1.2 小程序介绍

1.2.1 小程序技术发展历史

从技术的维度看，小程序并非凭空冒出来的一个概念。当微信中的 WebView 逐渐成为移动 Web 的一个重要入口时，微信就有相关的 JS API 了。

一些开发者应该对下面的代码有印象：

代码清单 1-1 使用 WeixinJSBridge 预览图片

```
WeixinJSBridge.invoke('imagePreview', {
    current: 'http://inews.gtimg.com/newsapp_bt/0/1693121381/641',
    urls: [ // 所有图片的 URL 列表，数组格式
        'https://img1.gtimg.com/10/1048/104857/10485731_980x1200_0.jpg',
        'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
        'https://img1.gtimg.com/10/1048/104857/10485729_980x1200_0.jpg'
    ]
}, function(res) {
    console.log(res.err_msg)
})
```

这是一个调用微信原生组件浏览图片的 JS API，相比于额外引入一个 JS 图片预览组件库，这种调用方式显得非常简洁和高效。

实际上，微信官方是没有对外暴露过如此调用的，此类 API 最初是提供给腾讯内部一些业务使用，很多外部开发者发现了之后，依葫芦画瓢地使用了，逐渐成为微信中网页的事实标准。

2015 年初，微信发布了一整套网页开发工具包，称之为 JS-SDK，开放了拍摄、录音、语音识别、二维码、地图、支付、分享、卡券等几十个 API。给所有的 Web 开发者打开了一扇全新的窗户，让所有开发者都可以使用到微信的原生能力，去完成一些之前做不到或者难以做到的事情了。

同样是调用原生的浏览图片，调用方式如代码清单 1-2 所示。

代码清单 1-2 使用 JS-SDK 调用图片预览组件

```
wx.previewImage({
  current: 'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
  urls: [ // 所有图片的 URL 列表，数组格式
    'https://img1.gtimg.com/10/1048/104857/10485731_980x1200_0.jpg',
    'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
    'https://img1.gtimg.com/10/1048/104857/10485729_980x1200_0.jpg'
  ],
  success: function(res) {
    console.log(res)
  }
})
```

JS-SDK 是对之前的 WeixinJSBrige 的一个包装，以及新能力的释放，并且由对内开放转为了对所有开发者开放，在很短的时间内获得了极大的关注。从数据监控来看，绝大部分在微信内传播的移动网页都使用到了相关的接口。

JS-SDK 解决了移动网页能力不足的问题，通过暴露微信的接口使得 Web 开发者能够拥有更多的能力，然而在更多的能力之外，JS-SDK 的模式并没有解决使用移动网页遇到的体验不良的问题。

用户在访问网页的时候，在浏览器开始显示之前都会有一个的白屏过程，在移动端，受限于设备性能和网络速度，白屏会更加明显。我们团队把很多技术精力放置在如何帮助平台上的 Web 开发者解决这个问题。因此我们设计了一个 JS-SDK 的增强版本，其中有一个重要的功能，称之为“微信 Web 资源离线存储”。

以下文字引用自内部的文档（没有最终对外开放）：

微信 Web 资源离线存储是面向 Web 开发者提供的基于微信内的 Web 加速方案。

通过使用微信离线存储，Web 开发者可借助微信提供的资源存储能力，直接从微信本地加载 Web 资源而不需要再从服务端拉取，从而减少网页加载时间，为微信用户提供更优质的网页浏览体验。每个公众号下所有 Web App 累计最多可缓存 5M 的资源。

这个设计有点类似 HTML5 的 Application Cache，但在设计上规避了一些 Application Cache 的不足。

在内部测试中，我们发现 离线存储 能够解决了一些问题，但是对于一些复杂的页面依然会有白屏的问题，例如页面加载了大量的 CSS 或者是 JavaScript 文件，这些文件的执行时间占用了大量的 UI 线程，这种时候，即使通过离线存储快速的加载资源，但是依旧会有页面的白屏现象，同时这样分文件的 Cache 在处理代码文件更新的时候操作较为繁杂，对开发者的要求较高。

除了白屏，影响 Web 体验的问题还有缺少操作的反馈，主要表现在两个方面：页面切换的生硬和点击的迟滞感。

对于一些有经验的 Web 开发者而言，会使用一些 SPA 的框架，来模拟客户端原生的页面切换过渡。通常的方式是在一个 WebView 中去模拟多个页面，通过 CSS 处理，加之精细化的脚本代码做到点击反馈和页面切换，获得较好的体验。然而并不是所有的开发者都有足够的时间和精力来使得页面的体验变得出色。

微信面临的问题是如何设计一个比较好的系统，使得所有开发者在微信中都能获得比较好的体验。这个问题是之前的 JS-SDK 所处理不了的，需要一个全新的系统来完成，它需要使得所有的开发者都能做到：

- *快速的加载*

- *更强大的能力*

- *原生的体验*

- *易用且安全的微信数据开放*

- *高效和简单的开发*

这一系统就是本书中需要详细阐述的小程序。

1.2.2 小程序与普通网页开发的区别

小程序的主要开发语言是 JavaScript ，所以通常小程序的开发会被用来同普通的网页开发来做对比。两者有很大的相似性，对于前端开发者而言，从网页开发迁移到小程序的开发成本并不高，但是二者还是有些许区别的。

网页开发渲染线程和脚本线程是互斥的，这也是为什么长时间的脚本运行可能会导致页面失去响应，而在小程序中，二者是分开的，分别运行在不同的线程中。网页开发者可以使用到各种浏览器暴露出来的 DOM API，进行 DOM 选中和操作。而如上文所述，小程序的逻辑层和渲染层是分开的，逻辑层运行在 JSCore 中，并没有一个完整浏览器对象，因而缺少相关的 DOM API 和 BOM API。这一区别导致了前端开发非常熟悉的一些库，例如 jQuery、 Zepto 等，在小程序中是无法运行的。同时 JSCore 的环境同 NodeJS 环境也是不尽相同，所以一些 NPM 的包在小程序中也是无法运行的。

网页开发者需要面对的环境是各式各样的浏览器，PC 端需要面对 IE、Chrome、QQ 浏览器等，在移动端需要面对 Safari、Chrome 以及 iOS、Android 系统中的各式 WebView 。而小程序开发过程中需要面对的是两大操作系统 iOS 和 Android 的微信客户端，以及用于辅助开发的小程序开发者工具，小程序中三大运行环境也是有所区别的，如表 1-1 所示。

表 1-1 小程序的运行环境

运行环境	逻辑层	渲染层
iOS	JavaScriptCore	WKWebView
安卓	X5 JSCore	X5 浏览器
小程序开发者工具	NWJS	Chrome WebView

网页开发者在开发网页的时候，只需要使用到浏览器，并且搭配上一些辅助工具或者编辑器即可。小程序的开发则有所不同，需要经过申请小程序帐号、安装小程序开发者工具、配置项目等等过程方可完成。