3.1 渲染层和逻辑层

小程序的运行环境分成渲染层和逻辑层，第 2 章提到过 WXML 模板和 WXSS 样式工作在渲染层，JS 脚本工作在逻辑层。小程序的渲染层和逻辑层分离是经过很多考虑得出来的模型，在第 6 章我们会详细阐述这个模型背后的原理以及产生的问题。在本章我们会先介绍这个模型的基本工作方式。

3.1.1 渲染“Hello World”页面

我们看看小程序是如何把脚本里边的数据渲染在界面上的。

WXML 模板使用 view 标签，其子节点用 {{ }} 的语法绑定一个 msg 的变量，如代码清单 3-1 所示。

代码清单 3-1 渲染“Hello World”WXML 代码

```
<view>{{ msg }}</view>
```

在 JS 脚本使用 this.setData 方法把 msg 字段设置成 “Hello World” ，如代码清单 3-2 所示。

代码清单 3-2 渲染“Hello World”JS 脚本

```
Page({
  onLoad: function () {
    this.setData({ msg: 'Hello World' })
  }
})
```

从这个例子我们可以看到 3 个点：

1. 渲染层和数据相关。

2. 逻辑层负责产生、处理数据。

3. 逻辑层通过 Page 实例的 setData 方法传递数据到渲染层。

关于第 1 点，涉及了“数据驱动”的概念，我们会在 3.1.3 节详细讨论，我们现在先看看第 3 点涉及的“通信模型”。

3.1.2 通信模型

小程序的渲染层和逻辑层分别由 2 个线程管理：渲染层的界面使用了 WebView 进行渲染；逻辑层采用 JsCore 线程运行 JS 脚本。一个小程序存在多个界面，所以渲染层存在多个 WebView 线程，这两个线程的通信会经由微信客户端（下文中也会采用 Native 来代指微信客户端）做中转，逻辑层发送网络请求也经由 Native 转发，小程序的通信模型如图 3-1 所示。

图 3-1 渲染层和逻辑层通信模型

3.1.3 数据驱动

在开发 UI 界面过程中，程序需要维护很多变量状态，同时要操作对应的 UI 元素。随着界面越来越复杂，我们需要维护很多变量状态，同时要处理很多界面上的交互事件，整个程序变得越来越复杂。通常界面视图和变量状态是相关联的，如果有某种“方法”可以让状态和视图绑定在一起（状态变更时，视图也能自动变更），那我们就可以省去手动修改视图的工作。

这个方法就是“数据驱动”，下边我们来介绍一下小程序的数据驱动基本原理。WXML 结构实际上等价于一棵 Dom 树，通过一个 JS 对象也可以来表达 Dom 树的结构，如图 3-2 所示。

图 3-2 WXML 结构和 JS 对象均可以表示一棵 Dom 树

WXML 可以先转成 JS 对象，然后再渲染出真正的 Dom 树，回到“Hello World”那个例子，我们可以看到转换的过程如图 3-3 所示。

图 3-3 WXML 结构转 JS 对象，再转 Dom 树

通过 setData 把 msg 数据从“Hello World”变成“Goodbye”，产生的 JS 对象对应的节点就会发生变化，此时可以对比前后两个 JS 对象得到变化的部分，然后把这个差异应用到原来的 Dom 树上，从而达到更新 UI 的目的，这就是“数据驱动”的原理，如图 3-4 所示。

图 3-4 状态更新的时候，通过对比前后 JS 对象变化，进而改变视图层的 Dom 树

3.1.4 双线程下的界面渲染

小程序的逻辑层和渲染层是分开的两个线程。在渲染层，宿主环境会把 WXML 转化成对应的 JS 对象，在逻辑层发生数据变更的时候，我们需要通过宿主环境提供的 setData 方法把数据从逻辑层传递到渲染层，再经过对比前后差异，把差异应用在原来的 Dom 树上，渲染出正确的 UI 界面，如图 3-5 所示。

图 3-5 逻辑层传递数据到渲染层