腾讯云小程序平台

小程序开发指南

产品文档



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

- 小程序开发指南
  - 小程序介绍与开发环境
  - 小程序代码组成
    - JSON 配置
    - WXML 模板
    - WXSS 样式
    - JavaScript 脚本
  - 指南
    - 了解小程序
      - 小程序宿主环境
      - 小程序运行机制
      - 小程序代码结构
      - 小程序目录结构
    - 配置小程序
      - 全局与页面配置
    - 小程序框架
      - 介绍
      - 逻辑层
      - 渲染层
    - 自定义组件
    - 基础能力
      - 网络
      - 存储
      - 文件系统
      - 分包加载
      - 画布
      - 自定义 tabBar
      - 多进程 Worker
      - DarkMode Adaptation Guide
  - 框架
    - 小程序配置
      - 项目配置
      - 全局配置
      - 页面配置
    - 框架接口

版权所有：腾讯云计算（北京）有限责任公司

App
页面
自定义组件
模块化
基础能力
WXML 语法参考
介绍
数据绑定
列表渲染
条件渲染
模板
引用
WXS 语法参考
介绍
WXS 模块
变量
注释
运算符
语句
数据类型
基础类库
基础能力
分包加载
可用性
组件
组件概览
原生组件
视图容器
基础内容
地图组件
媒体组件
开放能力
Canvas 画布
导航组件
表单组件
API
API 概览
基础

版权所有：腾讯云计算（北京）有限责任公司

- 路由
- 跳转
- 转发
- 界面
  - 窗口
  - 字体
  - 交互
  - 导航栏
  - 背景
  - TabBar
  - 下拉刷新
  - 滚动
  - 动画
  - 自定义组件
  - 菜单
- 网络
  - UDP 通信
  - mDNS
  - 发起请求
  - 上传
  - WebSocket
  - 下载
  - TCP 通信
- 数据缓存
- 数据分析
- 画布
  - 画布概要
  - OffscreenCanvas
  - Path2D
  - Canvas
  - canvasContext
  - Color
- Worker
- 媒体
  - 富文本
  - 视频
  - 录音
  - 背景音频

版权所有：腾讯云计算（北京）有限责任公司

- 地图
- 实时音视频
- 透明视频
- 相机
- 图片
- 音频

WXML

文件

设备

- 联系人
- 振动
- 电量
- 屏幕
- 内存
- 设备方向
- 无障碍
- 网络
- 键盘
- 电话
- 加速计
- 罗盘
- 陀螺仪
- 短信
- 扫码
- 加密
- NFC－Ndef
- NFC－NfcA
- NFC－NfcB
- NFC－NfcF
- NFC－NfcV
- WiFi
- 日历
- 剪切板
- NFC－getNFCAdapter
- NFC－NFCAdapter
- NFC－IsoDep
- NFC－MifareClassic
- NFC－MifareUltralight

版权所有：腾讯云计算（北京）有限责任公司

蓝牙-低功耗中心设备
蓝牙-低功耗外围设备
蓝牙-信标
蓝牙-通用
位置
开放接口
自定义API
敏感 API
JS SDK
Web-View
接口说明
基础库
基础库介绍
基础库维护
IDE 使用指南
概览
界面
设置
代码编辑
小程序调试
其他须知

版权所有：腾讯云计算（北京）有限责任公司

小程序开发指南
小程序介绍与开发环境

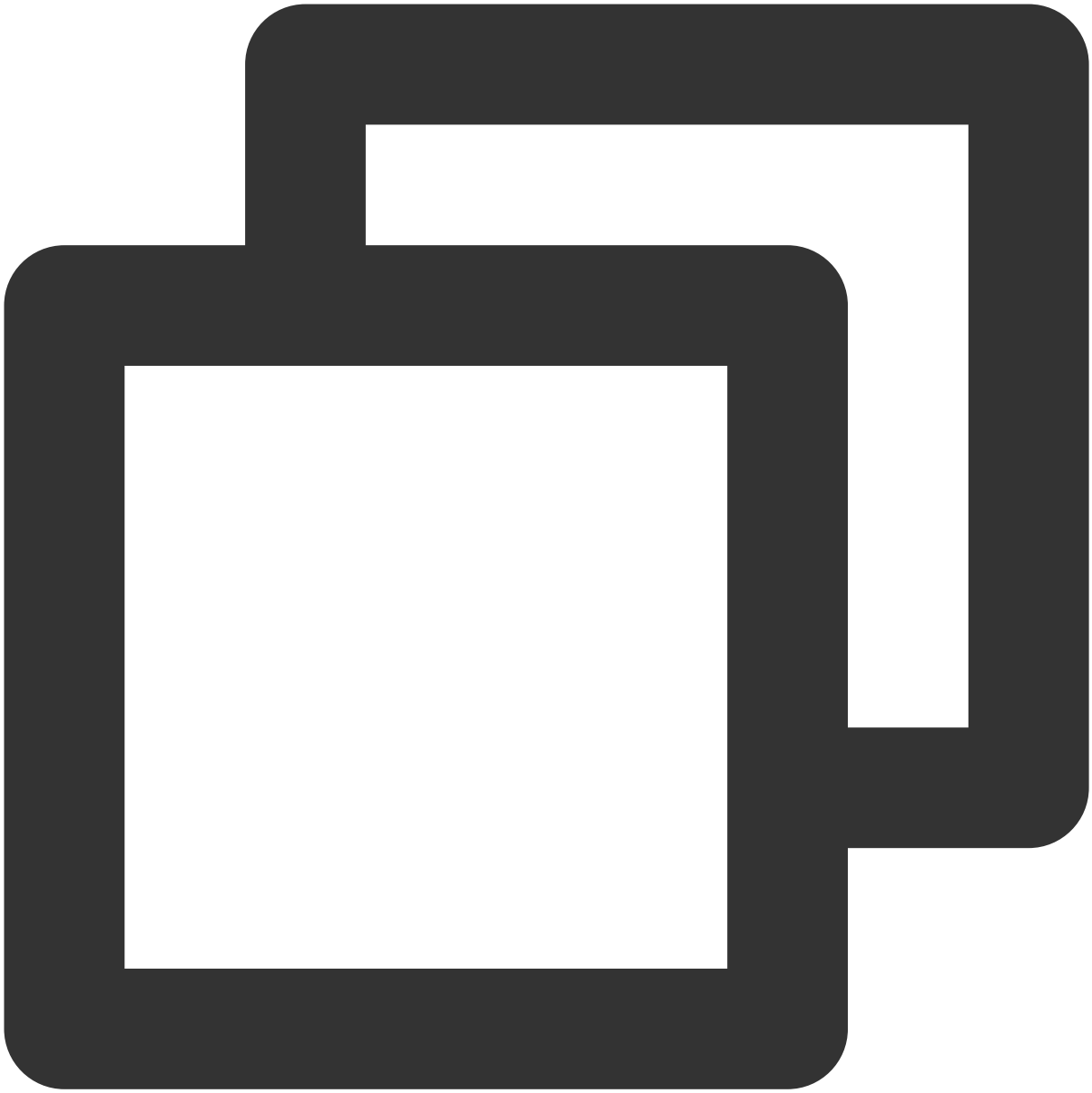
最近更新时间：2024-05-10 11:16:30

小程序是一种全新的连接用户与服务的方式，它可以在应用内被便捷地获取和传播，同时具有出色的使用体验。任何一个普通的开发者，经过简单的学习和练习后，都可以轻松地完成一个小程序的开发和发布。

1、Hello World

在详细介绍小程序历史和技术细节前，请先跟随我们的步骤完成开发Hello World例子。
第一步，根据自己的操作系统下载对应的安装包进行安装。
第二步，打开 TCMPP 开发者工具，选择新建小程序项目，我们先不需理解 AppID 的概念，新建项目时选择无AppID，并取消勾选“建立普通快速启动模板”的选项。
最后一步，我们来添加必要的代码。
在根目录下创建 app.json，其内容如下。

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "pages": ["pages/index/index"]
}
```

在根目录下新建pages目录，然后在pages目录下新建index目录，接着在 index 目录下创建两个文件 index.wxml 和 index.js。

index.wxml 的内容如下所示。 `<text>Hello World</text>`

index.js 的内容如下所示。 `Page({})`

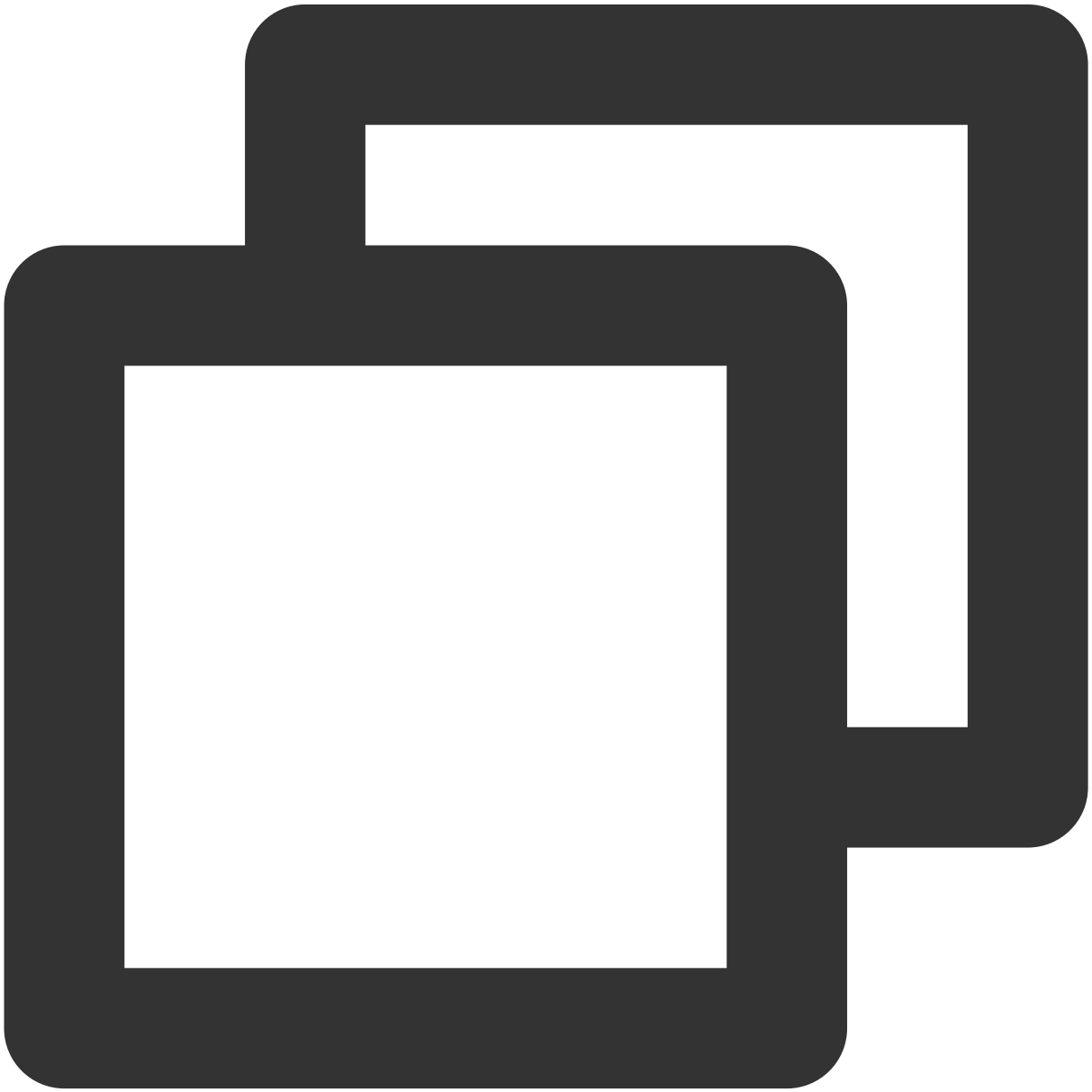
版权所有：腾讯云计算（北京）有限责任公司

通过编写以上短短的几行代码，TCMPP 开发者工具的模拟器界面上显示出 Hello World。

2、小程序介绍

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 WebView 逐渐成为移动 Web 的一个重要入口时，JS API 也应运而生了，如下代码所示：



```
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)
})
```

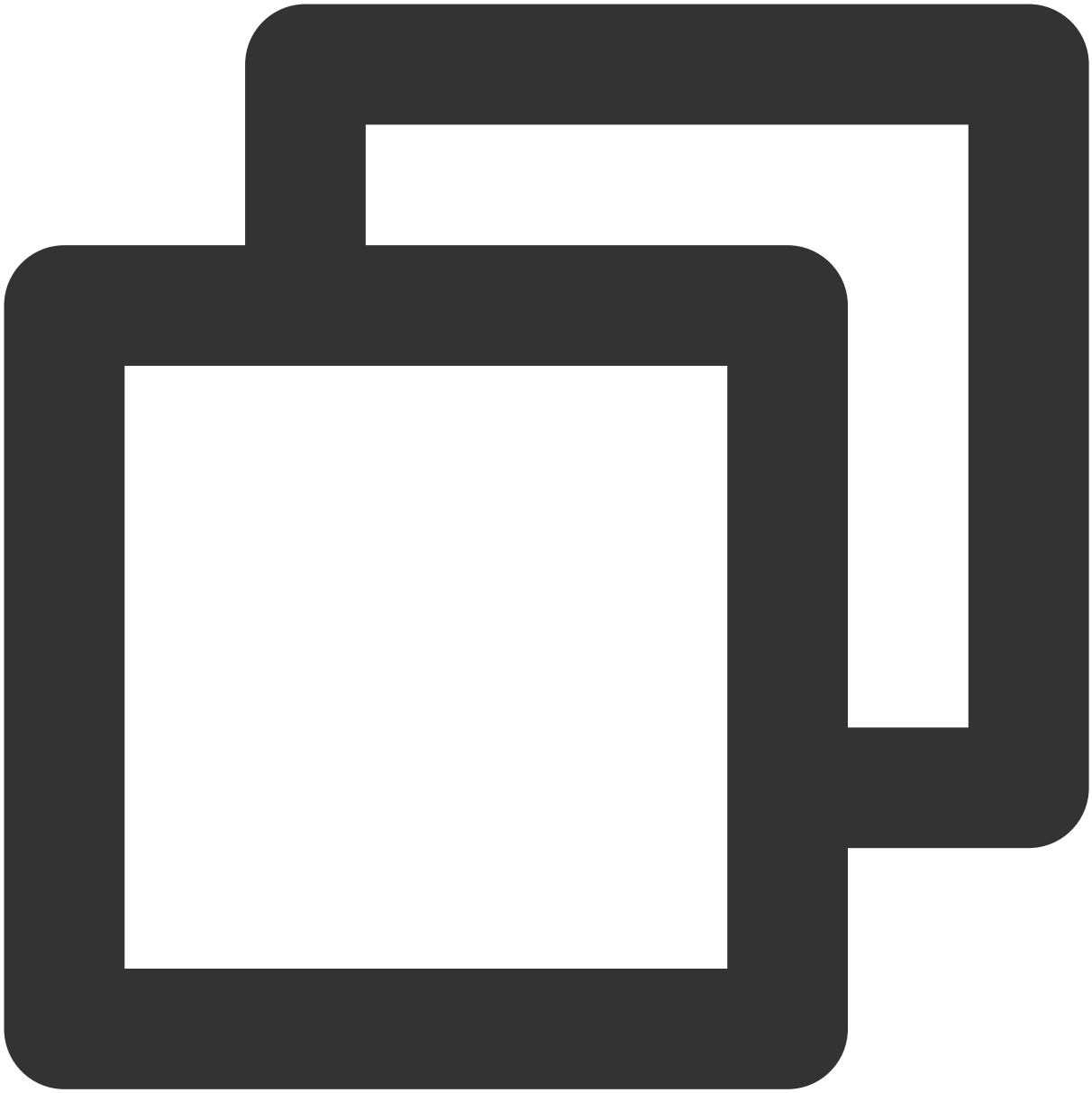
这是一个调用原生组件浏览图片的 JS API，相比于额外引入一个 JS 图片预览组件库，这种调用方式显得非常简洁和高效。

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。

平台内置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宿主应用内的原生能力，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，调用方式如下代码所示：

版权所有：腾讯云计算（北京）有限责任公司



```
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)
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

JS-SDK 解决了移动网页能力不足的问题，通过暴露宿主应用的接口使得 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“Web 资源离线存储”。
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。
宿主应用随之面临的问题是如何设计一个比较好的系统，使得所有开发者在宿主应用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。

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 的宿主应用客户端，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，如下表所示。

运行环境	逻辑层	渲染层

版权所有：腾讯云计算（北京）有限责任公司

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账号、安装小程序开发者工具、配置项目等等过程方可完成。

3、小程序的特色

对于普通用户，小程序实现了应用的触手可及，只需要通过扫描二维码、搜索或者是朋友的分享就可以直接打开，加上优秀的体验，小程序使得服务提供者的触达能力变得更强。
对于开发者而言，小程序框架本身所具有的快速加载和快速渲染能力，加之配套的云能力、运维能力和数据汇总能力，使得开发者不需要去处理琐碎的工作，可以把精力放置在具体的业务逻辑的开发上。

4、小程序开发准备

开发者可以在平台创建小程序后会自动生成对应的 App ID，并且提供不同版本的 IDE 下载地址，开发者可以自行选择合适版本进行下载，部分操作可参考 快速入门。

版权所有：腾讯云计算（北京）有限责任公司

小程序代码组成
JSON 配置

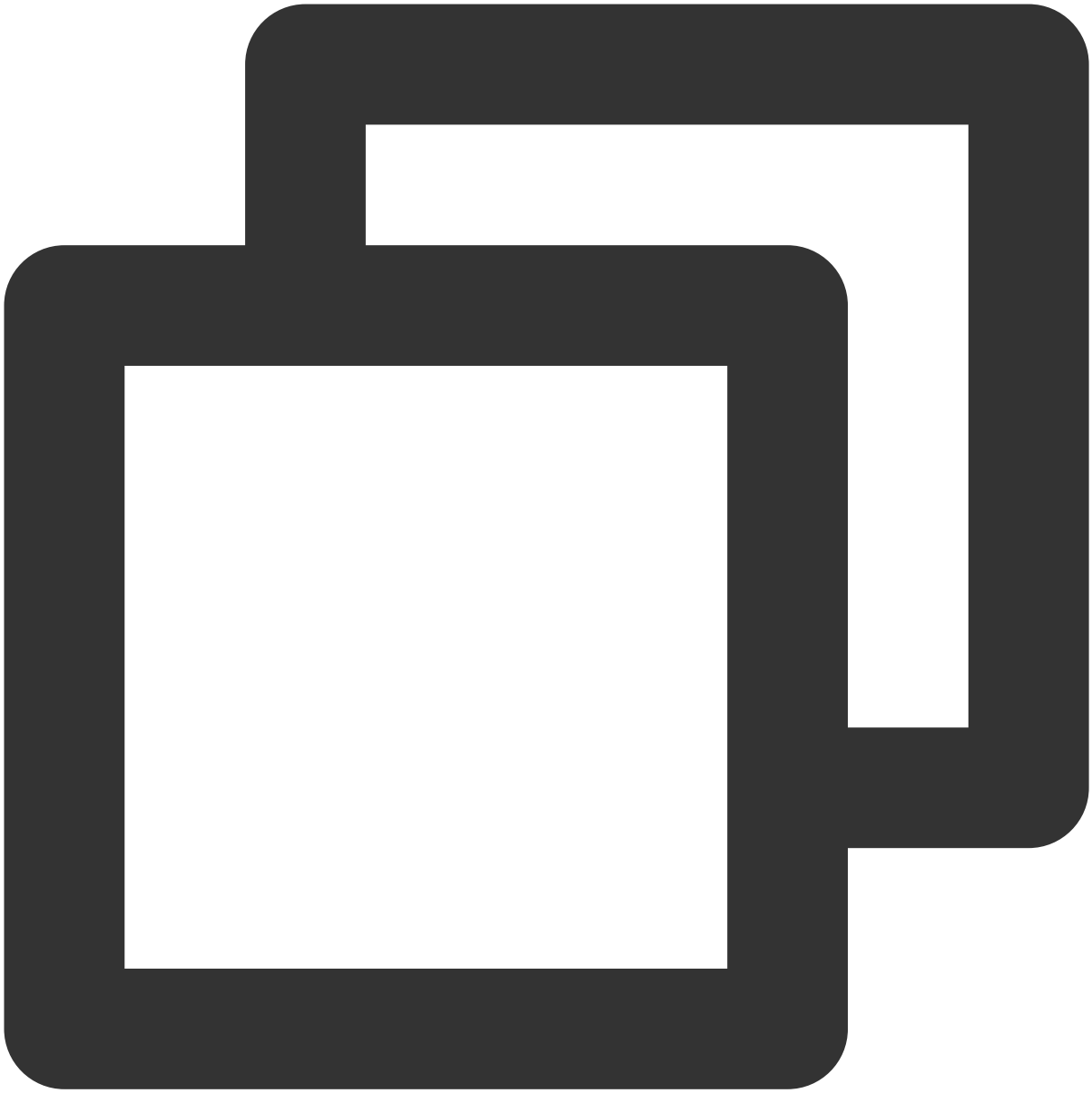
最近更新时间：2024-05-10 11:19:52

JSON 是一种数据格式，并不是编程语言，在小程序中，JSON扮演的静态配置的角色。

一个例子

先看一个例子，打开开发工具的编辑器，在根目录下可以找到 app.json 文件，双击打开，本章代码 app.json文件代码如下：

版权所有：腾讯云计算（北京）有限责任公司

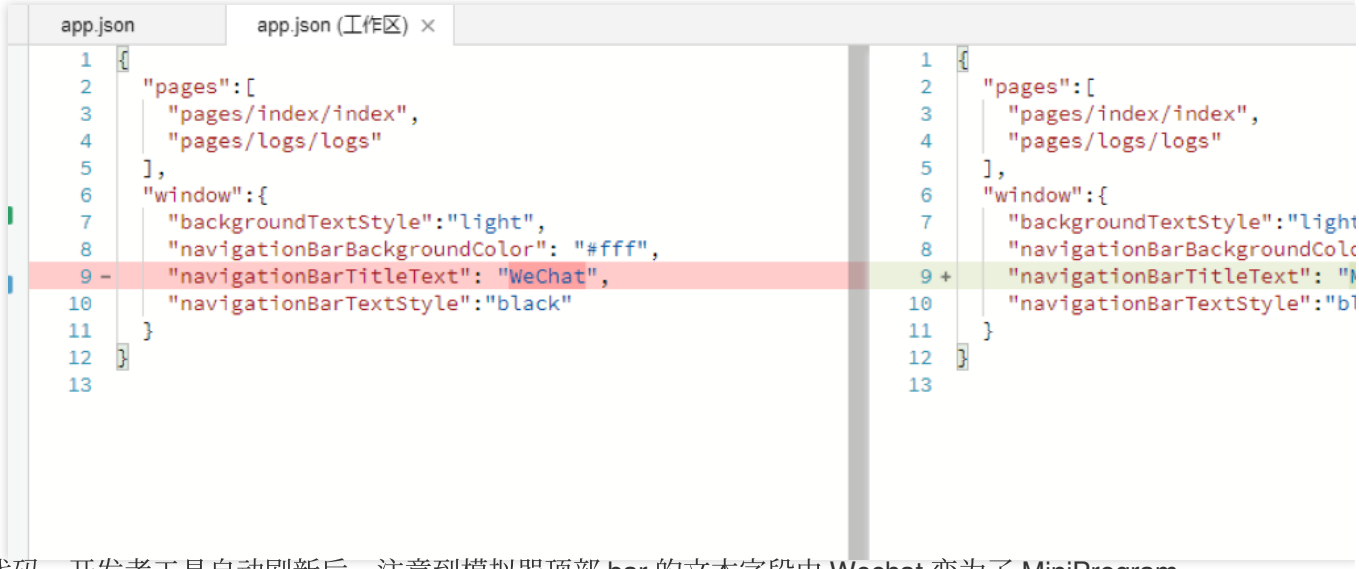


```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle":"black"
  }
```

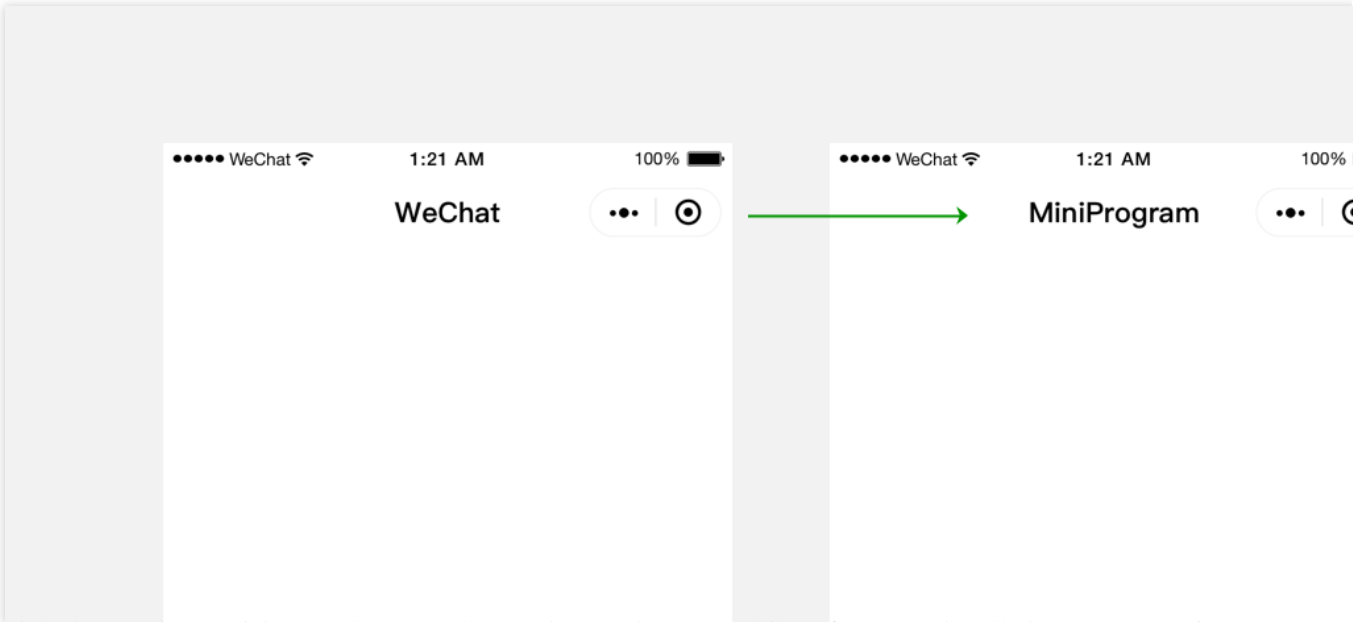
版权所有：腾讯云计算（北京）有限责任公司

```
}
```

尝试修改第 9 行为 "navigationBarTitleText": "MiniProgram" ，如下图所示。



保存代码，开发者工具自动刷新后，注意到模拟器顶部 bar 的文本字段由 Wechat 变为了 MiniProgram。



JSON 文件在小程序代码中扮演静态配置的作用，在小程序运行之前就决定了小程序一些表现，需要注意的是小程序是无法在运行过程中去动态更新 JSON 配置文件从而发生对应的变化的。

JSON 语法

相比于 XML ，JSON 格式最大的优点是易于人的阅读和编写，通常不需要特殊的工具，就能读懂和修改，是一种轻量级的数据交换格式。

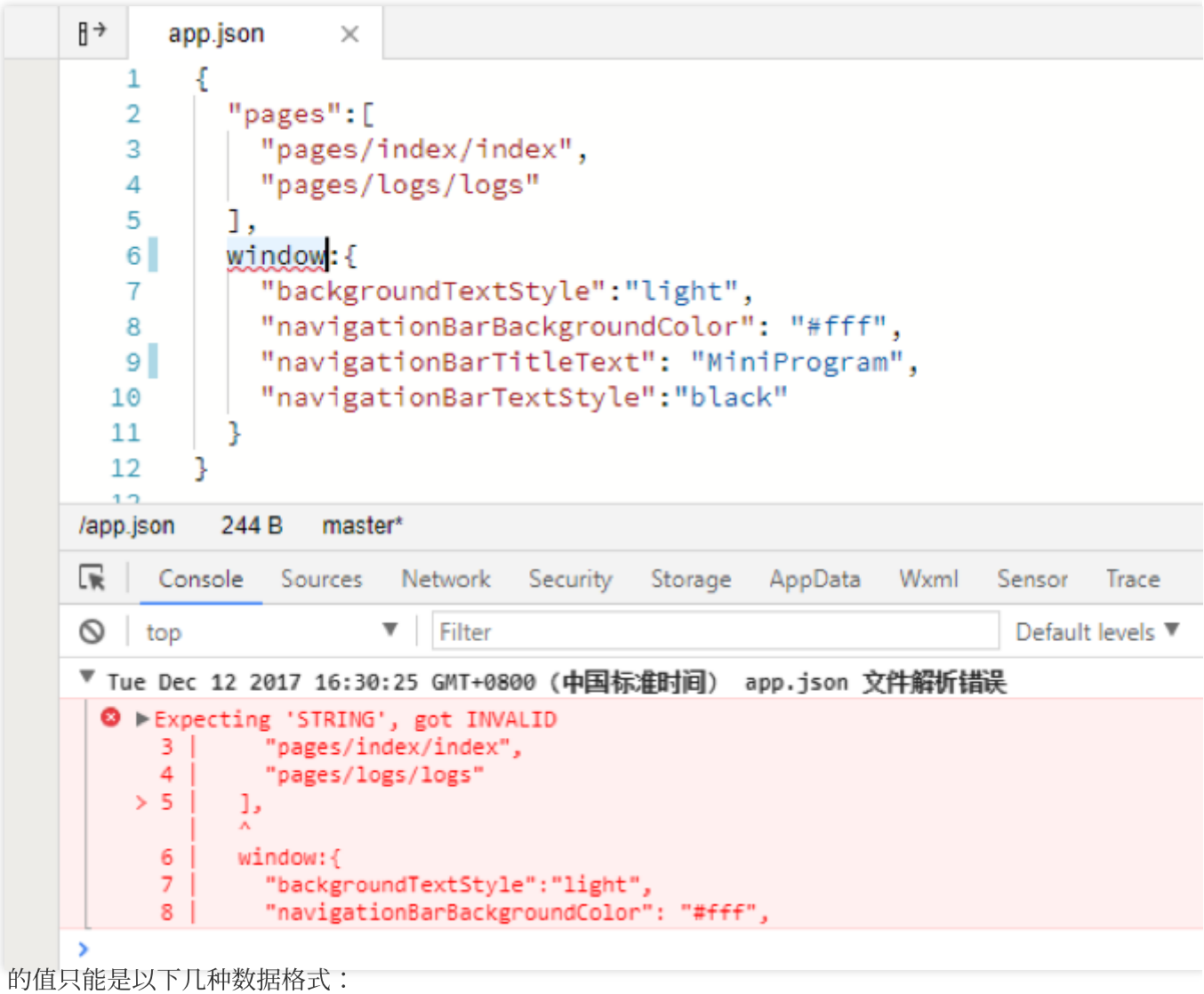
JSON 文件都是被包裹在一个大括号中 {}，通过 key-value 的方式来表达数据。



看起来同 JavaScript 的对象表达方式十分相似，但是有所不同。

JSON 的 Key 必须包裹在一个双引号中，在实践中，编写 JSON 的时候，忘了给 Key 值加双引号或者是把双引号写成单引号是常见错误。

版权所有：腾讯云计算（北京）有限责任公司

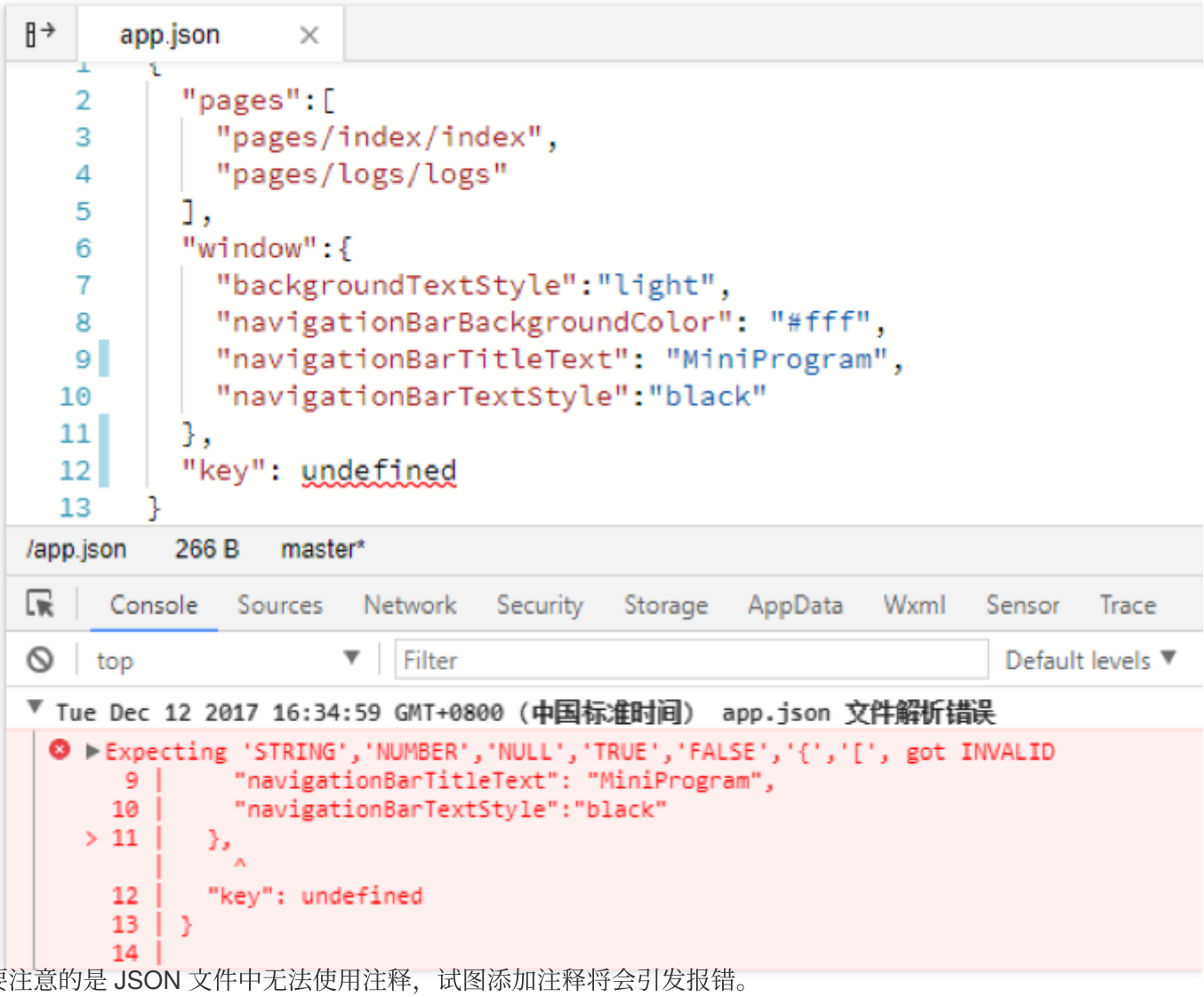


JSON 的值只能是以下几种数据格式：

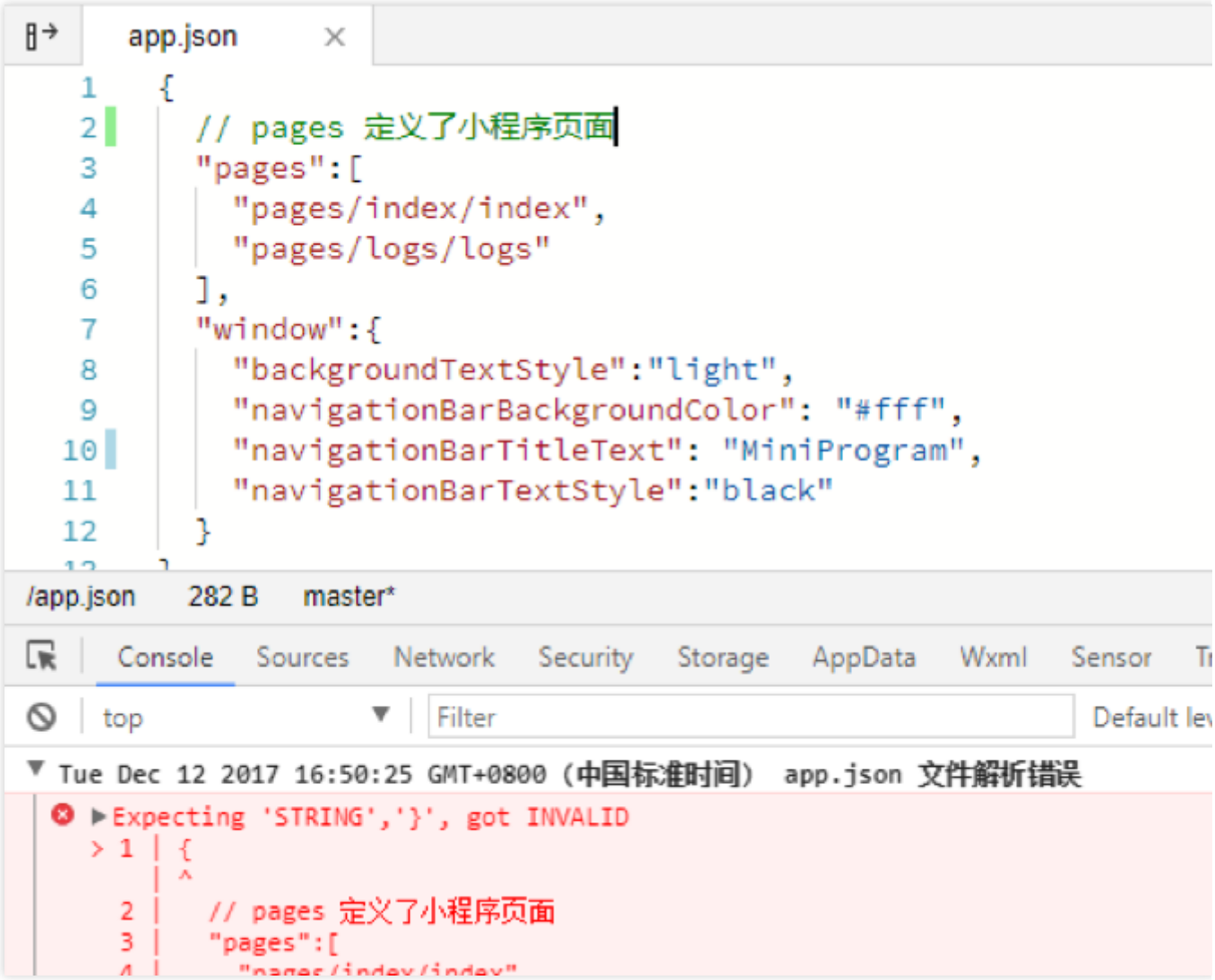
1. 数字，包含浮点数和整数；
2. 字符串，需要包裹在双引号中；
3. Bool值，true 或者 false；
4. 数组，需要包裹在方括号中 []；
5. 对象，需要包裹在大括号中 {}；
6. Null；

其他任何格式都会触发报错，例如 JavaScript 中的 undefined 。

版权所有：腾讯云计算（北京）有限责任公司



还需要注意的是 JSON 文件中无法使用注释，试图添加注释将会引发报错。


app.json
{
// pages 定义了小程序页面
"pages":[
"pages/index/index",
"pages/logs/logs"
],
"window":{
"backgroundTextStyle":"light",
"navigationBarBackgroundColor": "#fff",
"navigationBarTitleText": "MiniProgram",
"navigationBarTextStyle":"black"
}
/app.json 282 B master*
Console Sources Network Security Storage AppData Wxml Sensor
top
Filter
Default lev
Tue Dec 12 2017 16:50:25 GMT+0800 (中国标准时间) app.json 文件解析错误
Expecting 'STRING','}', got INVALID
> 1 | {
| ^
2 | // pages 定义了小程序页面
3 | "pages":[

WXML 模板

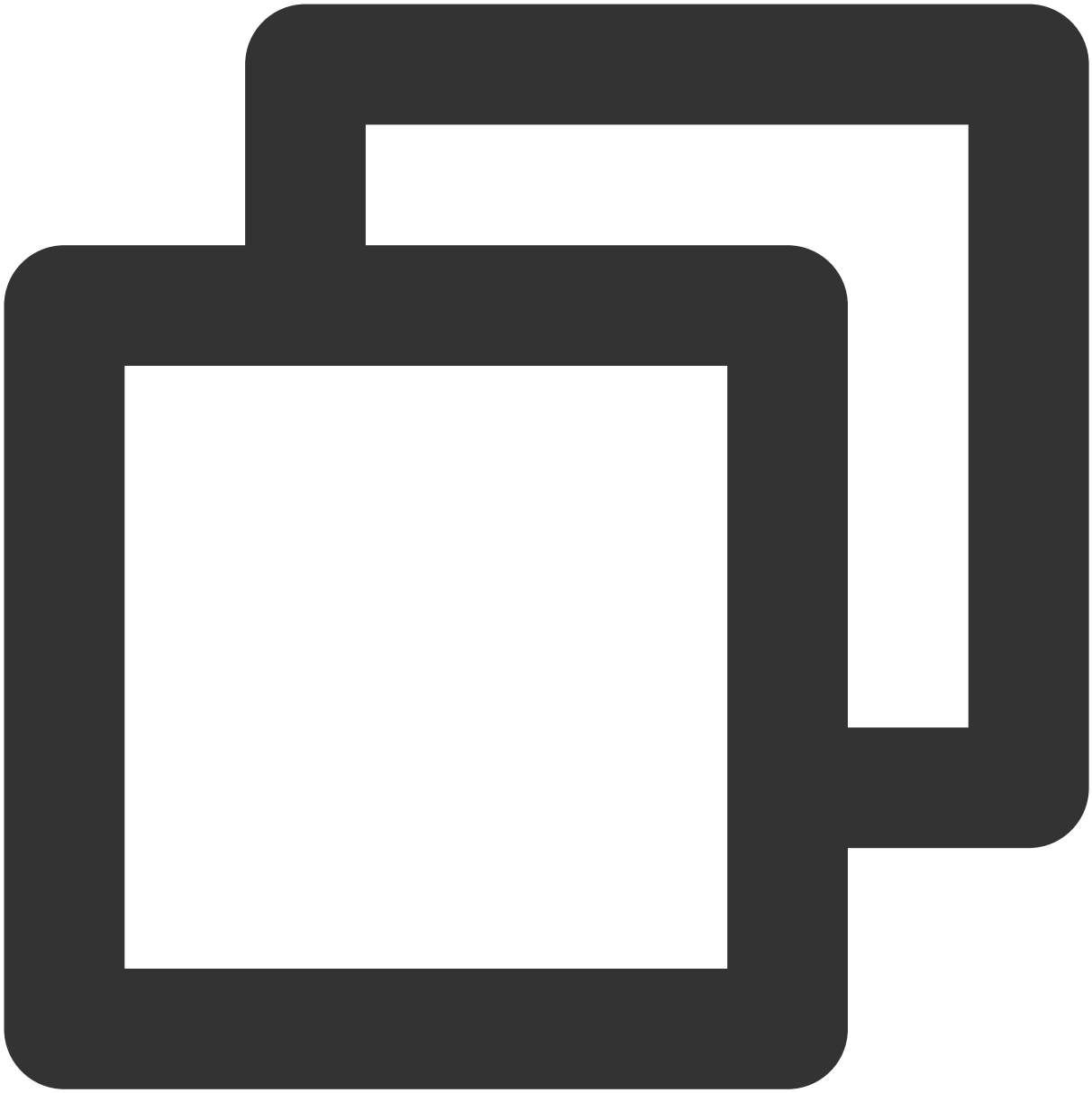
最近更新时间：2024-05-10 11:20:51

WXML 全称是 WeiXin Markup Language，是小程序框架设计的一套标签语言，结合小程序的基础组件、事件系统，可以构建出页面的结构。
打开开发工具的编辑器，在根目录下找到 app.json 文件，双击打开，在 "pages/index/index" 上新增一行 "pages/wxml/index" 保存文件。模拟器刷新后，读者可以在编辑器中找到 pages/wxml/index.wxml 文件，本小结的学习通过修改这个文件来完成。

介绍

WXML 文件后缀名是 `.wxml`，打开 pages/wxml/index.wxml 文件，有过 HTML 的开发经验的读者应该会很熟悉这种代码的书写方式，简单的 WXML 语句在语法上同 HTML 非常相似。

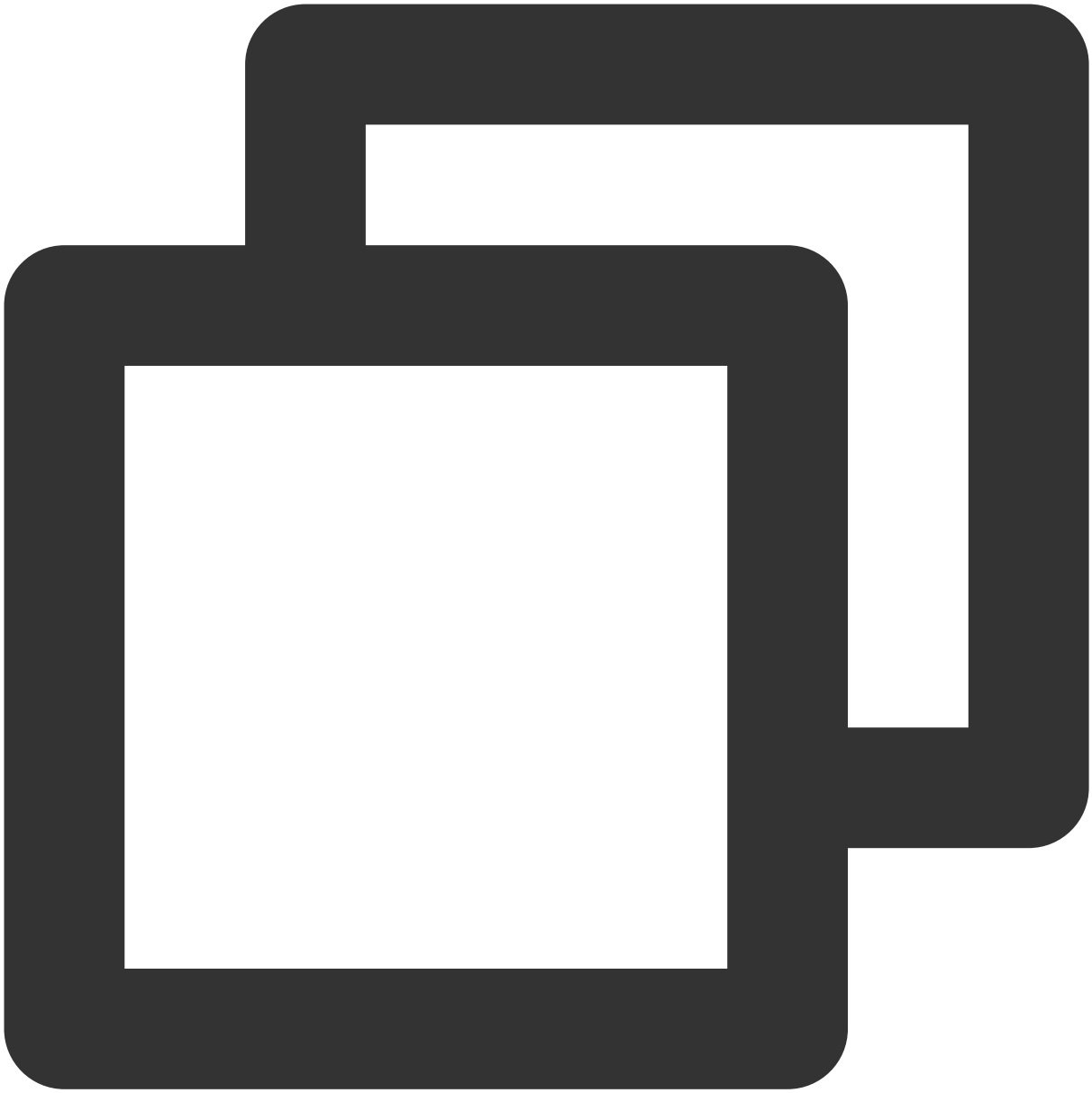
版权所有：腾讯云计算（北京）有限责任公司



```
<!--pages/wxml/index.wxml-->

<text>pages/wxml/index.wxml</text>
```

不带有任何逻辑功能的 WXML 基本语法如下：

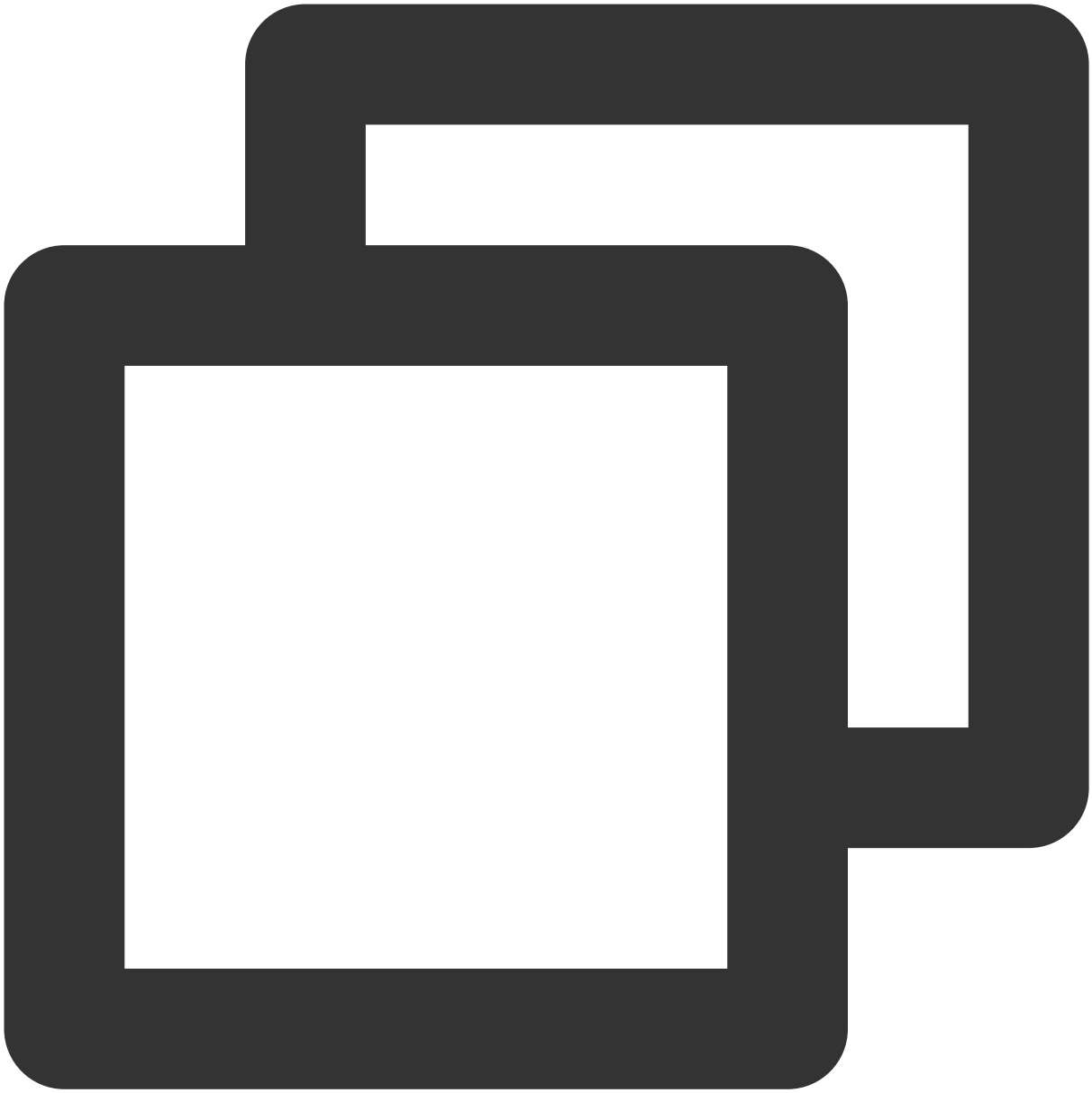


```
<!-- 在此处写注释 -->

<标签名 属性名1="属性值1" 属性名2="属性值2" ...> ...</标签名>
```

一个完整的 WXML 语句由一段开始标签和一段结束标签组成，在标签中可以是内容，也可以是其他的 WXML 语句，这一点上同 HTML 是一致的。有所不同的是，WXML 要求标签必须是严格闭合的，没有闭合将会导致编译错误，如下代码所示：

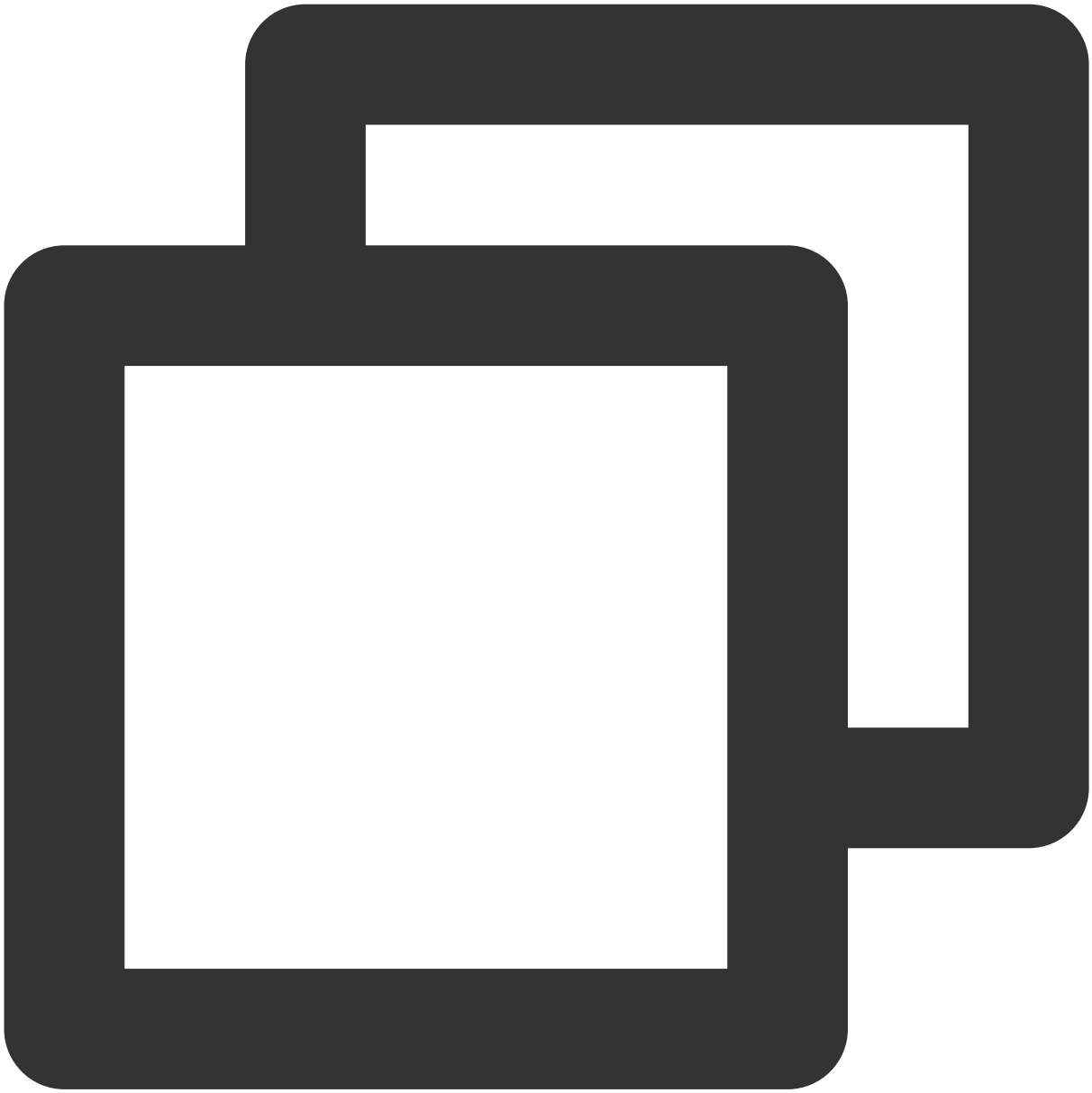
版权所有：腾讯云计算（北京）有限责任公司



```
<text>hello world


<!--
text 没有闭合，导致编译错误：
VM148:2 ./pages/wxml/index.wxml
 end tag missing, near text
> 1 | <text>hello world
    | ^
-->
```

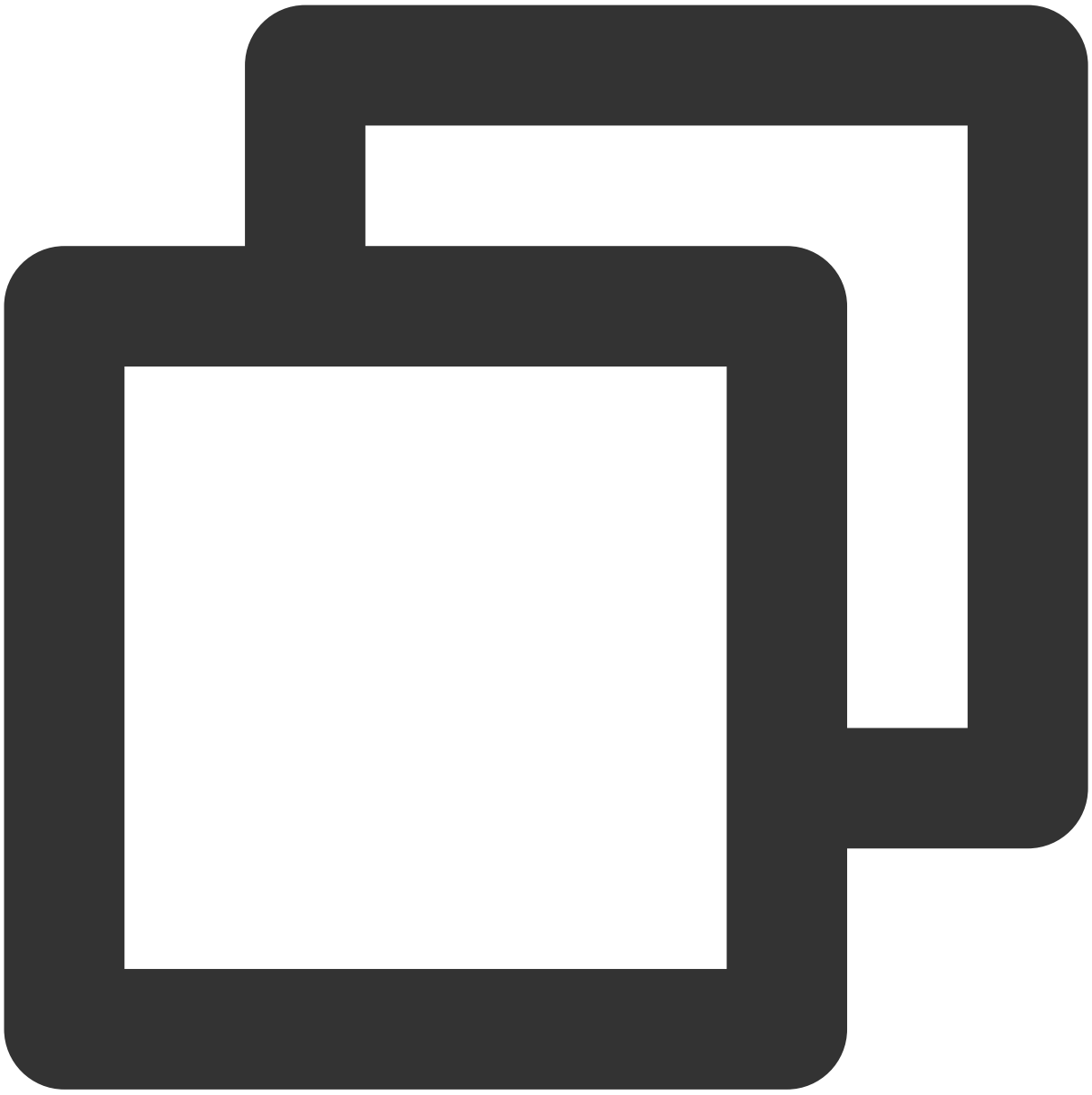
标签可以拥有属性，属性提供了有关的 WXML元素更多信息。属性总是定义在开始标签中，除了一些特殊的属性外，其余属性的格式都是key="value" 的方式成对出现。需要注意的是，WXML 中的属性是大小写敏感的，也就是说 class 和 Class 在 WXML 中是不同的属性，下方代码是一个文本标签的示例。



```
<!--一个简单的文本标签 -->
<text>hello world</text>

<!-- view 中包含了 text 标签 -->
<view>
  <text>hello world</text>
</view>
```

带属性的图片标签的例子如下所示：



```
<image class="userinfo-avatar" src="./image/a.png" ></image>
```

数据绑定

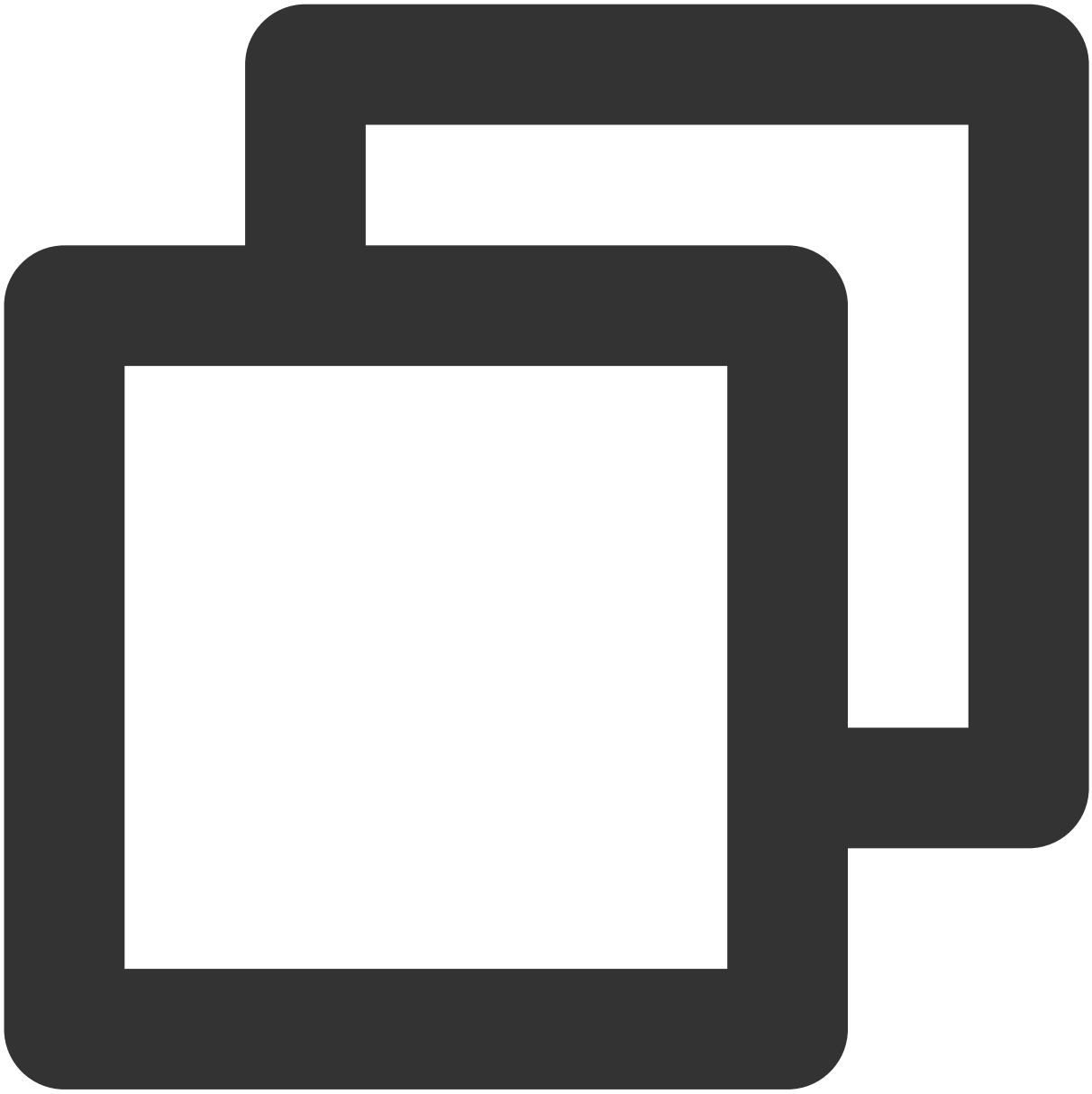
用户界面呈现会因为当前时刻数据不同而有所不同，或者是因为用户的操作发生动态改变，这就要求程序的运行过程中，要有动态的去改变渲染界面的能力。在 Web 开发中，开发者使用 JavaScript 通过 Dom 接口来完成界面的实

版权所有：腾讯云计算（北京）有限责任公司

时更新。在小程序中，使用 WXML 语言所提供的数据绑定功能，来完成此项功能。

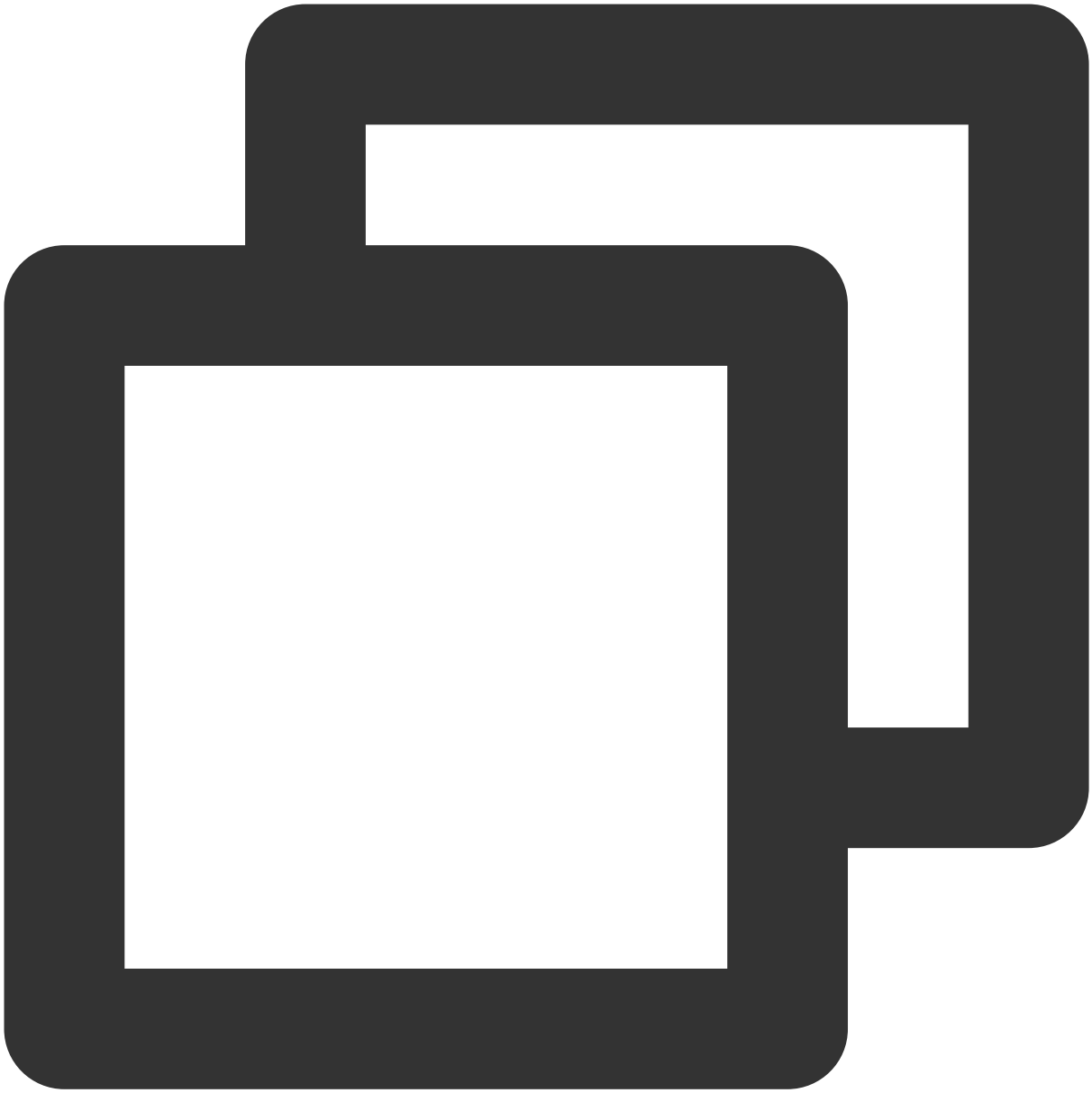
先看一个简单的例子。

将 pages/wxml/index.wxml 文件的内容做一些简单的修改，如下代码所示：



```
<!--pages/wxml/index.wxml-->
<text>当前时间：{{time}}</text>
```

保存后工具刷新，模拟器并没有显示出当前的时间，这是因为我们并没有给 time 设置任何初始值，请打开 pages/wxml/index.js 文件，在 data 的大括号中加入：`time: (new Date()).toString()`，如下代码所示：

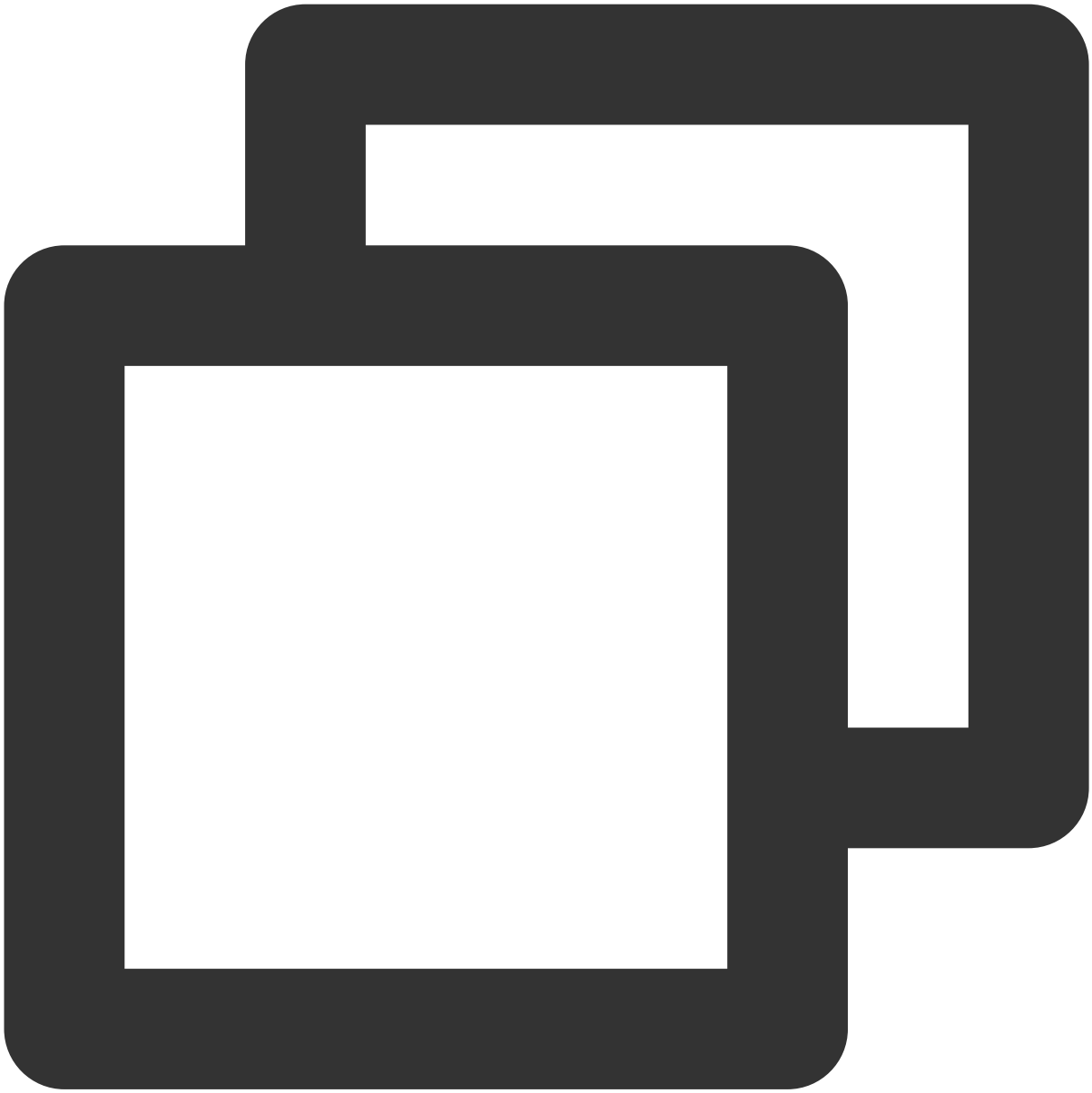


```
// pages/wxml/index.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    time: (new Date()).toString()
  },
})
```

保存，模拟器刷新后正确的展示了当前时间，并且每次编译时间都会被更新。

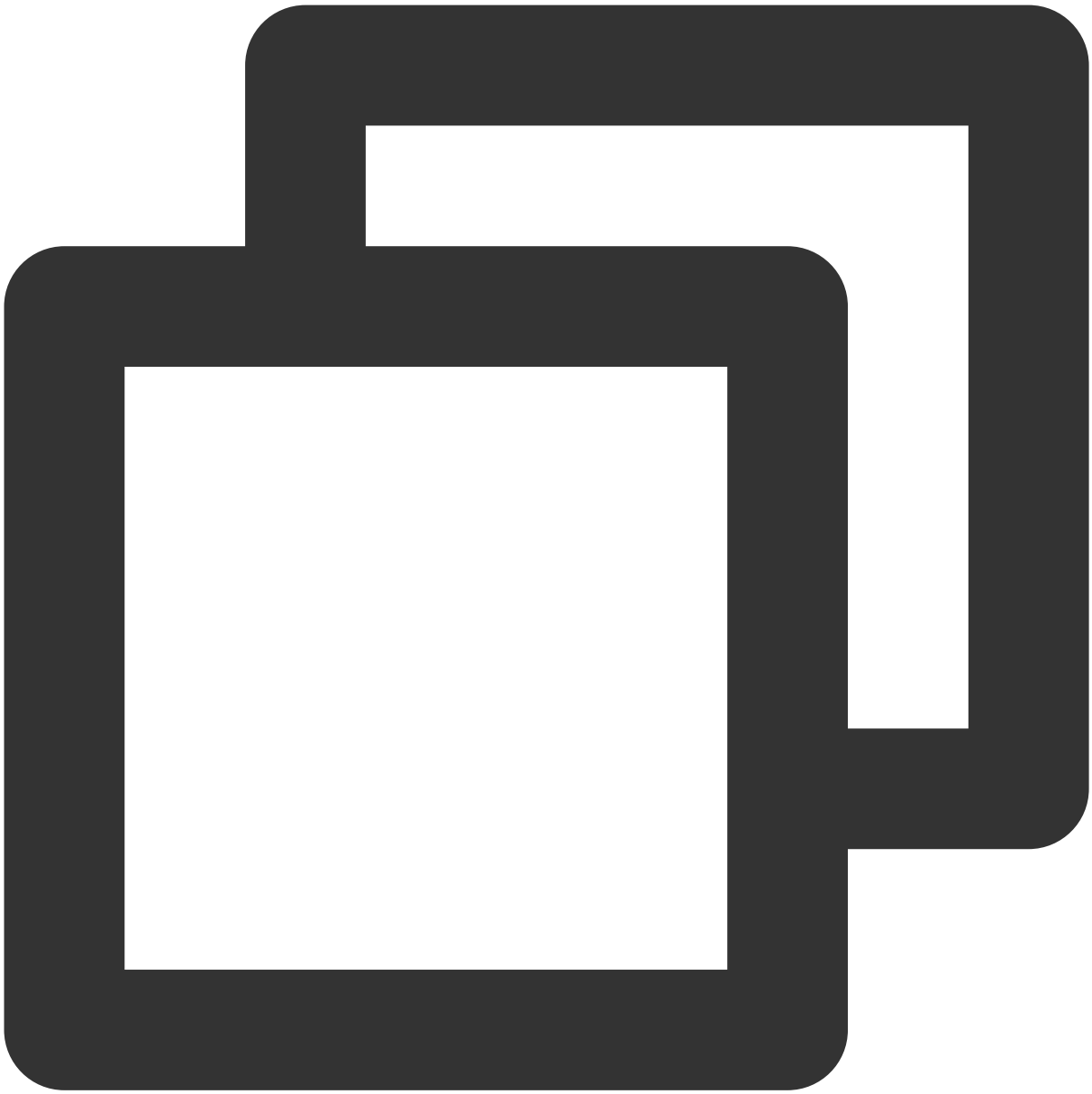
版权所有：腾讯云计算（北京）有限责任公司

WXML 通过 {{变量名}} 来绑定 WXML 文件和对应的 JavaScript 文件中的 data 对象属性。
后文中为了保持简单，通过以下格式来展示上述的代码逻辑，使用第一段注释来表明 WXML 对应的脚本文件中的 data 结构，如下代码所示：



```
<!--
{
  time: (new Date()).toString()
}
-->
<text>当前时间：{{time}}</text>
```

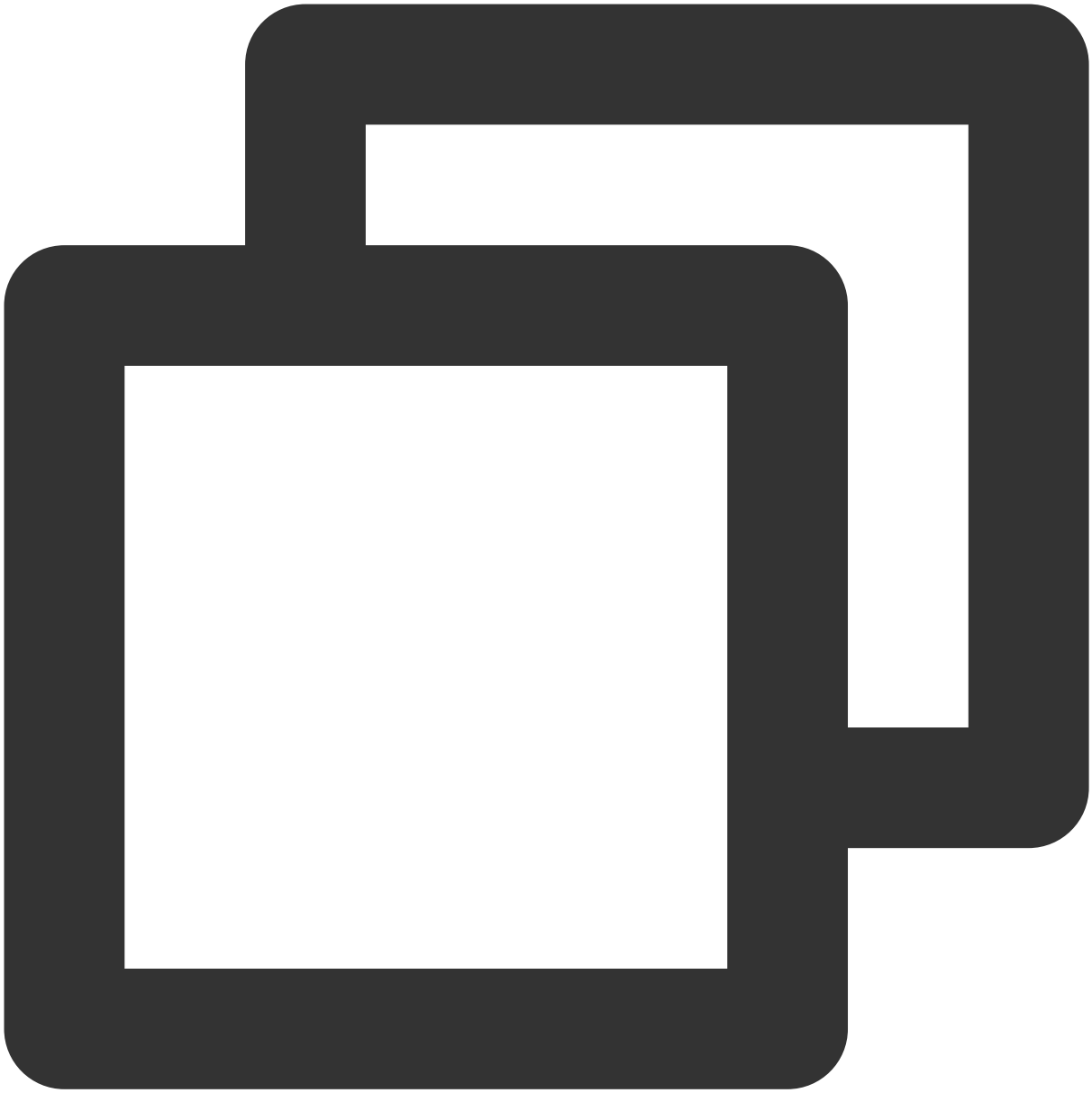
属性值也可以动态的去改变，有所不同的是，属性值必须被包裹在双引号中，如下代码所示：



```
<!-- 正确的写法 -->
<text data-test="{{test}}"> hello world</text>

<!-- 错误的写法  -->
<text data-test={{test}}> hello world </text >
```

需要注意的是**变量名是大小写敏感的，也就是说 {{name}} 和 {{Name}} 是两个不同的变量**，如下代码所示：

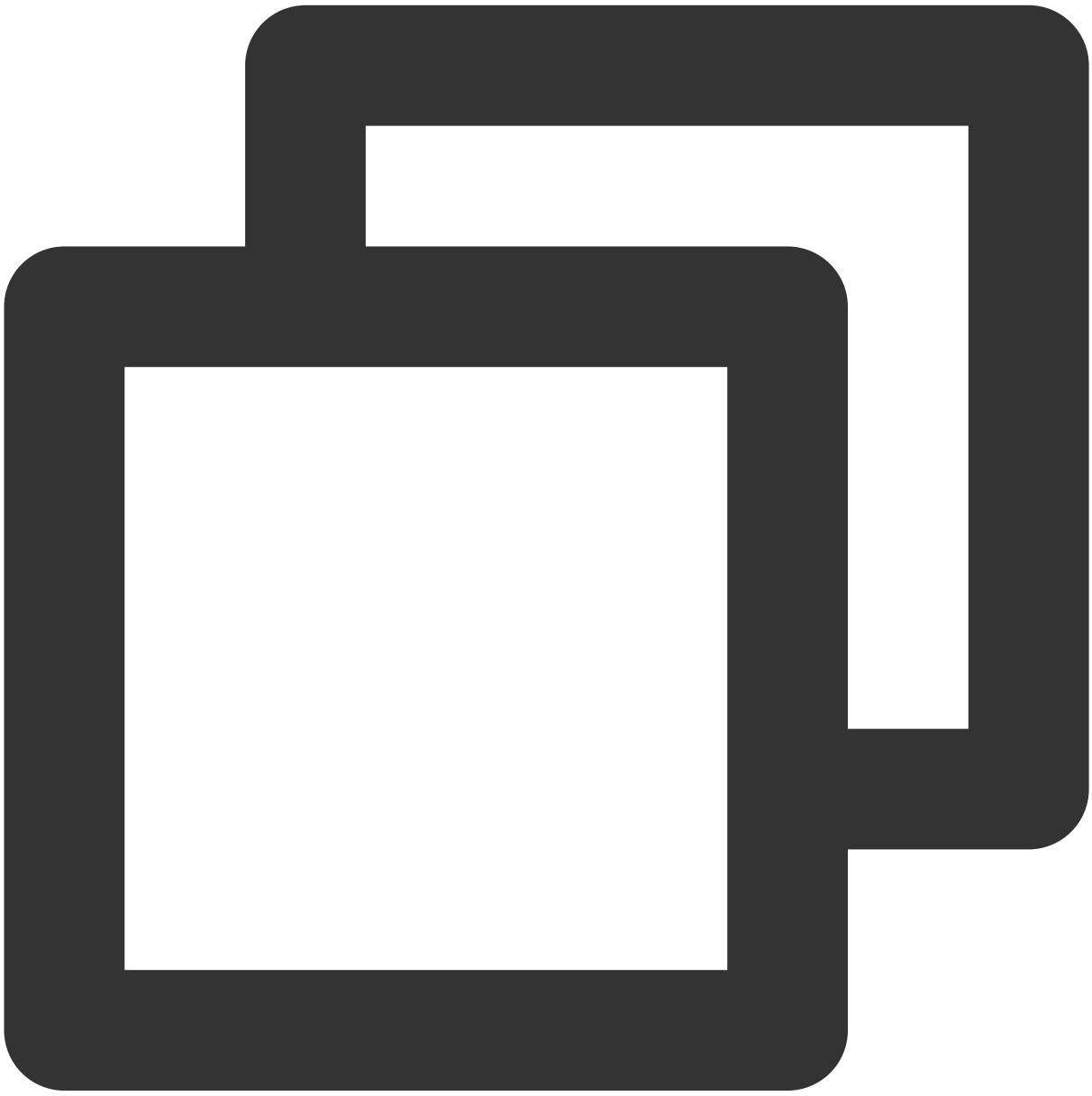


```
<!--
{
  w: 'w',
  W: 'W'
}
-->


<view>{{w}}</view>
<view>{{W}}</view>
```

```
<!-- 输出
w
W
-->
```

还需要注意，**没有被定义的变量的或者是被设置为 undefined 的变量不会被同步到 wxml 中**，如下代码所示：



```
<!--
{
  var2: undefined,
  var3: null,
```

```
  var4: "var4"
}
-->


<view>{{var1}}</view>
<view>{{var2}}</view>
<view>{{var3}}</view>
<view>{{var4}}</view>

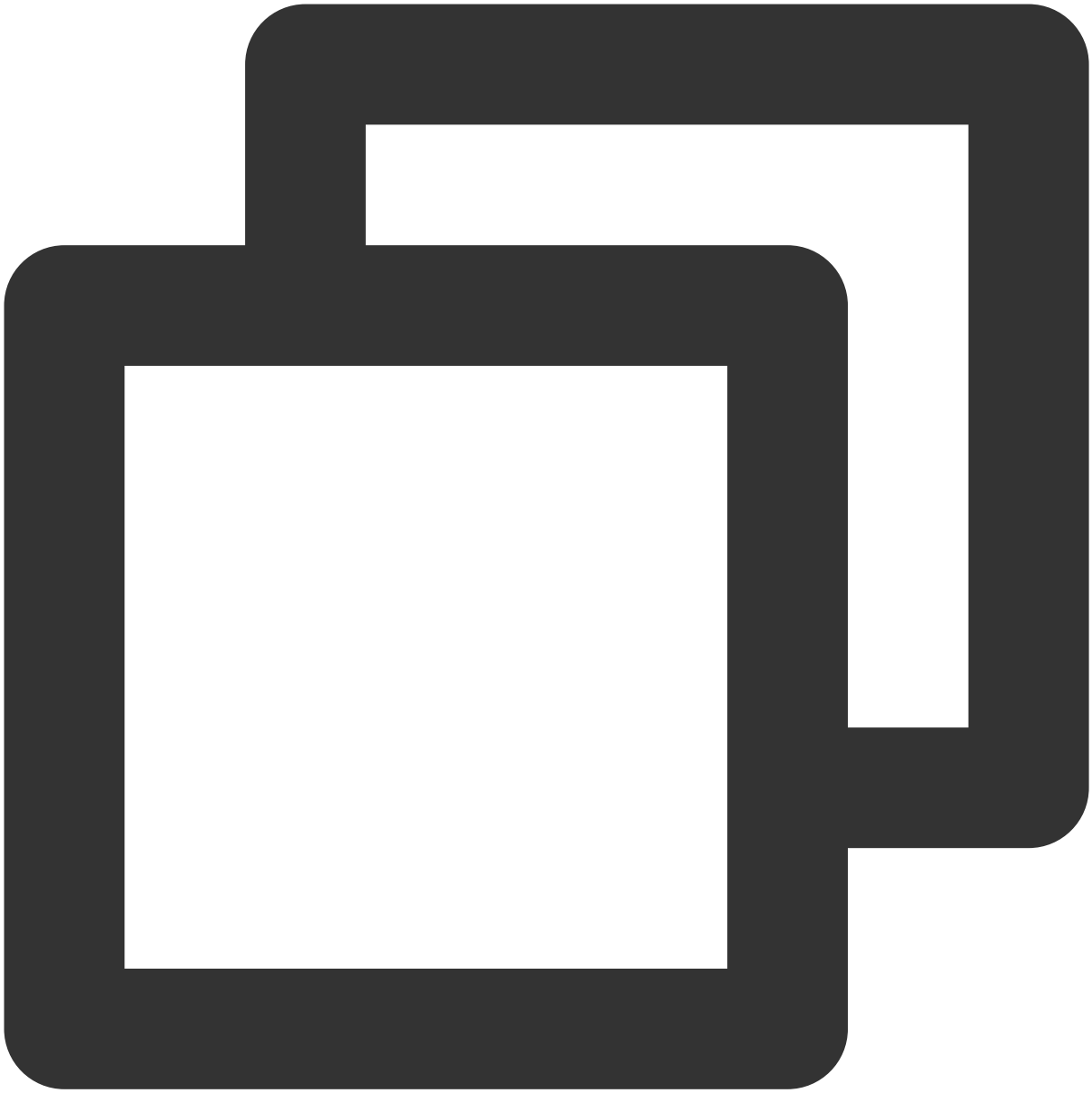

<!--
输出：
null
var4
-->
```

关于数据绑定的概念在第三章中有更为详细的介绍。

逻辑语法

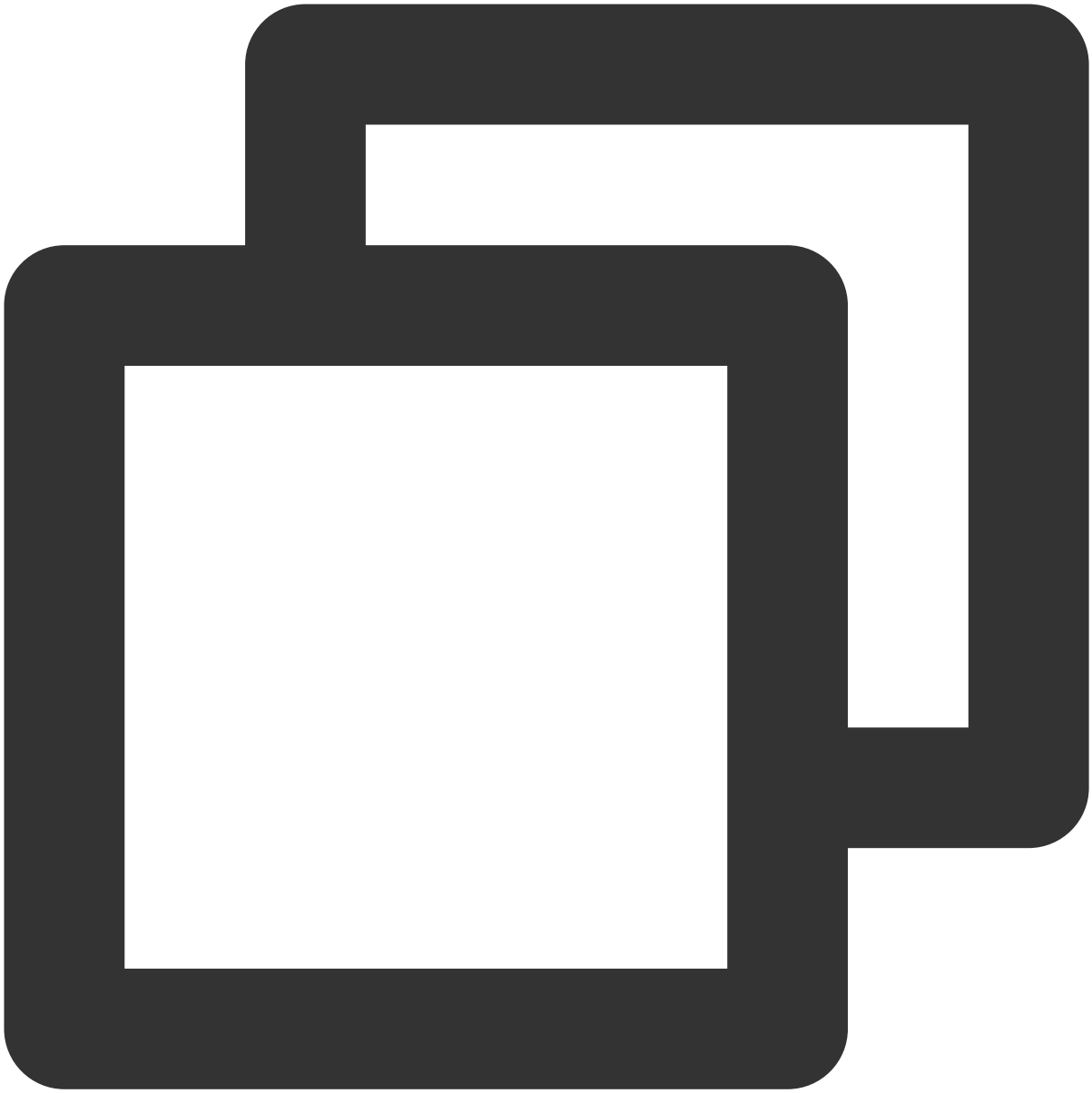
通过 {{ 变量名 }} 语法可以使得 WXML 拥有动态渲染的能力，除此外还可以在 {{ }} 内进行简单的逻辑运算。

三元运算：



```
<!-- 根据 a 的值是否等于 10 在页面输出不同的内容 -->
<text>{{ a === 10? "变量 a 等于10": "变量 a 不等于10"}}</text>
```

算数运算：

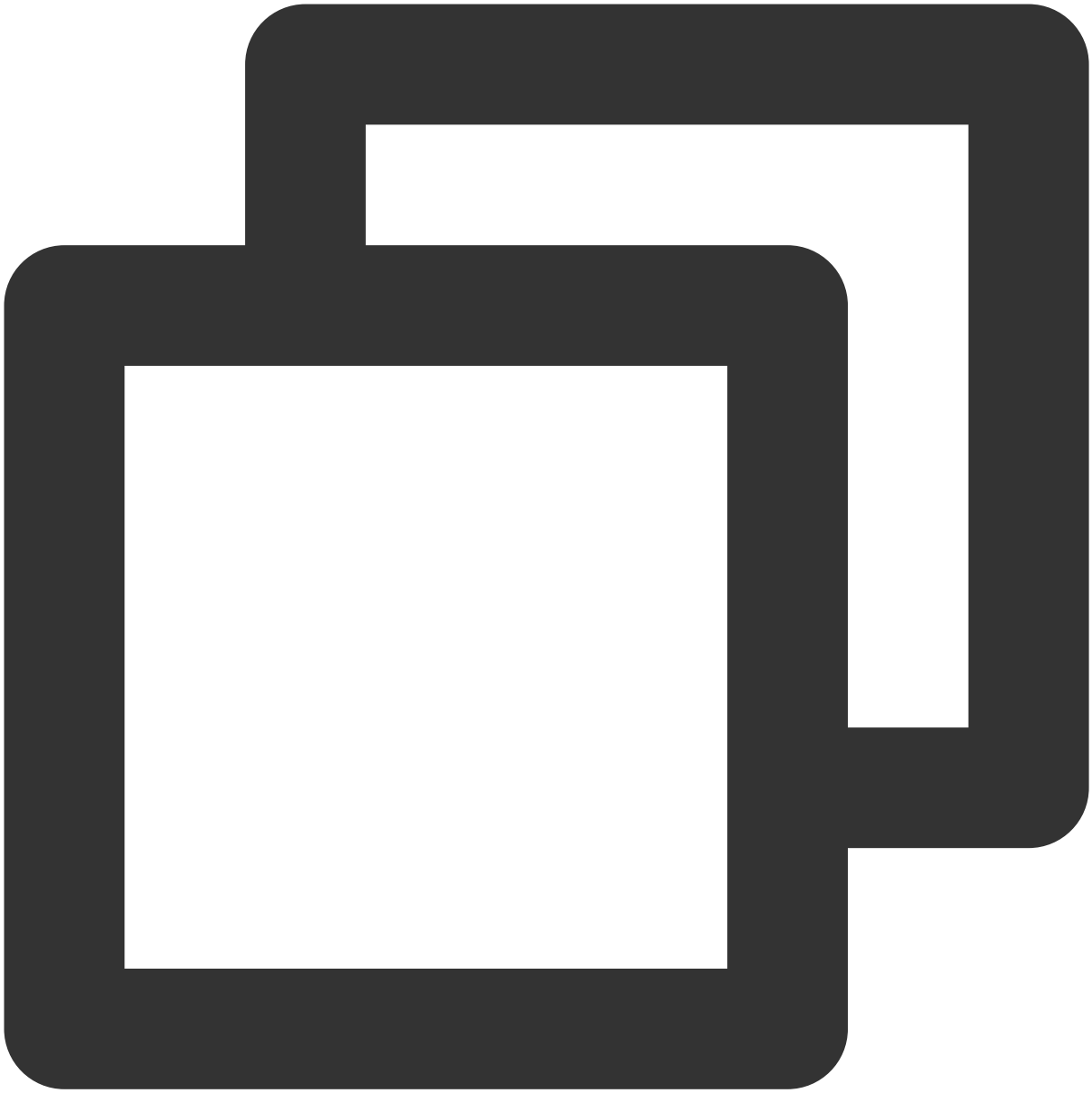


```
<!--
{ a: 1,  b: 2, c: 3 }
-->


<view> {{a + b}} + {{c}} + d </view>


<!-- 输出 3 + 3 + d -->
```

类似于算数运算，还支持字符串的拼接，如下代码所示：

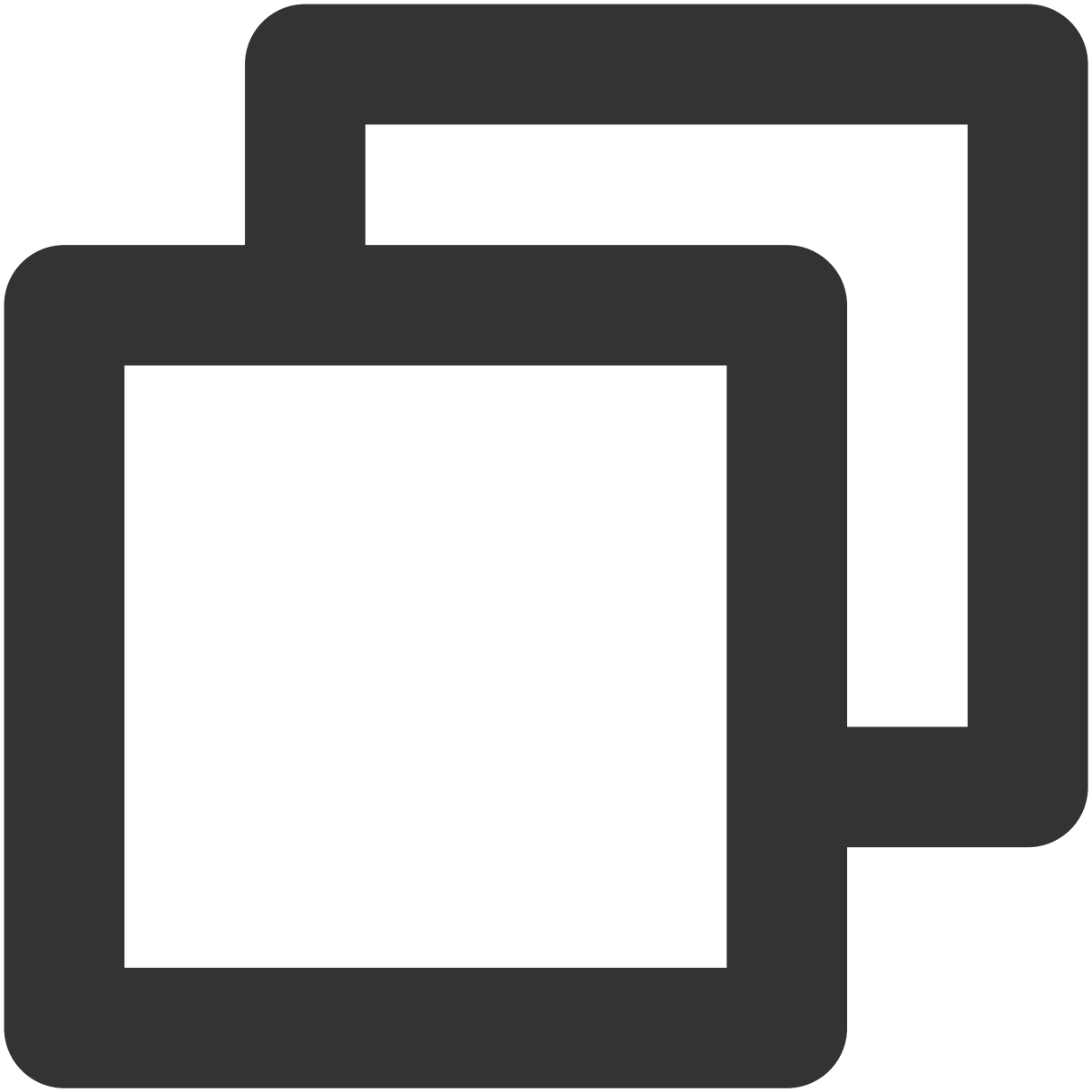


```
<!--
{ name: 'world' }
-->

<view>{{"hello " + name}}</view>

<!-- 输出 hello world -->
```

{{ }}中还可以直接放置数字、字符串或者是数组，如下代码所示：



```
<text>{{[1,2,3]}}</text>

<!-- 输出 1,2,3 -->

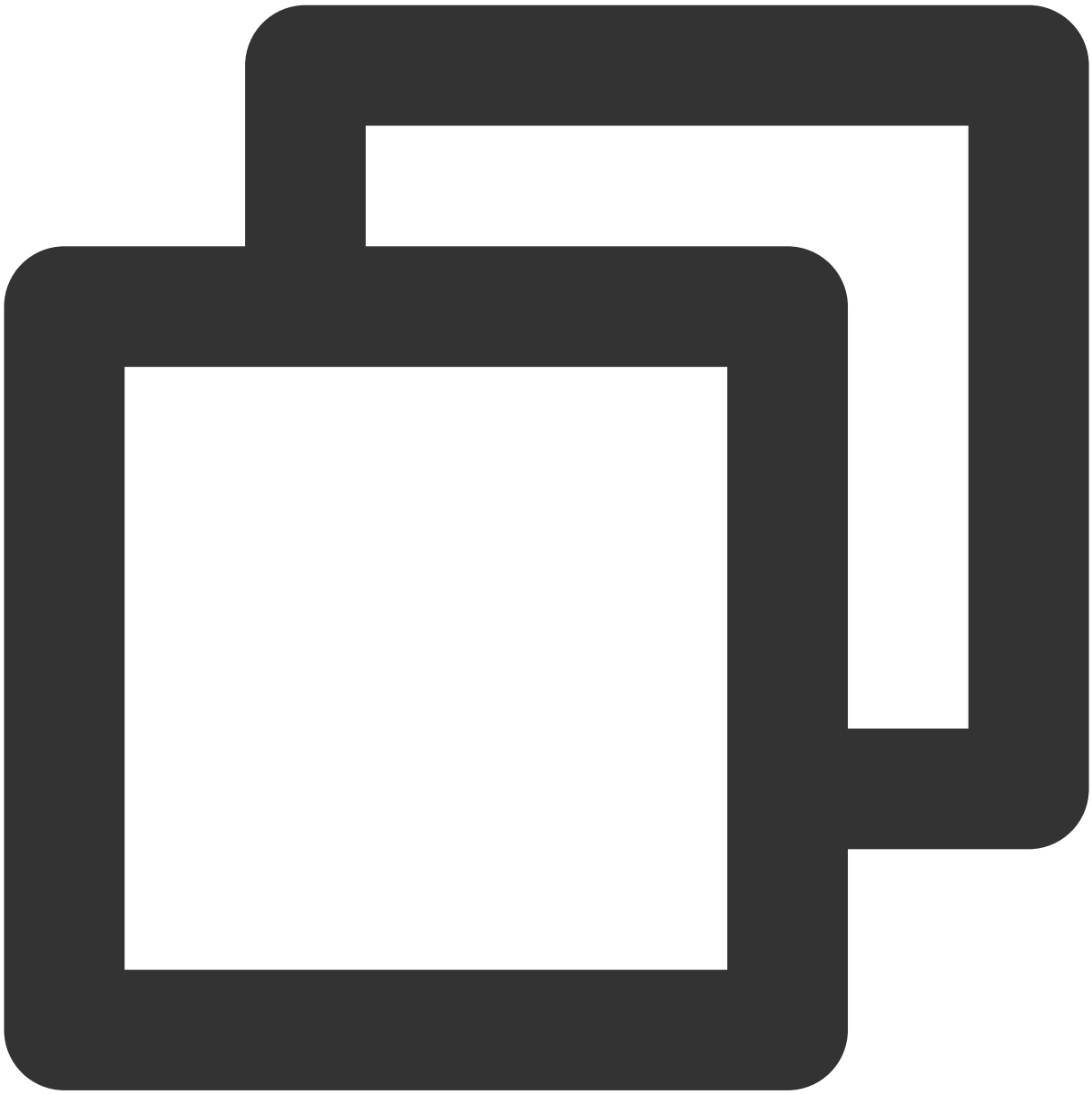


<text>{{"hello world"}}</text>

<!-- 输出 hello world -->
```

