Mini app markup, styling, and scripting

- Markup languages
  - Data binding
  - List rendering
  - Conditional rendering
  - Templates
- Styling
- Scripting
  - JavaScript bridge API
- Acknowledgements

{% Aside %} This post is part of an article collection where each article builds upon previous articles. If you just landed here, you may want to start reading from the beginning. {% endAside %}

Markup languages

As outlined before, rather than with plain HTML, mini apps are written with dialects of HTML. If you have ever dealt with Vue.js text interpolation and directives, you will feel immediately at home, but similar concepts existed way before that in XML Transformations (XSLT). Below, you can see code samples from WeChat's WXML, but the concept is the same for all mini apps platforms, namely Alipay's AXML, Baidu's Swan Element, ByteDance's TTML (despite the DevTools calling it Bxml), and Quick App's HTML. Just like with Vue.js, the underlying mini app programming concept is the model-view-viewmodel (MVVM).

Data binding

Data binding corresponds to Vue.js' text interpolation.

```
<!-- wxml -->
<view>{% raw %}{{message}}{% endraw %}</view>
```

```
// page.js
Page({
  data: {
    message: "Hello World!",
  },
});
```

List rendering

List rendering works like Vue.js `v-for` directive.

```
<!-- wxml -->
<view wx:for="{% raw %}{{array}}{% endraw %}">{% raw %}{{item}}{% endraw %}</view>
```

```
// page.js
Page({
  data: {
    array: [1, 2, 3, 4, 5],
  },
});
```

Conditional rendering

Conditional rendering works like Vue.js' `v-if` directive.

```
<!-- wxml -->
<view wx:if="{% raw %}{{view == 'one'}}{% endraw %}">One</view>
<view wx:elif="{% raw %}{{view == 'two'}}{% endraw %}">Two</view>
<view wx:else="{% raw %}{{view == 'three'}}{% endraw %}">Three</view>
```

```
// page.js
Page({
```

```
  data: {
    view: "three",
  },
});
```

Templates

Rather than requiring the imperative cloning of the `content` of an HTML template, WXML templates can be used declaratively via the `is` attribute that links to a template definition.

```
<!-- wxml -->
<template name="person">
  <view>
    First Name: {% raw %}{{firstName}}{% endraw %}, Last Name: {% raw %}{{lastName}}{% endraw
%}
  </view>
</template>
```

```
<template is="person" data="{% raw %}{{...personA}}{% endraw %}"></template>
<template is="person" data="{% raw %}{{...personB}}{% endraw %}"></template>
<template is="person" data="{% raw %}{{...personC}}{% endraw %}"></template>
```

```
// page.js
Page({
  data: {
    personA: { firstName: "Alice", lastName: "Foo" },
    personB: { firstName: "Bob", lastName: "Bar" },
    personC: { firstName: "Charly", lastName: "Baz" },
  },
});
```

Styling

Styling happens with dialects of CSS. WeChat's is named WXSS. For Alipay, theirs is called ACSS, Baidu's simply CSS, and for ByteDance, their dialect is referred to as TTSS. What they have in common is that they extend CSS with responsive pixels. When writing regular CSS, developers need to convert all pixel units to adapt to different mobile device screens with different widths and

pixel ratios. TTSS supports the `rpx` unit as its underlying layer, which means the mini app takes over the job from the developer and converts the units on their behalf, based on a specified screen width of `750rpx`. For example, on a Pixel 3a phone with a screen width of `393px` (and a device pixel ratio of `2.75`), responsive `200rpx` become `104px` on the real device when inspected with Chrome DevTools (393px / 750rpx * 200rpx ≈ 104px). In Android, the same concept is called density-independent pixel.

