6.3 原生组件

在内置组件中，有一些组件较为特殊，它们并不完全在 Exparser 的渲染体系下，而是由客户端原生参与组件的渲染，这类组件我们称为“原生组件”，这也是小程序 Hybrid 技术的一个应用。

6.3.1 原生组件运行机制

要介绍原生组件的运行机制，我们需要从一行代码看起。

代码清单 6-8 展示一个地图组件

```
<map latitude="39.92" longtitude="116.46"></map>
```

在原生组件内部，其节点树非常简单，基本上可以认为只有一个 div 元素。上面这行代码在渲染层开始运行时，会经历以下几个步骤：

1. 组件被创建，包括组件属性会依次赋值。
2. 组件被插入到 DOM 树里，浏览器内核会立即计算布局，此时我们可以读取出组件相对页面的位置（x，y 坐标）、宽高。
3. 组件通知客户端，客户端在相同的位置上，根据宽高插入一块原生区域，之后客户端就在这块区域渲染界面
4. 当位置或宽高发生变化时，组件会通知客户端做相应的调整

我们可以看出，原生组件在 WebView 这一层的渲染任务是很简单，只需要渲染一个占位元素，之后客户端在这块占位元素之上叠了一层原生界面。因此，原生组件的层级会比所有在 WebView 层渲染的普通组件要高。

图 6-1 原生组件层级示意图

引入原生组件主要有 3 个好处：

1. 扩展 Web 的能力。比如像输入框组件（input，textarea）有更好地控制键盘的能力。

2. 体验更好，同时也减轻 WebView 的渲染工作。比如像地图组件（map）这类较复杂的组件，其渲染工作不占用 WebView 线程，而交给更高效的客户端原生处理。
3. 绕过 setData、数据通信和重渲染流程，使渲染性能更好。比如像画布组件（canvas）可直接用一套丰富的绘图接口进行绘制。

表 6-1 常用的几个原生组件

组件名	名称	是否有**context**	描述
video	视频	是	播放视频
map	地图	是	展示地图
canvas	画布	是	提供一个可以自由绘图的区域
picker	弹出式选择器	否	初始时没有界面，点击时弹出选择器

交互比较复杂的原生组件都会提供“context”，用于直接操作组件。以 canvas 为例，小程序提供了 wx.createCanvasContext 方法来创建 canvas 的 context。这是一个可以用于操作 canvas 的对象，对象下提供了很多绘图的方法，如“setFillStyle”方法可以设置填充样式，“fillRect”方法用于绘制矩形（这些方法与 HTML DOM Canvas 兼容）。

代码清单 6-9 canvas 组件 context 对象示例（WXML 代码）

```
<canvas canvas-id="myCanvas"></canvas>
```

代码清单 6-10 canvas 组件 context 对象示例（JS 代码）

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 75)
ctx.draw()
```

这段代码可以创建 WXML 中对应 canvas 节点的 context，通过调用 context 中的方法，在画布上绘制一个矩形。

类似于 canvas，video、map 等原生组件都可以创建 context，context 中提供的方法非常丰富，这里就不一一列举了。

6.3.2 原生组件渲染限制

原生组件脱离在 WebView 渲染流程外，这带来了一些限制。最主要的限制是一些 CSS 样式无法应用于原生组件，例如，不能在父级节点使用 overflow:hidden 来裁剪原生组件的显示区域；不能使用 transformrotate 让原生组件产生旋转等。

开发者最为常见的问题是，原生组件会浮于页面其他组件之上（相当于拥有正无穷大的 z-index 值）使其它组件不能覆盖在原生组件上展示。想要解决这个问题，可以考虑使用 cover-view 和 cover-image 组件。这两个组件也是原生组件，同样是脱离 WebView 的渲染流程外，而原生组件之间的层级就可以按照一定的规则控制。