条件逻辑

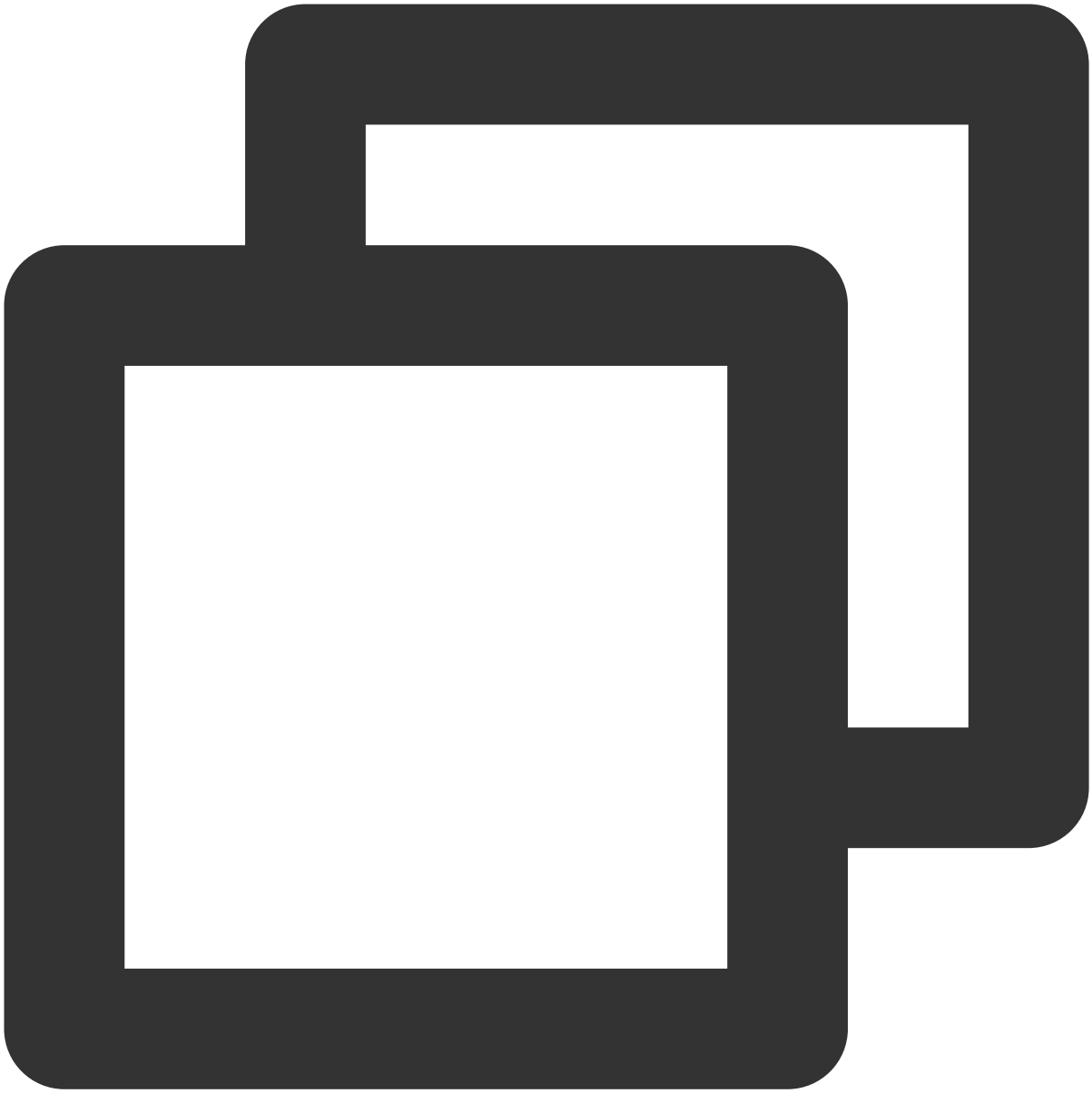
WXML 中，使用 wx:if="{{condition}}" 来判断是否需要渲染该代码块：



```
<view wx:if="{{condition}}"> True </view>
```

使用 wx:elif 和 wx:else 来添加一个 else 块：

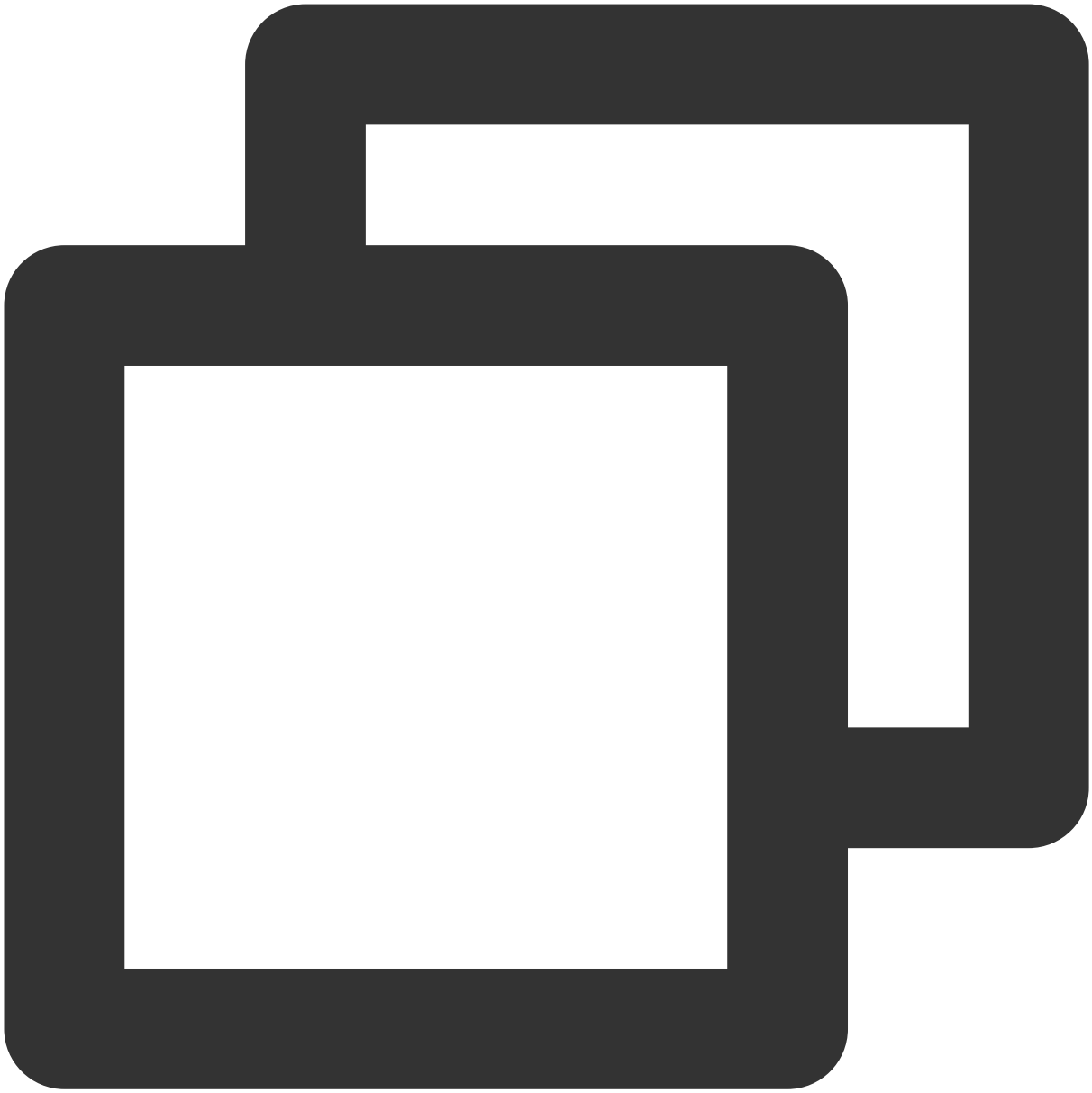
版权所有：腾讯云计算（北京）有限责任公司



```
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>
```

因为 wx:if 是一个控制属性，需要将它添加到一个标签上。如果要一次性判断多个组件标签，可以使用一个 <block/> 标签将多个组件包装起来，并在上边使用 wx:if 控制属性。

版权所有：腾讯云计算（北京）有限责任公司

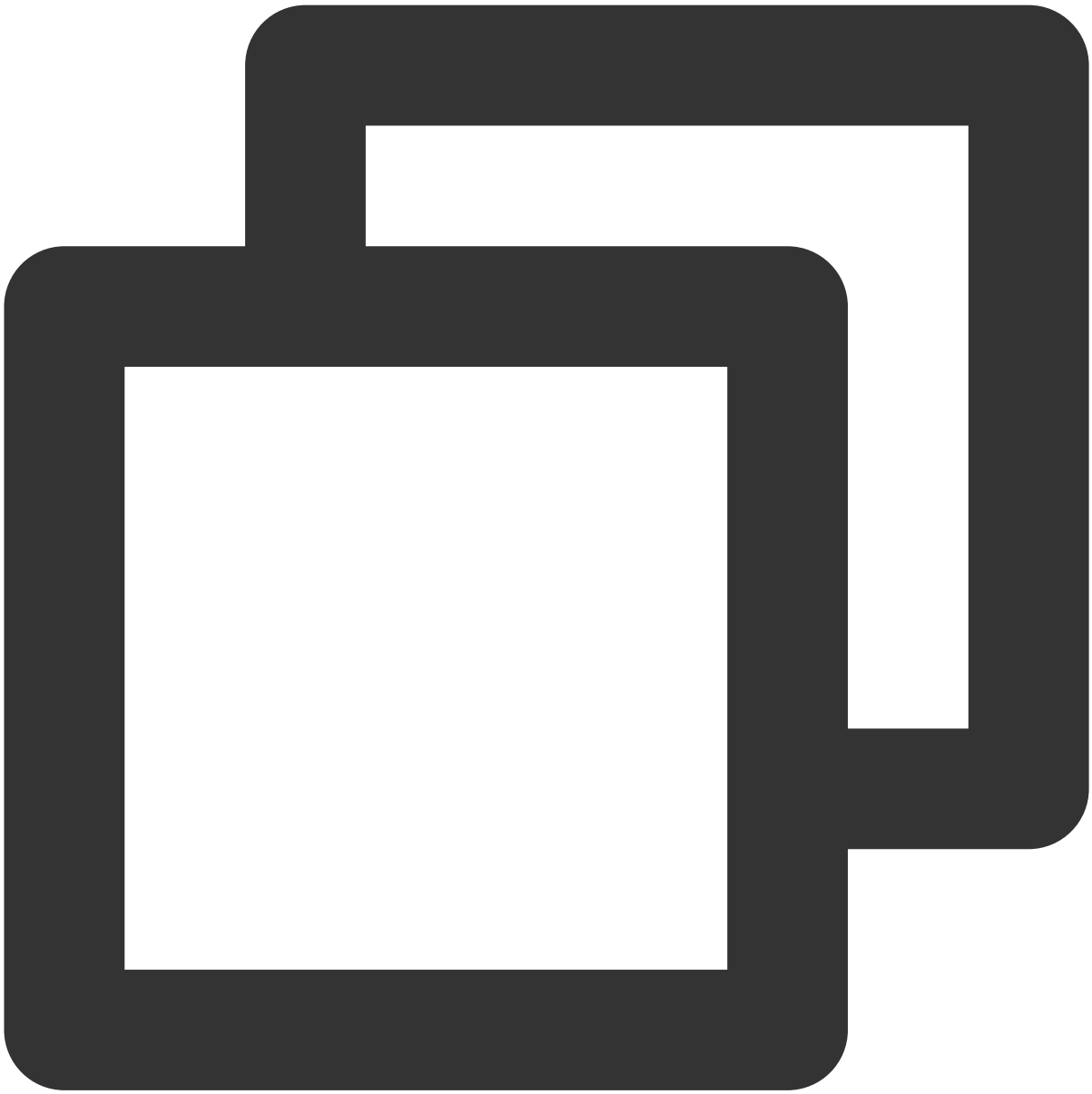


```
<block wx:if="{{true}}">
  <view> view1 </view>
  <view> view2 </view>
</block>
```

列表渲染

版权所有：腾讯云计算（北京）有限责任公司

在组件上使用 wx:for 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。默认数组的当前项的下标变量名默认为 index，数组当前项的变量名默认为 item，如下代码所示：

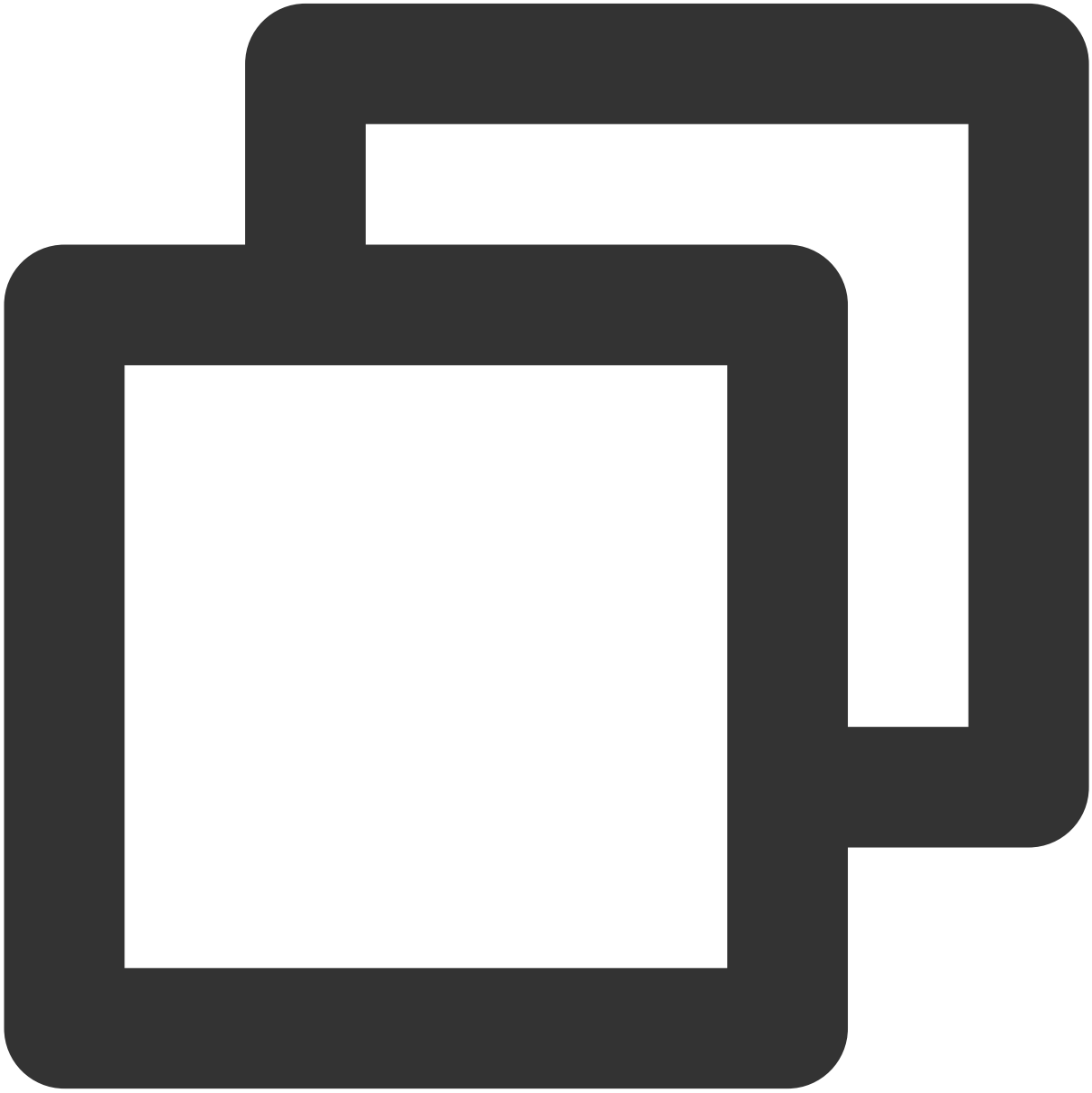


```
<!-- array 是一个数组 -->
<view wx:for="{{array}}">
  {{index}}: {{item.message}}
</view>

<!-- 对应的脚本文件
Page({
  data: {
```

```
    array: [{
      message: 'foo',
    }, {
      message: 'bar'
    }]
  }
})
-->
```

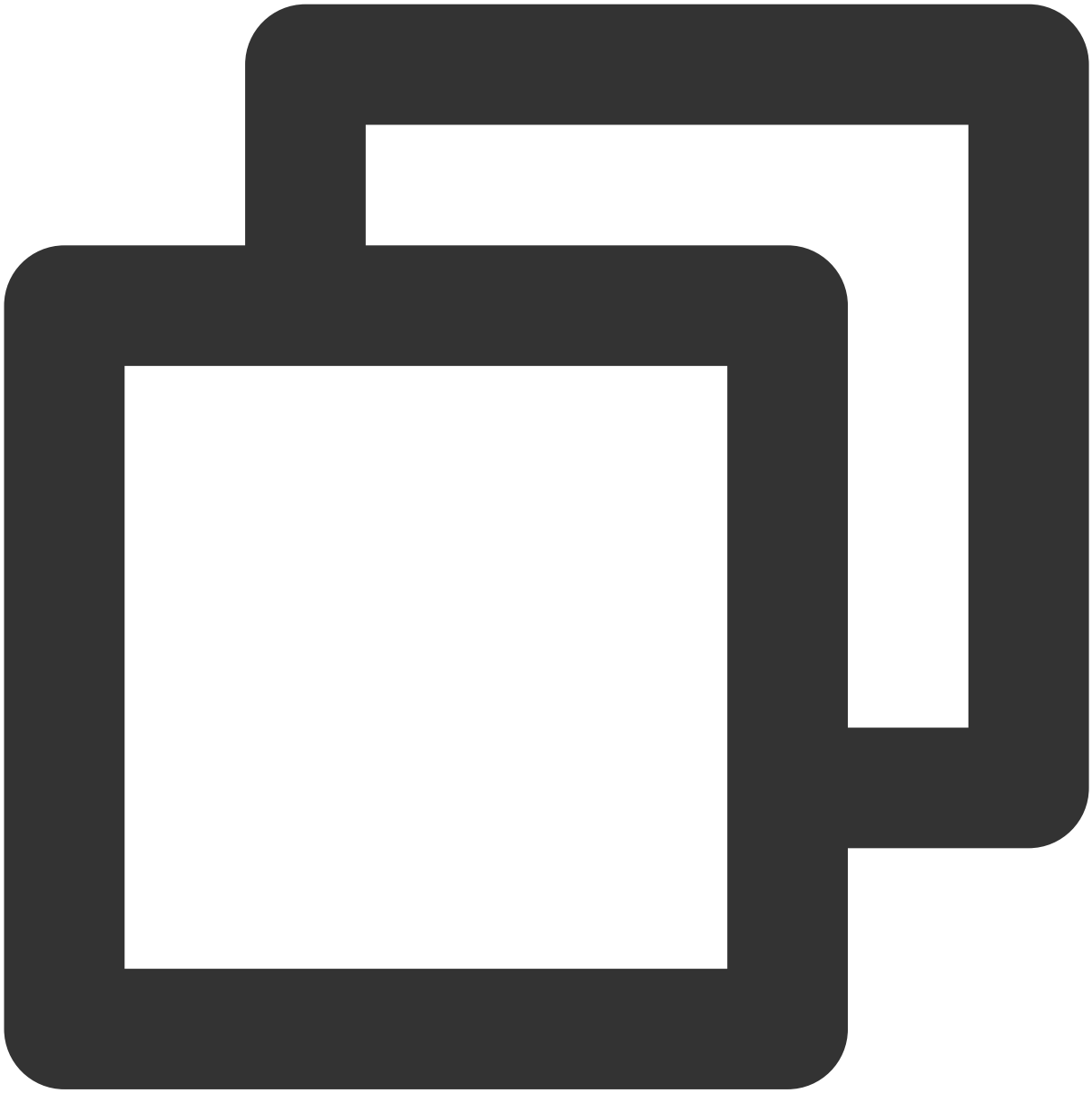
使用 wx:for-item 指定数组当前元素的变量名，使用 wx:for-index 指定数组当前下标的变量名：



```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
```

```
  {{idx}}: {{itemName.message}}
</view>
```

类似 block wx:if ，也可以将 wx:for 用在 <block/> 标签上，以渲染一个包含多节点的结构块，如下代码所示：



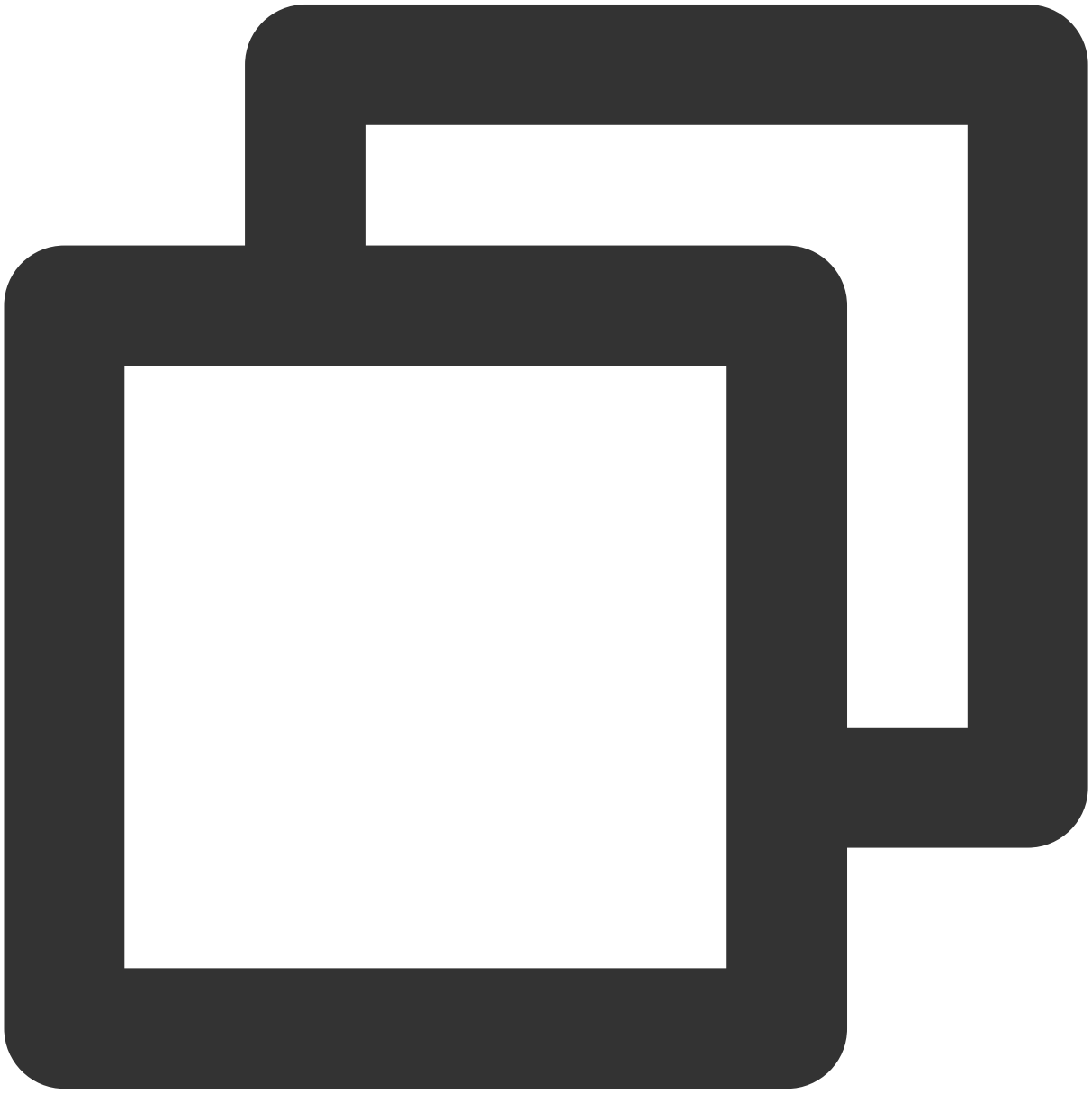
```
<block wx:for="{{[1, 2, 3]}}">
  <view> {{index}}: </view>
  <view> {{item}} </view>
</block>
```

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 `<input/>` 中的输入内容， `<switch/>` 的选中状态），需要使用 `wx:key` 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供：

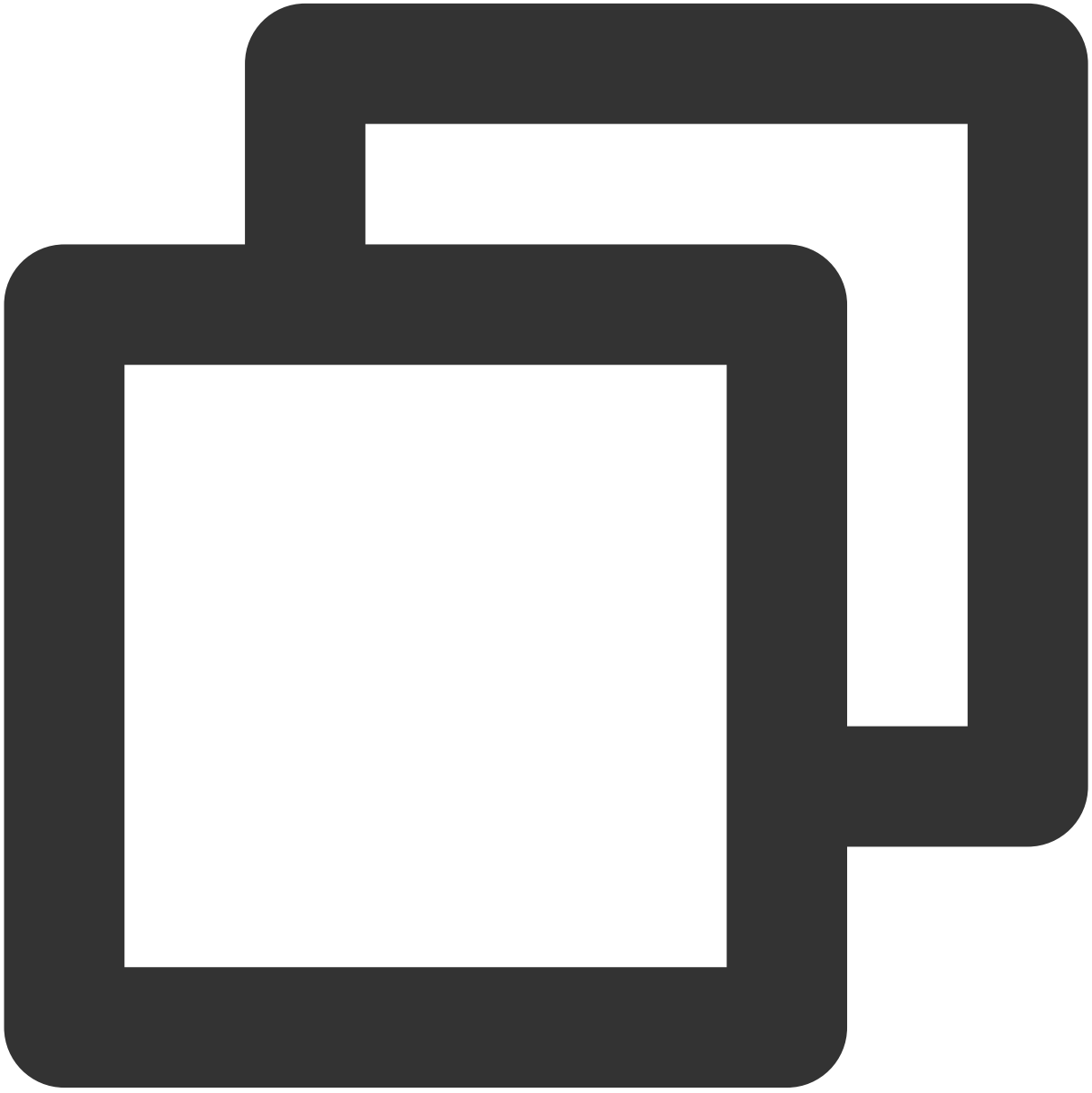
1. 字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。
2. 保留关键字 this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字，如：

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率，如下代码所示：



```
<switch wx:for="{{objectArray}}" wx:key="unique" > {{item.id}} </switch>
<button bindtap="switch"> Switch </button>
<button bindtap="addToFront"> Add to the front </button>


<switch wx:for="{{numberArray}}" wx:key="*this" > {{item}} </switch>
<button bindtap="addNumberToFront"> Add Number to the front </button>
```

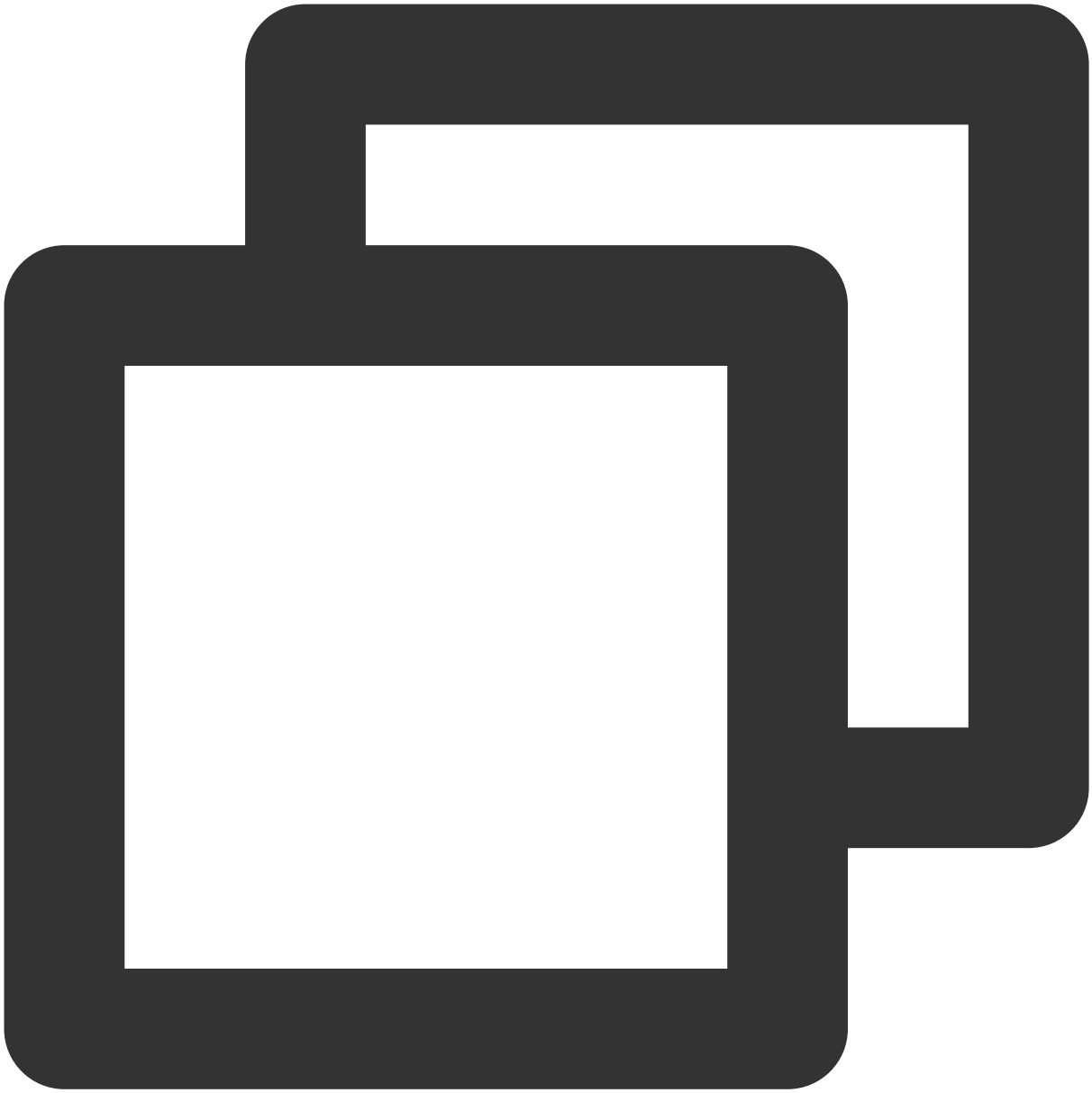


```
Page({
  data: {
    objectArray: [
```

```
        {id: 5, unique: 'unique_5'},
        {id: 4, unique: 'unique_4'},
        {id: 3, unique: 'unique_3'},
        {id: 2, unique: 'unique_2'},
        {id: 1, unique: 'unique_1'},
        {id: 0, unique: 'unique_0'},
      ],
      numberArray: [1, 2, 3, 4]
    },
    switch: function(e) {
      const length = this.data.objectArray.length
      for (let i = 0; i < length; ++i) {
        const x = Math.floor(Math.random() * length)
        const y = Math.floor(Math.random() * length)
        const temp = this.data.objectArray[x]
        this.data.objectArray[x] = this.data.objectArray[y]
        this.data.objectArray[y] = temp
      }
      this.setData({
        objectArray: this.data.objectArray
      })
    },
    addToFront: function(e) {
      const length = this.data.objectArray.length
      this.data.objectArray = [{id: length, unique: 'unique_' + length}].concat(this.
      this.setData({
        objectArray: this.data.objectArray
      })
    },
    addNumberToFront: function(e){
      this.data.numberArray = [ this.data.numberArray.length + 1 ].concat(this.data.n
      this.setData({
        numberArray: this.data.numberArray
      })
    }
  })
```

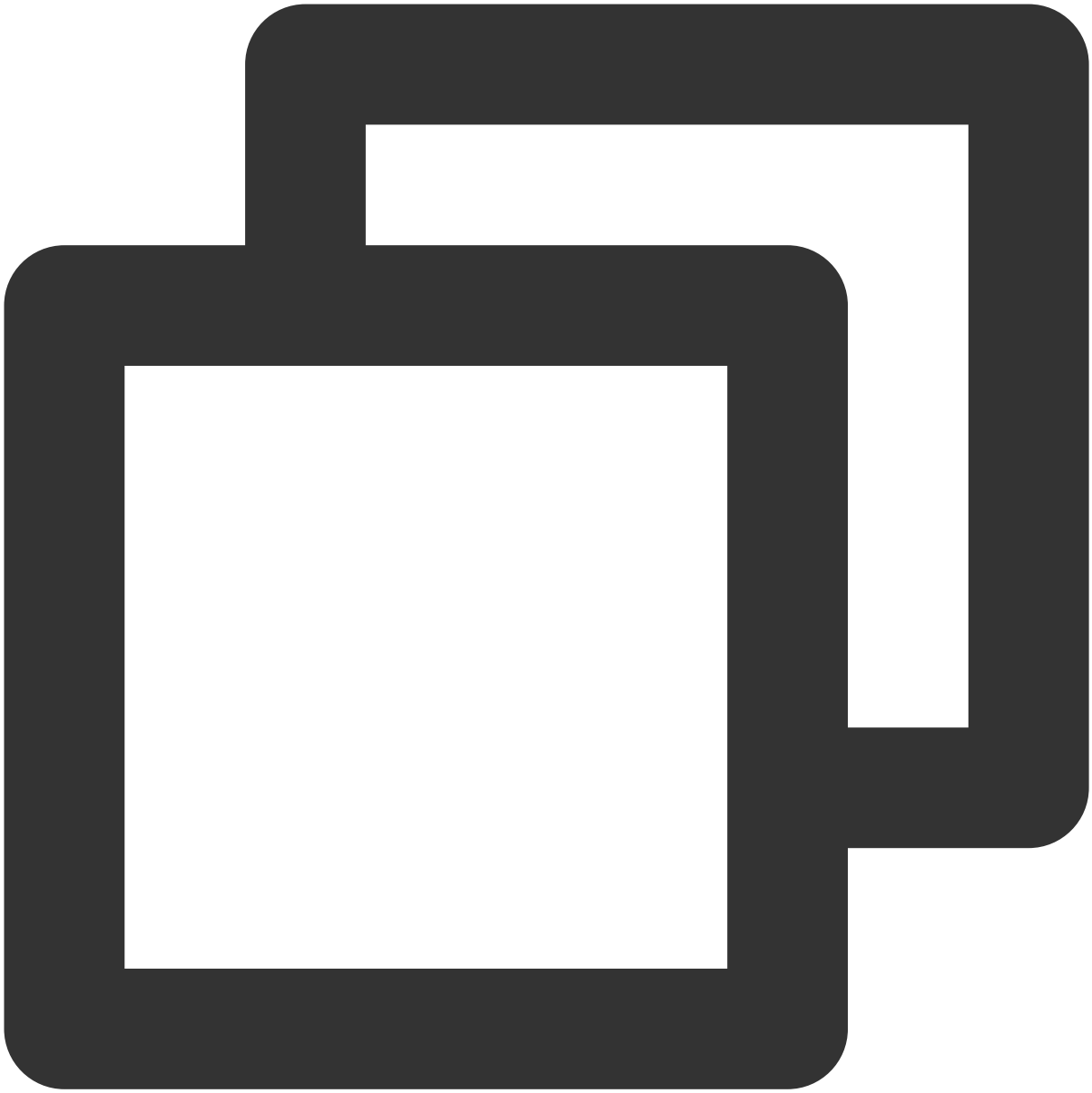
模板

WXML提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。使用 name 属性，作为模板的名字。然后在 <template/> 内定义代码片段，如下代码所示：



```
<template name="msgItem">
  <view>
    <text> {{index}}: {{msg}} </text>
    <text> Time: {{time}} </text>
  </view>
</template>
```

使用 is 属性，声明需要的使用的模板，然后将模板所需要的 data 传入，如下代码所示：



```
<!--
item: {
  index: 0,
  msg: 'this is a template',
  time: '2016-06-18'
}
-->

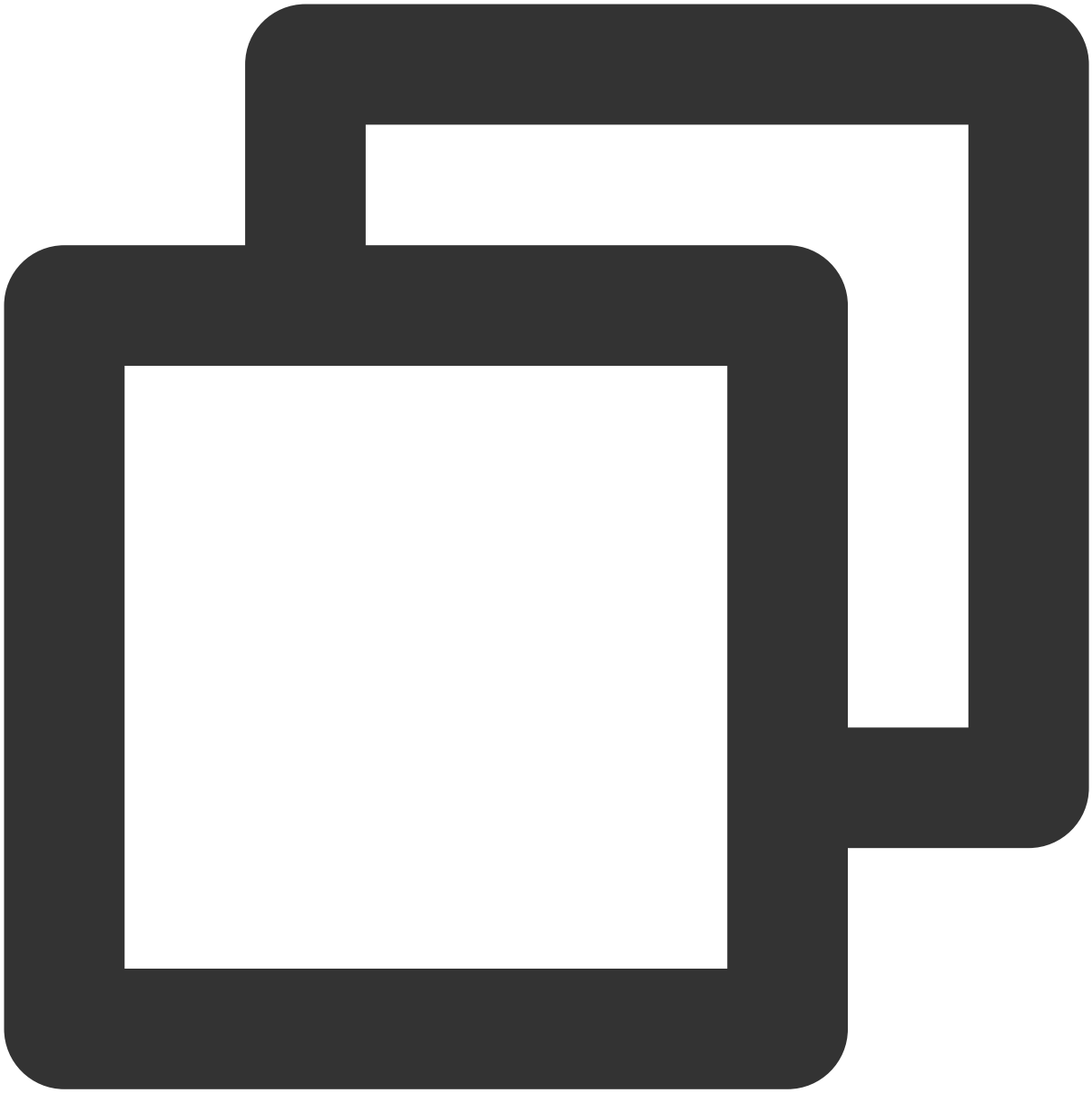

<template name="msgItem">
  <view>
```

```
    <text> {{index}}: {{msg}} </text>
    <text> Time: {{time}} </text>
  </view>
</template>


<template is="msgItem" data="{{...item}}"/>

<!-- 输出
0: this is a template Time: 2016-06-18
-->
```

使用 is 可以动态决定具体需要渲染哪个模板，如下代码所示：



```
<template name="odd">
  <view> odd </view>
</template>


<template name="even">
  <view> even </view>
</template>


<block wx:for="{{[1, 2, 3, 4, 5]}}">
```

版权所有：腾讯云计算（北京）有限责任公司

```
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}"/>
</block>

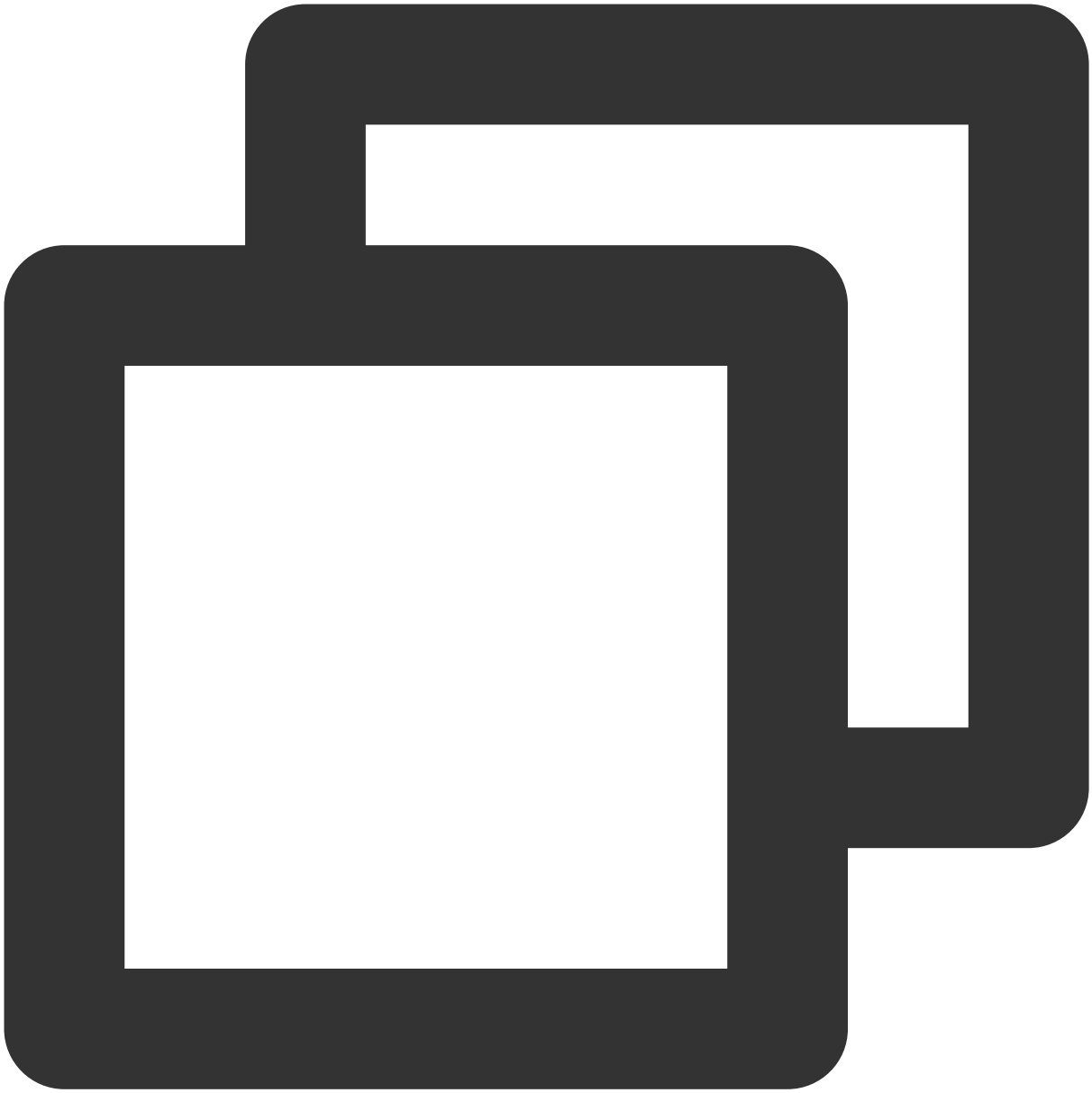


<!-- 输出
odd
even
odd
even
odd
-->
```

引用

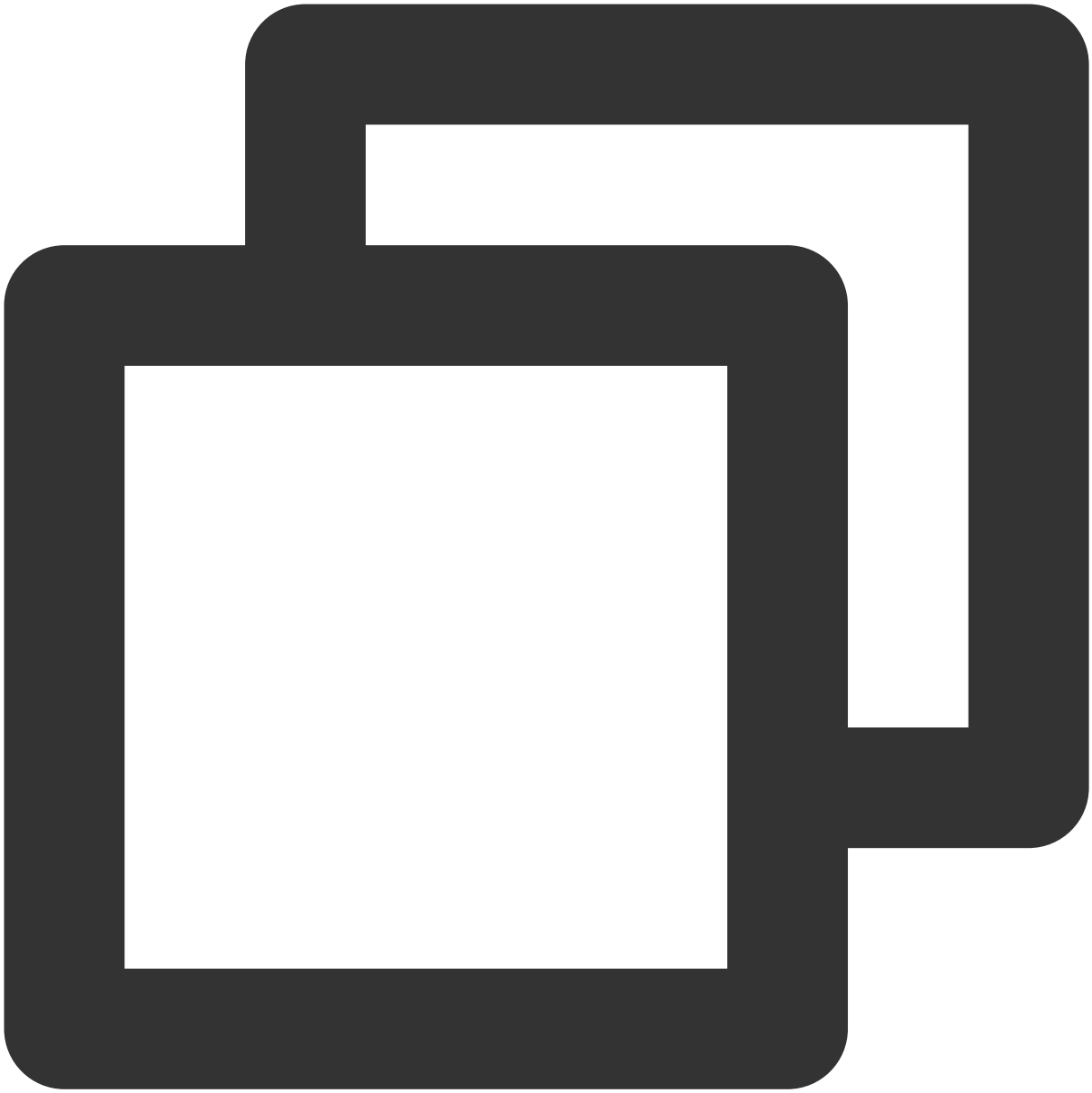
WXML 提供两种文件引用方式 import 和 include。

import 可以在该文件中使用目标文件定义的 template，如：在 item.wxml 中定义了一个叫 item 的 template。



```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 index.wxml 中引用了 item.wxml，就可以使用 item 模板：



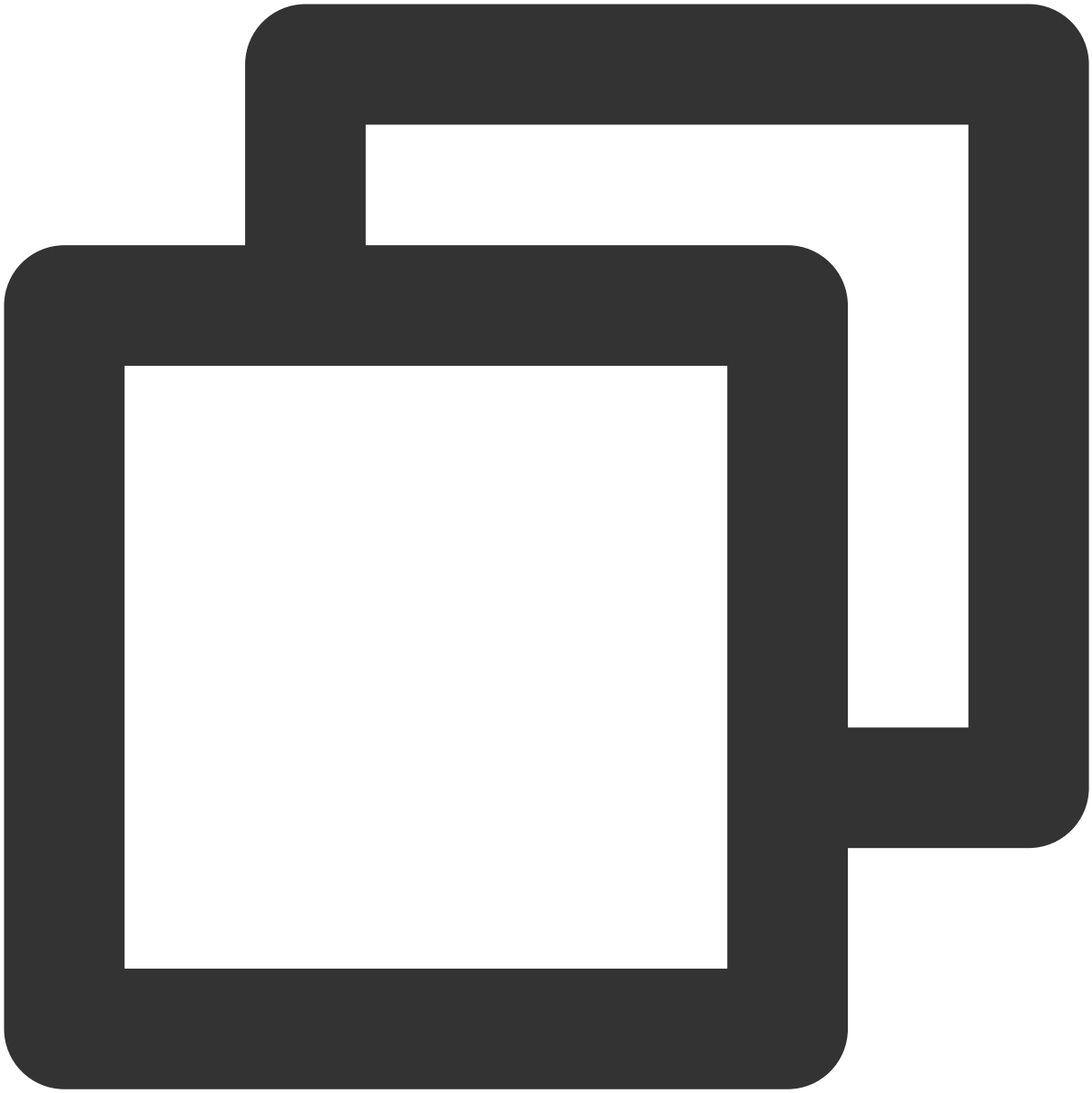
```
<import src="item.wxml"/>

<template is="item" data="{{text: 'forbar'}}"/>
```

需要注意的是 import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件中 import 的 template，简言之就是 import 不具有递归的特性。

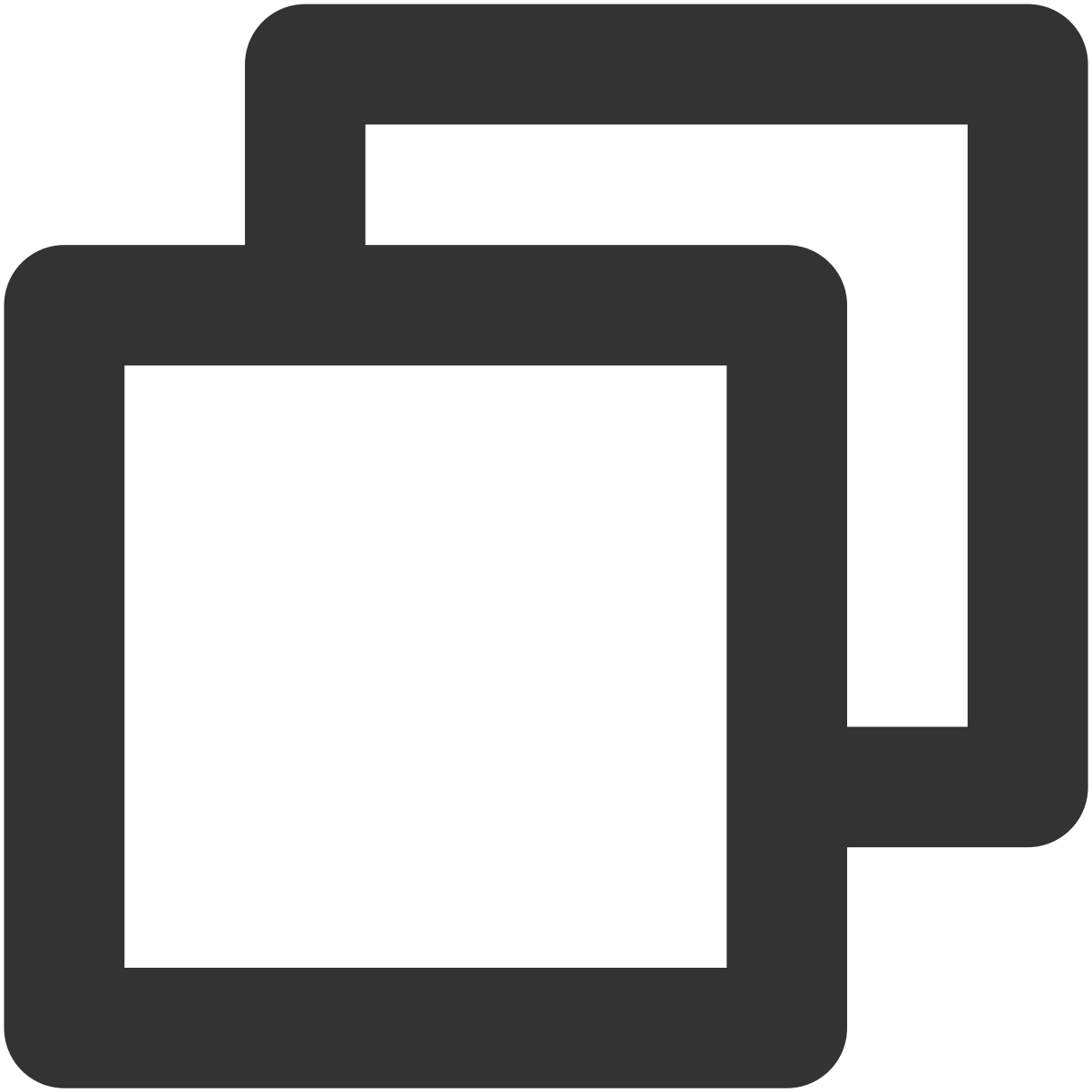
例如：C 引用 B，B 引用 A，在 C 中可以使用 B 定义的 template，在 B 中可以使用 A 定义的 template，但是 C 不能使用 A 定义的 template，如下代码所示：

版权所有：腾讯云计算（北京）有限责任公司

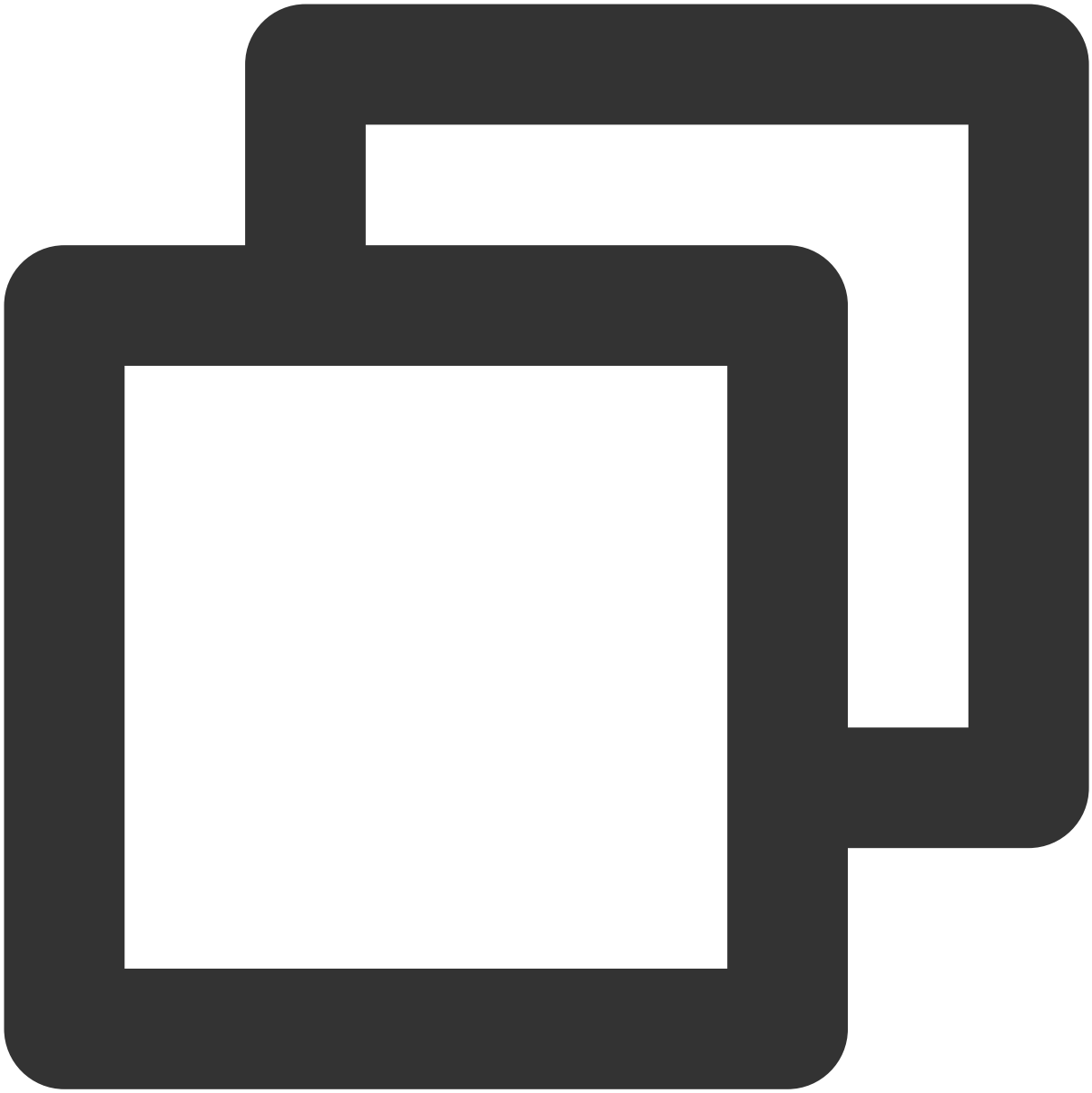


```
<!-- A.wxml -->
<template name="A">
  <text> A template </text>
</template>
```

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- B.wxml --><import src="a.wxml"/><template name="B">  <text> B template </text>
```

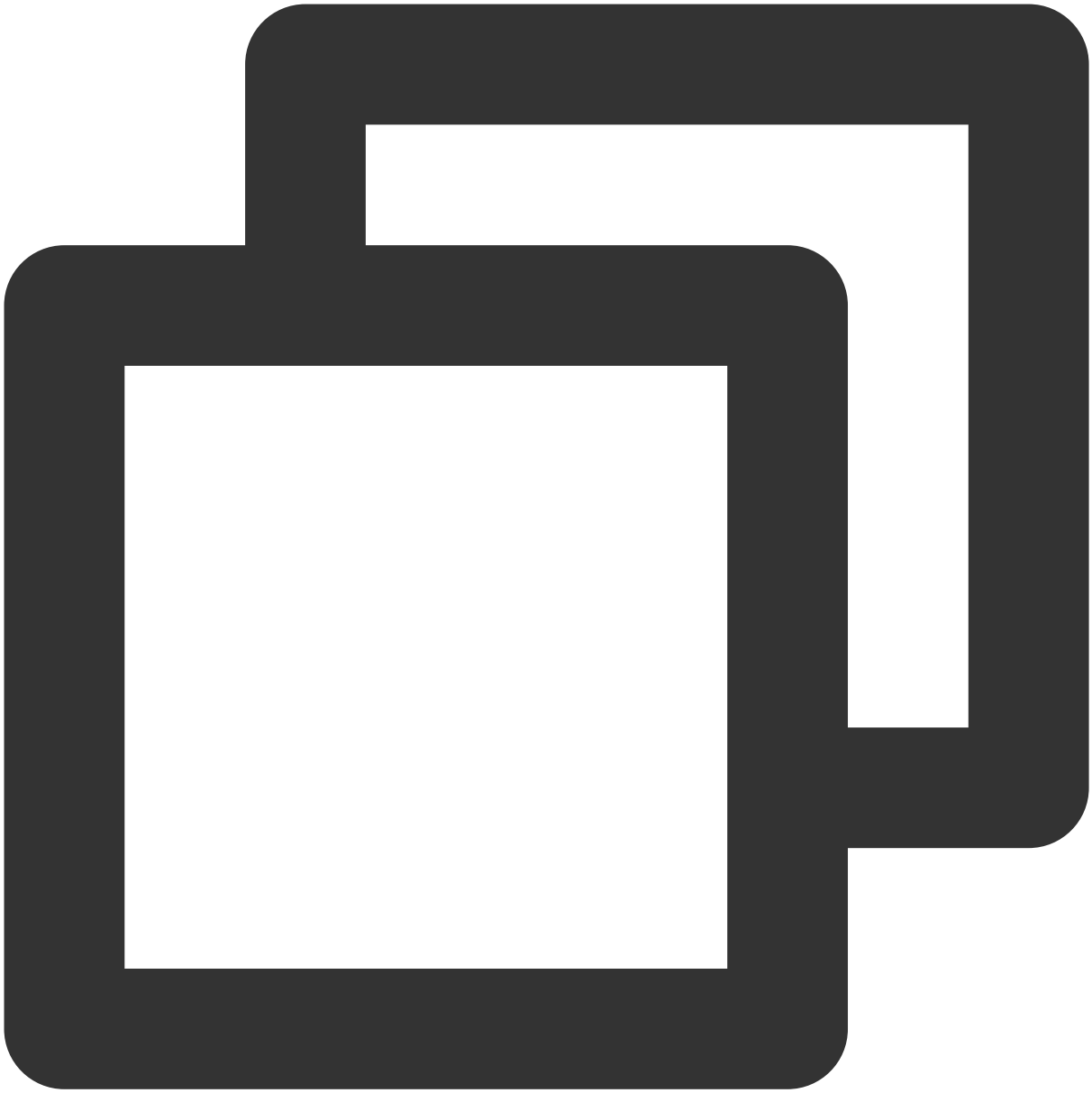


```
<!-- C.wxml -->
<import src="b.wxml"/>

<template is="A"/>  <!-- 这里将会触发一个警告，因为 b 中并没有定义模板 A -->

<template is="B"/>
```

include 可以将目标文件中除了 `<template/> <wxs/>` 外的整个代码引入，相当于是拷贝到 `include` 位置，如下代码所示：

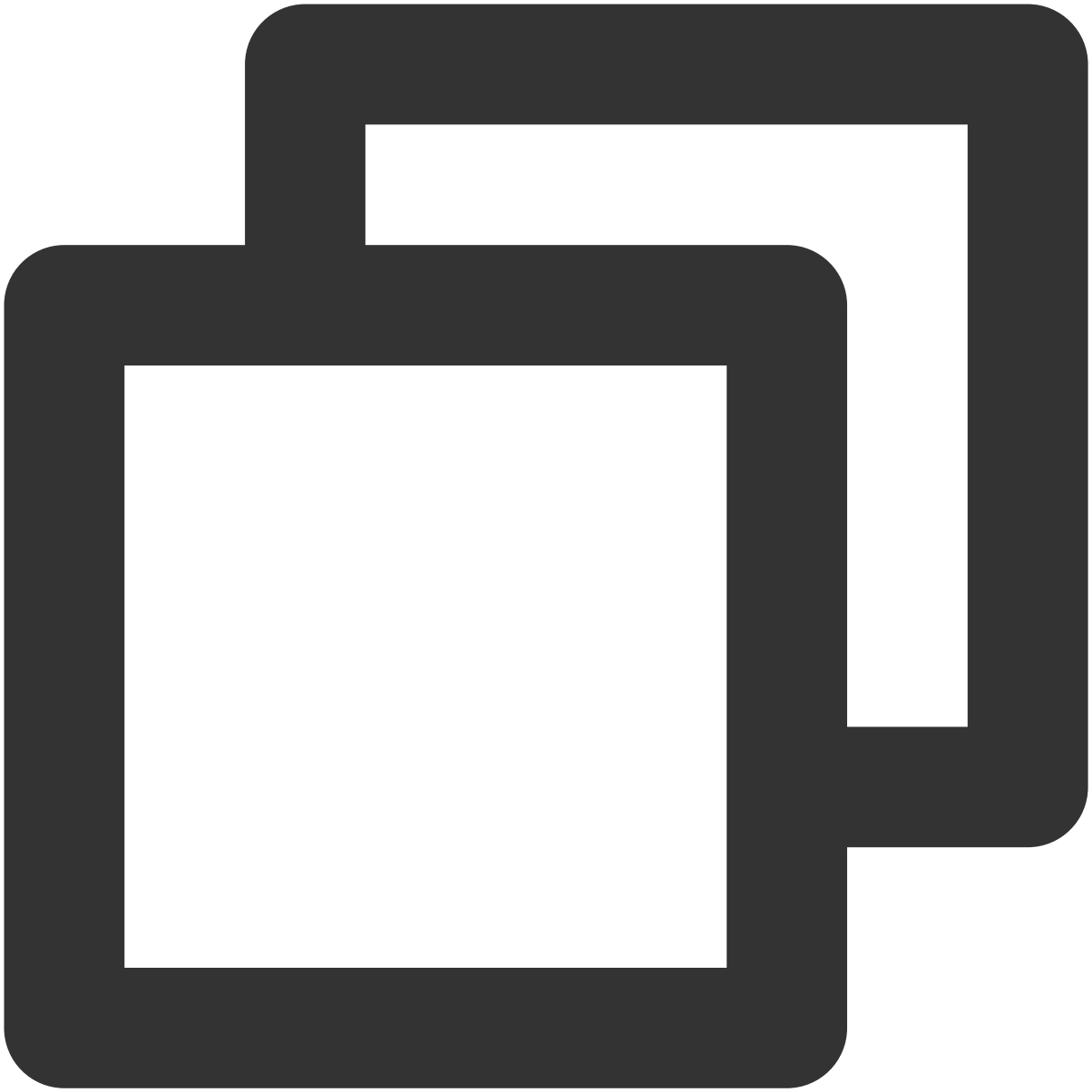


```
<!-- index.wxml -->
<include src="header.wxml"/>

<view> body </view>

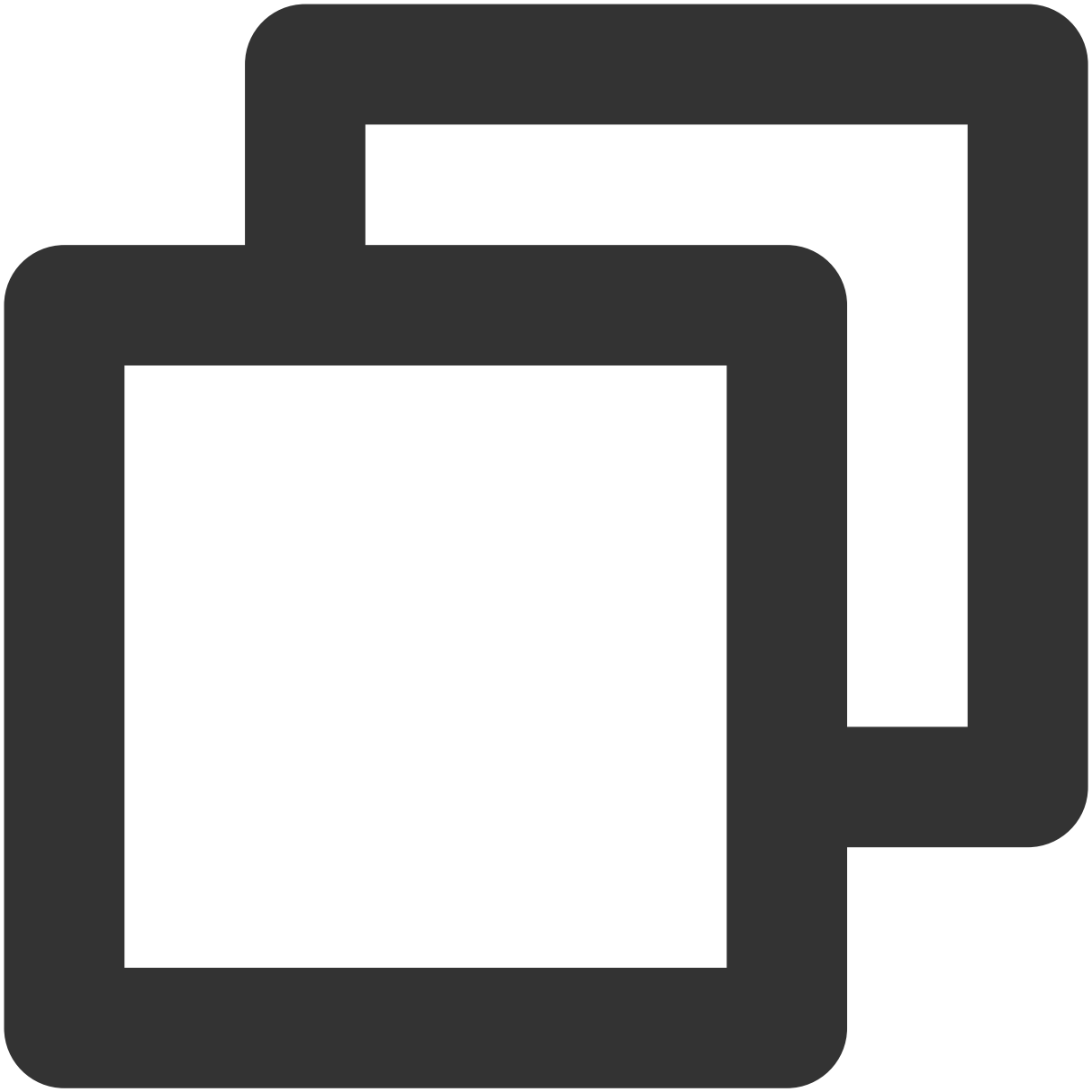
<include src="footer.wxml"/>
```

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- header.wxml -->
<view> header </view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- footer.wxml -->
<view> footer </view>
```

共同属性

所有 wxml 标签都支持的属性称之为共同属性，如下表所示。

属性名	类型	描述	注解

id	String	组件的唯一标识	整个页面唯一
class	String	组件的样式类	在对应的 WXSS 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data-*	Any	自定义属性	组件上触发的事件时，会发送给事件处理函数
bind*/catch*	EventHandler	组件的事件	-

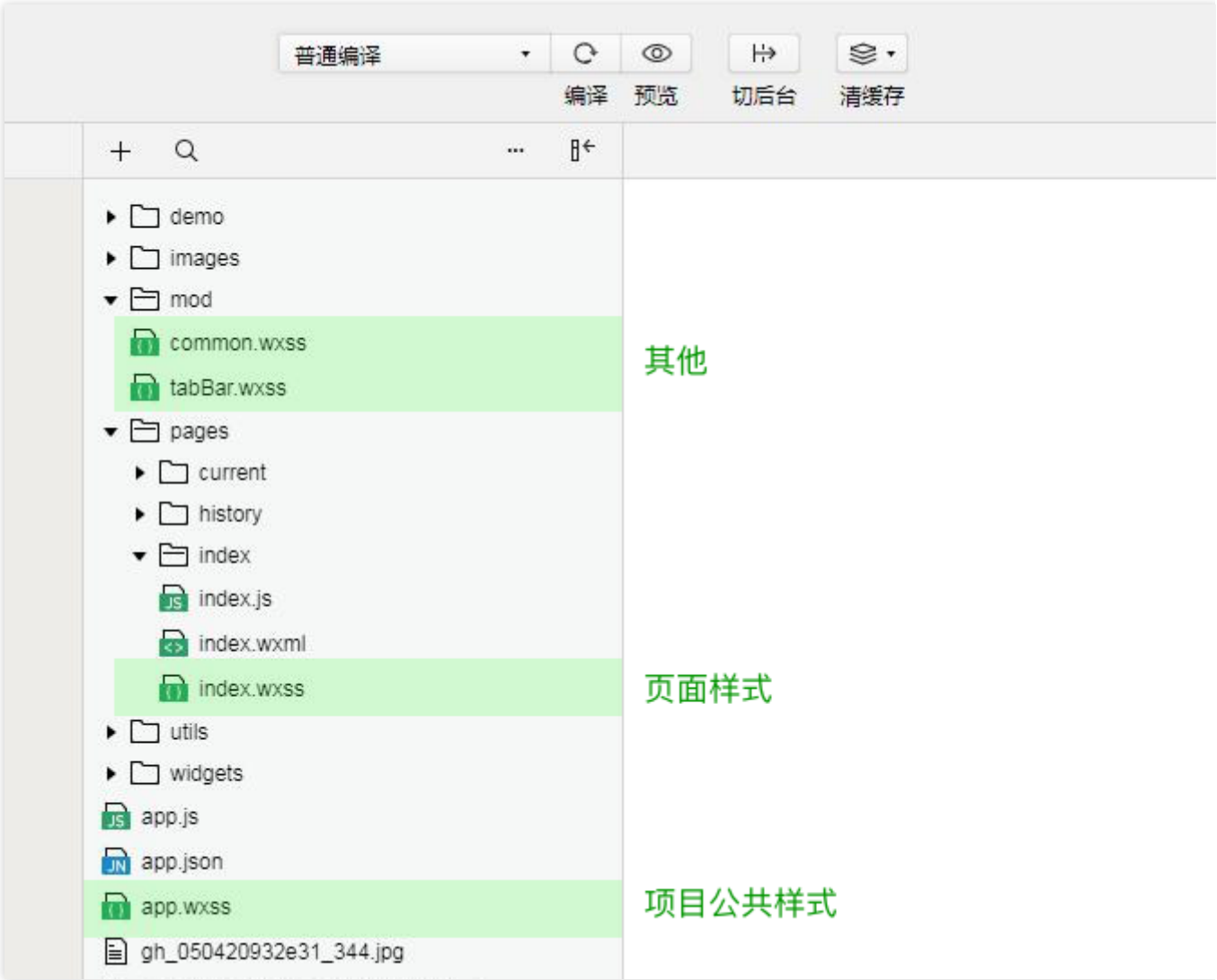
版权所有：腾讯云计算（北京）有限责任公司

WXSS 样式

最近更新时间：2024-05-10 11:24:15

WXSS（WeiXin Style Sheets）是一套用于小程序的样式语言，用于描述 WXML 的组件样式，也就是视觉上的效果。
WXSS 与 Web 开发中的 CSS 类似。为了更适合小程序开发，WXSS 对 CSS 做了一些补充以及修改。

文件组成



版权所有：腾讯云计算（北京）有限责任公司

app.json

```
{
  "pages":[
    "pages/rpx/index",
    "pages/box/index",
    "pages/inlinestyle/index",
    "pages/flex/index"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff
    "navigationBarTitleText": " ",
    "navigationBarTextStyle":"black"
  }
}
```

项目公共样式：根目录中的 app.wxss 为项目公共样式，它会被注入到小程序的每个页面。页面样式：与 app.json 注册过的页面同名且位置同级的 WXSS 文件。比如上图注册了pages/rpx/index 页面，那 pages/rpx/index.wxss 为页面 pages/rpx/index.wxml 的样式。

其它样式：其它样式可以被项目公共样式和页面样式引用，引用方法查看本章中的 WXSS 引用。

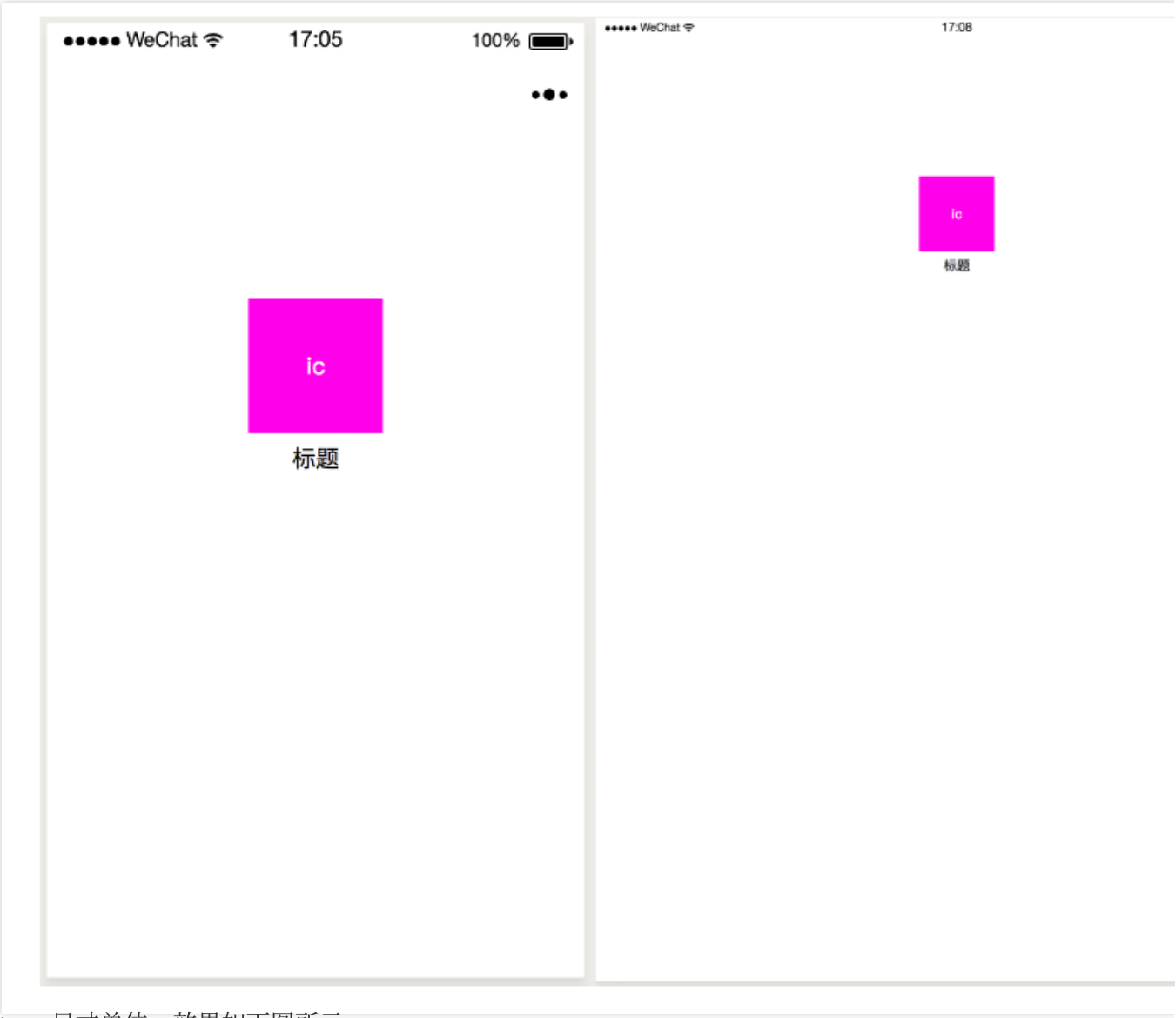
在小程序开发中，开发者不需要像 Web 开发那样去优化样式文件的请求数量，只需要考虑代码的组织即可。样式文件最终会被编译优化，具体的编译原理我们留在后面章节再做介绍。

尺寸单位

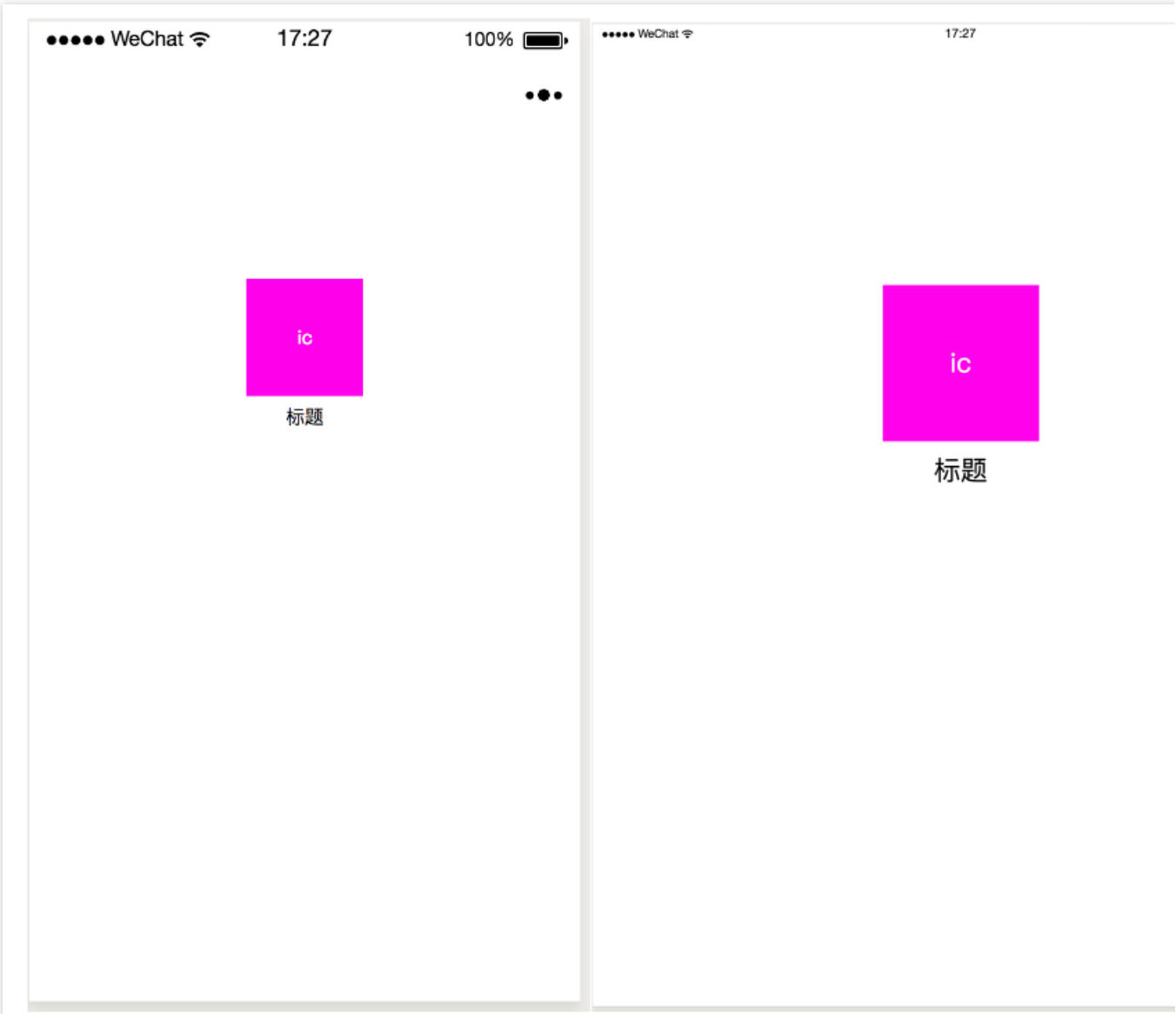
在 WXSS 中，引入了 rpx（responsive pixel）尺寸单位。引用新尺寸单位的目的是，适配不同宽度的屏幕，开发起来更简单。

如下图所示，同一个元素，在不同宽度的屏幕下，如果使用 px 为尺寸单位，有可能造成页面留白过多。

版权所有：腾讯云计算（北京）有限责任公司



修改为 rpx 尺寸单位，效果如下图所示。



小程序编译后，rpx 会做一次 px 换算。换算是以375个物理像素为基准，也就是在一个宽度为375物理像素的屏幕下，1rpx = 1px。

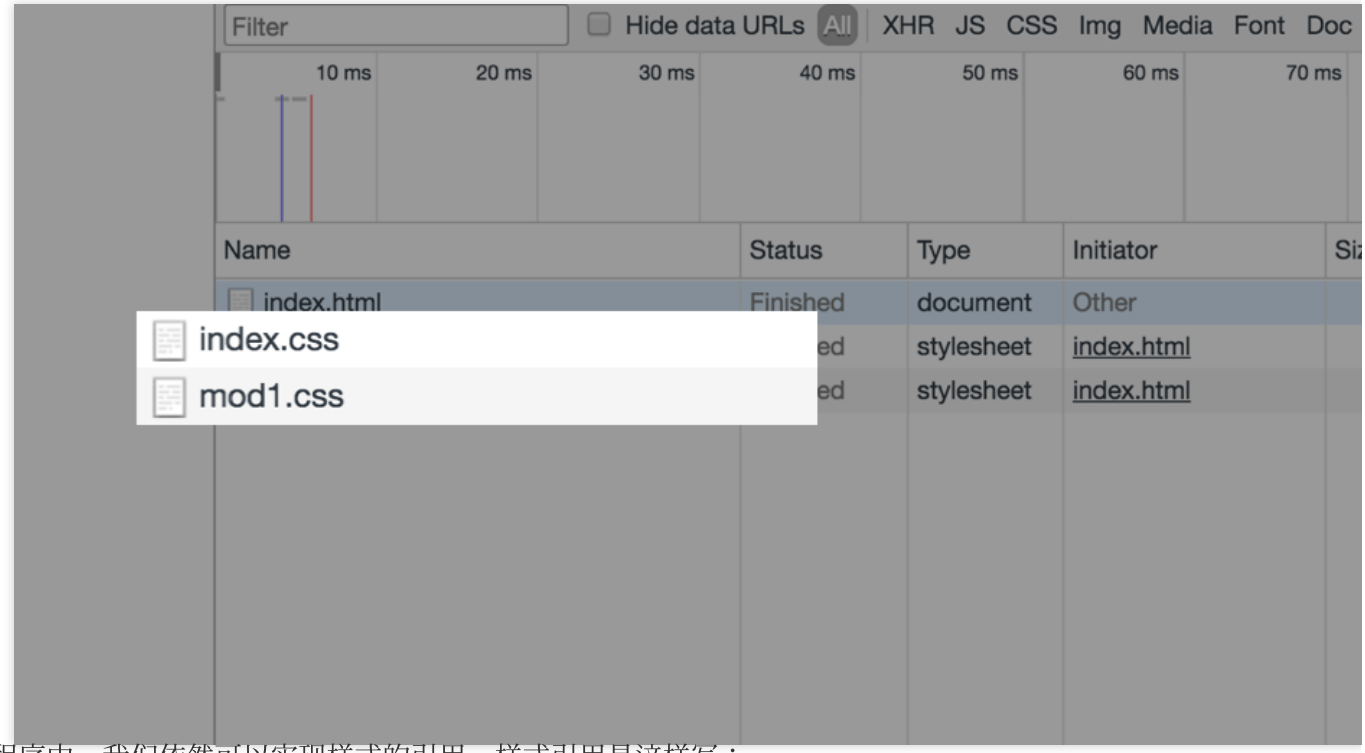
举个例子：iPhone6屏幕宽度为375px，共750个物理像素，那么1rpx = 375 / 750 px = 0.5px。

设备	rpx换算px (屏幕宽度/750)	px换算rpx (750/屏幕宽度)
iPhone5	1rpx = 0.42px	1px = 2.34rpx
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6 Plus	1rpx = 0.552px	1px = 1.81rpx

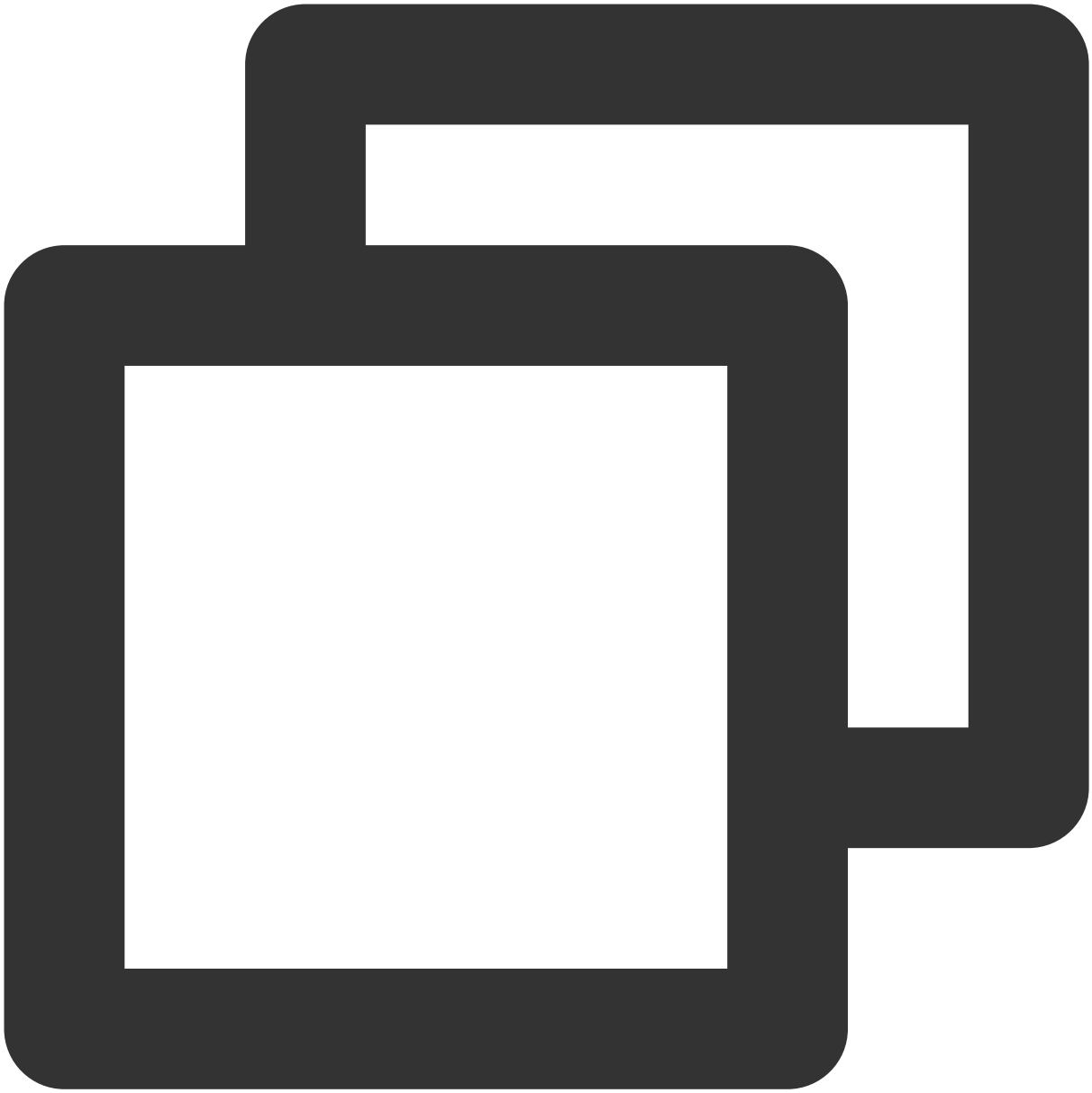
WXSS 引用

版权所有：腾讯云计算（北京）有限责任公司

在 CSS 中，开发者可以这样引用另一个样式文件：@import url('./test_0.css')
这种方法在请求上不会把 test_0.css 合并到 index.css 中，也就是请求 index.css 的时候，会多一个 test_0.css 的请求。



在小程序中，我们依然可以实现样式的引用，样式引用是这样写：

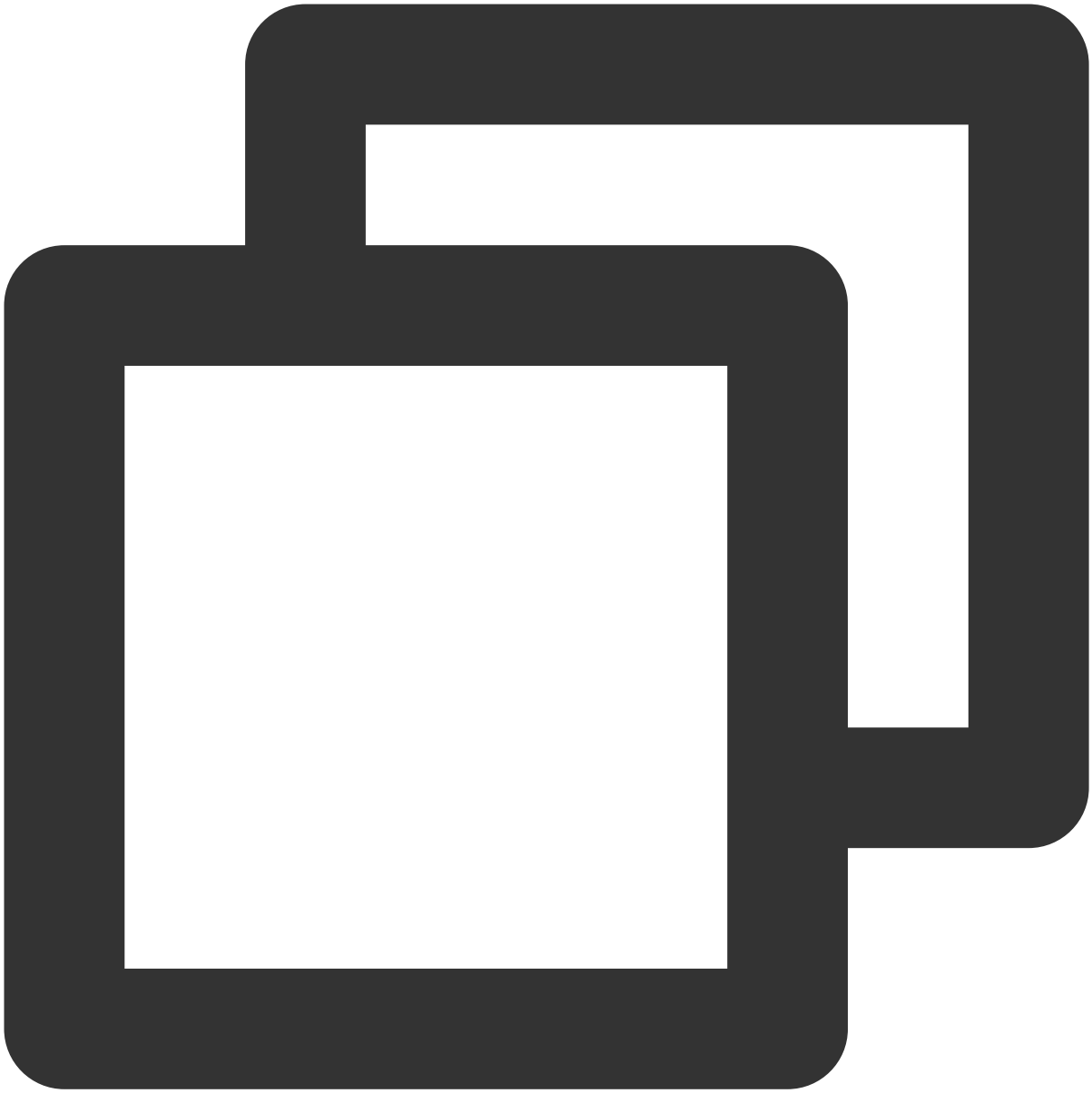


```
@import './test_0.wxss'
```

由于 WXSS 最终会被编译打包到目标文件中，用户只需要下载一次，在使用过程中不会因为样式的引用而产生多余的文件请求。

内联样式

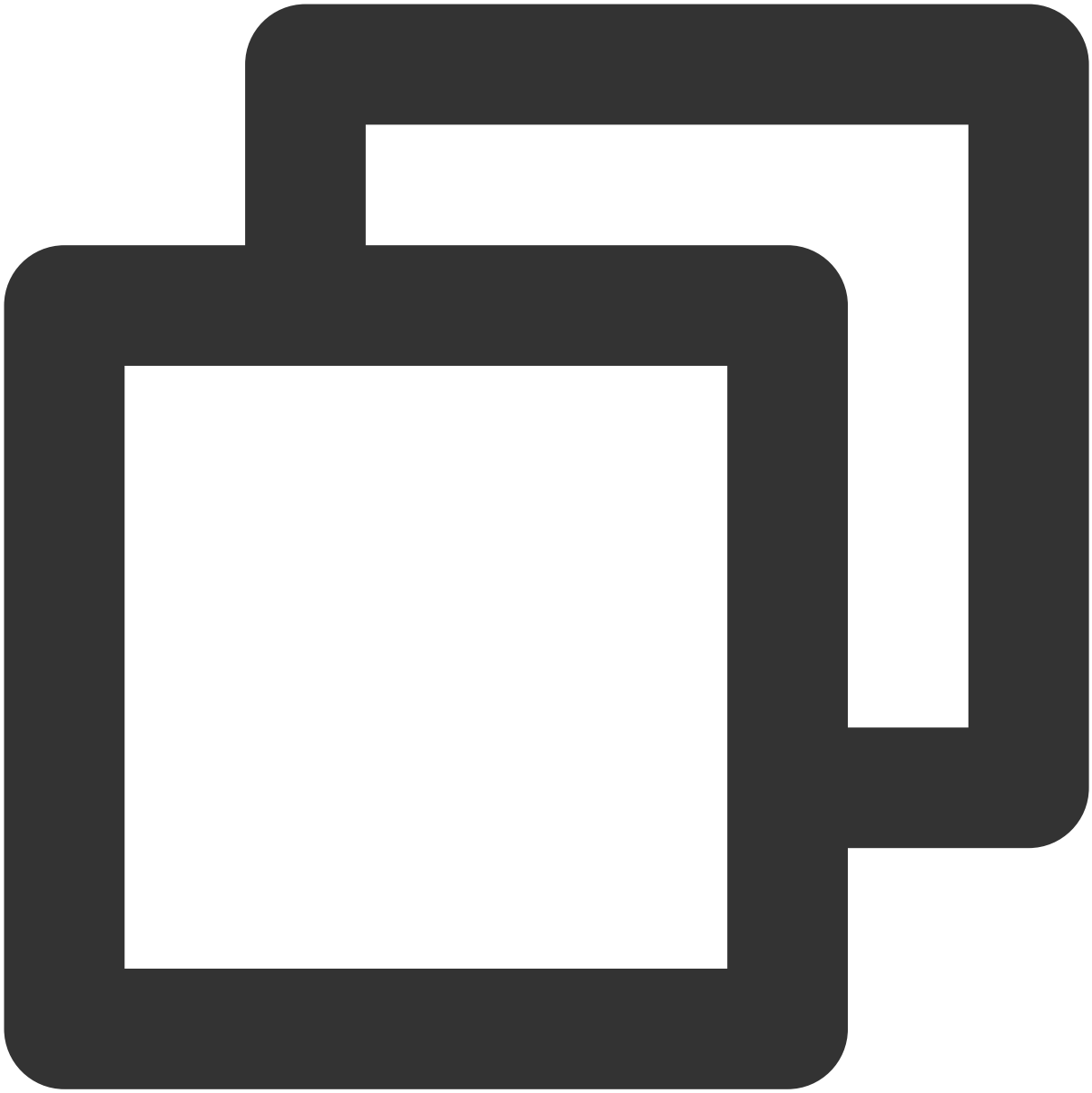
WXSS 内联样式与 Web 开发一致：



```
<!--index.wxml-->

<!--内联样式-->
<view style="color: red; font-size: 48rpx"></view>
```

小程序支持动态更新内联样式：



```
<!--index.wxml-->

<!--可动态变化的内联样式-->
<!--
{
  eleColor: 'red',
  eleFontsize: '48rpx'
}
-->
<view style="color: {{eleColor}}; font-size: {{eleFontsize}}"></view>
```

版权所有：腾讯云计算（北京）有限责任公司

选择器

目前支持的选择器如下表所示。

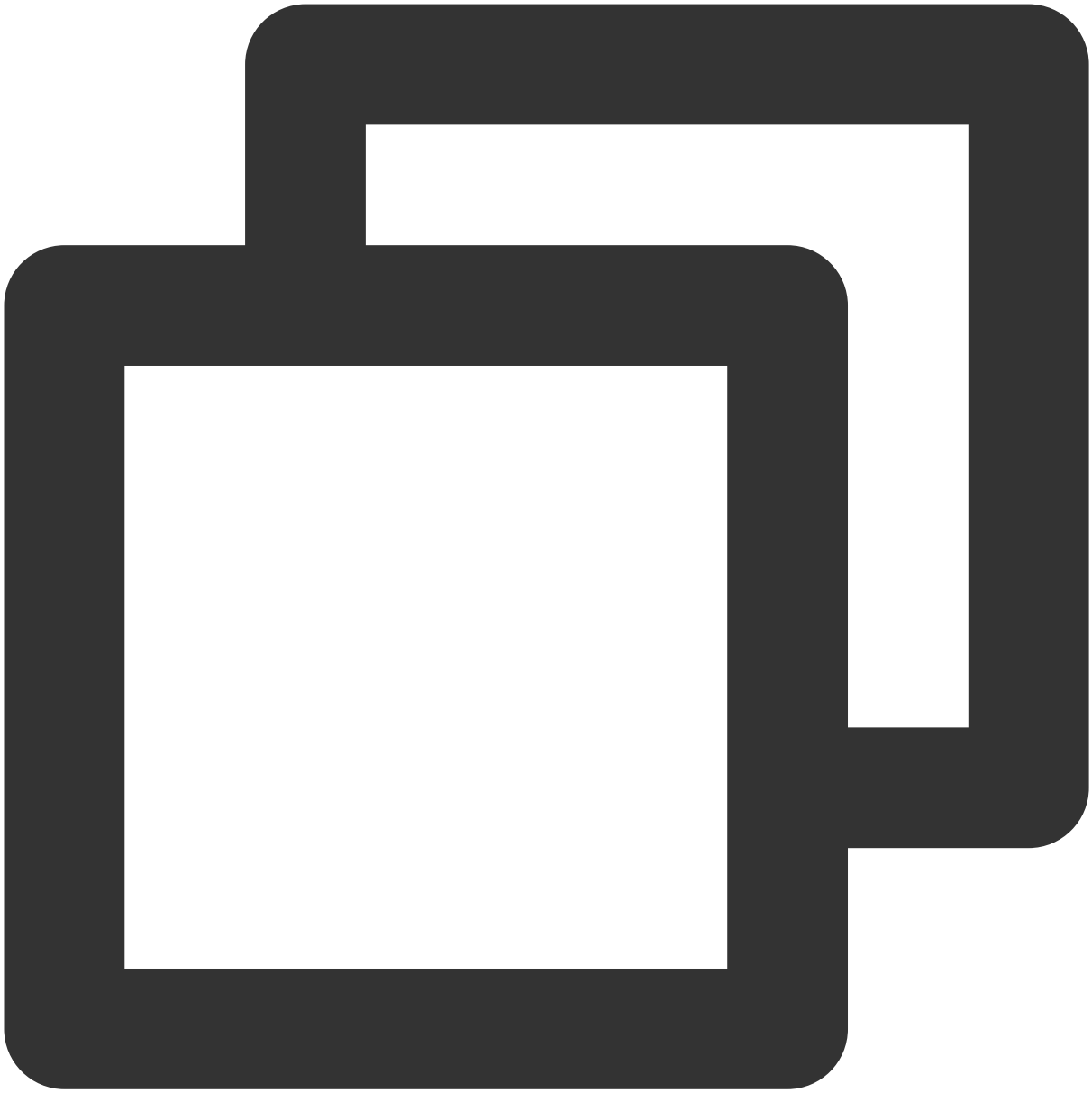
类型	选择器	样例	样例描述
类选择器	.class	.intro	选择所有拥有 class="intro" 的组件
id选择器	#id	#firstname	选择拥有 id="firstname" 的组件
元素选择器	element	view checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
伪元素选择器	::after	view::after	在 view 组件后边插入内容
伪元素选择器	::before	view::before	在 view 组件前边插入内容

WXSS 优先级与 CSS 类似，权重如下图所示。



权重越高越优先。在优先级相同的情况下，后设置的样式优先级高于先设置的样式。

WXSS 选择器优先级权重，代码如下所示：