> {% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/n26ptkMoSfiTDanfFh5F.png", alt="Inspecting a view with Chrome DevTools whose responsive pixel padding was specified with `200rpx` shows that it is actually `104px` on a Pixel 3a device.", width="800", height="385" %}
> Inspecting the actual padding on a Pixel 3a device with Chrome DevTools.

```
/* app.wxss */
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
  padding: 200rpx 0; /* ← responsive pixels */
  box-sizing: border-box;
}
```

Since components (see later) do not use shadow DOM, styles declared on a page reach into all components. The common stylesheet file organization is to have one root stylesheet for global styles, and individual per-page stylesheets specific to each page of the mini app. Styles can be imported with an `@import` rule that behaves like the `@import` CSS at-rule. Like in HTML, styles can also be declared inline, including dynamic text interpolation (see before).

```
<view style="color:{% raw %}{{color}}{% endraw %};" />
```

Scripting

Mini apps support a "safe subset" of JavaScript that includes support for modules using varying syntaxes that remind of CommonJS or RequireJS. JavaScript code cannot be executed via `eval()` and no functions can be created with `new Function()`. The scripting execution context is V8 or JavaScriptCore on devices, and V8 or NW.js in the simulator. Coding with ES6 or newer syntax is usually possible, since the particular DevTools automatically transpile the code to ES5 if the build

target is an operating system with an older WebView implementation (see later). The documentation of the super app providers explicitly mentions that their scripting languages are not to be confused with and are distinct from JavaScript. This statement, however, refers mostly just to the way modules work, that is, that they do not support standard ES Modules yet.

As mentioned before, the mini app programming concept is the model-view-viewmodel (MVVM). The logic layer and the view layer run on different threads, which means the user interface does not get blocked by long-running operations. In web terms, you can think of scripts running in a Web Worker.

WeChat's scripting language is called WXS, Alipay's SJS, ByteDance's likewise SJS. Baidu speaks of JS when referencing theirs. These scripts need to be included using a special kind of tag, for example, `<wxs>` in WeChat. In contrast, Quick App uses regular `<script>` tags and the ES6 JS syntax.

```
<wxs module="m1">
  var msg = "hello world";
  module.exports.message = msg;
</wxs>

<view>{% raw %}{{m1.message}}{% endraw %}</view>
```

Modules can also be loaded via a `src` attribute, or imported via `require()`.

```
// /pages/tools.wxs
var foo = "'hello world' from tools.wxs";
var bar = function (d) {
  return d;
};
module.exports = {
  FOO: foo,
  bar: bar,
};
module.exports.msg = "some msg";
```

```
<!-- page/index/index.wxml -->
<wxs src="./../tools.wxs" module="tools" />
<view>{% raw %}{{tools.msg}}{% endraw %}</view>
<view>{% raw %}{{tools.bar(tools.FOO)}}{% endraw %}</view>
```

```
// /pages/logic.wxs
var tools = require("./tools.wxs");

console.log(tools.FOO);
console.log(tools.bar("logic.wxs"));
console.log(tools.msg);
```

JavaScript bridge API

The JavaScript bridge that connects mini apps with the operating system makes it possible to use OS capabilities (see Access to powerful features. It also offers a number of convenience methods. For an overview, you can check out the different APIs of WeChat, Alipay, Baidu, ByteDance, and Quick App.

Feature detection is straightforward, since all platforms provide a (literally called like this) `canIUse()` method whose name seems inspired by the website caniuse.com. For example, ByteDance's `tt.canIUse()` allows for support checks for APIs, methods, parameters, options, components, and attributes.

```
// Testing if the `<swiper>` component is supported.
tt.canIUse("swiper");
// Testing if a particular field is supported.
tt.canIUse("request.success.data");
```

{% Aside 'success' %} The next chapter introduces mini app components. {% endAside %}

Acknowledgements

This article was reviewed by Joe Medley, Kayce Basques, Milica Mihajlija, Alan Kent, and Keith Gu.

□□□ #2
□ □□□ □□□ 2026-10-31 22:59:44 UTC
□ □□□ □□□ 2026-10-31 22:59:45 UTC