8.2 异常

程序不可避免会出现运行时错误，JavaScript 语言是一种非常灵活的脚本语言，由于没有静态编译的过程，在运行时就更容易出现异常现象，例如错误的把某个字符串类型变量当做整数类型去做处理等。

8.2.1 JS 运行异常

我们在网页开发中，如果在 Chrome 运行代码 8-5 片段，在 Chrome 浏览器的开发者工具 里的 Console 面板中出现脚本运行异常的错误信息，示例中 for 循环的条件表达式 i < l 中的 l 变量没定义，运行后出现异常，如图 8-1 所示。

代码清单 8-5 运行时产生异常的代码示例

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

图 8-1 网页 JS 运行错误

一般语法错误以及运行时错误，浏览器都会在 Console 里边显示对应的错误信息，以及出错的文件、行号、堆栈信息。

8.2.2 捕捉 JS 异常的方法

在 WebView 层有两种方法可以捕捉 JS 异常：

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。

图 8-2 和 8-3 分别展示了两个方案的使用方法以及出错呈现。

图 8-2 利用 try-catch 捕捉 JS 运行异常

图 8-3 通过 window.onerror 捕捉 JS 运行异常

这两个方案都无法捕捉代码的语法错误，但是一般在开发阶段，工具就已经能够显示出脚本的语法错误，因此这类异常完全是可以在开发阶段消除，运行阶段并不会有此类异常发生。对比 window.onerror 的方案，try-catch 的方案有个缺点：没法捕捉到全局的错误事件，也即是只有 try,catch 的块里边代码运行出错才会被捕捉到。逻辑层不存在 window 对象，因此逻辑层 AppService 侧无法通过 window.onerror 来捕捉异常。

所以小程序基础库在 WebView 侧使用 window.onerror 方案进行捕捉异常，在逻辑层 AppService 侧通过把 App 实例和 Page 实例的各个生命周期等方法包裹在 try-catch 里进行捕捉异常。同时在 App 构造器里提供了 onError 的回调，当业务代码运行产生异常时，这个回调被触发，同时能够拿到异常的具体信息，开发者自己根据业务情况处理对应的容错逻辑。

我们在基础库里捕捉到的运行时异常会上报到我们的服务器，然后产生类似图 8-4 所示的监控曲线，通过这个监控图来观察基础库的运行情况。

图 8-4 基础库异常监控曲线