```
view{ // 权重为 1
  color: blue
}

.ele{ // 权重为 10
  color: red
}

#ele{ // 权重为 100
  color: pink
}
```

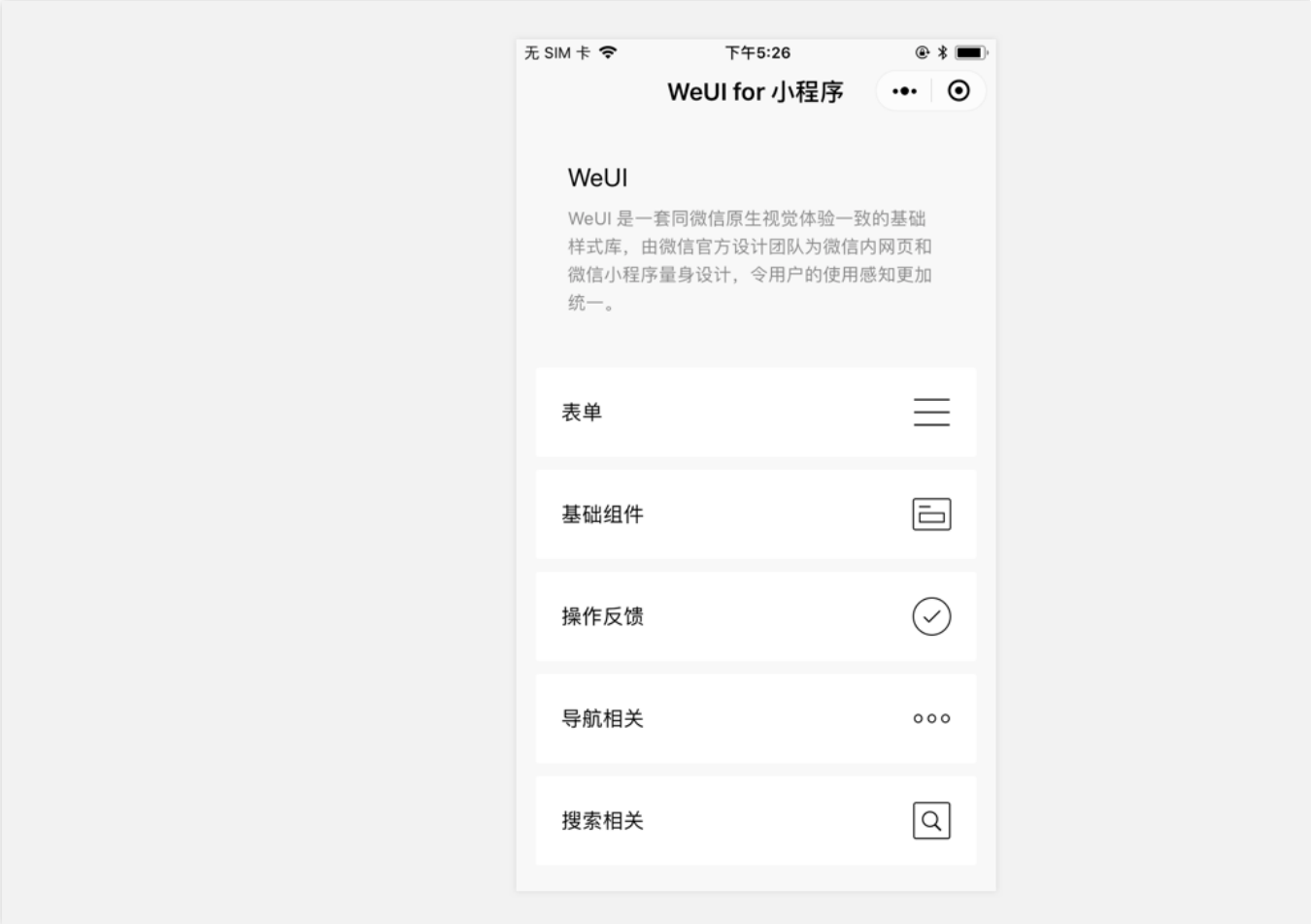
版权所有：腾讯云计算（北京）有限责任公司

```
view#ele{ // 权重为 1 + 100 = 101，优先级最高，元素颜色为orange
  color: orange
}

view.ele{ // 权重为 1 + 10 = 11
  color: green
}
```

官方样式库

为了减轻开发者样式开发的工作量，我们提供了 WeUI.wxss 基础样式库。
WeUI 是一套与微信原生视觉体验一致的基础样式库，由微信官方设计团队为微信内网页和微信小程序量身设计，令用户的使用感知更加统一。包含 button、cell、dialog、progress、toast、article、actionsheet、icon 等各式原生。



JavaScript 脚本

最近更新时间：2024-05-10 11:26:03

小程序的主要开发语言是 JavaScript，开发者使用 JavaScript 来开发业务逻辑以及调用小程序的 API 来完成业务需求。

ECMAScript

在大部分开发者看来，ECMAScript 和 JavaScript 表达的是同一种含义，但是严格的说，两者的意义是不同的。ECMAScript 是一种由 Ecma 国际通过 ECMA-262标准化的脚本程序设计语言， JavaScript 是 ECMAScript 的一种实现。理解 JavaScript 是 ECMAScript 一种实现后，可以帮助开发者理解小程序中的 JavaScript 同浏览器中的 JavaScript 以及 NodeJS 中的 JavaScript 是不相同的。
ECMA-262 规定了 ECMAScript 语言的几个重要组成部分：
1. 语法
2. 类型
3. 语句
4. 关键字
5. 操作符
6. 对象
浏览器中JavaScript 构成如下图所示：



浏览器中的 JavaScript 是由 ECMAScript 和 BOM（浏览器对象模型）以及 DOM（文档对象模型）组成的，Web 前端开发者会很熟悉这两个对象模型，它使得开发者可以去操作浏览器的一些表现，比如修改 URL、修改页面呈现、记录数据等等。NodeJS 中 JavaScript 构成如下图所示：

版权所有：腾讯云计算（北京）有限责任公司



NodeJS 中的 JavaScript 是由 ECMAScript 和 NPM 以及 Native 模块组成，NodeJS 的开发者会非常熟悉 NPM 的包管理系统，通过各种拓展包来快速的实现一些功能，同时通过使用一些原生的模块例如 FS、HTTP、OS等等来拥有一些语言本身所不具有的能力。

那么，同开发者所熟悉的这两个环境是不同的，小程序中 JavaScript 构成如下图所示。



小程序中的 JavaScript 是由ECMAScript 以及小程序框架和小程序 API 来实现的。同浏览器中的 JavaScript 相比没有 BOM 以及 DOM 对象，所以类似 JQuery、Zepto 这种浏览器类库是无法在小程序中运行起来的，同样的缺少 Native 模块和 NPM 包管理的机制，小程序中无法加载原生库，也无法直接使用大部分的 NPM 包。

小程序的执行环境

明白了小程序中的 JavaScript 同浏览器以及 NodeJS 有所不同后，开发者还需要注意到另外一个问题，不同的平台的小程序的脚本执行环境也是有所区别的。

小程序目前可以运行在三大平台：

iOS 平台，包括 iOS9、iOS10、iOS11。

Android 平台

小程序 IDE

这种区别主要是体现三大平台实现的 ECMAScript 的标准有所不同。截止到当前一共有七个版本的 ECMAScript 标准，目前开发者大部分使用的是 ECMAScript 5 和 ECMAScript 6 的标准，但是在小程序中， iOS9 和 iOS10 所使用

版权所有：腾讯云计算（北京）有限责任公司

的运行环境并没有完全的兼容到 ECMAScript 6 标准，一些 ECMAScript 6 中规定的语法和关键字是没有的或者同标准是有所不同的，例如：

箭头函数

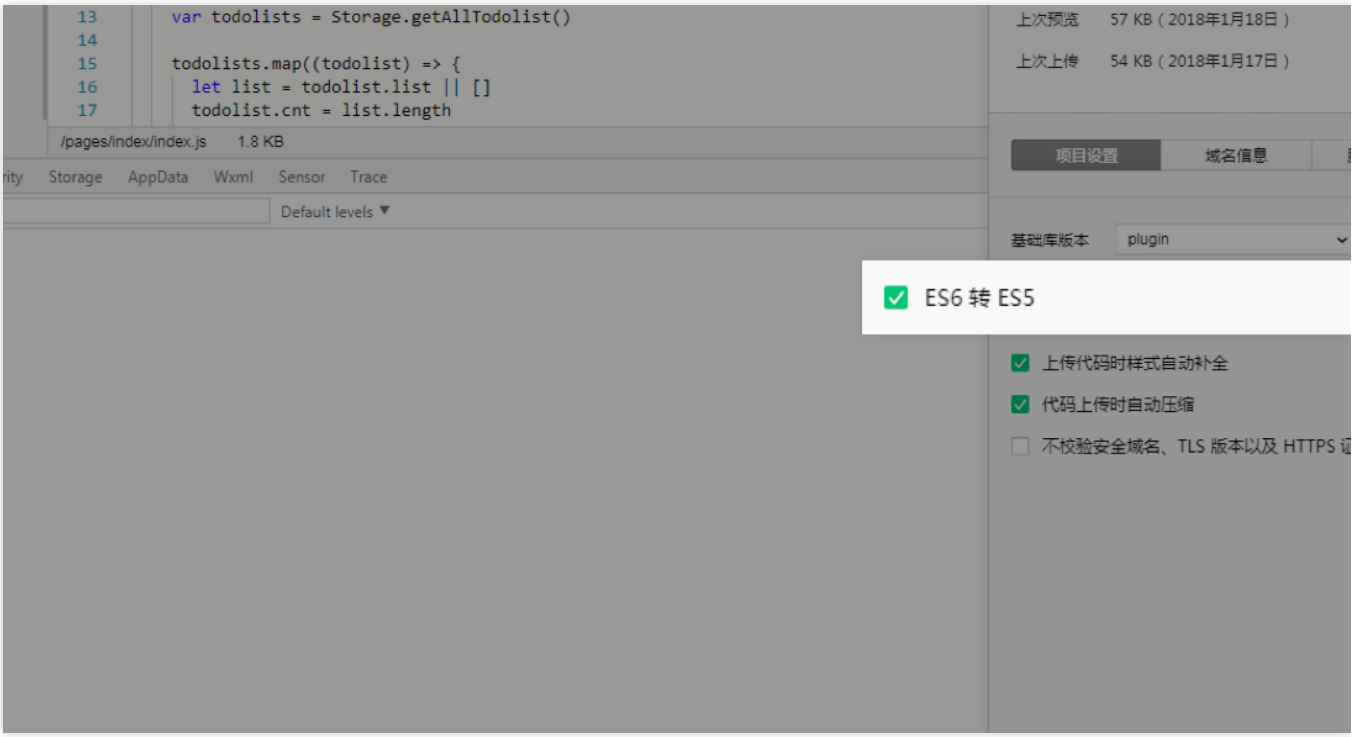
let const

模板字符串

...

所以一些开发者会发现有些代码在旧的手机操作系统上出现一些语法错误。为了帮助开发者解决这类问题，小程序 IDE 提供语法转码工具帮助开发者，将 ECMAScript 6代码转为 ECMAScript 5代码，从而在所有的环境都能得到很好的执行。

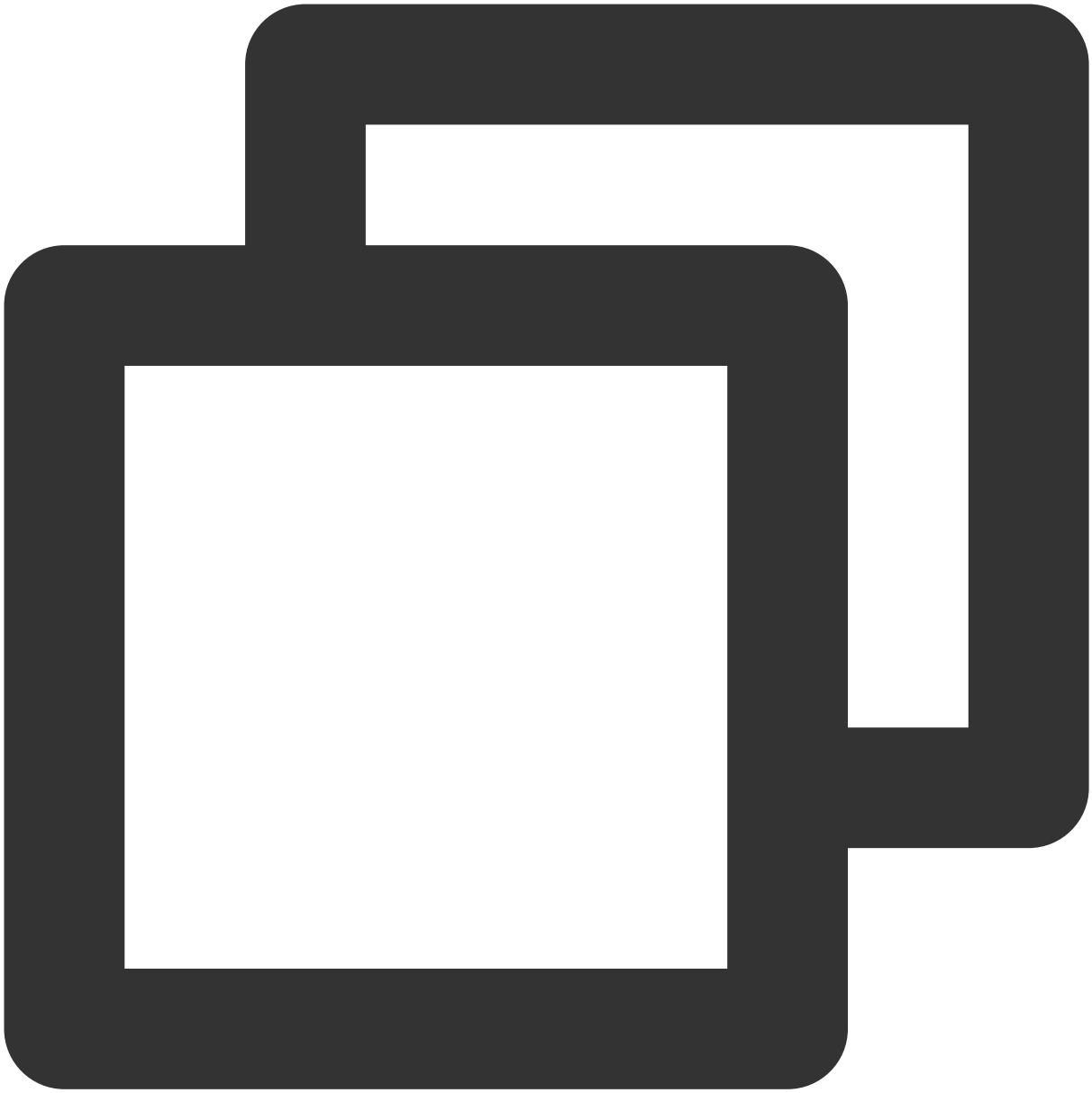
开发者需要在项目设置中，勾选 ES6 转 ES5 开启此功能。



模块化

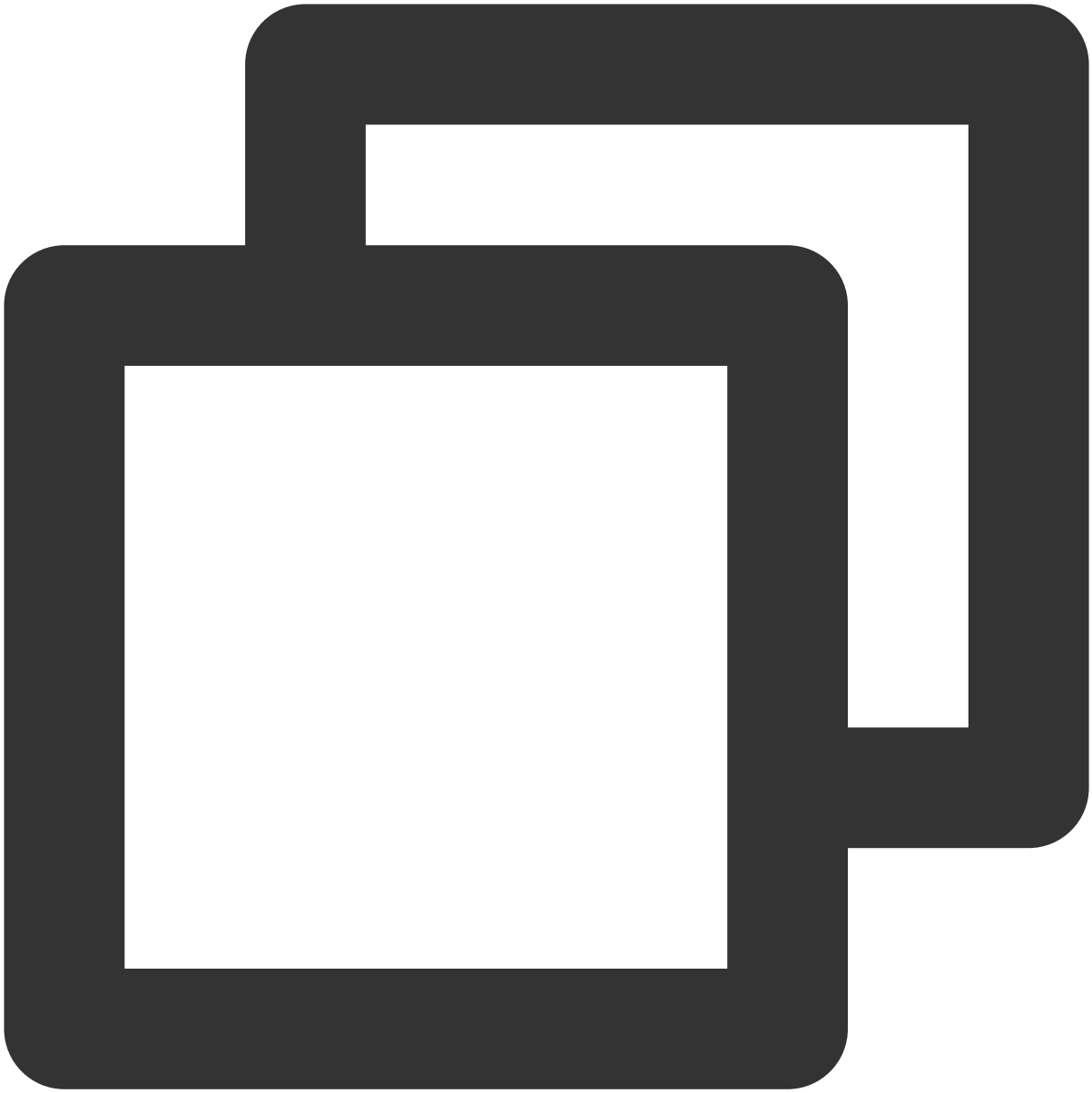
浏览器中，所有 JavaScript 是在运行在同一个作用域下的，定义的参数或者方法可以被后续加载的脚本访问或者改写。同浏览器不同，小程序中可以将任何一个 JavaScript 文件作为一个模块，通过 module.exports 或者 exports 对外暴露接口。

请看是一个简单模块示例，B.js 引用模块A，并使用A暴露的 multiplyBy2 方法完成一个变量乘以2的操作，如下代码所示：



```
// moduleA.js
module.exports = function( value ){
  return value * 2;
}
```

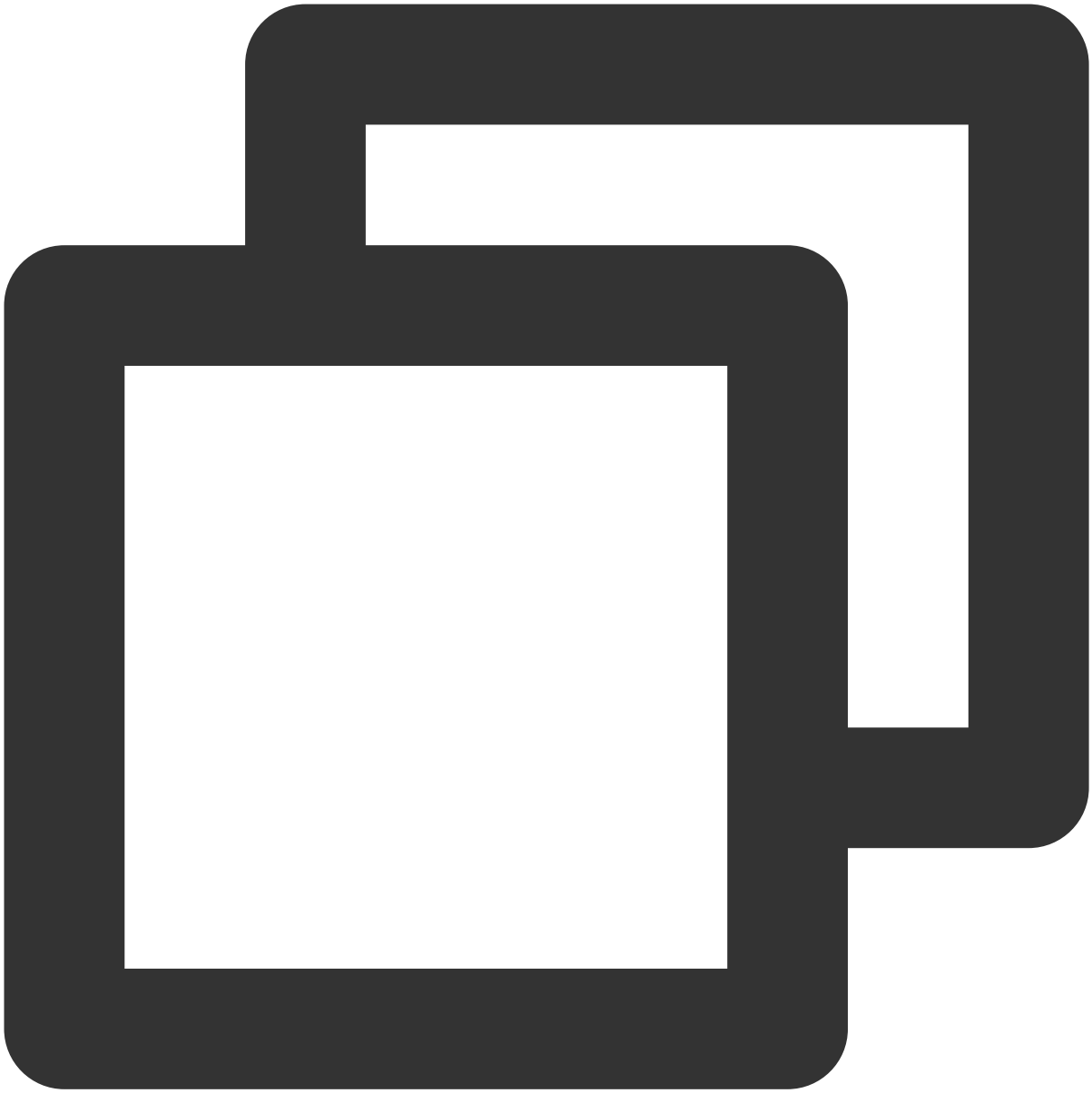
版权所有：腾讯云计算（北京）有限责任公司



```
// B.js

// 在B.js中引用模块A
var multiplyBy2 = require('./moduleA')
var result = multiplyBy2(4)
```

下方代码在需要使用这些模块的文件中，使用 require(path) 将公共代码引入：

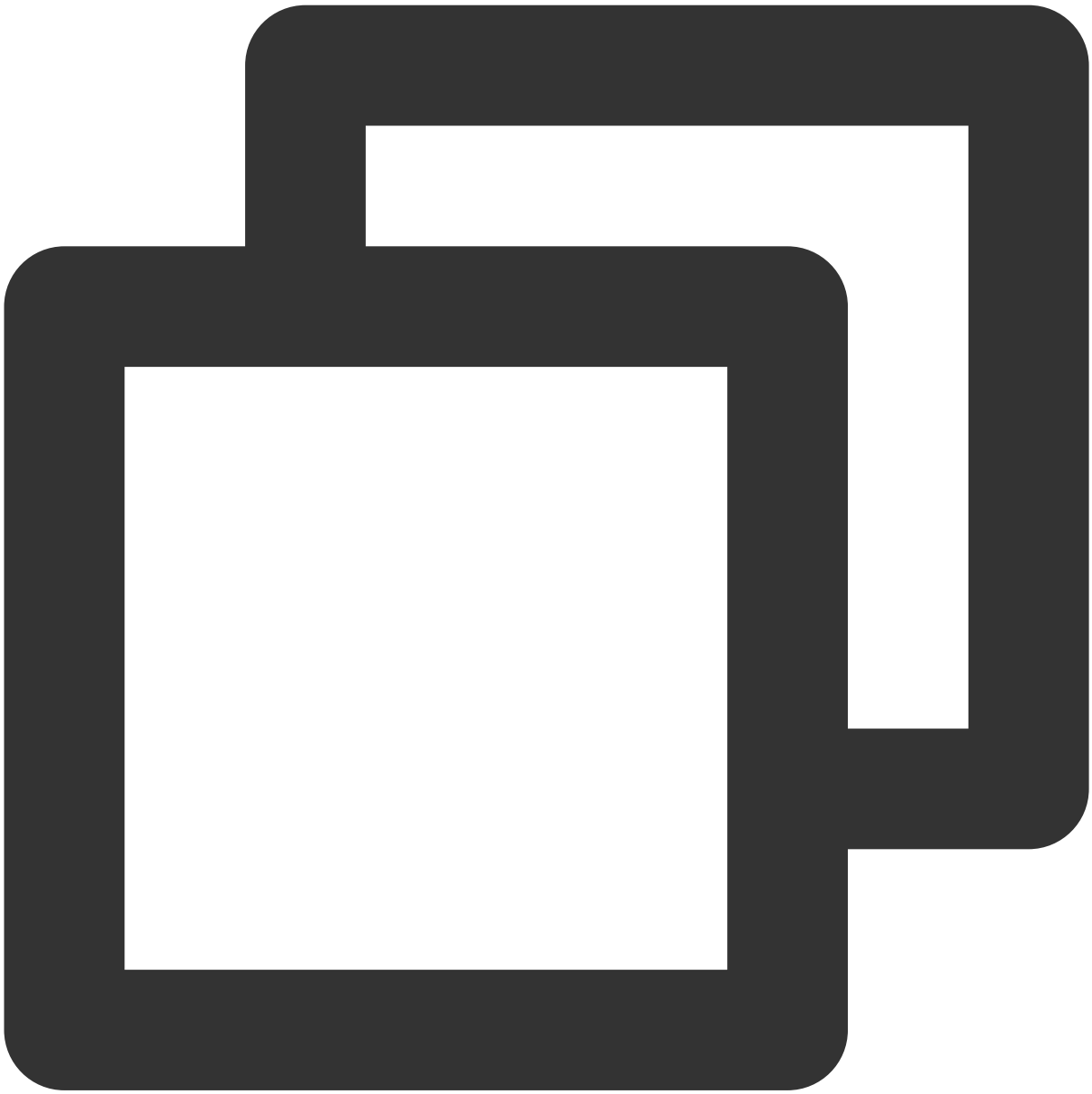


```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

脚本的执行顺序

浏览器中，脚本严格按照加载的顺序执行，如下代码所示：

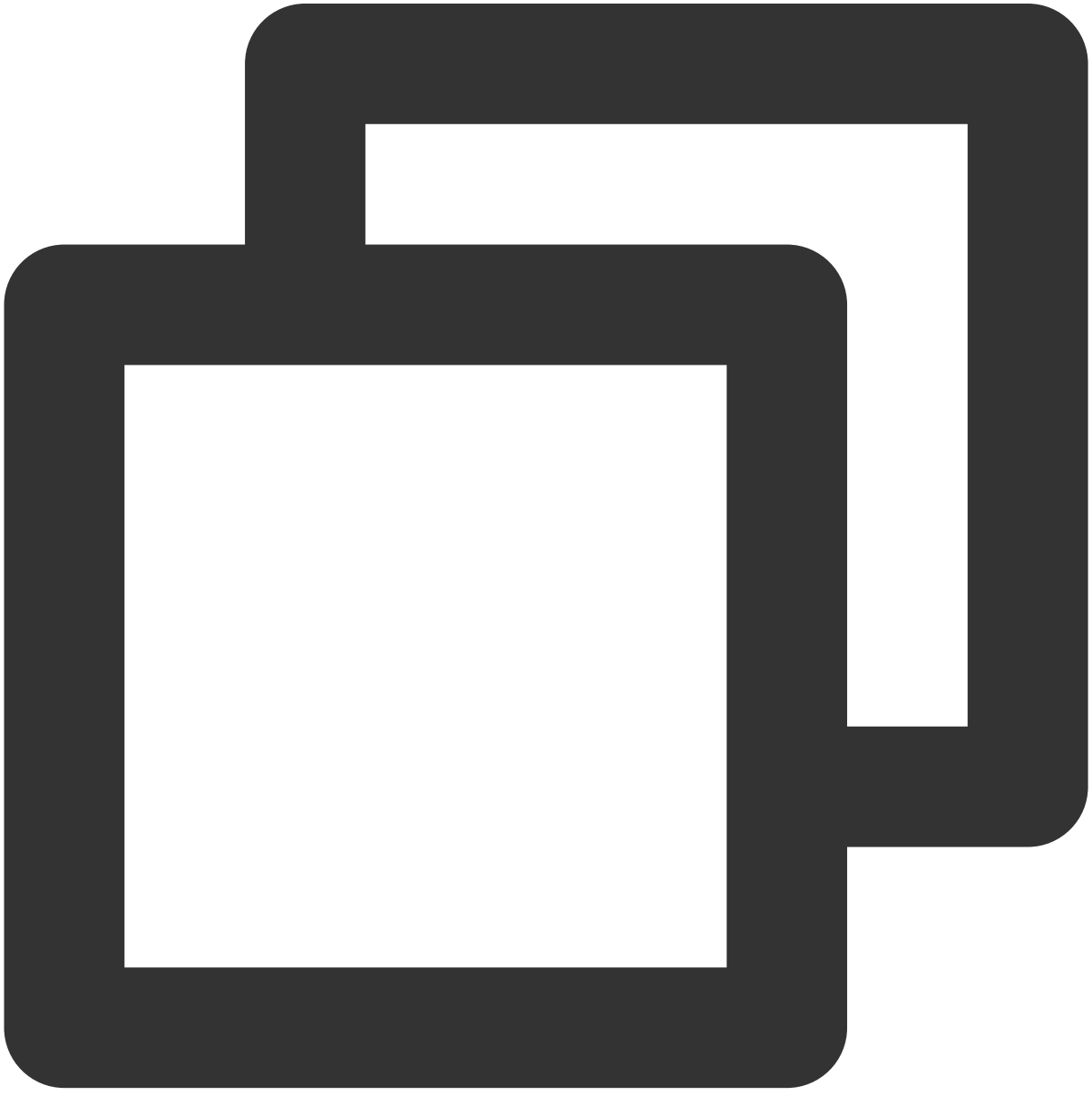


```
<html>
<head>
  <!-- a.js
  console.log('a.js')
   -->
  <script src ="a.js"></script>
  <script>
```

```
    console.log('inline script')
  </script>

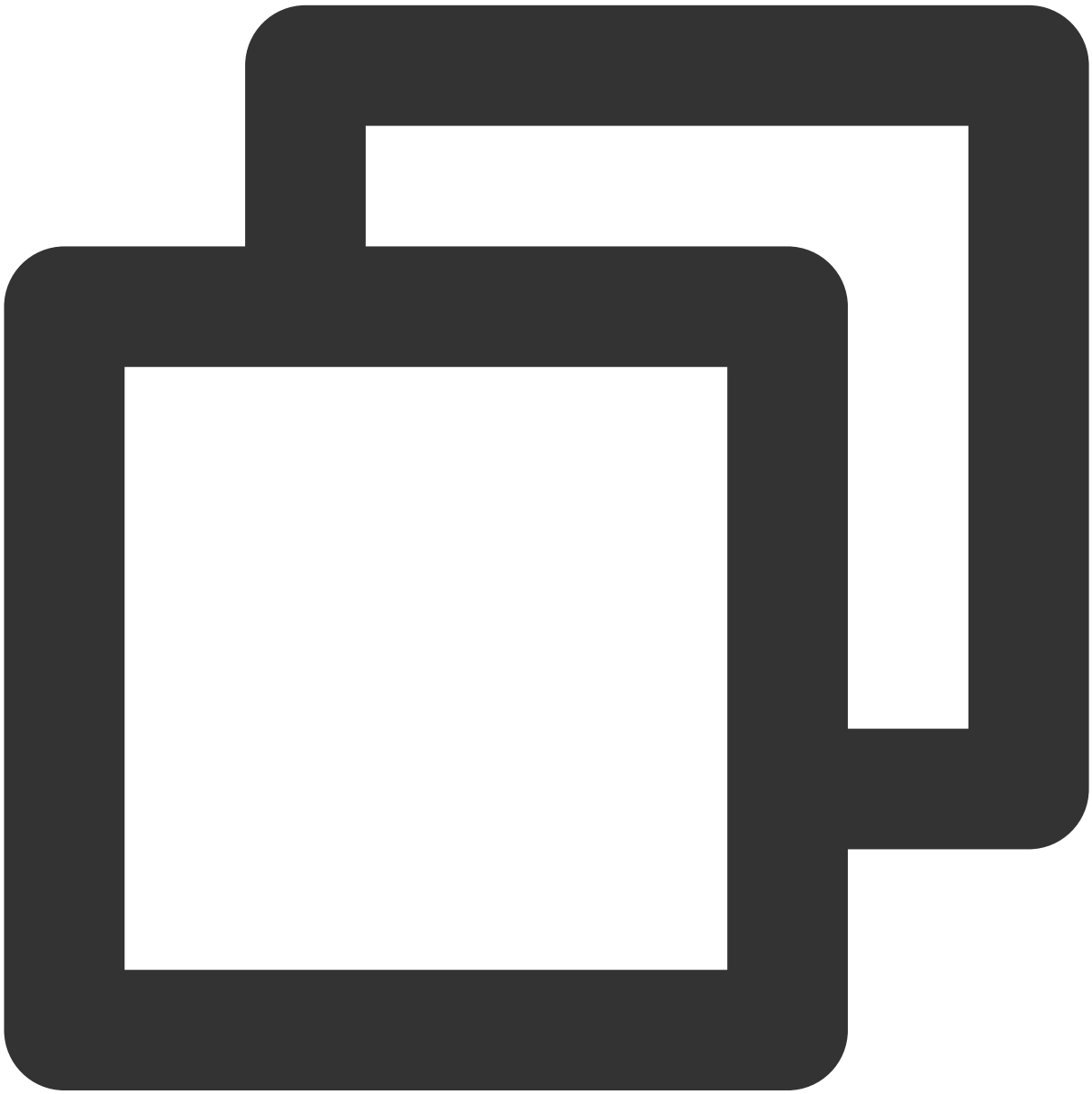
  <!-- b.js
  console.log('b.js')
   -->
  <script src ="b.js"></script>
</head>
</html>
```

以上代码的输出是：



```
a.js
inline script
b.js
```

而在小程序中的脚本执行顺序有所不同。小程序的执行的入口文件是 app.js 。并且会根据其中 require 的模块顺序决定文件的运行顺序，下方代码是一个 app.js 示例。

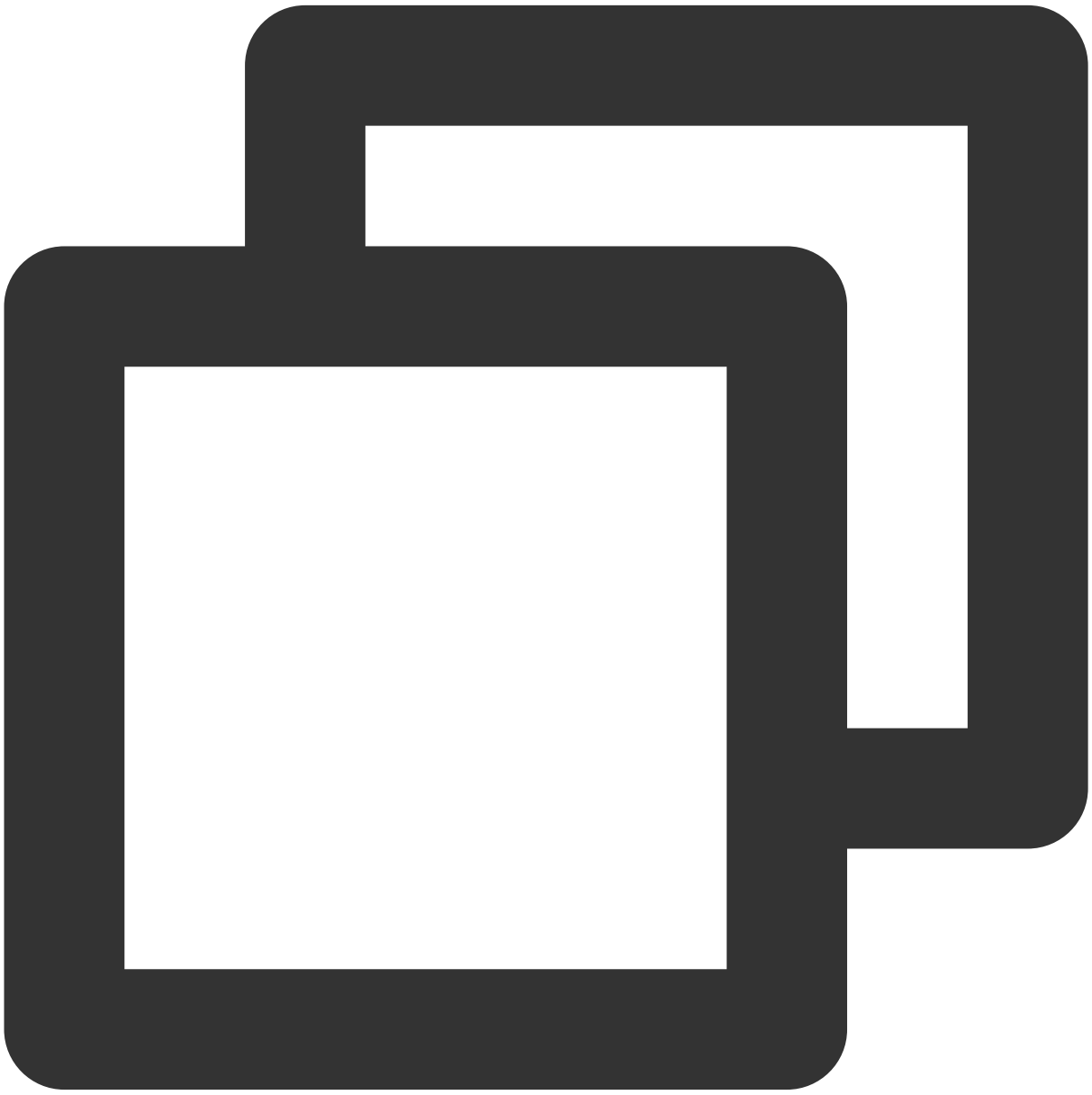


```
/* a.js
console.log('a.js')
*/
var a = require('./a.js')
```

```
console.log('app.js')

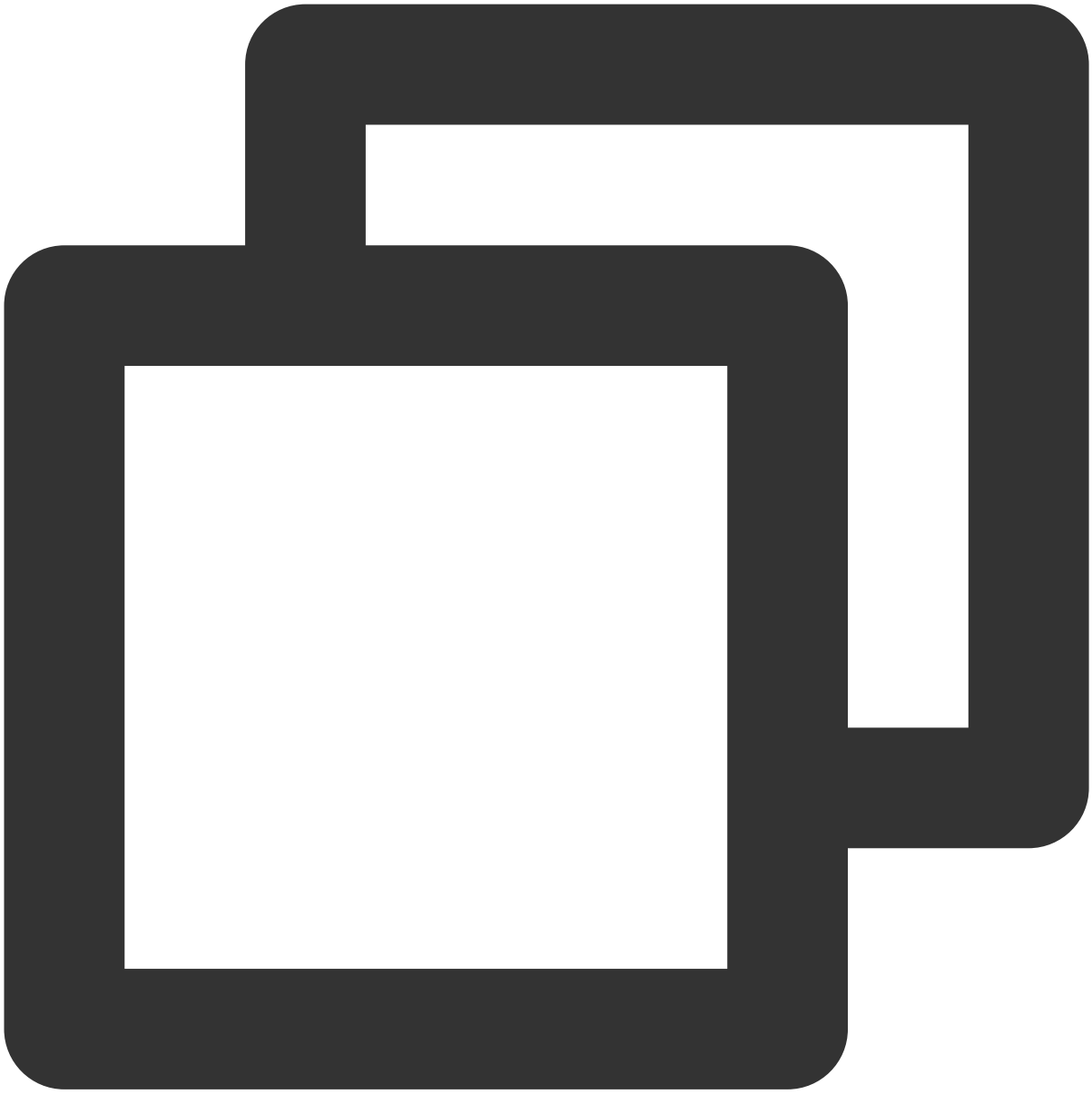
/* b.js
console.log('b.js')
*/
var b = require('./b.js')
```

以上代码的输出顺序是：



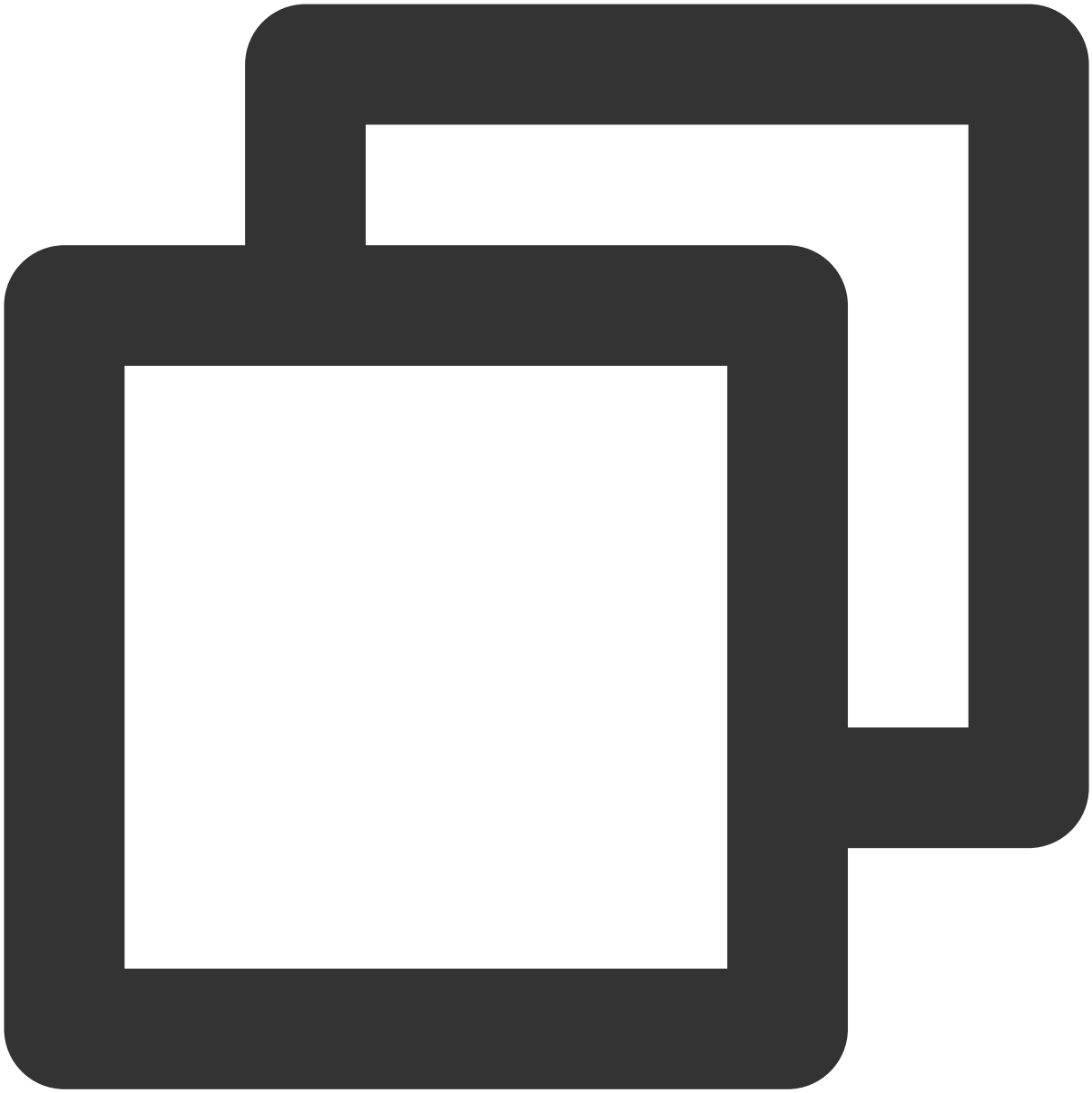
```
a.js
app.js
b.js
```

当 app.js 执行结束后，小程序会按照开发者在 app.json 中定义的 pages 的顺序，逐一执行，如下代码所示：



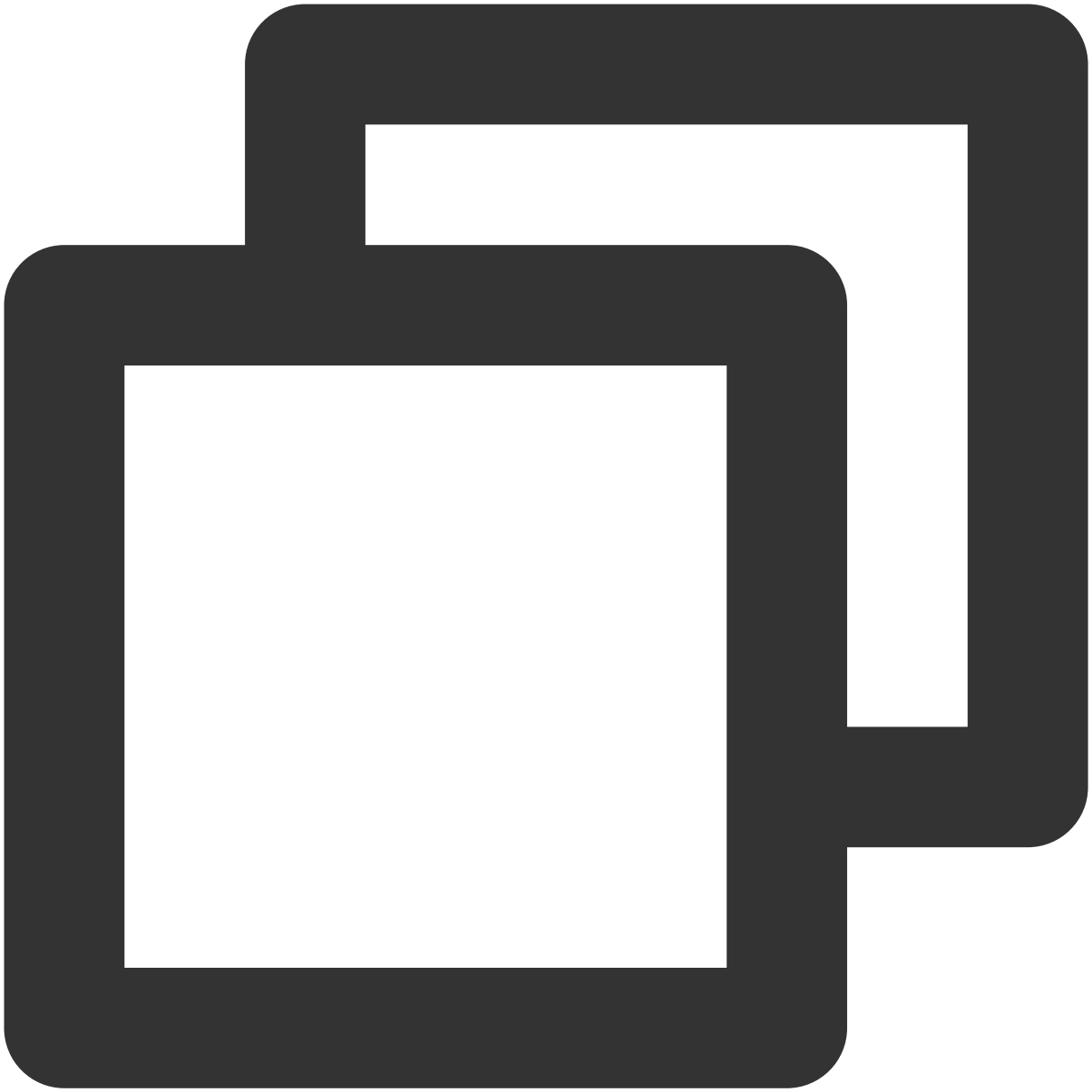
```
{
  "pages": [
    "pages/index/index",
    "pages/log/log",
    "pages/result/result"
  ],
  "window": {}
}
```

版权所有：腾讯云计算（北京）有限责任公司



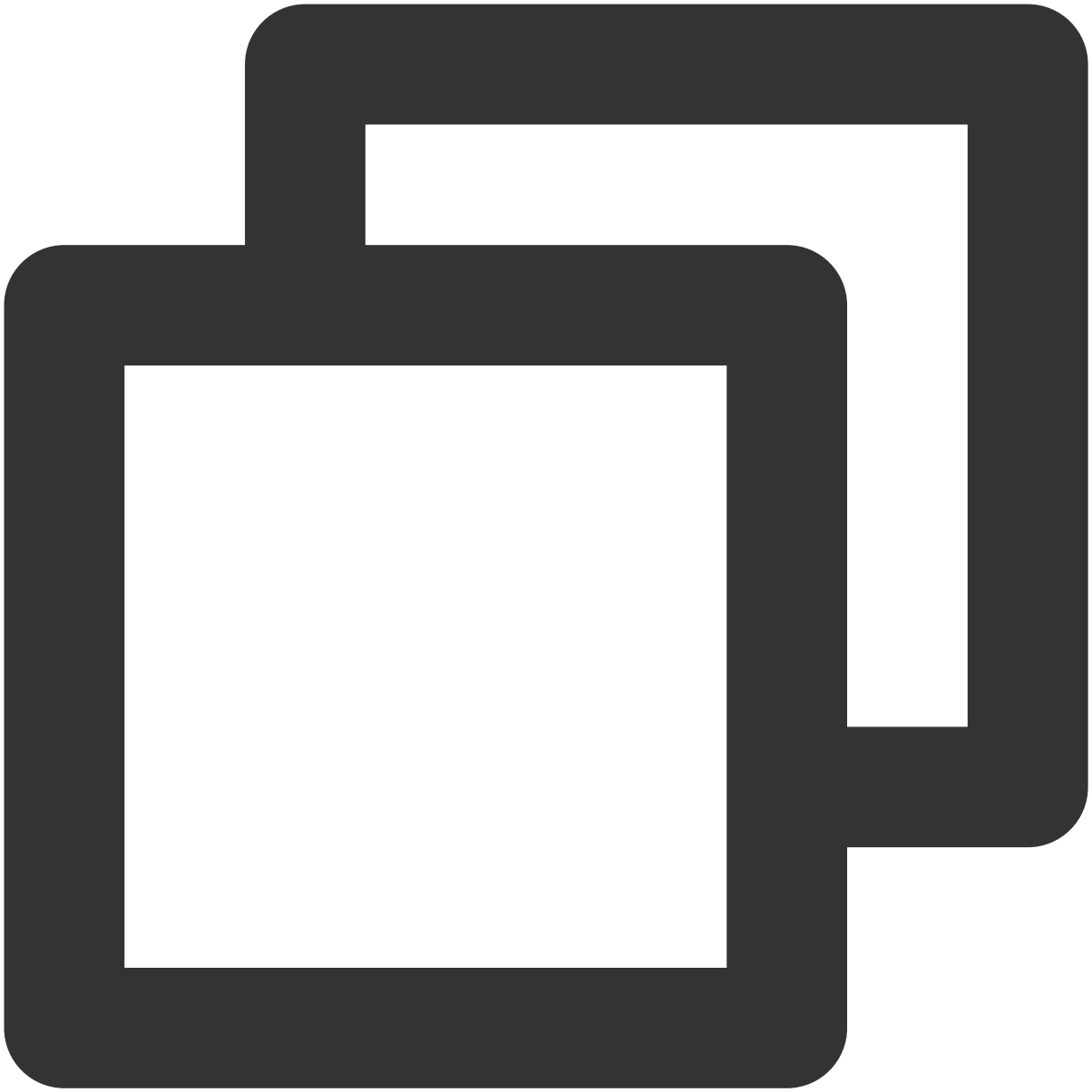
```
// app.js
console.log('app.js')
```

版权所有：腾讯云计算（北京）有限责任公司



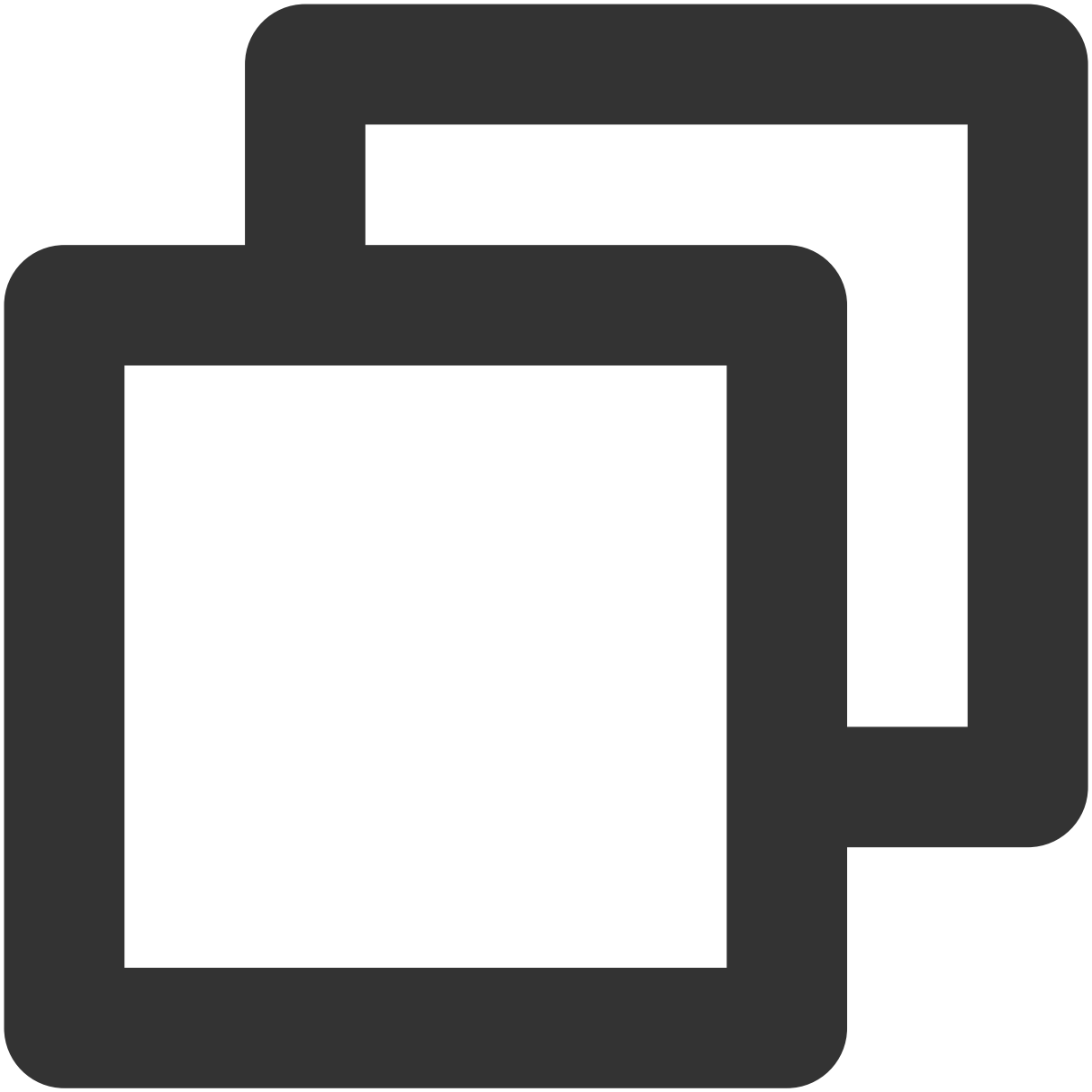
```
// pages/index/index
console.log('pages/index/index')
```

版权所有：腾讯云计算（北京）有限责任公司



```
// pages/log/log
console.log('pages/log/log')
```

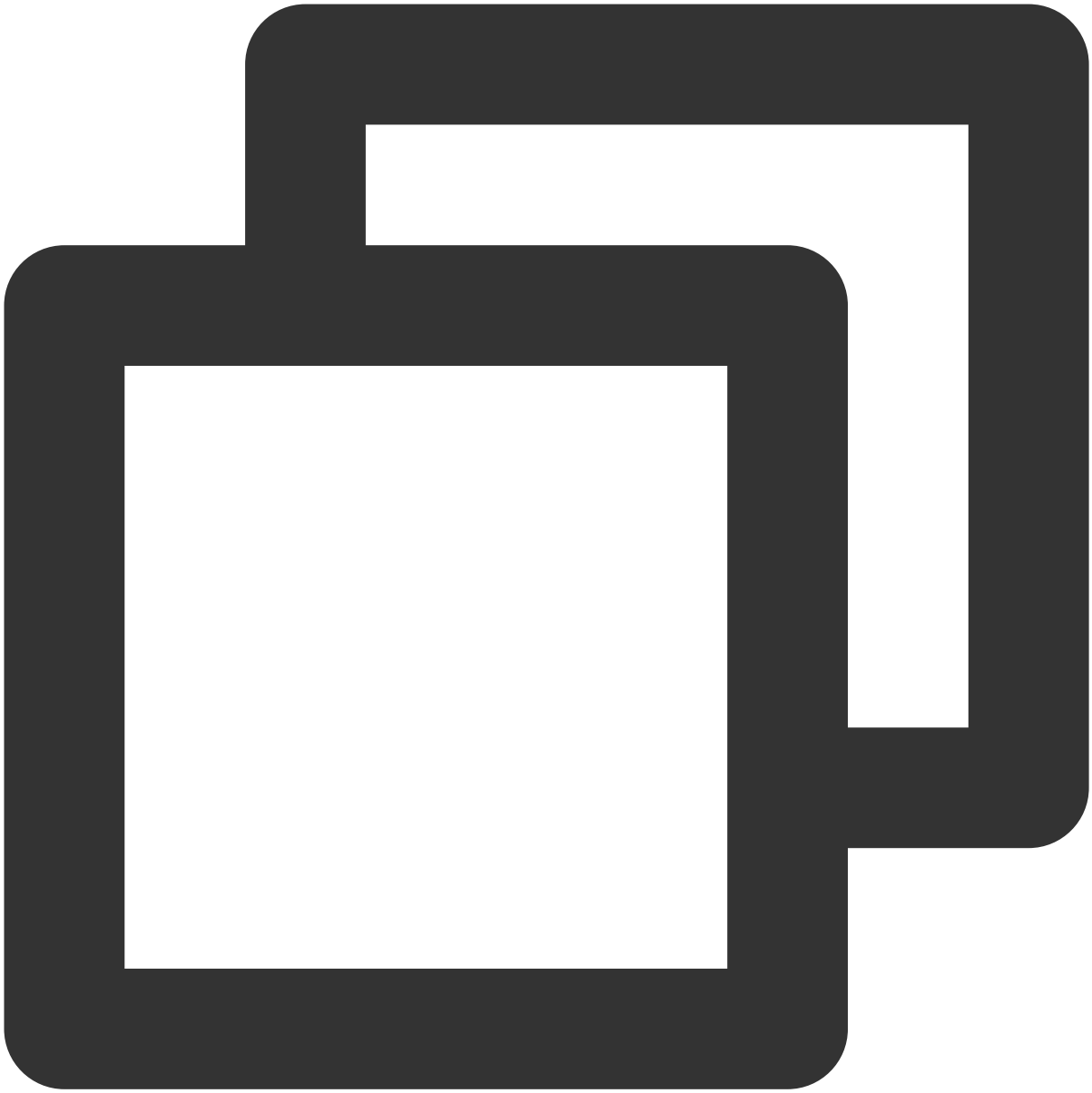
版权所有：腾讯云计算（北京）有限责任公司



```
// pages/result/result
console.log('pages/result/result')
```

以上文件执行后输出的结果如下：

版权所有：腾讯云计算（北京）有限责任公司

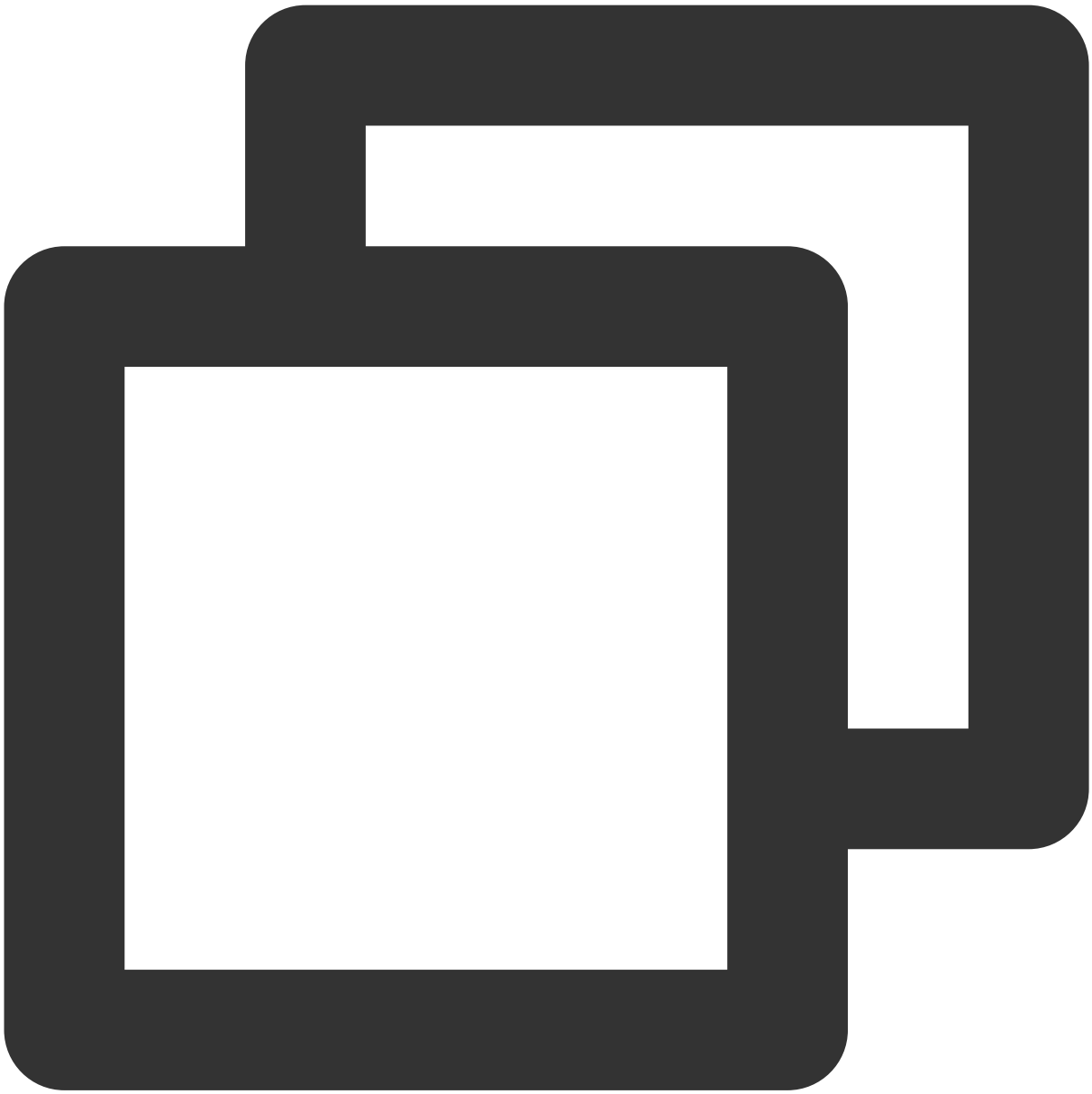


```
app.js
pages/index/index
pages/log/log
pages/result/result
```

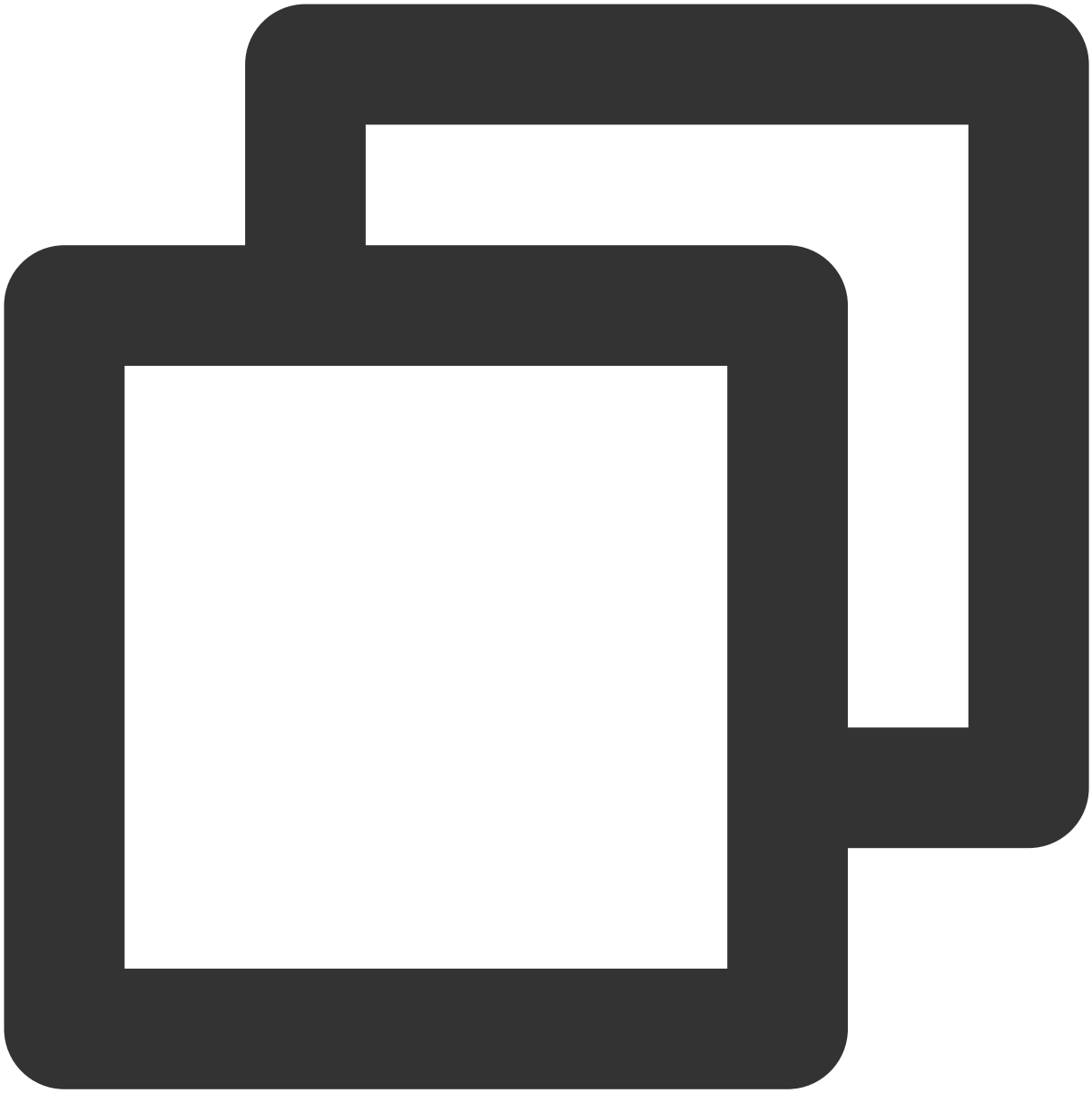
作用域

同浏览器中运行的脚本文件有所不同，小程序的脚本的作用域同 NodeJS 更为相似。

在文件中声明的变量和函数只在该文件中有效，不同的文件中可以声明相同名字的变量和函数，不会互相影响，如下代码所示：



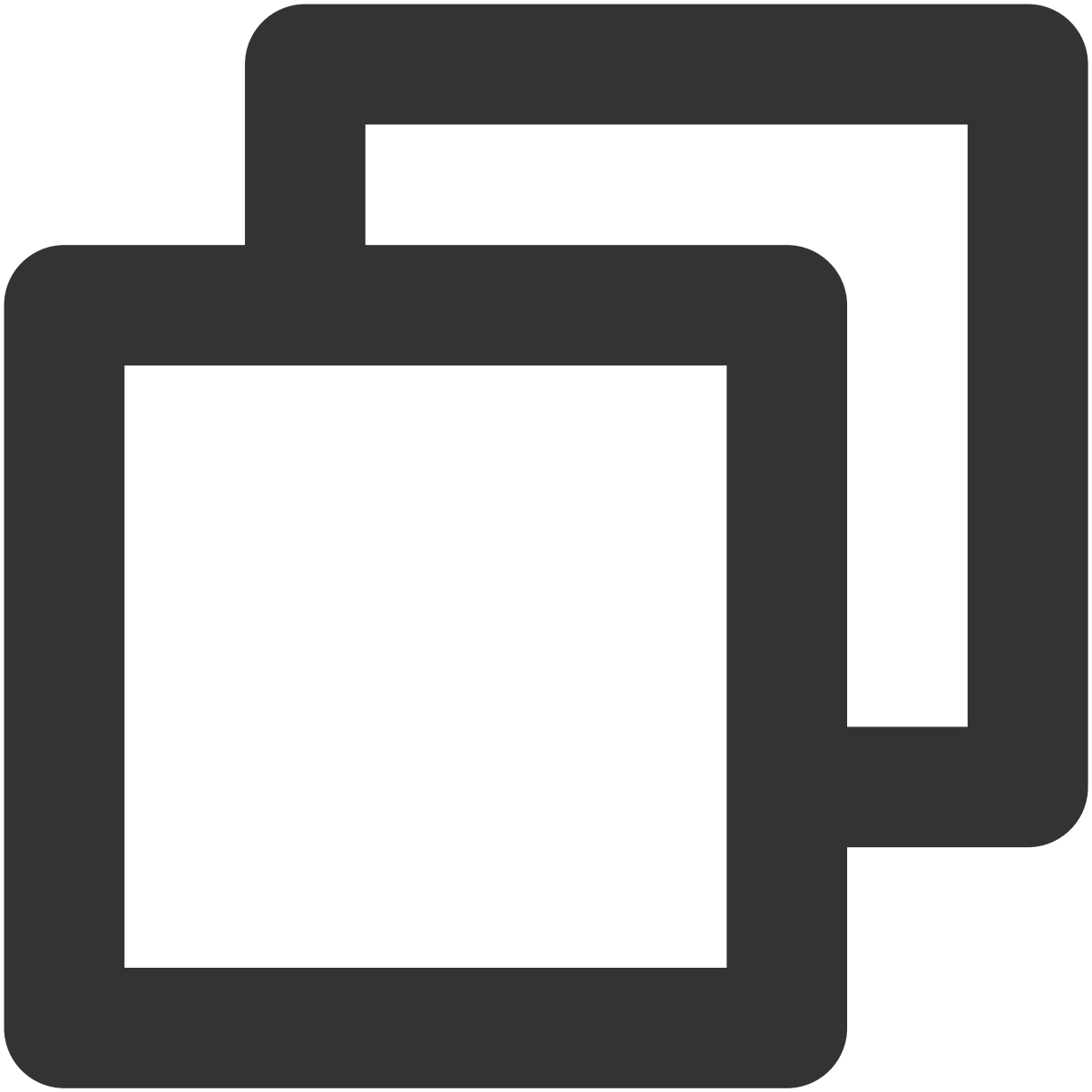
```
// a.js
// 定义局部变量
var localValue = 'a'
```



```
// b.js
//在脚本 b.js 中无法访问 a.js 定义的变量：
// 定义局部变量console.log(localValue) // 触发一个错误 b.js中无法访问 a.js 中定义的变量
```

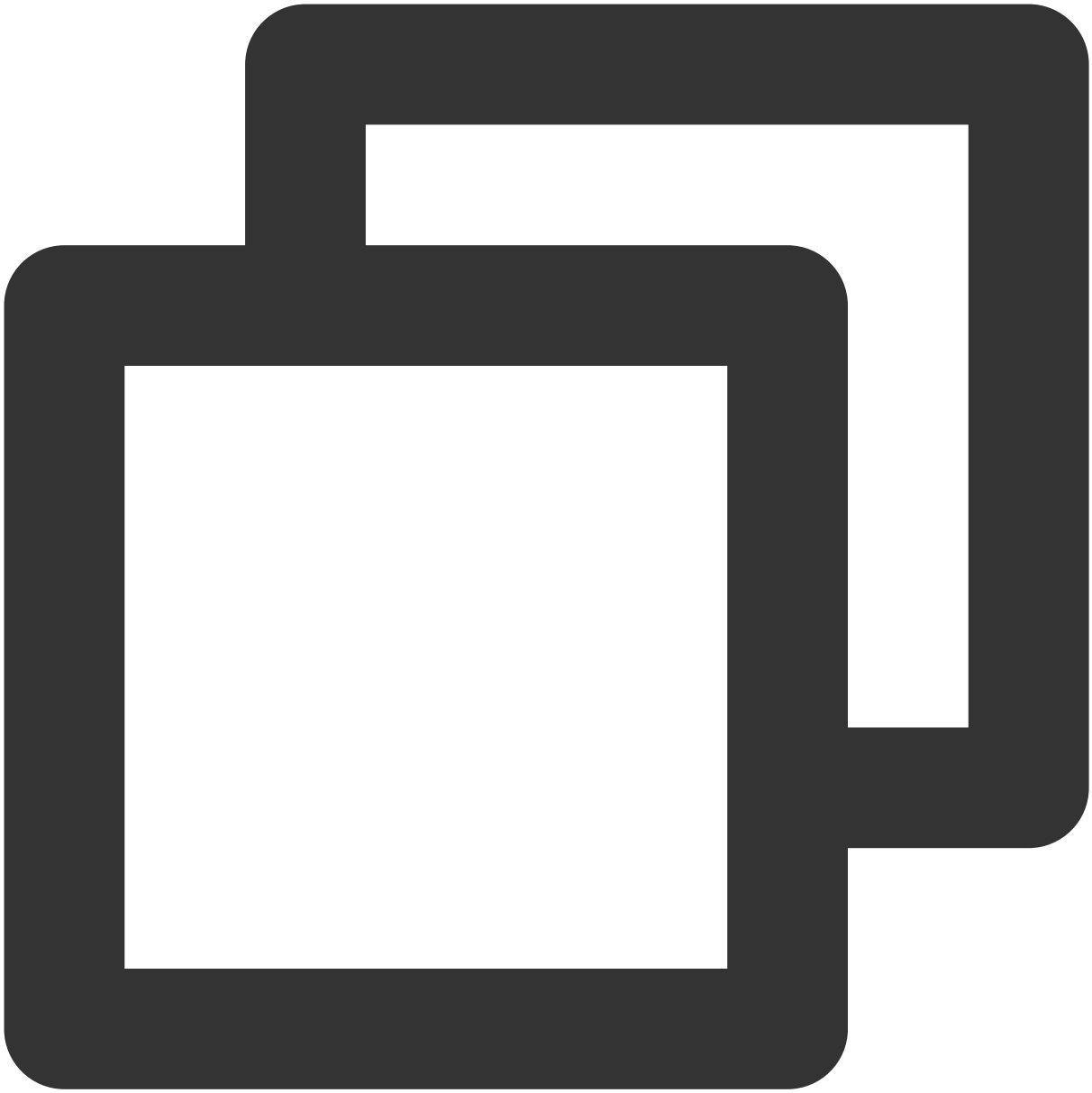
当需要使用全局变量的时，通过使用全局函数 getApp() 获取全局的实例，并设置相关属性值，来达到设置全局变量的目的，如下代码所示：

版权所有：腾讯云计算（北京）有限责任公司



```
// a.js
// 获取全局变量
var global = getApp()
global.globalValue = 'globalValue'
```

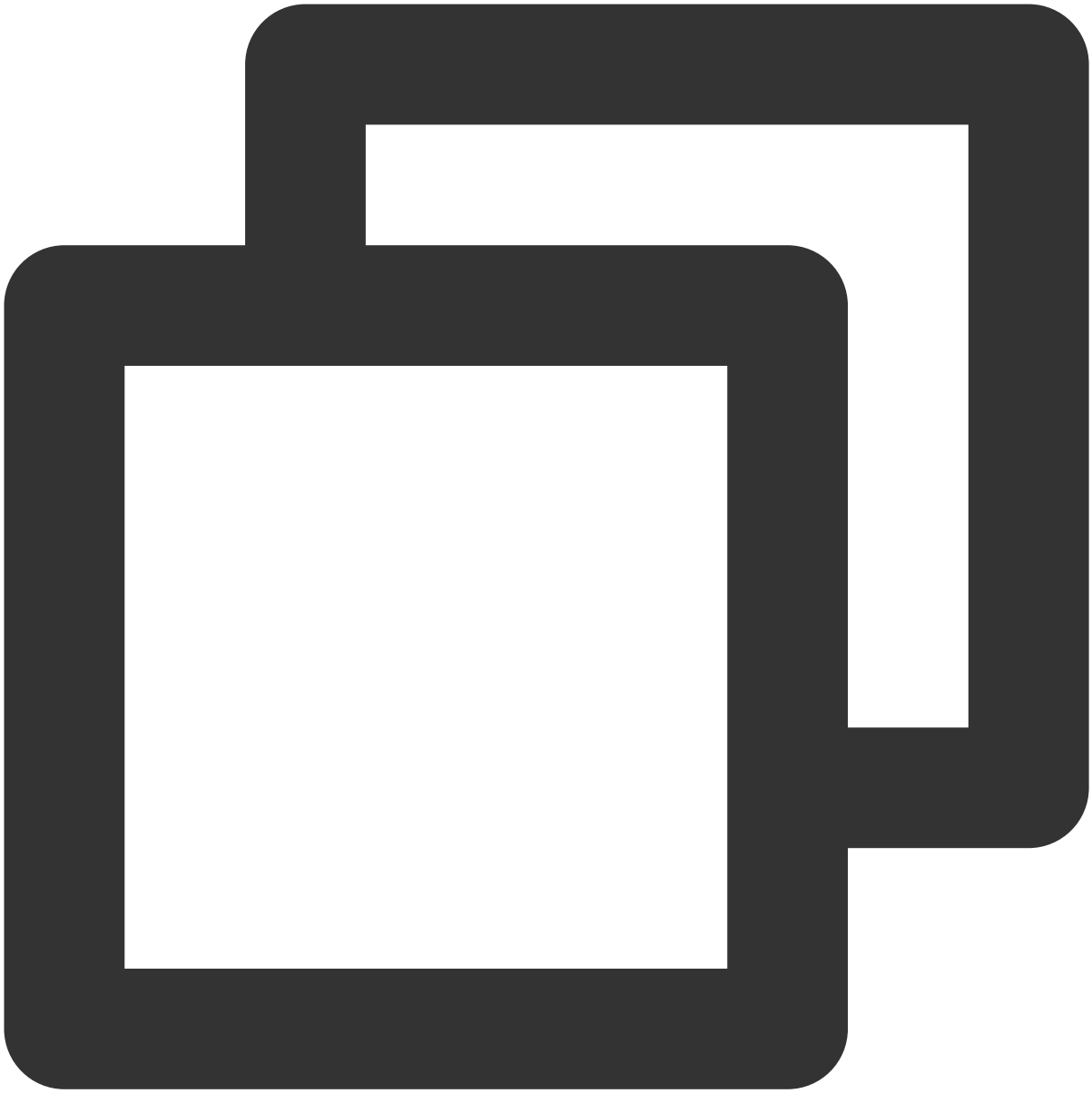
版权所有：腾讯云计算（北京）有限责任公司



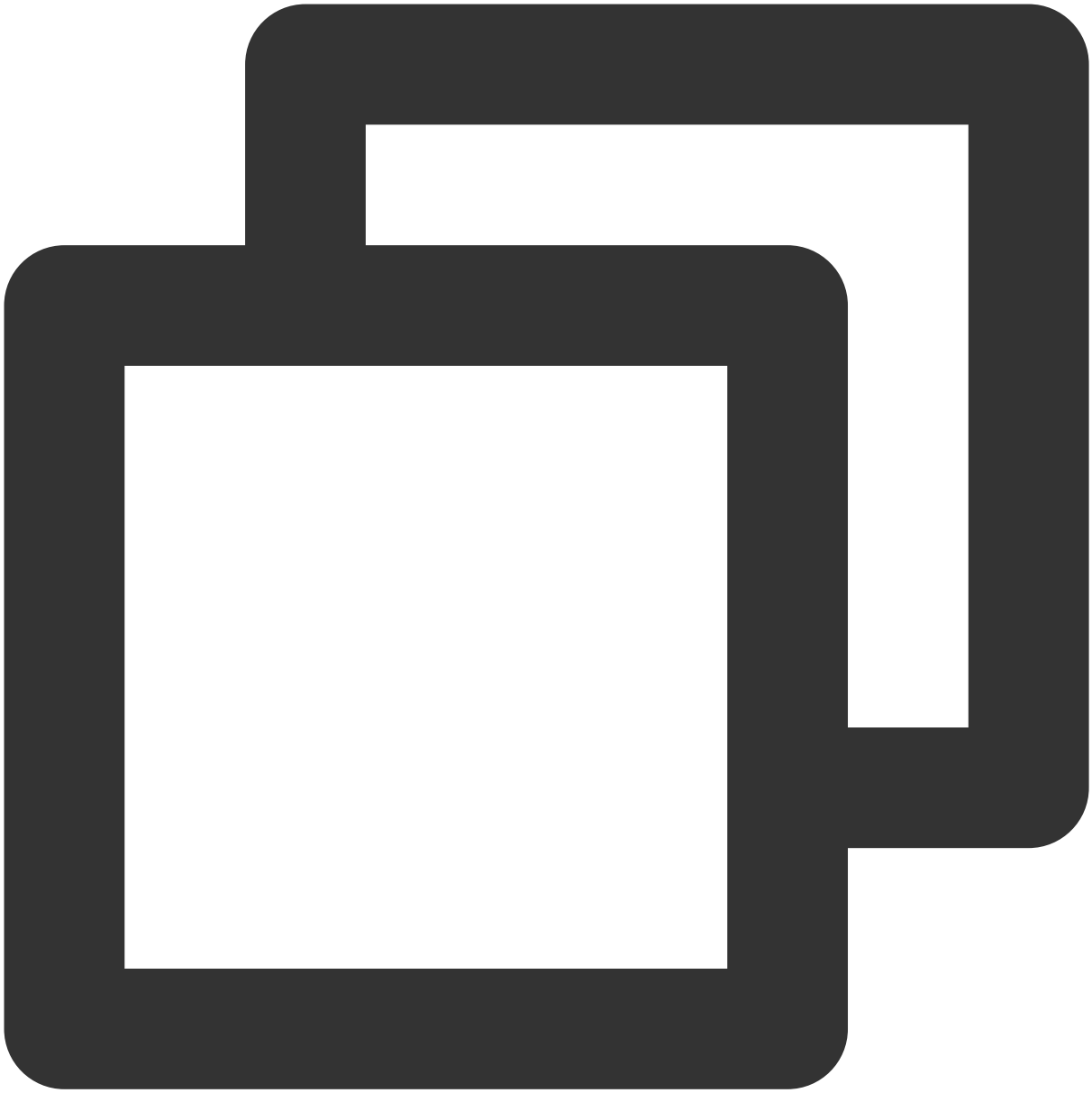
```
// b.js
// 访问全局变量
var global = getApp()
console.log(global.globalValue)
// 输出 globalValue
```

需要注意的是，上述示例只有在 a.js 比 b.js 先执行才有效，当需要保证全局的数据可以在任何文件中安全的被使用到，那么可以在 App() 中进行设置，如下代码所示：

版权所有：腾讯云计算（北京）有限责任公司



```
// app.js
App({
  globalData: 1
})
```

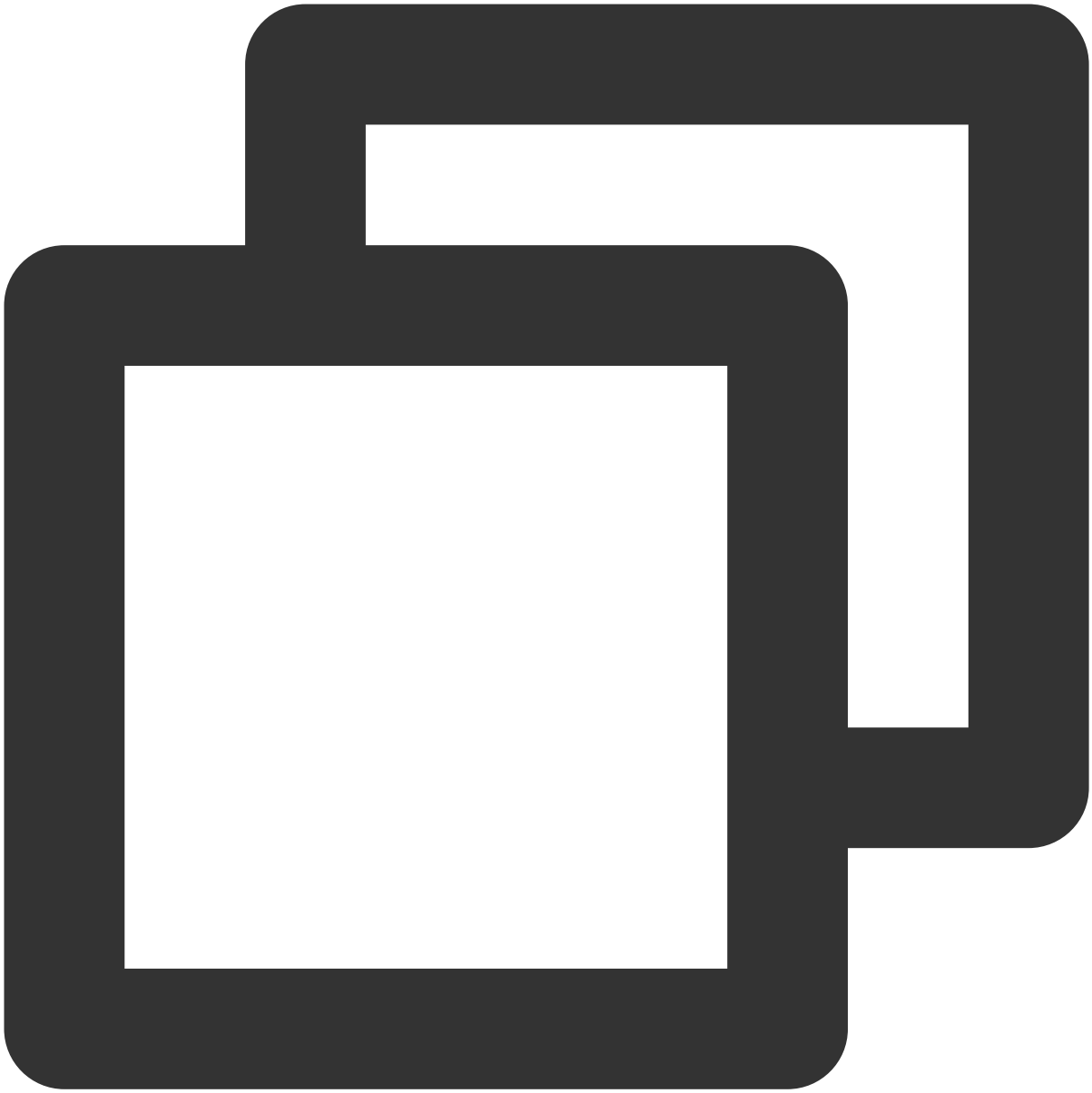


```
// a.js
// 局部变量
var localValue = 'a'

// 获取 global 变量
var app = getApp()

// 修改 global 变量
app.globalData++  // 执行后 globalData 数值为 2
```

版权所有：腾讯云计算（北京）有限责任公司



```
// b.js
// 定义另外的局部变量，并不会影响 a.js 中文件变量
var localValue = 'b'

// 如果先执行了 a.js 这里的输出应该是 2
console.log(getApp().globalData)
```

指南
了解小程序
小程序宿主环境

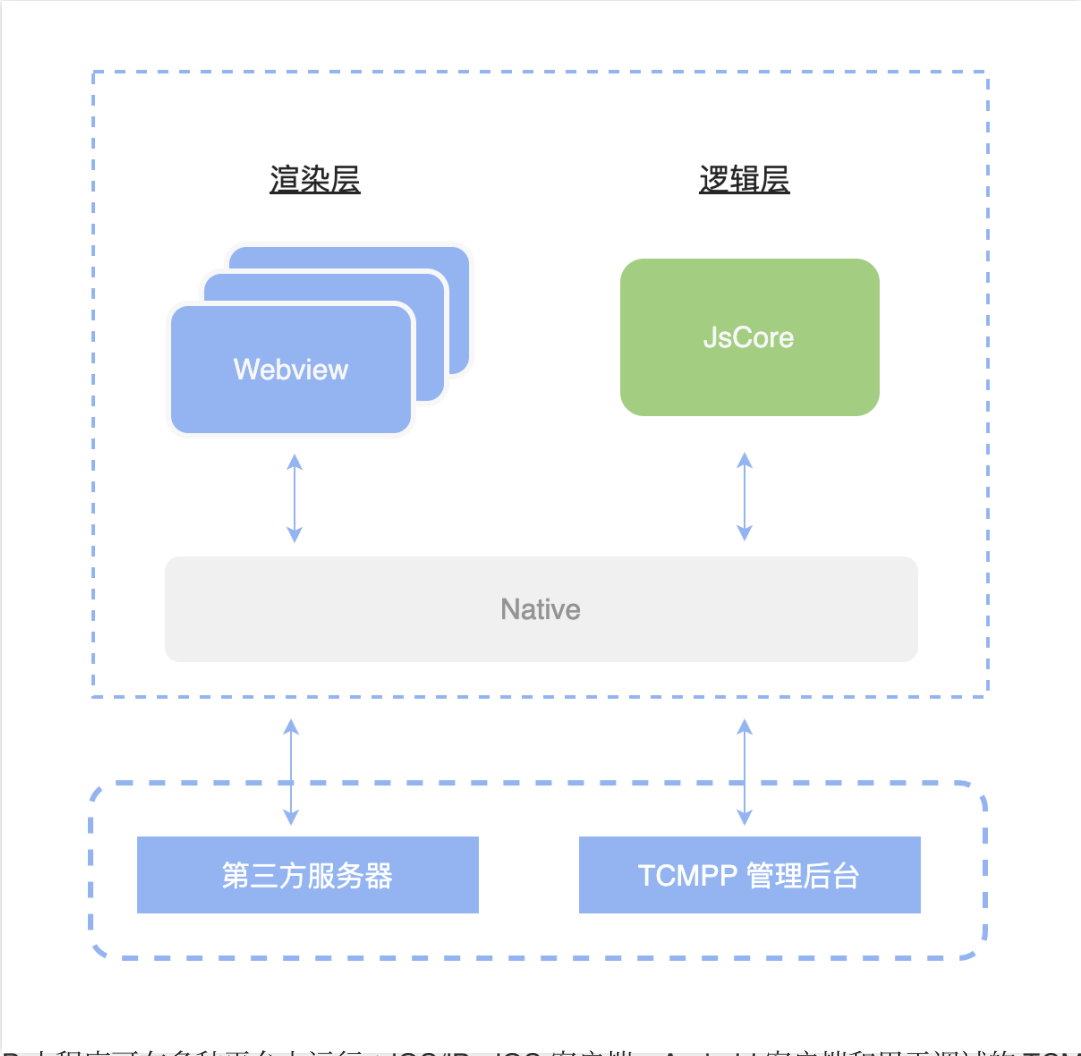
最近更新时间：2024-07-12 18:00:07

TCMPP 小程序可运行在各类型移动应用中，在加载启动小程序时，所需要的运行环境与能力需要由客户端应用向小程序提供。我们将这套环境称为 “ 宿主环境（宿主应用） ”，小程序借助宿主环境提供的能力，可以完成许多普通网页无法实现的功能。

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** 2个部分。
其中逻辑层用来加载小程序中负责处理业务逻辑的 JS 脚本，而渲染层则用来渲染 `WXML` 模板与 `WXSS` 样式，显示最终的页面。
在小程序中，逻辑层与渲染层则分别由 2 个独立线程进行管理。
逻辑层采用 JS Core 引擎运行 `JS` 脚本，用来处理业务逻辑；
渲染层界面使用 WebView 进行渲染（一个小程序中如果存在多个页面，则会存在多个 WebView 线程）。
因此一个小程序应用中只有一个 JS Core 线程，以及多个 WebView 线程，这两个线程的通信会经由宿主客户端（下文中也会采用 Native 来代指宿主客户端）做中转，逻辑层发送网络请求也经由 Native 转发。
小程序的通信模型下图所示：

版权所有：腾讯云计算（北京）有限责任公司

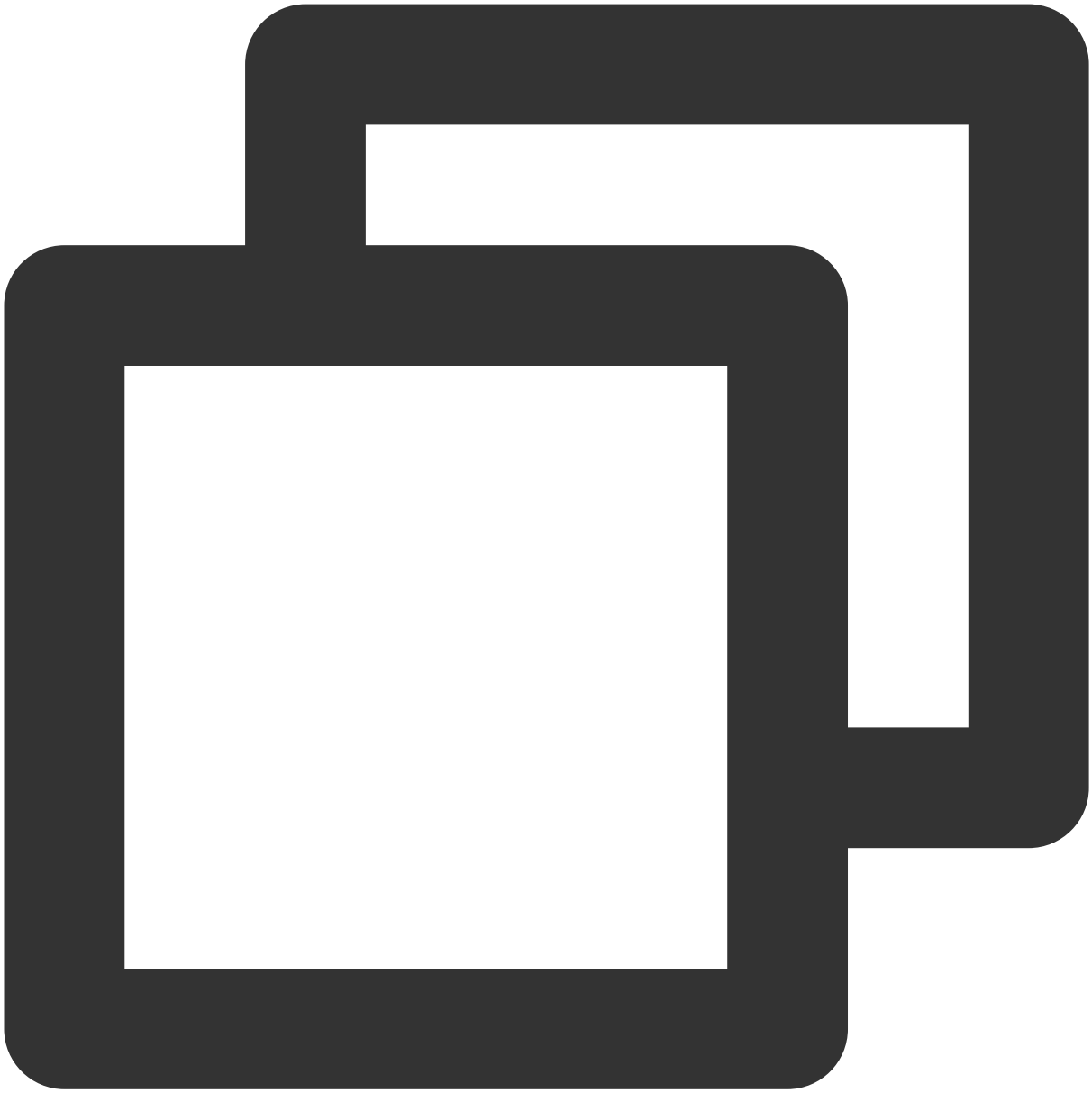


TCMPP 小程序可在多种平台上运行：iOS/iPadOS 客户端、Android 客户端和用于调试的 TCMPP 开发者工具等。
不同运行环境下，脚本执行环境和用于组件渲染的环境是不同的，性能表现也存在差异：
在 iOS、iPadOS 上，小程序逻辑层的 JavaScript 代码运行在 JavaScriptCore 中，视图层是由 WKWebView 来渲染的，环境有 iOS 14、iPad OS 14等。
在 Android 上，小程序逻辑层的 JavaScript 代码运行在 V8 中，视图层是由基于 Mobile Chromium 内核的自研 XWeb 引擎来渲染的。
在开发工具上，小程序逻辑层的 JavaScript 代码是运行在 NW.js 中，视图层是由 Chromium Webview 来渲染的。
JavaScriptCore 无法开启 JIT 编译 (Just-In-Time Compiler)，同等条件下的运行性能要明显低于其他平台。

2. 程序与页面

宿主客户端在打开小程序之前，会把整个小程序的代码包下载到本地。然后通过解析 `app.json` 里的 `pages` 字段，就可以知道该小程序的所有页面路径：

版权所有：腾讯云计算（北京）有限责任公司

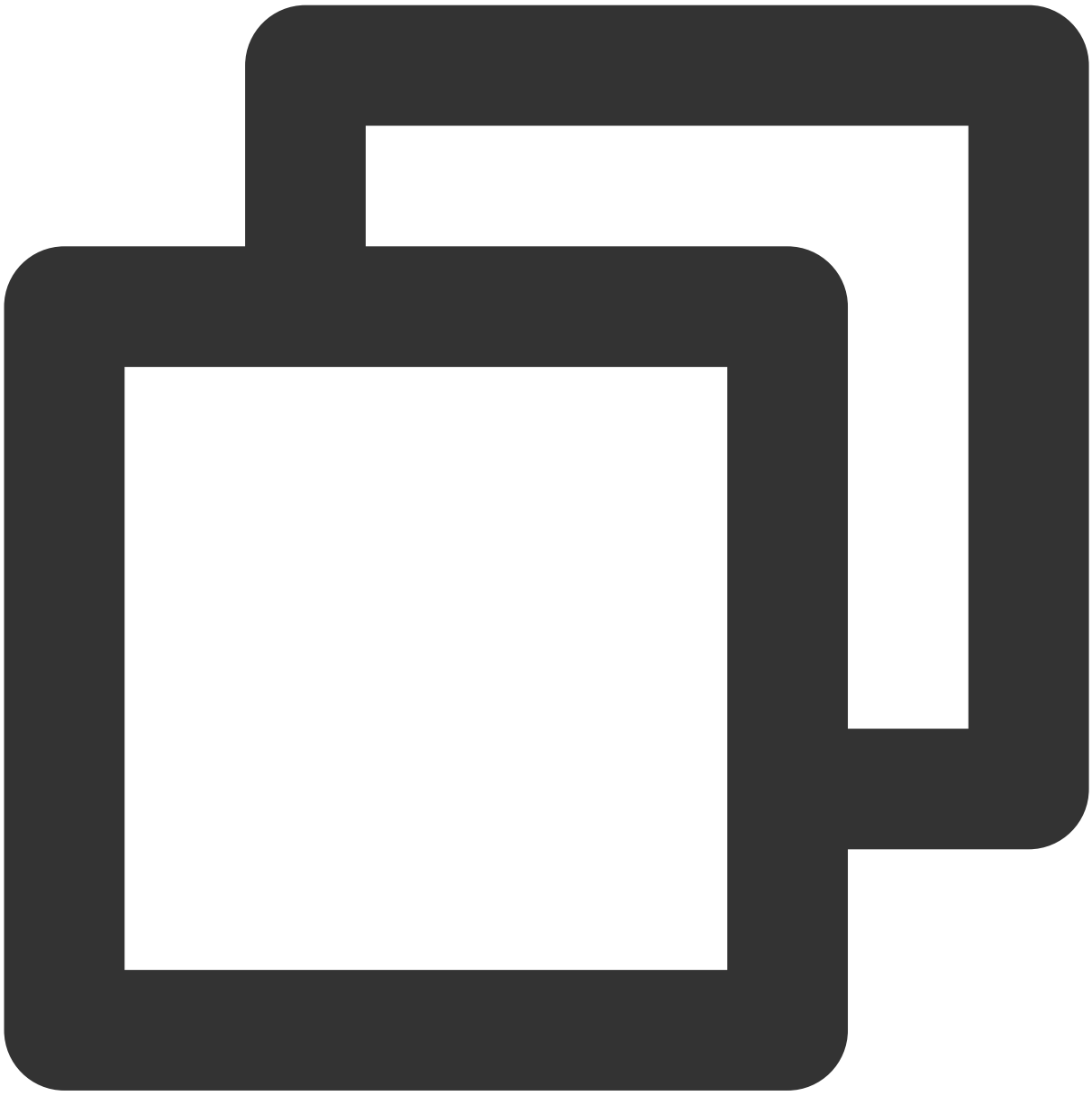


```
{
  "pages":[
     "pages/index/index",
     "pages/logs/logs"
   ]
}
```

这个配置说明在项目中定义了两个页面，分别位于 `pages/index/index` 和 `pages/logs/logs` 。而写在 pages 字段的第一个页面就是这个小程序的首页（打开小程序看到的第一个页面）。

