这样折腾后一样能获取到准确的出错信息。

## 最终方案

我们先来比较两个方案各自的特点。

try,catch的方案有如下特点：

1. 无法捕捉到语法错误，只能捕捉运行时错误；
2. 可以拿到出错的信息，堆栈，出错的文件、行号、列号；
3. 需要借助工具把所有的function块以及文件块加入try,catch，可以在这个阶段打入更多的静态信息。

window.onerror的方案有如下特点：

1. 可以捕捉语法错误，也可以捕捉运行时错误；
2. 可以拿到出错的信息，堆栈，出错的文件、行号、列号；
3. 只要在当前页面执行的js脚本出错都会捕捉到，例如：浏览器插件的javascript、或者flash抛出的异常等。
4. 跨域的资源需要特殊头部支持。

window.onerror的方法要比try,catch方法更加完善，我们允许少量由于插件带来的脚本错误，最后window.onerror实现的方式是：

```
window.onerror = function(msg,url,line,col,error){
    //没有URL不上报！上报也不知道错误
    if (msg != "Script error." && !url){
        return true;
    }
    //采用异步的方式
    //我遇到过在window.onunload进行ajax的堵塞上报
    //由于客户端强制关闭webview导致这次堵塞上报有Network Error
    //我猜测这里window.onerror的执行流在关闭前是必然执行的
    //而离开文章之后的上报对于业务来说是可丢失的
    //所以我把这里的执行流放到异步事件去执行
    //脚本的异常数降低了10倍
    setTimeout(function(){
        var data = {};
```

```

//不一定所有浏览器都支持col参数
col = col || (window.event && window.event.errorCharacter) || 0;

data.url = url;
data.line = line;
data.col = col;
if (!!error && !!error.stack){
    //如果浏览器有堆栈信息
    //直接使用
    data.msg = error.stack.toString();
}else if (!!arguments.callee){
    //尝试通过callee拿堆栈信息
    var ext = [];
    var f = arguments.callee.caller, c = 3;
    //这里只拿三层堆栈信息
    while (f && (--c>0)) {
        ext.push(f.toString());
        if (f === f.caller) {
            break;//如果有环
        }
        f = f.caller;
    }
    ext = ext.join(",");
    data.msg = error.stack.toString();
}
//把data上报到后台!
},0);

return true;
};

```

## 后话

通过部署代码异常监控之后，不仅仅用以监控平时上线的异常，同时还发现了不少旧有代码的错误。例如以下代码：

```

src = img.getAttribute("src");
src.indexOf("http://rapheal.sinaapp.com/");

```

在某些情况下，文章里边的src可能是null，导致这里调用null的indexOf方法发生异常。也检测到微信webview里边的一些客户端抛出的异常，可以进一步让客户端开发的同事去做bug fix。

上线的稳定性不仅仅依托于代码异常的监控，代码异常监控只能监控到你代码的健康性，而很多时候业务的稳定还需要监控一些业务数据，例如昨天有1000个人点击了关注按钮，今天上线后突然变成了300人点击，除非你很清楚你上线的行为是会导致点击数下降，否则我们就应该重新审查这次上线是否存在问题，必要时还应该回退这次上线。