9.3 模拟器

小程序模拟器模拟小程序在微信客户端的逻辑和界面表现，方便开发者实时查看代码效果。由于系统差异以及微信客户端特有的一些交互流程，少部分的 API 无法在模拟器上进行模拟，但对于绝大部分的 API 均能够在模拟器上呈现出正确的状态。同时微信开发者工具提供多种机型尺寸以及自定义机型尺寸功能，方便开发者进行界面样式的机型适配。

图 9-6 小程序模拟器

9.3.2 逻辑层模拟

在 iOS 微信客户端上，小程序的 JavaScript 代码是运行在 JavaScriptCore 中，在 Android 微信客户端上，小程序的 JavaScript 代码是通过 X5 JSCore 来解析的。而在微信开发者工具上我们采用了一个隐藏着的 Webivew 来模拟小程序的逻辑运行环境。

图 9-7 微信客户端小程序运行环境模型简图

图 9-8 微信开发者工具小程序运行环境模型简图

在微信开发者工具上 WebView 是一个 chrome 的 `<webview />` 标签。与`<iframe />`标签不同的是，`<webview/>`标签是采用独立的线程运行的。

用于模拟小程序逻辑层的`<webview/>` 加载的链接是

```
http://127.0.0.1:9973/appservice/appservice
```

我们在开发者工具底层搭建了一个本地 HTTP 服务器来处理小程序模拟器的网络请求。其中：

./__asdebug/asdebug.js： 是开发者工具注入的脚本。

./__dev__/WAService.js：是小程序逻辑层基础库。

./util.js、./app.js、./index.js：开发者 JS 代码。

WebView 在请求开发者 JS 代码时，开发者工具读取 JS 代码进行必要的预处理后，将处理结果返回，然后由 WebView 解析执行。虽然开发者工具上是没有对 JS 代码进行合并的，但是还是按照相同的加载顺序进行解析执行。

图 9-9 appservice 内容

WebView 是一个浏览器环境，而 JsCore 是一个单纯的脚本解析器，浏览器中的 BOM 对象无法在 JSCore 中使用，开发者工具做了一个很巧妙的工作，将开发者的代码包裹在 define 域的时候，将浏览器的 BOM 对象局部变量化，从而使得在开发阶段就能发现问题。

图 9-10 BOM 对象局部变量化

9.3.3 渲染层模拟

微信开发者工具使用 chrome 的 <webview />标签来加载渲染层页面，每个渲染层 WebView 加载

```
http://127.0.0.1:9973/pageframe/pageframe.html
```

开发者工具底层搭建的 HTTP 本地服务器在收到这个请求的时候，就会编译 WXML 文件和 WXSS 文件，然后将编译结果作为 HTTP 请求的返回包。当确定加载页面的路径之后，如 index 页面，开发工具会动态注入如下一段脚本：

```
// 改变当前 webview 的路径，确保之后的图片网络请求能得到正确的相对路径

history.pushState('', '', 'pageframe/index')
```

```
// 创建自定义事件，将页面结构生成函数派发出去，由小程序渲染层基础库处理
document.dispatchEvent(new CustomEvent("generateFuncReady", {
   detail: {
     generateFunc: $gwx('./index.wxml')
   }
}))
// 注入对应页面的样式，这段函数由 WXSS 编译器生成
setCssToHead()
```

9.3.4 客户端模拟

微信客户端为丰富小程序的功能提供了大量的 API。在微信开发者工具上，通过借助 BOM（浏览器对象模型）以及 node.js 访问系统资源的能力，同时模拟客户端的 UI 和交互流程，使得大部分的 API 能够正常执行。

借助 BOM，如 wx.request 使用 XMLHttpRequest 模拟、wx.connectSocket 使用 WebSocket、wx.startRecord 使用 MediaRecorder、wx.playBackgroundAudio 使用 <audio/>标签；

借助 node.js，如使用 fs 实现 wx.saveFile、wx.setStorage、wx.chooseImage 等 API 功能。

借助模拟 UI 和交互流程，实现 wx.navigateTo、wx.showToast、wx.openSetting、wx.addCard 等。

9.3.5 通讯模拟

上文已经叙述了小程序的逻辑层、渲染层以及客户端在微信开发者工具上的模拟实现，除此之外，我们需要一个有效的通讯方案使得小程序的逻辑层、渲染层和客户端之间进行数据交流，才能将这三个部分串联成为一个有机的整体。

微信开发者工具的有一个消息中心底层模块维持着一个 WebSocket 服务器，小程序的逻辑层的 WebView 和渲染层页面的 WebView 通过 WebSocket 与开发者工具底层建立长连，使用 WebSocket 的 protocol 字段来区分 Socket 的来源。

代码清单 10-2 逻辑层中的消息模块

```
// <webview/>的 userAgent 是可定制的
// 通过 userAgent 中获取开发者工具 WebSocket 服务器监听的端口
var port = window.navigator.userAgent.match(/port\/(\d*)/)[1]
// 通过指定 protocol == 'APPSERVICE' 告知开发者工具这个链接是来自逻辑层
var ws = new WebSocket(`ws://127.0.0.1:${port}`, 'APPSERVICE')
ws.onmessage = (evt) => {
  let msg = JSON.parse(evt.data)
  // …处理来自开发者工具的信息
   }
// 调用 API 接口 wx.navigateBack
ws.send(JSON.stringify({
  command: 'APPSERVICE_INVOKE',
  data: {
     api: 'navigateBack',
     args: {}
  }
}))
```