版权所有：腾讯云计算（北京）有限责任公司

每个页面的路径，在程序代码包里都可以找到对应的文件路径。例如 `pages/index/index` ，在该路径下，存在着这个页面对应的 `WXML` 、 `WXSS` 、 `JS` 文件与 `JSON` 配置文件。
小程序启动之后，在 `app.js` 定义的 App 实例的 onLaunch 回调会被执行：

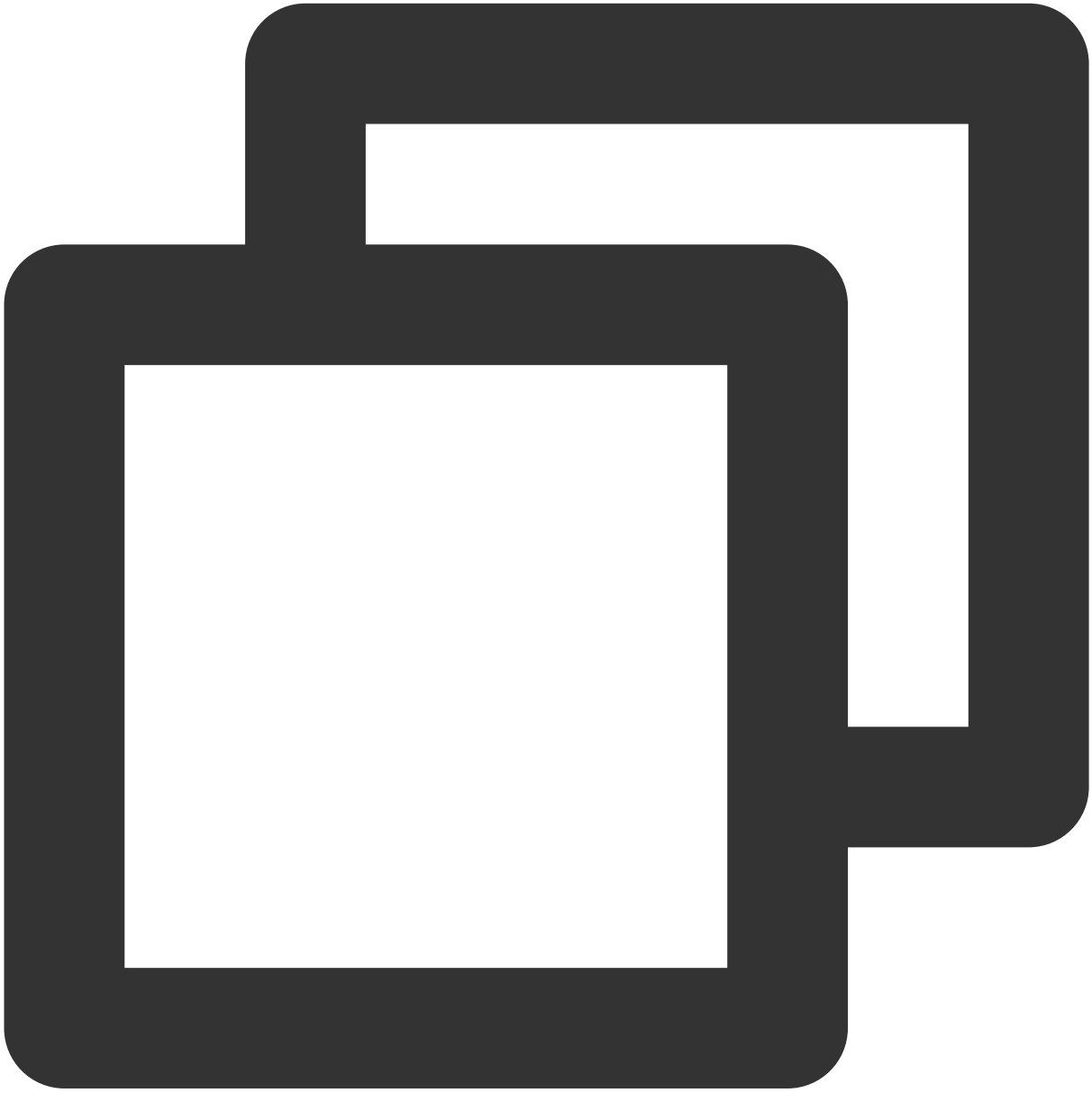


```
App({
  onLaunch() {
    // 小程序启动之后 触发
  }
})
```

整个小程序只有一个 App 实例，是全部页面共享的，更多的事件回调参考文档 注册小程序。

接下来我们简单看看小程序的一个页面是怎么写的。

你可以观察到 `pages/logs/logs` 下其实是包括了4种文件的，宿主客户端会先根据 `logs.json` 配置生成一个界面，顶部的颜色和文字你都可以在这个 json 文件里边定义好。紧接着客户端就会装载这个页面的 `WXML` 结构和 `WXSS` 样式。最后客户端会装载 `logs.js` ，你可以看到 `logs.js` 的大体内容就是：



```
Page({
  data: { // 参与页面渲染的数据
    logs: []
  },
  onLoad() {
```

```
    // 页面渲染后 执行
  }
})
```

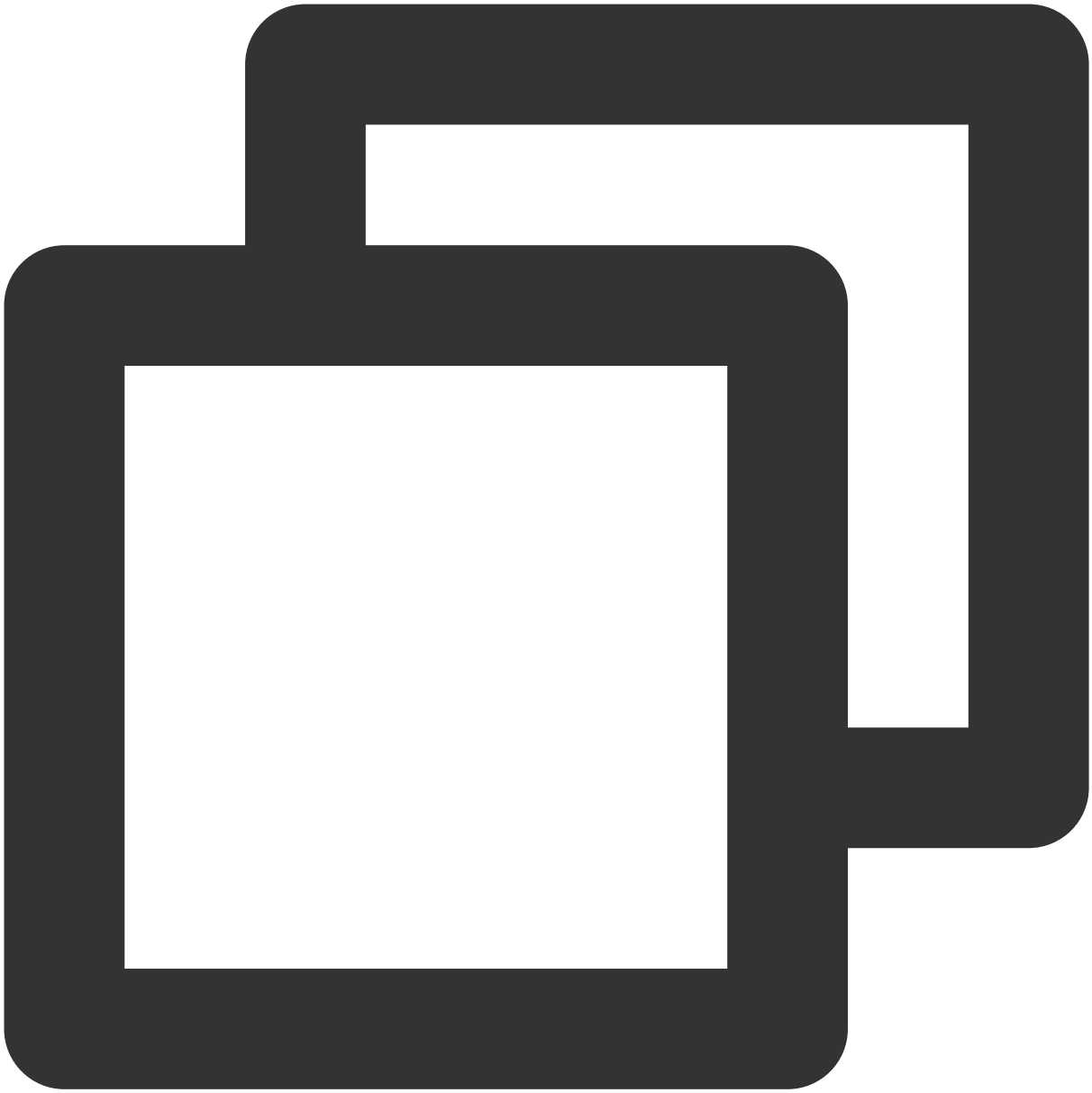
Page 是一个页面构造器，这个构造器就生成了一个页面， `data` 表示的是该页面需要用到的渲染数据。在生成页面的时候，小程序框架会把 `data` 数据和 `index.wxml` 一起渲染出最终的结构，于是就得到了你看到的小程序的样子。
在渲染完界面之后，页面实例就会收到一个 `onLoad` 的回调，你可以在这个回调处理你的逻辑。

> 有关于 Page 构造器更多详细的文档参考 注册页面。

3. 组件

组件是小程序提供给开发者来创建页面 UI，自定义 `WXML` 的。就像 HTML 的 , <p> 等标签一样，在小程序里边，你只需要在 WXML 写上对应的组件标签名字就可以把该组件显示在界面上。例如:

版权所有：腾讯云计算（北京）有限责任公司



```
<view>Welcome to TCMPP</view>
```

该页面就显示一行文案：`Welcome to TCMPP`。开发者可以通过 style 和 class 来控制该组件的样式，来指定宽、高、颜色等。

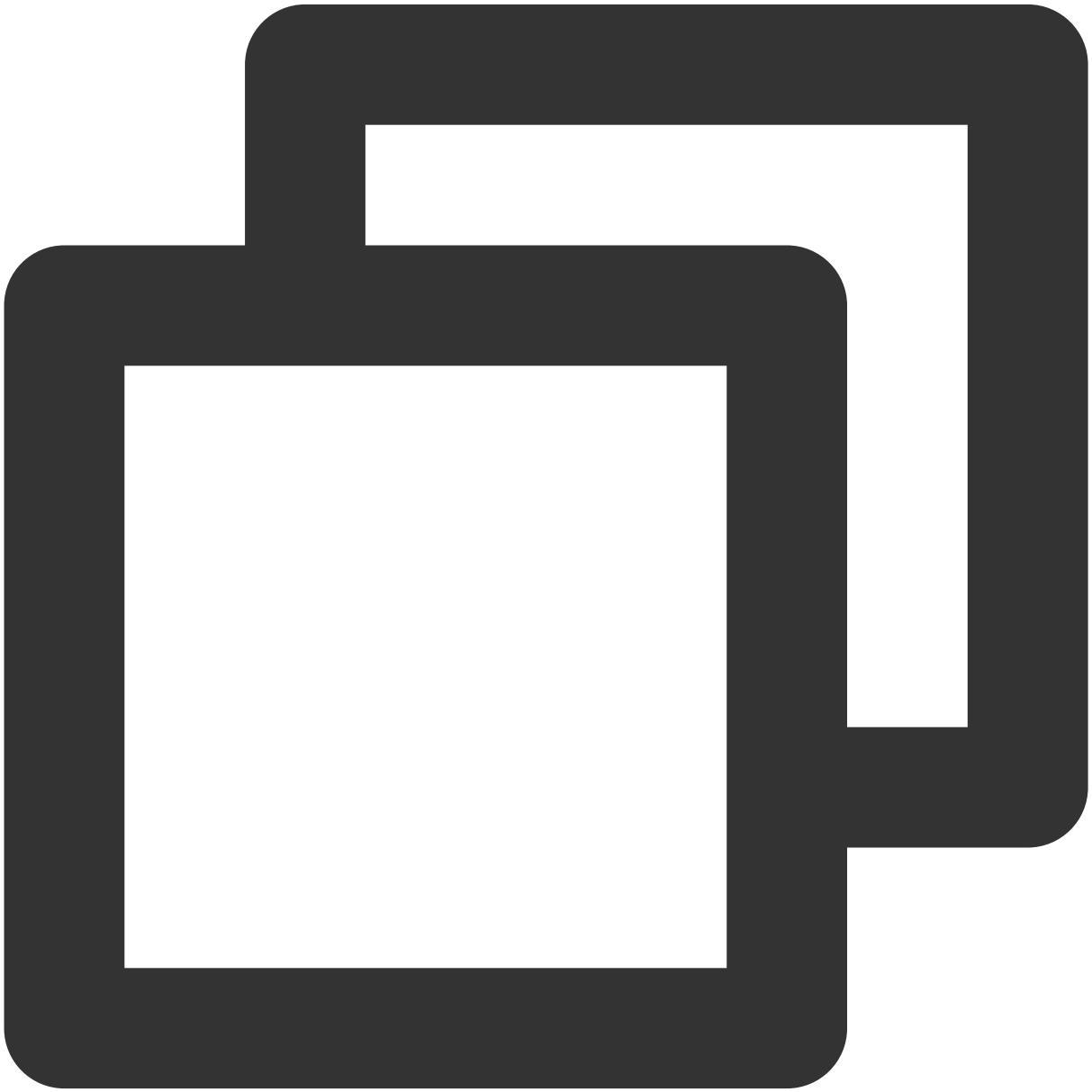
> 更多的组件可以参考 组件概览。

4. API

为了让开发者更方便的使用小程序所提供的能力，TCMPP 提供了多种 API 供开发者使用，开发者可以使用这些 API 对小程序界面进行修改，获取设备当前位置，播放视频、音频等。

API 分为同步与异步两种，同步 API 会直接返回结果，而异步 API 会通过回调返回结果。开发者需要根据 API 的回调方式处理逻辑，传入正确的参数处理业务。

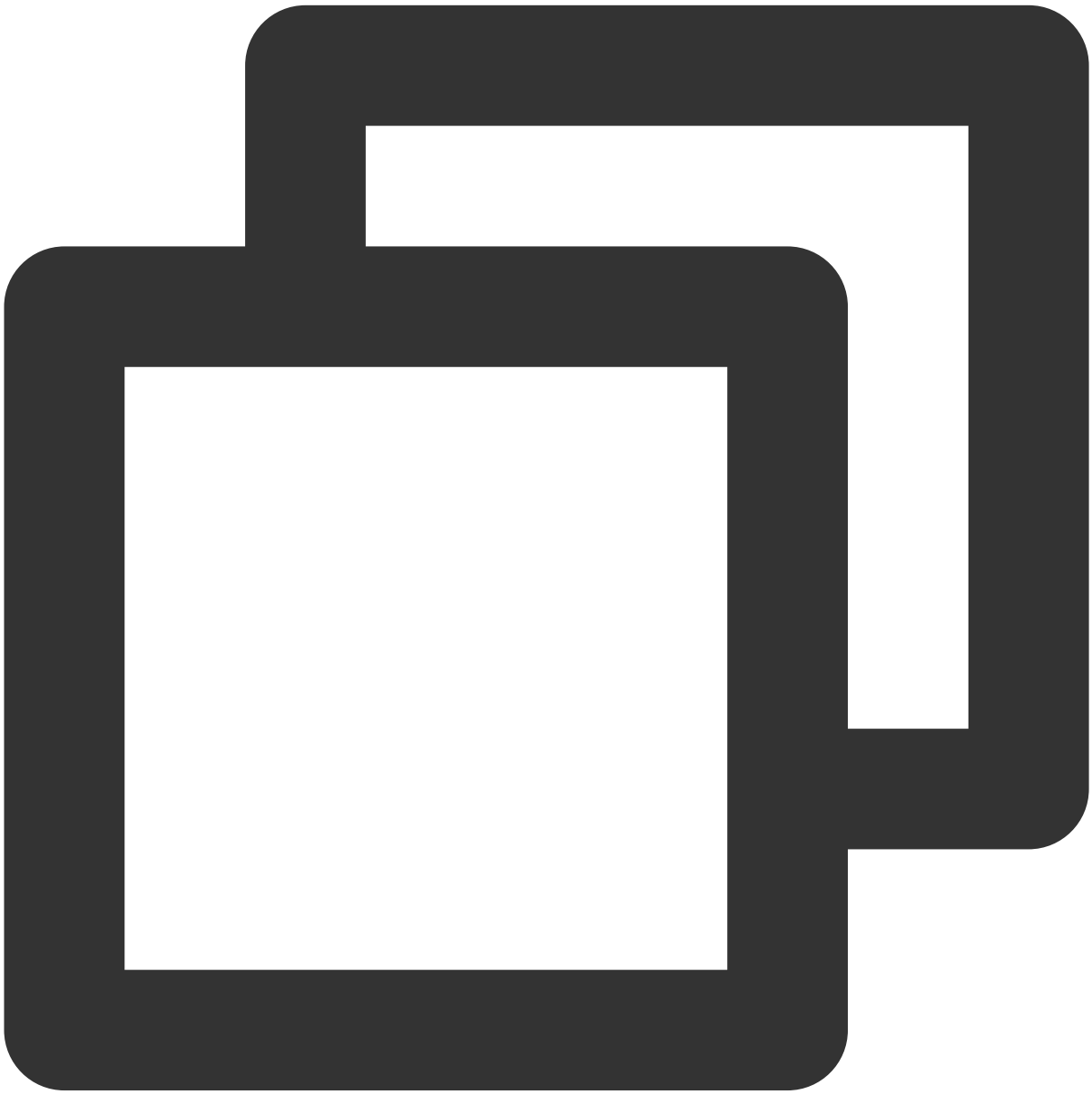
要获取用户地理位置时，只需要异步调用 `wx.getLocation` ：



```
wx.getLocation({
  type: 'wgs84',
```

```
  success: (res) => {
    var latitude = res.latitude // 纬度
    var longitude = res.longitude // 经度
  }
})
```

要获取设备信息时，只需要同步或异步调用 `wx.getSystemInfo`：



```
// 异步调用
wx.getSystemInfo({
  success (res) {
```

```
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
    console.log(res.platform)
  }
})

// 同步调用
try {
  wx.getSystemInfoSync();
} catch(e) {}
```

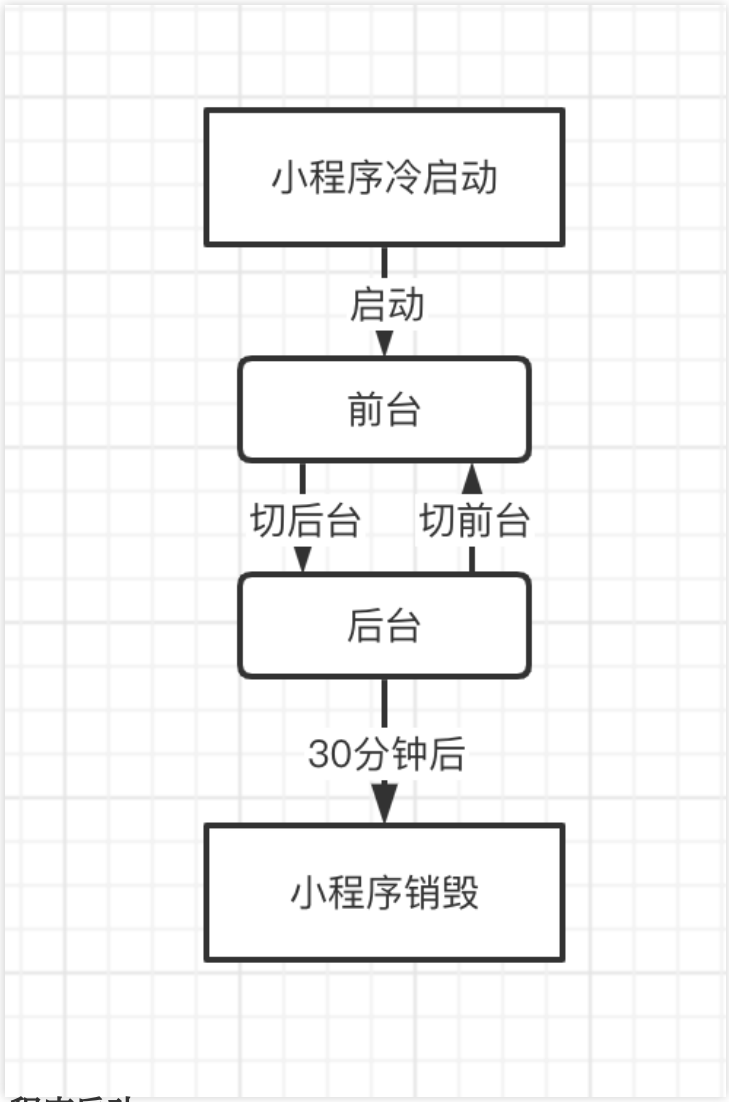
更多的 API 能力见 API 概览。

小程序运行机制

最近更新时间：2024-07-12 17:58:26

1. 小程序生命周期

小程序从启动到最终被销毁，会经历多种状态，不同状态下表现不同。



1.1 小程序启动

小程序会有两种情况，一种是 `冷启动` ，一种是 `热启动` ；

热启动：指用户在一定时间后想要重新打开已打开的小程序时，将 `后台` 运行的小程序切换到 `前台` 的过程。

冷启动：当用户第一次打开小程序或被宿主应用主动销毁后重新打开小程序，此时小程序需要重新加载启动，即冷启动。

1.2 前台/后台状态

小程序启动后，用户直接可以看到小程序的界面，此时小程序处于 `前台` 状态。
当用户单击右上角的小程序控制按钮「关闭」小程序或按手机上的 Home 键离开宿主应用时，小程序没有被销毁，而是在 `后台` 运行，此时小程序还可以短暂运行一小段时间，但部分 API 的使用会受到限制。
切 `后台` 的方式包括但不限于以下几种：
点击右上角胶囊按钮离开小程序；
iOS 从屏幕左侧右滑离开小程序；
安卓点击返回键离开小程序；
小程序前台运行时直接把宿主应用切后台（手势或 Home 键）；
小程序前台运行时直接锁屏。
当用户再次进入客户端并打开小程序时，小程序将从 `后台` 运行到 `前台` 。

1.3 小程序销毁机制

如果用户很久没有使用小程序，或者系统资源紧张，小程序会被「**销毁**」，即完全终止运行。具体而言包括以下几种情形：
当小程序进入后台并被「挂起」后，如果很长时间（目前是 30 分钟）都未再次进入前台，小程序会被销毁。
当小程序占用系统资源过高，可能会被系统销毁或被宿主应用主动回收。
在 iOS 系统上，当宿主应用在一定时间间隔内连续收到系统内存告警时，会根据一定的策略，主动销毁小程序，并提示用户 `运行内存不足，请重新打开该小程序` 。
具体策略会持续进行调整优化。

2. 小程序打开逻辑

2.1 冷启动打开页面

小程序冷启动时，打开的页面有以下情况；
若启动的场景不带 path，则进入首页；
若启动的场景带 path，则启动进入对应 path 的页面。

2.2 热启动打开页面

小程序热启动时，打开的页面有以下情况；
若启动的场景中不带 path，则保留上次的浏览的状态；
若启动的场景中带有 path，则 reLaunch 到对应 path 的页面。

版权所有：腾讯云计算（北京）有限责任公司

小程序代码结构

最近更新时间：2024-07-12 17:54:49

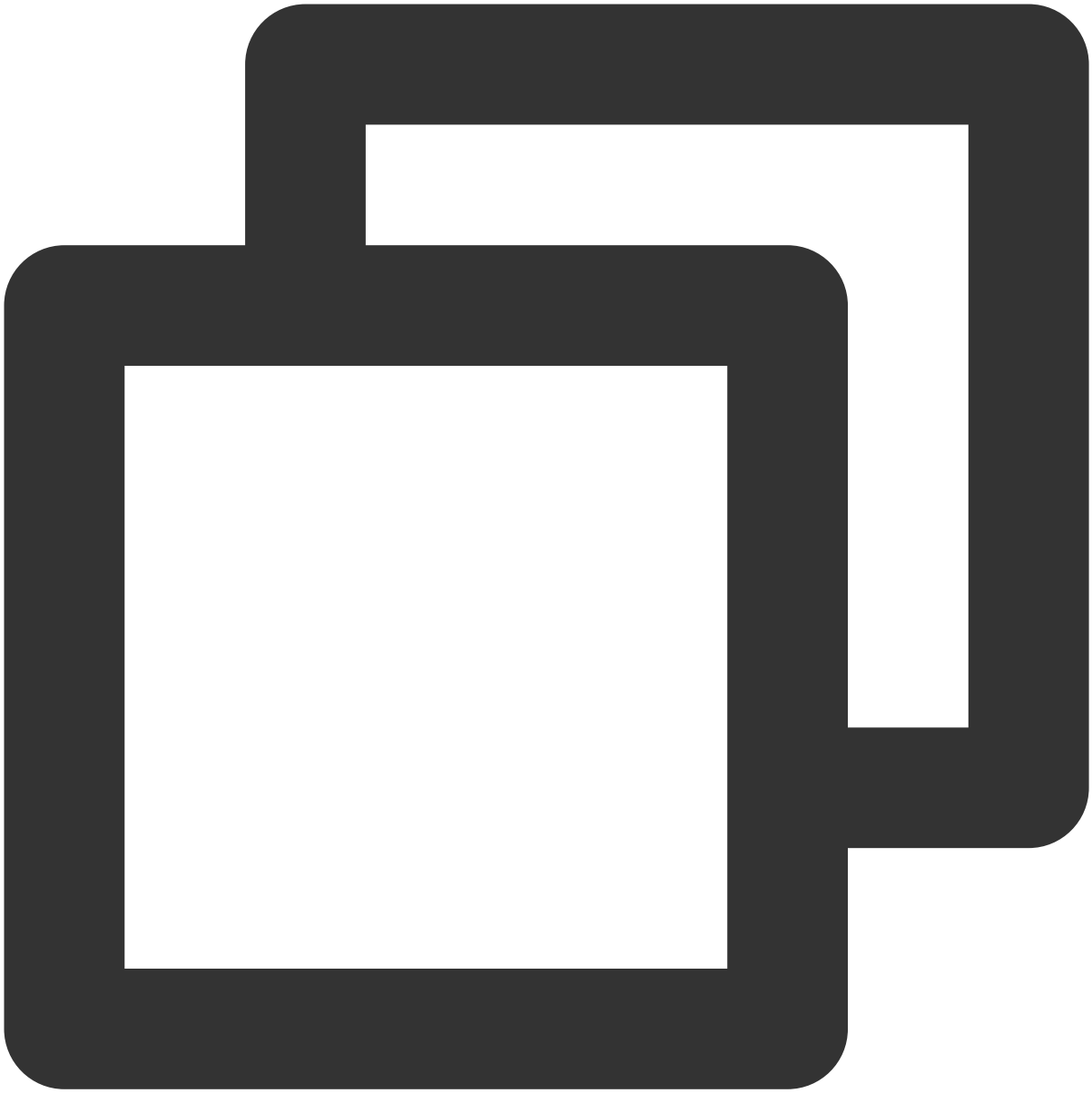
1. JSON配置

JSON 是一种数据格式，并不是编程语言，在小程序中，JSON 扮演的静态配置的角色。
我们可以看到在项目的根目录有一个 `app.json` 和 `project.config.json`，此外在 `pages/logs` 目录下还有一个 `logs.json`，我们依次来说明一下它们的用途。

1.1 小程序配置 app.json

`app.json` 是当前小程序的全局配置，包括了小程序的所有页面路径、界面表现、网络超时时间、底部 tab 等。

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "TCMPP",
    "navigationBarTextStyle":"black"
  }
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
```

我们简单说一下这个配置各个项的含义:

pages 字段：用于描述当前小程序所有页面路径，这是为了让宿主客户端知道当前您的小程序页面定义在哪个目录。

Window 字段：定义小程序所有页面的顶部背景颜色，文字颜色定义等。

> 其他配置项细节可以参考文档 小程序配置 app.json 。

1.2 工具配置 project.config.json

通常大家在使用一个工具的时候，都会针对各自喜好做一些个性化配置，例如编译配置，当您换了另外一台电脑重新安装工具的时候，您还要重新配置。

考虑到这点，小程序开发者工具在每个项目的根目录都会生成一个 `project.config.json` ，您在开发者工具上做的任何配置都会写入到这个文件，当您重新安装工具或者换电脑工作时，您只要载入同一个项目的代码包，开发者工具就自动会帮您恢复到当时您开发项目时的个性化配置，其中会包括项目名字、代码上传时自动压缩等等一系列选项。

1.3 页面配置 page.json

这里的 `page.json` 其实用来表示 `pages/logs` 目录下的 `logs.json` 这类和小程序页面相关的配置。

如果您整个小程序的风格是蓝色调，那么您可以在 `app.json` 里边声明顶部颜色是蓝色即可。实际情况可能每个页面都有不一样的色调来区分不同功能模块，因此我们提供了 `page.json` ，让开发者可以独立定义每个页面的一些属性，例如刚刚说的顶部颜色、是否允许下拉刷新等等。

> 其他配置项细节可以参考文档 页面配置 。

1.4 JSON语法

这里说一下小程序里 JSON 配置的一些注意事项。

JSON 文件都是被包裹在一个大括号中 {}，通过 key-value 的方式来表达数据。JSON 的 Key 必须包裹在一个双引号中，在实践中，编写 JSON 的时候，忘了给 Key 值加双引号或者是把双引号写成单引号是常见错误。

注意：

1. JSON 的值只能是以下几种数据格式，其他任何格式都会触发报错，例如 JavaScript 中的 undefined。

数字，包含浮点数和整数；

字符串，需要包裹在双引号中；

Bool 值，true 或者 false；

数组，需要包裹在方括号中 []；

对象，需要包裹在大括号中 {}；

版权所有：腾讯云计算（北京）有限责任公司

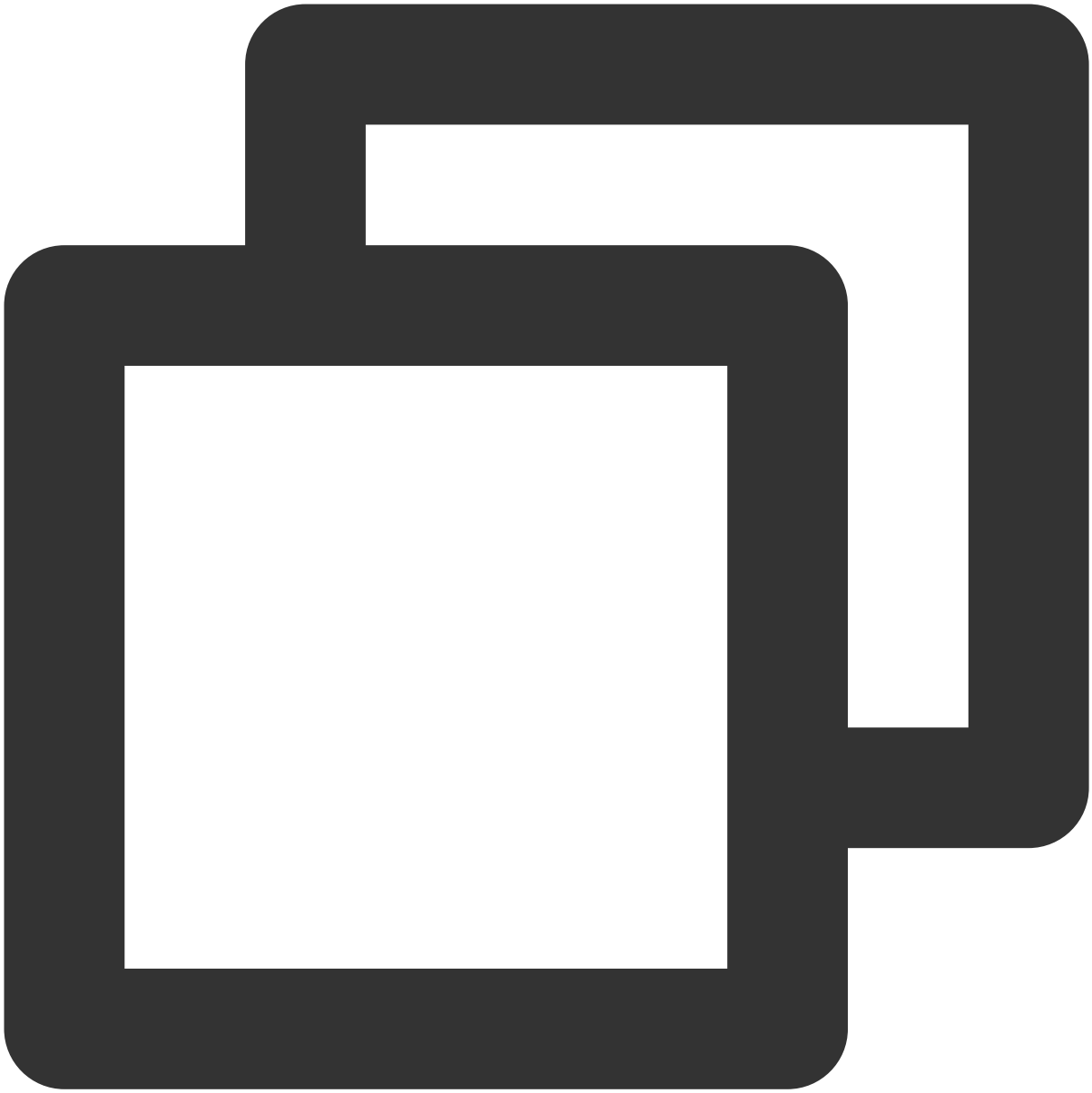
Null。

2. 还需要注意的是 JSON 文件中无法使用注释，试图添加注释将会引发报错。

2. WXML 模板

从事过网页编程的人知道，网页编程采用的是 HTML + CSS + JS 这样的组合，其中 `HTML` 是用来描述当前这个页面的结构， `CSS` 用来描述页面的样子， `JS` 通常是用来处理这个页面和用户的交互。

同样道理，在小程序中也有同样的角色，其中 `WXML` 充当的就是类似 `HTML` 的角色。打开 `pages/index/index.wxml` ，您会看到以下的内容：



```
<view class="container">
  <view class="userinfo">
    <button wx:if="{{!hasUserInfo && canIUse}}"> getUserInfo </button>
    <block wx:else>
      <image src="{{userInfo.avatarUrl}}" background-size="cover"></image>
      <text class="userinfo-nickname">{{userInfo.nickName}}</text>
    </block>
  </view>
  <view class="usermotto">
    <text class="user-motto">{{motto}}</text>
  </view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
</view>
```

和 HTML 非常相似， WXML 由标签、属性等等构成。但是也有很多不一样的地方，我们来一一阐述一下：

1. 标签名字不一样

往往写 HTML 的时候，经常会用到的标签是 ， <p> ， ，开发者在写一个页面的时候可以根据这些基础的标签组合出不一样的组件，例如日历、弹窗等。换个思路，既然大家都需要这些组件，为什么我们不能把这些常用的组件包装起来，大大提高我们的开发效率。

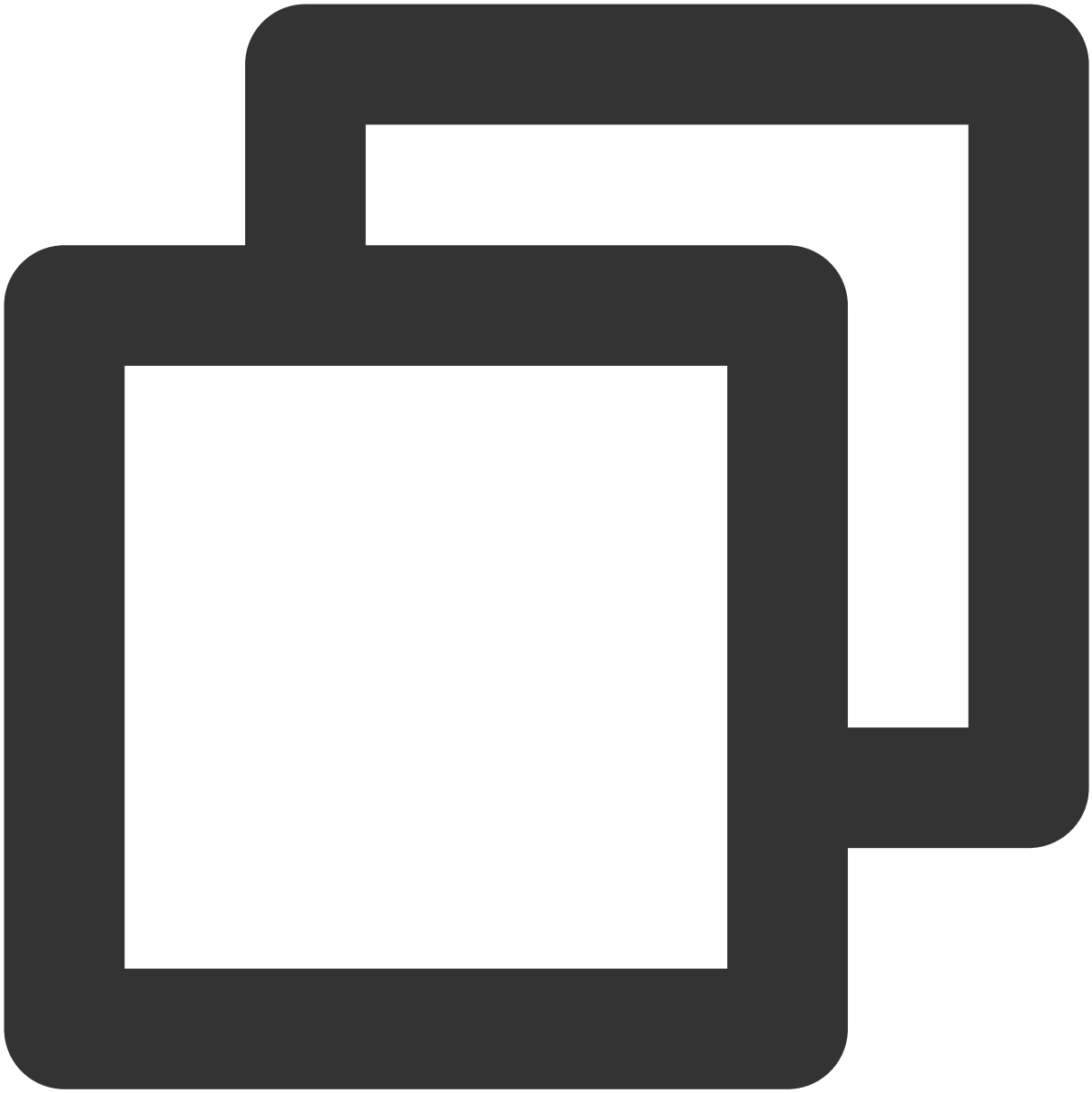
从上边的例子可以看到，小程序的 WXML 用的标签是 view ， button ， text 等，这些标签就是小程序给开发者包装好的基本能力，我们还提供了地图、视频、音频等等组件能力。

2. 多了一些 wx:if 这样的属性以及 {{ }} 这样的表达式

在打开网页的流程中，我们通常会通过 JS 操作 DOM (对应 HTML 的描述产生的树)，以引起界面的一些变化响应用户的行为。例如，用户单击某个按钮的时候， JS 会记录一些状态到 JS 变量里边，同时通过 DOM API 操控 DOM 的属性或者行为，进而引起界面一些变化。当项目越来越大的时候，您的代码会充斥着非常多的界面交互逻辑和程序的各种状态变量，显然这不是一个很好的开发模式，因此就有了 MVVM 的开发模式（例如 React ， Vue ），提倡把渲染和逻辑分离。简单来说就是不要再让 JS 直接操控 DOM ， JS 只需要管理状态即可，然后再通过一种模板语法来描述状态和界面结构的关系即可。

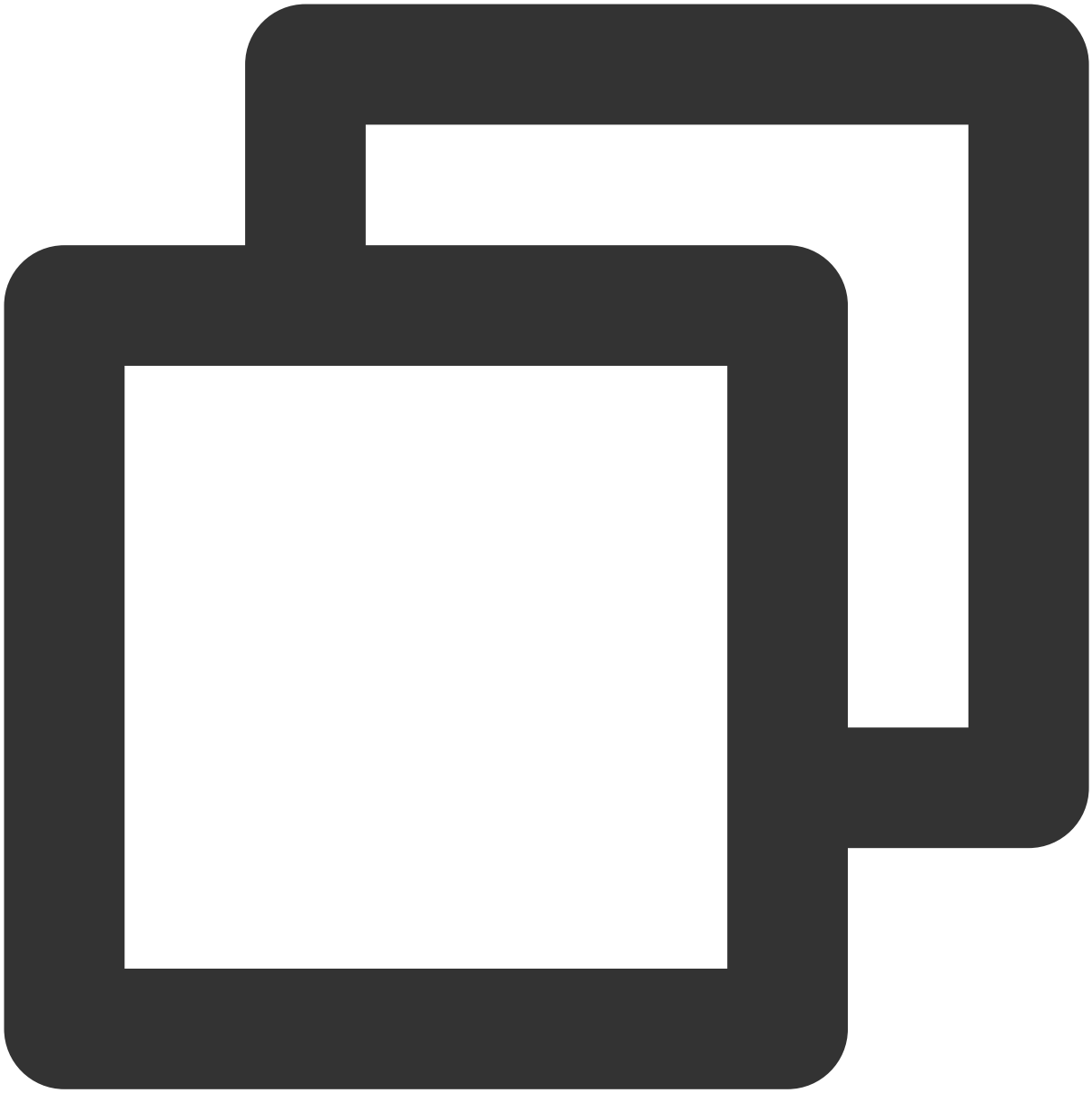
小程序的框架也是用到了这个思路，如果您需要把一个 Hello World 的字符串显示在界面上。

WXML 需要这么写：



```
<text>{{msg}}</text>
```

`JS` 只需要管理状态即可：



```
this.setData({ msg: "Hello World" })
```

通过 {{ }} 的语法把一个变量绑定到界面上，我们称为数据绑定。仅仅通过数据绑定还不够完整的描述状态和界面的关系，还需要 `if` / `else` , `for` 等控制能力，在小程序里边，这些控制能力都用 `wx:` 开头的属性来表达。

更详细的文档可以参考 WXML 语法参考。

3. WXSS样式

`WXSS` 具有 `CSS` 大部分的特性，小程序在 `WXSS` 也做了一些扩充和修改。

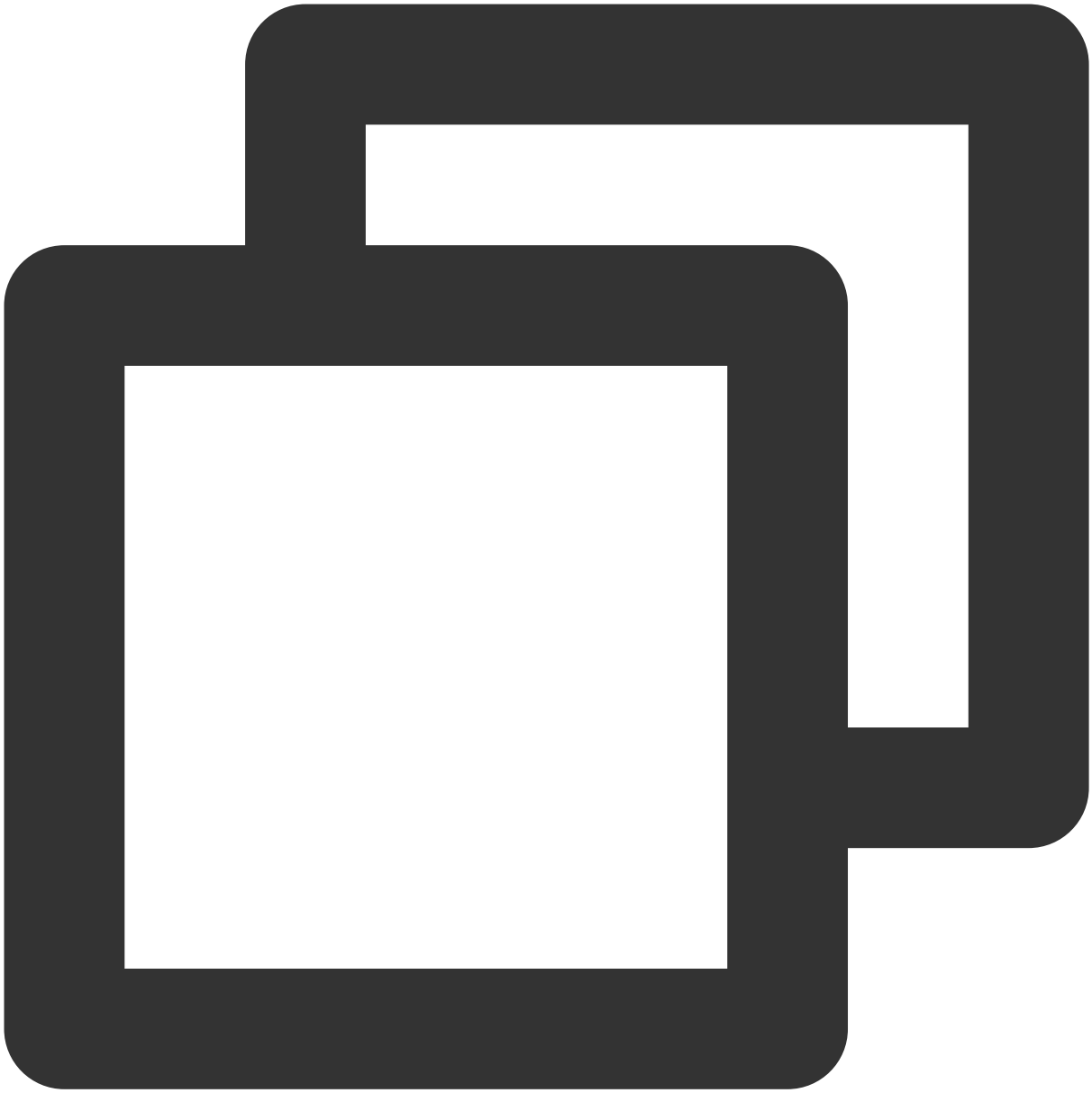
1. 新增了尺寸单位。在写 `CSS` 样式时，开发者需要考虑到手机设备的屏幕会有不同的宽度和设备像素比，采用一些技巧来换算一些像素单位。 `WXSS` 在底层支持新的尺寸单位 `rpx` ，开发者可以免去换算的烦恼，只要交给小程序底层来换算即可，由于换算采用的浮点数运算，所以运算结果会和预期结果有一点点偏差。
2. 提供了全局的样式和局部样式。和前边 `app.json` , `page.json` 的概念相同，您可以写一个 `app.wxss` 作为全局样式，会作用于当前小程序的所有页面，局部页面样式 `page.wxss` 仅对当前页面生效。
3. 此外 `WXSS` 仅支持部分 `CSS` 选择器。

> 更详细的文档可以参考 渲染层-WXSS 。

4. JS 逻辑交互

一个服务仅仅只有界面展示是不够的，还需要和用户做交互：响应用户的单击、获取用户的位置等等。在小程序里边，我们就通过编写 `JS` 脚本文件来处理用户的操作。

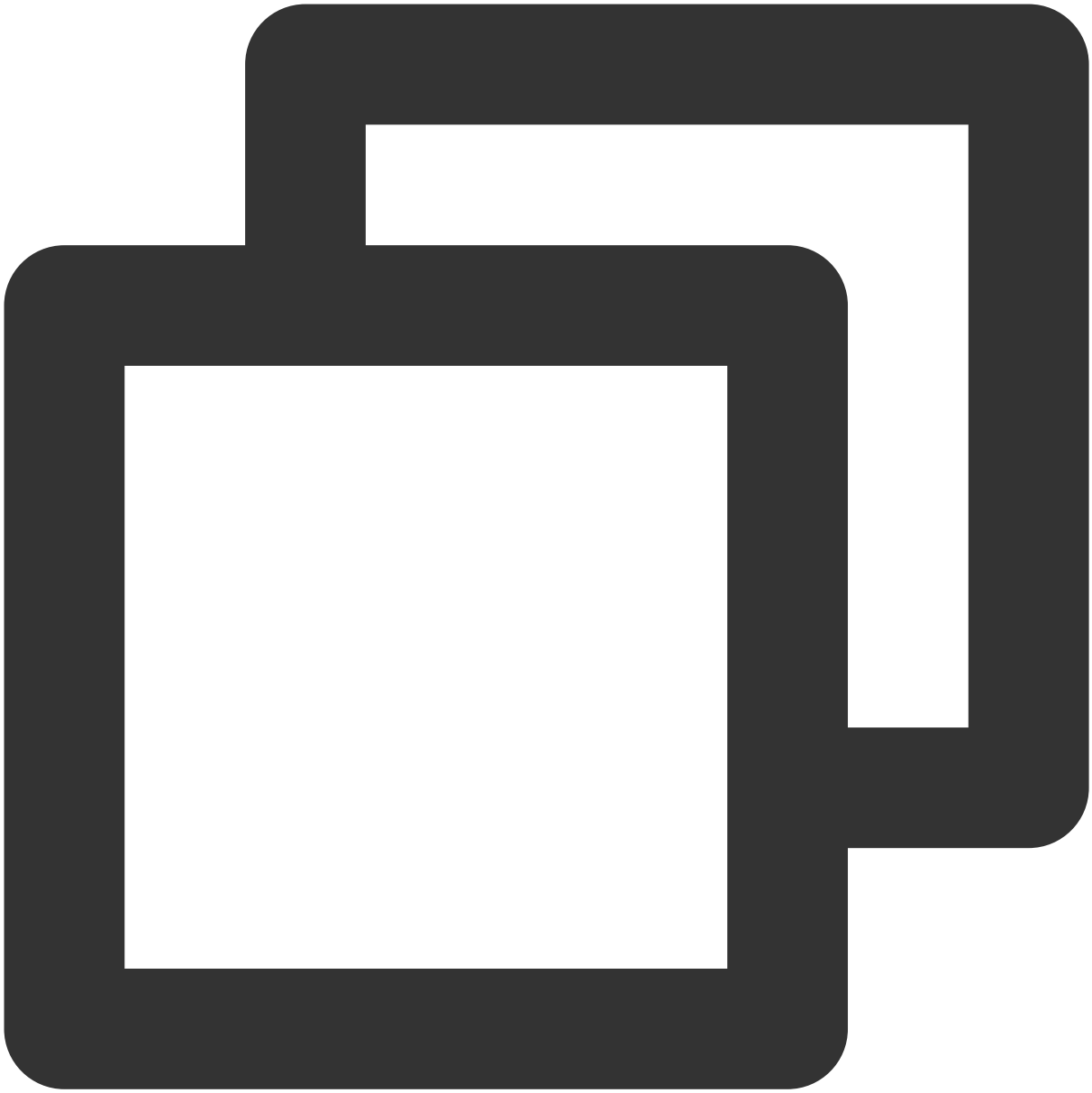
版权所有：腾讯云计算（北京）有限责任公司



```
<view>{{ msg }}</view>
<button bindtap="clickMe">点击我</button>
```

单击 `button` 按钮的时候，我们希望把界面上 `msg` 显示成 `"Hello World"`，于是我们在 `button` 上声明一个属性: `bindtap` ，在 JS 文件里边声明了 `clickMe` 方法来响应这次点击操作：

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  clickMe: function() {
    this.setData({ msg: "Hello World" })
  }
})
```

响应用户的操作就是这么简单，更详细的事件可以参考文档 渲染层-事件。

小程序目录结构

最近更新时间：2024-07-12 19:03:25

小程序包含一个描述整体程序的 `app` 和多个描述各自页面的 `page` 。

1. 一个小程序主体部分由三个文件组成，必须放在项目的根目录，如下：

文件	必需	作用
app.js	是	小程序逻辑
app.json	是	小程序公共配置
app.wxss	否	小程序公共样式

2. 一个小程序页面由四个文件组成，分别是：

文件类型	必需	作用
js	是	页面逻辑
wxml	是	页面结构
json	否	页面配置
wxss	否	页面样式表

注意：

为了方便开发者减少配置项，描述页面的四个文件必须具有相同的路径与文件名。

允许被上传的文件

在项目目录中，以下文件会经过编译，因此上传之后无法直接访问到：.js、app.json、.wxml、*.wxss（其中 wxml 和 wxss 文件仅针对在 app.json 中配置了的页面）。除此之外，只有后缀名在白名单内的文件可以被上传，不在白名单列表内文件在开发工具能被访问到，但无法被上传。具体白名单列表如下：

1. wxs
2. png
3. jpg
4. jpeg
5. gif
6. svg
7. json
8. cer

版权所有：腾讯云计算（北京）有限责任公司

9. mp3
10. aac
11. m4a
12. mp4
13. wav
14. ogg
15. silk
16. wasm
17. br
18. cert

版权所有：腾讯云计算（北京）有限责任公司

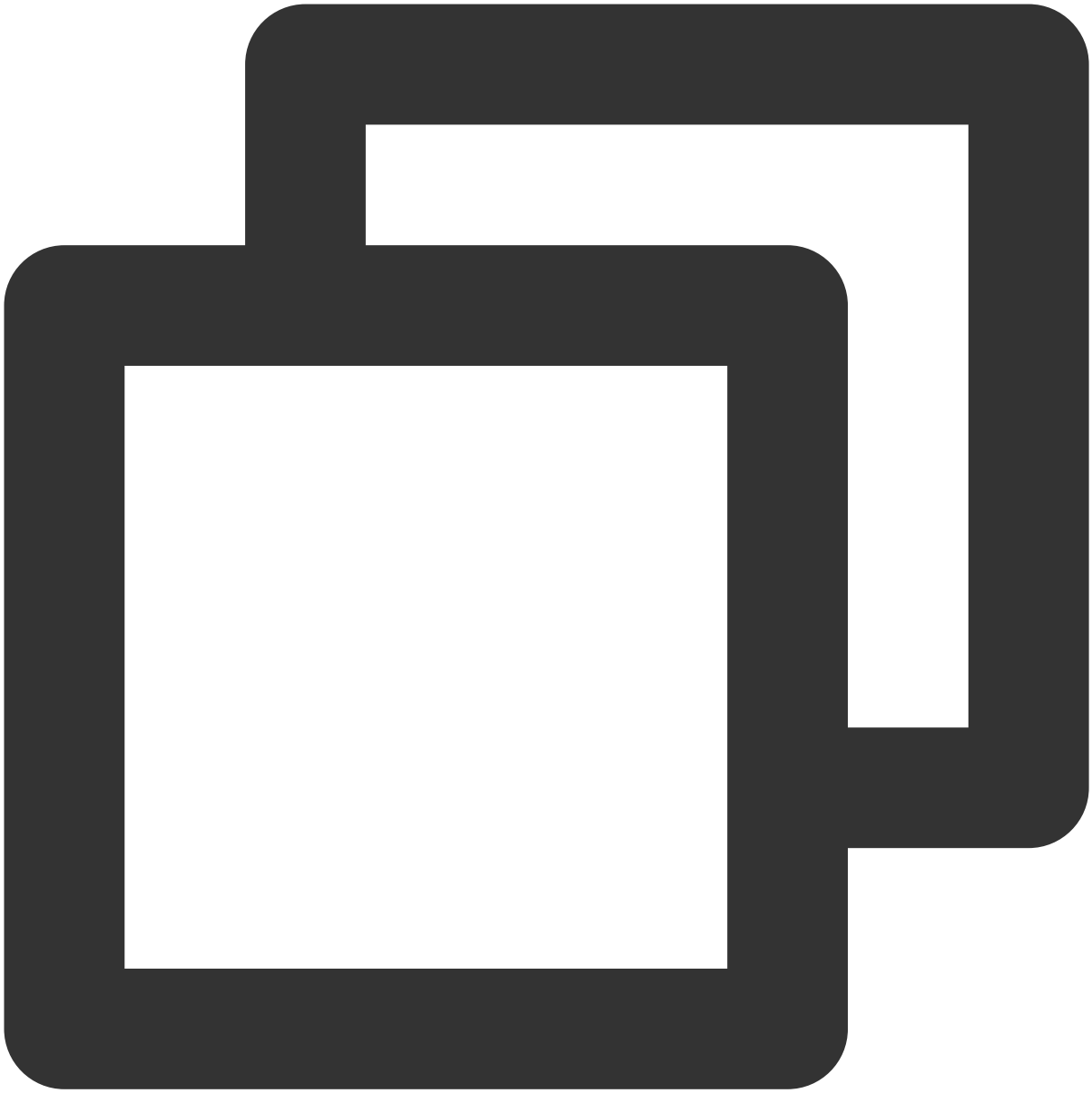
配置小程序
全局与页面配置

最近更新时间：2024-07-12 17:51:28

1. 全局配置

小程序根目录下的 `app.json` 文件用来对微信小程序进行全局配置，决定页面文件的路径、窗口表现、设置网络超时时间、设置多 tab 等。
以下是一个包含了部分常用配置选项的 `app.json` ：

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "pages": [
    "pages/index/index",
    "pages/logs/index"
  ],
  "window": {
    "navigationBarTitleText": "Demo"
  },
  "tabBar": {
    "list": [{
      "pagePath": "pages/index/index",
```

版权所有：腾讯云计算（北京）有限责任公司

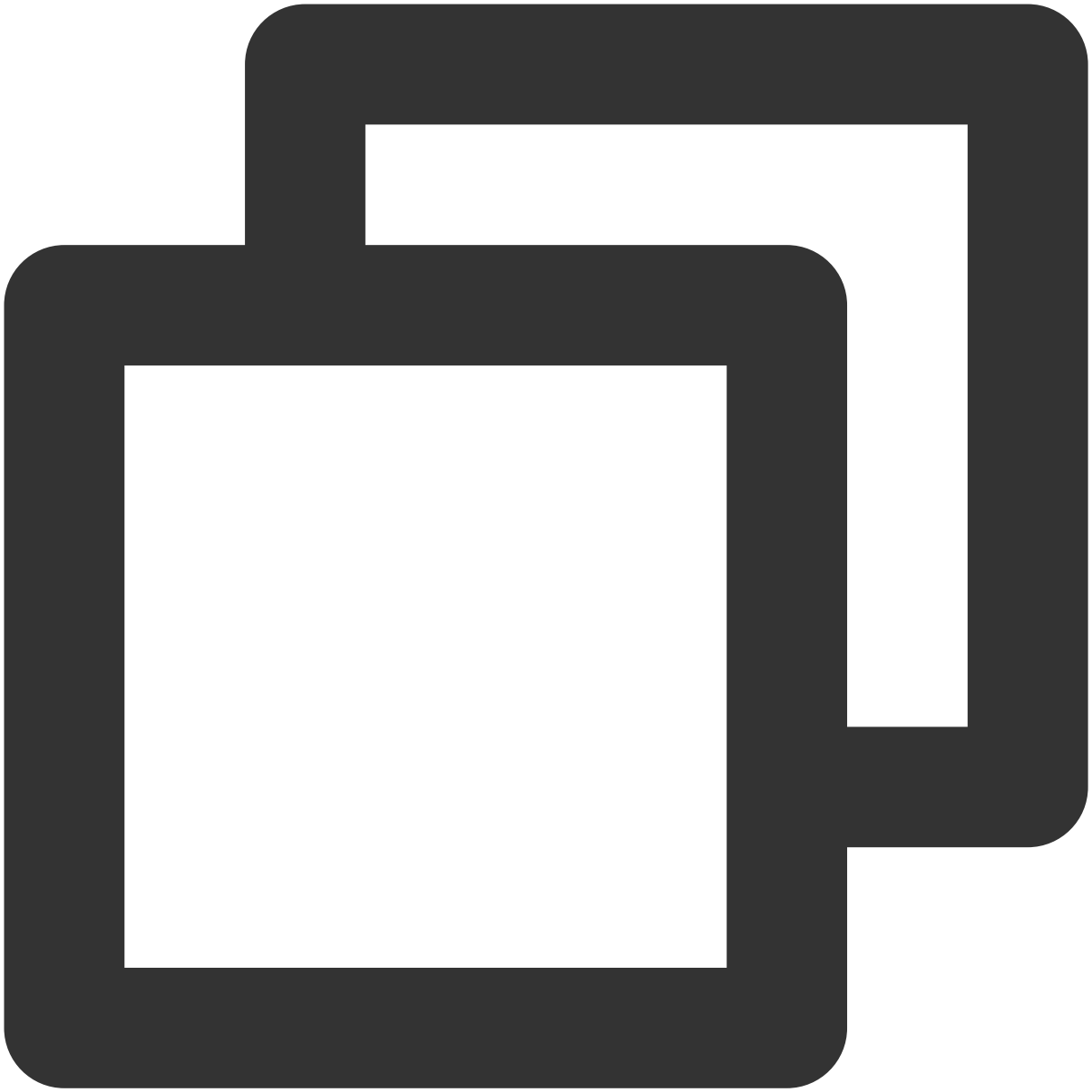
```
      "text": "首页"
    }, {
      "pagePath": "pages/logs/index",
      "text": "日志"
    }]
  },
  "debug": true
}
```

完整配置项说明请参考 小程序全局配置。

2. 页面配置

每一个小程序页面也可以使用同名 .json 文件来对本页面的窗口表现进行配置，页面中配置项会覆盖 `app.json` 的 `window` 中相同的配置项。

例如：



```
{
  "navigationBarBackgroundColor": "#ffffff",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "title",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
}
```

完整配置项说明请参考小程序页面配置。

版权所有：腾讯云计算（北京）有限责任公司

小程序框架
介绍

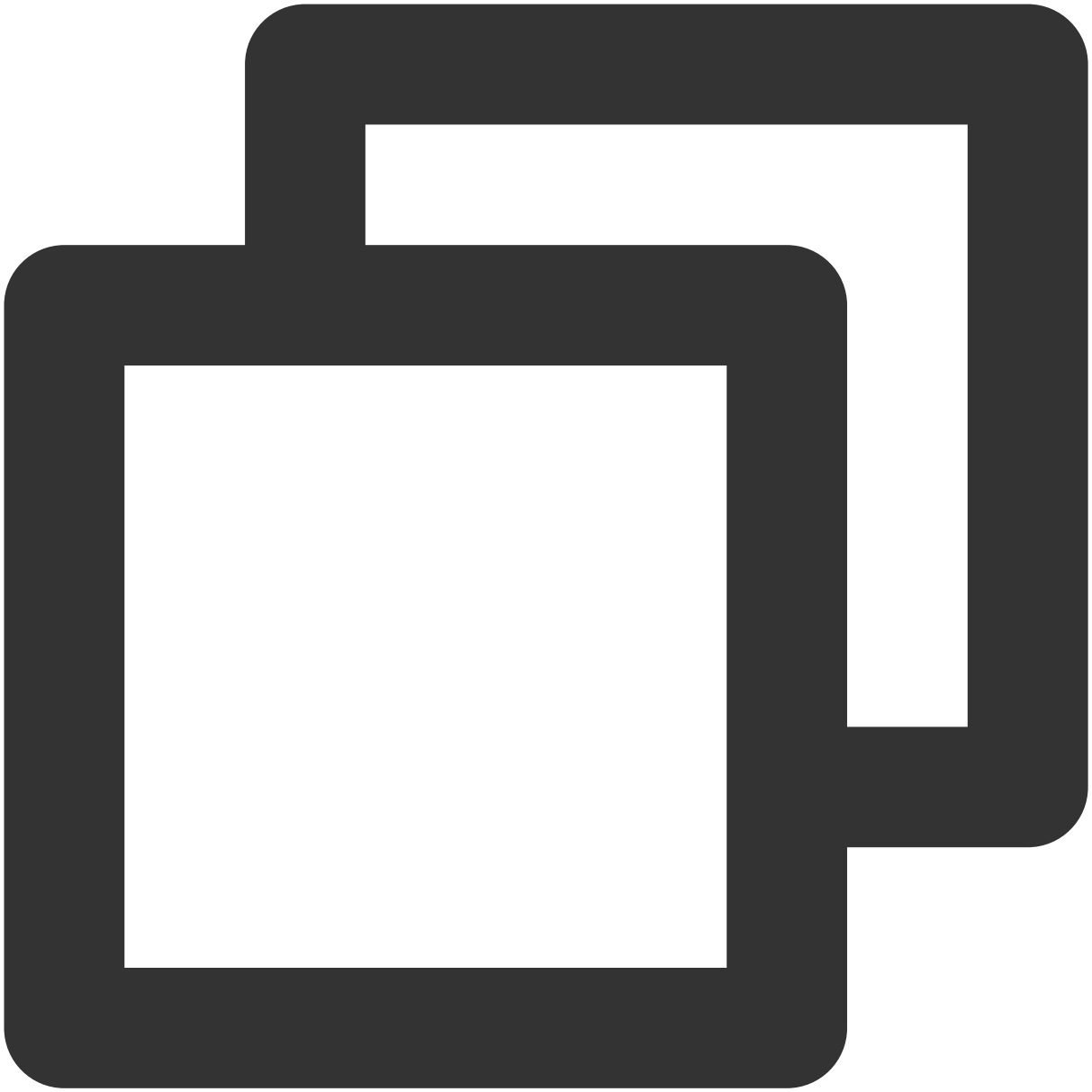
最近更新时间：2024-07-12 17:48:50

小程序开发框架的目标是通过尽可能简单、高效的方式让开发者可以在客户端中开发具有原生 APP 体验的服务。整个小程序框架系统分为两部分：逻辑层（App Service）和渲染层（View）。小程序提供了自己的渲染层描述语言 `WXML` 和 `WXSS`，以及基于 `JavaScript` 的逻辑层框架，并在渲染层与逻辑层间提供了数据传输和事件系统，让开发者能够专注于数据与逻辑。

响应的数据绑定

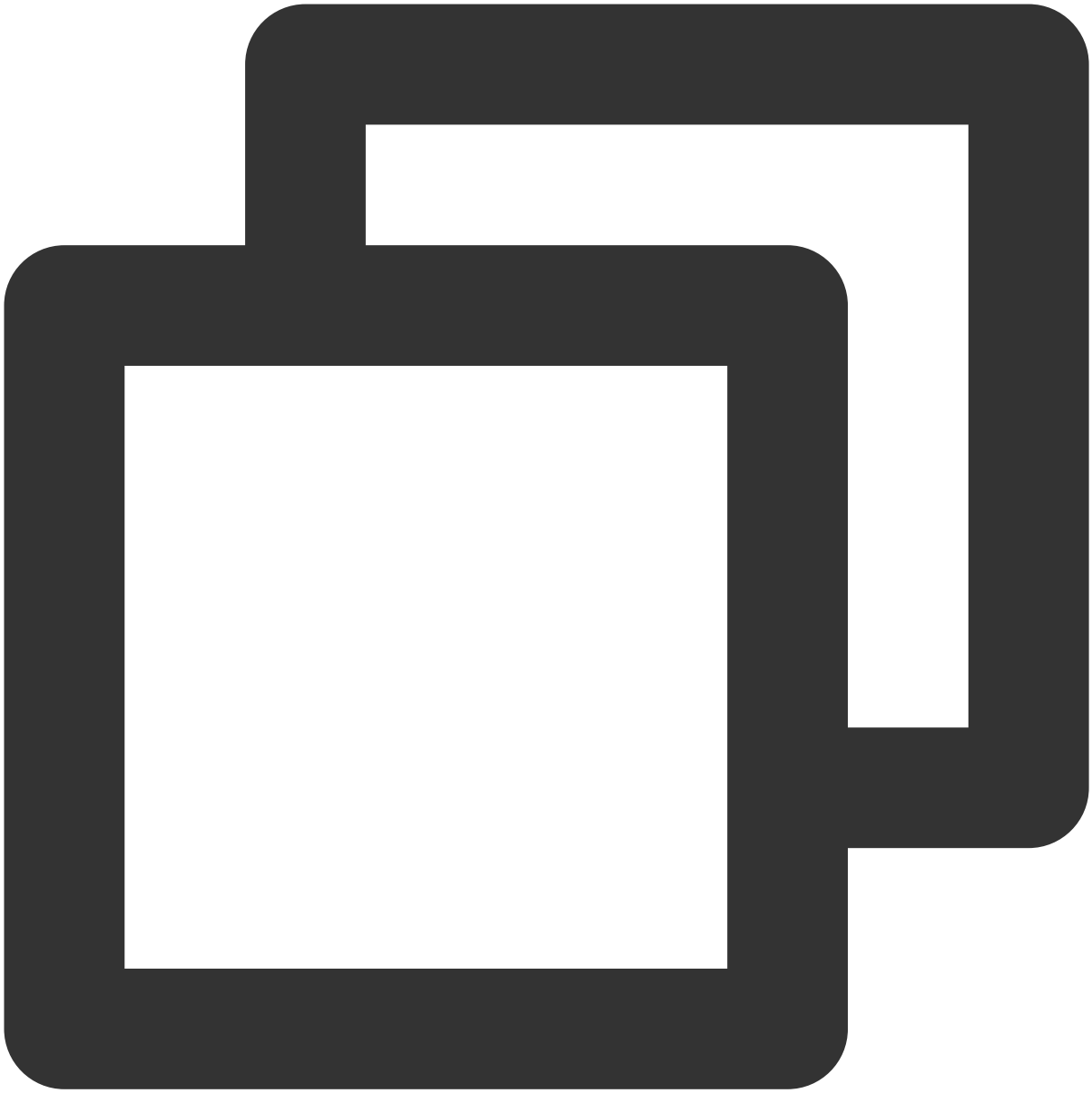
框架的核心是一个响应的数据绑定系统，可以让数据与视图非常简单地保持同步。当做数据修改的时候，只需要在逻辑层修改数据，渲染层就会做相应的更新。
通过这个简单的例子来看：

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- This is our View -->
<view> Hello {{name}}! </view>
<button bindtap="changeName"> Click me! </button>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// This is our App Service.
// This is our data.
var helloData = {
  name: 'TCMPP'
}

// Register a Page.
Page({
  data: helloData,
  changeName: function(e) {
    // sent data change to view
```

```
    this.setData({
      name: 'World'
    })
  }
})
```

开发者通过框架将逻辑层数据中的 `name` 与视图层的 `name` 进行了绑定，所以在页面一打开的时候会显示 `Hello TCMPP!`

当单击按钮的时候，视图层会发送 `changeName` 的事件给逻辑层，逻辑层找到并执行对应的事件处理函数；

回调函数触发后，逻辑层执行 `setData` 的操作，将 `data` 中的 `name` 从 `TCMPP` 变为 `World`，因为该数据和渲染层已经绑定了，从而渲染层会自动改变为 `Hello World!`

页面管理

`框架` 管理了**整个小程序**的页面路由，可以做到页面间的无缝切换，并给以页面完整的生命周期。开发者需要做的只是将页面的数据、方法、生命周期函数注册到 `框架` 中，其他的一切复杂的操作都交由 `框架` 处理。

基础组件

`框架` 提供了一套基础的组件，这些组件自带统一风格的样式以及特殊的逻辑，开发者可以通过组合基础组件，创建出强大的 TCMPP 小程序。

丰富的API

`框架` 提供丰富的原生 API，可以方便的调起小程序 SDK 提供的能力，如获取地理位置信息，本地存储等。

版权所有：腾讯云计算（北京）有限责任公司

逻辑层

最近更新时间：2024-07-12 17:48:22

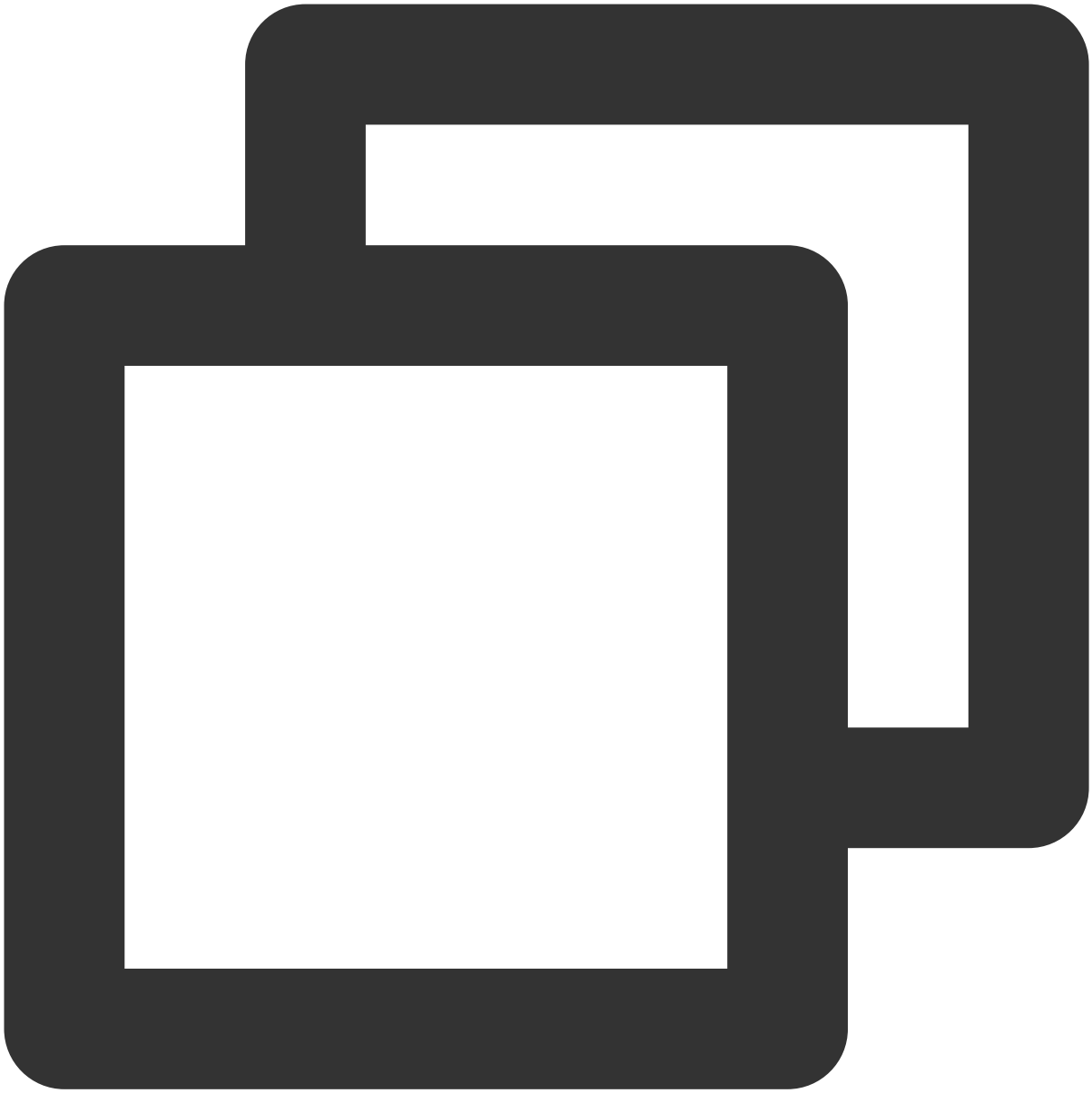
1. 介绍

小程序开发框架的逻辑层使用 `JavaScript` 引擎为小程序提供开发 `JavaScript` 代码的运行环境以及 TCMPP 小程序的特有功能。
逻辑层将数据进行处理后发送给视图层，同时接受视图层的事件反馈。
开发者写的所有代码最终将会打包成一份 `JavaScript` 文件，并在小程序启动的时候运行，直到小程序销毁。这一行为类似 SeviceWorker，所以逻辑层也称之为 `App Service` 。
在 `JavaScript` 的基础上，我们增加了一些功能，以方便小程序的开发：
增加 `App` 和 `Page` 方法，进行 程序注册 和 页面注册。
增加 `getApp` 和 `getCurrentPages` 方法，分别用来获取 `App` 实例和当前页面栈。
提供丰富的 API，如获取地理位置，扫一扫等特有能力。
提供 模块化 能力，每个页面有独立的作用域。
注意：
小程序框架的逻辑层并非运行在浏览器中，因此 JavaScript 在 web 中一些能力都无法使用，如 window，document 等。

2. 注册小程序

每个小程序都需要在 `app.js` 中调用 `App(Object)` 方法注册小程序实例，绑定生命周期回调函数、错误监听和页面不存在监听函数等。
详细的参数含义和使用请参考 App。

版权所有：腾讯云计算（北京）有限责任公司

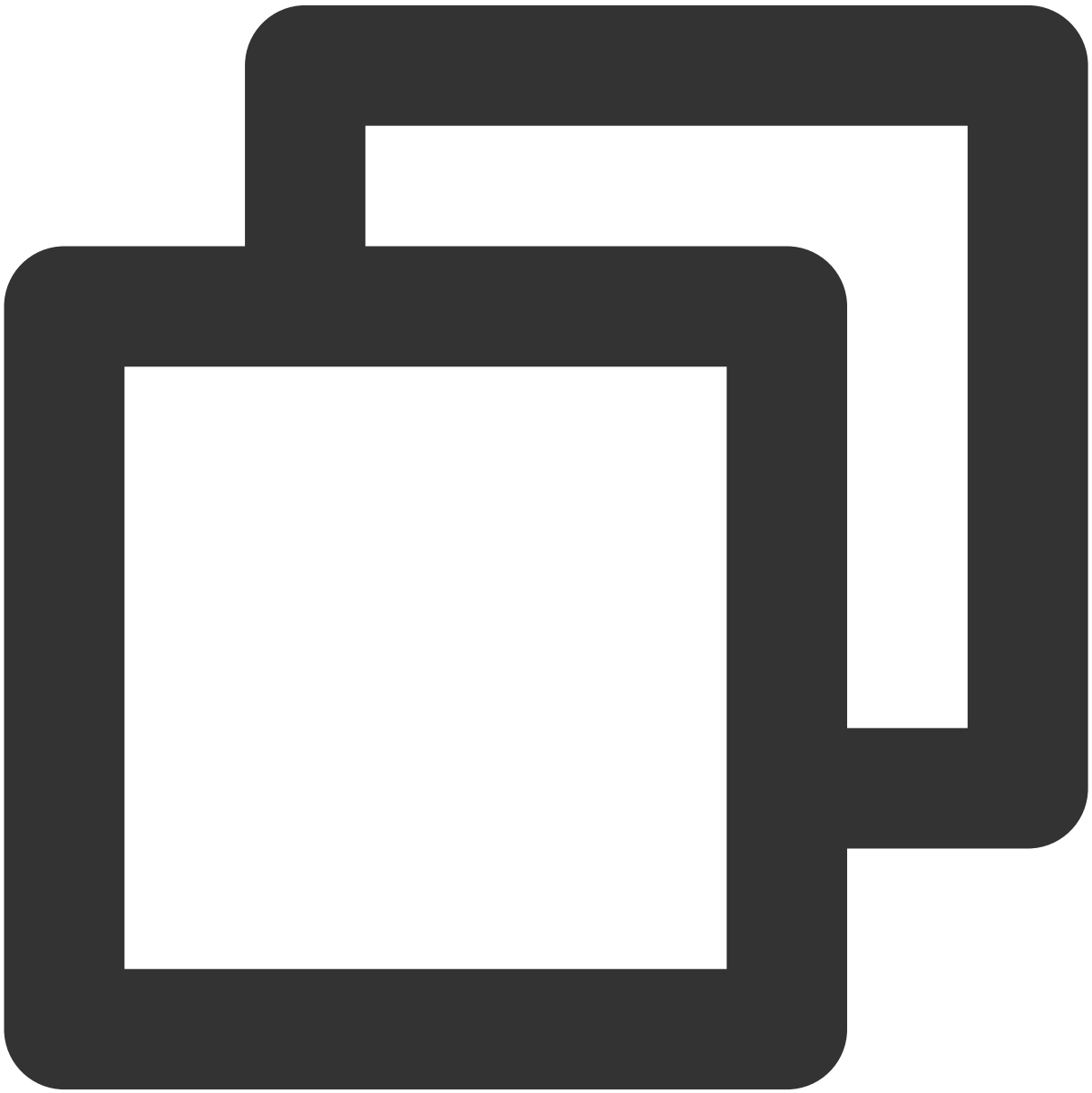


```
// app.js
App({
  onLaunch (options) {
    // Do something initial when launch.
  },
  onShow (options) {
    // Do something when show.
  },
  onHide () {
    // Do something when hide.
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  onError (msg) {
    console.log(msg)
  },
  globalData: 'I am global data'
})
```

整个小程序只有一个 App 实例，是全部页面共享的。开发者可以通过 `getApp` 方法获取到全局唯一的 App 实例，获取 App 上的数据或调用开发者注册在 App 上的函数。



```
// xxx.js
const appInstance = getApp()
```

```
console.log(appInstance.globalData) // I am global data
```

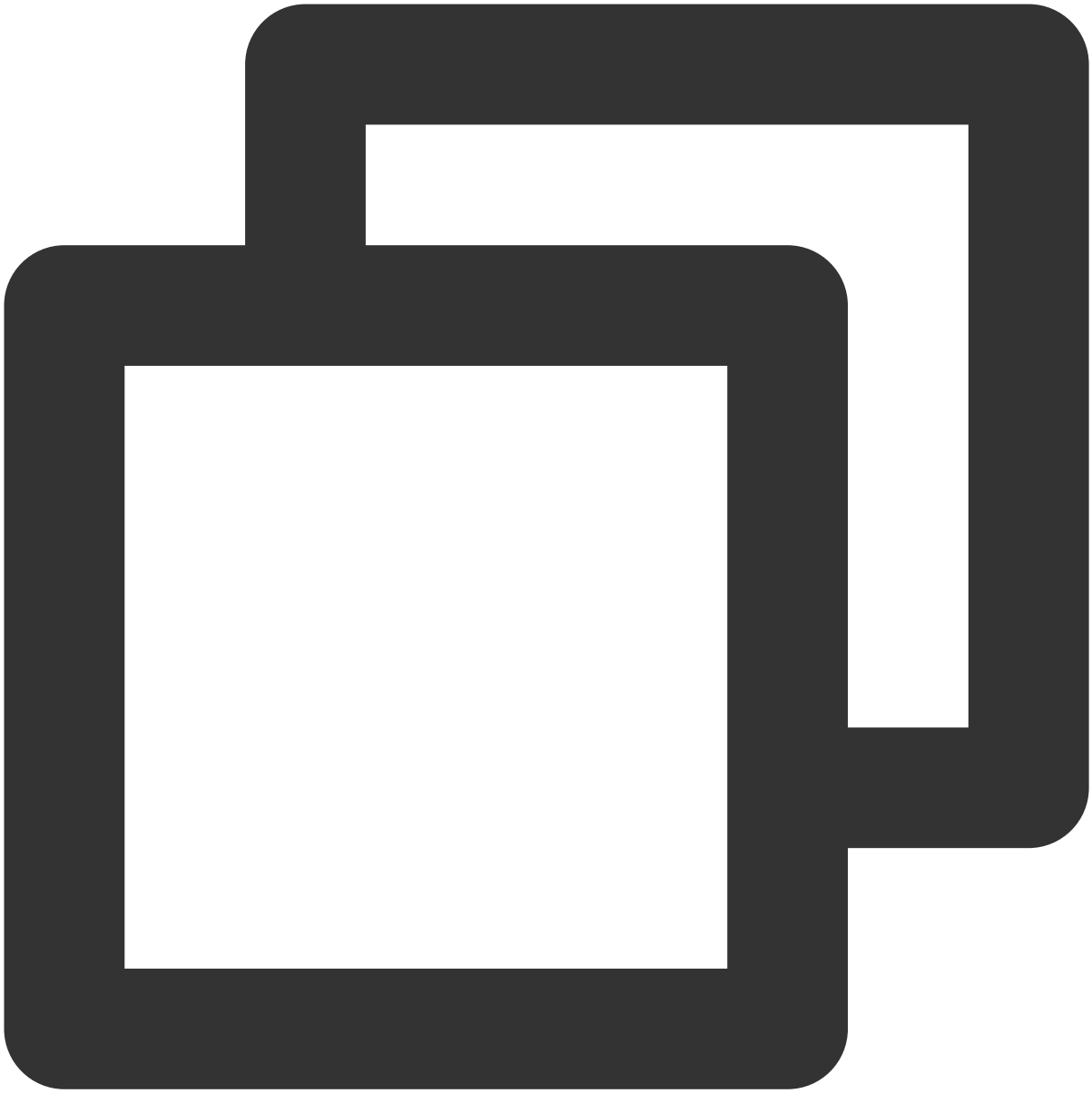
3. 注册页面

对于小程序中的每个页面，都需要在页面对应的 js 文件中进行注册，指定页面的初始数据、生命周期回调、事件处理函数等。

3.1 使用 Page 构造器注册页面

简单的页面可以使用 `Page(Object)` 进行构造。

代码示例：



```
//index.js
Page({
  data: {
    text: "This is page data."
  },
  onLoad: function(options) {
    // 页面创建时执行
  },
  onShow: function() {
    // 页面出现在前台时执行
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  onReady: function() {
    // 页面首次渲染完毕时执行
  },
  onHide: function() {
    // 页面从前台变为后台时执行
  },
  onUnload: function() {
    // 页面销毁时执行
  },
  onPullDownRefresh: function() {
    // 触发下拉刷新时执行
  },
  onReachBottom: function() {
    // 页面触底时执行
  },
  onShareAppMessage: function () {
    // 页面被用户分享时执行
  },
  onPageScroll: function() {
    // 页面滚动时执行
  },
  onResize: function() {
    // 页面尺寸变化时执行
  },
  onTabItemTap(item) {
    // tab 点击时执行
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  },
  // 事件响应函数
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    }, function() {
      // this is setData callback
    })
  },
  // 自由数据
  customData: {
    hi: 'MINA'
  }
})
```

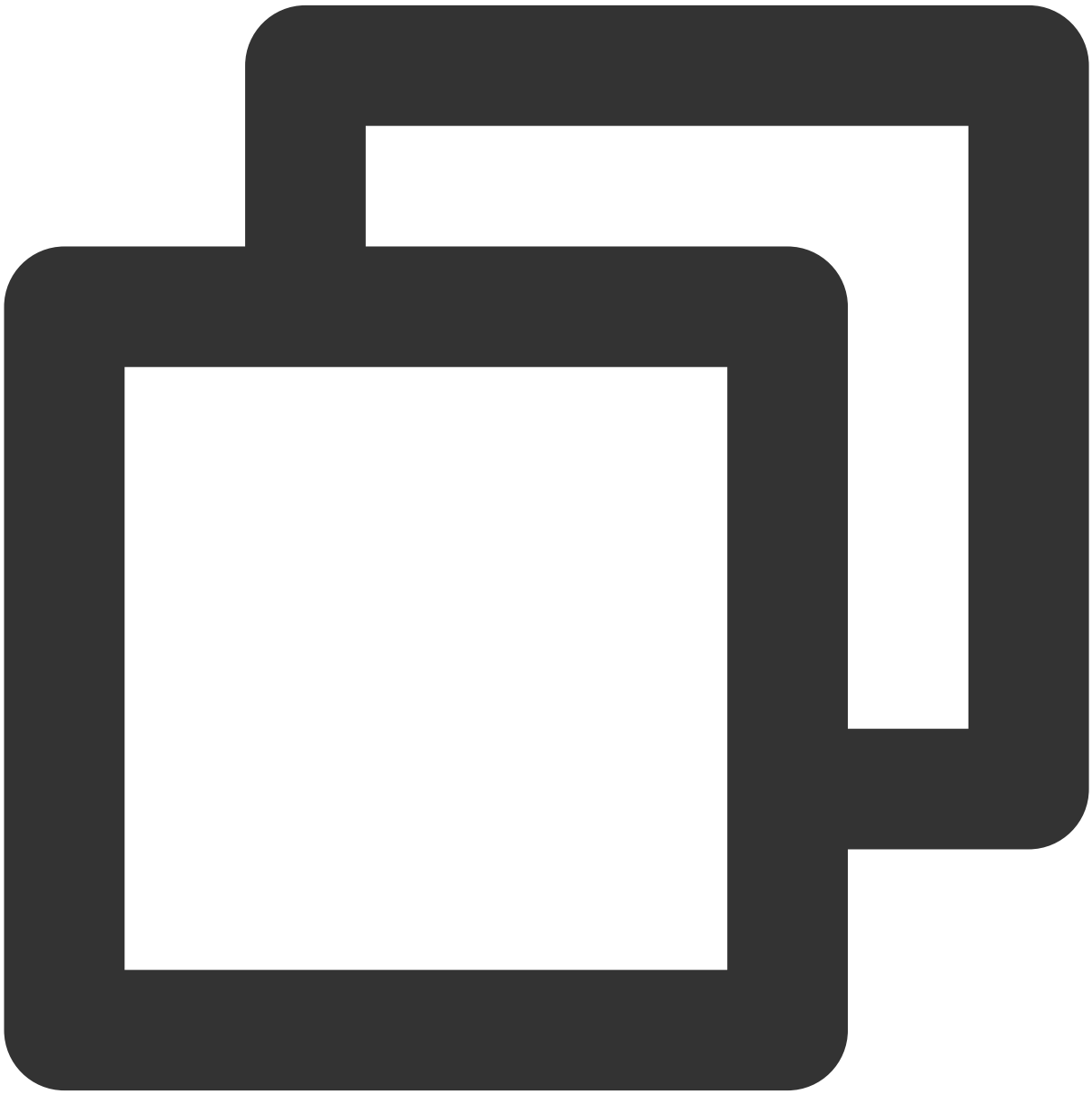
详细的参数含义和使用请参考 页面。

3.2 使用 Component 构造器构造页面

`Page` 构造器适用于简单的页面。但对于复杂的页面，`Page` 构造器可能并不好用。

此时，可以使用 `Component` 构造器来构造页面。`Component` 构造器的主要区别是：方法需要放在 `methods: { }` 里面。

代码示例：



```
Component({
  data: {
```

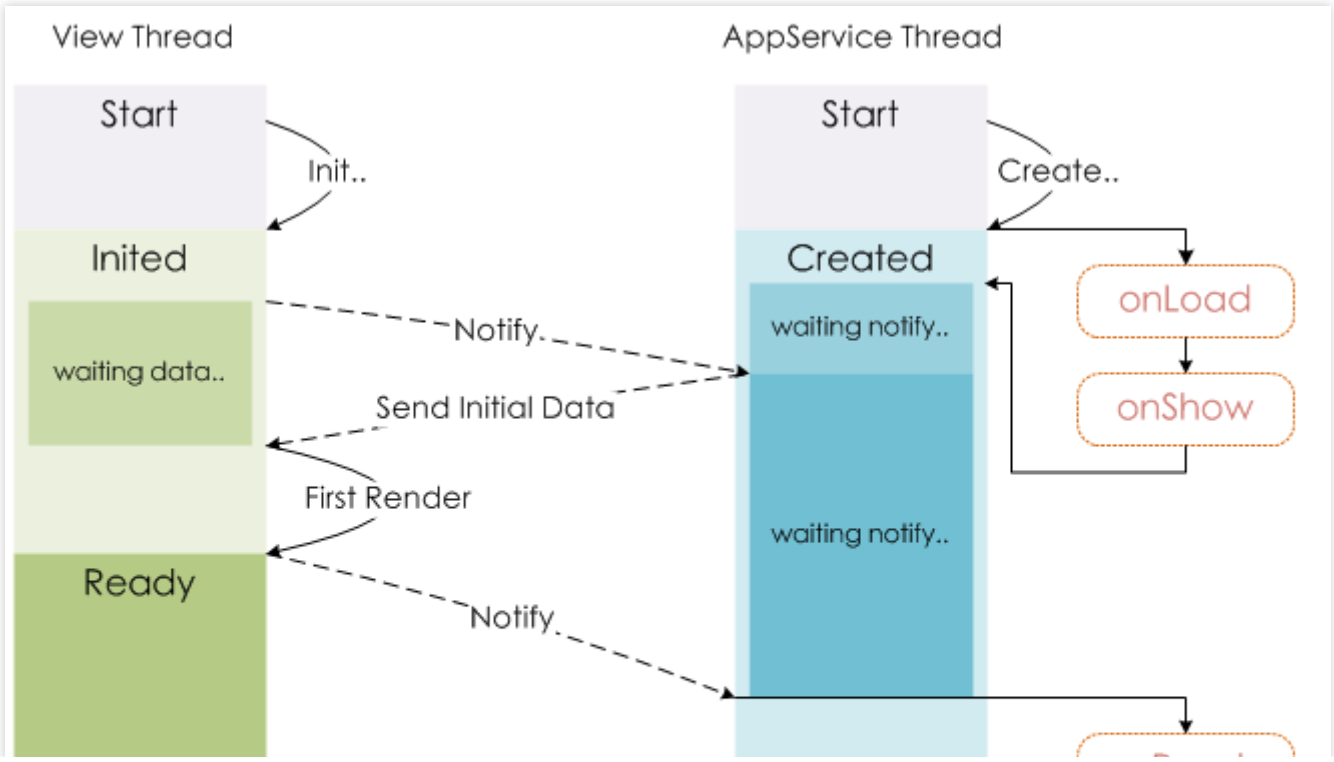
```
    text: "This is page data."
  },
  methods: {
    onLoad: function(options) {
      // 页面创建时执行
    },
    onPullDownRefresh: function() {
      // 下拉刷新时执行
    },
    // 事件响应函数
    viewTap: function() {
      // ...
    }
  }
})
```

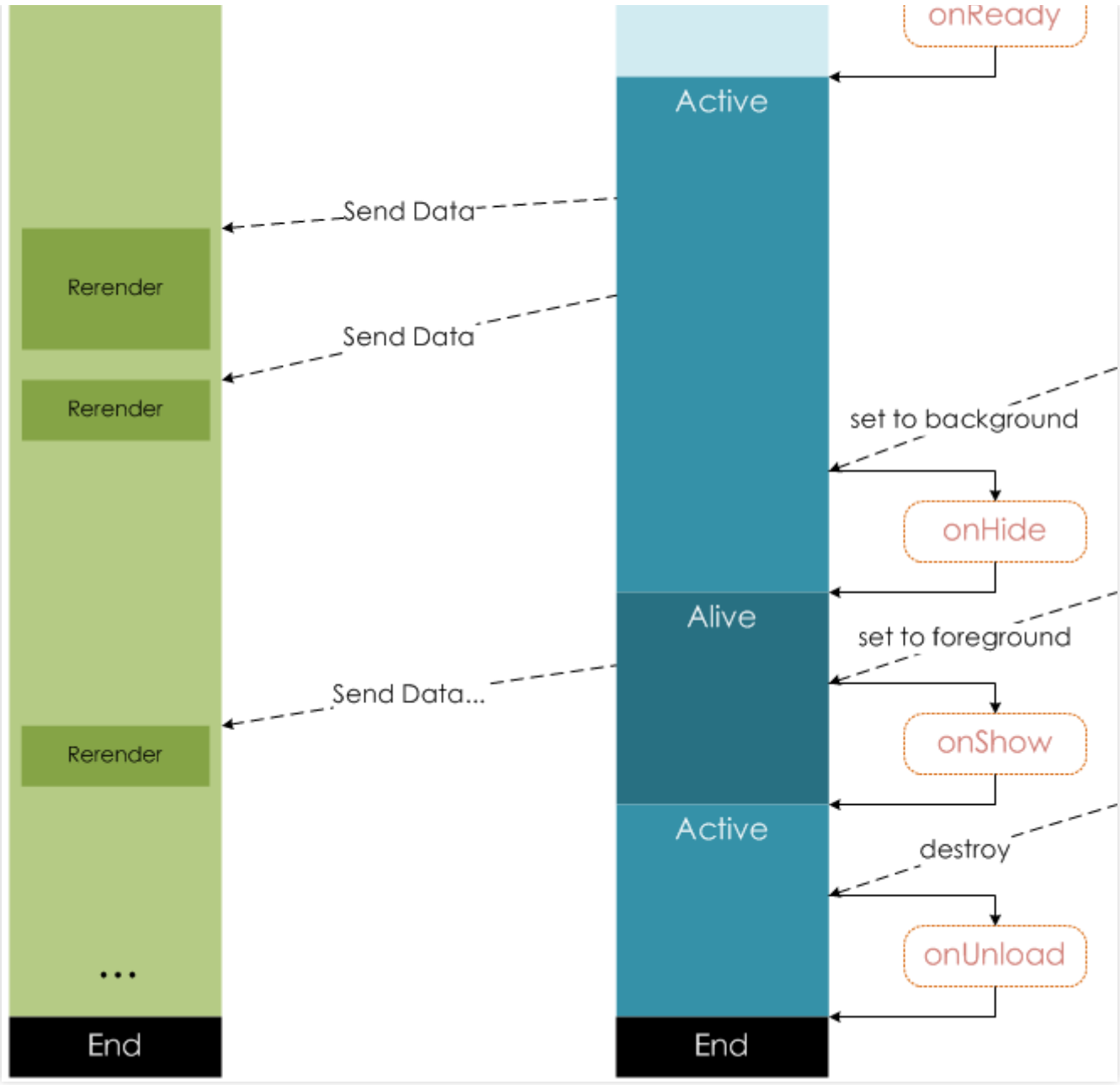
这种创建方式非常类似于 自定义组件，可以像自定义组件一样使用 `behaviors` 等高级特性。

> 具体细节请阅读 Component 构造器 章节。

4. 页面生命周期

下图说明了页面 Page 实例的生命周期：





5. 页面路由

在小程序中所有页面的路由全部由 `框架` 进行管理。

5.1 页面栈

框架以栈的形式维护了当前的所有页面。 当发生路由切换的时候，页面栈的表现如下：

路由方式	页面栈表现
初始化	新页面入栈
打开新页面	新页面入栈
页面重定向	当前页面入栈，新页面入栈

版权所有：腾讯云计算（北京）有限责任公司

页面返回	页面不断出栈，直到目标返回页
Tab切换	页面全部出栈，只留下新的Tab页面
重加载	页面全部出栈，只留下新的页面

开发者可以使用 getCurrentPages() 函数获取当前页面栈，以数组形式按栈的顺序给出，第一个元素为首页，最后一个元素为当前页面。

注意：

不要尝试修改页面栈，会导致路由以及页面状态错误。

不要在 `App.onLaunch` 的时候调用 `getCurrentPages()`，此时 `page` 还没有生成。

5.2 路由方式

对于路由的触发方式以及页面生命周期函数如下：

路由方式	触发时机	路由前页面	路由后页面
初始化	小程序打开的第一个页面	-	onLoad, onShow
打开新页面	调用 API wx.navigateTo 或使用组件 <navigator open-type="navigateTo"/>	onHide	onLoad, onShow
页面重定向	调用 API wx.redirectTo 或使用组件 <navigator open-type="redirectTo"/>	onUnload	onLoad, onShow
页面返回	调用 API wx.navigateBack 或使用组件<navigator open-type="navigateBack">或用户按左上角返回按钮	onUnload	onShow
Tab 切换	调用 API wx.switchTab 或使用组件 <navigator open-type="switchTab"/> 或用户切换 Tab	-	各种情况请参考下表
重启动	调用 API wx.reLaunch 或使用组件 <navigator open-type="reLaunch"/>	onUnload	onLoad, onShow

Tab 切换对应的生命周期（以 A、B 页面为 `Tabbar` 页面，C 是从 A 页面打开的页面，D 页面是从 C 页面打开的页面为例）：

当前页面	路由后页面	触发的生命周期（按顺序）
A	A	Nothing happend
A	B	A.onHide(), B.onLoad(), B.onShow()

A	B（再次打开）	A.onHide(), B.onShow()
C	A	C.onUnload(), A.onShow()
C	B	C.onUnload(), B.onLoad(), B.onShow()
D	B	D.onUnload(), C.onUnload(), B.onLoad(), B.onShow()
D（从转发进入）	A	D.onUnload(), A.onLoad(), A.onShow()
D（从转发进入）	B	D.onUnload(), B.onLoad(), B.onShow()

注意：

navigateTo, redirectTo 只能打开非 tabBar 页面。

switchTab 只能打开 tabBar 页面。

reLaunch 可以打开任意页面。

页面底部的 tabBar 由页面决定，即只要是定义为 tabBar 的页面，底部都有 tabBar。

调用页面路由带的参数可以在目标页面的 onLoad 中获取。

6. 模块化

6.1 模块化

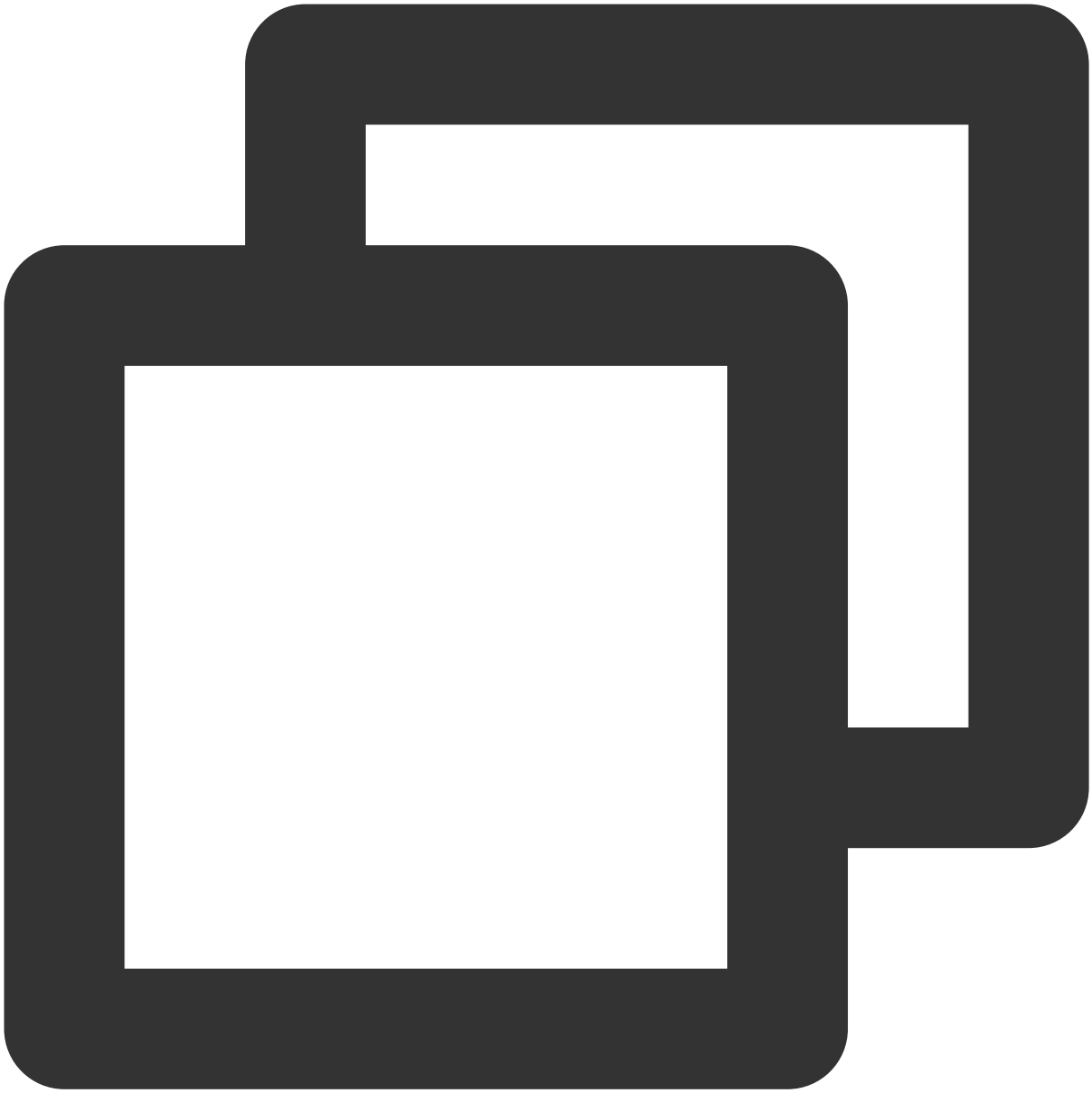
可以将一些公共的代码抽离成为一个单独的 js 文件，作为一个模块。模块只有通过 module.exports 或者 exports 才能对外暴露接口。

注意：

`exports` 是 module.exports 的一个引用，因此在模块里边随意更改 `exports` 的指向会造成未知的错误。所以更推荐开发者采用 module.exports 来暴露模块接口，除非您已经清晰知道这两者的关系。

小程序目前不支持直接引入 `node_modules` ，开发者需要使用到 `node_modules` 时候建议拷贝出相关的代码到小程序的目录中，或者使用小程序支持的 `npm` 功能。

版权所有：腾讯云计算（北京）有限责任公司

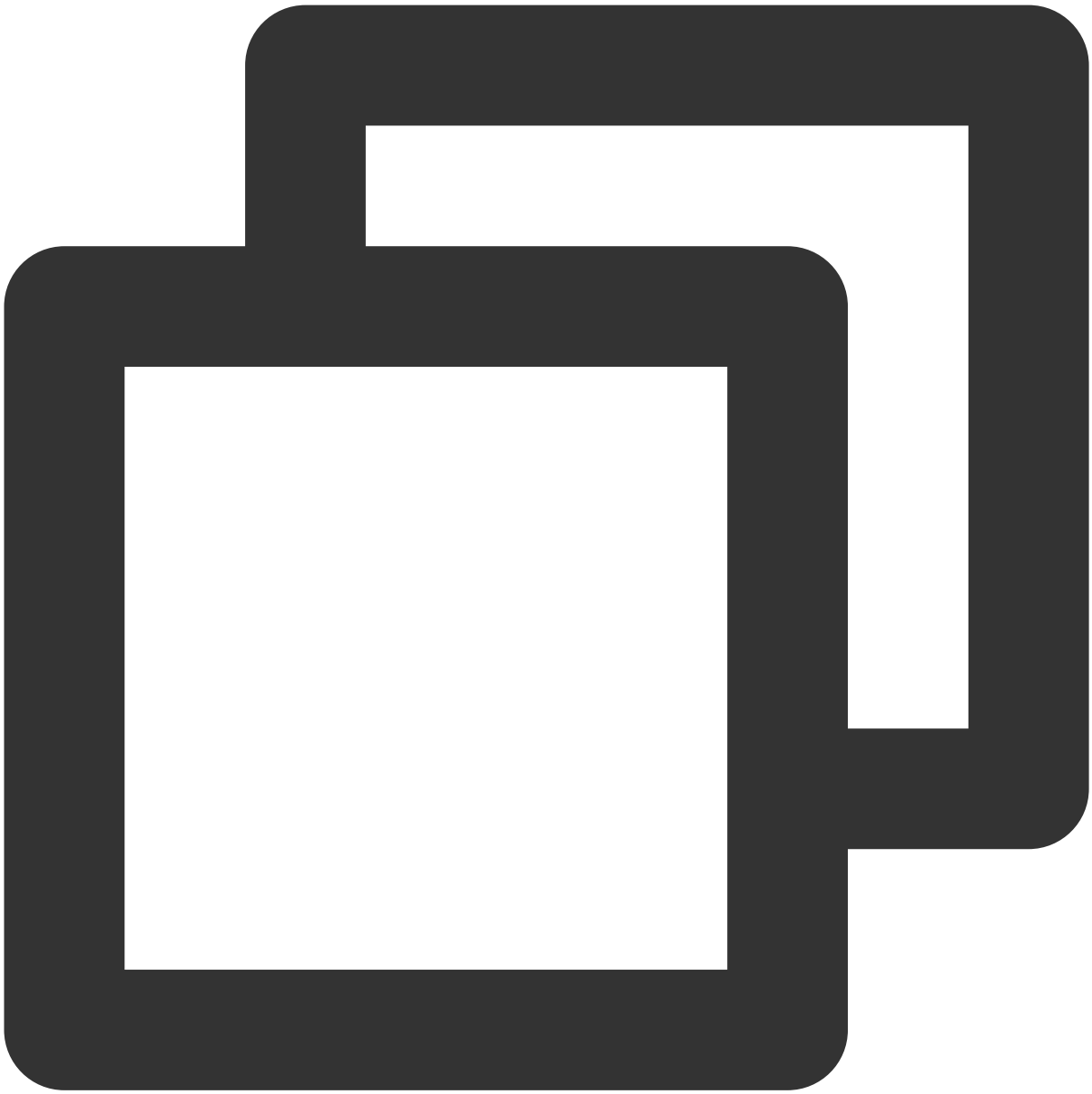


```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}

module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

版权所有：腾讯云计算（北京）有限责任公司

在需要使用这些模块的文件中，使用 `require` 将公共代码引入：

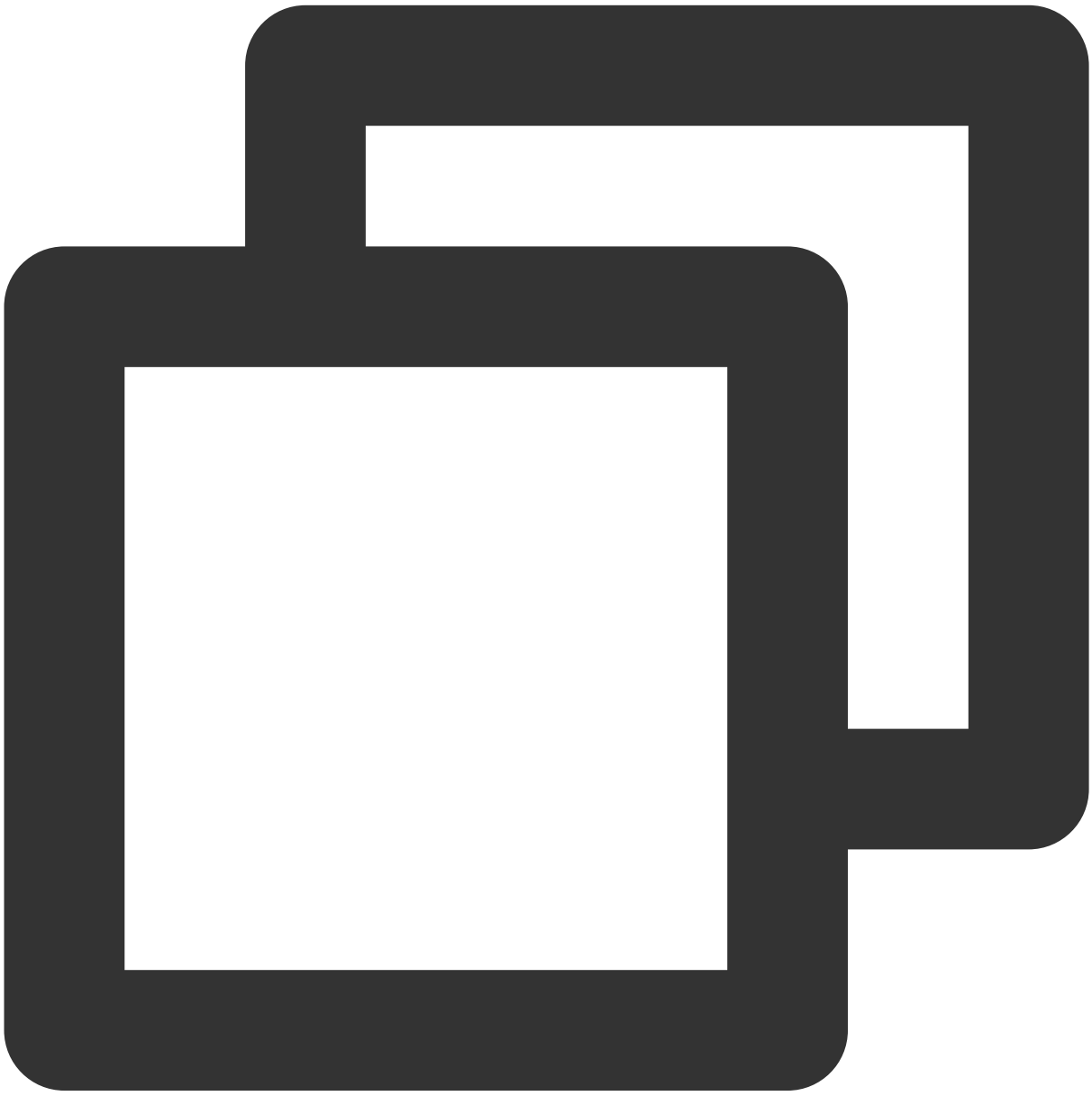


```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
})
```

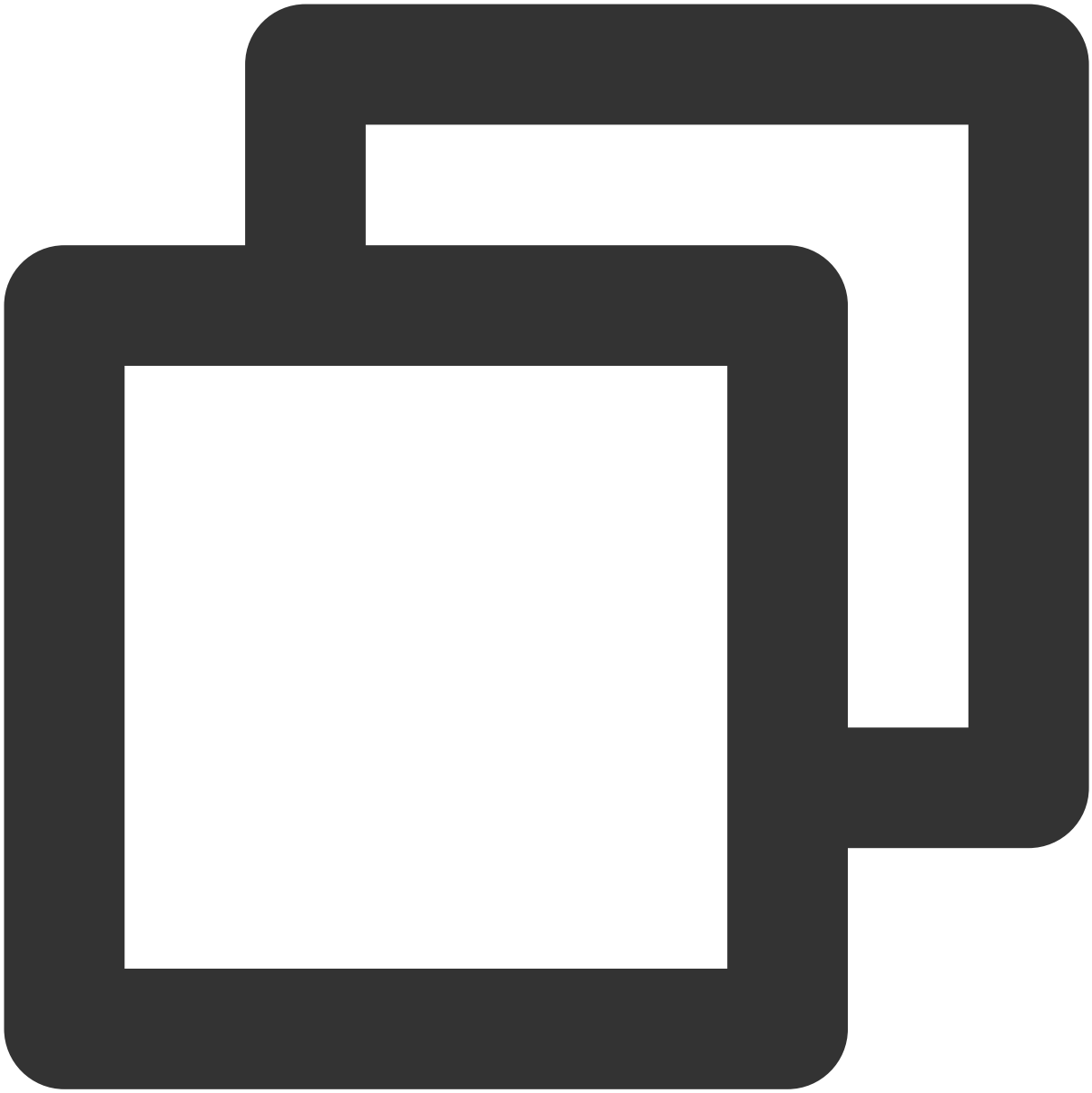
6.2 文件作用域

在 JavaScript 文件中声明的变量和函数只在该文件中有效；不同的文件中可以声明相同名字的变量和函数，不会互相影响。

通过全局函数 `getApp` 可以获取全局的应用实例，如果需要全局的数据可以在 `App()` 中设置，如：

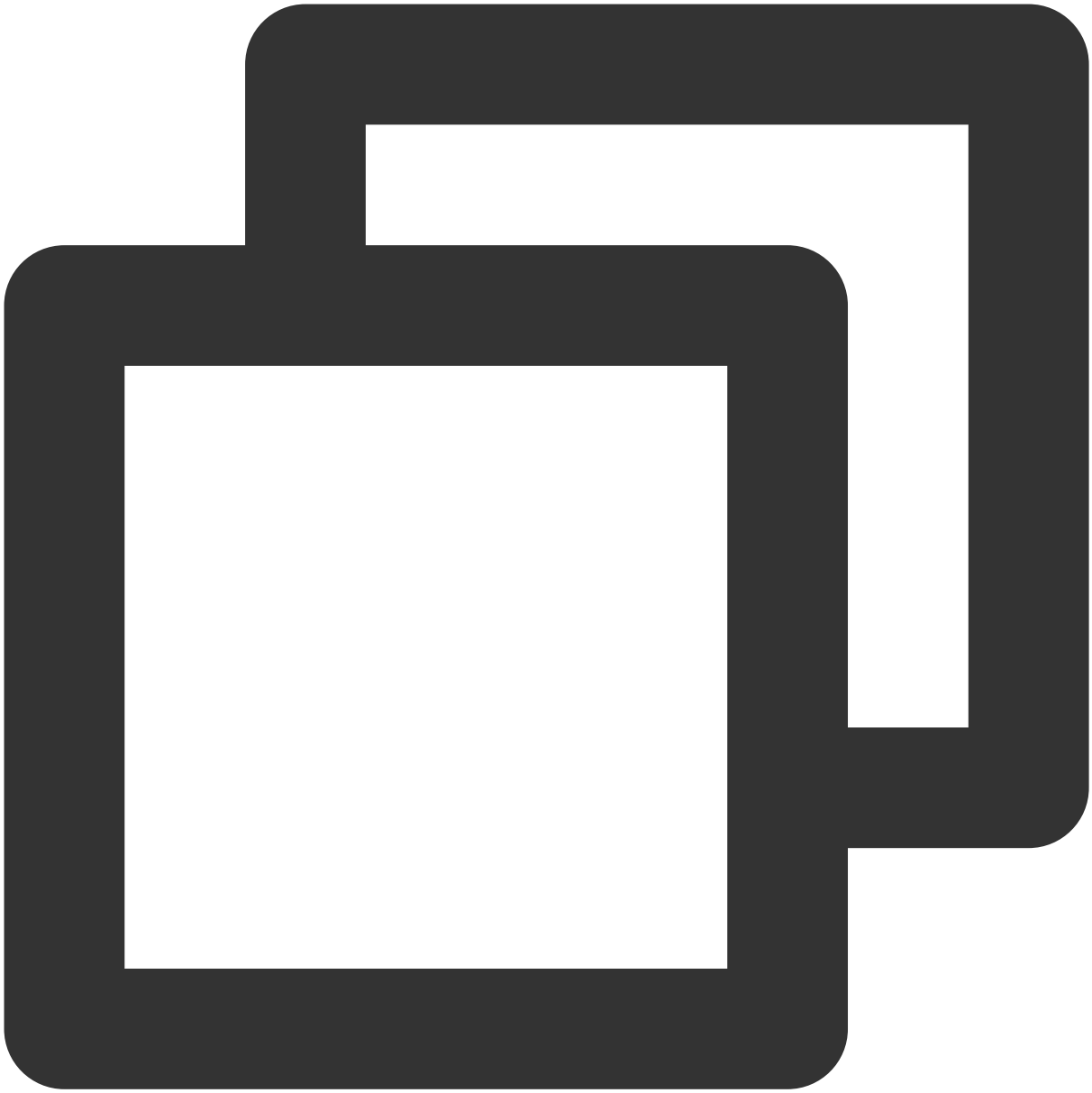


```
// app.js
App({
  globalData: 1
})
```



```
// a.js
// The localValue can only be used in file a.js.
var localValue = 'a'
// Get the app instance.
var app = getApp()
// Get the global data and change it.
app.globalData++
```

版权所有：腾讯云计算（北京）有限责任公司



```
// a.js
// The localValue can only be used in file a.js.
var localValue = 'a'
// Get the app instance.
var app = getApp()
// Get the global data and change it.
app.globalData++
```

说明：

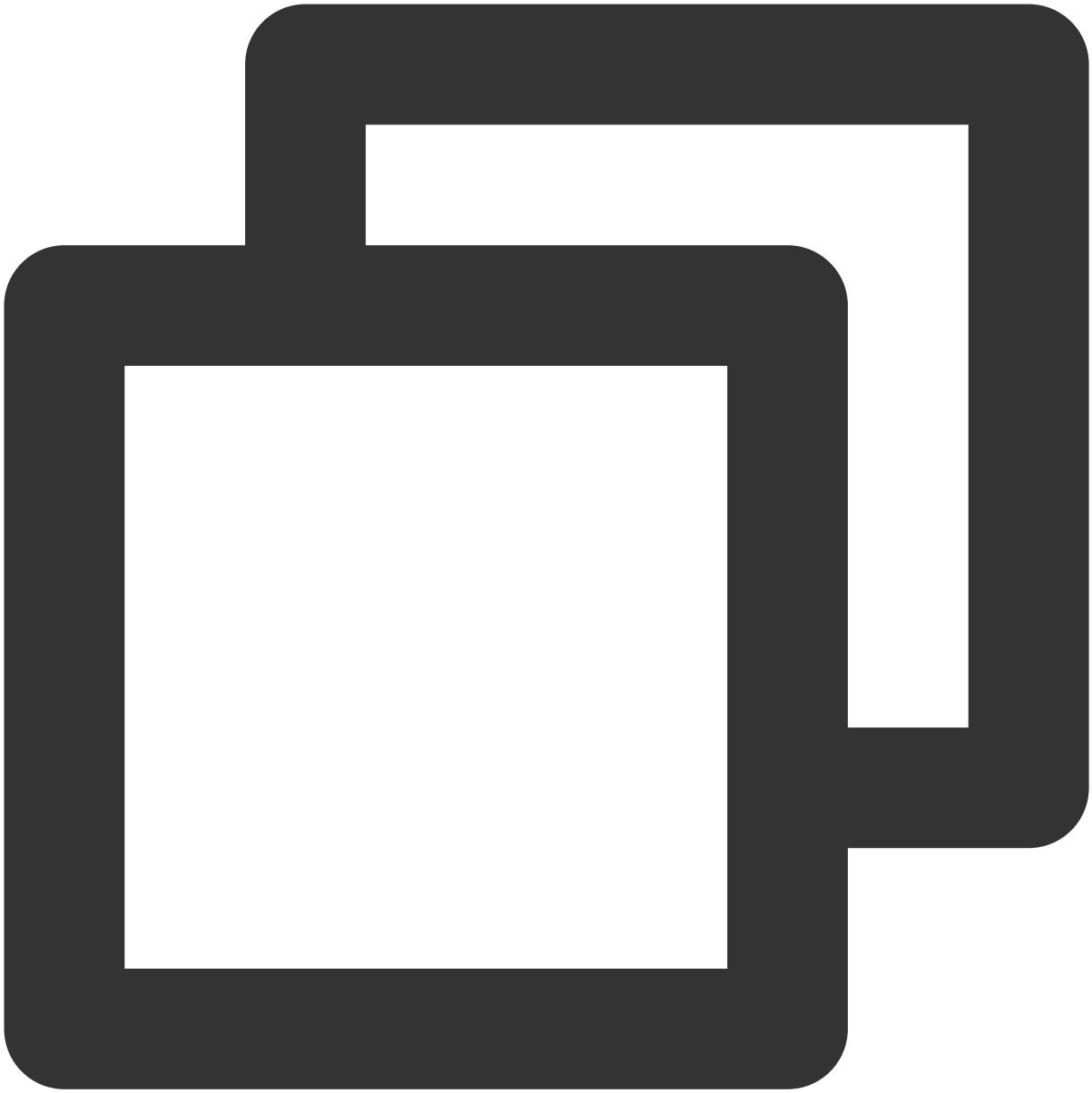
值得注意的是，`require` 引入模块时，需要使用**相对路径**。

7. API

小程序开发框架提供丰富的原生 API，可以方便的调起小程序SDK提供的能力，如获取用户信息，本地存储等。详细介绍请参考 API 文档。
通常，在小程序 API 有以下几种类型：

7.1 事件监听 API

我们约定，以 `on` 开头的 API 用来监听某个事件是否触发，如：wx.onSocketOpen，wx.onCompassChange 等。这类 API 接受一个回调函数作为参数，当事件触发时会调用这个回调函数，并将相关数据以参数形式传入。
代码示例：



```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

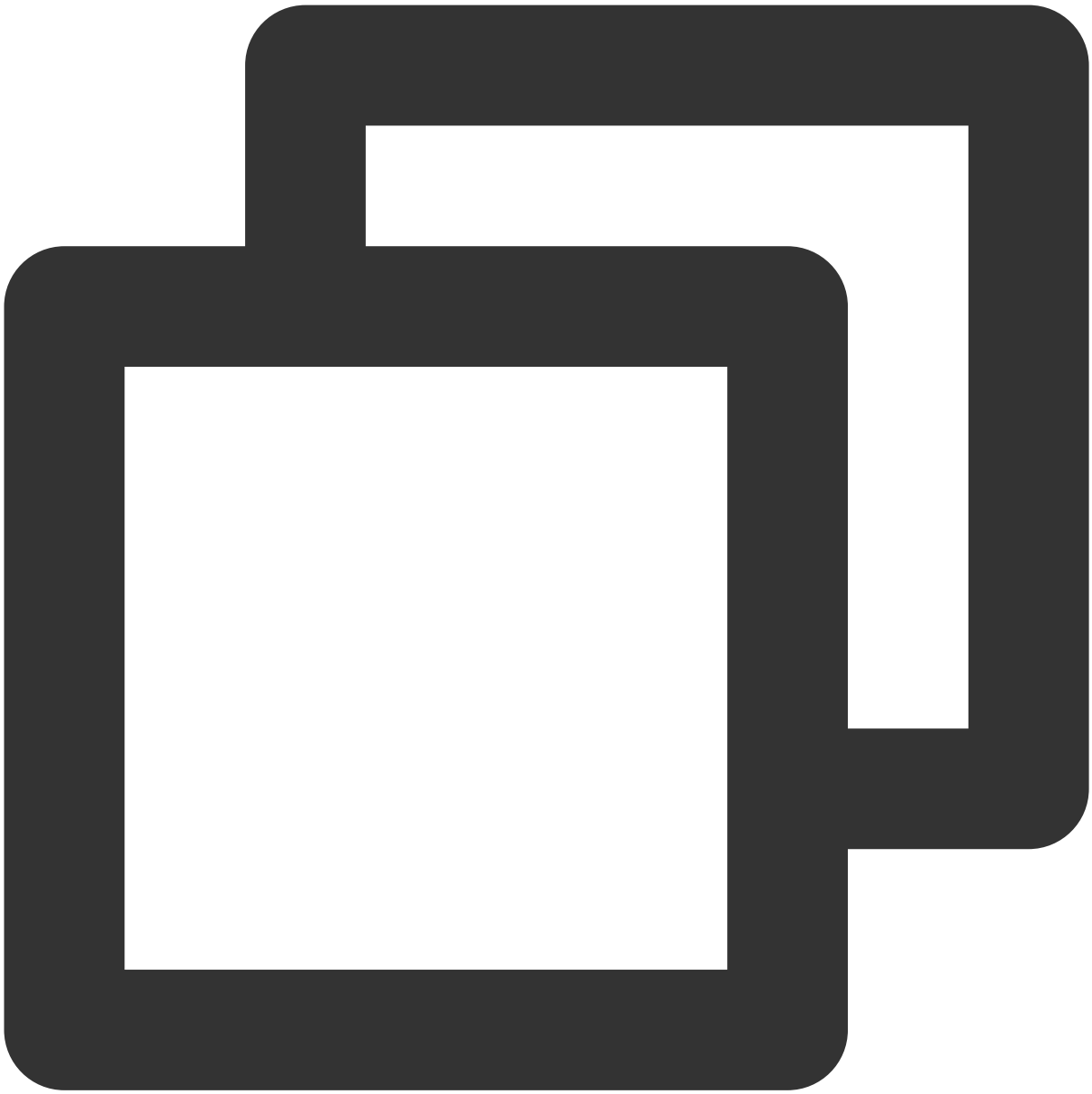
7.2 同步 API

我们约定，以 `Sync` 结尾的 API 都是同步 API， 如 wx.setStorageSync，wx.getSystemInfoSync 等。此外，也有一些其他的同步 API，如 wx.createWorker，wx.getBackgroundAudioManager 等，详情参见 API 文档中的说明。

同步 API 的执行结果可以通过函数返回值直接获取，如果执行出错会抛出异常。

版权所有：腾讯云计算（北京）有限责任公司

代码示例：



```
try {
  wx.setStorageSync('key', 'value')
} catch (e) {
  console.error(e)
}
```

7.3 异步API

大多数 API 都是异步 API，如 wx.request，wx.login 等。这类 API 接口通常都接受一个 `Object` 类型的参数，这个参数都支持按需指定以下字段来接收接口调用结果：

Object 参数说明

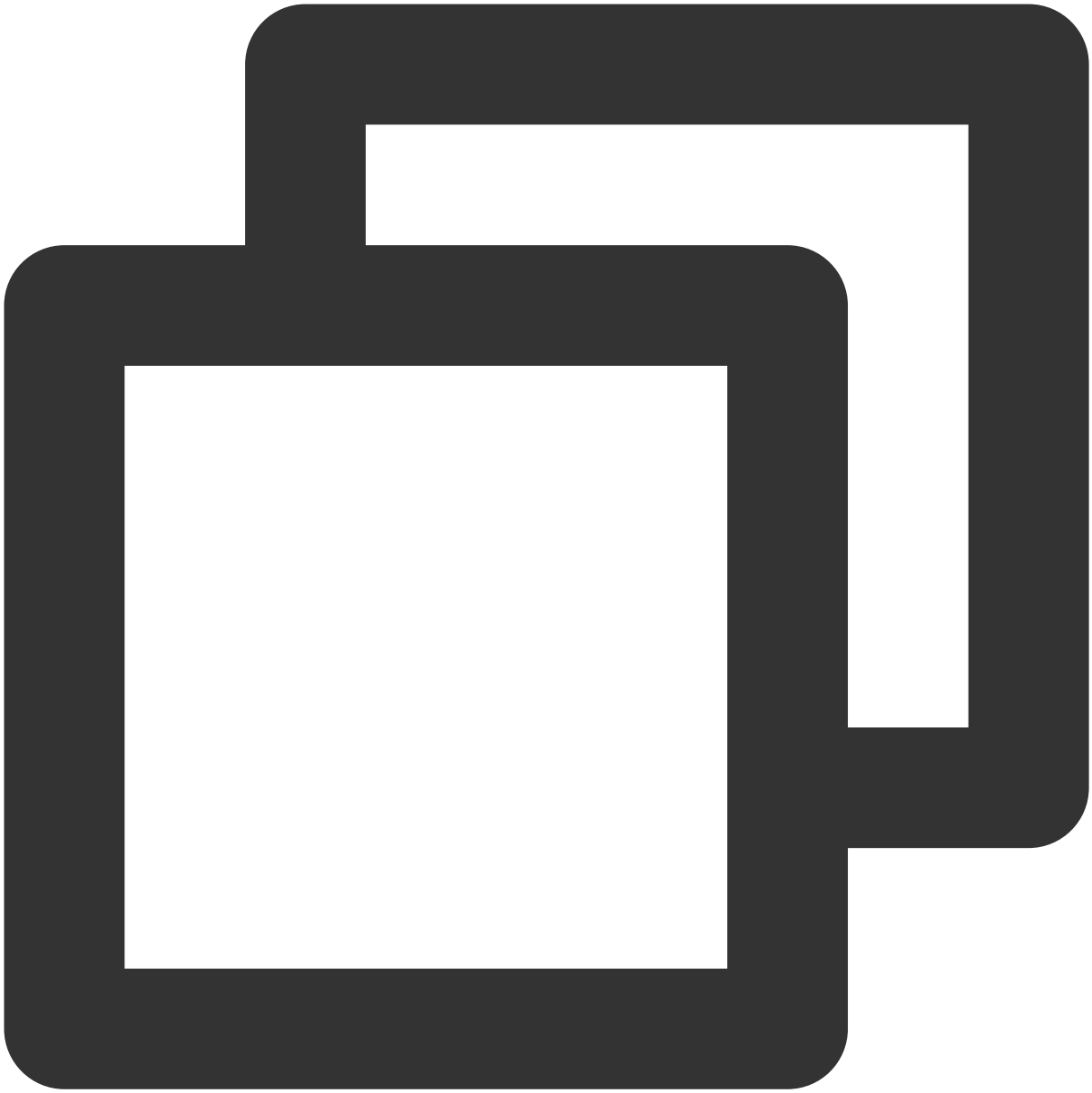
参数名	类型	必填	说明
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用成功的回调函数
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）
其他	any	-	接口定义的其他参数

回调函数的参数

`success`，`fail`，`complete` 函数调用时会传入一个 `Object` 类型参数，包含以下字段：

属性	类型	说明
errMsg	string	错误信息，如果调用成功返回 `${apiName}:ok`
errCode	number	错误码，仅部分 API 支持，具体含义请参考对应 API 文档，成功时为 `0`。
其他	any	接口返回的其他数据

代码示例：



```
wx.login({
  success(res) {
    console.log(res.code)
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

版权所有：腾讯云计算（北京）有限责任公司

渲染层

最近更新时间：2024-07-12 18:58:47

框架的渲染层由 WXML 与 WXSS 编写，由组件来进行展示。将逻辑层的数据反映成视图，同时将渲染层的事件发送给逻辑层。

WXML(WeiXin Markup language) 用于描述页面的结构。

WXS(WeiXin Script) 是小程序的一套脚本语言，结合 WXML，可以构建出页面的结构。

WXSS(WeiXin Style Sheet) 用于描述页面的样式。

组件(Component)是视图的基本组成单元。

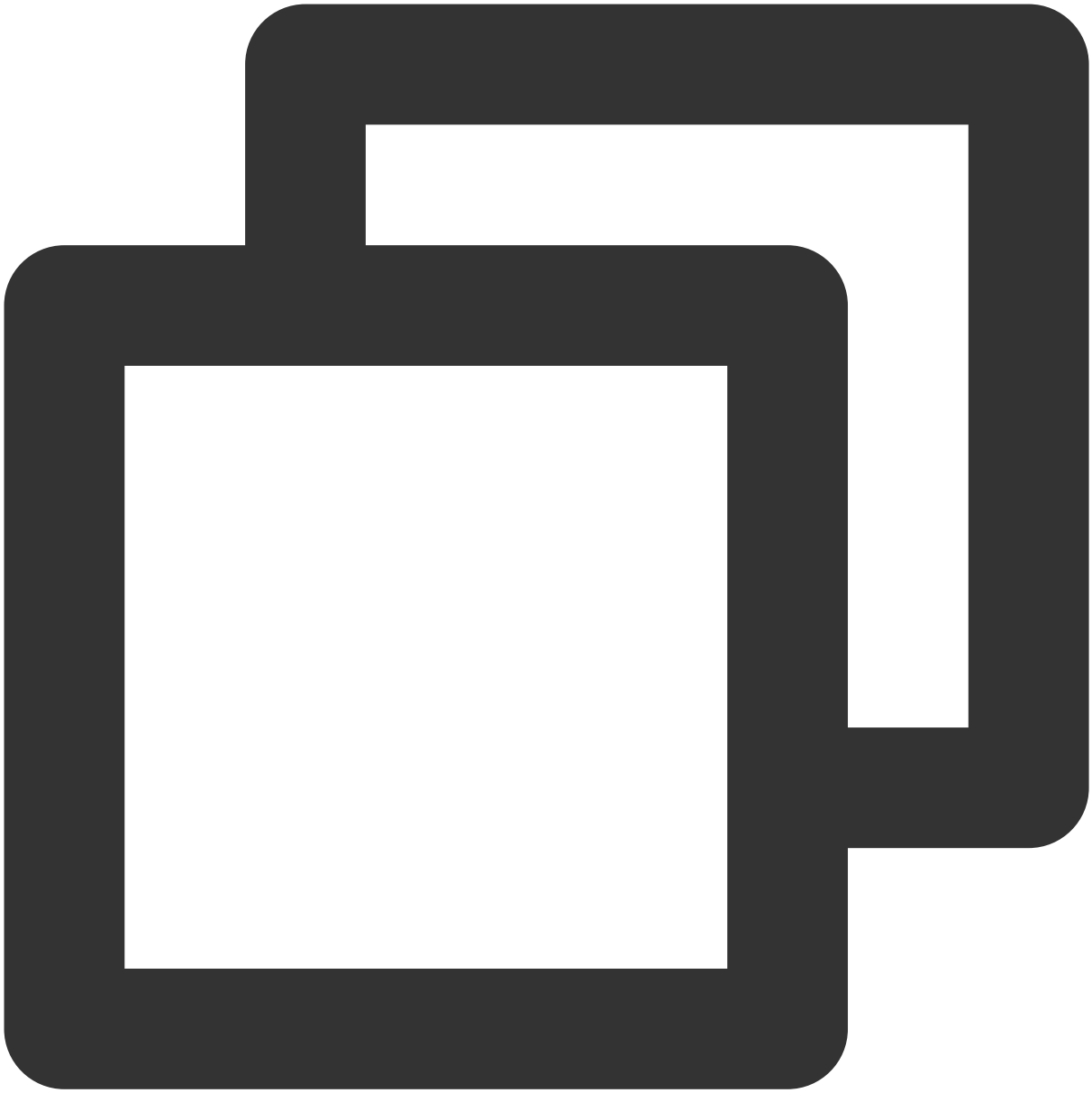
1. WXML

WXML（WeiXin Markup Language）是框架设计的一套标签语言，结合基础组件、事件系统可以构建出页面的结构。

要完整了解 WXML 语法，请参考 WXML 语法参考。

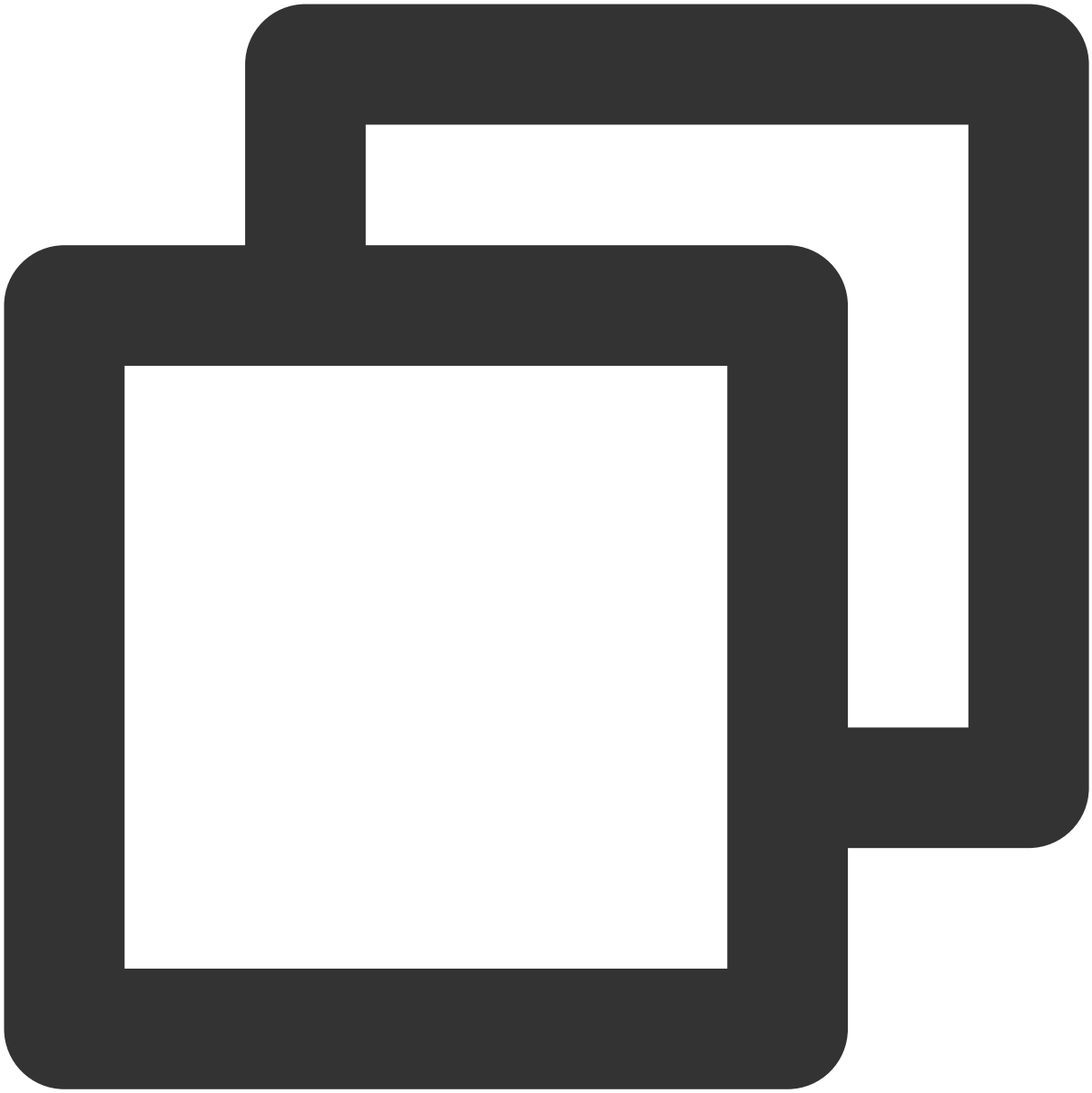
用以下一些简单的例子来看看 WXML 具有什么能力：

1.1 数据绑定



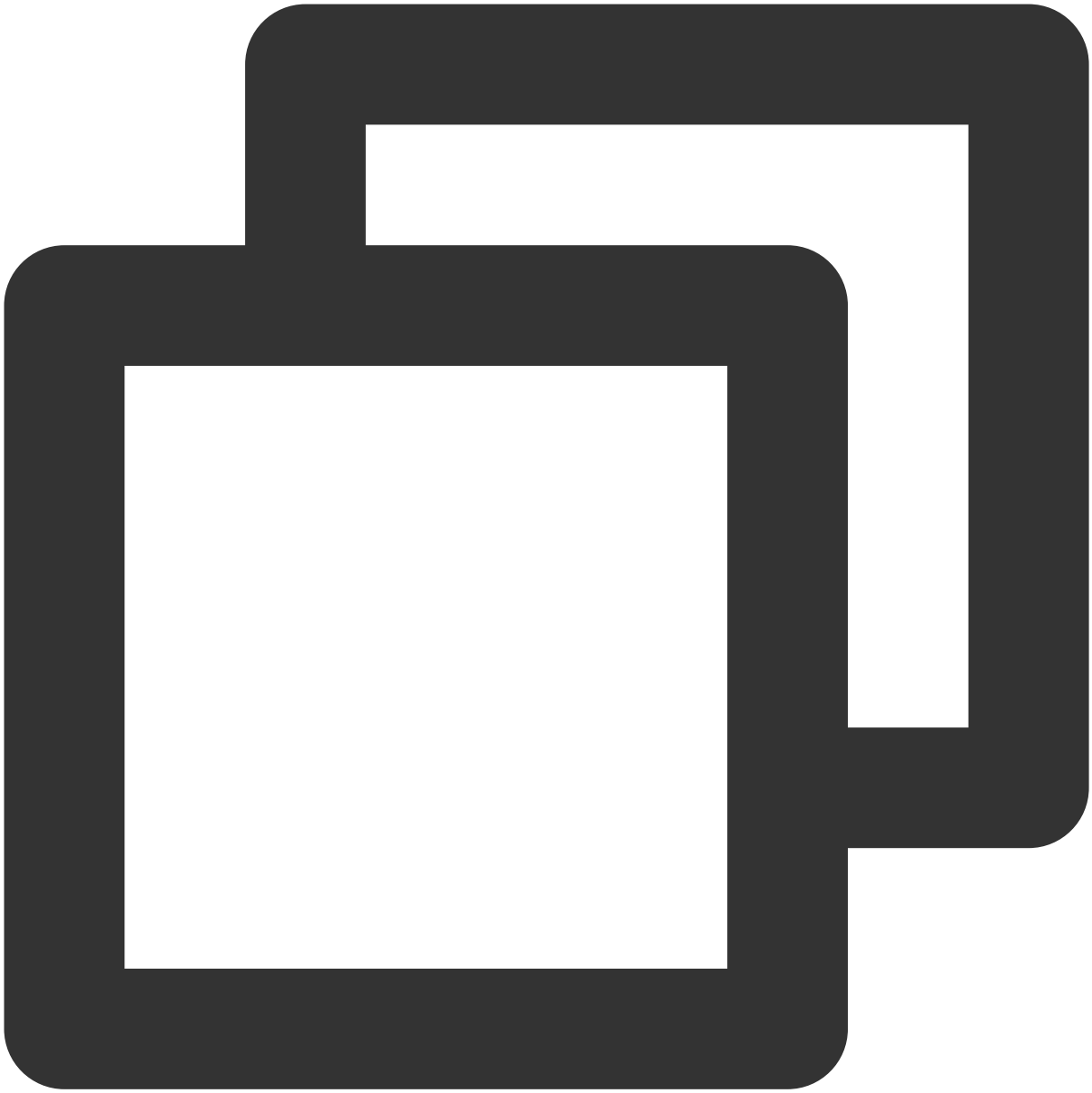
```
<!-- wxml -->
<view> {{message}} </view>
```

版权所有：腾讯云计算（北京）有限责任公司



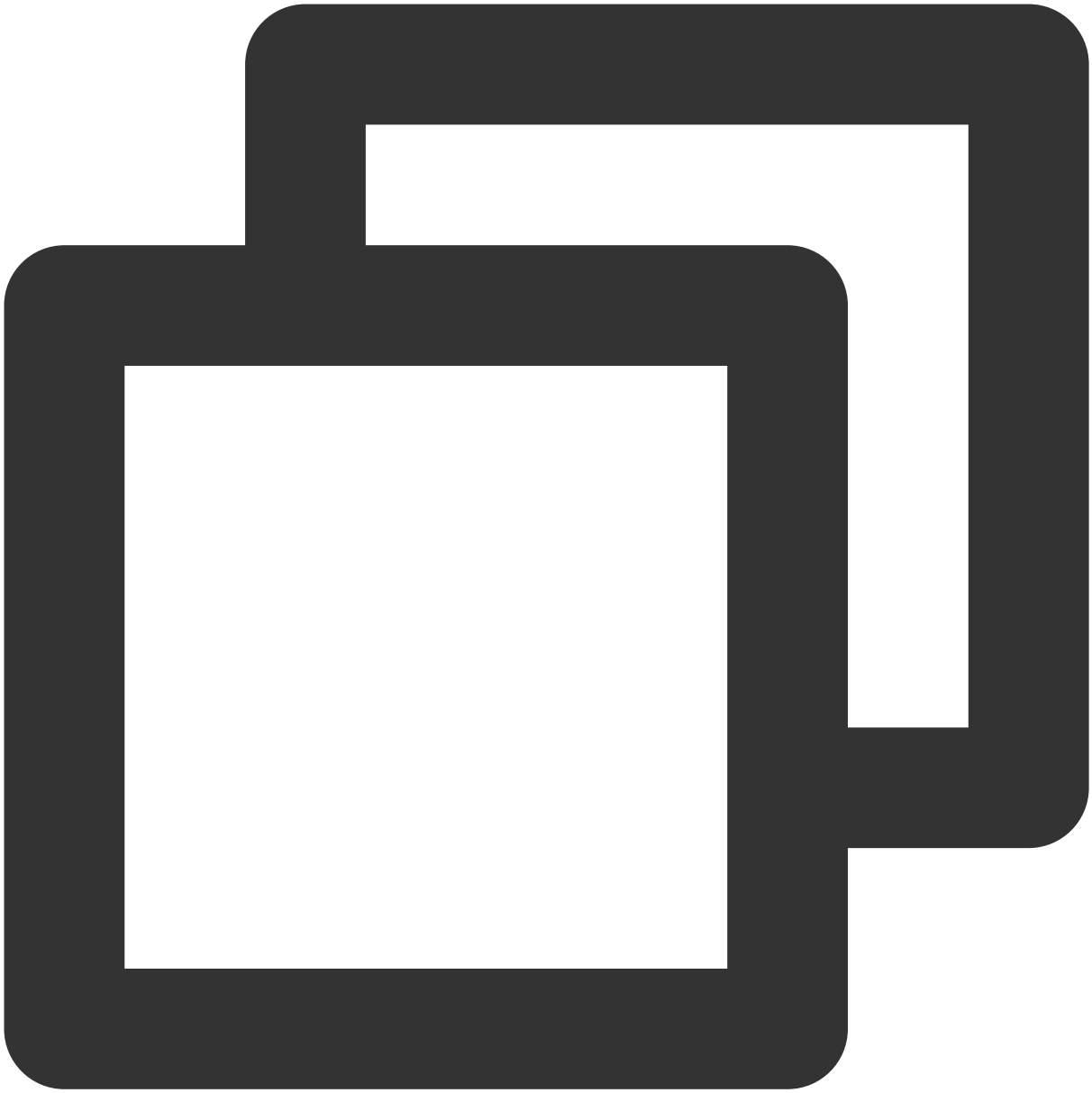
```
// page.js
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

1.2 列表渲染



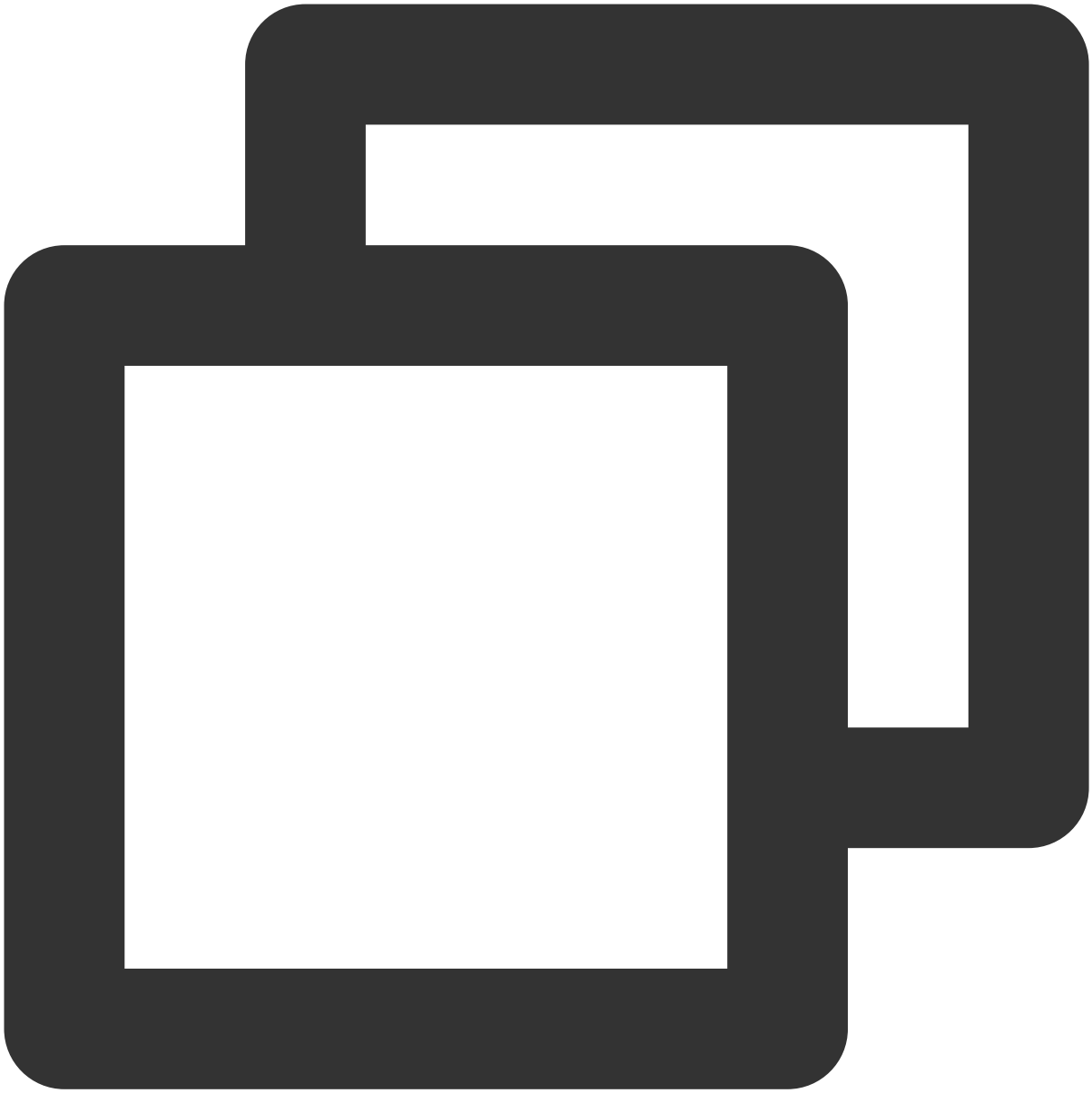
```
<!--wxml-->
<view wx:for="{{array}}"> {{item}} </view>
```

1.3 条件渲染



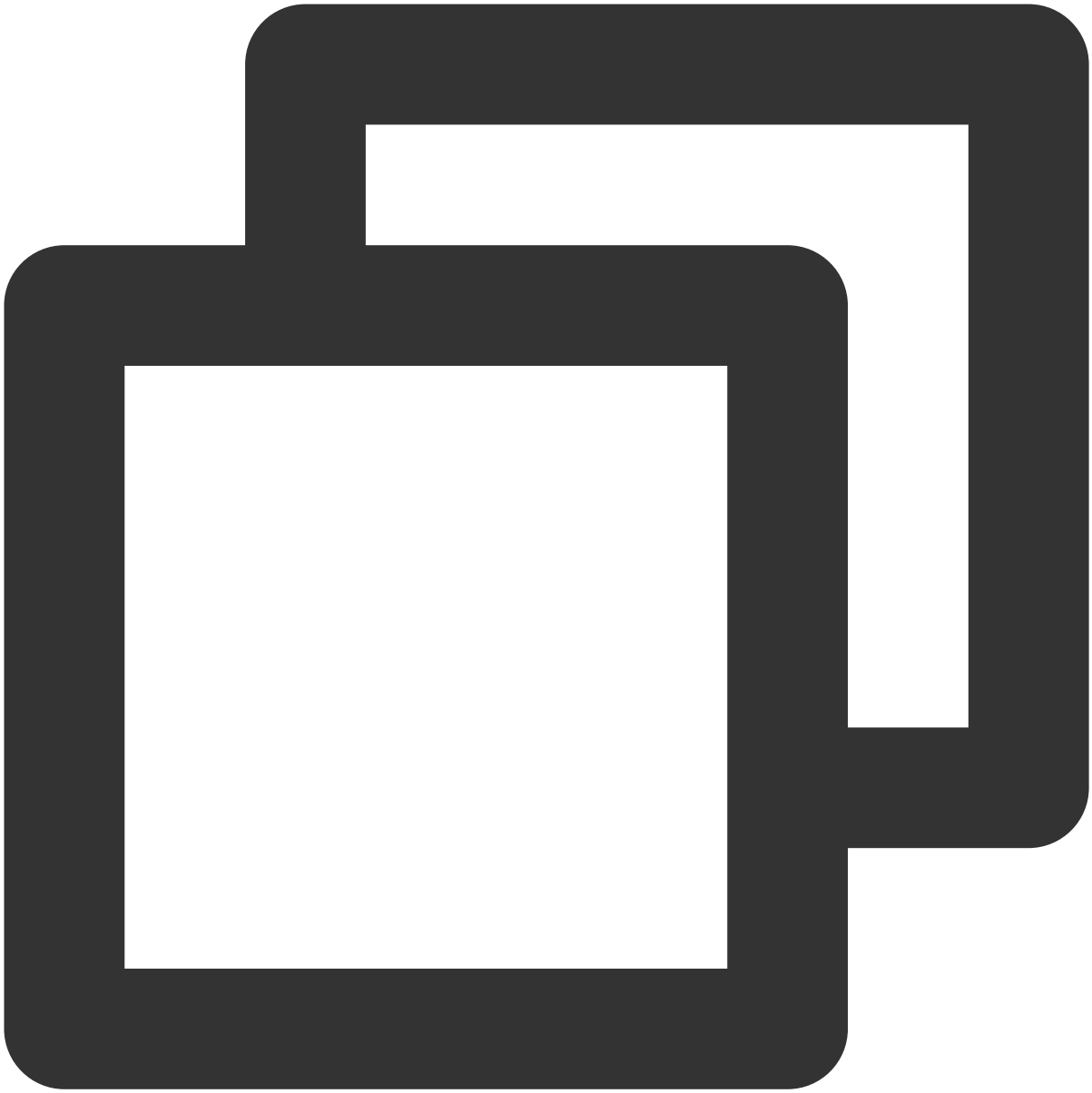
```
<!--wxml-->
<view wx:if="{{view == 'WEBVIEW'}}"> WEBVIEW </view>
<view wx:elif="{{view == 'APP'}}"> APP </view>
<view wx:else="{{view == 'MINA'}}"> MINA </view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// page.js
Page({
  data: {
    view: 'MINA'
  }
})
```

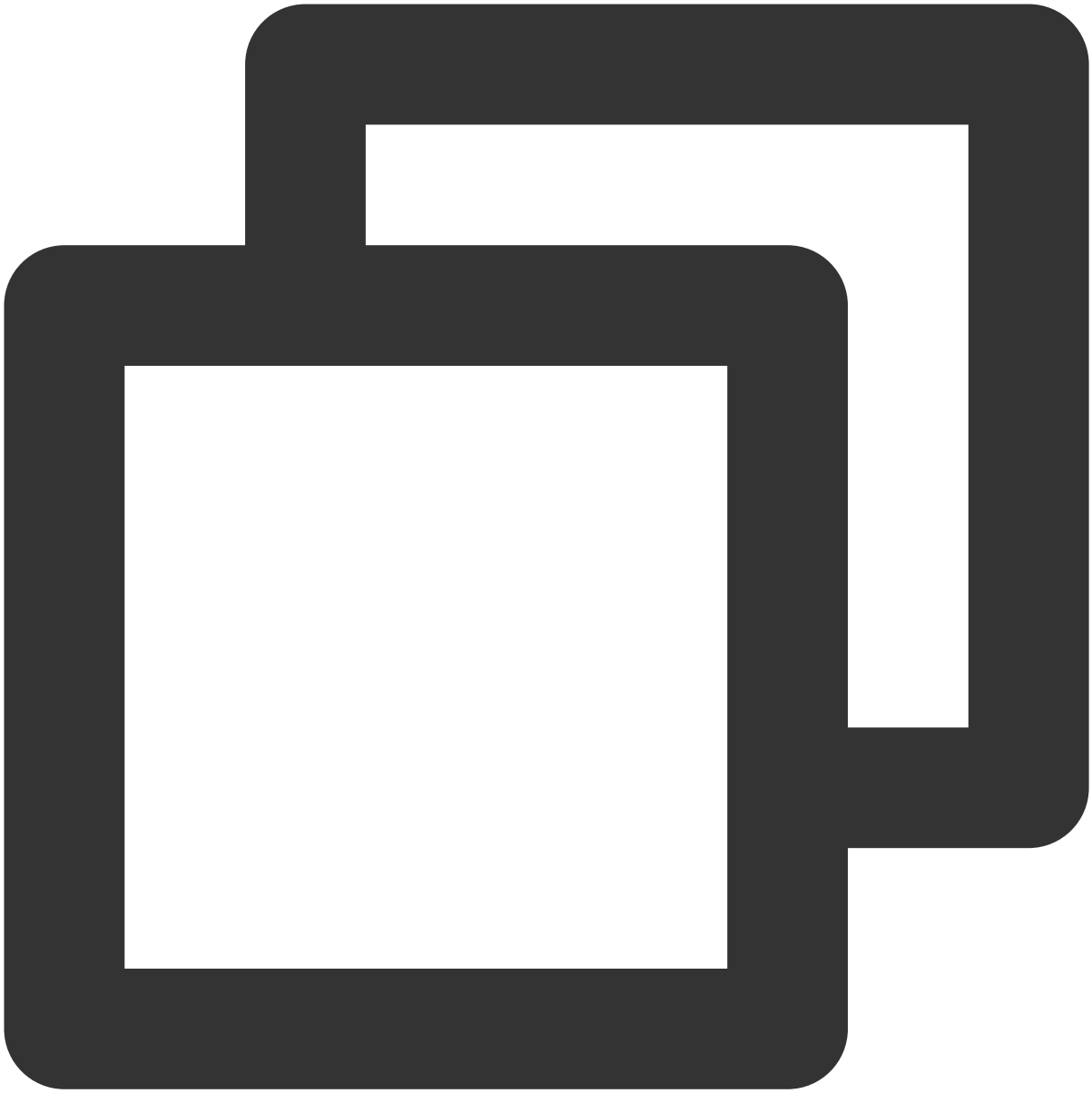
1.4 模板



```
<!--wxml-->
<template name="staffName">
  <view>
    FirstName: {{firstName}}, LastName: {{lastName}}
  </view>
</template>

<template is="staffName" data="{{...staffA}}"></template>
<template is="staffName" data="{{...staffB}}"></template>
<template is="staffName" data="{{...staffC}}"></template>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// page.js
Page({
  data: {
    staffA: {firstName: 'Hulk', lastName: 'Hu'},
    staffB: {firstName: 'Shang', lastName: 'You'},
    staffC: {firstName: 'Gideon', lastName: 'Lin'}
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

2. WXSS

WXSS (WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的组件样式。
WXSS 用来决定 WXML 的组件应该怎么显示。
为了适应广大的前端开发者，WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，WXSS 对 CSS 进行了扩充以及修改。
与 CSS 相比，WXSS 扩展的特性有：
尺寸单位
样式导入
注意：
iOS 8 及以下的 iOS 版本，如使用 flexbox 布局，需添加 `display: -webkit-flex` 属性。

2.1 尺寸单位

rpx（responsive pixel）：可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx。如在 iPhone6 上，屏幕宽度为375px，共有750个物理像素，则750rpx = 375px = 750物理像素，1rpx = 0.5px = 1物理像素。
根据不同的设备屏幕宽度，rpx 和 px 的换算关系也会有所不同。例如：

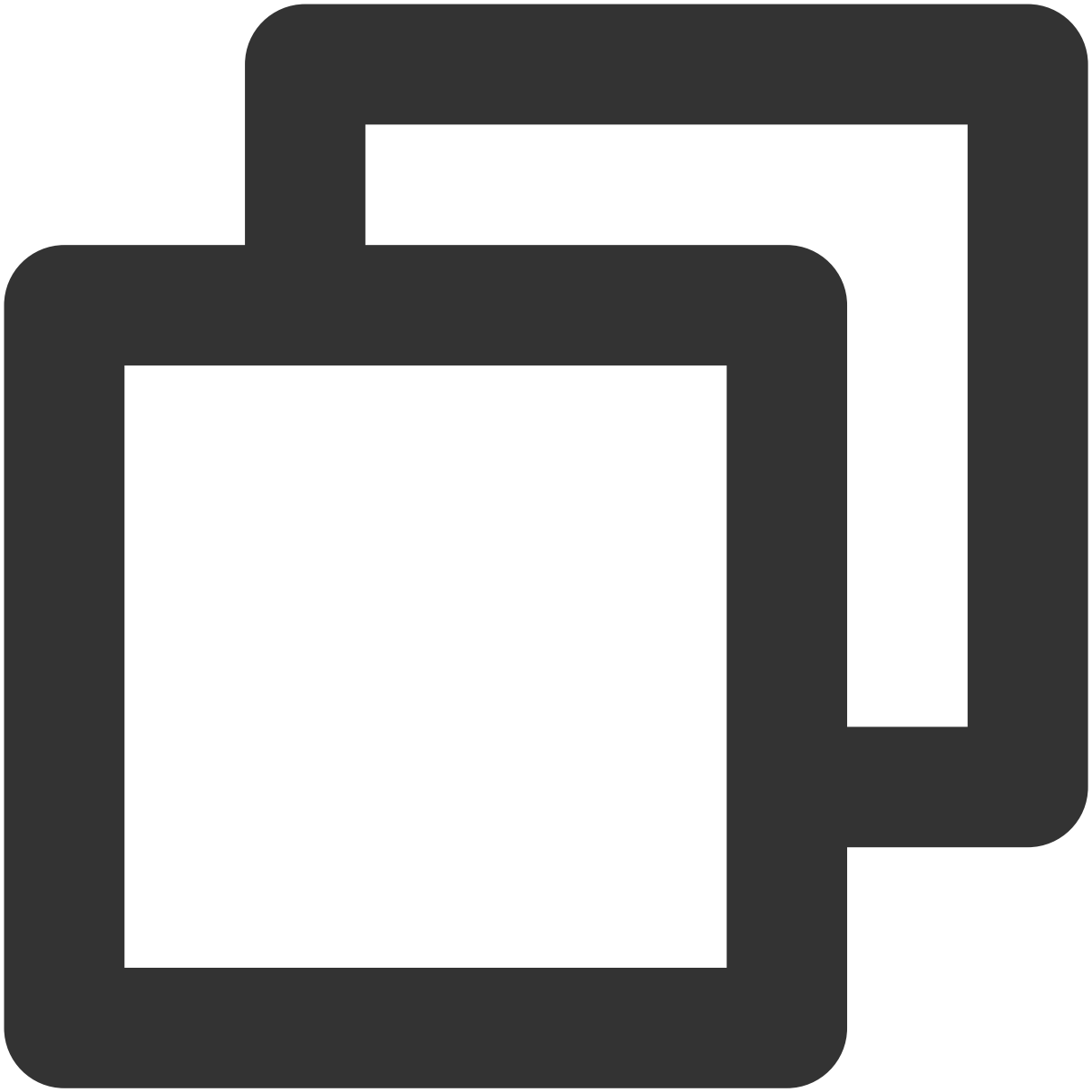
设备	屏幕宽度	rpx 换算 px（屏幕宽度/750）	px 换算 rpx (750/屏幕宽度)
iPhone5	320px	1rpx = 0.42px	1px = 2.34rpx
iPhone6	375px	1rpx = 0.5px	1px = 2rpx
iPhone6 Plus	414px	1rpx = 0.552px	1px = 1.81rpx

说明：
建议： 开发微信小程序时设计师可以用 **iPhone6** 作为视觉稿的标准。
注意： 在较小的屏幕上不可避免的会有一些毛刺，请在开发时尽量避免这种情况。

2.2 样式导入

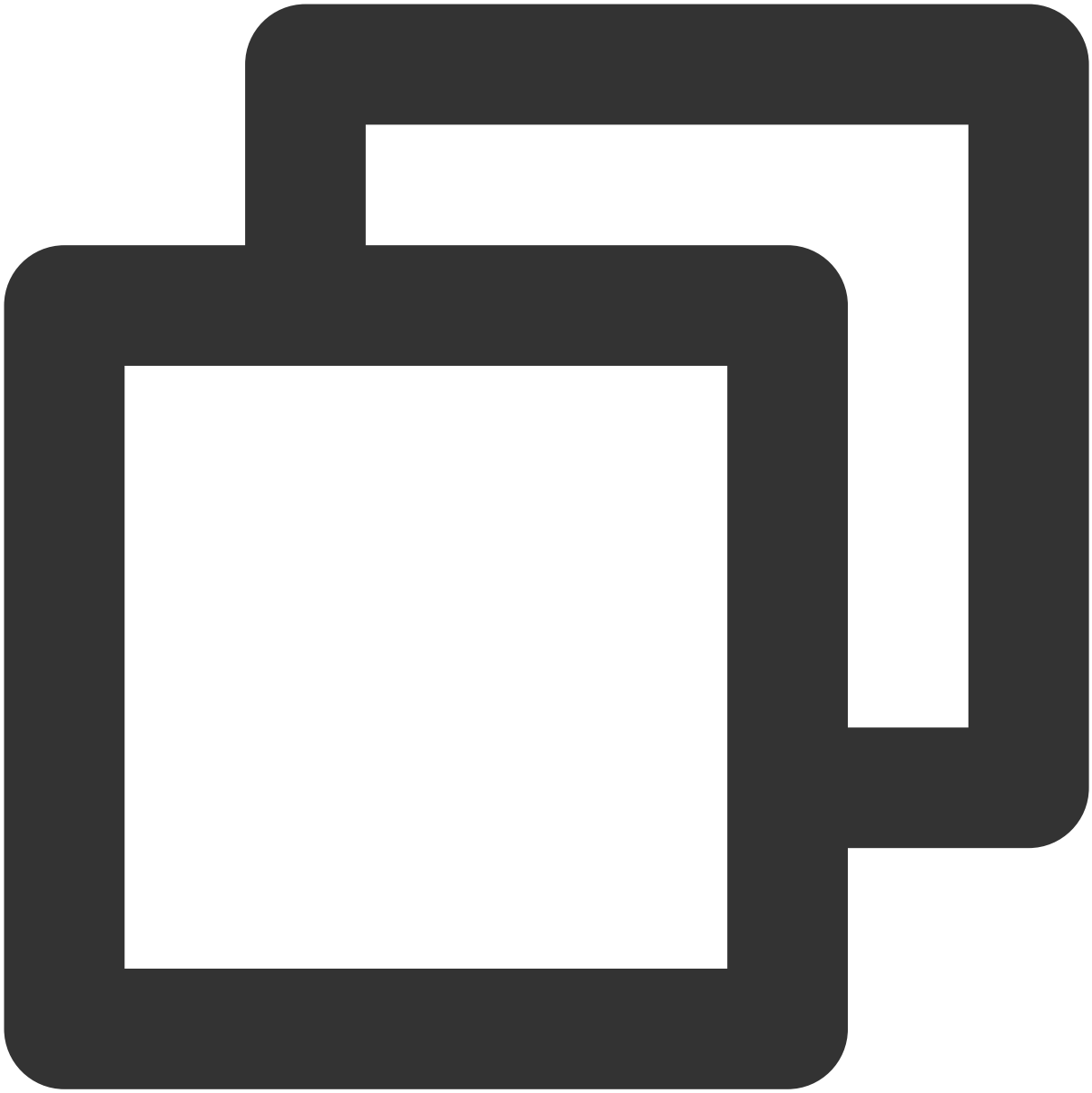
使用 `@import` 语句可以导入外联样式表， `@import` 后跟需要导入的外联样式表的相对路径，用 `;` 表示语句结束。
示例代码：

版权所有：腾讯云计算（北京）有限责任公司



```
/** common.wxss **/
.small-p {
  padding:5px;
}
```

版权所有：腾讯云计算（北京）有限责任公司

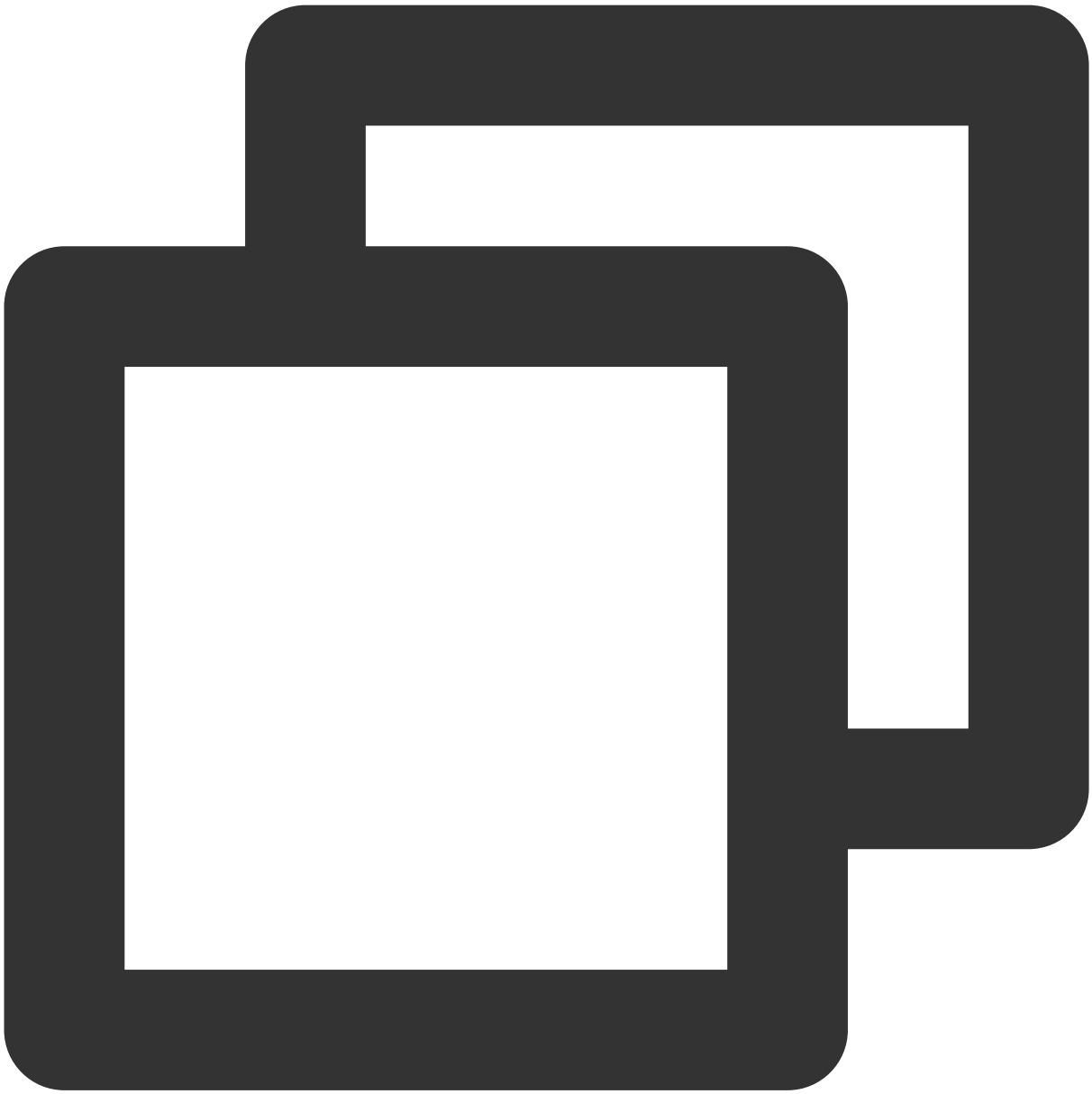


```
/** app.wxss **/
@import "common.wxss";
.middle-p {
  padding:15px;
}
```

2.3 内联样式

框架组件上支持使用 style、class 属性来控制组件的样式。

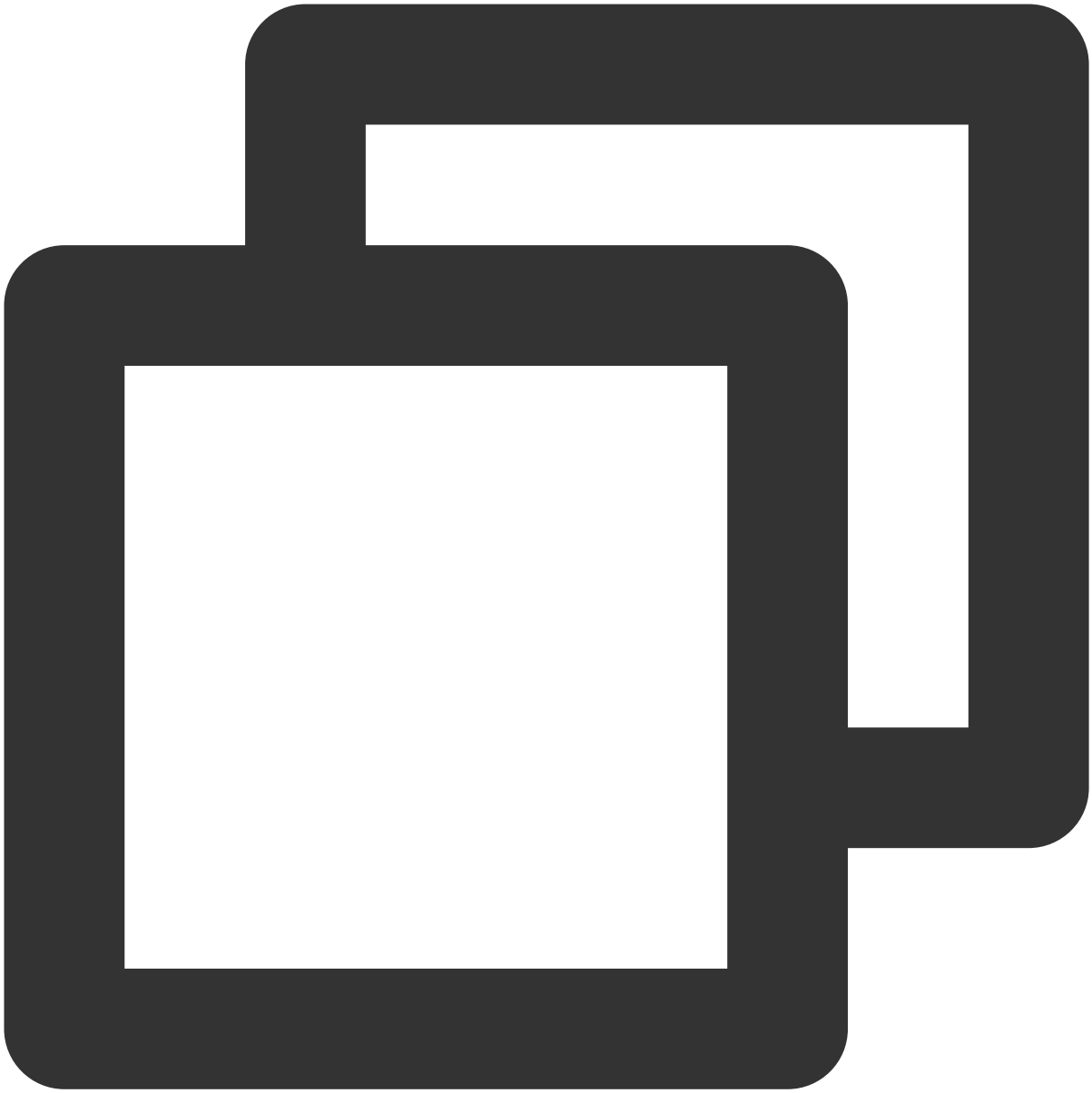
style：静态的样式统一写到 class 中。style 接收动态的样式，在运行时会进行解析，请尽量避免将静态的样式写进 style 中，以免影响渲染速度。



```
/** common.wxss **/
.small-p {
  padding:5px;
}
```

class：用于指定样式规则，其属性值是样式规则中类选择器名(样式类名)的集合，样式类名不需要带上，样式类名之间用空格分隔。

版权所有：腾讯云计算（北京）有限责任公司



```
<view class="normal_view" />
```

2.4 选择器

目前支持的选择器有：

选择器	样例	样例描述
.class	.intro	选择所有拥有 class="intro" 的组件
#id	#firstname	选择拥有 id="firstname" 的组件

版权所有：腾讯云计算（北京）有限责任公司

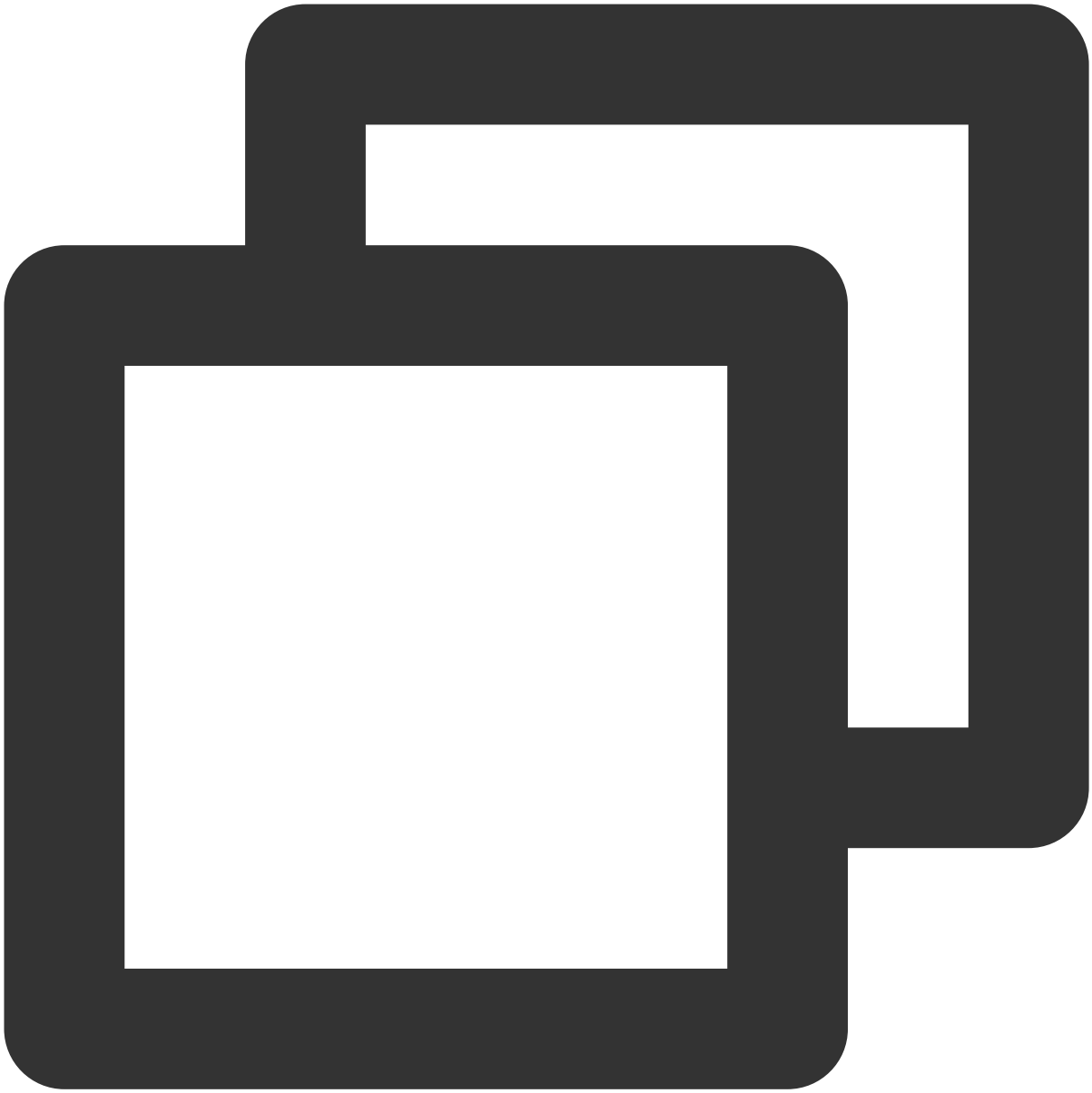
element	view	选择所有 view 组件
element，element	view, checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

2.5 全局样式与局部样式

定义在 app.wxss 中的样式为全局样式，作用于每一个页面。在 page 的 wxss 文件中定义的样式为局部样式，只作用在对应的页面，并会覆盖 app.wxss 中相同的选择器。

3. WXS

WXS（WeiXin Script）是内联在 WXML 中的脚本段。通过 WXS 可以在模板中内联少量处理脚本，丰富模板的数据预处理能力。另外，WXS 还可以用来编写简单的 WXS 事件响应函数。
从语法上看，WXS 类似于有少量限制的 JavaScript 。要完整了解 WXS 语法，请参考WXS - 介绍。
以下是一些使用 WXS 的简单示例。
页面渲染：

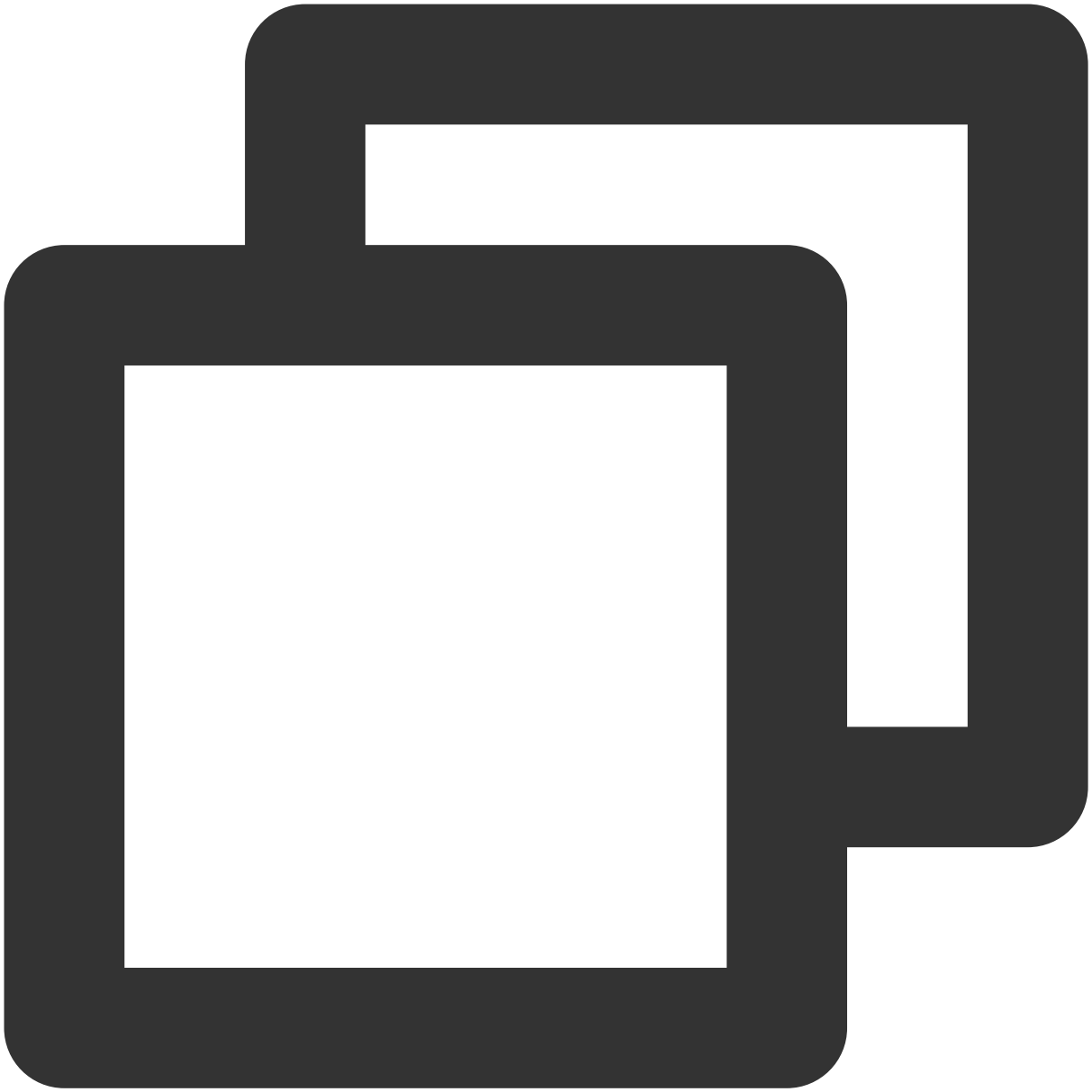


```
<!--wxml-->
<wxs module="m1">
var msg = "hello world";

module.exports.message = msg;
</wxs>

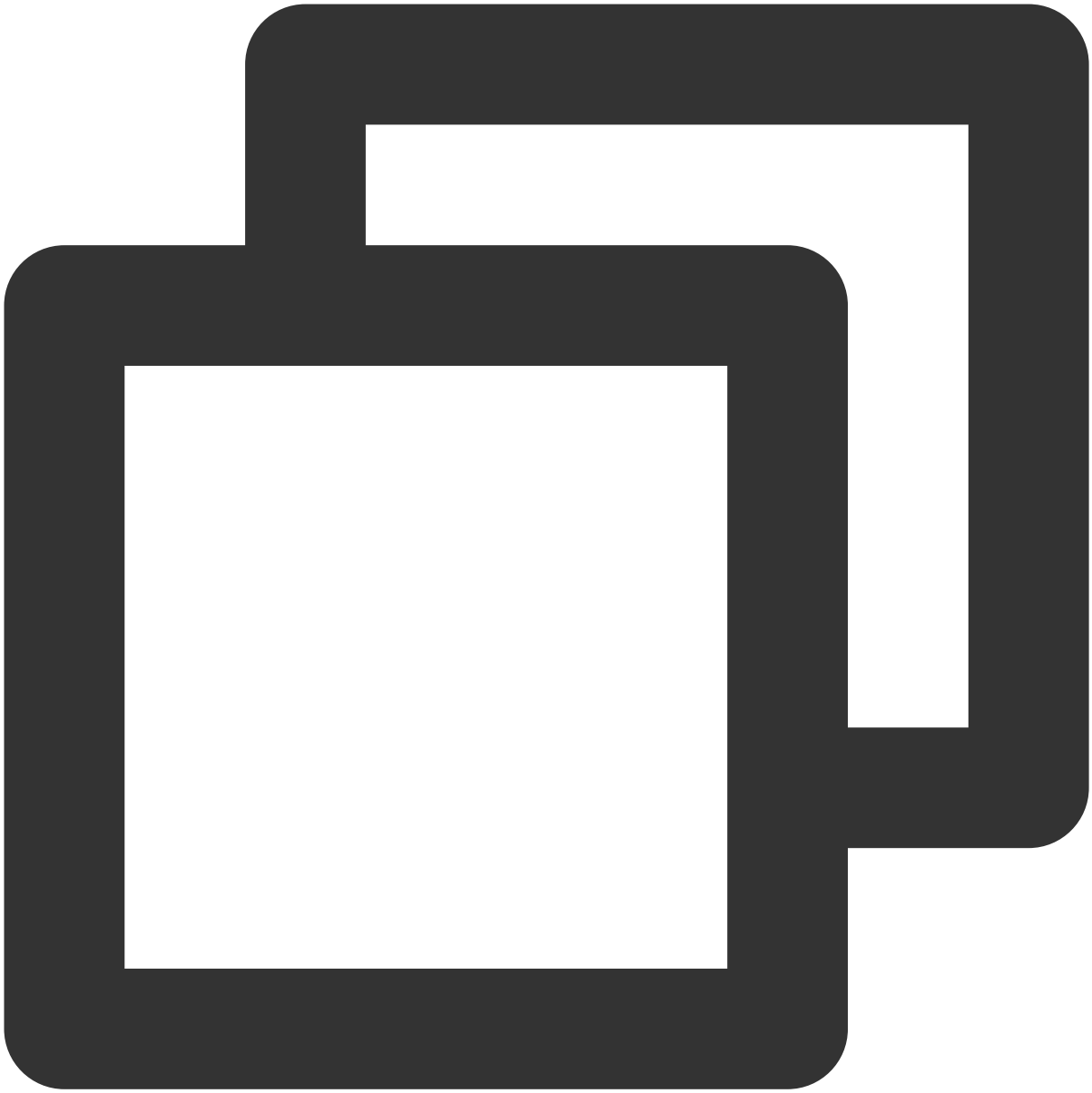
<view> {{m1.message}} </view>
```

页面输出：



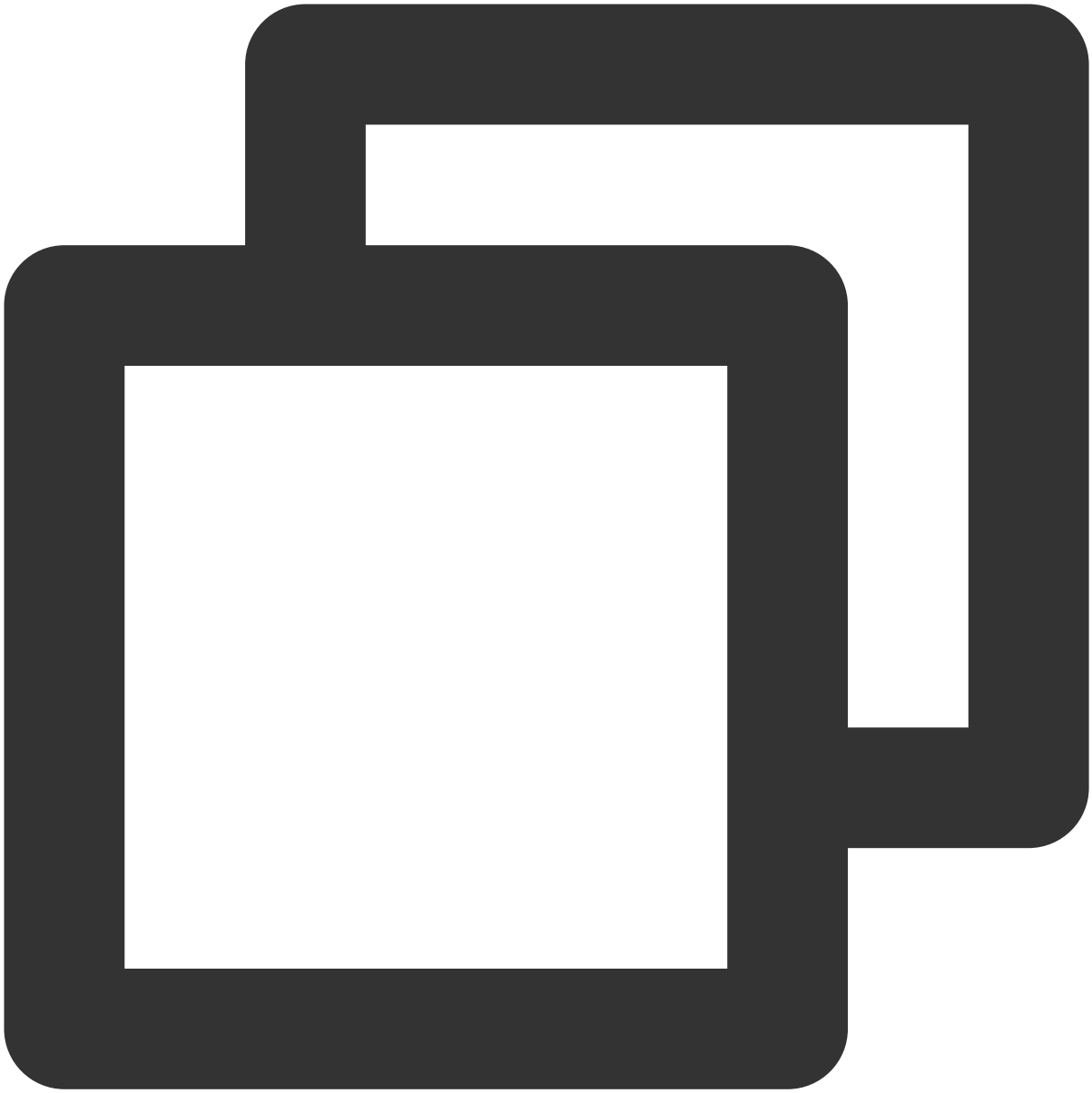
```
hello world
```

数据处理：



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

版权所有：腾讯云计算（北京）有限责任公司



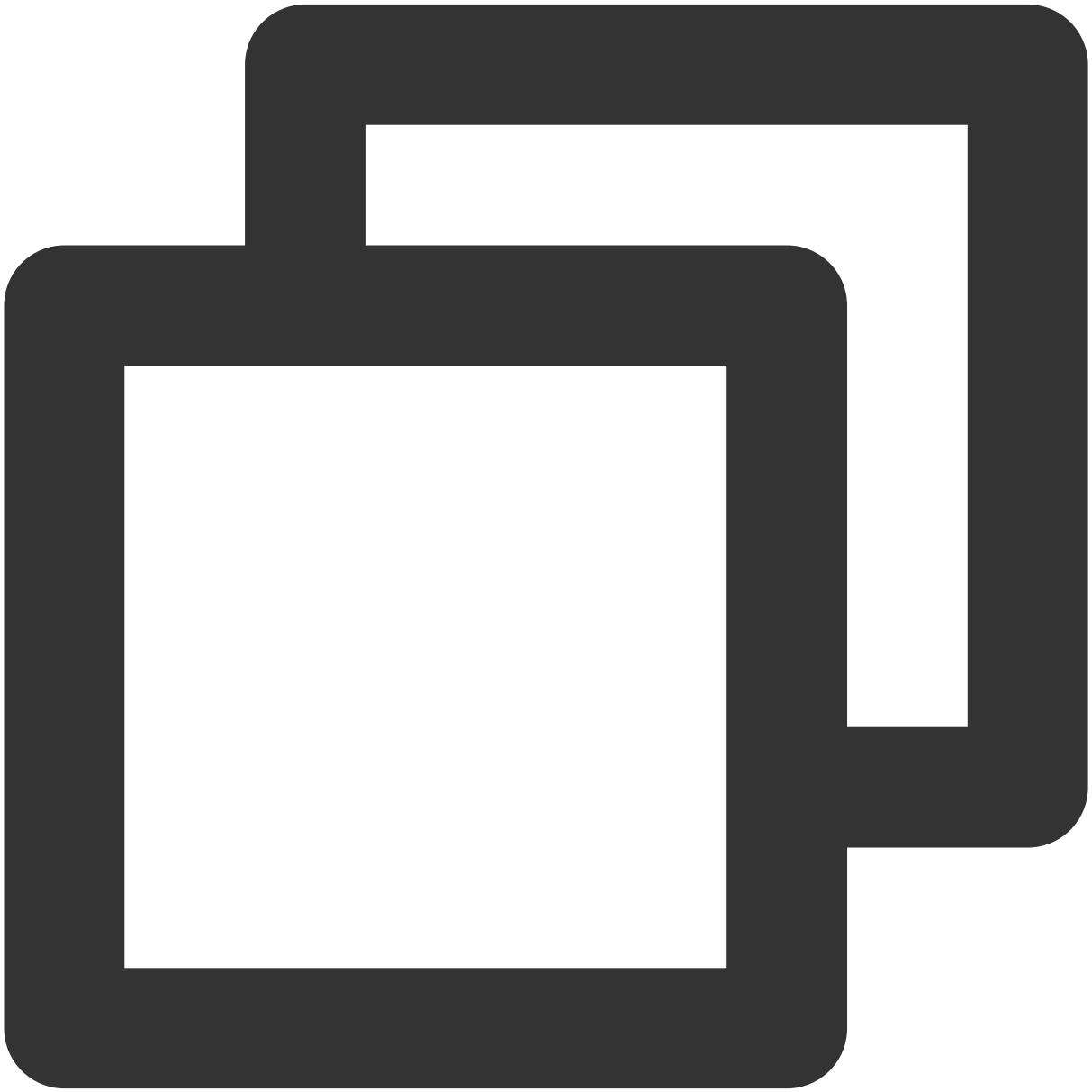
```
<!--wxml-->
<!-- 下面的 getMax 函数，接受一个数组，且返回数组中最大的元素的值 -->
<wxs module="m1">
var getMax = function(array) {
  var max = undefined;
  for (var i = 0; i < array.length; ++i) {
    max = max === undefined ?
      array[i] :
      (max >= array[i] ? max : array[i]);
  }
  return max;
```

```
}

module.exports.getMax = getMax;
</wxs>

<!-- 调用 wxs 里面的 getMax 函数, 参数为 page.js 里面的 array -->
<view> {{m1.getMax(array)}} </view>
```

页面输出：



```
5
```

4. 事件系统

4.1 什么是事件

事件是渲染层到逻辑层的通讯方式。

事件可以将用户的行为反馈到逻辑层进行处理。

事件可以绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数。

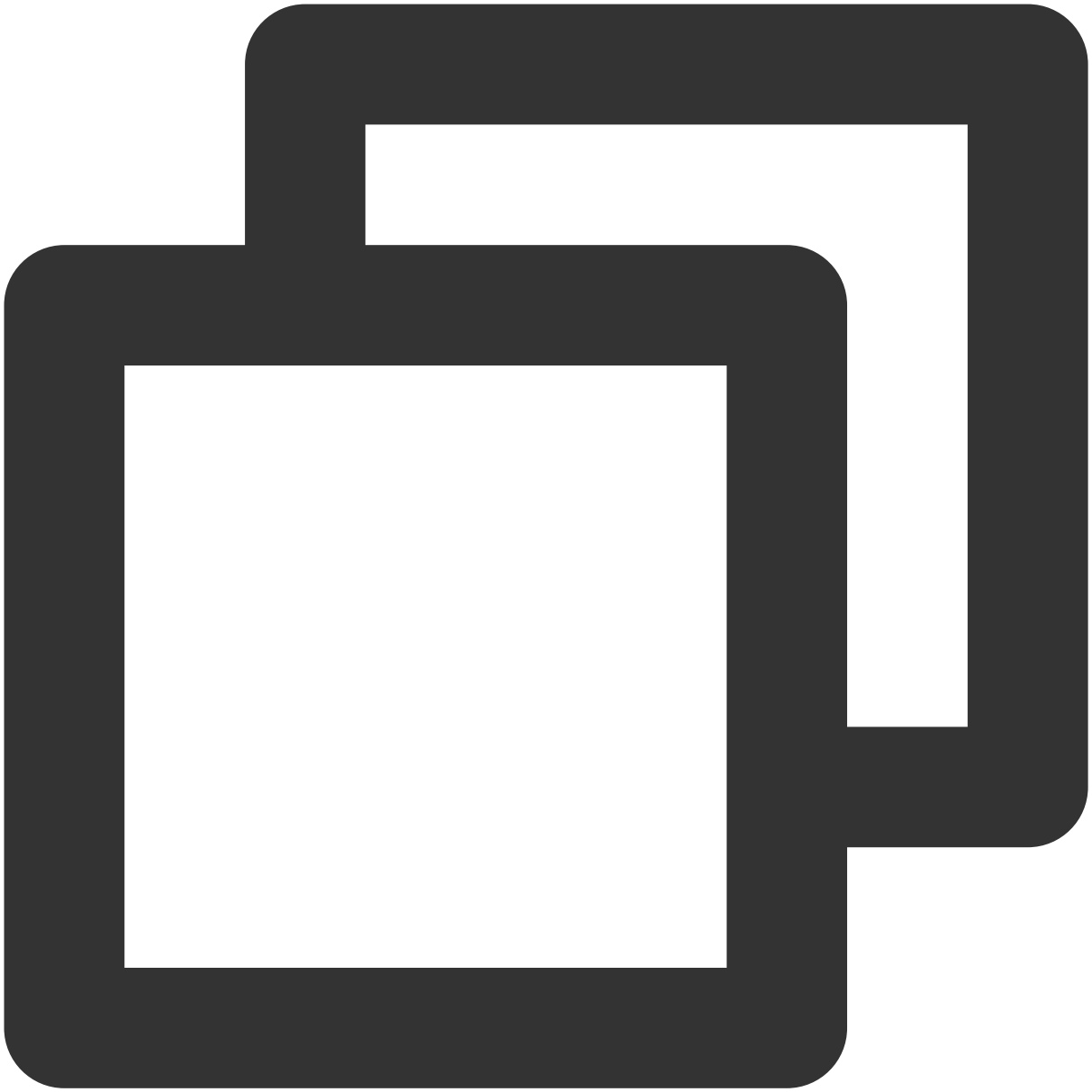
事件对象可以携带额外信息，如 id, dataset, touches。

4.2 事件的使用方式

在组件中绑定一个事件处理函数。

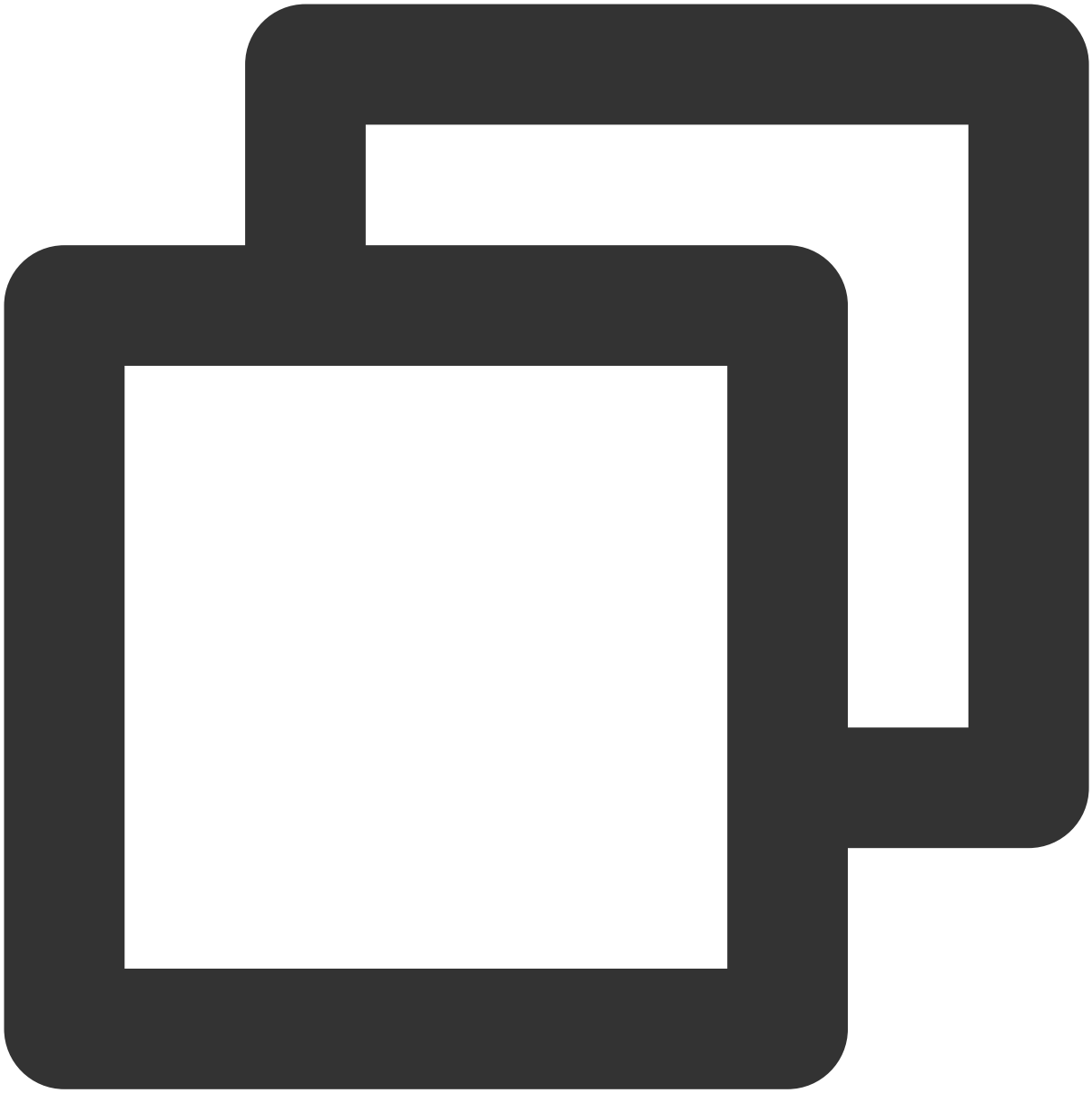
如 `bindtap` ，当用户点击该组件的时候会在该页面对应的Page中找到相应的事件处理函数。

版权所有：腾讯云计算（北京）有限责任公司



```
<view id="tapTest" data-hi="Weixin" bindtap="tapName"> Click me! </view>
```

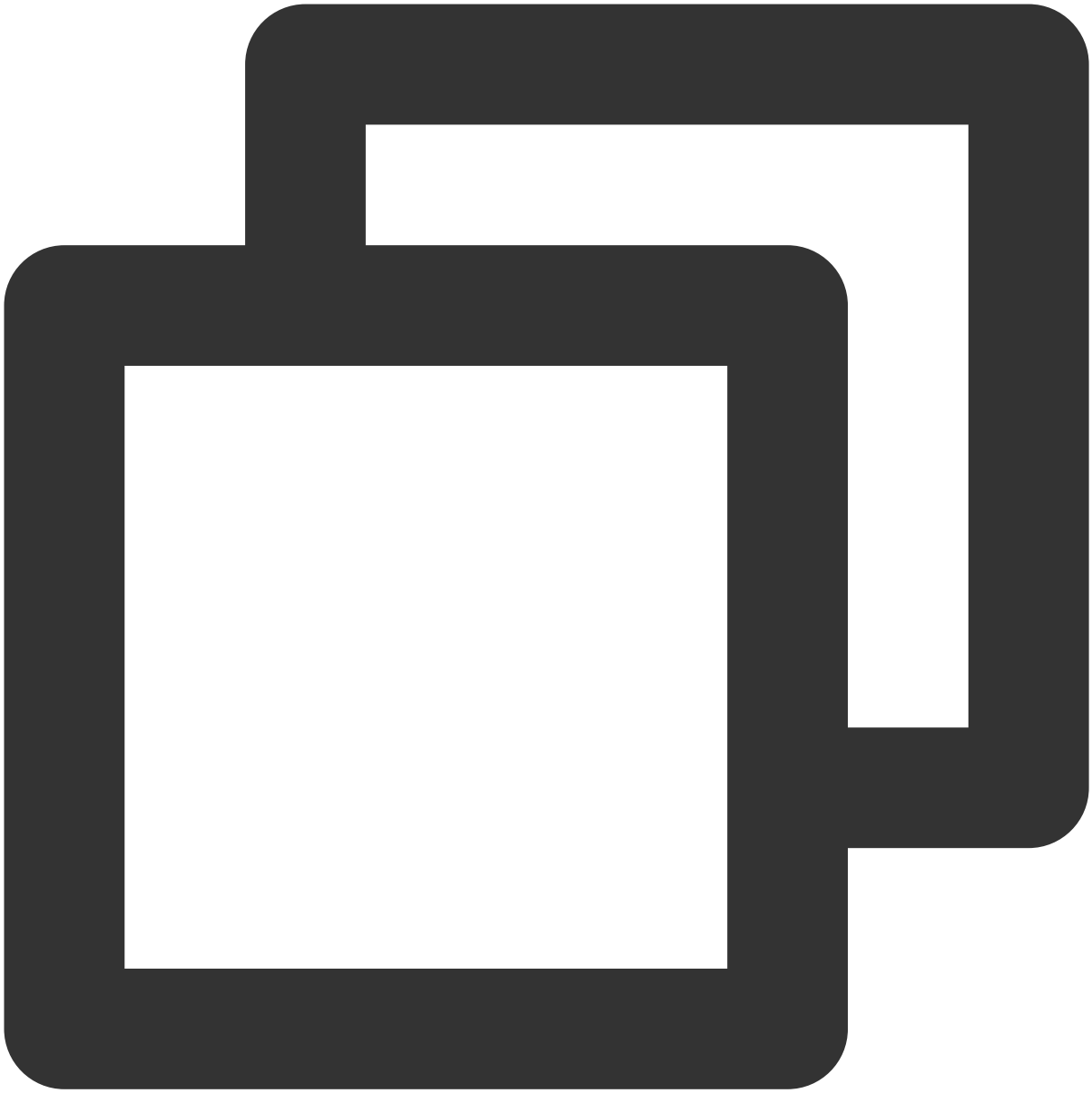
在相应的 `Page` 定义中写上相应的事件处理函数，参数是 `event` 。



```
Page({
  tapName: function(event) {
    console.log(event)
  }
})
```

可以看到log出来的信息大致如下：

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "type":"tap",
  "timeStamp":895,
  "target": {
    "id": "tapTest",
    "dataset":  {
      "hi":"Weixin"
    }
  },
  "currentTarget":  {
    "id": "tapTest",
```

```
      "dataset": {
        "hi":"Weixin"
      }
    },
    "detail": {
      "x":53,
      "y":14
    },
    "touches":[{
      "identifier":0,
      "pageX":53,
      "pageY":14,
      "clientX":53,
      "clientY":14
    }],
    "changedTouches":[{
      "identifier":0,
      "pageX":53,
      "pageY":14,
      "clientX":53,
      "clientY":14
    }]
  }
```

4.3 使用WXS函数响应事件

4.3.1 背景

有频繁用户交互的效果在小程序上表现是比较卡顿的，例如页面有2个元素 A 和 B，用户在 A 上做 touchmove 手势，要求 B 也跟随移动，movable-view 就是一个典型的例子。一次 touchmove 事件的响应过程为：
touchmove 事件从渲染层（Webview）抛到逻辑层（App Service）；
逻辑层（App Service）处理 touchmove 事件，再通过 setData 来改变 B 的位置。
一次 touchmove 的响应需要经过 2 次的逻辑层和渲染层的通信以及一次渲染，通信的耗时比较大。此外 setData 渲染也会阻塞其它脚本执行，导致了整个用户交互的动画过程会有延迟。

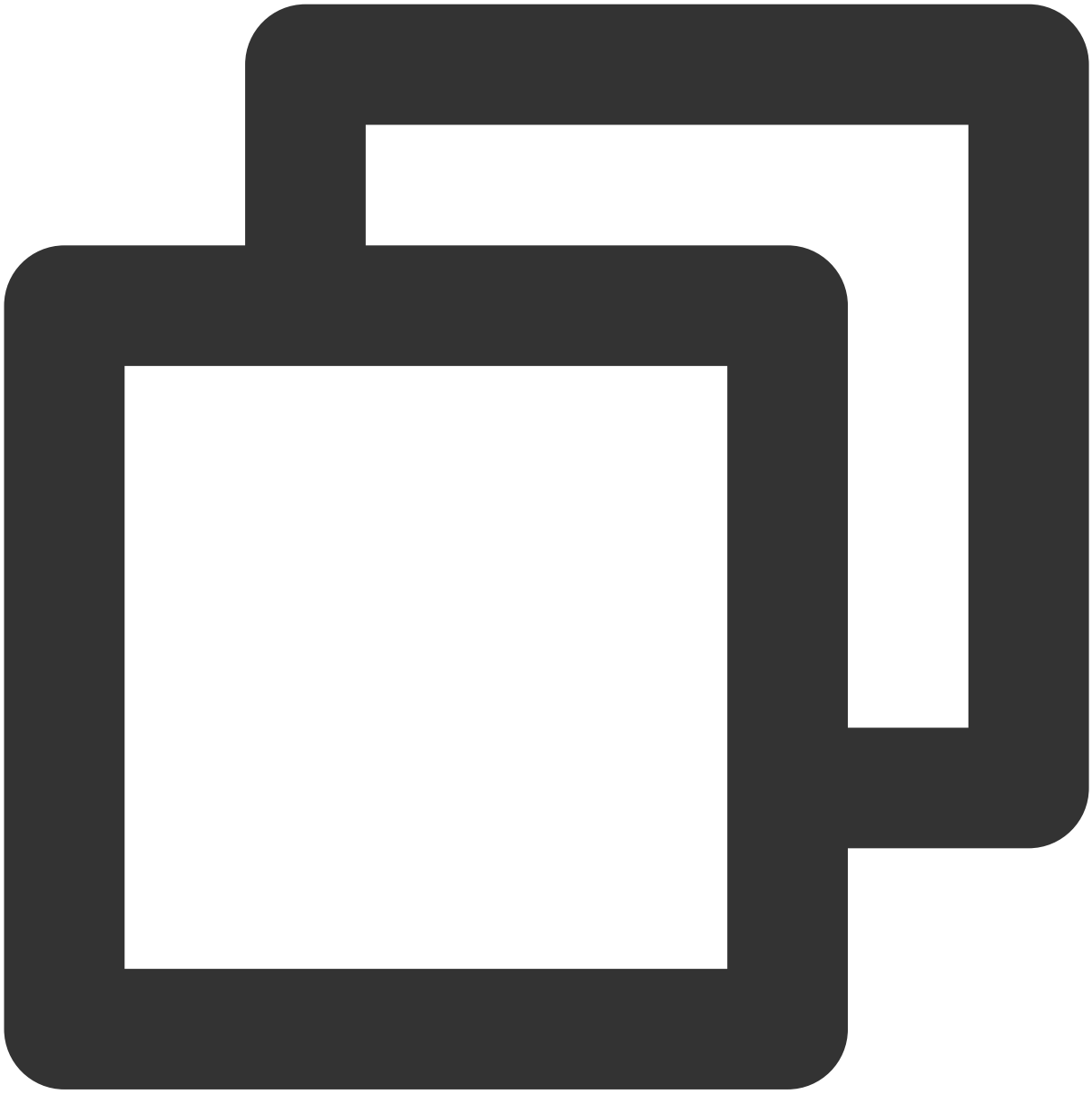
4.3.2 解决方案

本方案基本的思路是减少通信的次数，让事件在渲染层（Webview）响应。小程序的框架分为渲染层（Webview）和逻辑层（App Service），这样分层的目的是管控，开发者的代码只能运行在逻辑层（App Service），而这个思路就必须要让开发者的代码运行在渲染层（Webview）。
使用 WXS 函数用来响应小程序事件，目前只能响应内置组件的事件，不支持自定义组件事件。
WXS 函数的除了纯逻辑的运算，还可以通过封装好的 ComponentDescriptor 实例来访问以及设置组件的 class 和样式，对于交互动画，设置 style 和 class 足够了。

版权所有：腾讯云计算（北京）有限责任公司

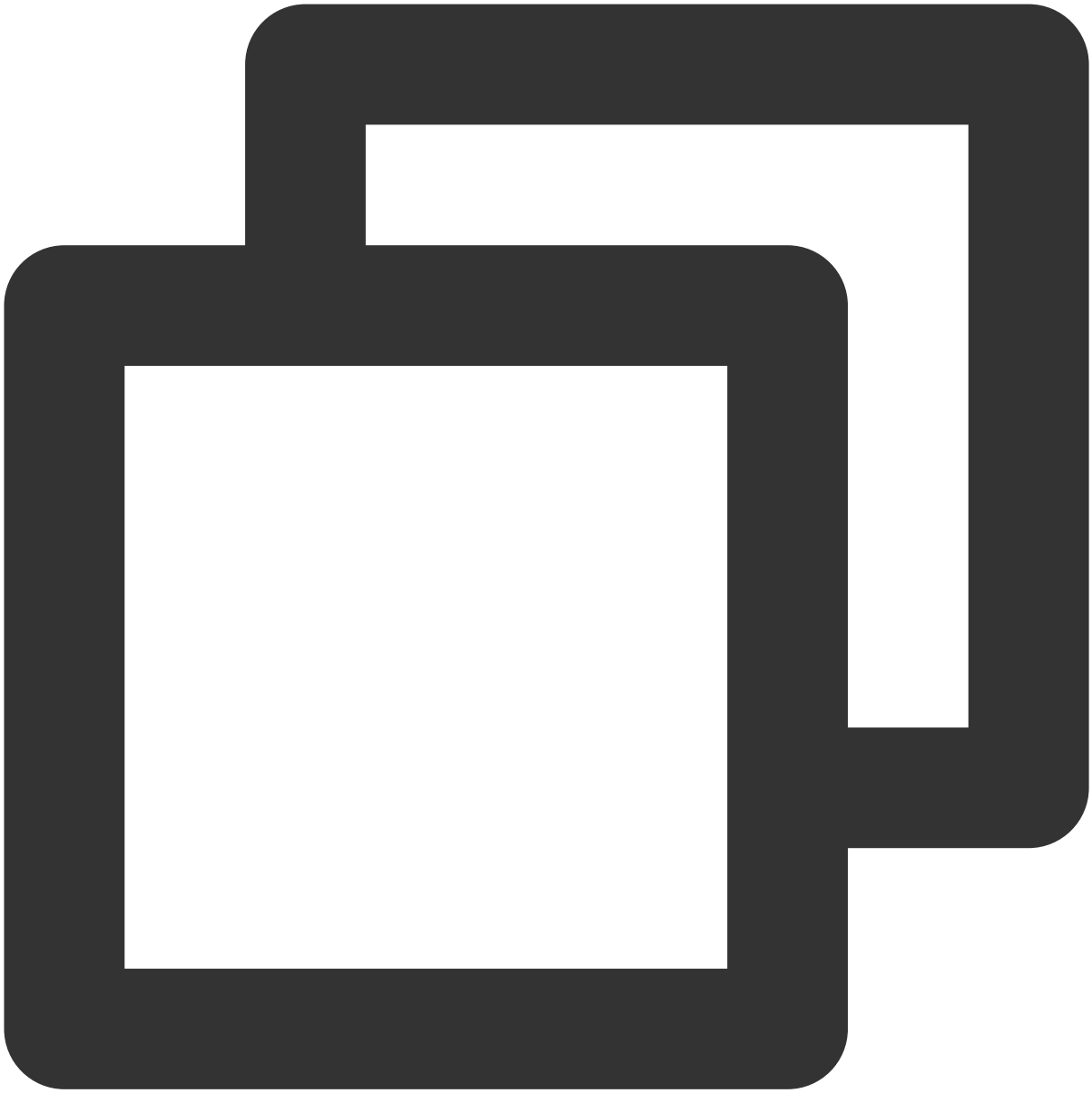
WXS 函数接受2个参数，第一个是 event，在原有的 event 的基础上加了 `event.instance` 对象，第二个参数是 `ownerInstance` 和 `event.instance` 一样是一个 `ComponentDescriptor` 对象。具体使用如下：

在组件中绑定和注册事件处理的 WXS 函数：



```
<wxs module="wxs" src="./test.wxs"></wxs>
<view id="tapTest" data-hi="Weixin" bindtap="{{wxs.tapName}}"> Click me! </view>
**注：绑定的WXS函数必须用{{}}括起来**
```

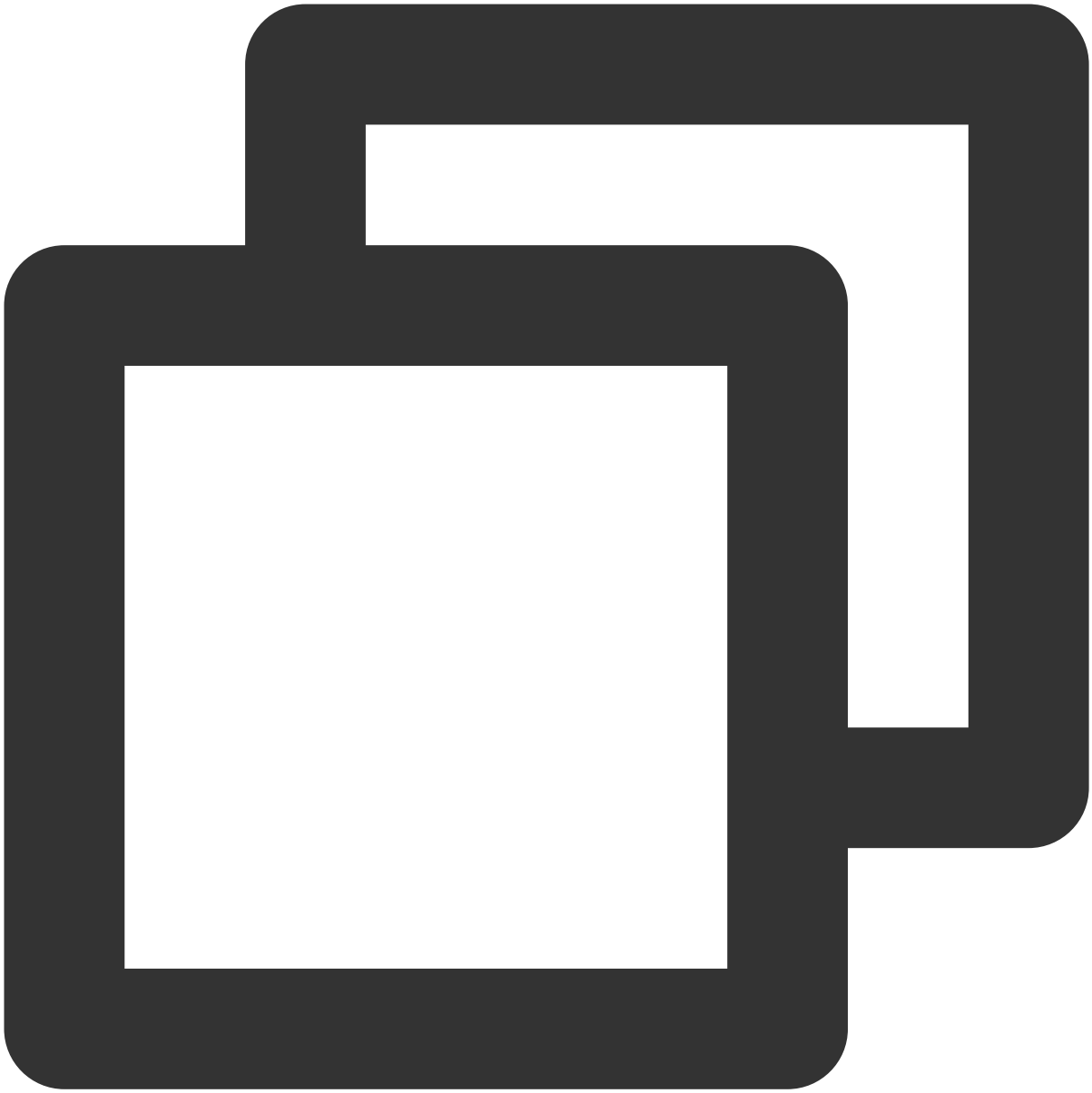
test.wxs 文件实现 tapName 函数：



```
function tapName(event, ownerInstance) {
  console.log('tap Weixin', JSON.stringify(event))
}
module.exports = {
  tapName: tapName
}
```

ownerInstance 包含了一些方法，可以设置组件的样式和 class。

WXS 函数的例子如下：



```
var wxsFunction = function(event, ownerInstance) {
  var instance = ownerInstance.selectComponent('.classSelector') // 返回组件的实例
  instance.setStyle({
      "font-size": "14px" // 支持rpx
  })
  instance.getDataset()
  instance.setClass(className)
  // ...
  return false // 不往上冒泡，相当于调用了同时调用了stopPropagation和preventDefault
}
```

版权所有：腾讯云计算（北京）有限责任公司

其中入参 `event` 是小程序事件对象基础上多了 `event.instance` 来表示触发事件的组件的 `ComponentDescriptor` 实例。

`ownerInstance` 表示的是触发事件的组件所在的组件的 `ComponentDescriptor` 实例，如果触发事件的组件是在页面内的，则 `ownerInstance` 表示的是页面实例。

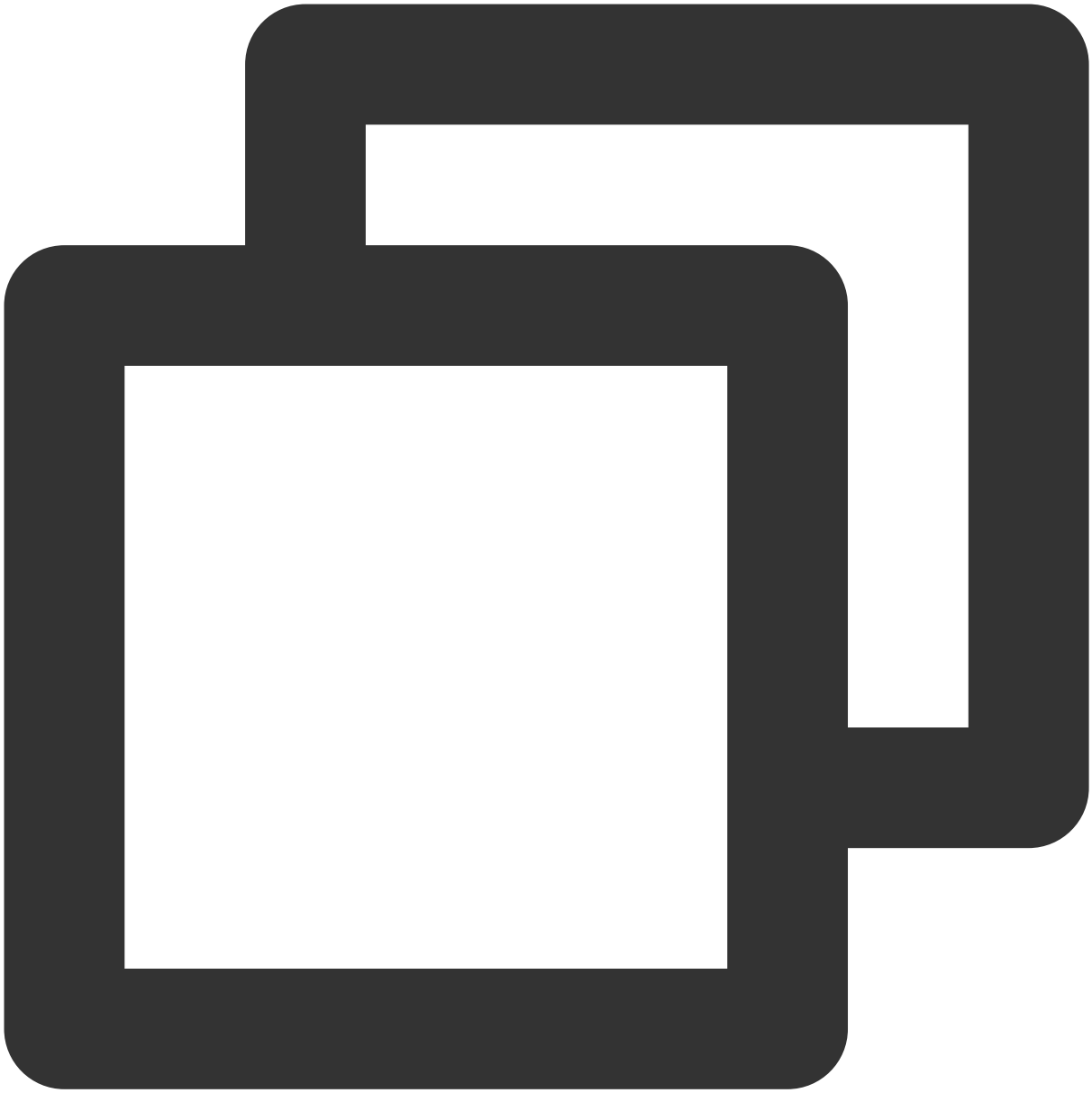
`ComponentDescriptor` 目前支持的 API 如下:

方法	参数	描述
selectComponent	selector 对象	返回组件的 `ComponentDescriptor` 实例。
selectAllComponents	selector对象数组	返回组件的 `ComponentDescriptor` 实例数组。
setStyle	Object/string	设置组件样式，支持 `rpx` 。设置的样式优先级比组件 wxml 里面定义的样式高。不能设置最顶层页面的样式。
addClass/removeClass/hasClass	string	设置组件的 class。设置的 class 优先级比组件 wxml 里面定义的 class 高。不能设置最顶层页面的 class。
getDataset	无	返回当前组件/页面的 dataset 对象。
callMethod	(funcName:string, args:object)	调用当前组件/页面在逻辑层（App Service）定义的函数。funcName 表示函数名称，args 表示函数的参数。
getState	无	返回一个 object 对象，当有局部变量需要存储起来后续使用的时候用这个方法。
triggerEvent	(eventName, detail)	和组件的triggerEvent一致。

4.3.3 使用方法

WXML 定义事件：

版权所有：腾讯云计算（北京）有限责任公司



```
<wxs module="test" src="./test.wxs"></wxs>
<view
  change:prop="{{test.propObserver}}"
  prop="{{propValue}}"
  bindtouchmove="{{test.touchmove}}"
  class="movable">
</view>
```

上面的 change:prop（属性前面带 `change:` 前缀）是在 prop 属性被设置的时候触发 WXS 函数，值必须用 `{{}}` 括起来。类似 Component 定义的 properties 里面的 observer 属性，在 `setData({propValue:`

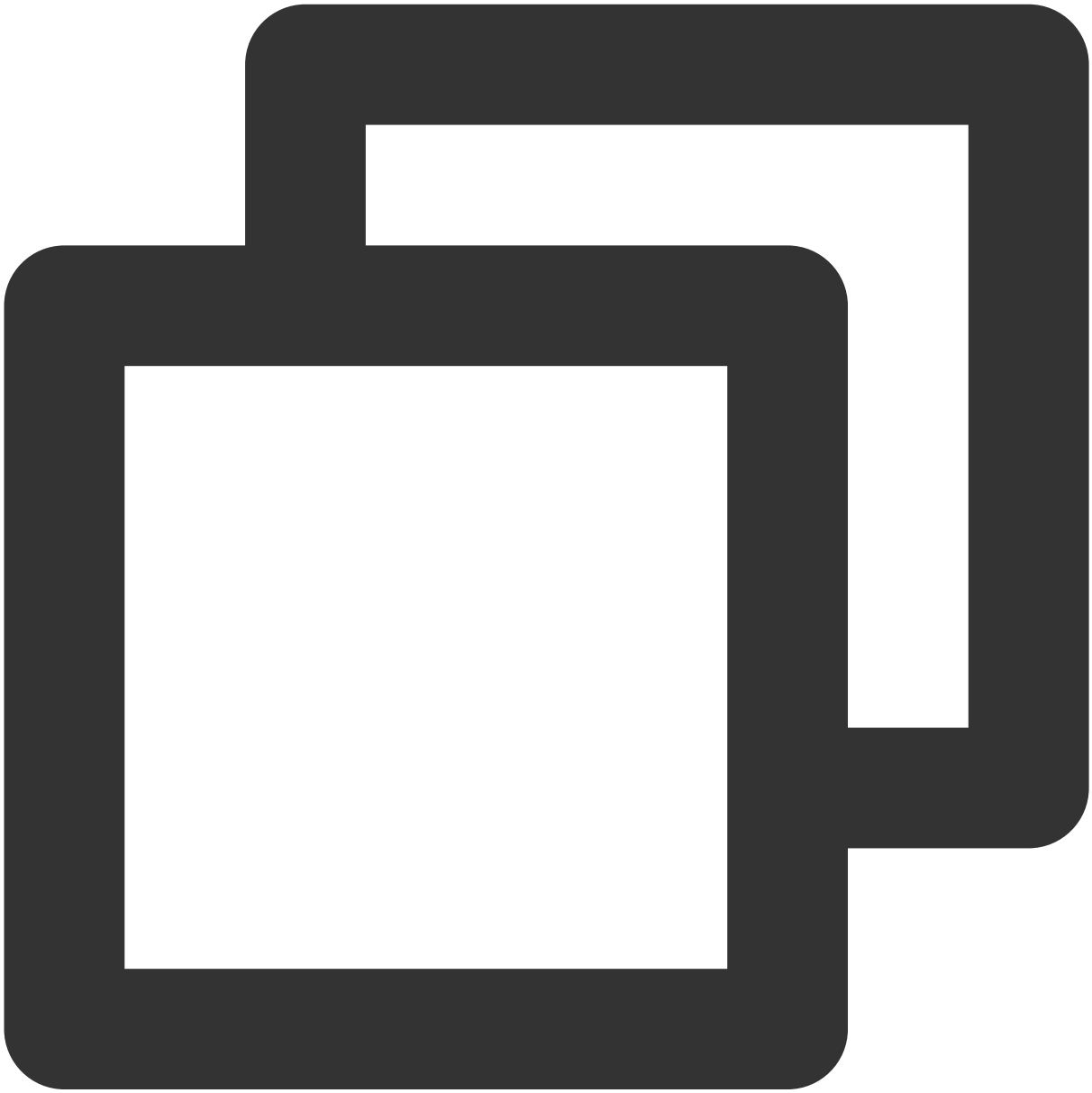
版权所有：腾讯云计算（北京）有限责任公司

`newValue})` 调用之后会触发。

注意：

WXS 函数必须用 `{{}}` 括起来。当 prop 的值被设置 WXS 函数就会触发，而不只是值发生改变，所以在页面初始化的时候会调用一次 WxsPropObserver 的函数。

WXS 文件 `test.wxs` 里面定义并导出事件处理函数和属性改变触发的函数：



```
module.exports = {
  touchmove: function(event, instance) {
      console.log('log event', JSON.stringify(event))
  },
```

```
    propObserver: function(newValue, oldValue, ownerInstance, instance) {
        console.log('prop observer', newValue, oldValue)
    }
}
```

4.4 事件详解

4.4.1 事件分类

事件分为冒泡事件和非冒泡事件：

冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递；

非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML的冒泡事件列表：

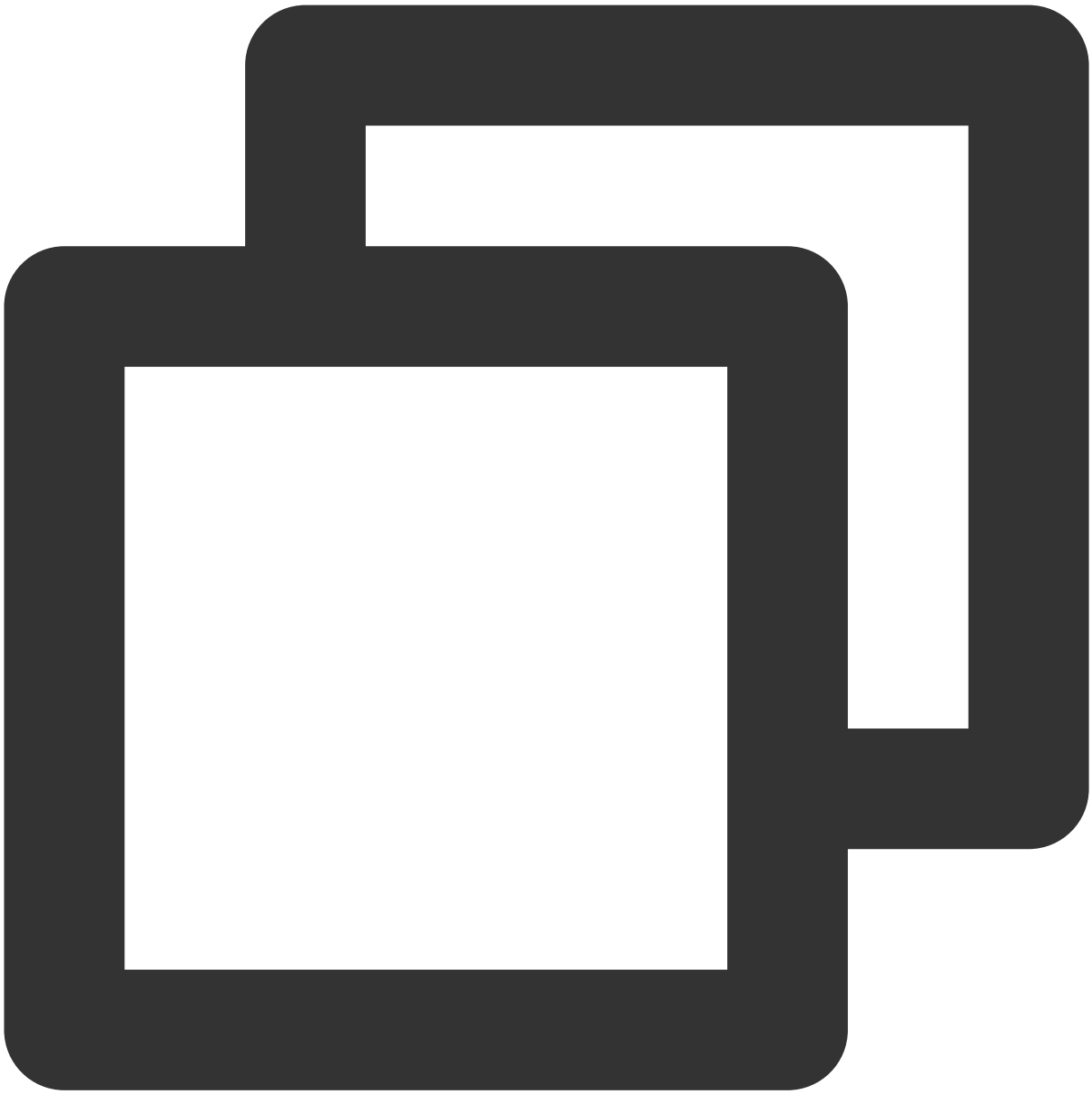
类型	触发条件
touchstart	手指触摸动作开始
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断，如来电提醒，弹窗
touchend	手指触摸动作结束
tap	手指触摸后马上离开
longpress	手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap 事件将不被触发
longtap	手指触摸后，超过350ms再离开（推荐使用 longpress 事件代替）
transitionend	会在 WXSS transition 或 wx.createAnimation 动画结束后触发
animationstart	会在一个 WXSS animation 动画开始时触发
animationiteration	会在一个 WXSS animation 一次迭代结束时触发
animationend	会在一个 WXSS animation 动画完成时触发
touchforcechange	在支持 3D Touch 的 iPhone 设备，重按时会触发

说明：

除上表之外的其他组件自定义事件如无特殊声明都是非冒泡事件，如 form 的submit事件，input 的input事件，scroll-view 的scroll事件，(详见 组件概览)

4.4.2 普通事件绑定

事件绑定的写法类似于组件的属性，如：

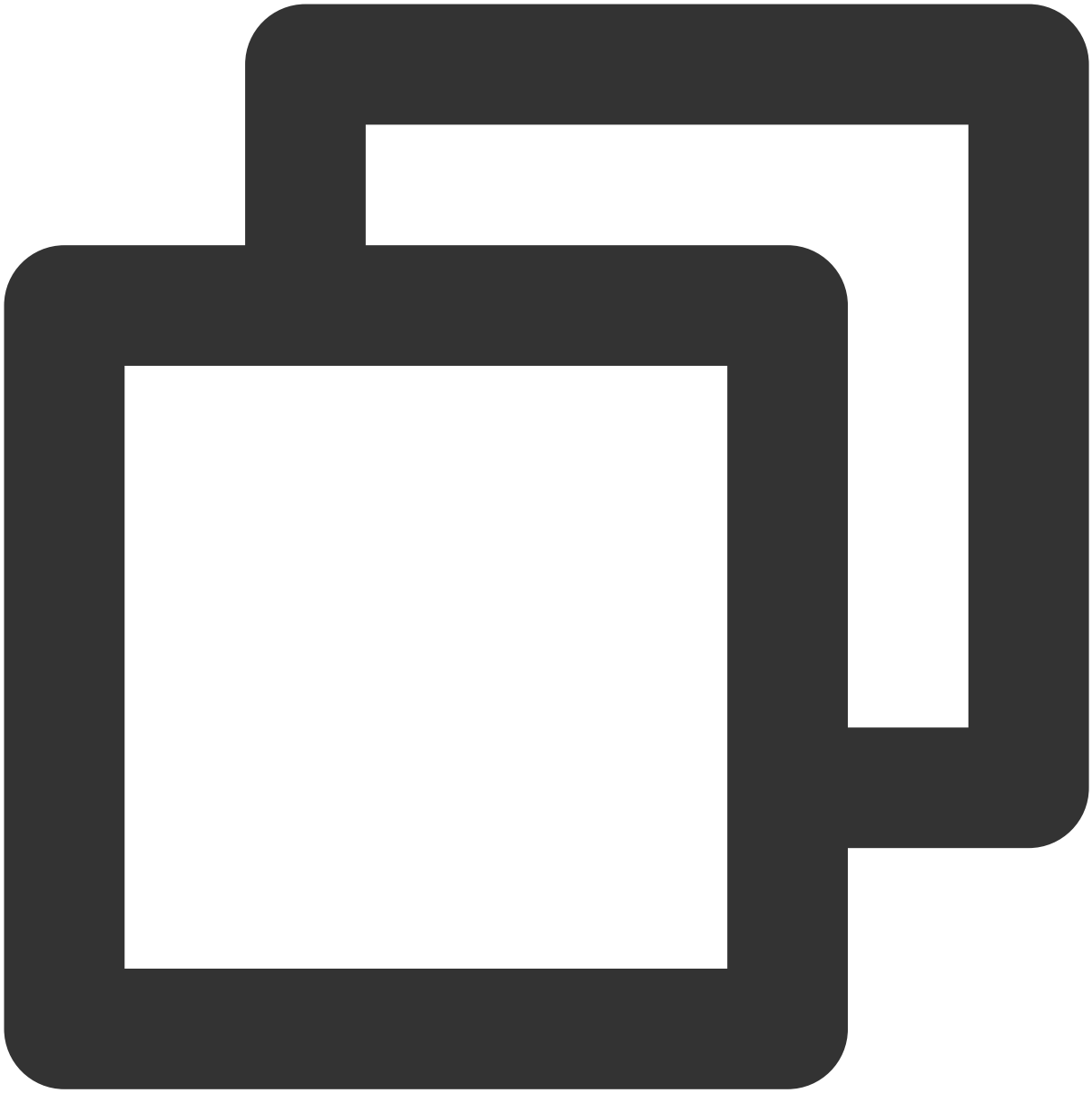


```
<view bindtap="handleTap">
    Click here!
</view>
```

如果用户点击这个 view，则页面的 `handleTap` 会被调用。

事件绑定函数可以是一个数据绑定，如：

版权所有：腾讯云计算（北京）有限责任公司



```
<view bindtap="{{ handlerName }}">
  Click here!
</view>
```

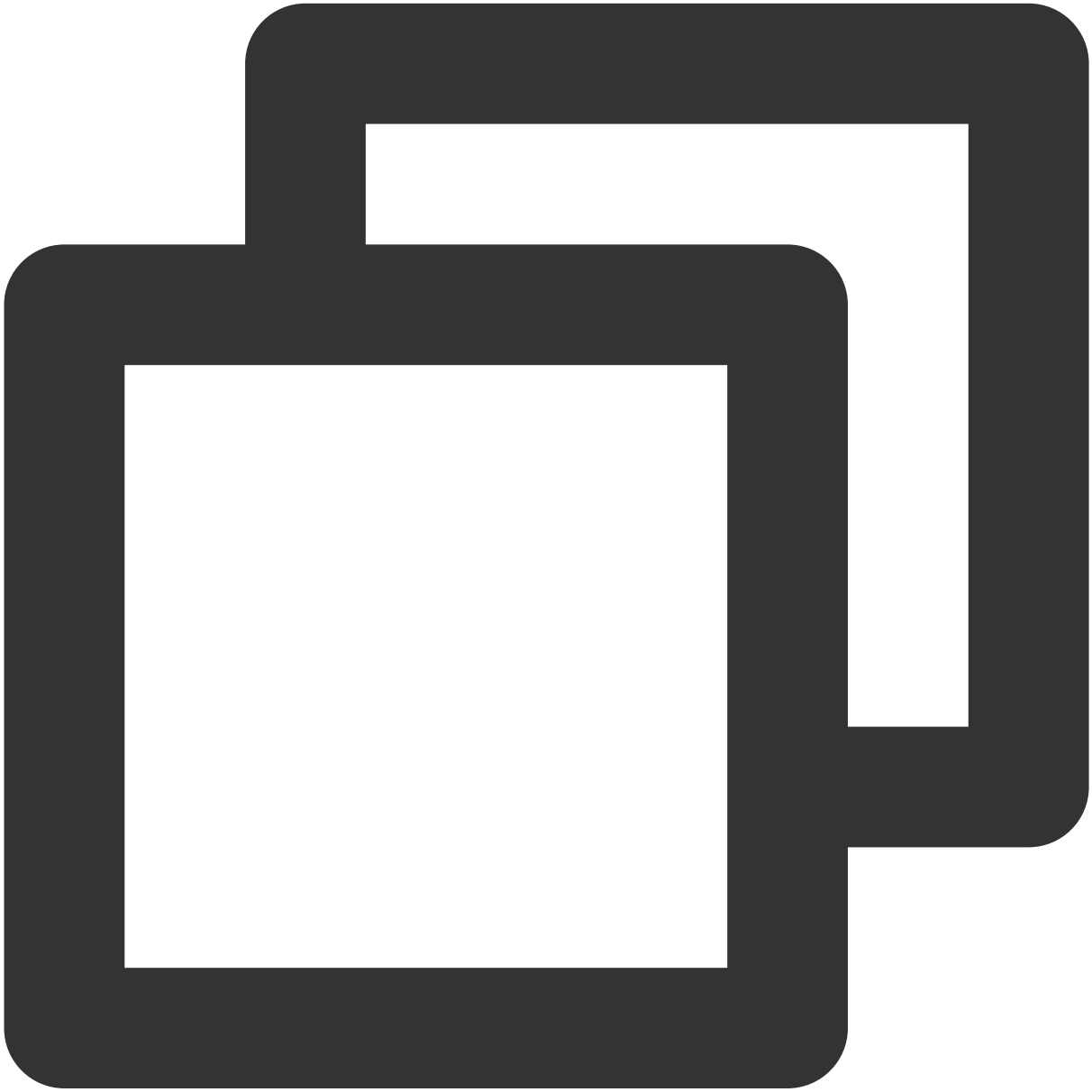
此时，页面的 `this.data.handlerName` 必须是一个字符串，指定事件处理函数名；如果它是个空字符串，则这个绑定会失效（可以利用这个特性来暂时禁用一些事件）。

在大多数组件和自定义组件中， `bind` 后可以紧跟一个冒号，其含义不变，如 `bind:tap`

4.4.3 绑定并阻止事件冒泡

除 `bind` 外，也可以用 `catch` 来绑定事件。与 `bind` 不同， `catch` 会阻止事件向上冒泡。

例如在下边这个例子中，单击 inner view 会先后调用 `handleTap3` 和 `handleTap2` (因为 tap 事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递)，单击 middle view 会触发 `handleTap2` ，单击 outer view 会触发 `handleTap1` 。



```
<view id="outer" bindtap="handleTap1">
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">
```

版权所有：腾讯云计算（北京）有限责任公司

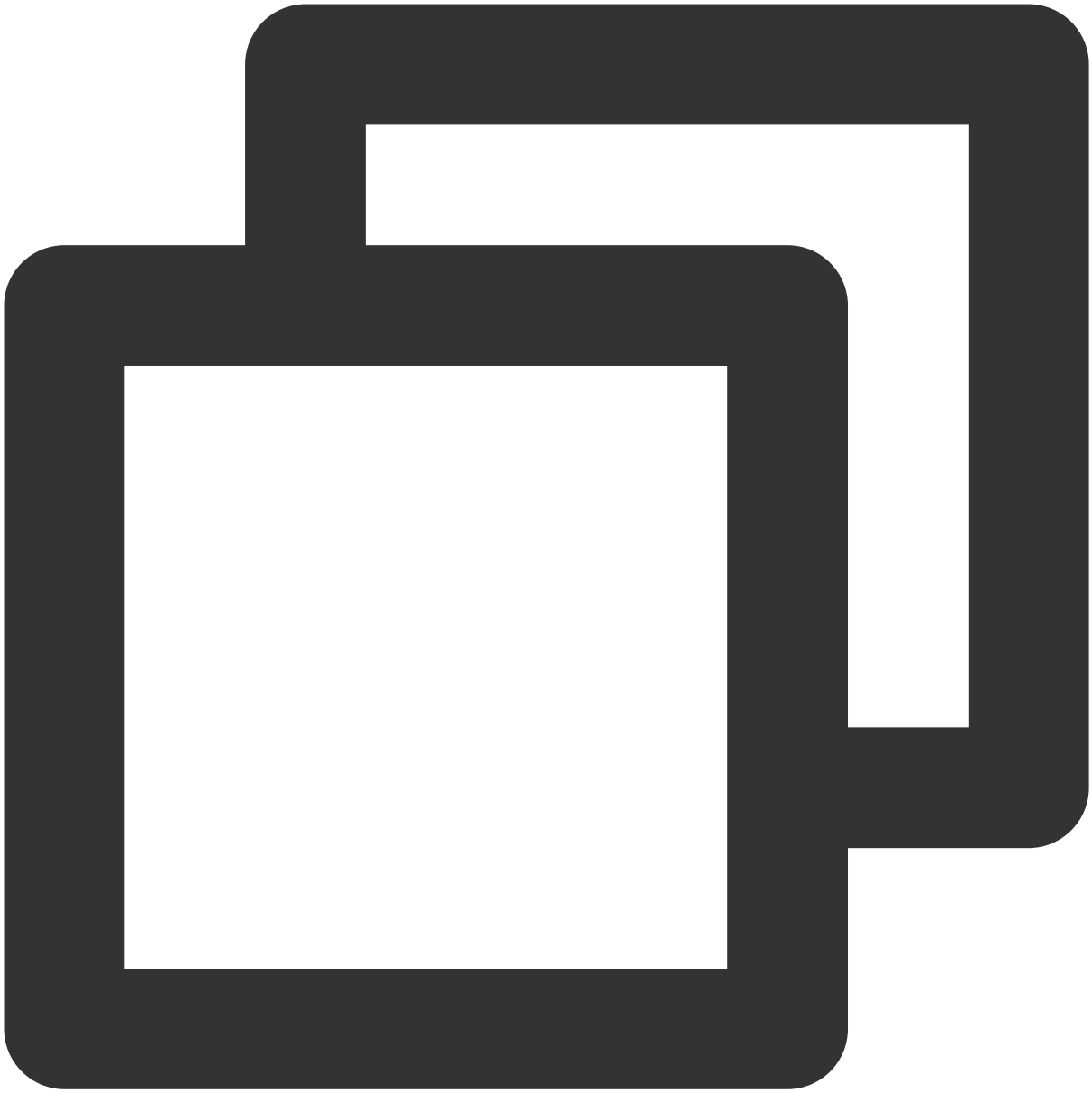
```
      inner view
    </view>
  </view>
</view>
```

4.4.4 互斥事件绑定

自基础库版本 1.5.44 起，除 `bind` 和 `catch` 外，还可以使用 `mut-bind` 来绑定事件。一个 `mut-bind` 触发后，如果事件冒泡到其他节点上，其他节点上的 `mut-bind` 绑定函数不会被触发，但 `bind` 绑定函数和 `catch` 绑定函数依旧会被触发。

换而言之，所有 `mut-bind` 是“互斥”的，只会有其中一个绑定函数被触发。同时，它完全不影响 `bind` 和 `catch` 的绑定效果。

例如在下边这个例子中，单击 inner view 会先后调用 `handleTap3` 和 `handleTap2` ，单击 middle view 会调用 `handleTap2` 和 `handleTap1` 。



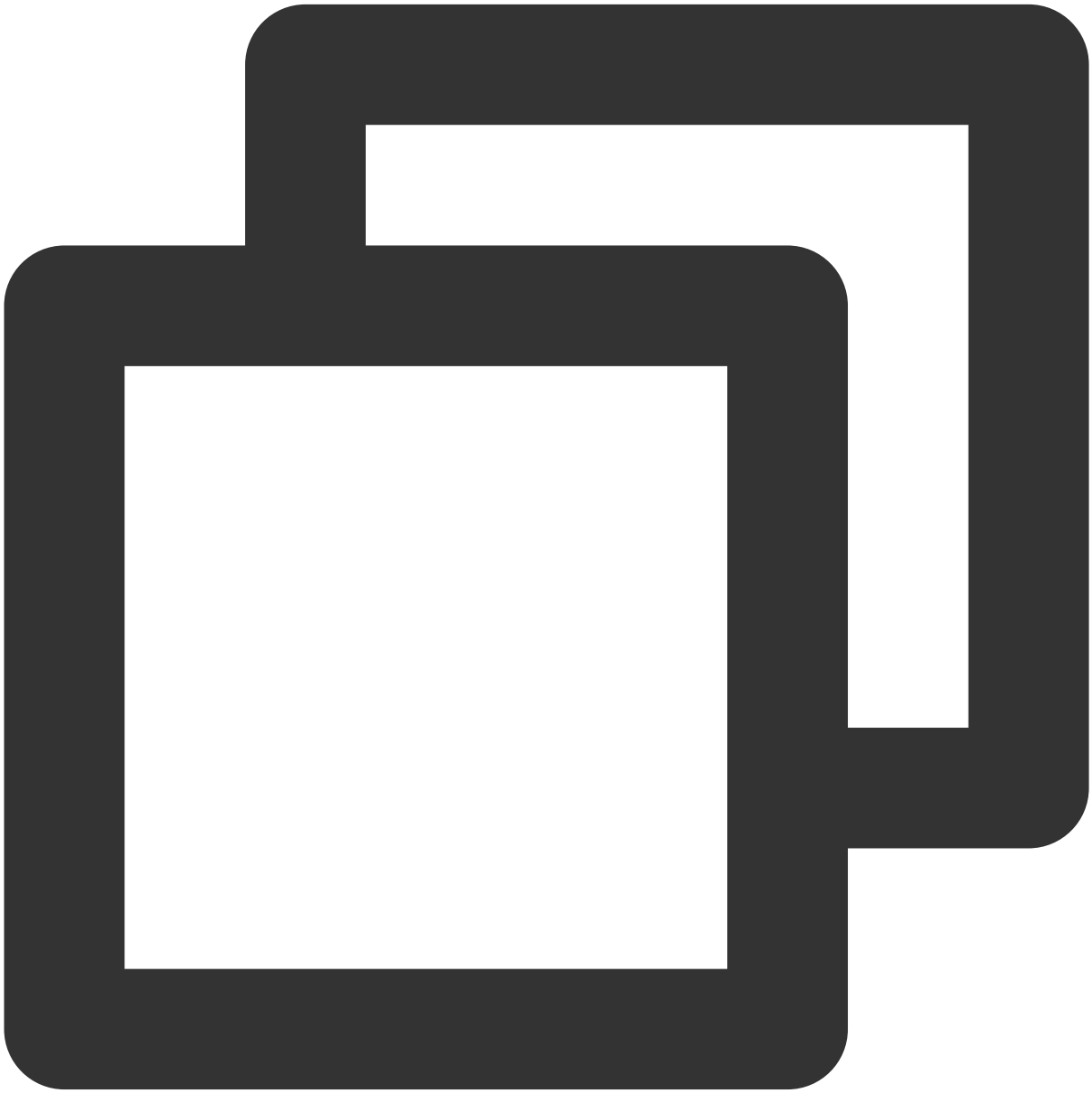
```
<view id="outer" mut-bind:tap="handleTap1">
  outer view
  <view id="middle" bindtap="handleTap2">
    middle view
    <view id="inner" mut-bind:tap="handleTap3">
      inner view
    </view>
  </view>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司

4.4.5 事件的捕获阶段

触摸类事件支持捕获阶段。捕获阶段位于冒泡阶段之前，且在捕获阶段中，事件到达节点的顺序与冒泡阶段恰好相反。需要在捕获阶段监听事件时，可以采用 `capture-bind` 、 `capture-catch` 关键字，后者将中断捕获阶段和取消冒泡阶段。

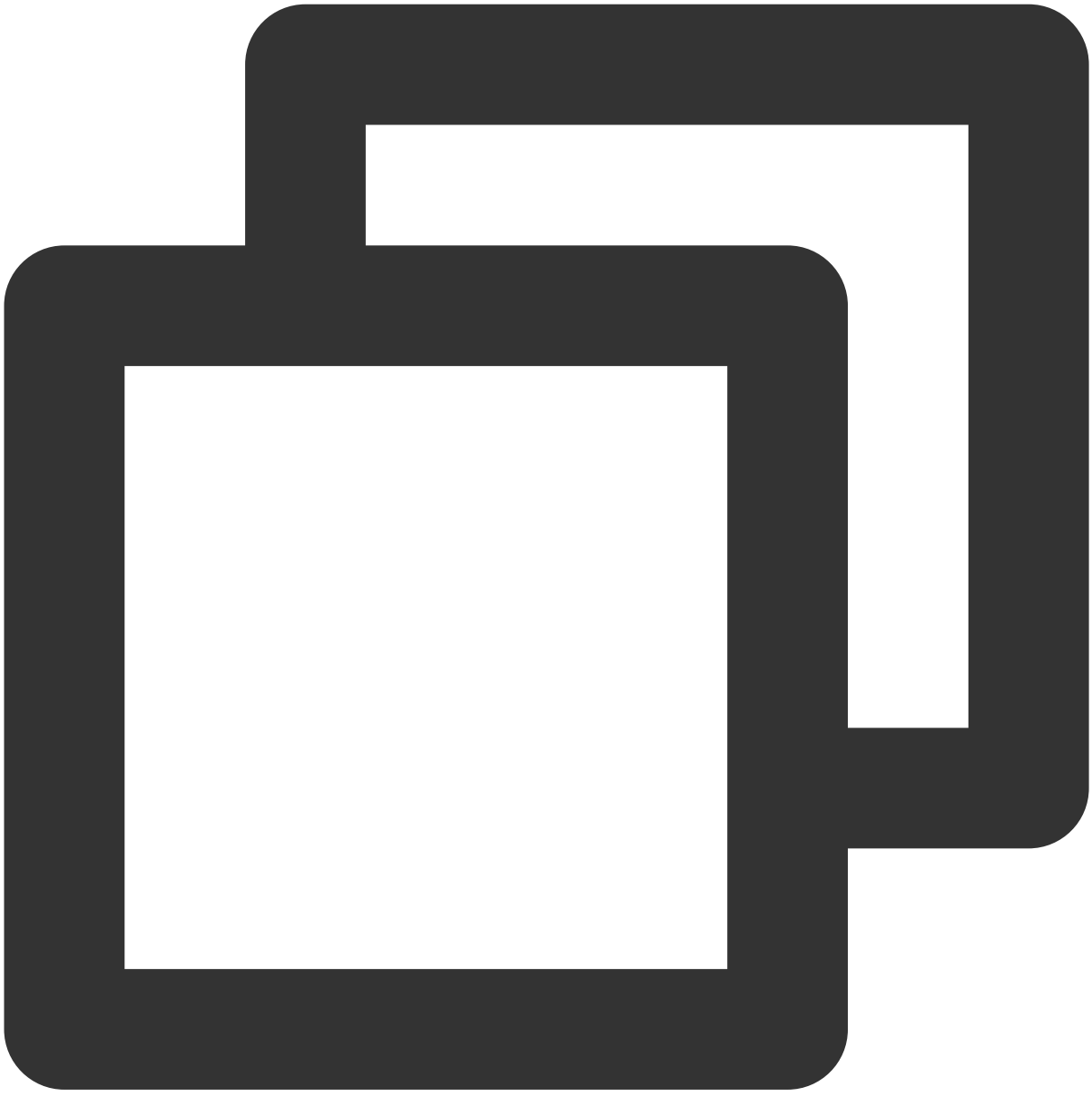
在下面的代码中，单击 inner view 会先后调用 `handleTap2` 、 `handleTap4` 、 `handleTap3` 、 `handleTap1` 。



```
<view id="outer" bind:touchstart="handleTap1" capture-bind:touchstart="handleTap2">
  outer view
  <view id="inner" bind:touchstart="handleTap3" capture-bind:touchstart="handleTap4
```

```
    inner view
  </view>
</view>
```

如果将上面代码中的第一个 capture-bind 改为 capture-catch，将只触发 handleTap2。



```
<view id="outer" bind:touchstart="handleTap1" capture-catch:touchstart="handleTap2"
  outer view
  <view id="inner" bind:touchstart="handleTap3" capture-bind:touchstart="handleTap4
    inner view
  </view>
</view>
```

4.4.6 事件对象

如无特殊说明，当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。

BaseEvent 基础事件对象属性列表：

属性	类型	说明
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳
target	Object	触发事件的组件的一些属性值集合
currentTarget	Object	当前组件的一些属性值集合

CustomEvent 自定义事件对象属性列表（继承 BaseEvent）：

属性	类型	说明
detail	Object	额外的信息

TouchEvent 触摸事件对象属性列表（继承 BaseEvent）：

属性	类型	说明
touches	Array	触摸事件，当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	触摸事件，当前变化的触摸点信息的数组

特殊事件：canvas 中的触摸事件不可冒泡，所以没有 currentTarget。

4.4.7 type

代表事件的类型。

4.4.8 timeStamp

页面打开到触发事件所经过的毫秒数。

4.4.9 target

触发事件的源组件。

属性	类型	说明

版权所有：腾讯云计算（北京）有限责任公司

id	String	事件源组件的 id
dataset	Object	事件源组件上由 data - 开头的自定义属性组成的集合

4.4.10 currentTarget

事件绑定的当前组件。

属性	类型	说明
id	String	当前组件的 id
dataset	Object	当前组件上由 data - 开头的自定义属性组成的集合

说明：

target 和 currentTarget 可以参考上例中，单击 inner view 时，handleTap3 收到的事件对象 target 和 currentTarget 都是 inner，而 handleTap2 收到的事件对象 target 就是 inner，currentTarget 就是 middle。

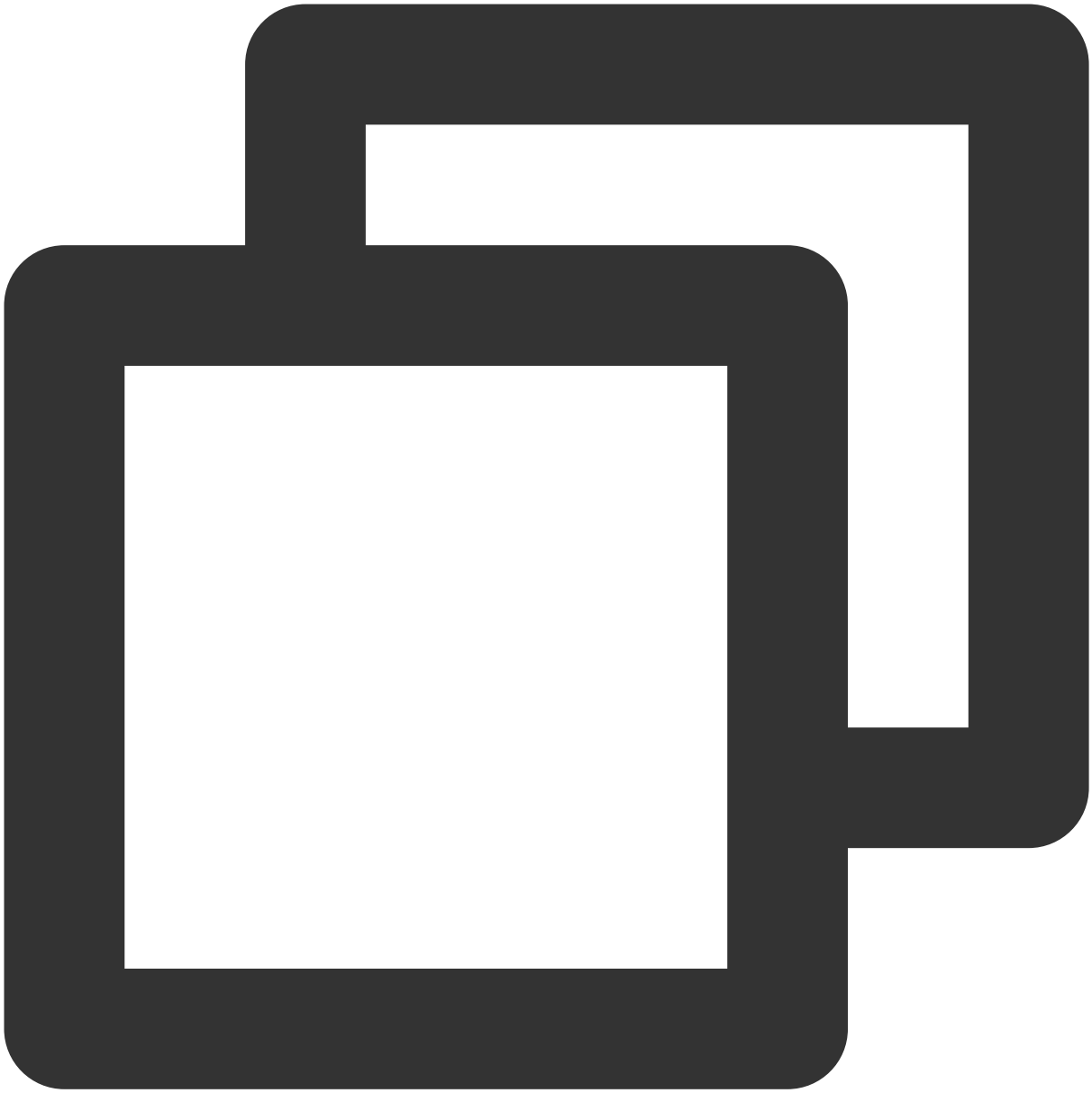
4.4.11 dataset

在组件节点中可以附加一些自定义数据。这样，在事件中可以获取这些自定义的节点数据，用于事件的逻辑处理。在 WXML 中，这些自定义数据以 data- 开头，多个单词由连字符 - 连接。这种写法中，连字符写法会转换成驼峰写法，而大写字符会自动转成小写字符。如：

data-element-type ，最终会呈现为 event.currentTarget.dataset.elementType ；

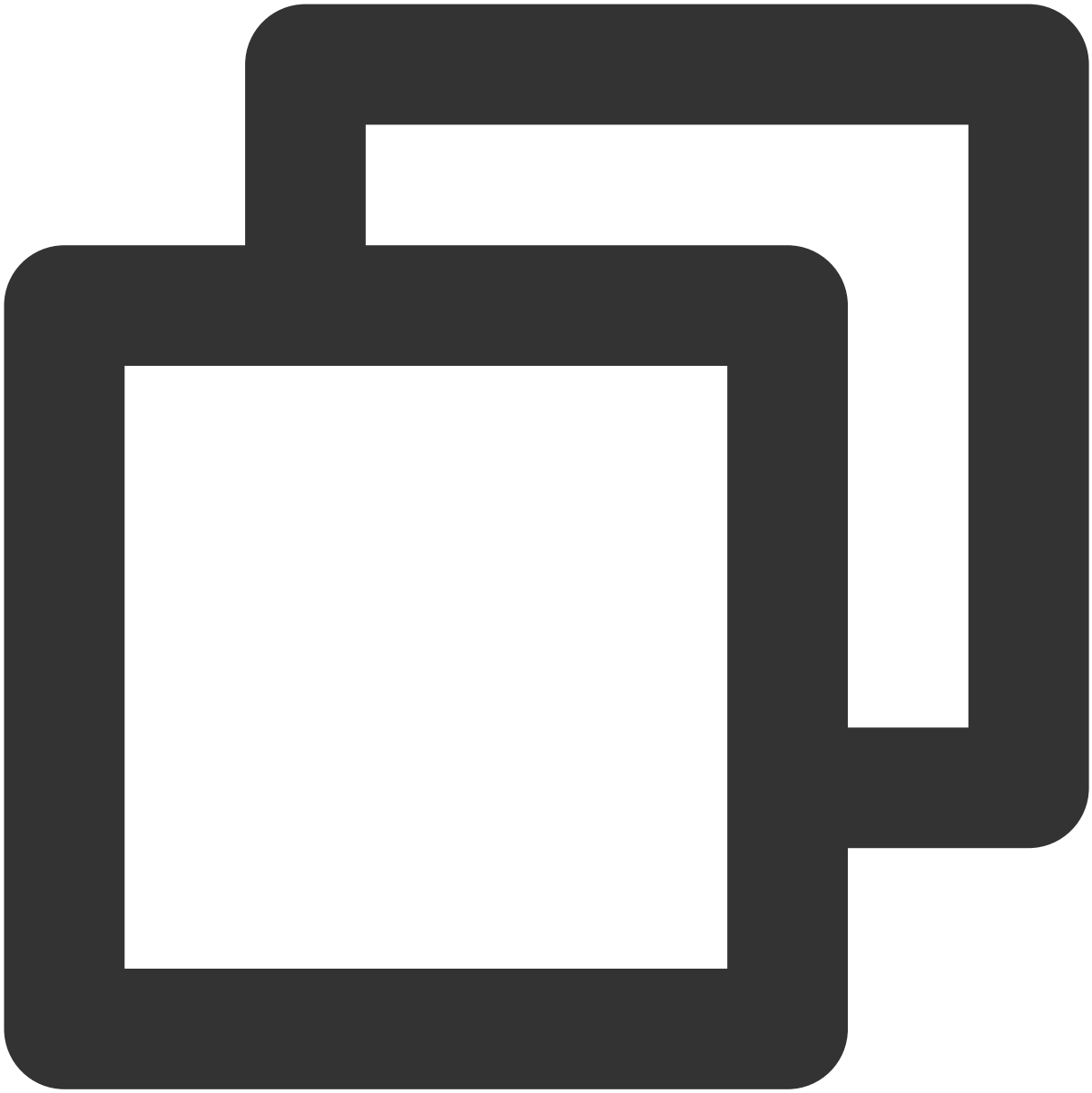
data-elementType ，最终会呈现为 event.currentTarget.dataset.elementtype 。

代码示例：



```
<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap"> DataSet Test </
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  bindViewTap:function(event){
    event.currentTarget.dataset.alphaBeta === 1 // - 会转为驼峰写法
    event.currentTarget.dataset.alphabeta === 2 // 大写会转为小写
  }
})
```

4.4.12 touches

touches 是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。表示当前停留在屏幕上的触摸点。

Touch 对象

属性	类型	说明
identifier	Number	触摸点的标识符
pageX, pageY	Number	距离文档左上角的距离，文档的左上角为原点，横向为X轴，纵向为Y轴
clientX, clientY	Number	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为X轴，纵向为Y轴

CanvasTouch对象

属性	类型	说明
identifier	Number	触摸点的标识符
x, y	Number	距离 Canvas 左上角的距离，Canvas 的左上角为原点，横向为X轴，纵向为Y轴

4.4.13 changedTouches

changedTouches 数据格式同 touches。 表示有变化的触摸点，如从无变有（touchstart），位置变化（touchmove），从有变无（touchend、touchcancel）。

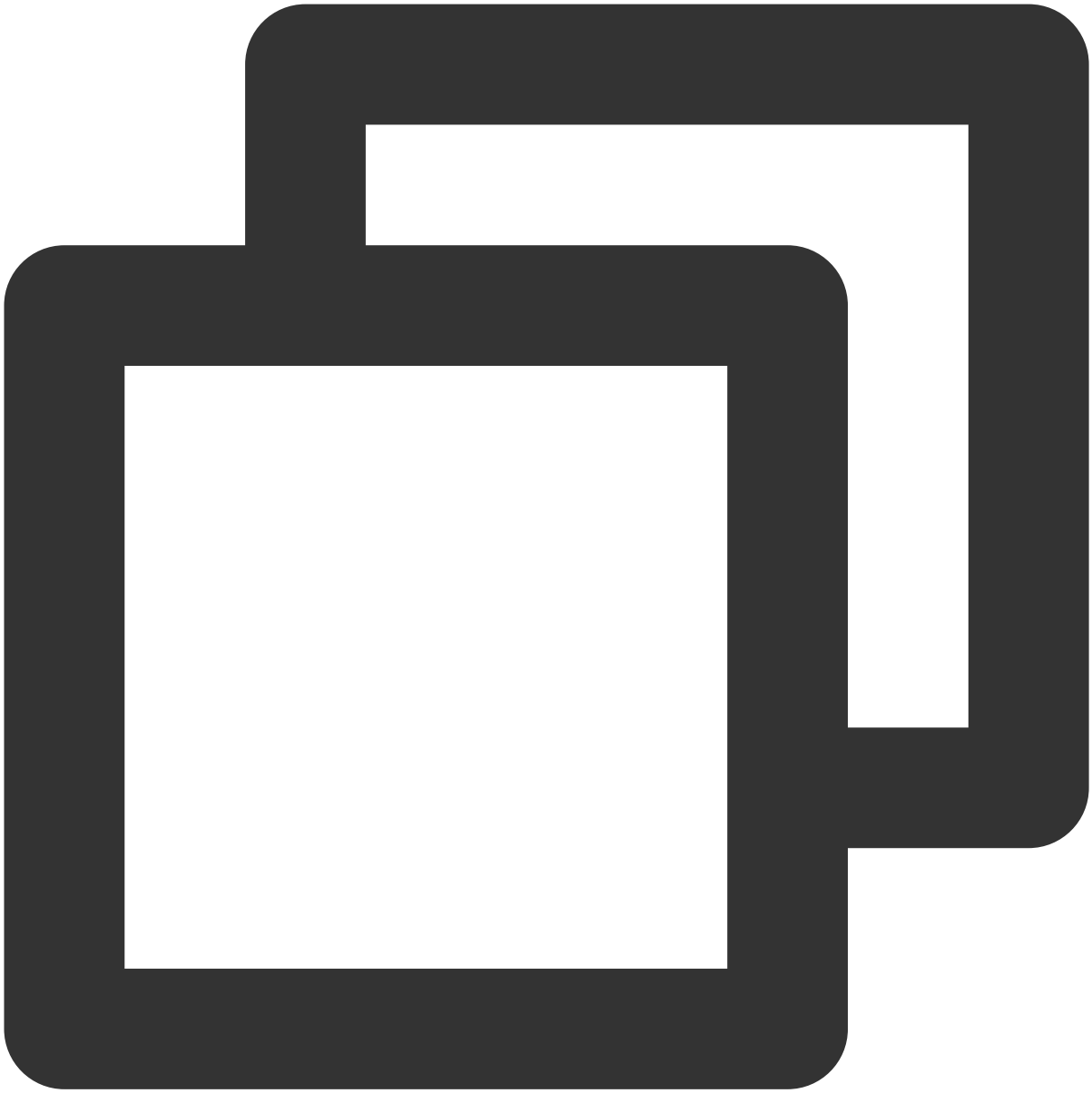
4.4.14 detail

自定义事件所携带的数据，如表单组件的提交事件会携带用户的输入，媒体的错误事件会携带错误信息，详见 组件概览 定义中各个事件的定义。
单击事件的 `detail` 带有的 x, y 同 pageX, pageY 代表距离文档左上角的距离。

5. 基础组件

框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。详细介绍请参考 组件概览。
什么是组件：
组件是视图层的基本组成单元。
一个组件通常包括 `开始标签` 和 `结束标签` ， `属性` 用来修饰这个组件， `内容` 在两个标签之内。

版权所有：腾讯云计算（北京）有限责任公司



```
<tagname property="value">
Content goes here ...
</tagname>
```

注意：

所有组件与属性都是小写，以连字符-连接。

属性类型：

类型	描述	注解
Boolean	布尔值	组件写上该属性，不管是什么值都被当作 `true` ；只有组件上没有该属性

版权所有：腾讯云计算（北京）有限责任公司

		时，属性值才为 `false` 。 如果属性值为变量，变量的值会被转换为 Boolean 类型。
Number	数字	1, 2.5
String	字符串	"string"
Array	数组	[1, "string"]
Object	对象	{ key: value }
EventHandler	事件处理函数名	"handlerName" 是 Page 中定义的事件处理函数名
Any	任意属性	-

公共属性

所有组件都有以下属性：

属性名	类型	描述	注解
id	String	组件的唯一标示	保持整个页面唯一
class	String	组件的样式类	在对应的 WXSS 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data-*	Any	自定义属性	组件上触发的事件时，会发送给事件处理函数
bind* / catch*	EventHandler	组件的事件	详见事件

特殊属性

几乎所有组件都有各自定义的属性，可以对该组件的功能或样式进行修饰，请参考各个 组件概览 的定义。

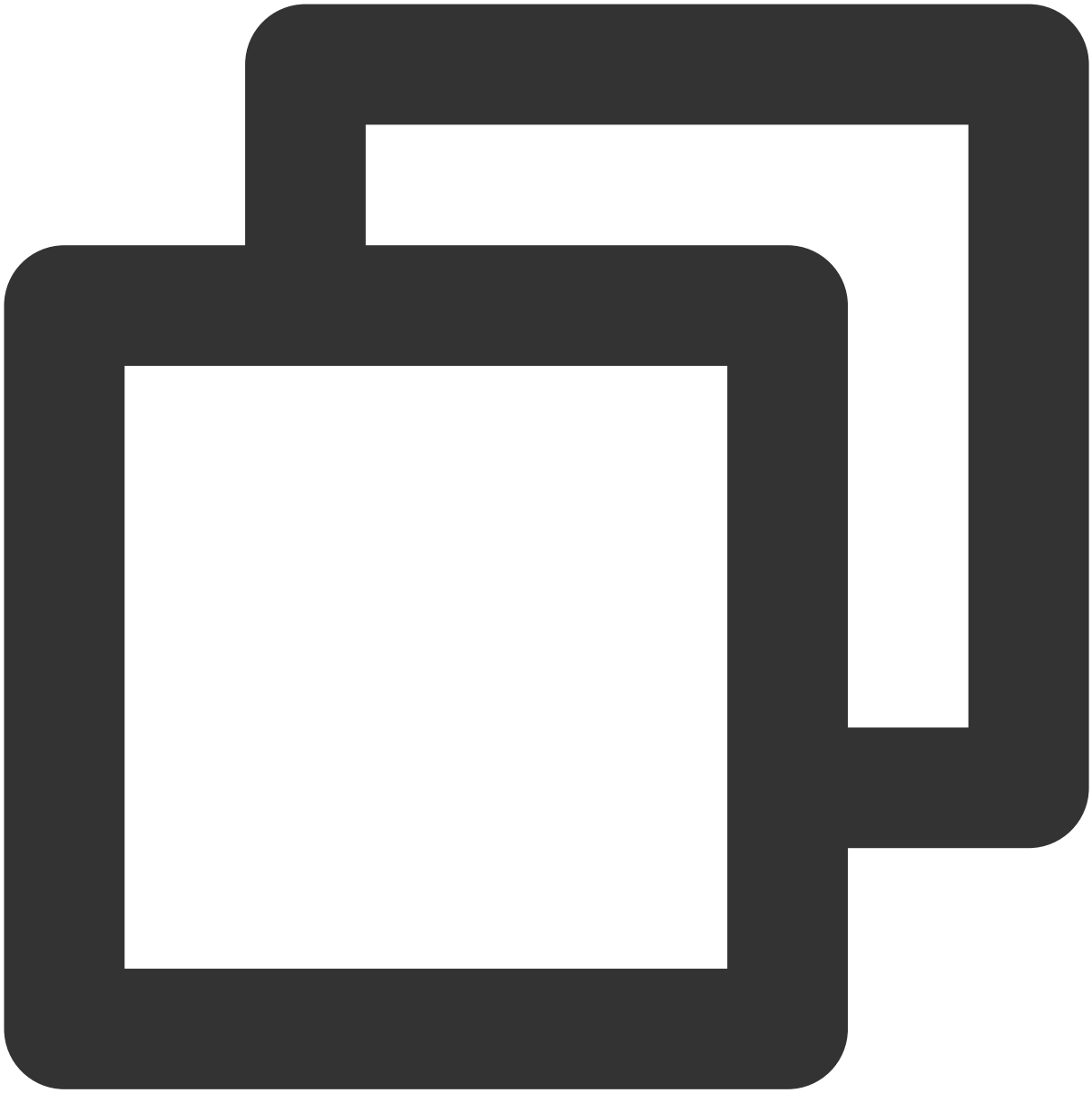
6. 获取界面上的节点信息

6.1 WXML节点信息

节点信息查询 API 可以用于获取节点属性、样式、在界面上的位置等信息。

最常见的用法是使用这个接口来查询某个节点的当前位置，以及界面的滚动位置。

示例代码：



```
const query = wx.createSelectorQuery()
query.select('#the-id').boundingClientRect(function(res){
  res.top // #the-id 节点的上边界坐标（相对于显示区域）
})
query.selectViewport().scrollOffset(function(res){
  res.scrollTop // 显示区域的竖直滚动位置
})
query.exec()
```

版权所有：腾讯云计算（北京）有限责任公司

上述示例中，`#the-id` 是一个节点选择器，与 CSS 的选择器相近但略有区别，请参见 SelectorQuery.select 的相关说明。

在自定义组件或包含自定义组件的页面中，推荐使用 this.createSelectorQuery 来代替 wx.createSelectorQuery，这样可以确保在正确的范围内选择节点。

6.2 WXML节点布局相交状态

节点布局相交状态 API 可用于监听两个或多个组件节点在布局位置上的相交状态。这一组 API 常常可以用于推断某些节点是否可以被用户看见、有多大比例可以被用户看见。

这一组 API 涉及的主要概念如下。

参照节点：监听的参照节点，取它的布局区域作为参照区域。如果有多个参照节点，则会取它们布局区域的 交集 作为参照区域。页面显示区域也可作为参照区域之一；

目标节点：监听的目标，默认只能是一个节点（使用 selectAll 选项时，可以同时监听多个节点）；

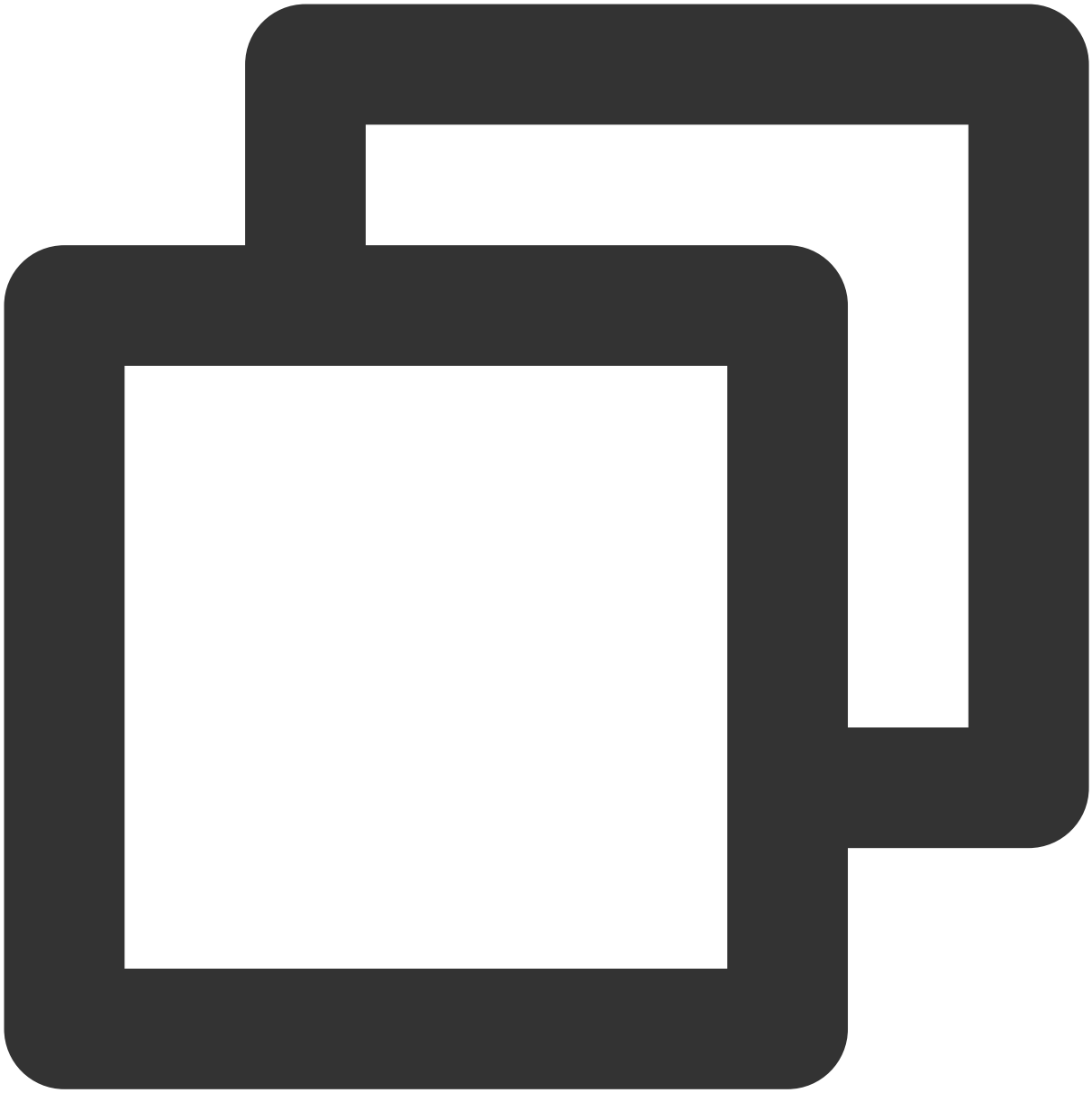
相交区域：目标节点的布局区域与参照区域的相交区域；

相交比例：相交区域占参照区域的比例；

阈值：相交比例如果达到阈值，则会触发监听器的回调函数。阈值可以有多个；

以下示例代码可以在目标节点（用选择器 `.target-class` 指定）每次进入或离开页面显示区域时，触发回调函数。

示例代码：



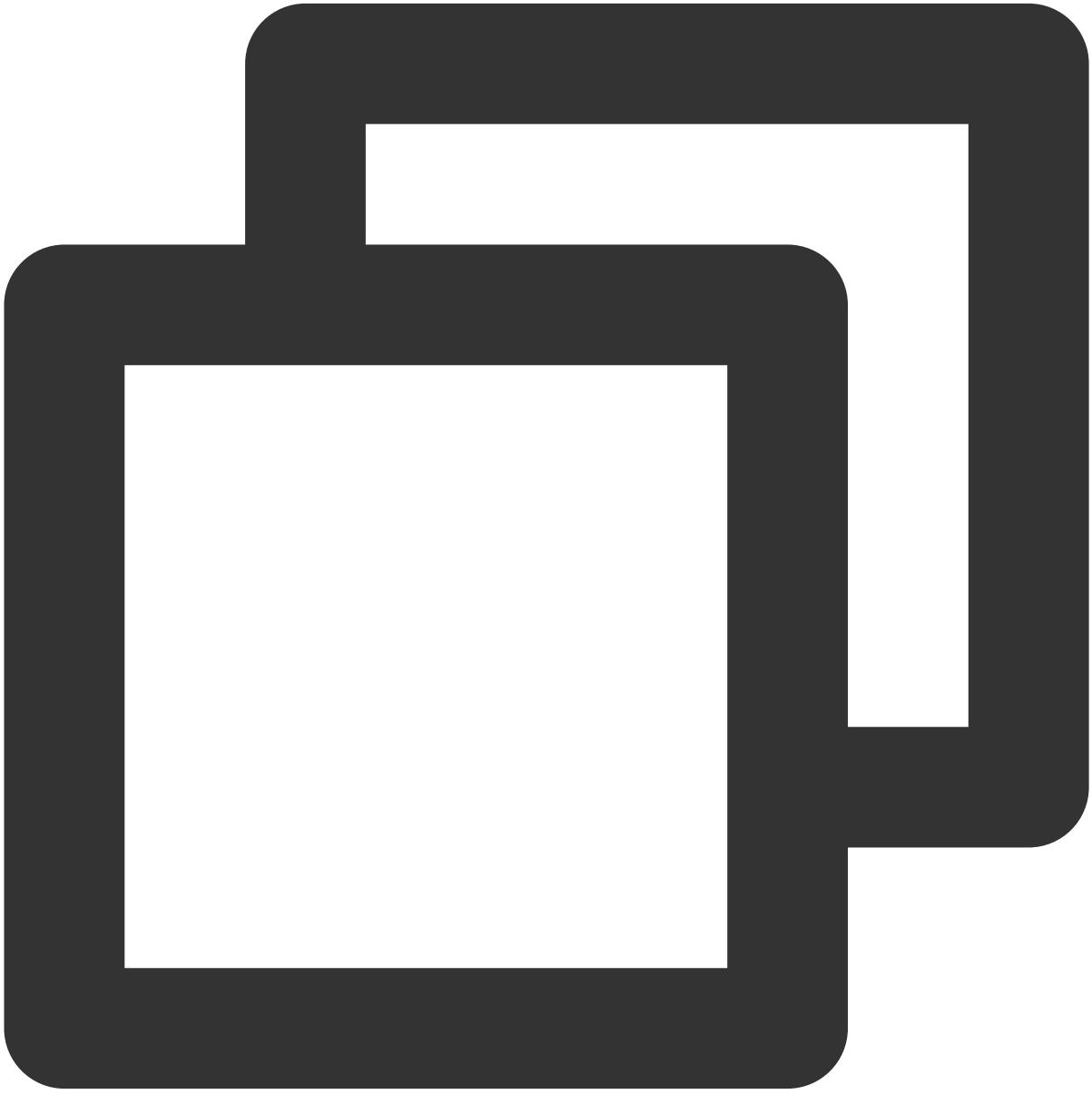
```
Page({
  onLoad: function(){
    wx.createIntersectionObserver().relativeToViewport().observe('.target-class', (
      res.id // 目标节点 id
      res.dataset // 目标节点 dataset
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
```

版权所有：腾讯云计算（北京）有限责任公司

```
    })
  }
})
```

以下示例代码可以在目标节点（用选择器 **.target-class** 指定）与参照节点（用选择器 **.relative-class** 指定）在页面显示区域内相交或相离，且相交或相离程度达到目标节点布局区域的20%和50%时，触发回调函数。

示例代码：



```
Page({
  onLoad: function(){
    wx.createIntersectionObserver(this, {
```

```
      thresholds: [0.2, 0.5]
    }).relativeTo('.relative-class').relativeToViewport().observe('.target-class',
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
    })
  }
})
```

注意：

与页面显示区域的相交区域并不准确代表用户可见的区域，因为参与计算的区域是“布局区域”，布局区域可能会在绘制时被其他节点裁剪隐藏（如遇祖先节点中 overflow 样式为 hidden 的节点）或遮盖（如遇 fixed 定位的节点）。

在自定义组件或包含自定义组件的页面中，推荐使用 `this.createIntersectionObserver` 来代替 wx.createIntersectionObserver ，这样可以确保在正确的范围内选择节点。

7. 响应显示区域变化

7.1 显示区域尺寸

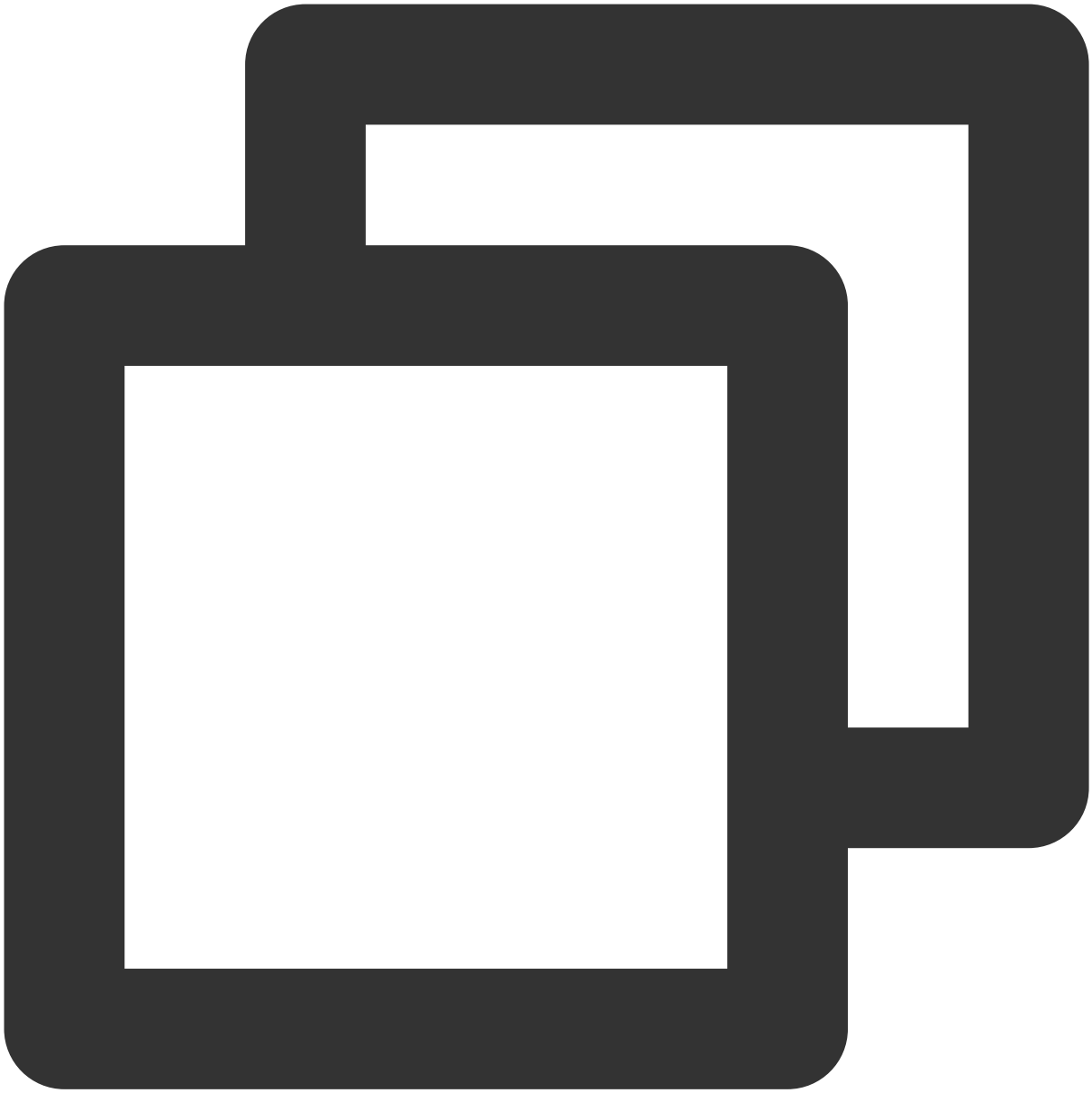
显示区域指小程序界面中可以自由布局展示的区域。在默认情况下，小程序显示区域的尺寸自页面初始化起就不会发生变化。但以下三种方式都可以改变这一默认行为。

在手机上启用屏幕旋转支持

小程序在手机上支持屏幕旋转。使小程序中的页面支持屏幕旋转的方法是：在 `app.json` 的 `window` 段中设置 `"pageOrientation": "auto"` ，或在页面 json 文件中配置 `"pageOrientation": "auto"` 。

以下是在单个页面 json 文件中启用屏幕旋转的示例。

代码示例：



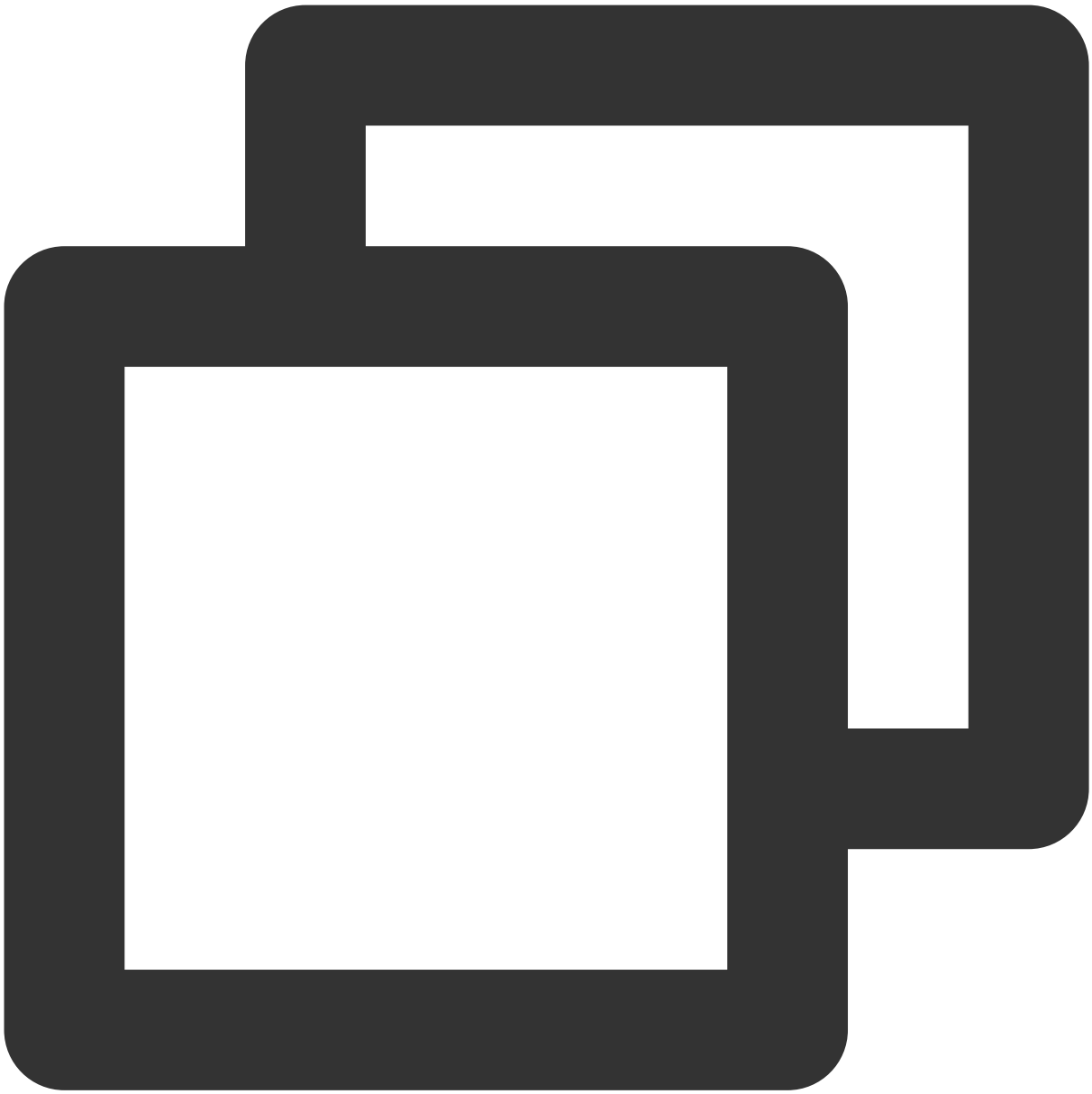
```
{
"pageOrientation": "auto"
}
```

如果页面添加了上述声明，则在屏幕旋转时，这个页面将随之旋转，显示区域尺寸也会随着屏幕旋转而变化。 `pageOrientation` 还可以被设置为 `landscape` ，表示固定为横屏显示。

在iPad上启用屏幕旋转支持

在 iPad 上运行的小程序可以支持屏幕旋转。使小程序支持 iPad 屏幕旋转的方法是：在 app.json 中添加 "resizable": true 。

代码示例：



```
{
"resizable": true
}
```

如果小程序添加了上述声明，则在屏幕旋转时，小程序将随之旋转，显示区域尺寸也会随着屏幕旋转而变化。

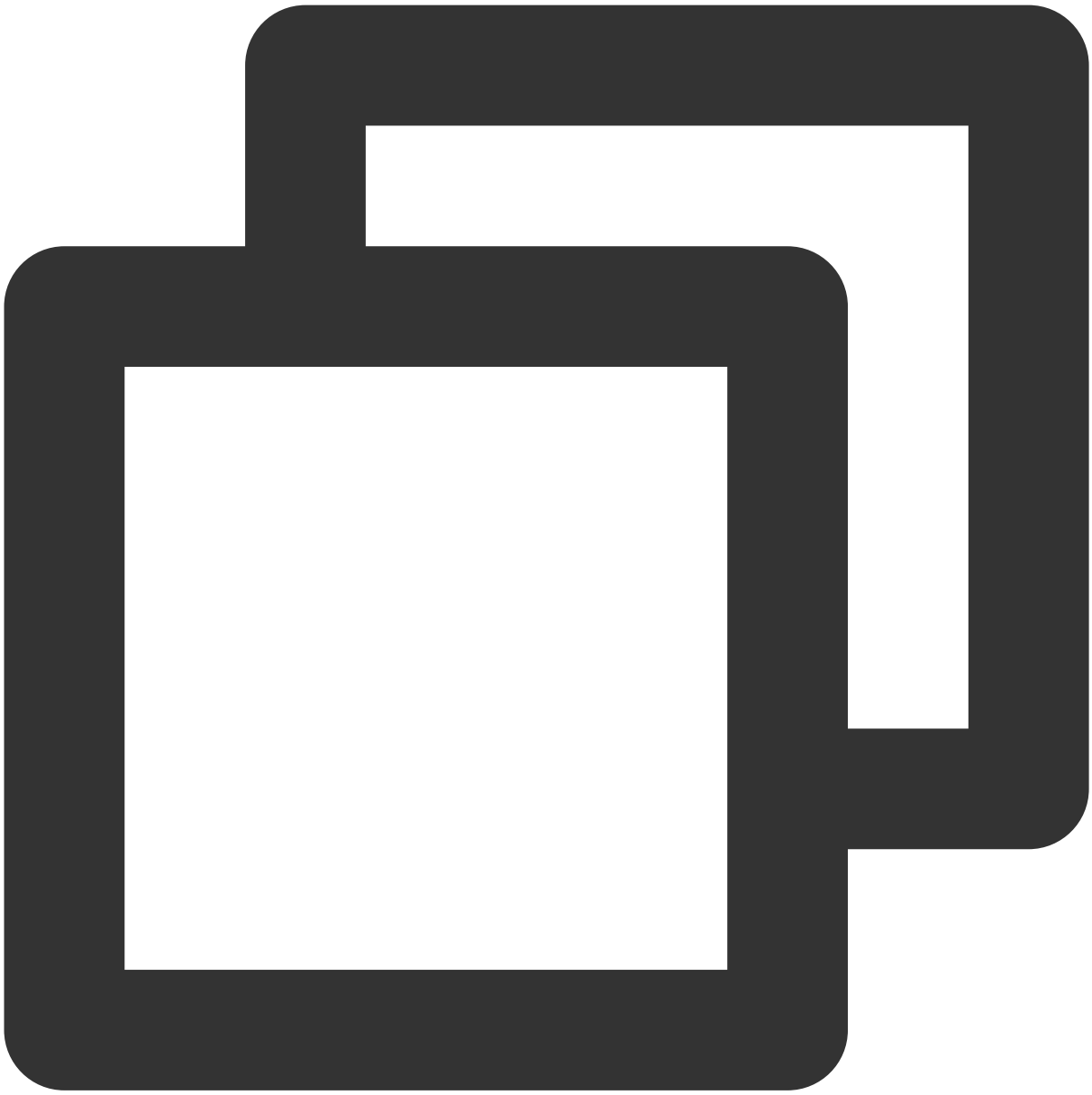
注意：

在 iPad 上不能单独配置某个页面是否支持屏幕旋转。

7.2 Media Query

有时，对于不同尺寸的显示区域，页面的布局会有所差异。此时可以使用 media query 来解决大多数问题。

代码示例：



```
.my-class {
  width: 40px;
}

@media (min-width: 480px) {
  /* 仅在 480px 或更宽的屏幕上生效的样式规则 */
  .my-class {
    width: 200px;
```

```
    }
}
```

7.3 屏幕旋转事件

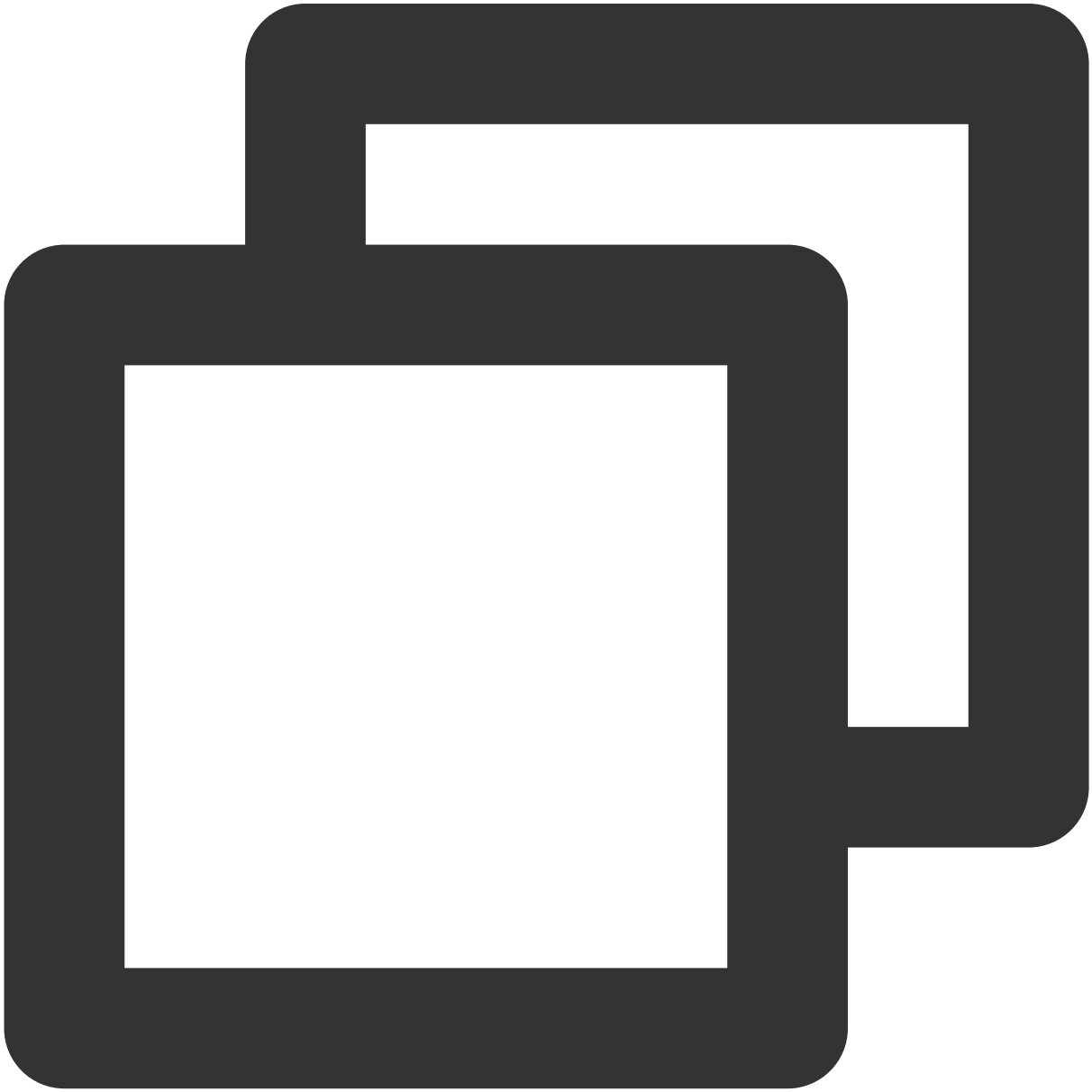
有时，仅仅使用 media query 无法控制一些精细的布局变化。此时可以使用 js 作为辅助。

在 js 中读取页面的显示区域尺寸，可以使用 selectorQuery.selectViewport 。

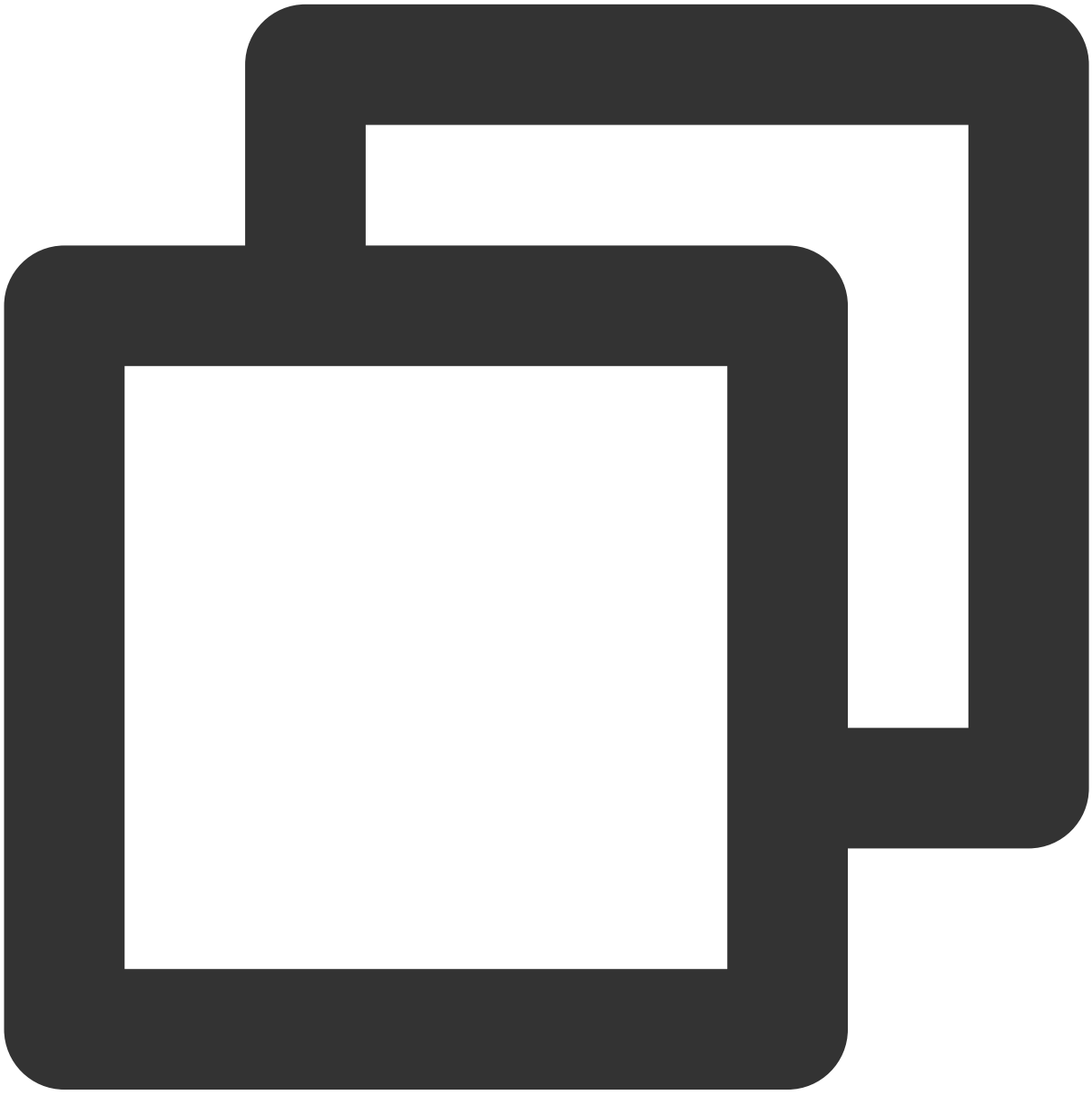
页面尺寸发生改变的事件，可以使用页面的 onResize 来监听。对于自定义组件，可以使用 resize 生命周期来监听。

回调函数中将返回显示区域的尺寸信息。

代码示例：



```
Page({
  onResize(res) {
    res.size.windowWidth // 新的显示区域宽度
    res.size.windowHeight // 新的显示区域高度
  }
})
```



```
Component({
  pageLifetimes: {
    resize(res) {
```

```
        res.size.windowWidth // 新的显示区域宽度
        res.size.windowHeight // 新的显示区域高度
      }
    }
})
```

此外，还可以使用 wx.onWindowResize 来监听（但这不是推荐的方式）。

8. 页面动画

8.1 界面动画的常见方式

在小程序中，通常可以使用 CSS 渐变 和 CSS 动画 来创建简易的界面动画。
动画过程中，可以使用 `bindtransitionend` `bindanimationstart` `bindanimationiteration` `bindanimationend` 来监听动画事件。

事件名	含义
transitionend	CSS 渐变结束或 wx.createAnimation 结束一个阶段
animationstart	CSS 动画开始
animationiteration	CSS 动画结束一个阶段
animationend	CSS 动画结束

说明：
这几个事件都不是冒泡事件，需要绑定在真正发生了动画的节点上才会生效。
同时，还可以使用 wx.createAnimation 接口来动态创建简易的动画效果（新版小程序基础库中推荐使用下述的关键帧动画接口代替）。

8.2 高级的动画方式

在一些复杂场景下，上述的动画方法可能并不适用。
WXS 响应事件 的方式可以通过使用 WXS 来响应事件的方法来动态调整节点的 style 属性。通过不断改变 style 属性的值可以做到动画效果。同时，这种方式也可以根据用户的触摸事件来动态地生成动画。
连续使用 setData 来改变界面的方法也可以达到动画的效果。这样可以任意地改变界面，但通常会产生较大的延迟或卡顿，甚至导致小程序僵死。此时可以通过将页面的 setData 改为 自定义组件 中的 setData 来提升性能。

版权所有：腾讯云计算（北京）有限责任公司

自定义组件

最近更新时间：2024-07-12 18:41:45

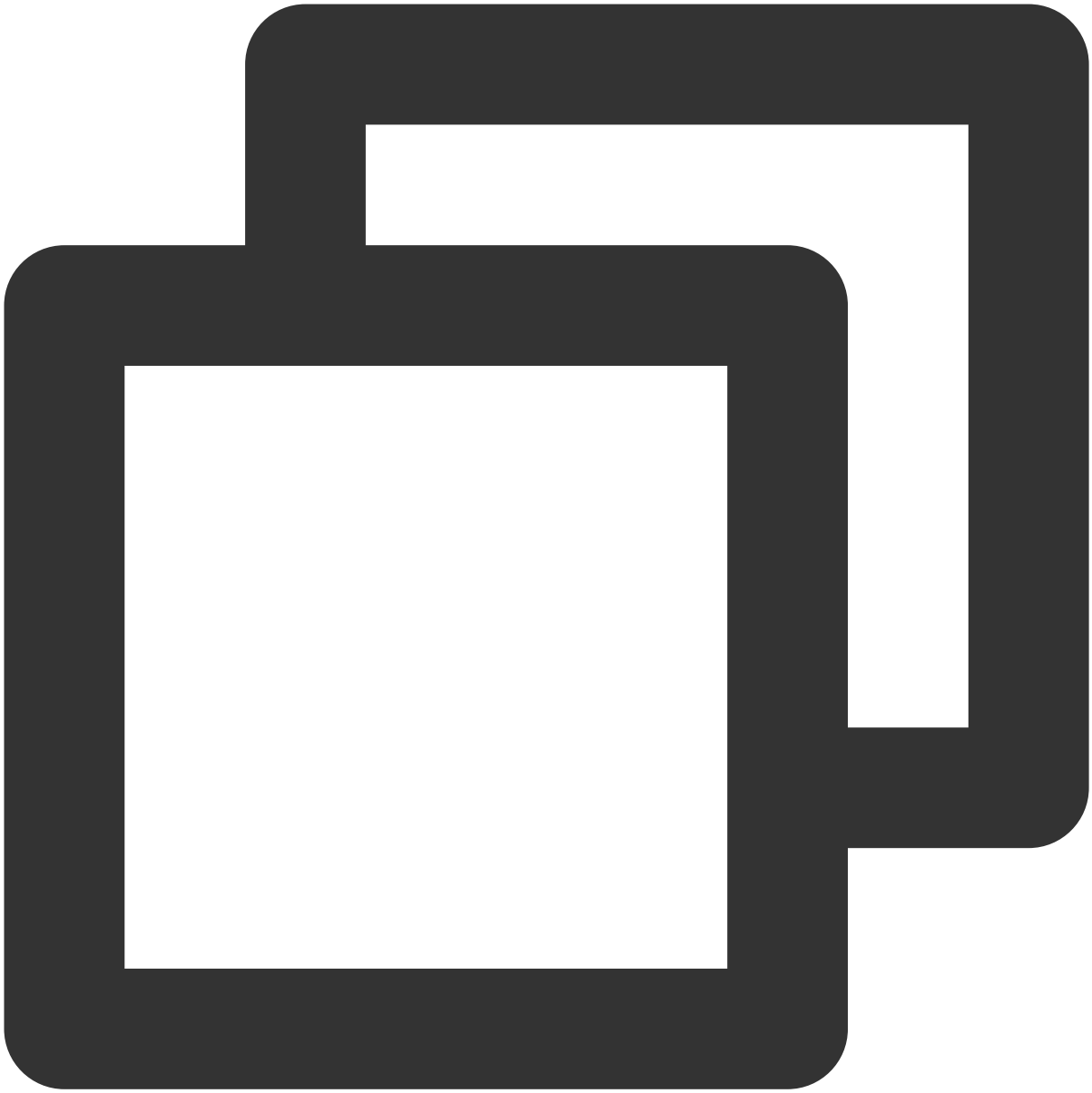
1. 介绍

开发者可以将页面内的功能模块抽象成自定义组件，以便在不同的页面中重复使用；也可以将复杂的页面拆分成多个低耦合的模块，有助于代码维护。自定义组件在使用时与基础组件非常相似。

1.1 创建自定义组件

类似于页面，一个自定义组件由 `json` `wxml` `wxss` `js` 4个文件组成。要编写一个自定义组件，首先需要在 `json` 文件中进行自定义组件声明（将 `component` 字段设为 `true` 可将这一组文件设为自定义组件）：

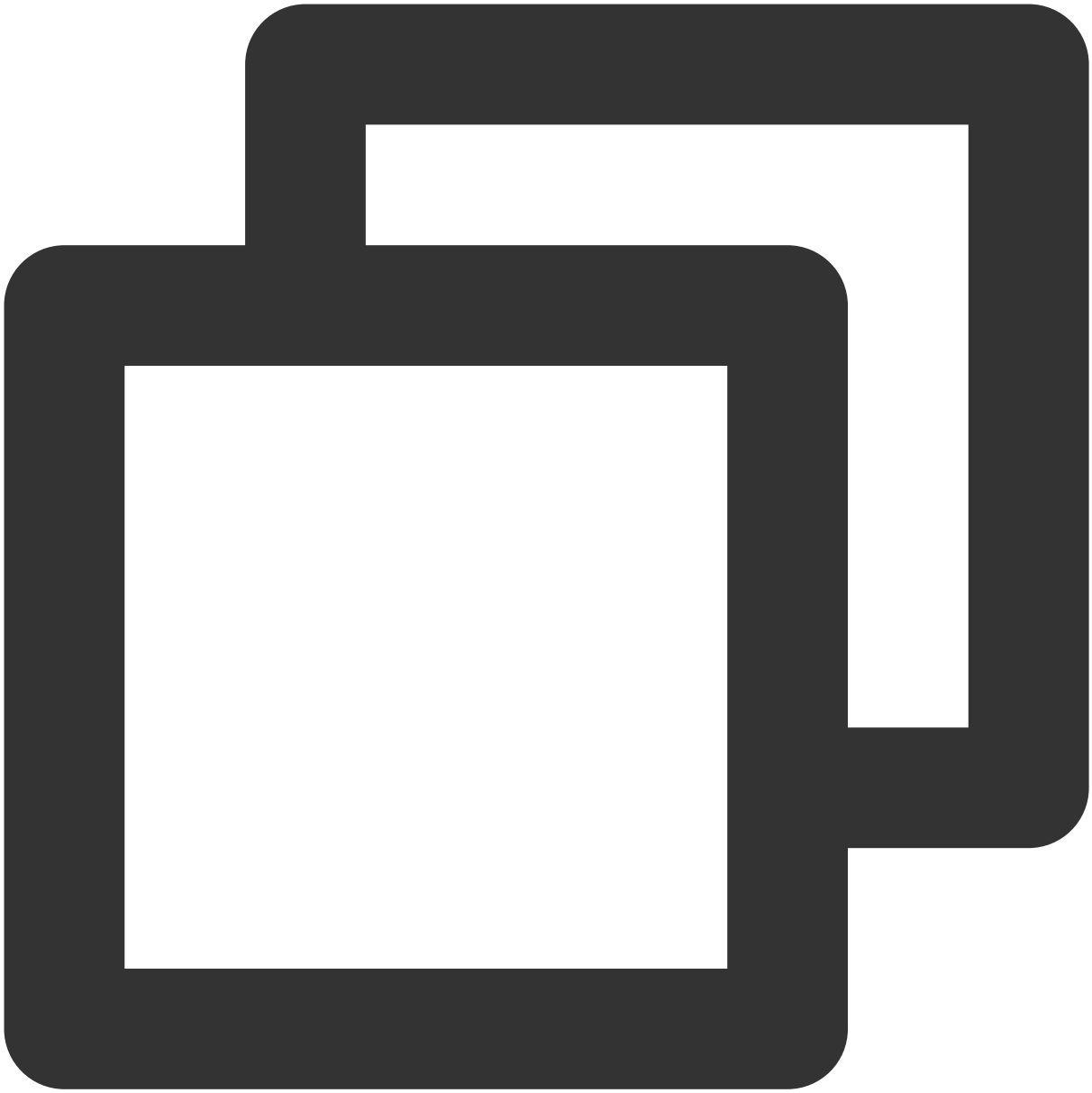
版权所有：腾讯云计算（北京）有限责任公司



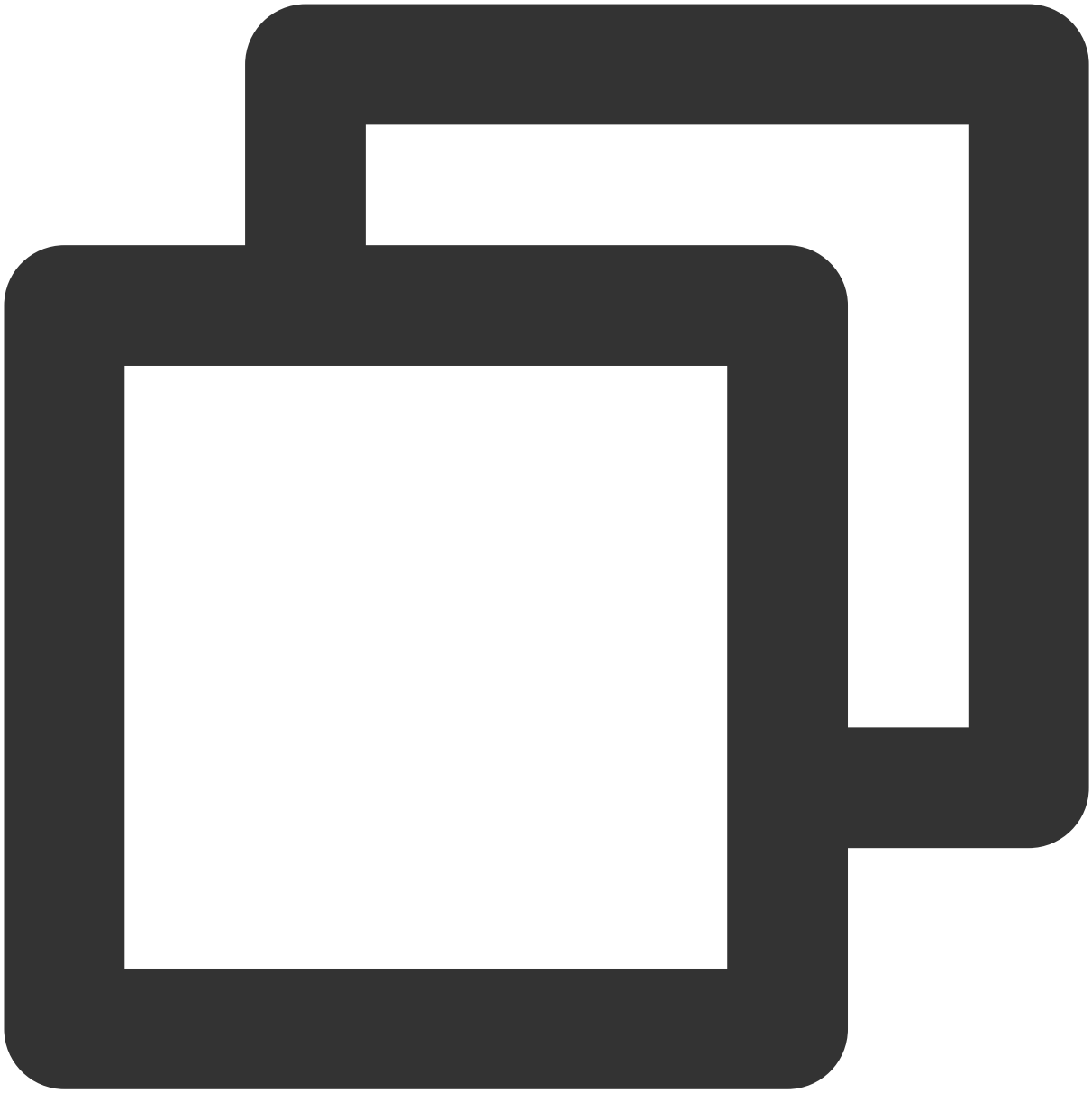
```
{
  "component": true
}
```

同时，还要在 `wxml` 文件中编写组件模板，在 `wxss` 文件中加入组件样式，它们的写法与页面的写法类似。具体细节和注意事项参见 组件模板和样式。

代码示例：



```
<!-- 这是自定义组件的内部WXML结构 -->
<view class="inner">
  {{innerText}}
</view>
<slot></slot>
```



```
/* 这里的样式只应用于这个自定义组件 */
.inner {
  color: red;
}
```

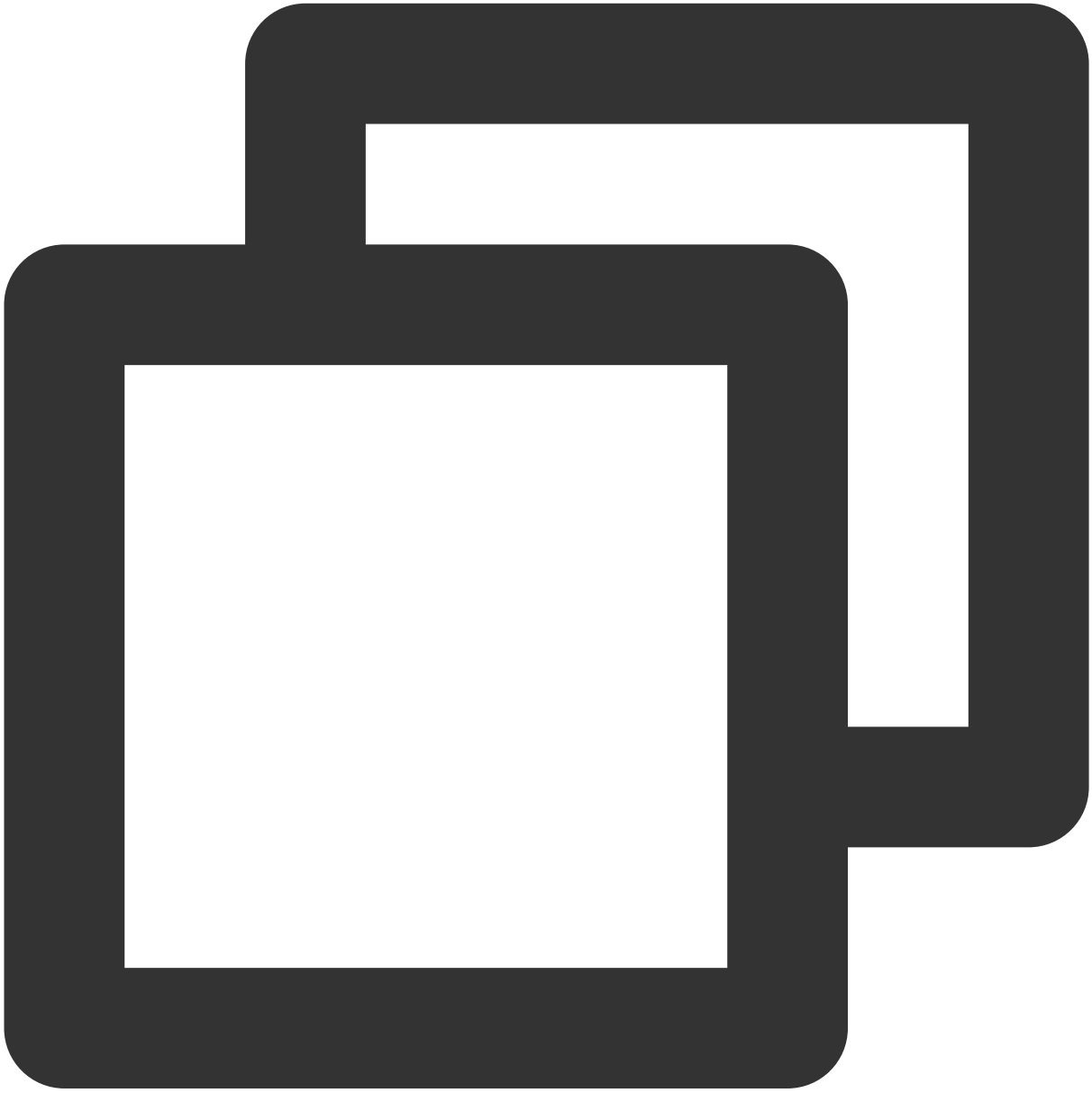
注意：

在组件wxss中不应使用ID选择器、属性选择器和标签名选择器。

在自定义组件的 js 文件中，需要使用 `Component()` 来注册组件，并提供组件的属性定义、内部数据和自定义方法。

组件的属性值和内部数据将被用于组件 `wxml` 的渲染，其中，属性值是可由组件外部传入的。更多细节参见Component构造器。

代码示例：

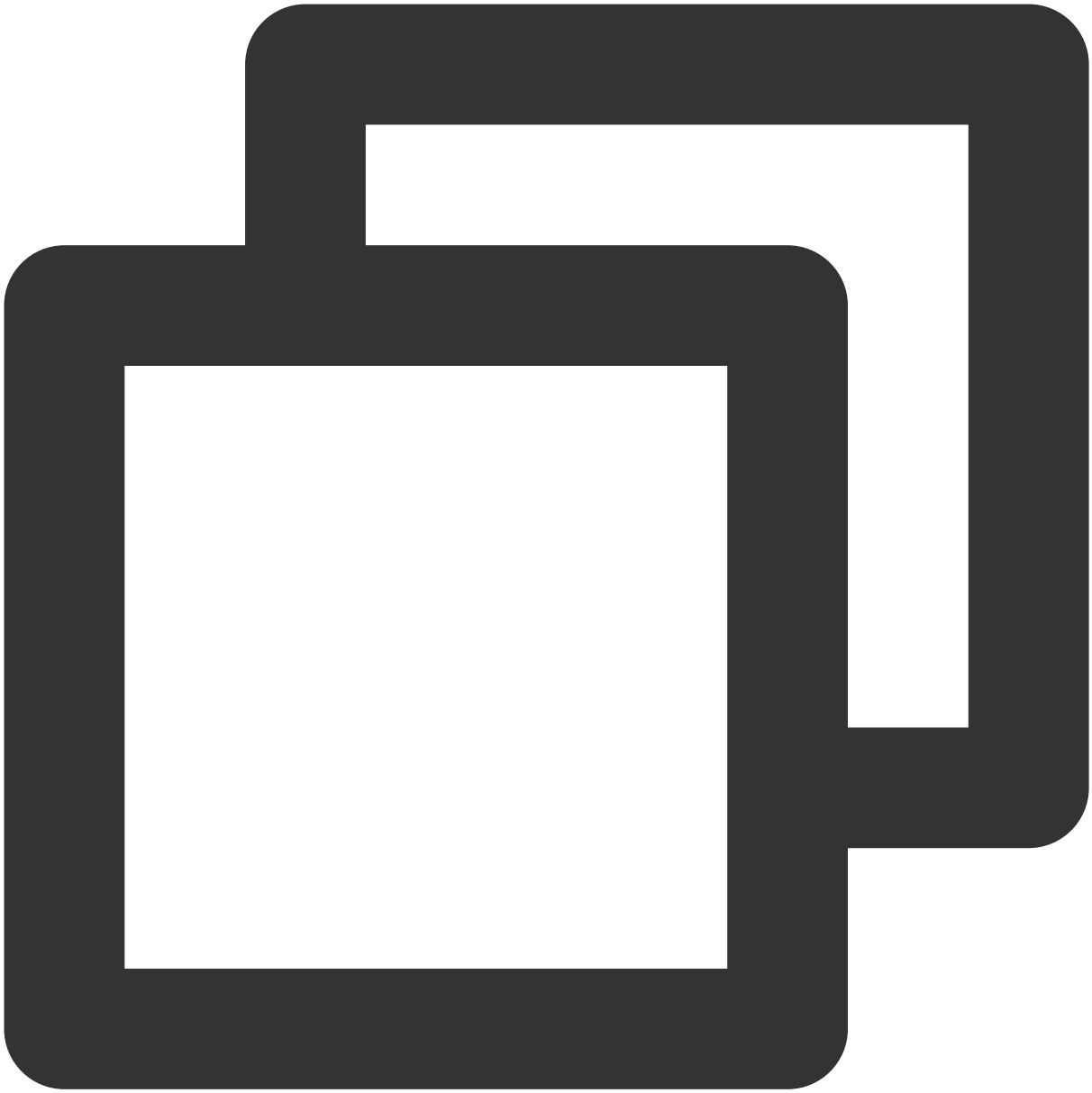


```
Component({
  properties: {
    // 这里定义了innerText属性，属性值可以在组件使用时指定
    innerText: {
      type: String,
      value: 'default value',
    }
```

```
  },
  data: {
    // 这里是一些组件内部数据
    someData: {}
  },
  methods: {
    // 这里是一个自定义方法
    customMethod: function(){}
  }
})
```

1.2 使用自定义组件

使用已注册的自定义组件前，首先要在页面的 `json` 文件中进行引用声明。此时需要提供每个自定义组件的标签名和对应的自定义组件文件路径：



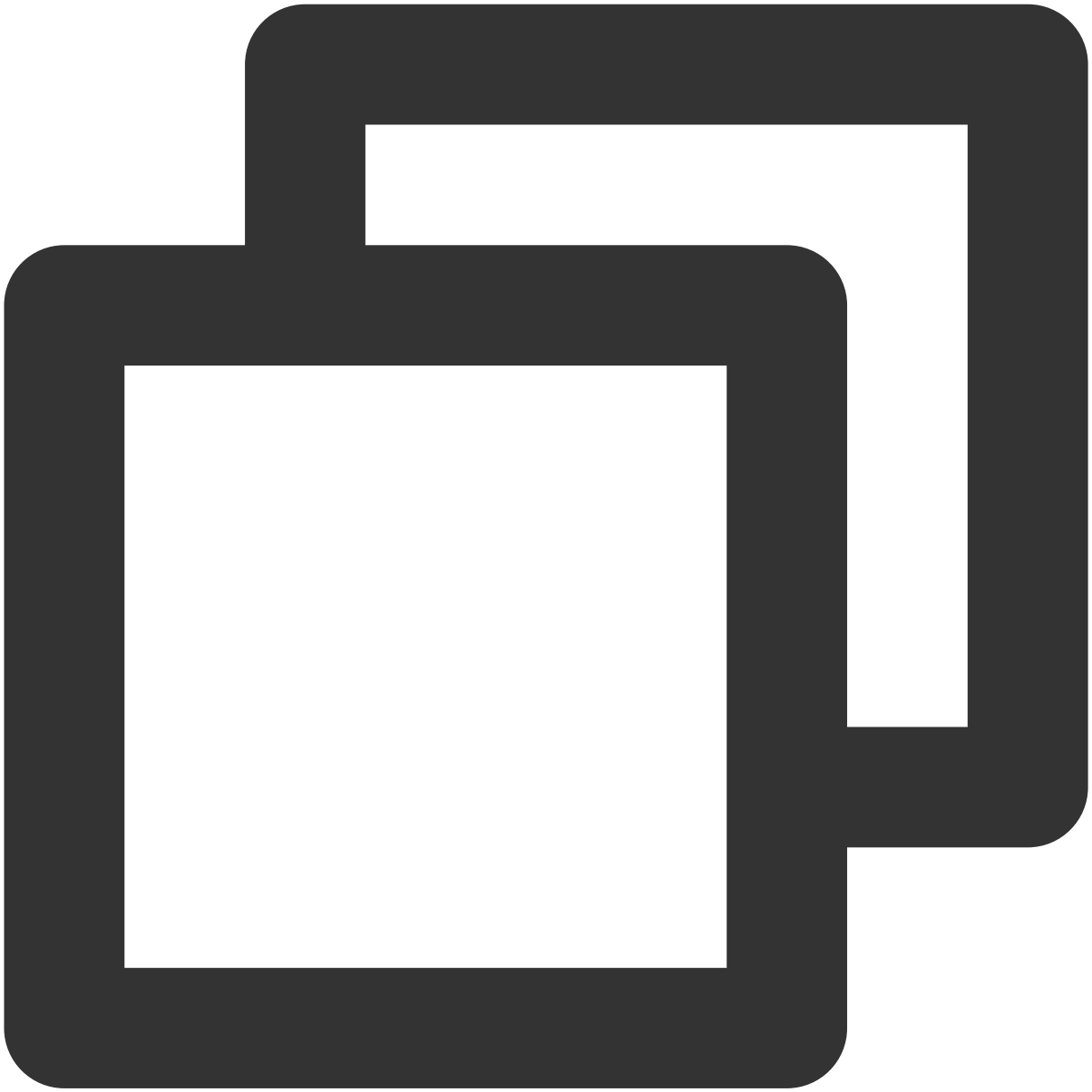
```
{
  "usingComponents": {
    "component-tag-name": "path/to/the/custom/component"
  }
}
```

这样，在页面的 `wxml` 中就可以像使用基础组件一样使用自定义组件。节点名即自定义组件的标签名，节点属性即传递给组件的属性值。

说明：

在 app.json 中声明的 usingComponents 字段视为全局自定义组件，在小程序内的页面或自定义组件中可以直接使用而无需再声明。

代码示例：



```
<view>
  <!-- 以下是对一个自定义组件的引用 -->
  <component-tag-name inner-text="Some text"></component-tag-name>
</view>
```

自定义组件的 `wxml` 节点结构在与数据结合之后，将被插入到引用位置内。

1.3 注意事项

一些需要注意的细节：

因为 WXML 节点标签名只能是**小写字母**、**中划线**和**下划线**的组合，所以自定义组件的标签名也只能包含这些字符。

自定义组件也是可以引用自定义组件的，引用方法类似于页面引用自定义组件的方式（使用 usingComponents 字段）。

自定义组件和页面所在项目根目录名不能以“wx-”为前缀，否则会报错。

注意，是否在页面文件中使用 `usingComponents` 会使得页面的 `this` 对象的原型稍有差异，包括：

使用 `usingComponents` 页面的原型与不使用时不一致，即 `Object.getPrototypeOf(this)` 结果不同。

使用 `usingComponents` 时会多一些方法，如 `selectComponent` 。

出于性能考虑，使用 `usingComponents` 时， `setData` 内容不会被直接深复制，即 `this.setData({ field: obj })` 后 `this.data.field === obj` 。（深复制会在这个值被组件间传递时发生。）

如果页面比较复杂，新增或删除 `usingComponents` 定义段时建议重新测试一下。

2. 组件模板和样式

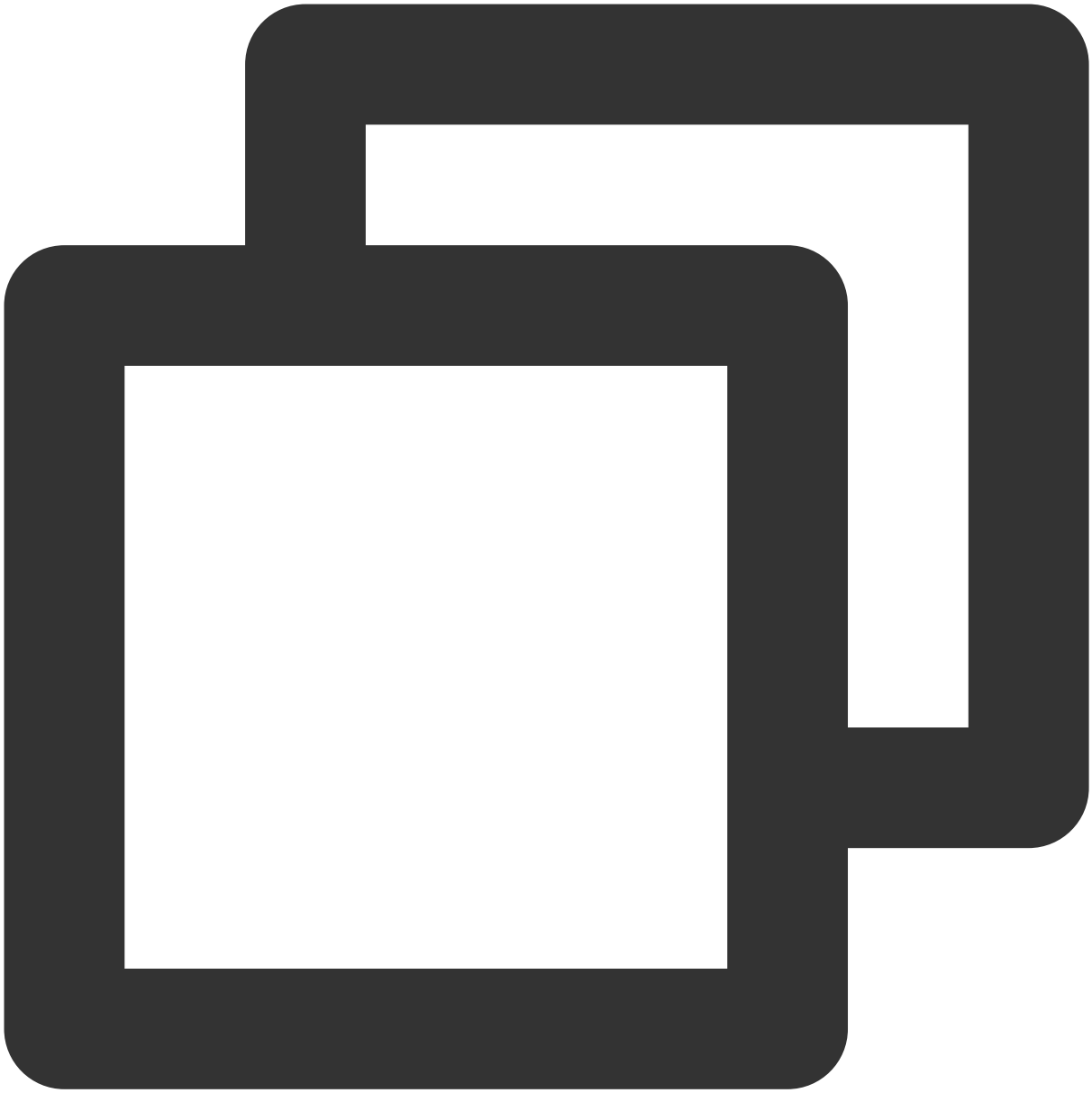
类似于页面，自定义组件拥有自己的 `wxml` 模板和 `wxss` 样式。

2.1 组件模板

组件模板的写法与页面模板相同。组件模板与组件数据结合后生成的节点树，将被插入到组件的引用位置上。

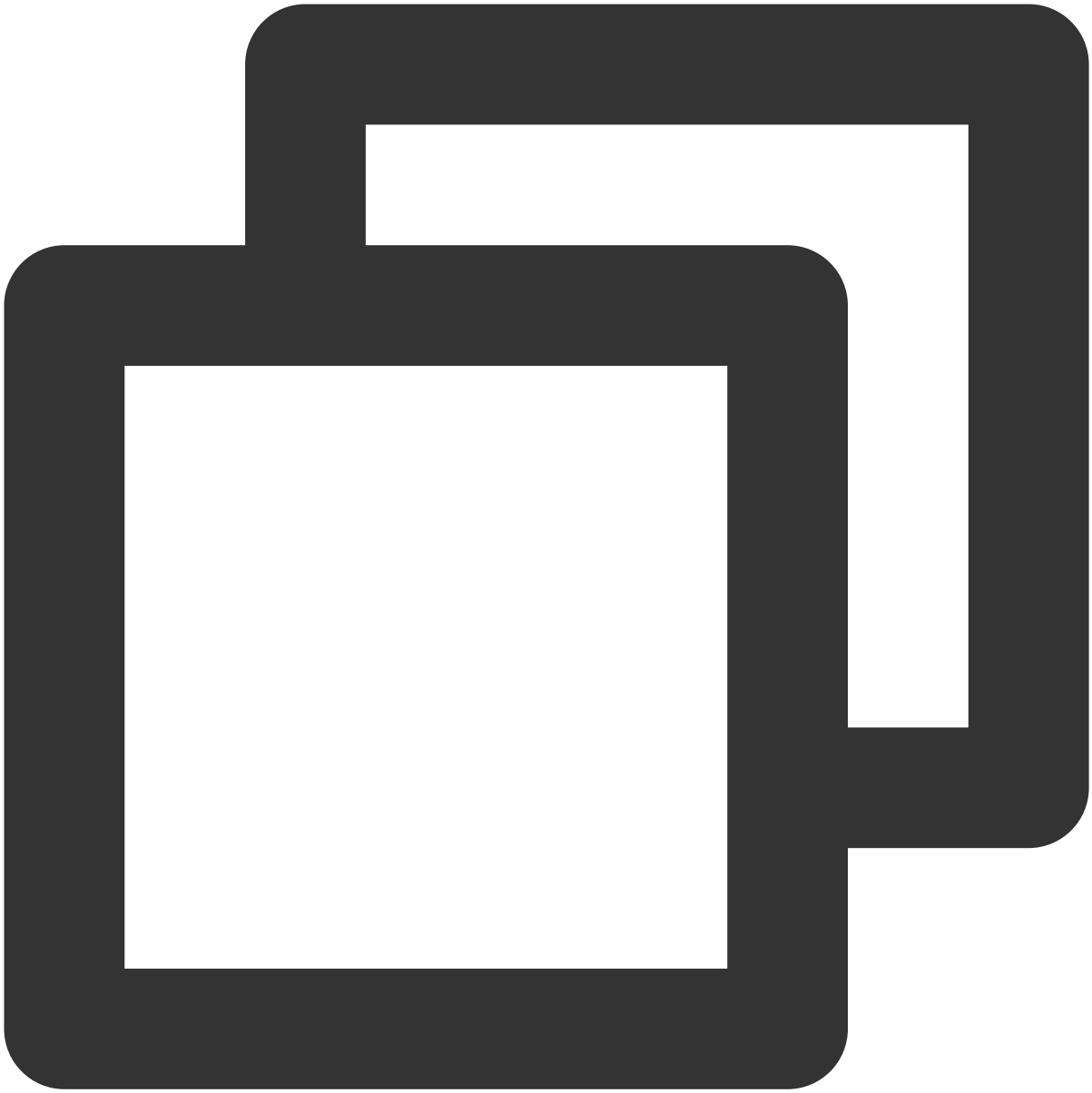
在组件模板中可以提供一个 `<slot>` 节点，用于承载组件引用时提供的子节点。

代码示例：



```
<!-- 组件模板 -->
<view class="wrapper">
  <view>这里是组件的内部节点</view>
  <slot></slot>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>
```

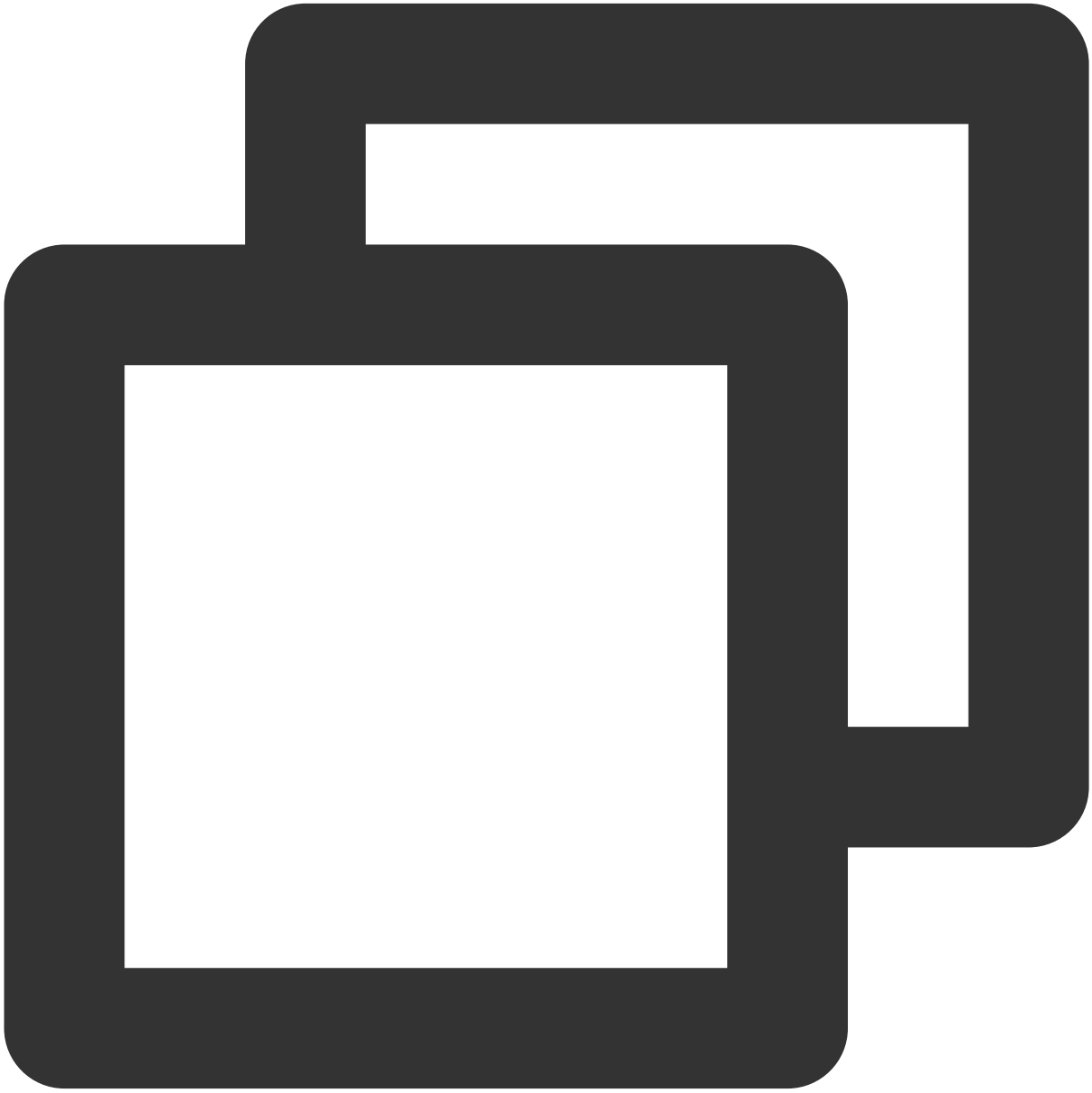
注意：

在模板中引用到的自定义组件及其对应的节点名需要在 json 文件中显式定义，否则会被当作一个无意义的节点。除此以外，节点名也可以被声明为抽象节点。

2.2 模板数据绑定

与普通的 WXML 模板类似，可以使用数据绑定，这样就可以向子组件的属性传递动态数据。

代码示例：



```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name prop-a="{{dataFieldA}}" prop-b="{{dataFieldB}}">
```

```
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>
```

在以上例子中，组件的属性 `propA` 和 `propB` 将收到页面传递的数据。页面可以通过 `setData` 来改变绑定的数据字段。

注意：

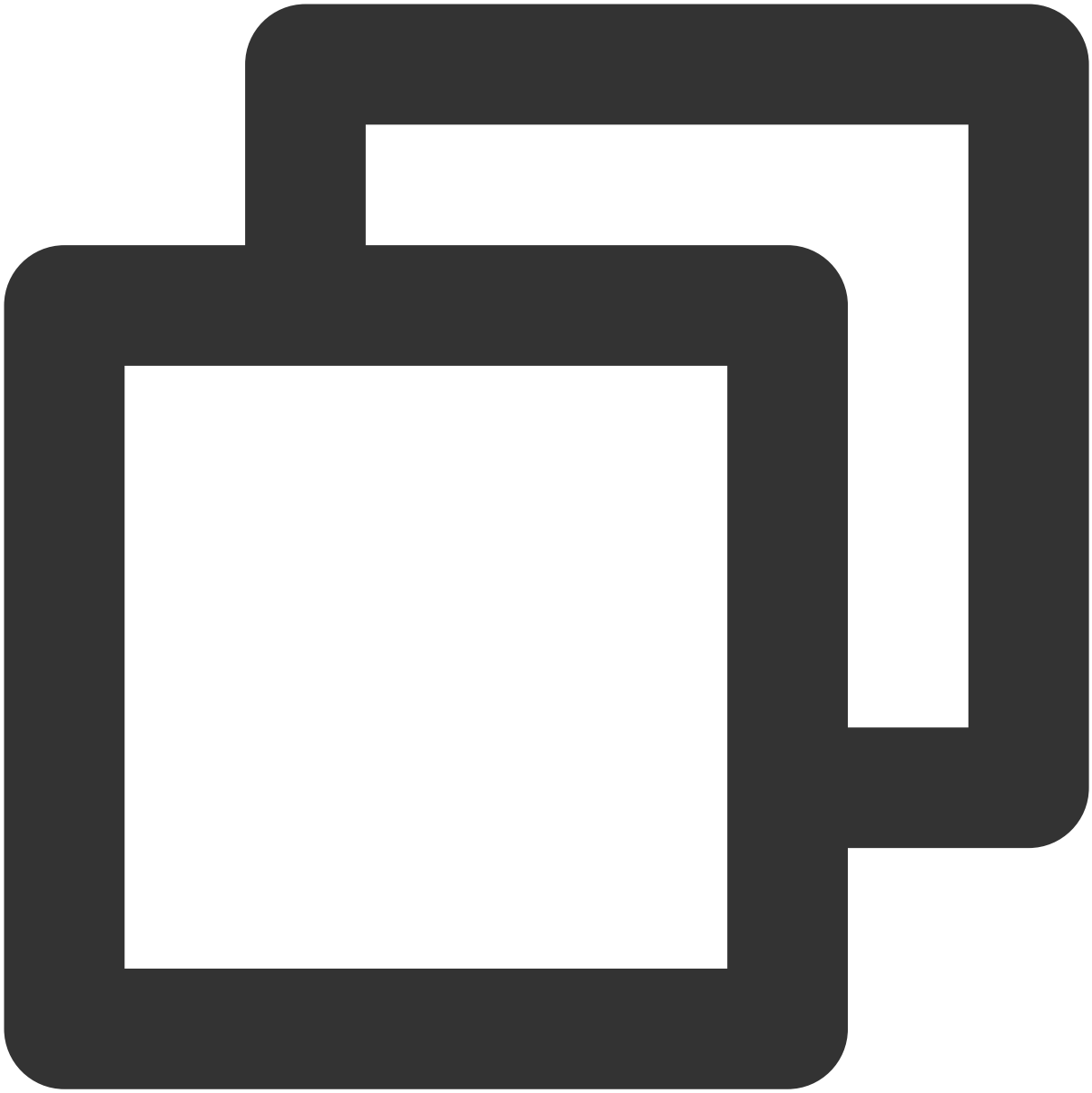
这样的数据绑定只能传递 JSON 兼容数据。

2.3 组件 WXML 的 slot

在组件的 wxml 中可以包含 `slot` 节点，用于承载组件使用者提供的 wxml 结构。

默认情况下，一个组件的 wxml 中只能有一个 `slot` 。需要使用多 `slot` 时，可以在组件 js 中声明启用。

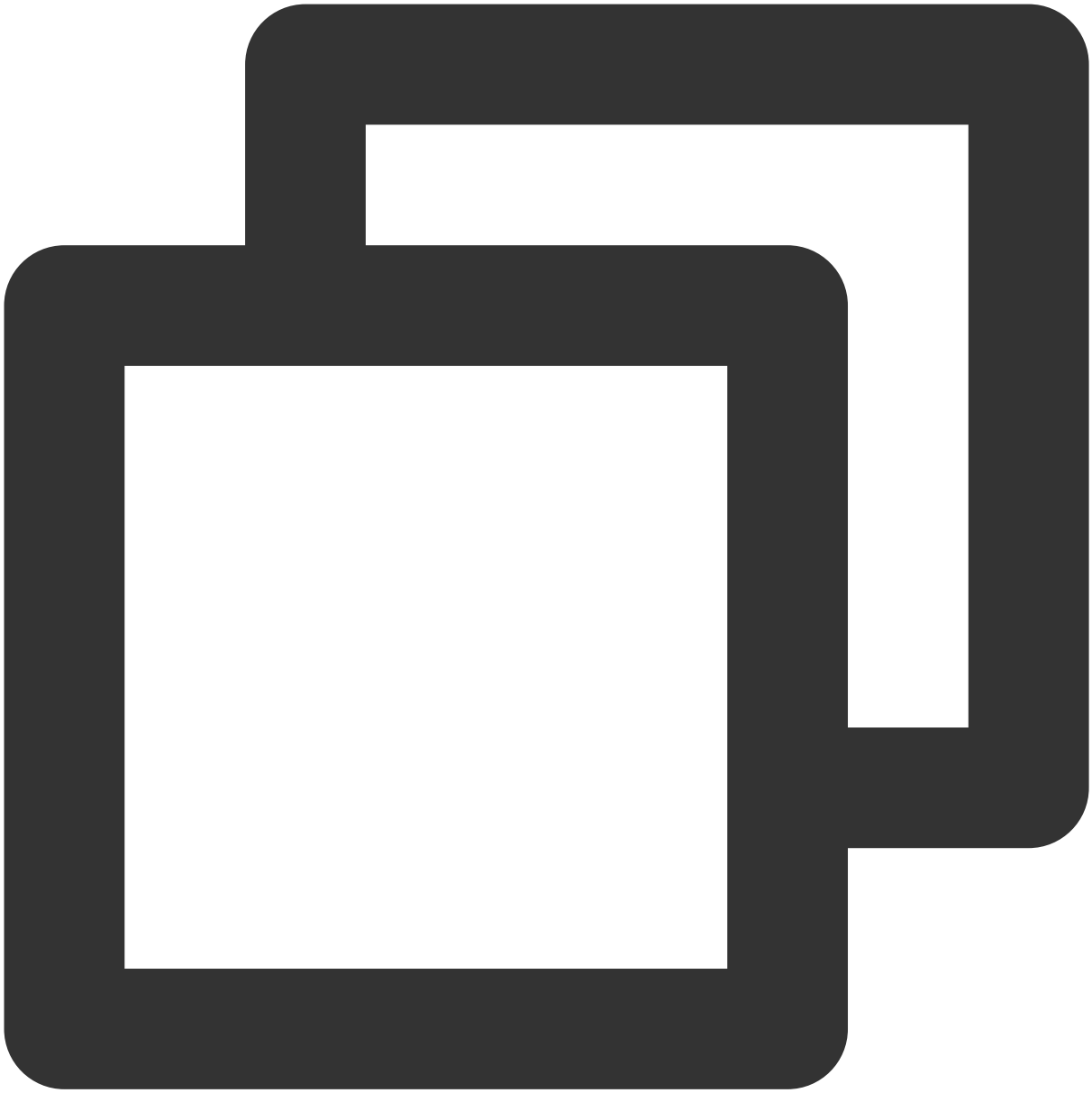
版权所有：腾讯云计算（北京）有限责任公司



```
Component({
  options: {
    multipleSlots: true // 在组件定义时的选项中启用多slot支持
  },
  properties: { /* ... */ },
  methods: { /* ... */ }
})
```

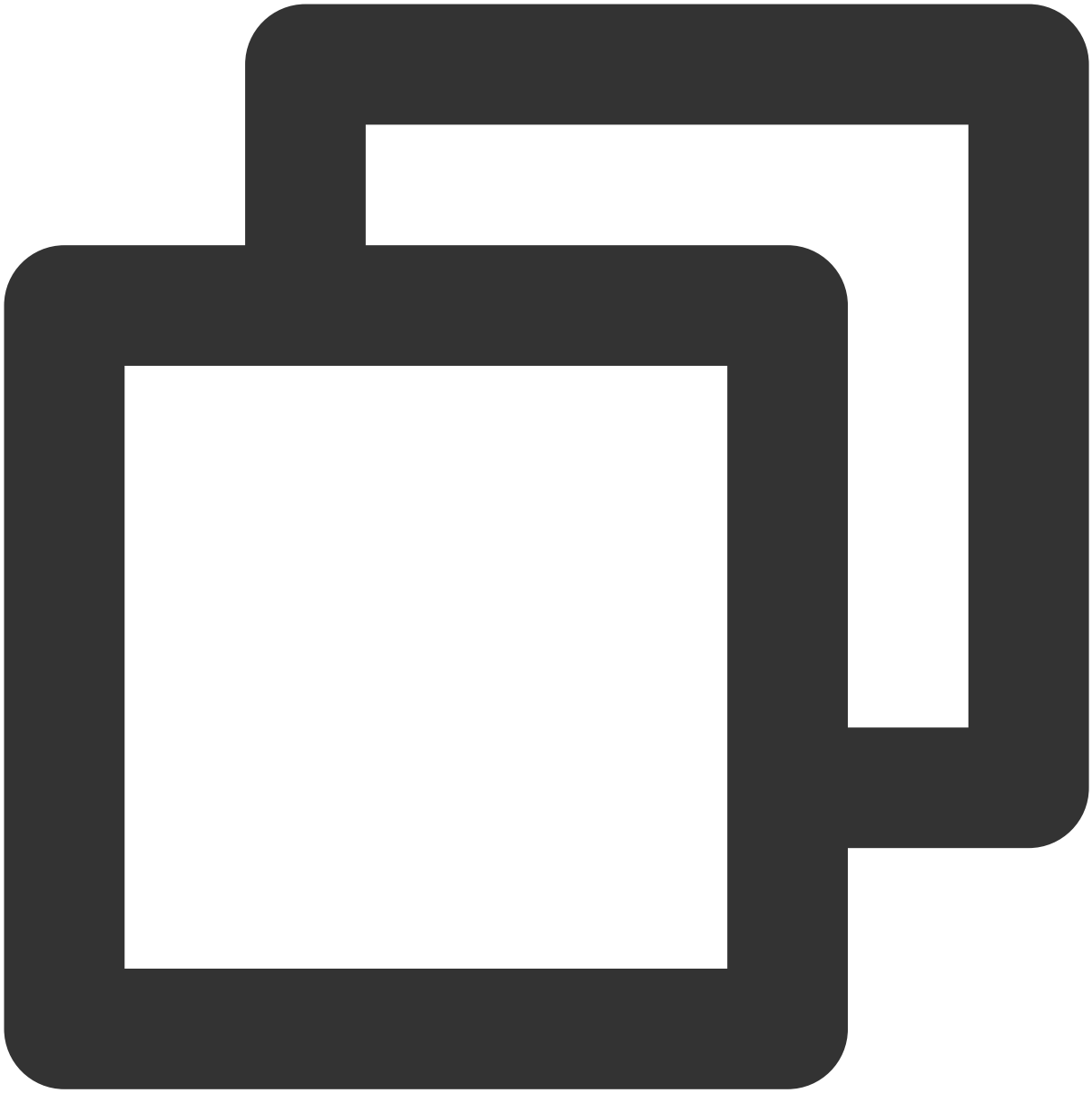
此时，可以在这个组件的 wxml 中使用多个 `slot` ，以不同的 `name` 来区分。

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- 组件模板 -->
<view class="wrapper">
  <slot name="before"></slot>
  <view>这里是组件的内部细节</view>
  <slot name="after"></slot>
</view>
```

使用时，用 `slot` 属性来将节点插入到不同的 slot 上。



```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->
    <view slot="before">这里是插入到组件slot name="before"中的内容</view>
    <!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->
    <view slot="after">这里是插入到组件slot name="after"中的内容</view>
  </component-tag-name>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司

2.4 组件样式

组件对应 `wxss` 文件的样式，只对组件 wxml 内的节点生效。编写组件样式时，需要注意以下几点：

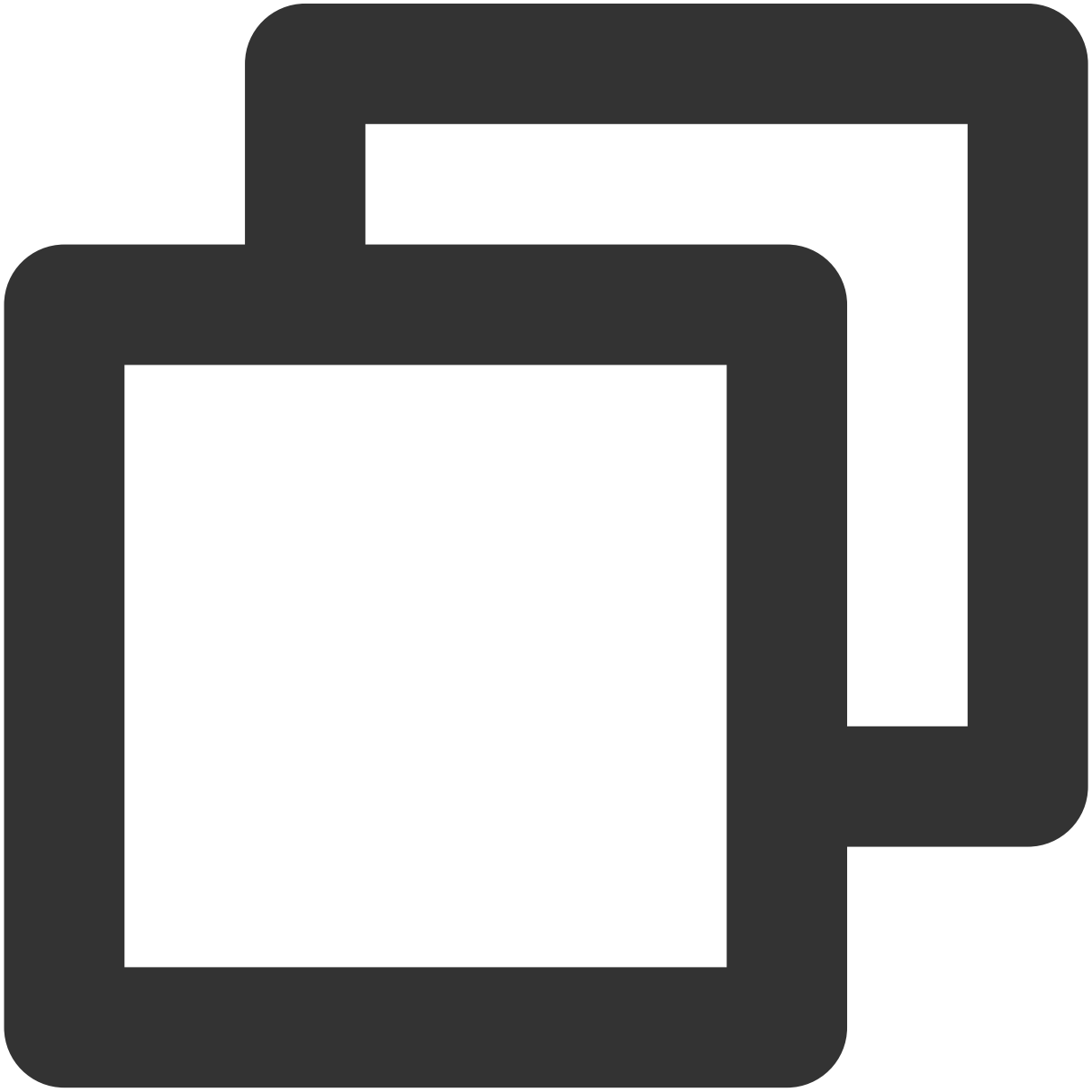
组件和引用组件的页面不能使用 id 选择器（ `#a` ）、属性选择器（ `[a]` ）和标签名选择器，请改用 class 选择器。

组件和引用组件的页面中使用后代选择器（ `.a .b` ）在一些极端情况下会有非预期的表现，如遇，请避免使用。

子元素选择器（ `.a>.b` ）只能用于 view 组件与其子节点之间，用于其他组件可能导致非预期的情况。

继承样式，如 `font` 、 `color` ，会从组件外继承到组件内。

除继承样式外， `app.wxss` 中的样式、组件所在页面的样式对自定义组件无效（除非更改组件样式隔离选项）。

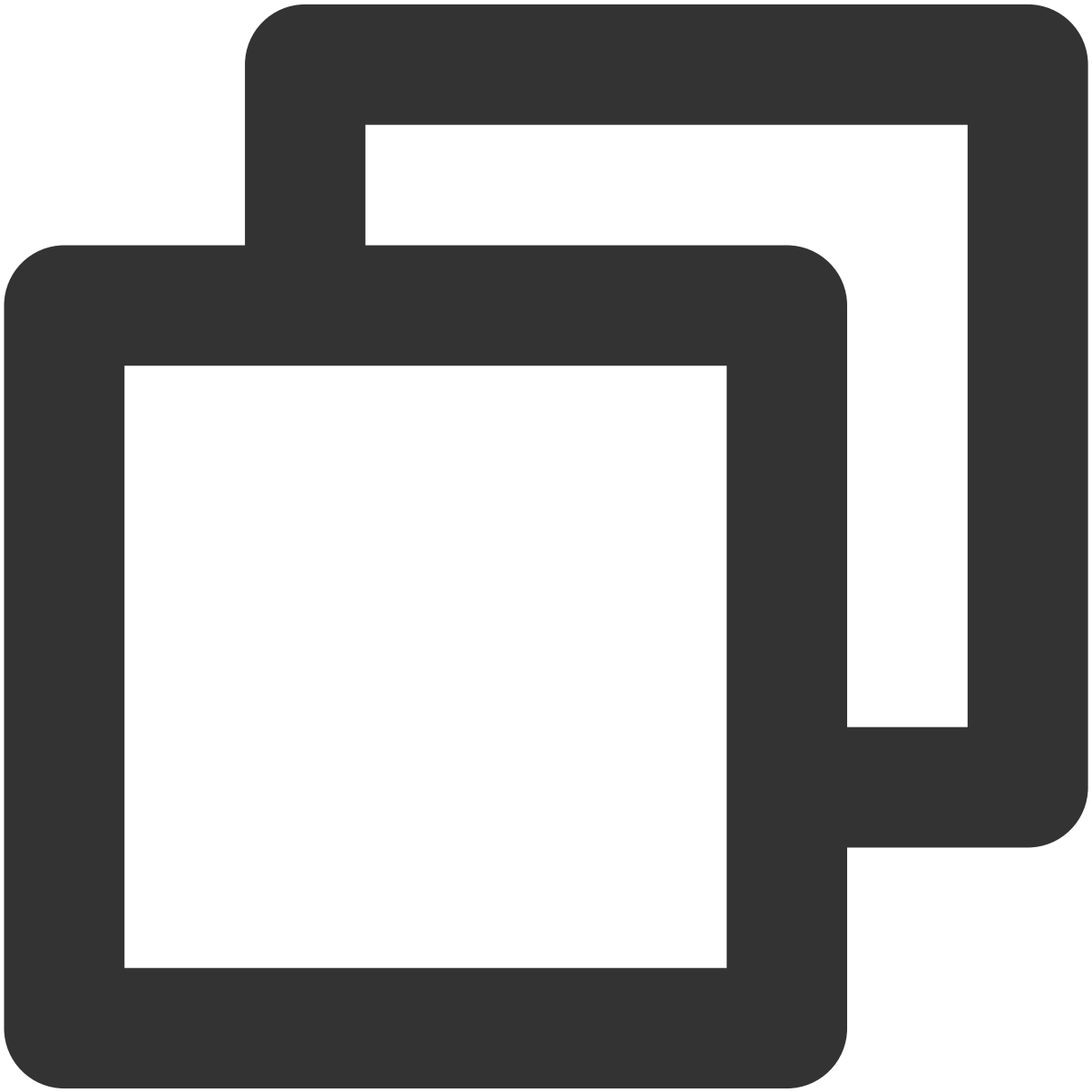


版权所有：腾讯云计算（北京）有限责任公司

```
#a { } /* 在组件中不能使用 */
[a] { } /* 在组件中不能使用 */
button { } /* 在组件中不能使用 */
.a > .b { } /* 除非 .a 是 view 组件节点，否则不一定会生效 */
```

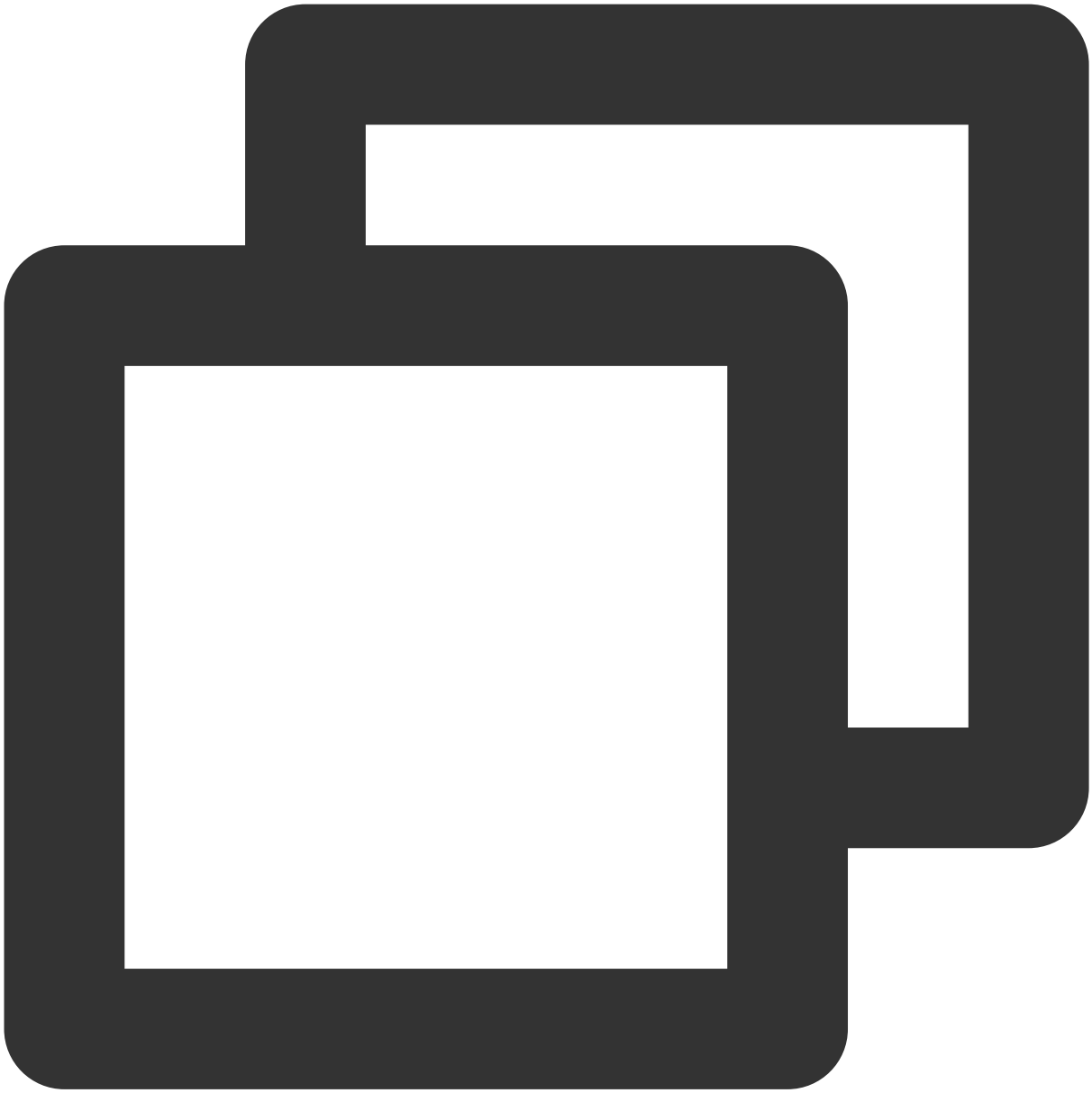
除此以外，组件可以指定它所在节点的默认样式，使用 `:host` 选择器。

代码示例：



```
/* 组件 custom-component.wxss */
:host {
```

```
  color: yellow;
}
```



```
<!-- 页面的 WXML -->
<custom-component>这段文本是黄色的</custom-component>
```

2.5 外部样式

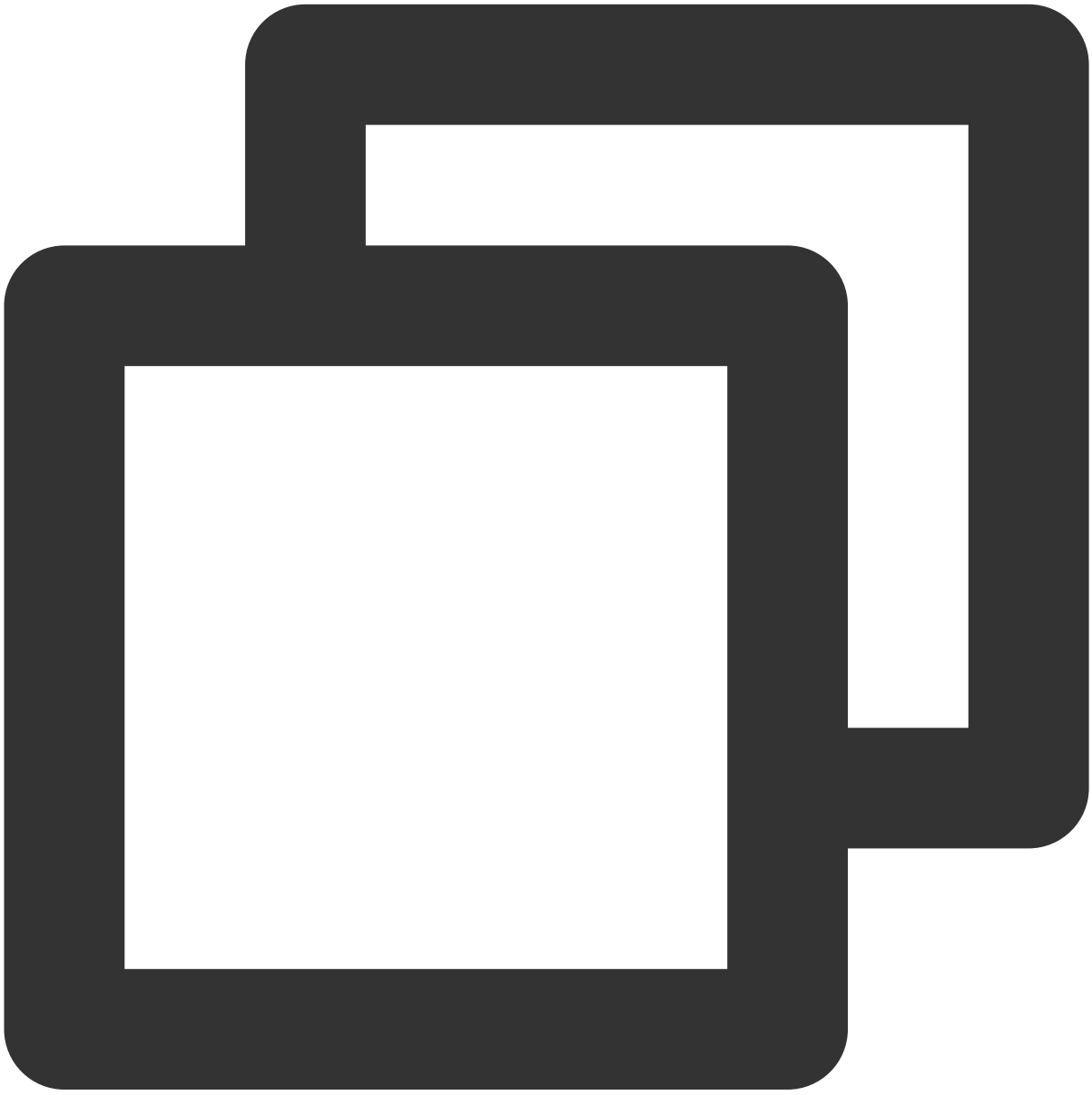
有时，组件希望接受外部传入的样式类。此时可以在 `Component` 中用 `externalClasses` 定义段定义若干个外部样式类。

这个特性可以用于实现类似于 `view` 组件的 `hover-class` 属性：页面可以提供一个样式类，赋予 `view` 的 `hover-class` ，这个样式类本身写在页面中而非 `view` 组件的实现中。

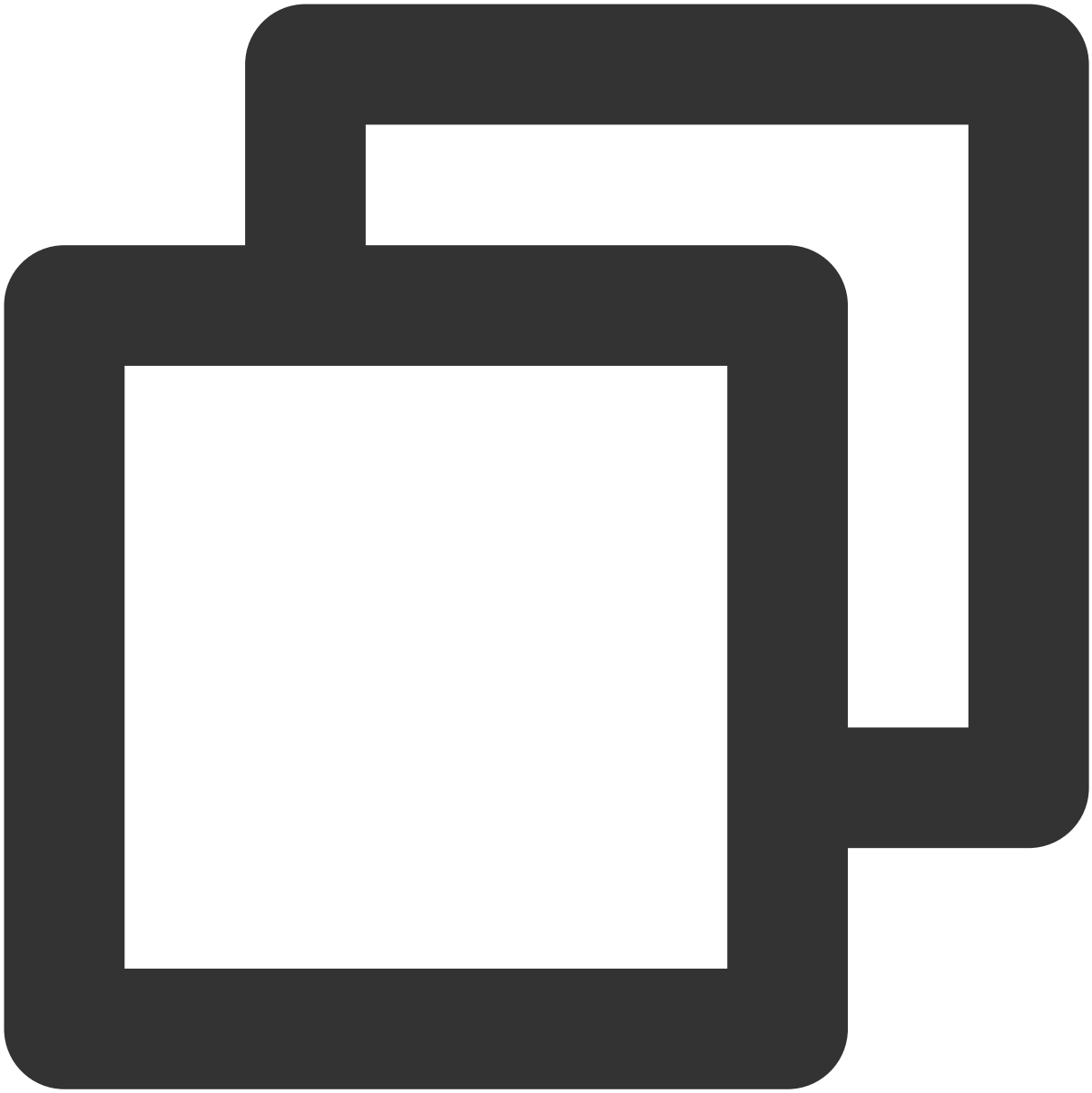
注意：

在同一个节点上使用普通样式类和外部样式类时，两个类的优先级是未定义的，因此最好避免这种情况。

代码示例：



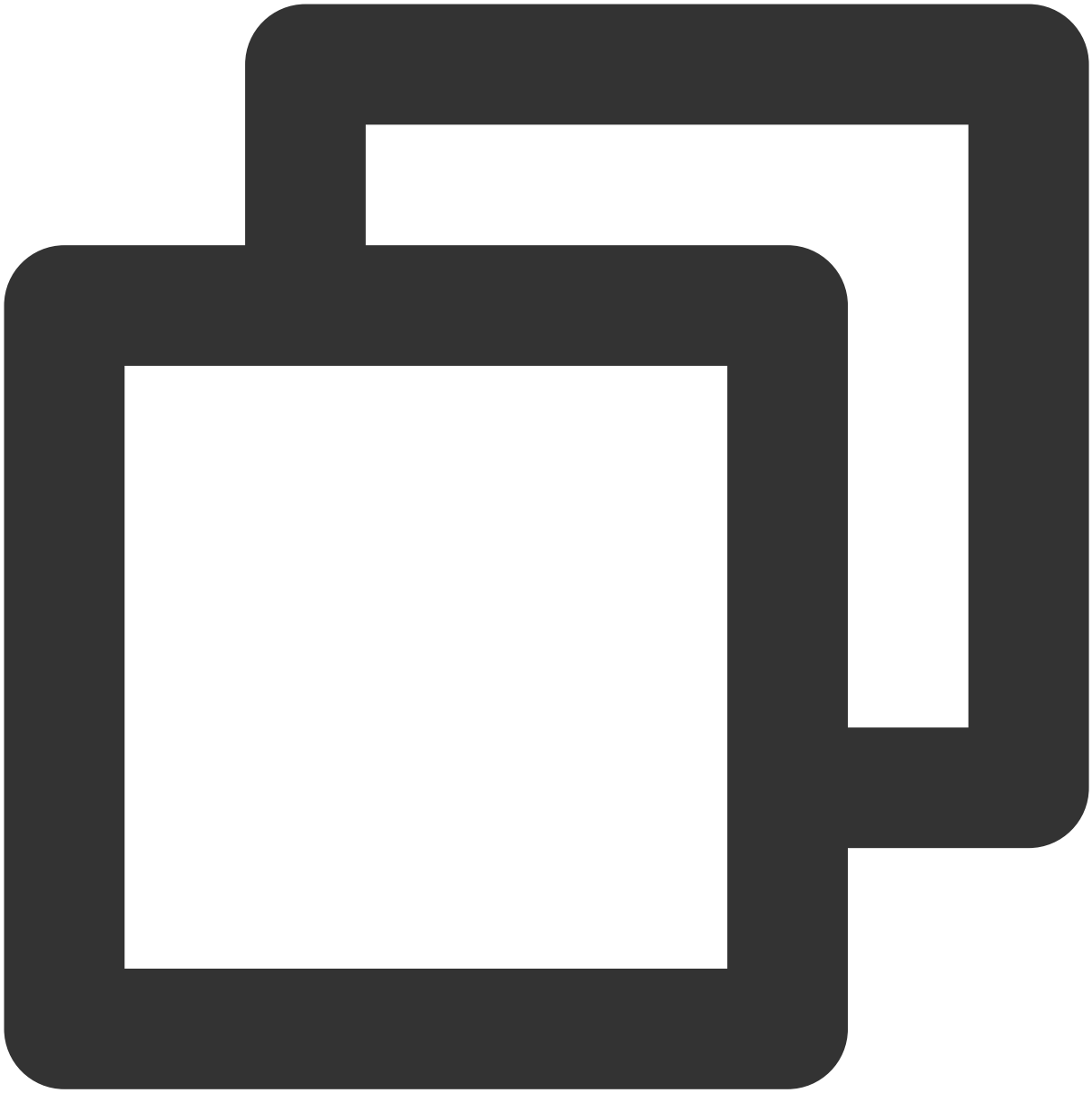
```
/* 组件 custom-component.js */
Component({
  externalClasses: ['my-class']
})
```



```
<!-- 组件 custom-component.wxml -->
<custom-component class="my-class">这段文本的颜色由组件外的 class 决定</custom-component
```

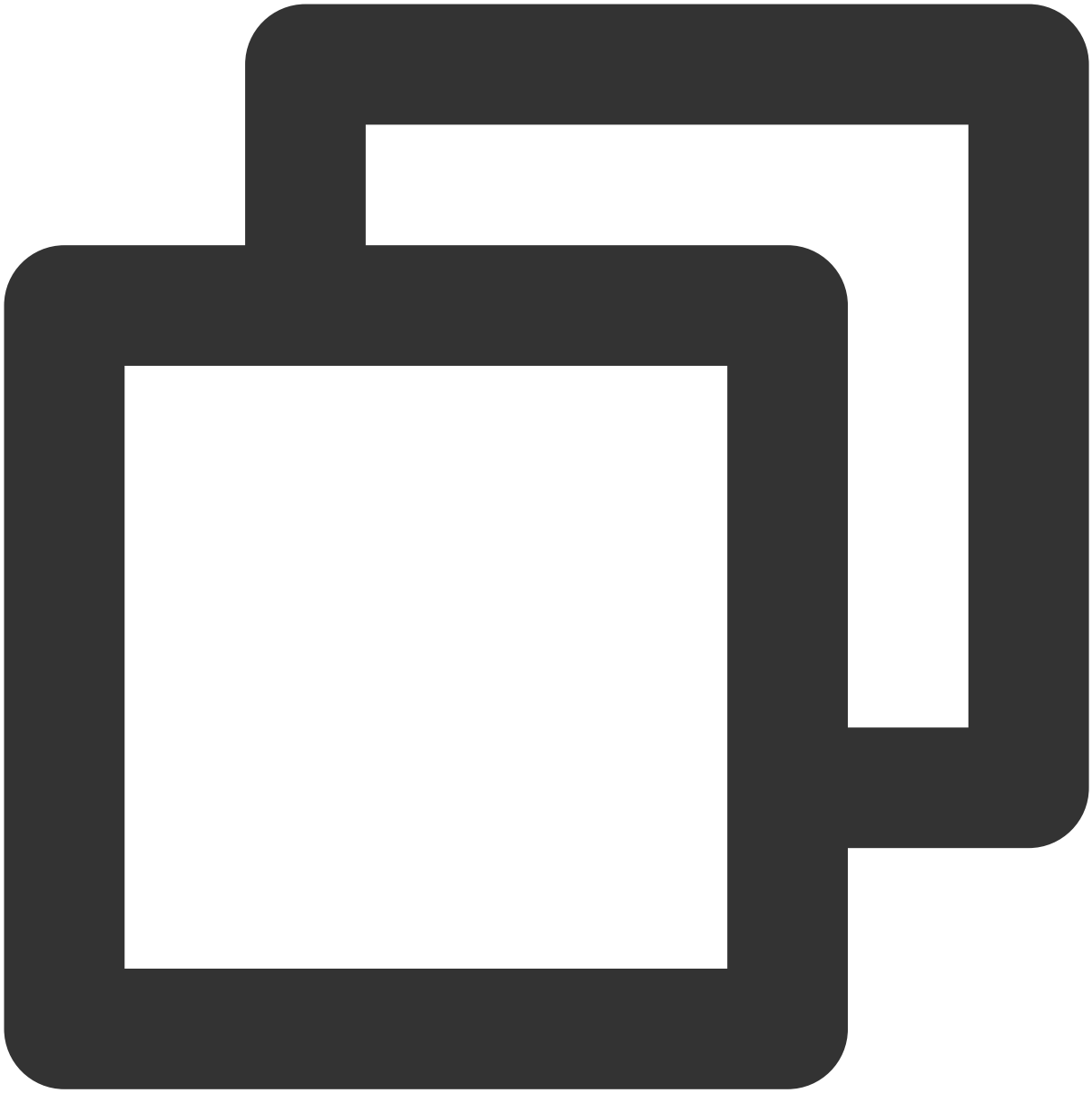
这样，组件的使用者可以指定这个样式类对应的 class ，就像使用普通属性一样。

代码示例：



```
<!-- 页面的 WXML -->
<custom-component my-class="red-text" />
<custom-component my-class="large-text" />
```

版权所有：腾讯云计算（北京）有限责任公司



```
.red-text {
  color: red;
}
.large-text {
  font-size: 1.5em;
}
```

2.6 使组件接受全局样式

默认情况下，自定义组件的样式只受到自定义组件 wxss 的影响。除非以下两种情况：

`app.wxss` 或页面的 `wxss` 中使用了标签名选择器（或一些其他特殊选择器）来直接指定样式，这些选择器会影响到页面和全部组件。通常情况下这是不推荐的做法。

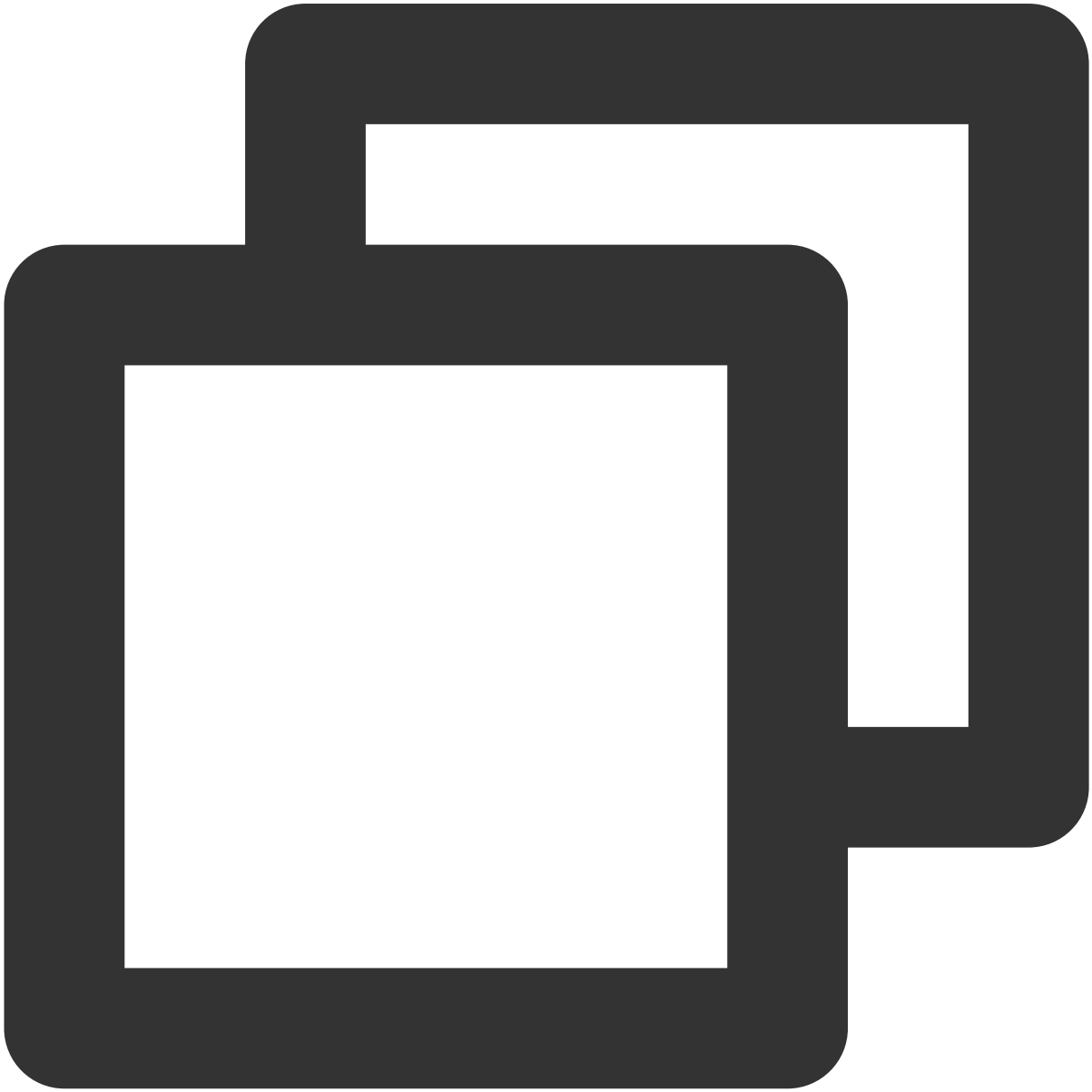
在特定的自定义组件激活了 `addGlobalClass` 选项，这使得这个自定义组件能被 `app.wxss` 或页面的 `wxss` 中的所有的样式定义影响。

要激活 `addGlobalClass` 选项，只需要在 `Component` 构造器中将 `options.addGlobalClass` 字段置为 `true` 。

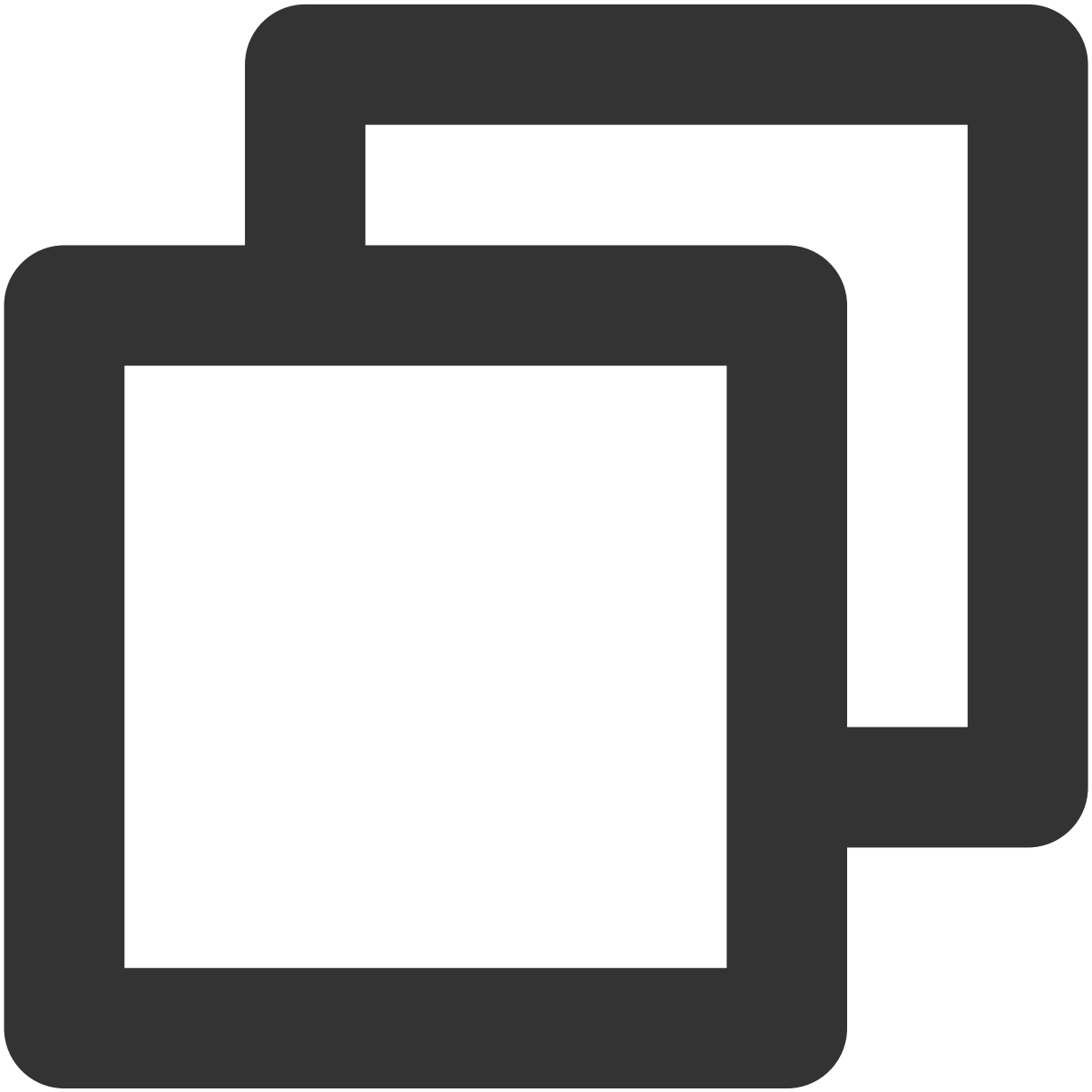
注意：

当激活了 `addGlobalClass` 选项后，存在外部样式污染组件样式的风险，请谨慎选择。

代码示例：



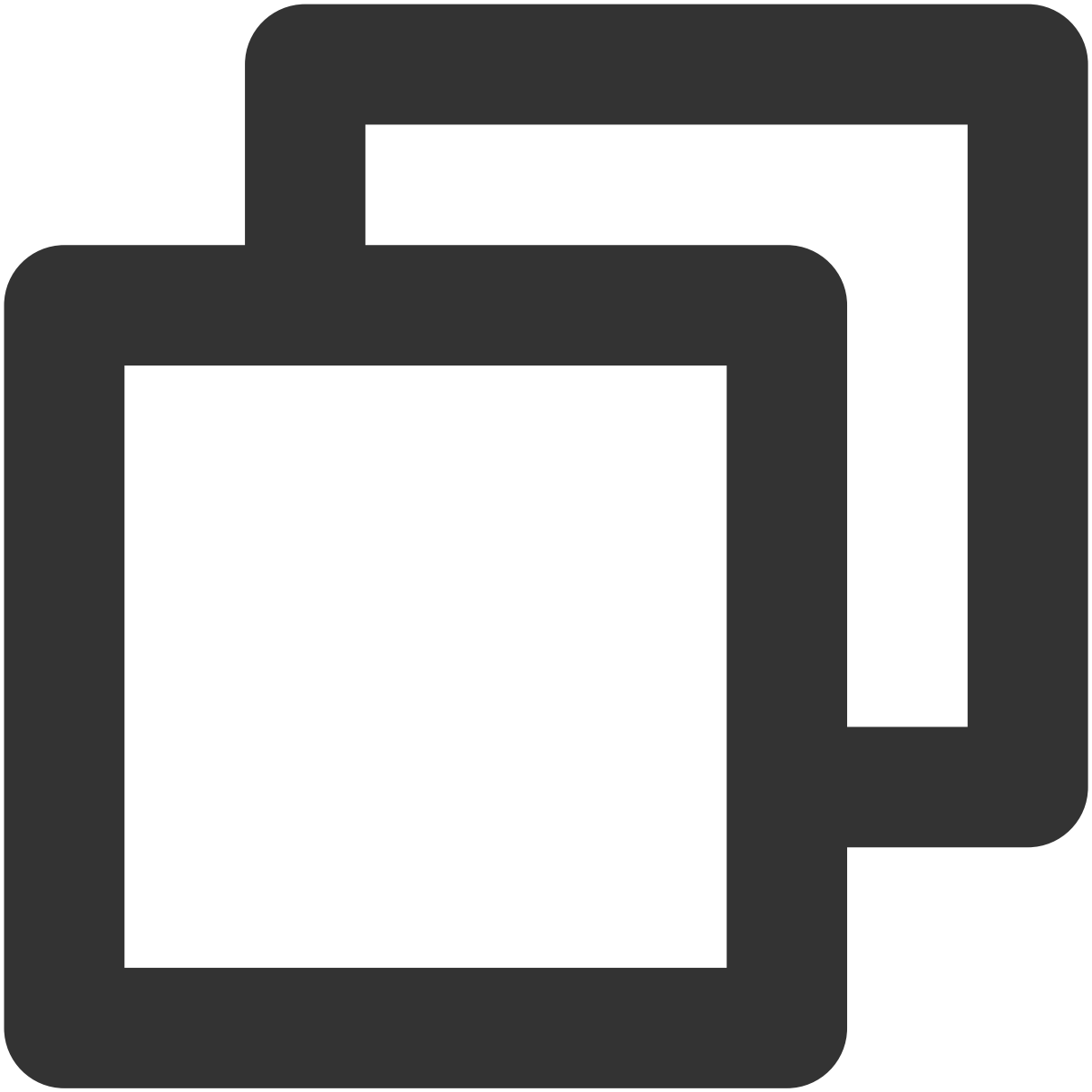
```
/* 组件 custom-component.js */
Component({
  options: {
    addGlobalClass: true,
  }
}
```



```
<!-- 组件 custom-component.wxml -->
<text class="red-text">
  这段文本的颜色由 `app.wxss` 和页面 `wxss` 中的样式定义来决定
```

版权所有：腾讯云计算（北京）有限责任公司

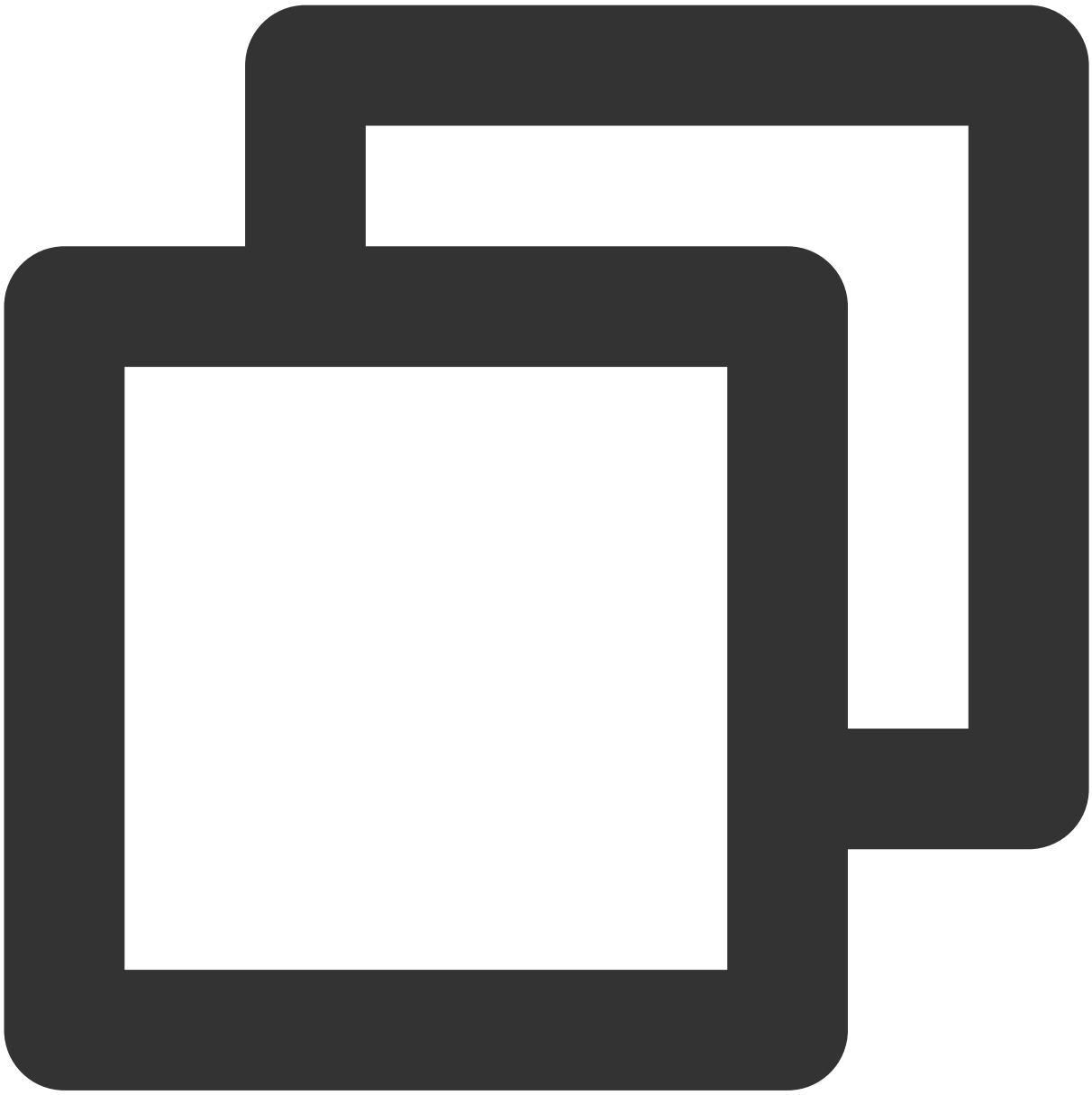
```
</text>
```



```
/* app.wxss */
.red-text {
  color: red;
}
```

3. Component 构造器

`Component` 构造器可用于定义组件，调用 `Component` 构造器时可以指定组件的属性、数据、方法等。详细的参数含义和使用请参考 Component 组件。



```
Component({

  behaviors: [],

  properties: {
    myProperty: { // 属性名
      type: String, // 类型（必填），目前接受的类型包括：String, Number, Boolean, Object,
      value: '', // 属性初始值（可选），如果未指定则会根据类型选择一个
```

```
      observer(newVal, oldVal, changedPath) {
        // 属性被改变时执行的函数（可选），也可以写成在methods段中定义的方法名字符串，如：'_pro
        // 通常 newVal 就是新设置的数据， oldVal 是旧数据
      }
    },
    myProperty2: String // 简化的定义方式
  },
  data: {}, // 私有数据，可用于模板渲染

  lifetimes: {
    // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
    attached() { },
    moved() { },
    detached() { },
  },

  // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
  attached() { }, // 此处attached的声明会被lifetimes字段中的声明覆盖
  ready() { },

  pageLifetimes: {
    // 组件所在页面的生命周期函数
    show() { },
    hide() { },
    resize() { },
  },

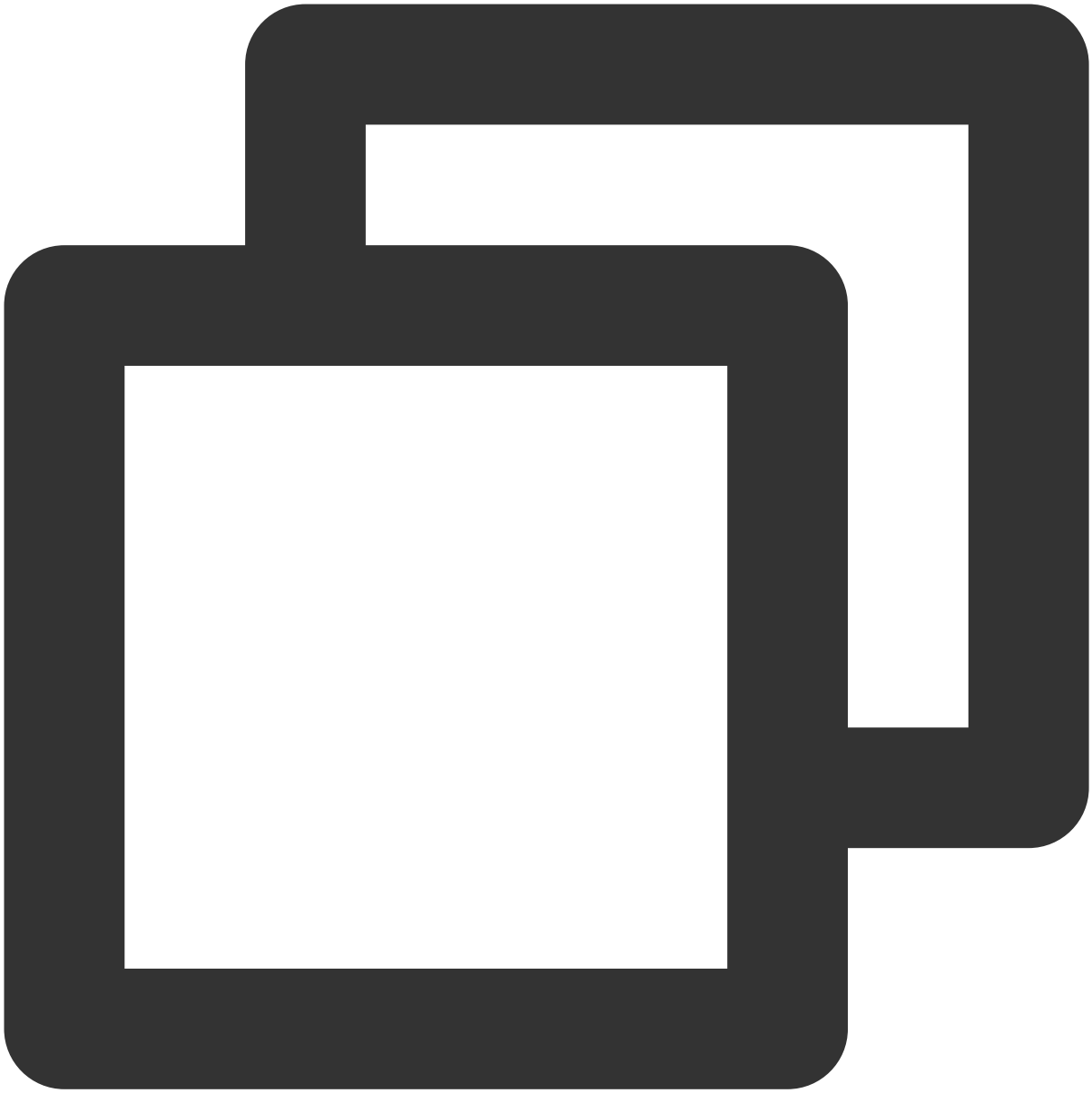
  methods: {
    onMyButtonTap() {
      this.setData({
        // 更新属性和数据的方法与更新页面数据的方法类似
      })
    },
    // 内部方法建议以下划线开头
    _myPrivateMethod() {
      // 这里将 data.A[0].B 设为 'myPrivateData'
      this.setData({
        'A[0].B': 'myPrivateData'
      })
    },
    _propertyChange(newVal, oldVal) {

    }
  }
})
```

在 `properties` 定义段中，属性名采用驼峰写法（ `propertyName` ）；在 wxml 中，指定属性值时则对应使用连字符写法（ `component-tag-name property-name="attr value"` ），应用于数据绑定时采用驼峰写法（ `attr="{{propertyName}}"` ）。

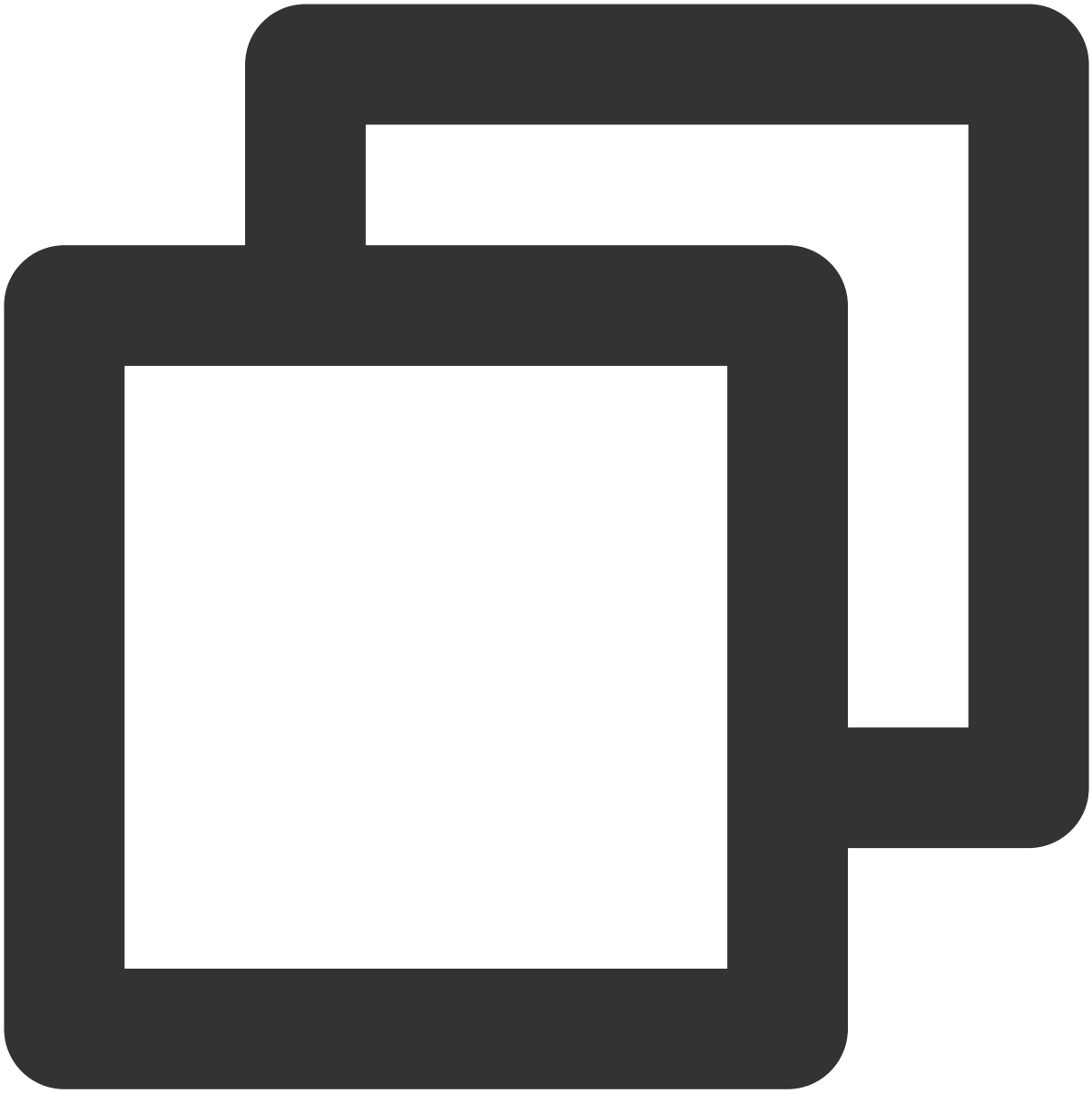
3.1 使用 Component 构造器构造页面

事实上，小程序的页面也可以视为自定义组件。因而，页面也可以使用 `Component` 构造器构造，拥有与普通组件一样的定义段与实例方法。但此时要求对应 json 文件中包含 `usingComponents` 定义段。
此时，组件的属性可以用于接收页面的参数，如访问页面 `/pages/index/index?paramA=123&paramB=xyz` ，如果声明有属性 `paramA` 或 `paramB` ，则它们会被赋值为 `123` 或 `xyz` 。
页面的生命周期方法（即 `on` 开头的方法），应写在 `methods` 定义段中。
代码示例：



```
{
  "usingComponents": {}
}
```

版权所有：腾讯云计算（北京）有限责任公司



```
Component({
  properties: {
    paramA: Number,
    paramB: String,
  },

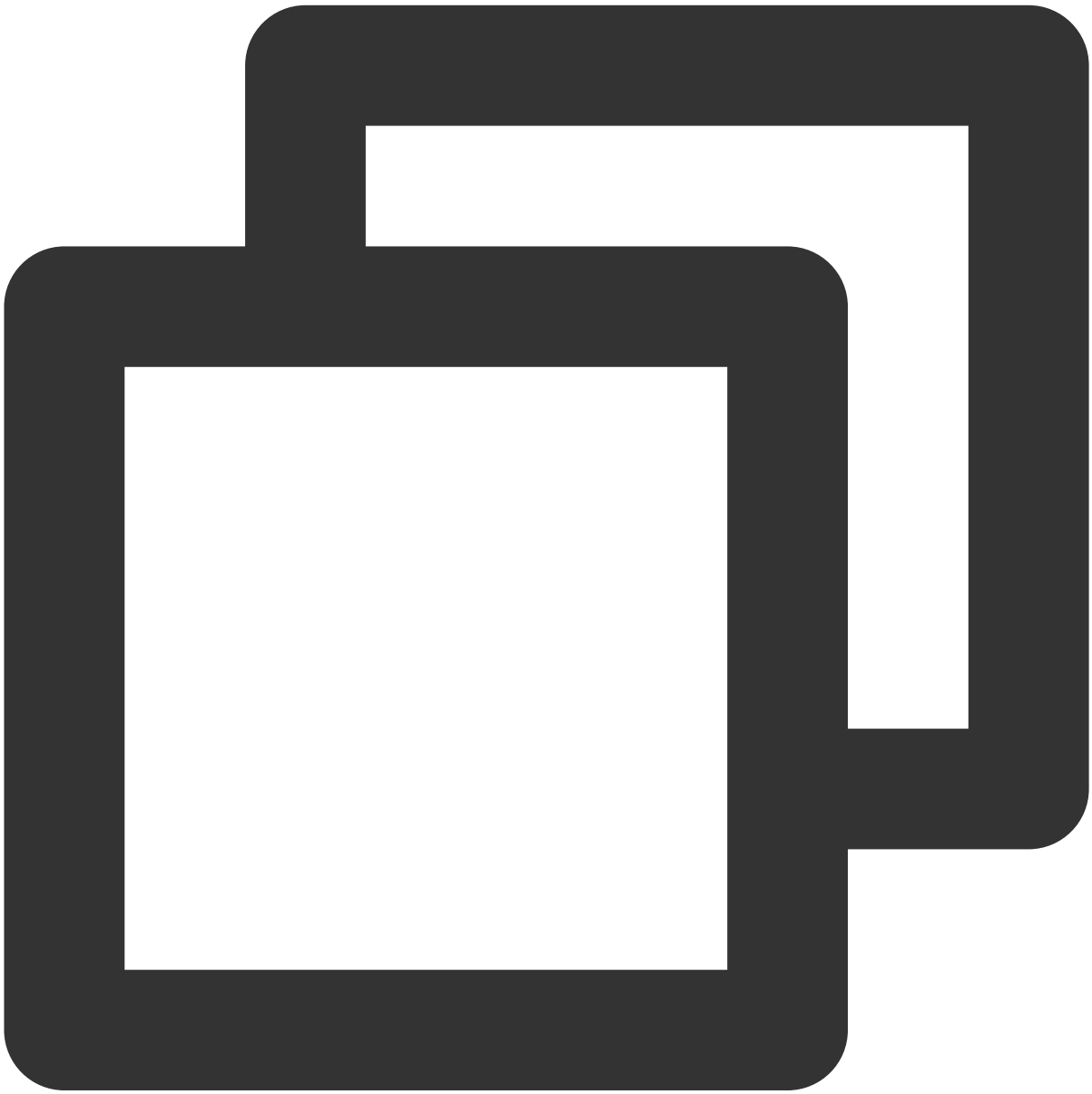
  methods: {
    onLoad: function() {
      this.data.paramA // 页面参数 paramA 的值
      this.data.paramB // 页面参数 paramB 的值
    }
```

版权所有：腾讯云计算（北京）有限责任公司

```
  }
})
```

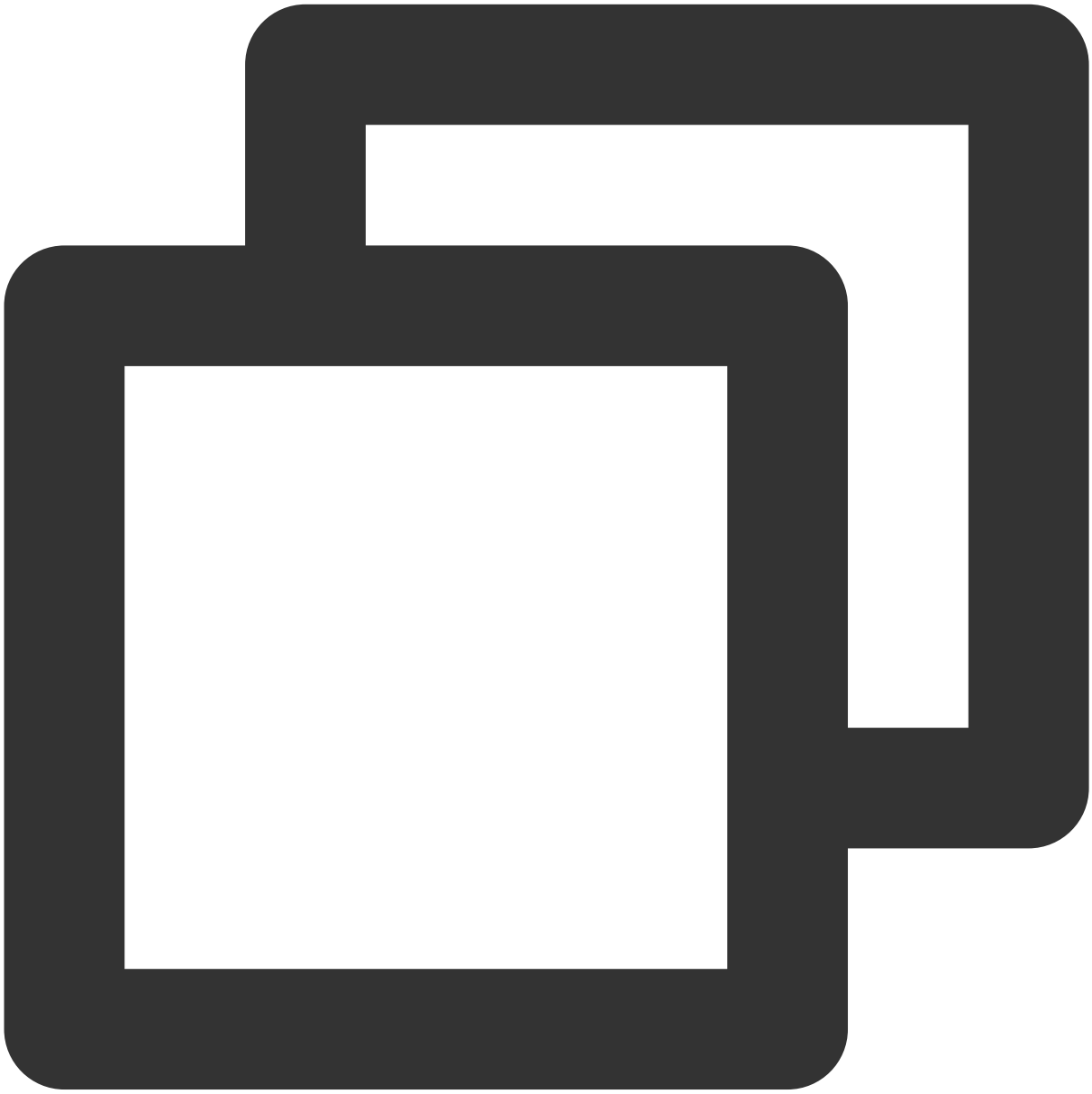
使用 `Component` 构造器来构造页面的一个好处是可以使用 `behaviors` 来提取所有页面中公用的代码段。例如，在所有页面被创建和销毁时都要执行同一段代码，就可以把这段代码提取到 `behaviors` 中。

代码示例：



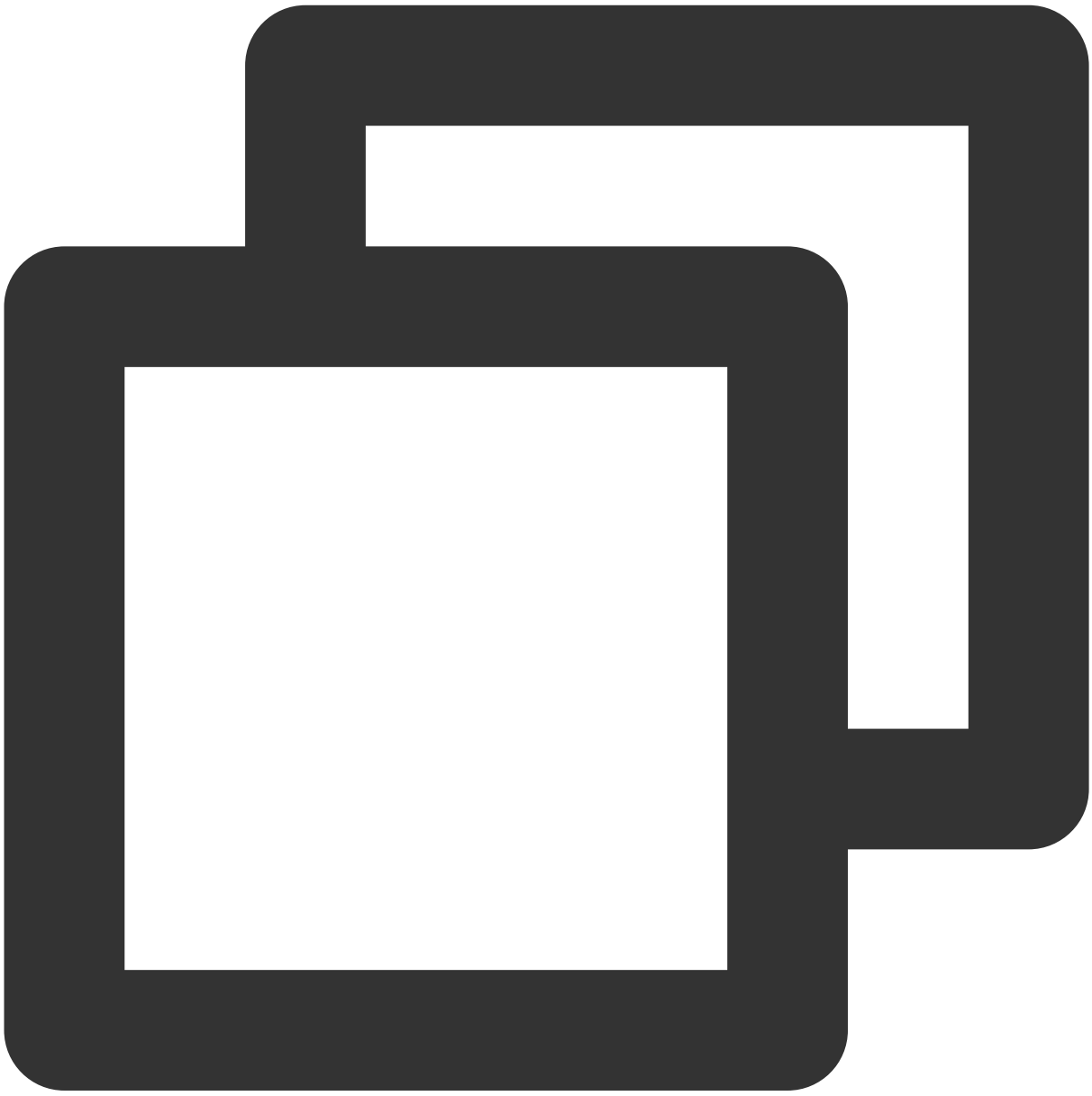
```
// page-common-behavior.js
module.exports = Behavior({
  attached: function() {
    // 页面创建时执行
```

```
    console.info('Page loaded!')
  },
  detached: function() {
    // 页面销毁时执行
    console.info('Page unloaded!')
  }
})
```



```
// 页面 A
var pageCommonBehavior = require('./page-common-behavior')
Component({
```

```
  behaviors: [pageCommonBehavior],
  data: { /* ... */ },
  methods: { /* ... */ },
})
```



```
// 页面 B
var pageCommonBehavior = require('./page-common-behavior')
Component({
  behaviors: [pageCommonBehavior],
  data: { /* ... */ },
  methods: { /* ... */ },
```

```
})
```

4. 组件间通信与事件

4.1 组件间通信

组件间的基本通信方式有以下几种。

WXML 数据绑定：用于父组件向子组件的指定属性设置数据，仅能设置 JSON 兼容数据。具体在 组件模板和样式 章节中介绍。

事件：用于子组件向父组件传递数据，可以传递任意数据。

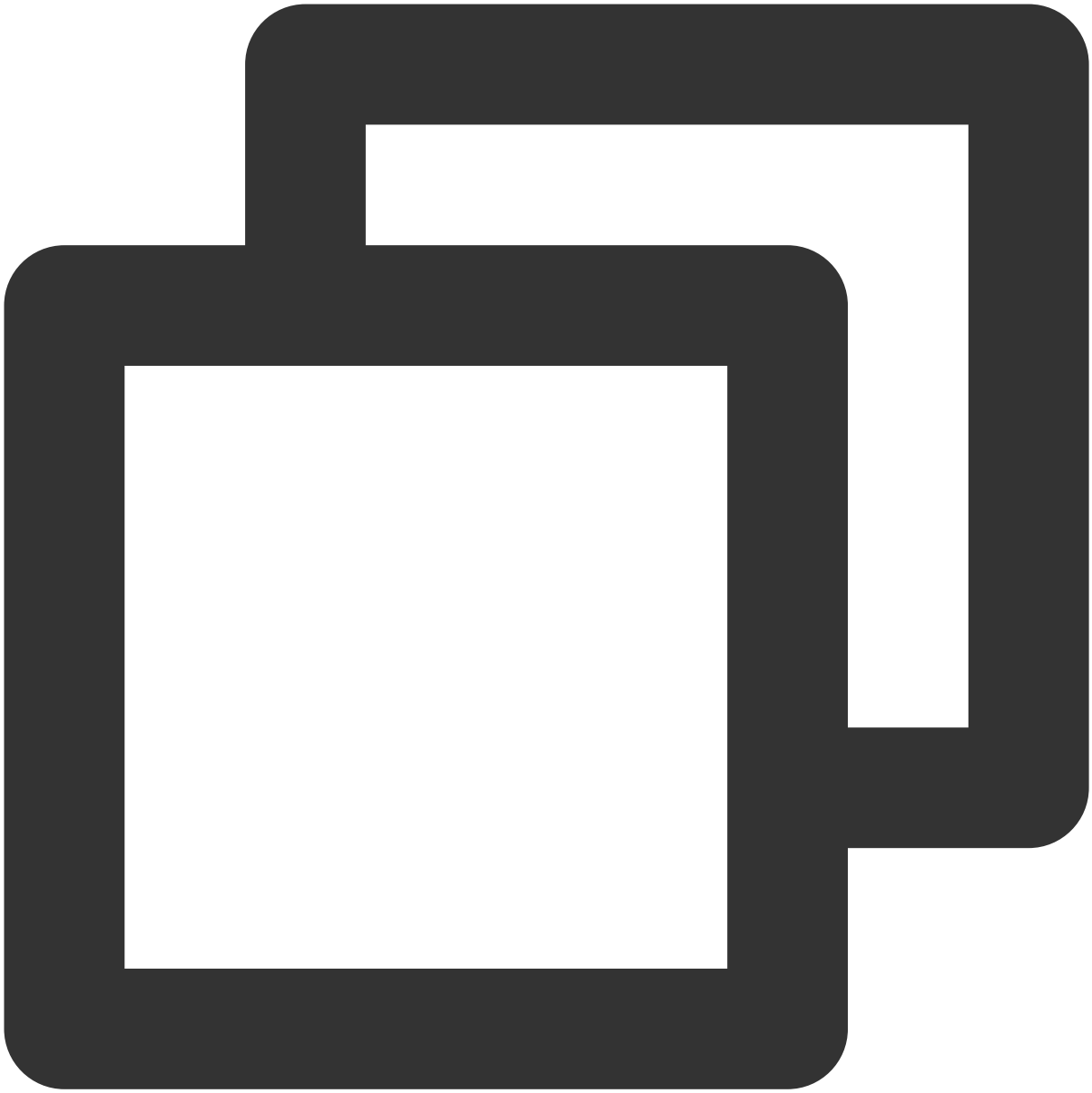
如果以上两种方式不足以满足需要，父组件还可以通过 `this.selectComponent` 方法获取子组件实例对象，这样就可以直接访问组件的任意数据和方法。

4.2 监听事件

事件系统是组件间通信的主要方式之一。自定义组件可以触发任意的事件，引用组件的页面可以监听这些事件。关于事件的基本概念和用法，参见 事件。

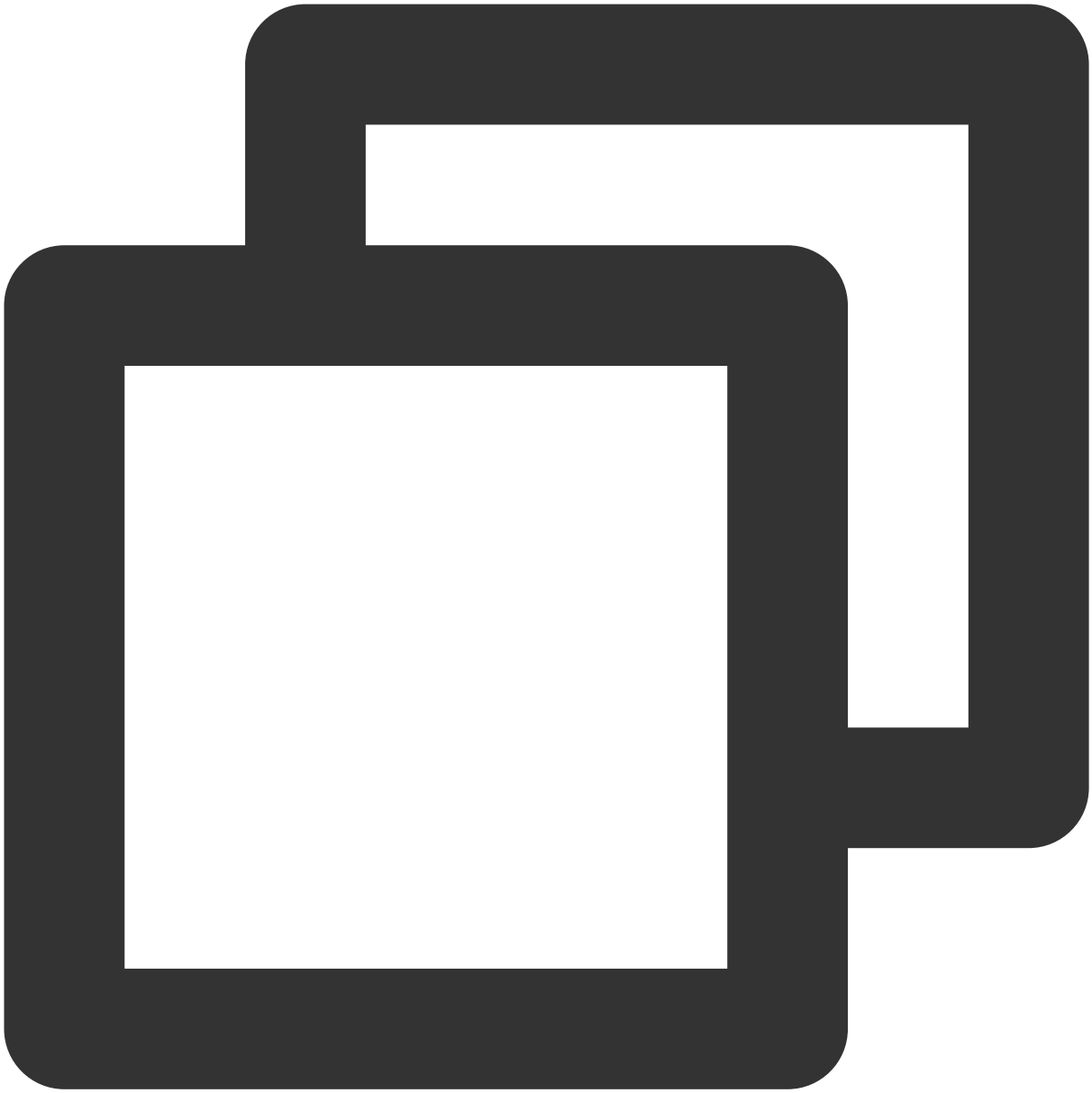
监听自定义组件事件的方法与监听基础组件事件的方法完全一致：

代码示例：



```
<!-- 当自定义组件触发“myevent”事件时，调用“onMyEvent”方法 -->
<component-tag-name bindmyevent="onMyEvent" />
<!-- 或者可以写成 -->
<component-tag-name bind:myevent="onMyEvent" />
```

版权所有：腾讯云计算（北京）有限责任公司

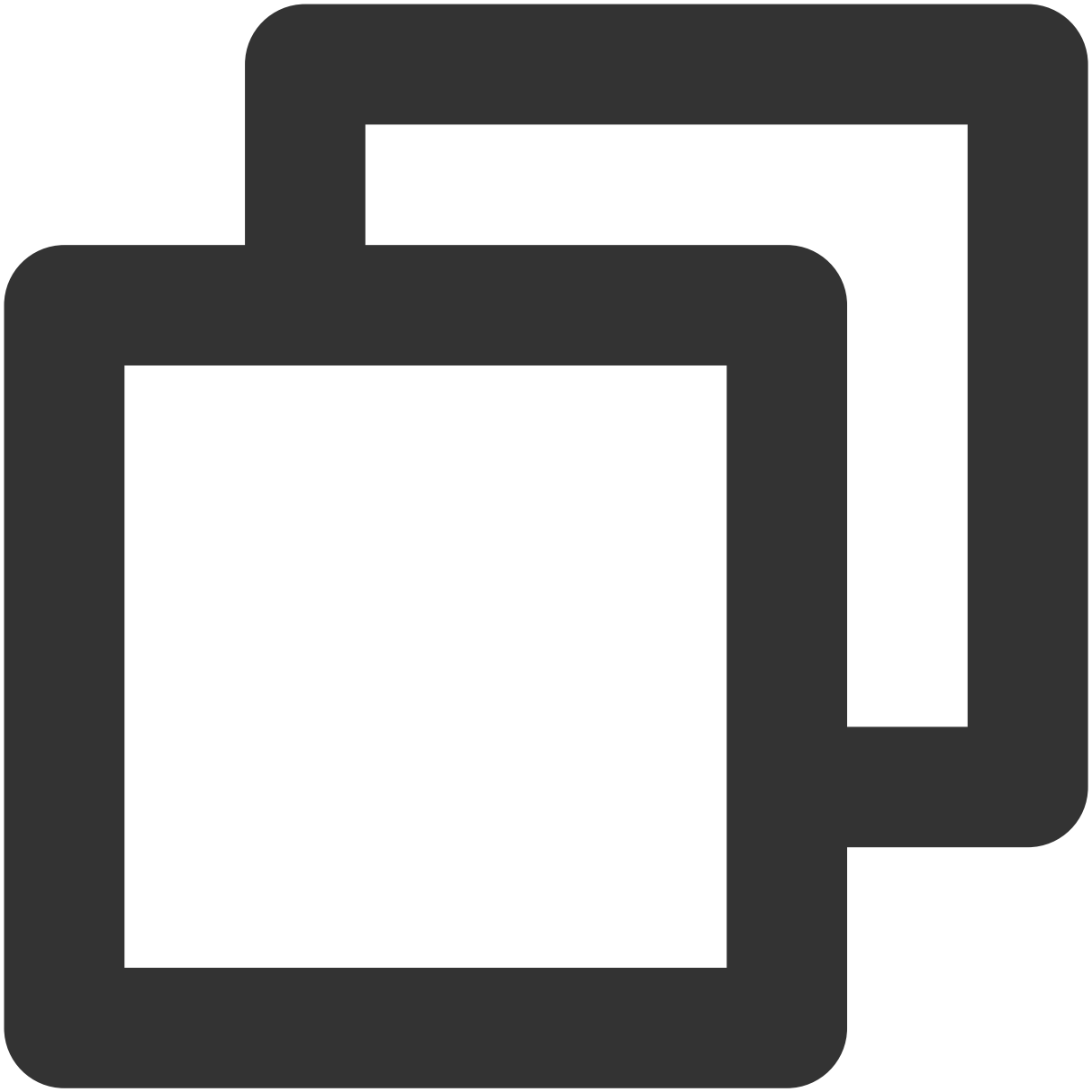


```
Page({
  onMyEvent: function(e){
    e.detail // 自定义组件触发事件时提供的detail对象
  }
})
```

4.3 触发事件

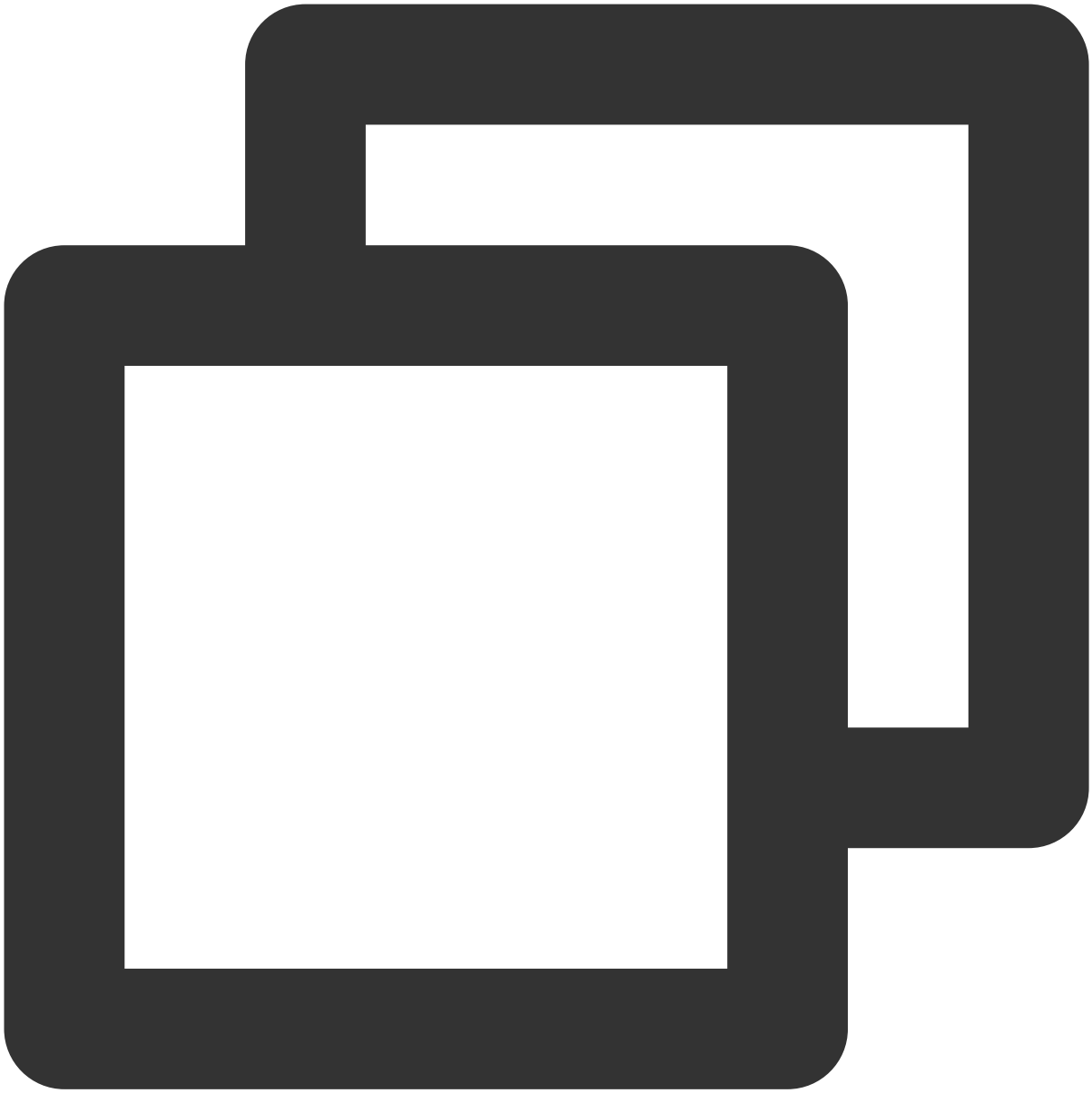
自定义组件触发事件时，需要使用 `triggerEvent` 方法，指定事件名、detail 对象和事件选项：

代码示例：



```
<!-- 在自定义组件中 -->
<button bindtap="onTap">点击这个按钮将触发“myevent”事件</button>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Component({
  properties: {},
  methods: {
    onTap: function(){
      var myEventDetail = {} // detail对象，提供给事件监听函数
      var myEventOption = {} // 触发事件的选项
      this.triggerEvent('myevent', myEventDetail, myEventOption)
    }
  }
})
```

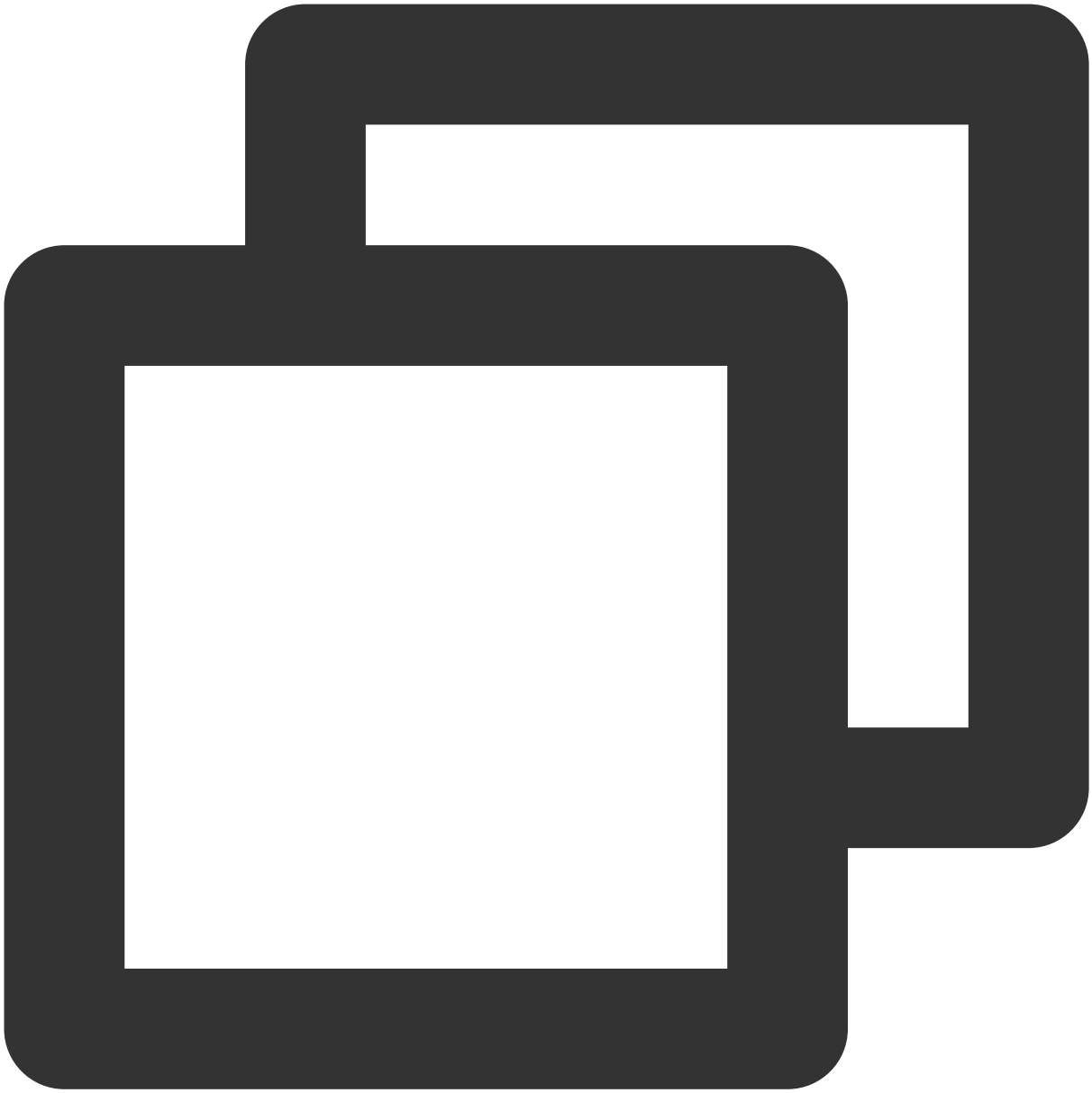
版权所有：腾讯云计算（北京）有限责任公司

触发事件的选项包括：

选项名	类型	是否必填	默认值	描述
bubbles	Boolean	否	false	事件是否冒泡
composed	Boolean	否	false	事件是否可以穿越组件边界，为 false 时，事件将只能在引用组件的节点树上触发，不进入其他任何组件内部
capturePhase	Boolean	否	false	事件是否拥有捕获阶段

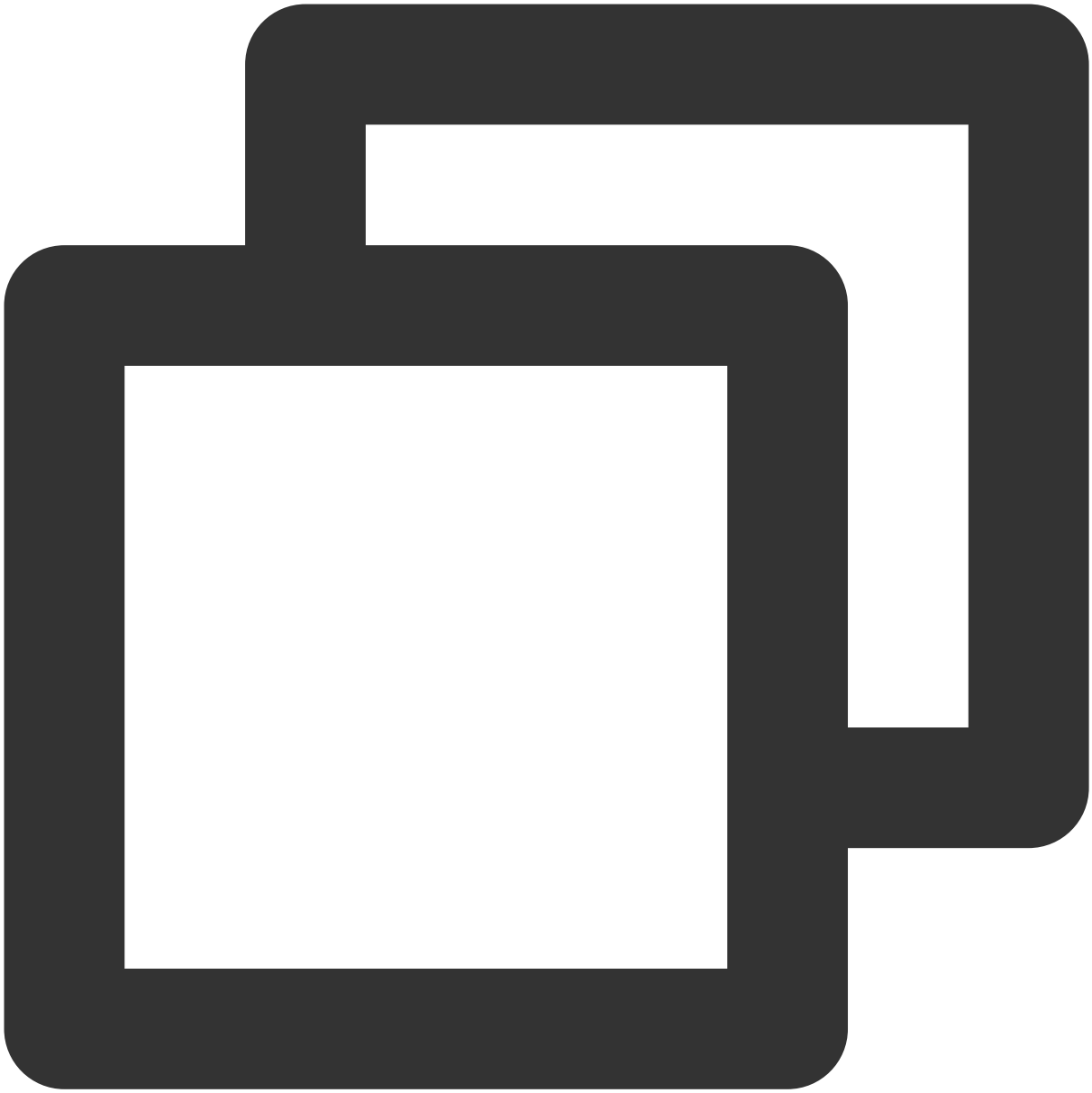
关于冒泡和捕获阶段的概念，请阅读 事件详解 章节中的相关说明。

代码示例：



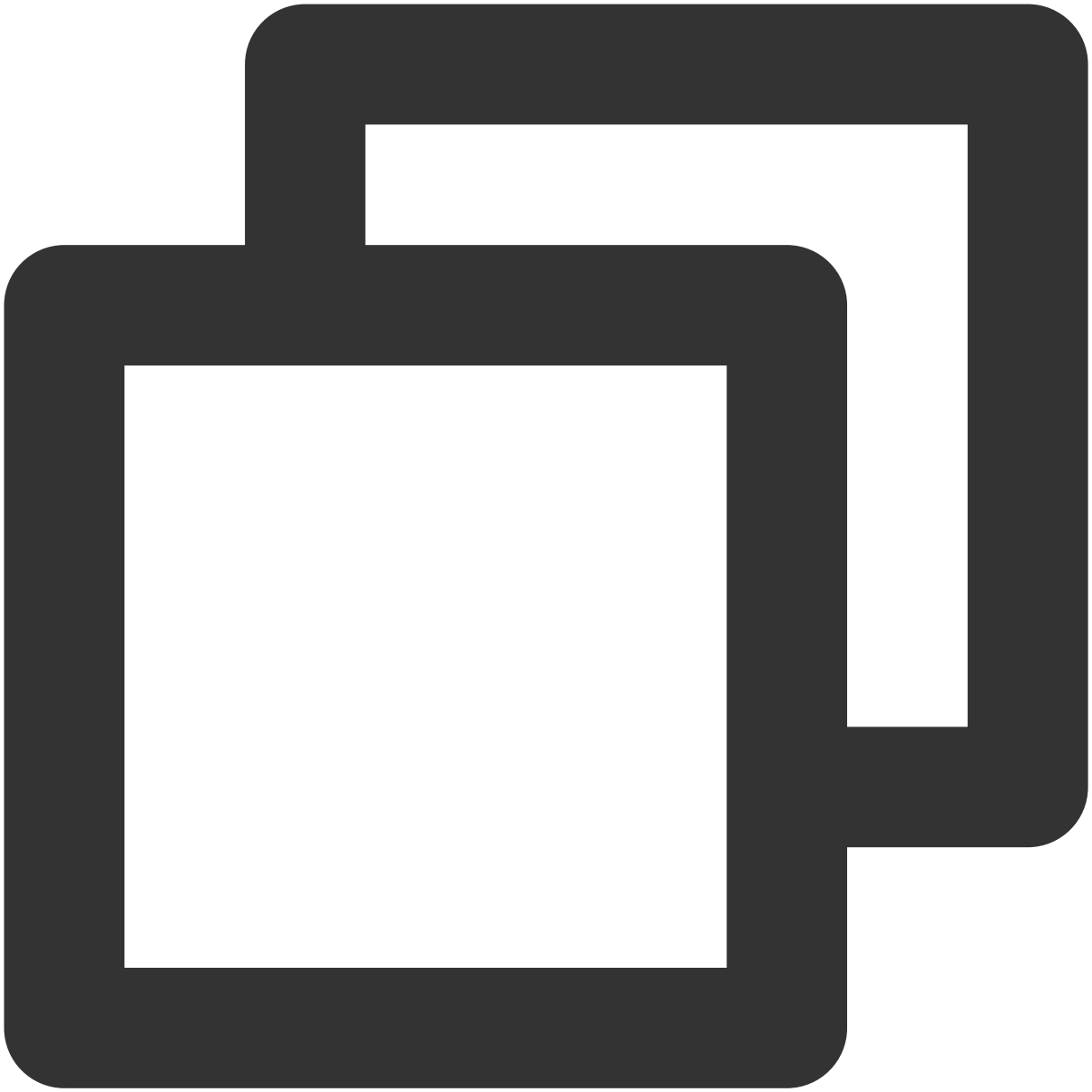
```
// 页面 page.wxml
<another-component bindcustomevent="pageEventListener1">
  <my-component bindcustomevent="pageEventListener2"></my-component>
</another-component>
```

版权所有：腾讯云计算（北京）有限责任公司



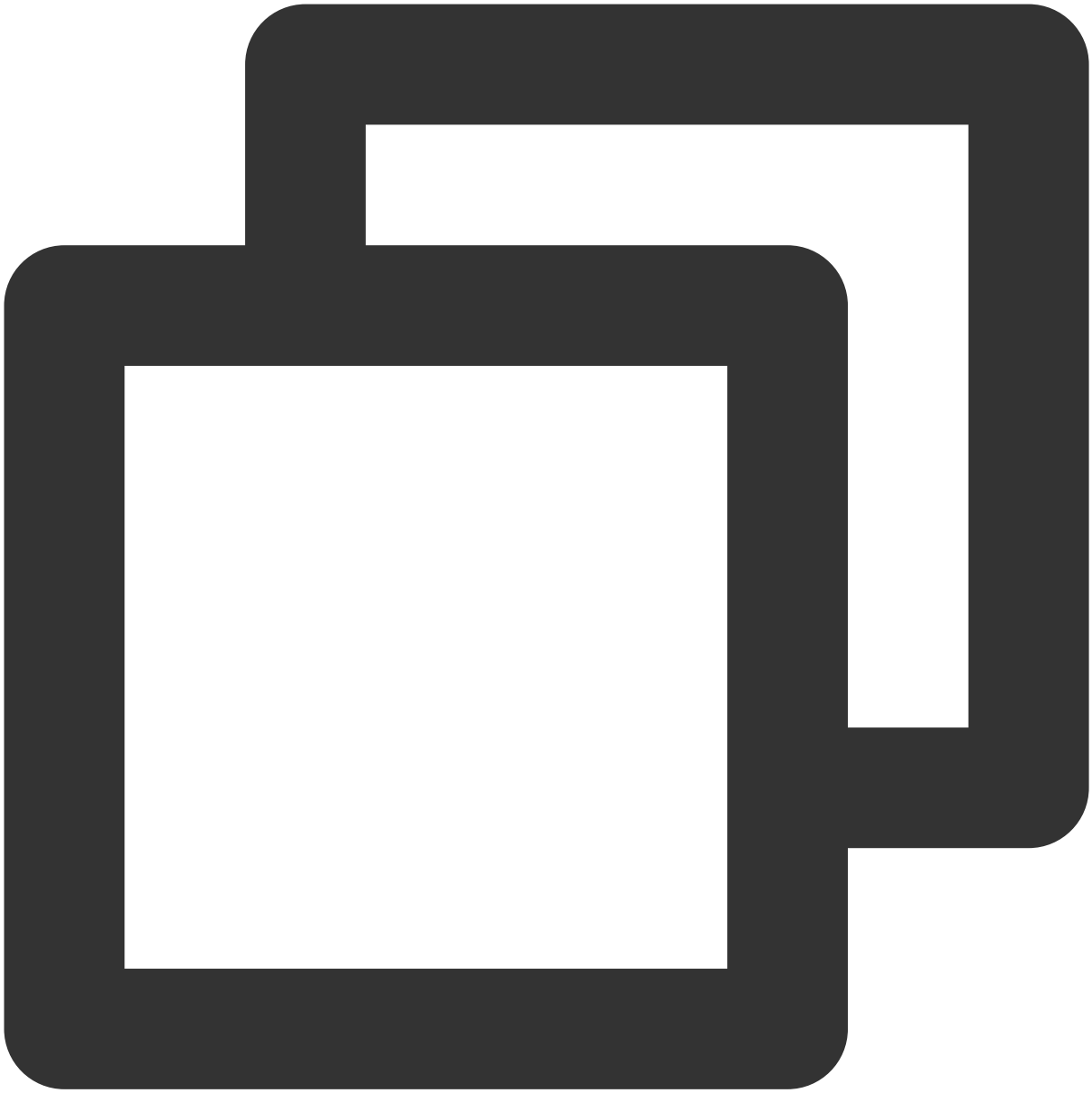
```
// 组件 another-component.wxml
<view bindcustomevent="anotherEventListener">
  <slot />
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// 组件 my-component.wxml
<view bindcustomevent="myEventListener">
  <slot />
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// 组件 my-component.js
Component({
  methods: {
    onTap: function(){
      this.triggerEvent('customevent', {}) // 只会触发 pageEventListener2
      this.triggerEvent('customevent', {}, { bubbles: true }) // 会依次触发 pageEvent
      this.triggerEvent('customevent', {}, { bubbles: true, composed: true }) // 会
    }
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

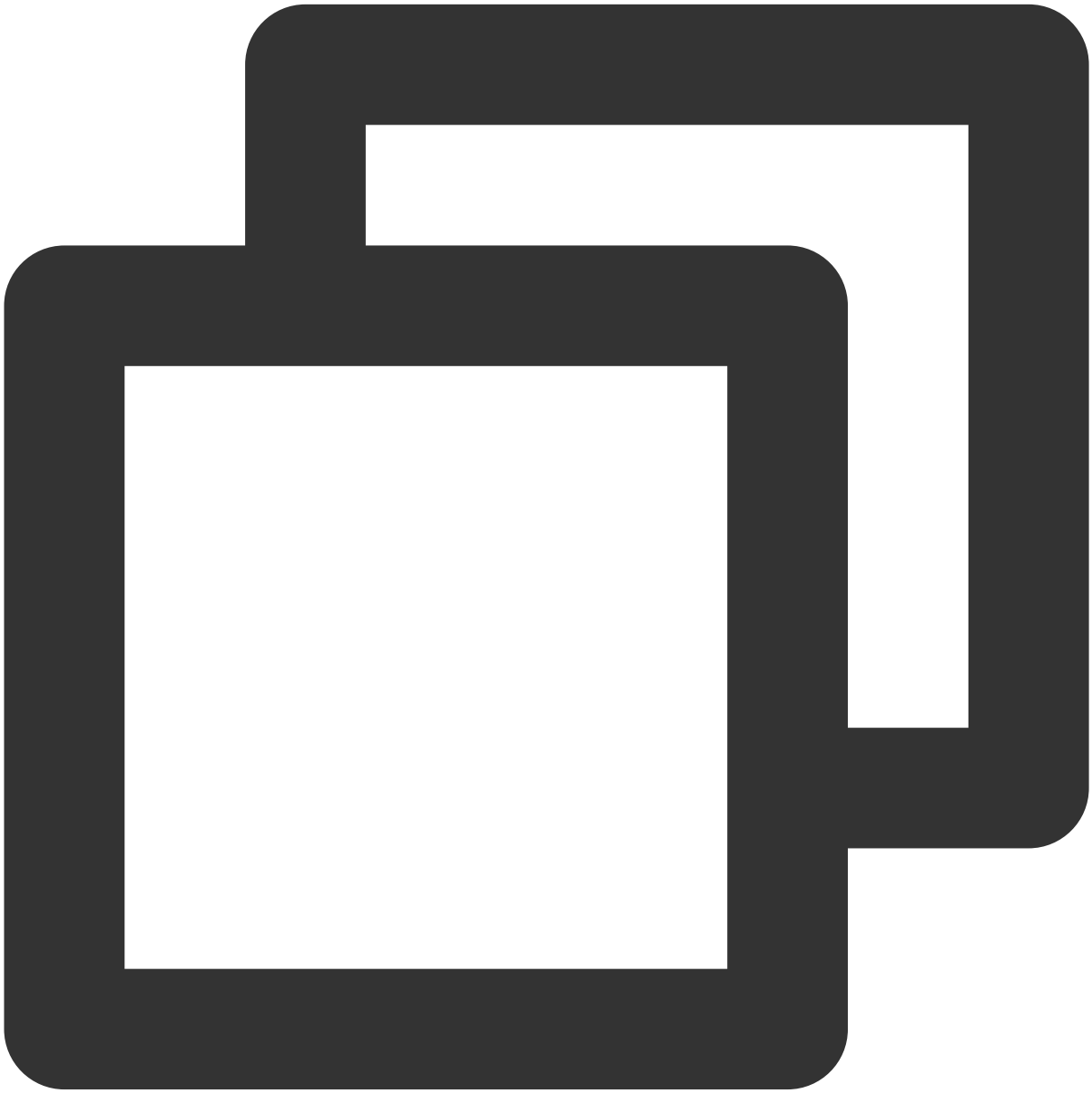
4.4 获取组件实例

可在父组件里调用 `this.selectComponent` ，获取子组件的实例对象。

调用时需要传入一个匹配选择器 `selector` ，如： `this.selectComponent(".my-component")` 。

`selector` 详细语法可查看 selector 语法参考文档。

代码示例：



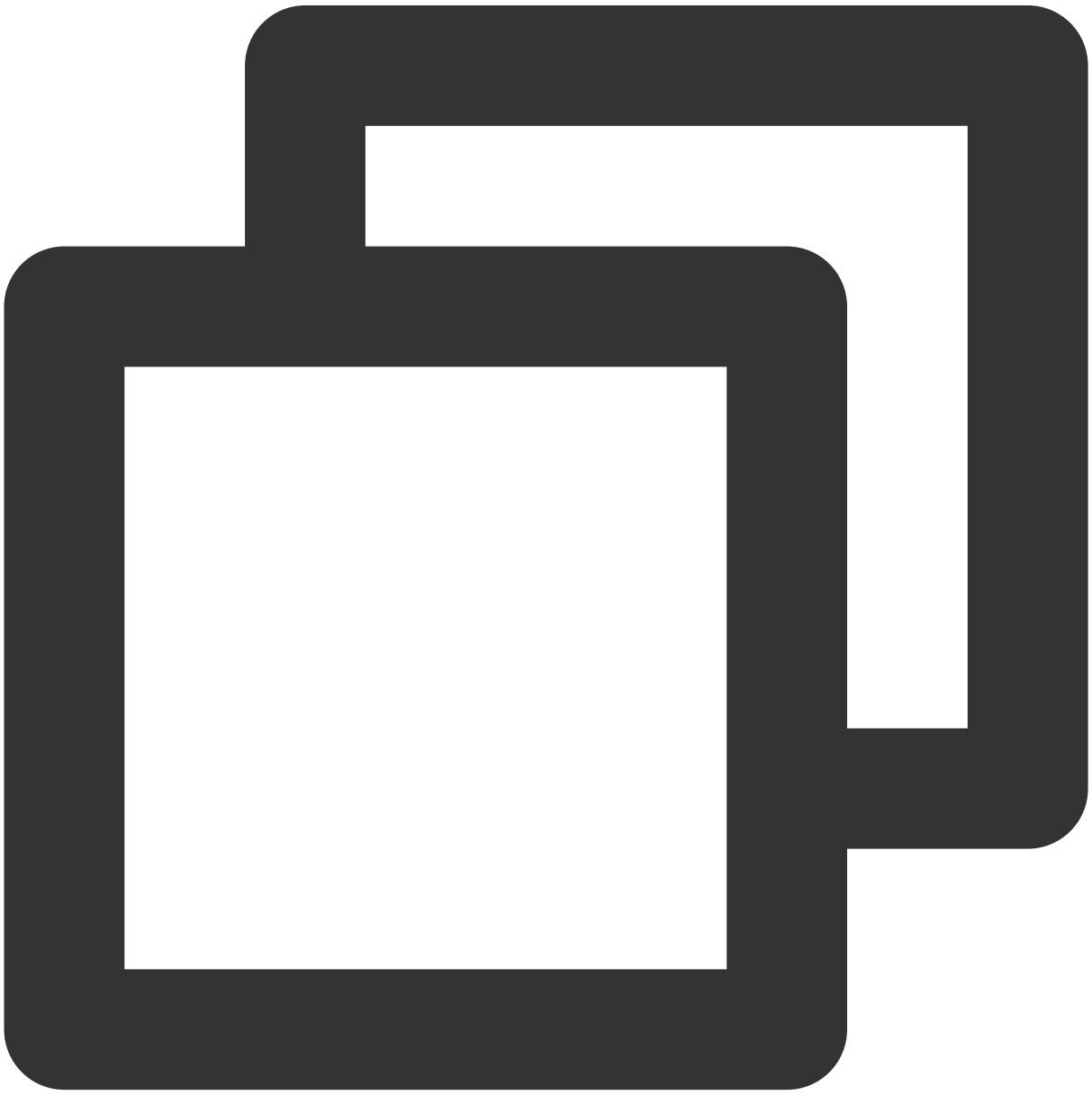
```
// 父组件
Page({
  data: {},
  getChildComponent: function () {
```

```
    const child = this.selectComponent('.my-component');
    console.log(child)
  }
})
```

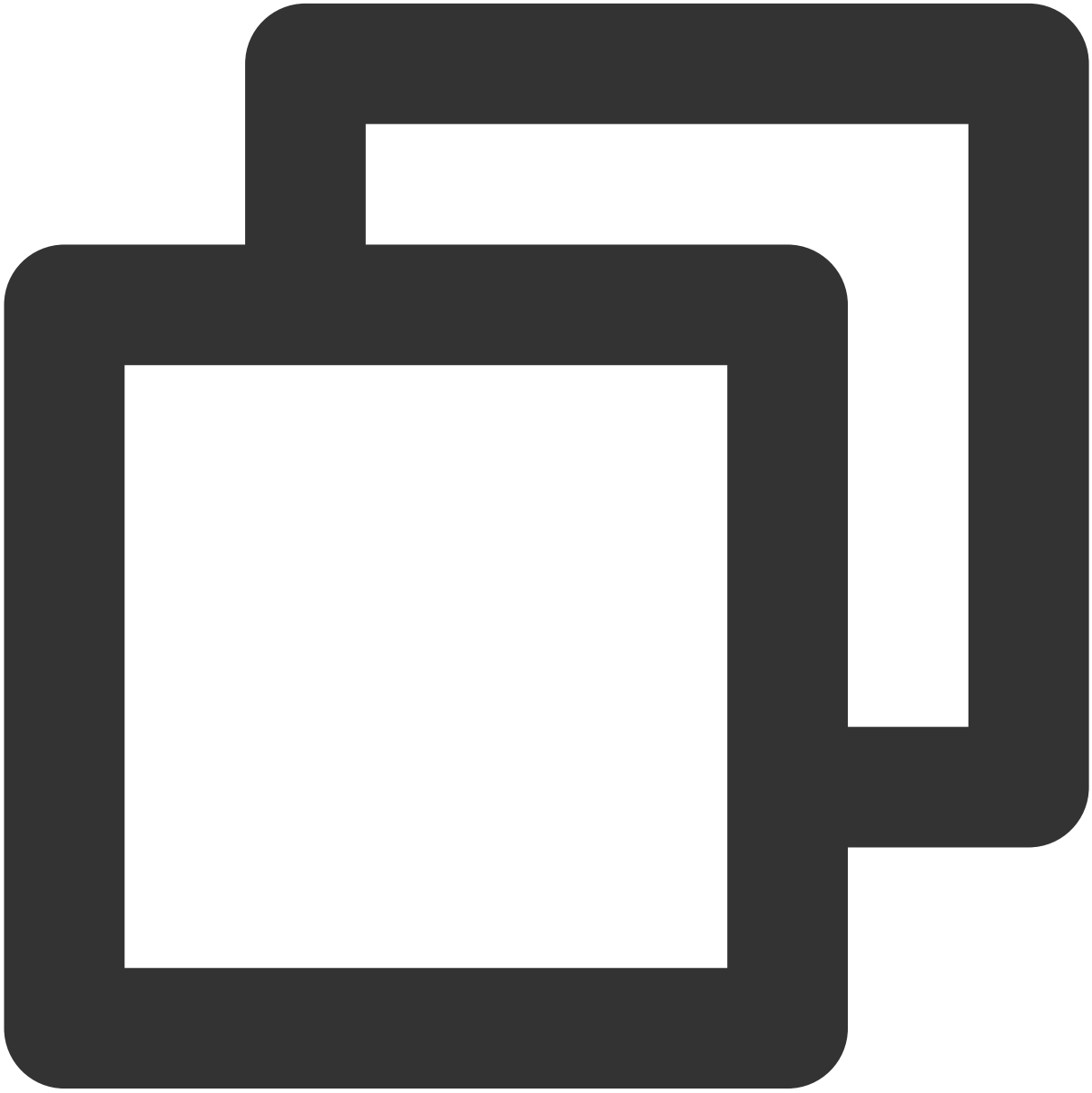
在上例中，父组件将会获取 `class` 为 `my-component` 的子组件实例对象，即子组件的 this 。

若需要自定义 `selectComponent` 返回的数据，可使用内置 `behavior: wx://component-export` 使用该 `behavior` 时，自定义组件中的 `export` 定义段将用于指定组件被 `selectComponent` 调用时的返回值。

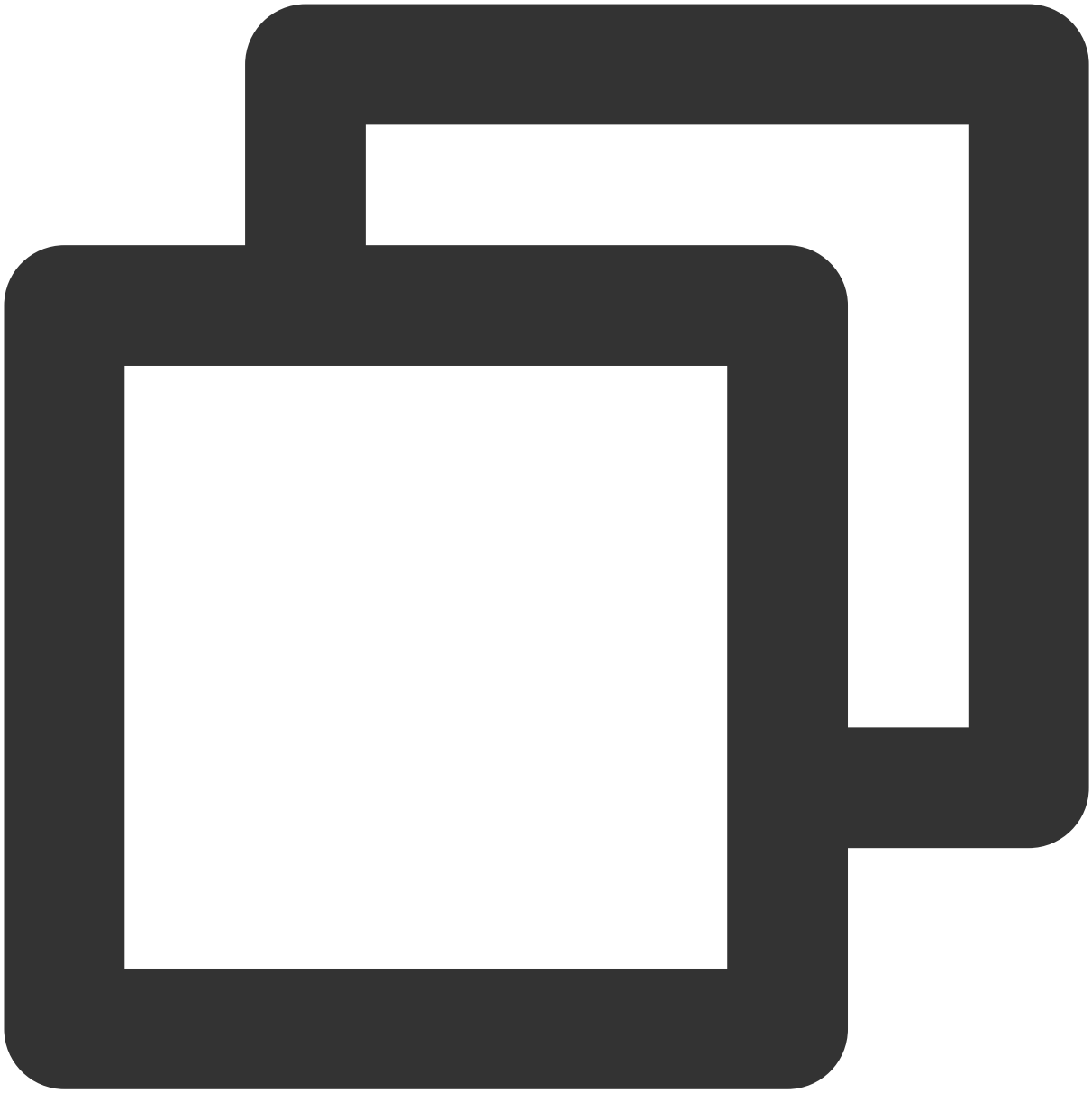
代码示例：



```
// 自定义组件 my-component 内部
Component({
  behaviors: ['wx://component-export'],
  export() {
    return { myField: 'myValue' }
  }
})
```



```
<!-- 使用自定义组件时 -->
<my-component id="the-id" />
```



```
// 父组件调用
const child = this.selectComponent('#the-id') // 等于 { myField: 'myValue' }
```

在上例中，父组件获取 `id` 为 `the-id` 的子组件实例的时候，得到的是对象 `{ myField: 'myValue' }`。

5. 组件生命周期

组件的生命周期，指的是组件自身的一些函数，这些函数在特殊的时间点或遇到一些特殊的框架事件时被自动触发。

其中，最重要的生命周期是 `created` `attached` `detached` ，包含一个组件实例生命流程的最主要时间点。

组件实例刚刚被创建好时， `created` 生命周期被触发。此时，组件数据 `this.data` 就是在 `Component` 构造器中定义的数据 `data` 。 **此时还不能调用 `setData`** 。 通常情况下，这个生命周期只应该用于给组件 `this` 添加一些自定义属性字段。

在组件完全初始化完毕、进入页面节点树后， `attached` 生命周期被触发。此时， `this.data` 已被初始化为组件的当前值。这个生命周期很有用，绝大多数初始化工作可以在这个时机进行。

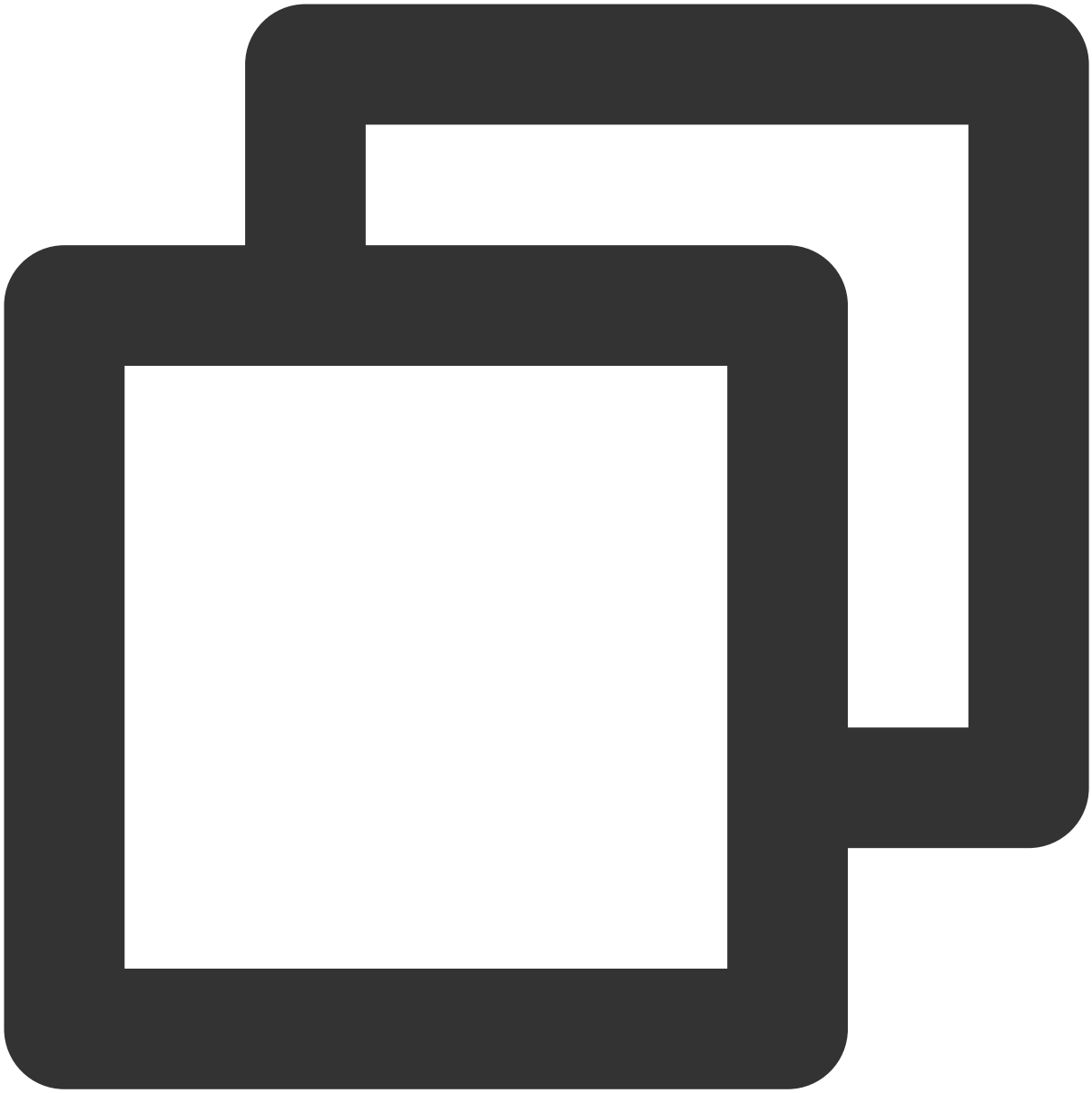
在组件离开页面节点树后， `detached` 生命周期被触发。退出一个页面时，如果组件还在页面节点树中，则 `detached` 会被触发。

5.1 定义生命周期方法

生命周期方法可以直接定义在 `Component` 构造器的第一级参数中。

组件的生命周期也可以在 `lifetimes` 字段内进行声明（这是推荐的方式，其优先级最高）。

代码示例：



```
Component({
  lifetimes: {
    attached: function() {
      // 在组件实例进入页面节点树时执行
    },
    detached: function() {
      // 在组件实例被从页面节点树移除时执行
    },
  },
  // 以下是旧式的定义方式，可以保持对 <1.0.2 版本基础库的兼容
  attached: function() {
```

```
    // 在组件实例进入页面节点树时执行
  },
  detached: function() {
    // 在组件实例被从页面节点树移除时执行
  },
  // ...
})
```

在 behaviors 中也可以编写生命周期方法，同时不会与其他 behaviors 中的同名生命周期相互覆盖。但要注意，如果一个组件多次直接或间接引用同一个 behavior ，这个 behavior 中的生命周期函数在一个执行时机内只会执行一次。可用的全部生命周期如下表所示。

生命周期	参数	描述
created	无	在组件实例刚刚被创建时执行
attached	无	在组件实例进入页面节点树时执行
ready	无	在组件在视图层布局完成后执行
moved	无	在组件实例被移动到节点树另一个位置时执行
detached	无	在组件实例被从页面节点树移除时执行
error	`Object Error`	每当组件方法抛出错误时执行

5.2 组件所在页面的生命周期

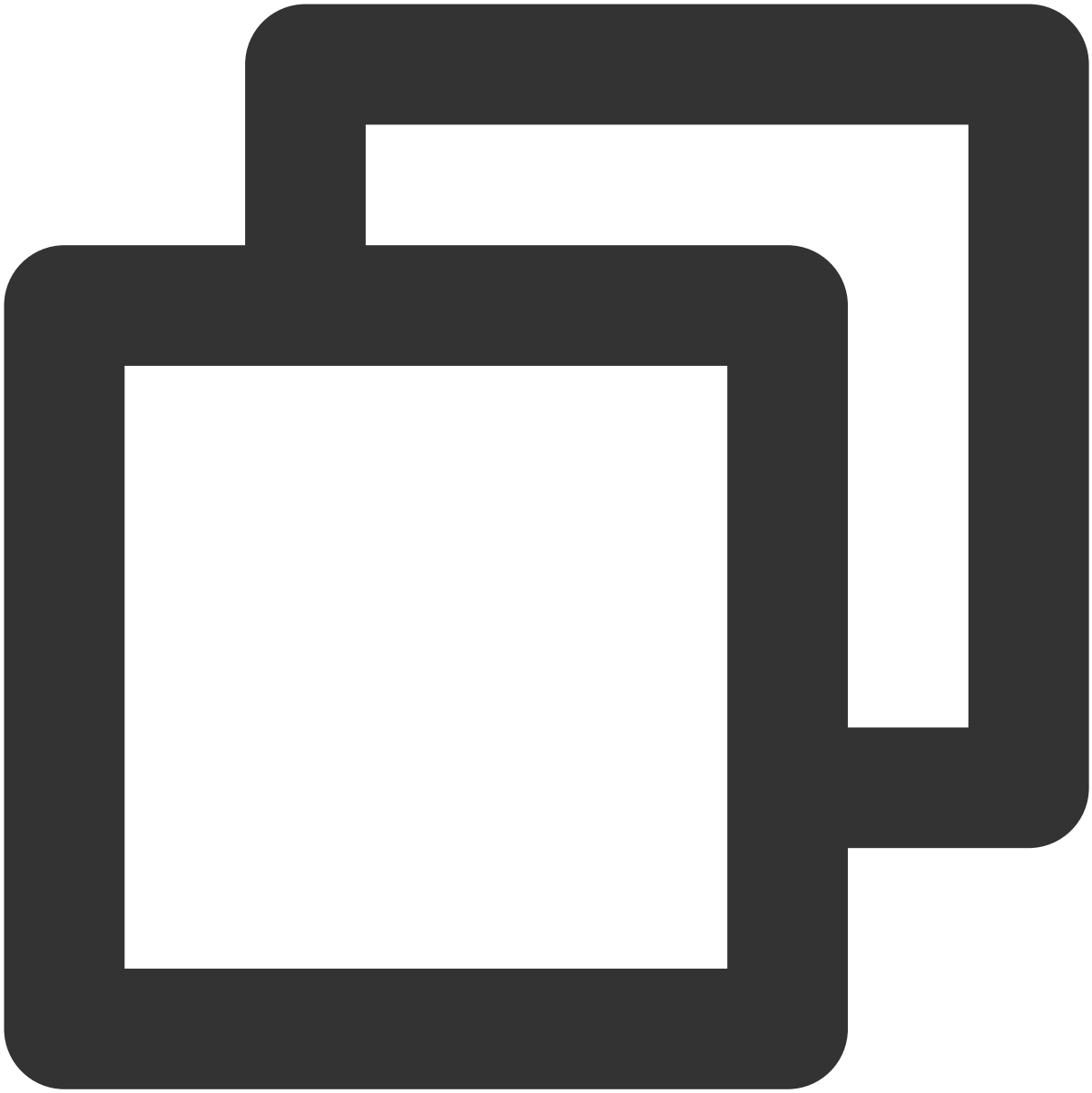
还有一些特殊的生命周期，它们并非与组件有很强的关联，但有时组件需要获知，以便组件内部处理。这样的生命周期称为“组件所在页面的生命周期”，在 `pageLifetimes` 定义段中定义。其中可用的生命周期包括：

生命周期	参数	描述
show	无	组件所在的页面被展示时执行
hide	无	组件所在的页面被隐藏时执行
resize	`Object Size`	组件所在的页面尺寸变化时执行

注意：

自定义 tabBar 的 pageLifetime 不会触发。

代码示例：



```
Component({
  pageLifetimes: {
    show: function() {
      // 页面被展示
    },
    hide: function() {
      // 页面被隐藏
    },
    resize: function(size) {
      // 页面尺寸变化
    }
```

版权所有：腾讯云计算（北京）有限责任公司

```
    }
})
```

6. behaviors

`behaviors` 是用于组件间代码共享的特性，类似于一些编程语言中的 “mixins” 或 “traits”。

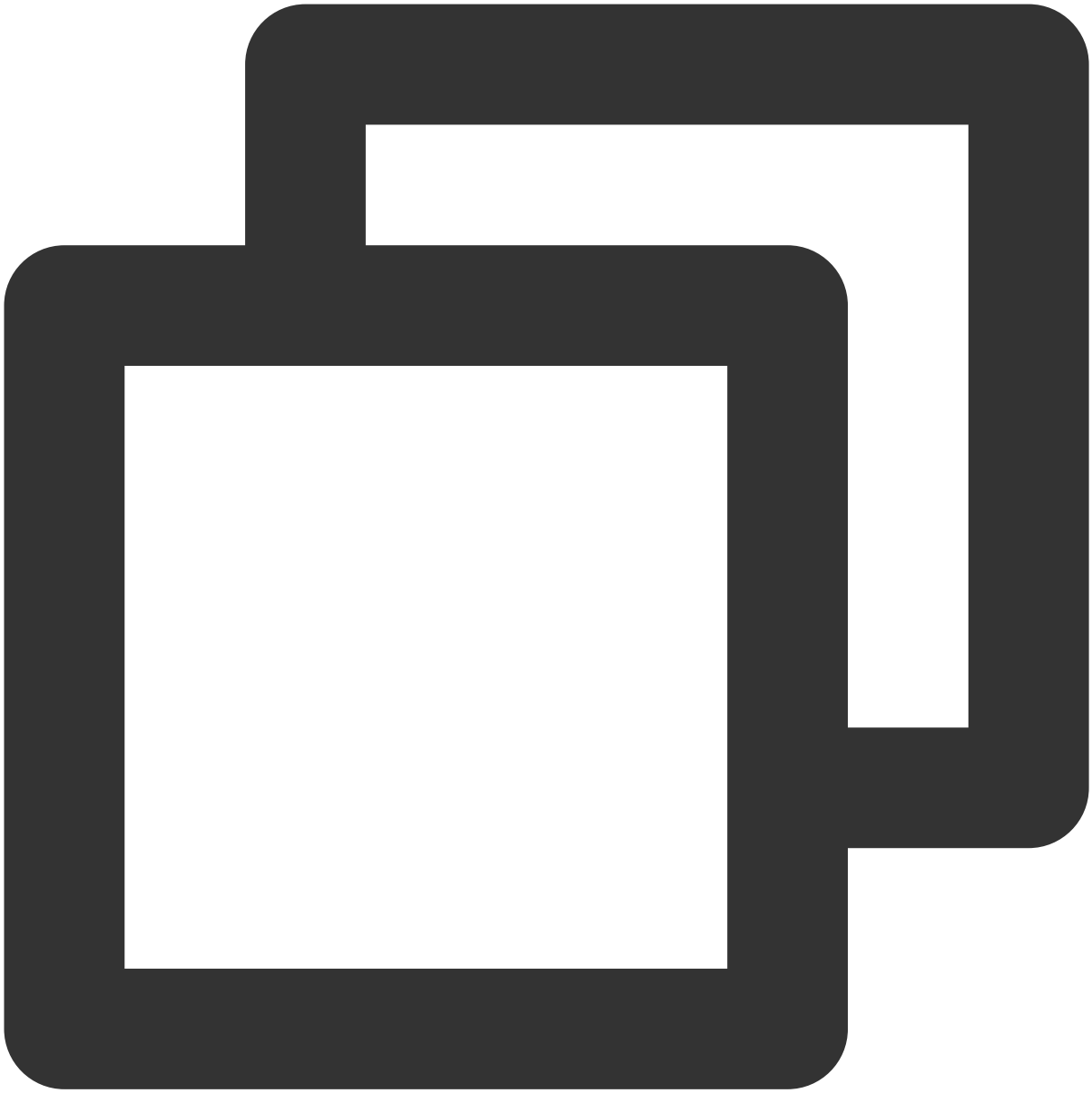
每个 `behavior` 可以包含一组属性、数据、生命周期函数和方法。**组件引用它时，它的属性、数据和方法会被合并到组件中，生命周期函数也会在对应时机被调用**。 每个组件可以引用多个 `behavior` ， `behavior` 也可以引用其它 `behavior` 。

详细的参数含义和使用请参考 Behavior 参考文档。

6.1 组件中使用

`behavior` 需要使用 `Behavior()` 构造器定义。

代码示例：

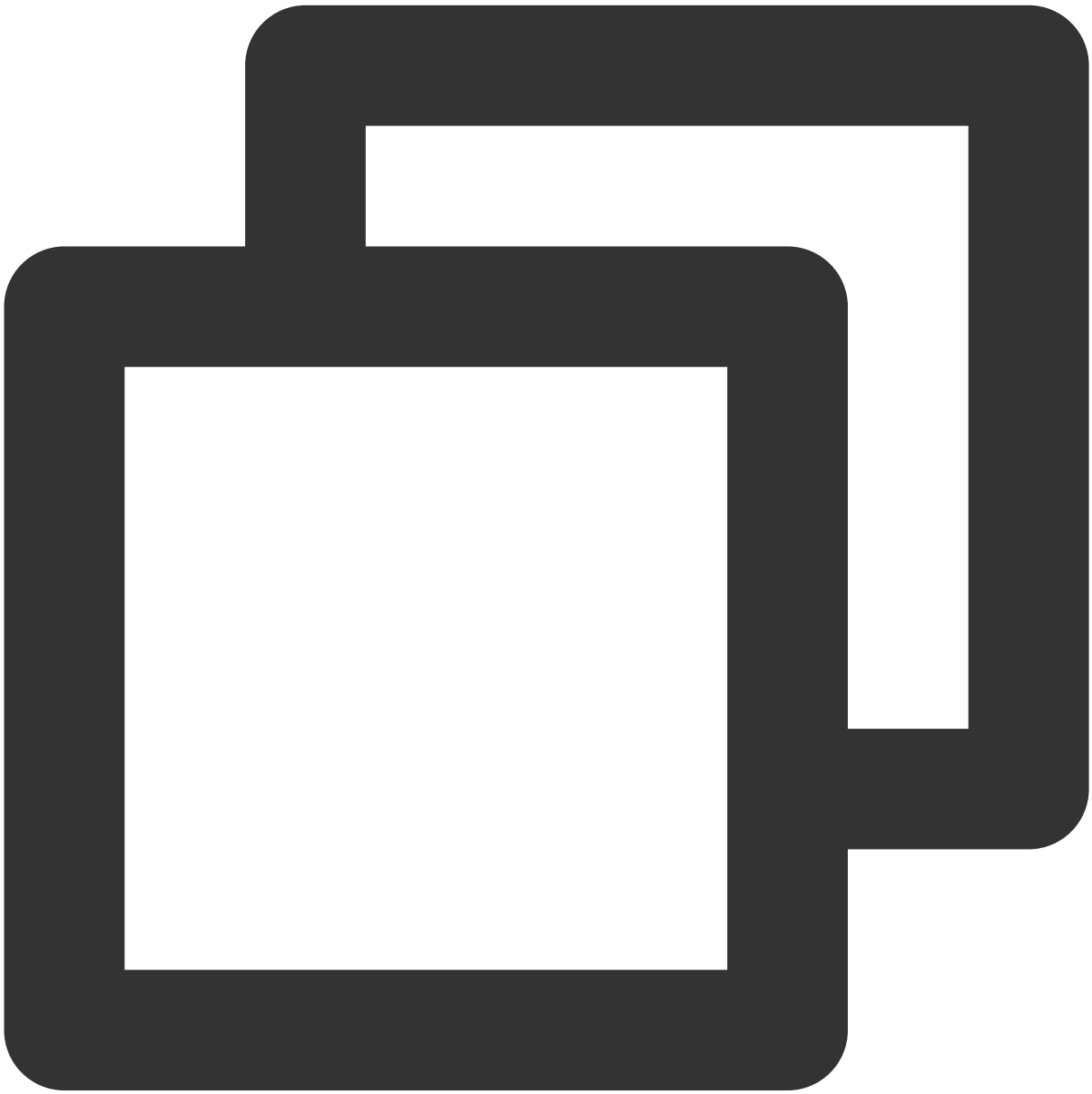


```
// my-behavior.js
module.exports = Behavior({
  behaviors: [],
  properties: {
    myBehaviorProperty: {
      type: String
    }
  },
  data: {
    myBehaviorData: {}
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  attached() {},
  methods: {
    myBehaviorMethod() {}
  }
})
```

组件引用时，在 `behaviors` 定义段中将它们逐个列出即可。



```
// my-component.js
var myBehavior = require('my-behavior')
Component({
```

```
  behaviors: [myBehavior],
  properties: {
    myProperty: {
      type: String
    }
  },
  data: {
    myData: 'my-component-data'
  },
  created: function () {
    console.log('[my-component] created')
  },
  attached: function () {
    console.log('[my-component] attached')
  },
  ready: function () {
    console.log('[my-component] ready')
  },
  methods: {
    myMethod: function () {
      console.log('[my-component] log by myMethod')
    },
  }
})
```

在上例中，`my-component` 组件定义中加入了 `my-behavior`，

而 `my-behavior` 的结构为：

属性：`myBehaviorProperty`

数据字段：`myBehaviorData`

方法：`myBehaviorMethod`

生命周期函数：`attached`、`created`、`ready`

这将使 my-component 最终结构为：

属性：`myBehaviorProperty`、`myProperty`

数据字段：`myBehaviorData`、`myData`

方法：`myBehaviorMethod`、`myMethod`

生命周期函数：`attached`、`created`、`ready`

当组件触发生命周期时，上例生命周期函数执行顺序为：

1. [my-behavior] created
2. [my-component] created
3. [my-behavior] attached
4. [my-component] attached
5. [my-behavior] ready

版权所有：腾讯云计算（北京）有限责任公司

6. [my-component] ready

详细规则参考 同名字段的覆盖和组合规则。

6.2 同名字段的覆盖和组合规则

组件和它引用的 `behavior` 中可以包含同名的字段，对这些字段的处理方法如下：

如果有同名的属性 (properties) 或方法 (methods)：

若组件本身有这个属性或方法，则组件的属性或方法会覆盖 `behavior` 中的同名属性或方法；

若组件本身无这个属性或方法，则在组件的 `behaviors` 字段中定义靠后的 `behavior` 的属性或方法会覆盖靠前的同名属性或方法；

在 2 的基础上，若存在嵌套引用 `behavior` 的情况，则规则为：引用者 `behavior` 覆盖 被引用的 `behavior` 中的同名属性或方法。

如果有同名的数据字段 (data)：

若同名的数据字段都是对象类型，会进行对象合并；

其余情况会进行数据覆盖，覆盖规则为：引用者 `behavior` > 被引用的 `behavior` 、靠后的 `behavior` > 靠前的 `behavior` 。（优先级高的覆盖优先级低的，最大的为优先级最高）。

生命周期函数和 observers 不会相互覆盖，而是在对应触发时机被逐个调用：

对于不同的生命周期函数之间，遵循组件生命周期函数的执行顺序；

对于同种生命周期函数和同字段 observers，遵循如下规则：

`behavior` 优先于组件执行；

被引用的 `behavior` 优先于 引用者 `behavior` 执行；

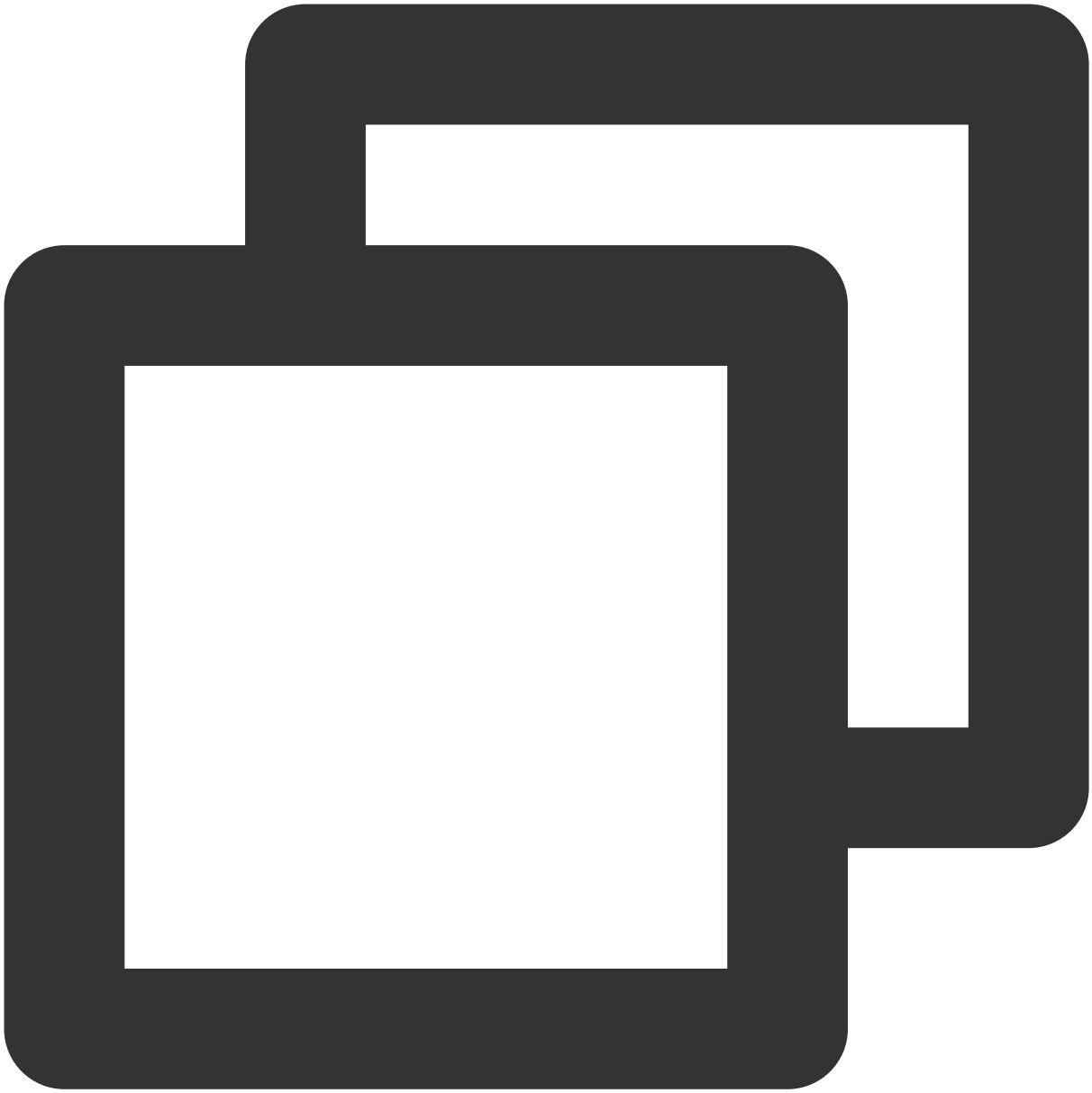
靠前的 `behavior` 优先于 靠后的 `behavior` 执行；

如果同一个 `behavior` 被一个组件多次引用，它定义的生命周期函数和 observers 不会重复执行。

6.3 内置 behaviors

自定义组件可以通过引用内置的 `behavior` 来获得内置组件的一些行为。

版权所有：腾讯云计算（北京）有限责任公司



```
Component({
  behaviors: ['wx://form-field']
})
```

在上例中， `wx://form-field` 代表一个内置 `behavior` ，它使得这个自定义组件有类似于表单控件的行为。

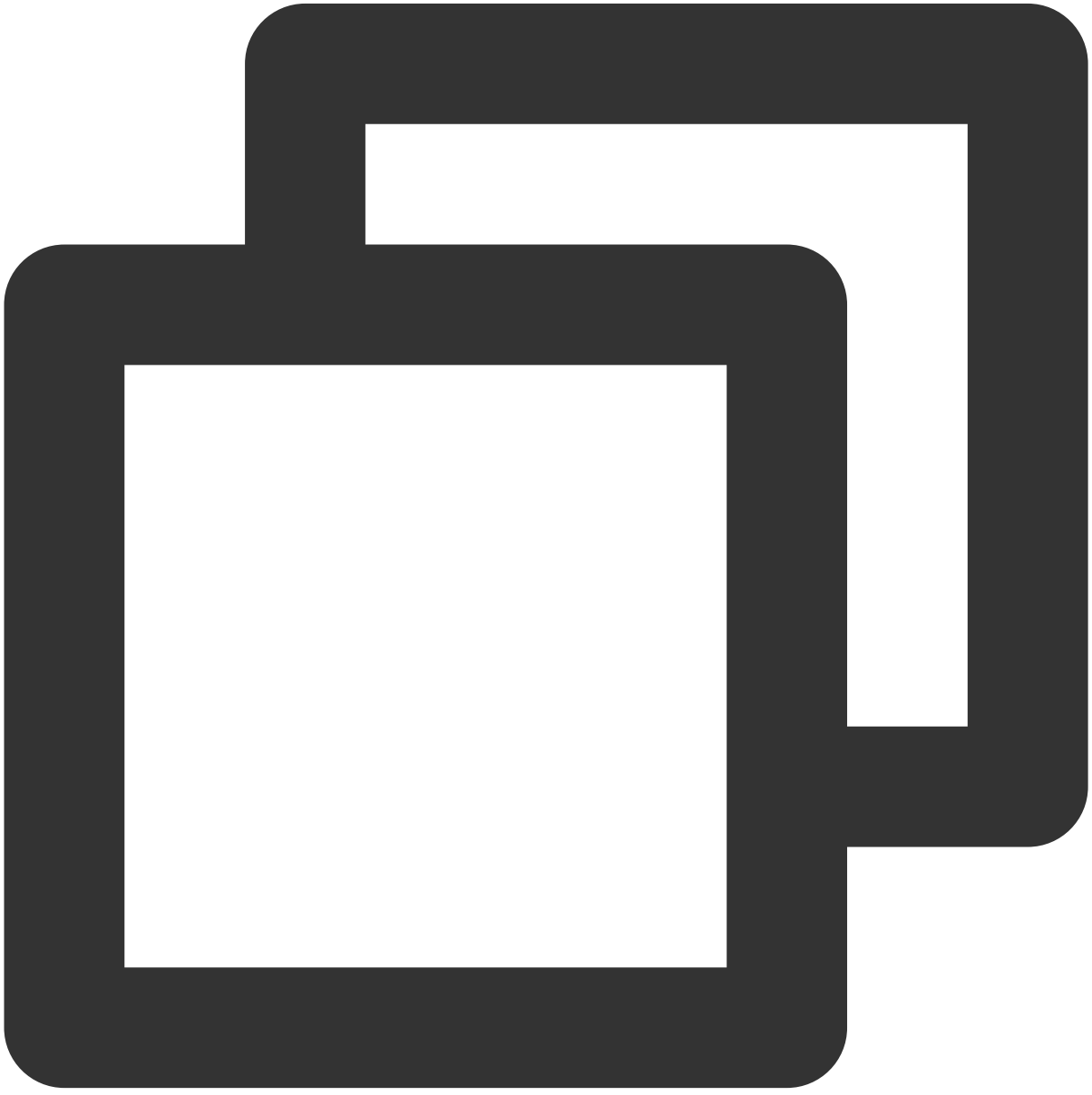
内置 `behavior` 往往会为组件添加一些属性。在没有特殊说明时，组件可以覆盖这些属性来改变它的 `type` 或添加 `observer` 。

6.3.1 wx://form-field

使自定义组件有类似于表单控件的行为。 form 组件可以识别这些自定义组件，并在 submit 事件中返回组件的字段名及其对应字段值。

属性名	类型	描述
name	String	在表单中的字段名
value	任意	在表单中的字段值

代码示例：



```
// custom-form-field.js
Component({
  behaviors: ['wx://form-field'],
  data: {
    value: ''
  },
  methods: {
    onChange: function (e) {
      this.setData({
        value: e.detail.value,
      })
    }
  }
})
```

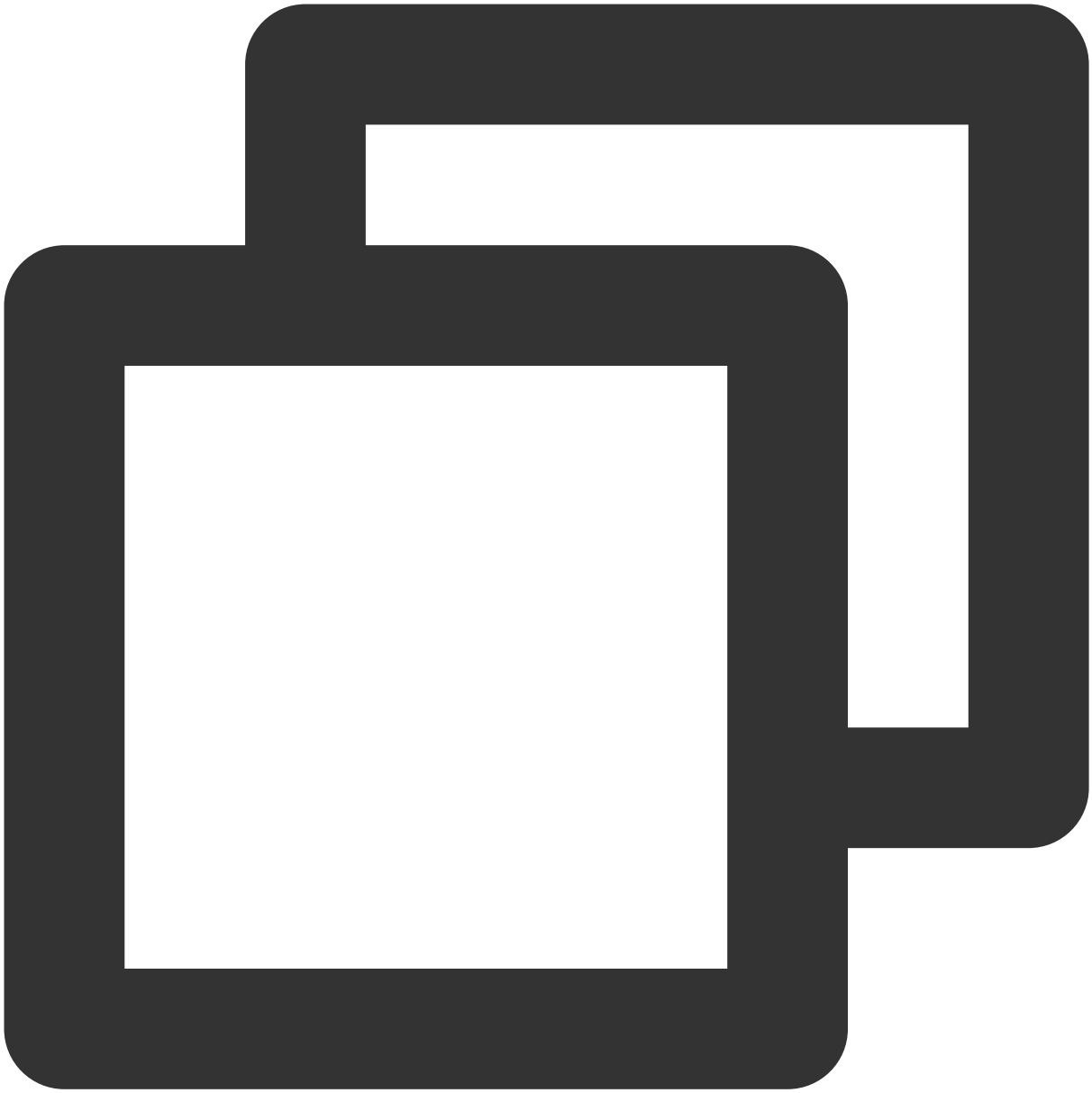
6.3.2 wx://component-export

使自定义组件支持 `export` 定义段。这个定义段可以用于指定组件被 `selectComponent` 调用时的返回值。详细用法以及代码示例可见：selectComponent 参考文档。

7. 组件间关系

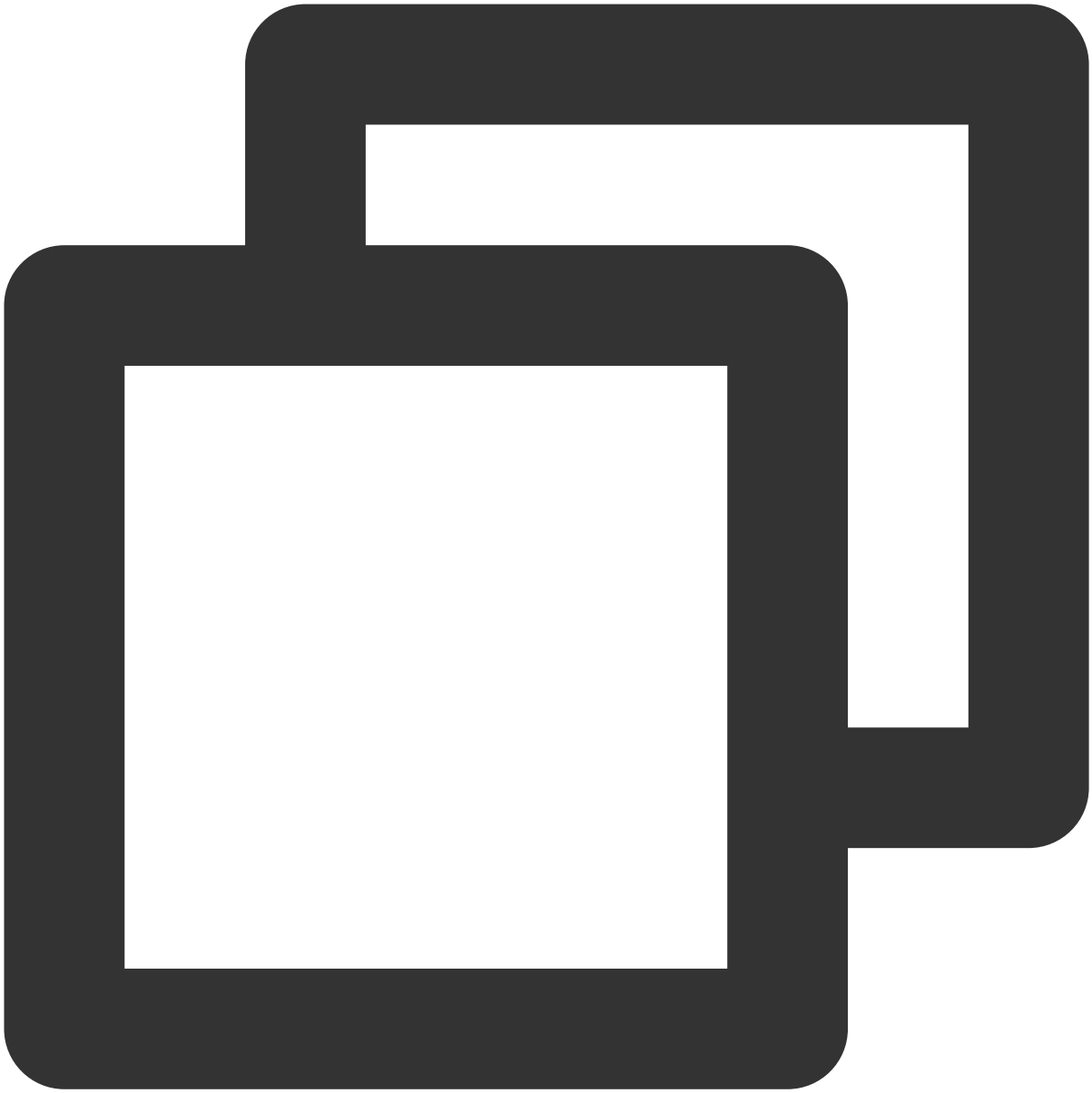
7.1 定义和使用组件间关系

有时需要实现这样的组件：



```
<custom-ul>
  <custom-li> item 1 </custom-li>
  <custom-li> item 2 </custom-li>
</custom-ul>
```

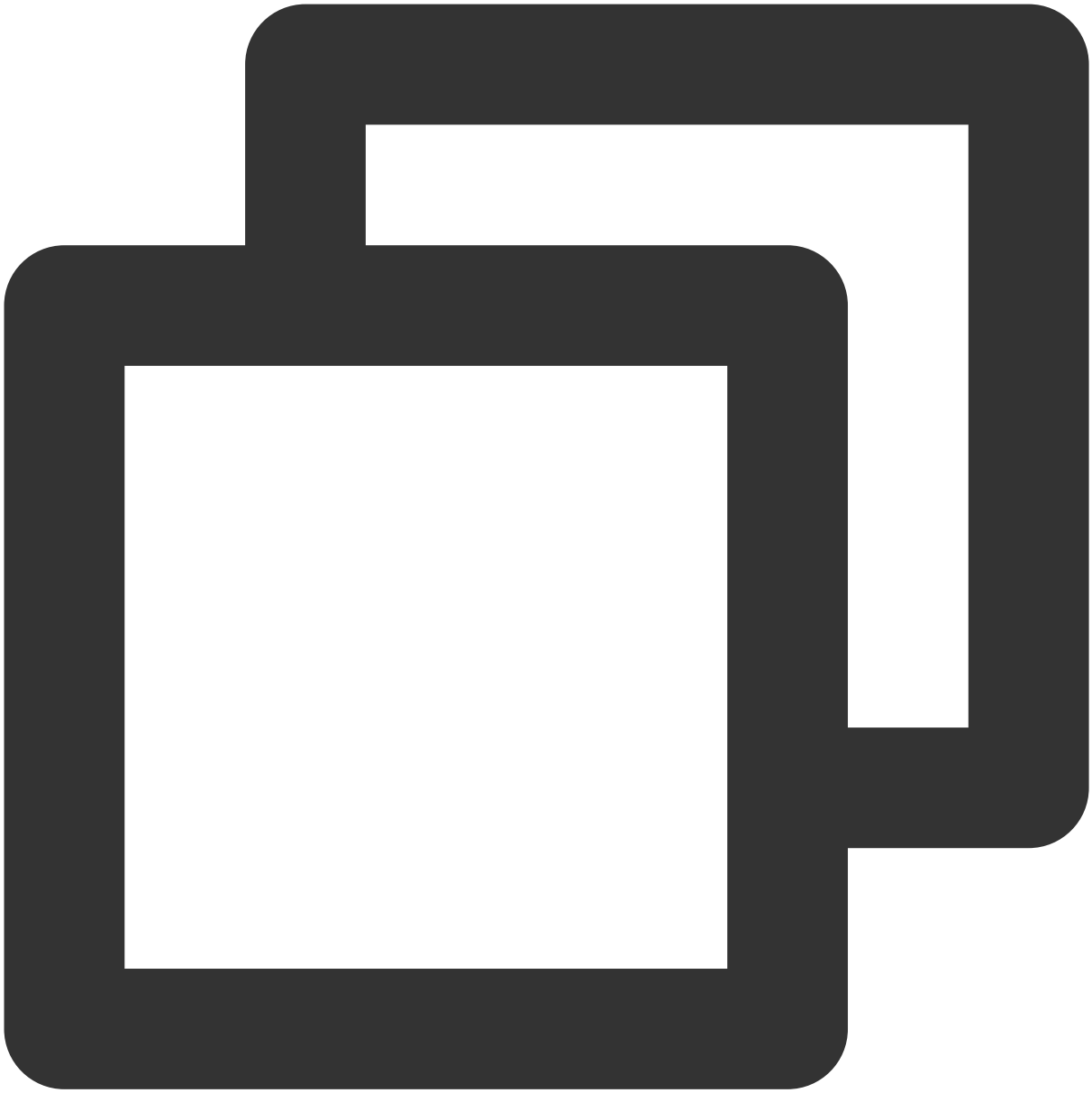
这个例子中，`custom-ul` 和 `custom-li` 都是自定义组件，它们有相互间的关系，相互间的通信往往比较复杂。此时在组件定义时加入 `relations` 定义段，可以解决这样的问题。示例：



```
// path/to/custom-ul.js
Component({
  relations: {
    './custom-li': {
      type: 'child', // 关联的目标节点应为子节点
      linked: function(target) {
        // 每次有custom-li被插入时执行，target是该节点实例对象，触发在该节点attached生命周期之
      },
      linkChanged: function(target) {
        // 每次有custom-li被移动后执行，target是该节点实例对象，触发在该节点moved生命周期之后
      },
```

版权所有：腾讯云计算（北京）有限责任公司

```
      unlinked: function(target) {
        // 每次有custom-li被移除时执行，target是该节点实例对象，触发在该节点detached生命周期
      }
    }
  },
  methods: {
    _getAllLi: function(){
      // 使用getRelationNodes可以获得nodes数组，包含所有已关联的custom-li，且是有序的
      var nodes = this.getRelationNodes('path/to/custom-li')
    }
  },
  ready: function(){
    this._getAllLi()
  }
})
```



```
// path/to/custom-li.js
Component({
  relations: {
    './custom-ul': {
      type: 'parent', // 关联的目标节点应为父节点
      linked: function(target) {
        // 每次被插入到custom-ul时执行，target是custom-ul节点实例对象，触发在attached生命周
      },
      linkChanged: function(target) {
        // 每次被移动后执行，target是custom-ul节点实例对象，触发在moved生命周期之后
      },
```

版权所有：腾讯云计算（北京）有限责任公司

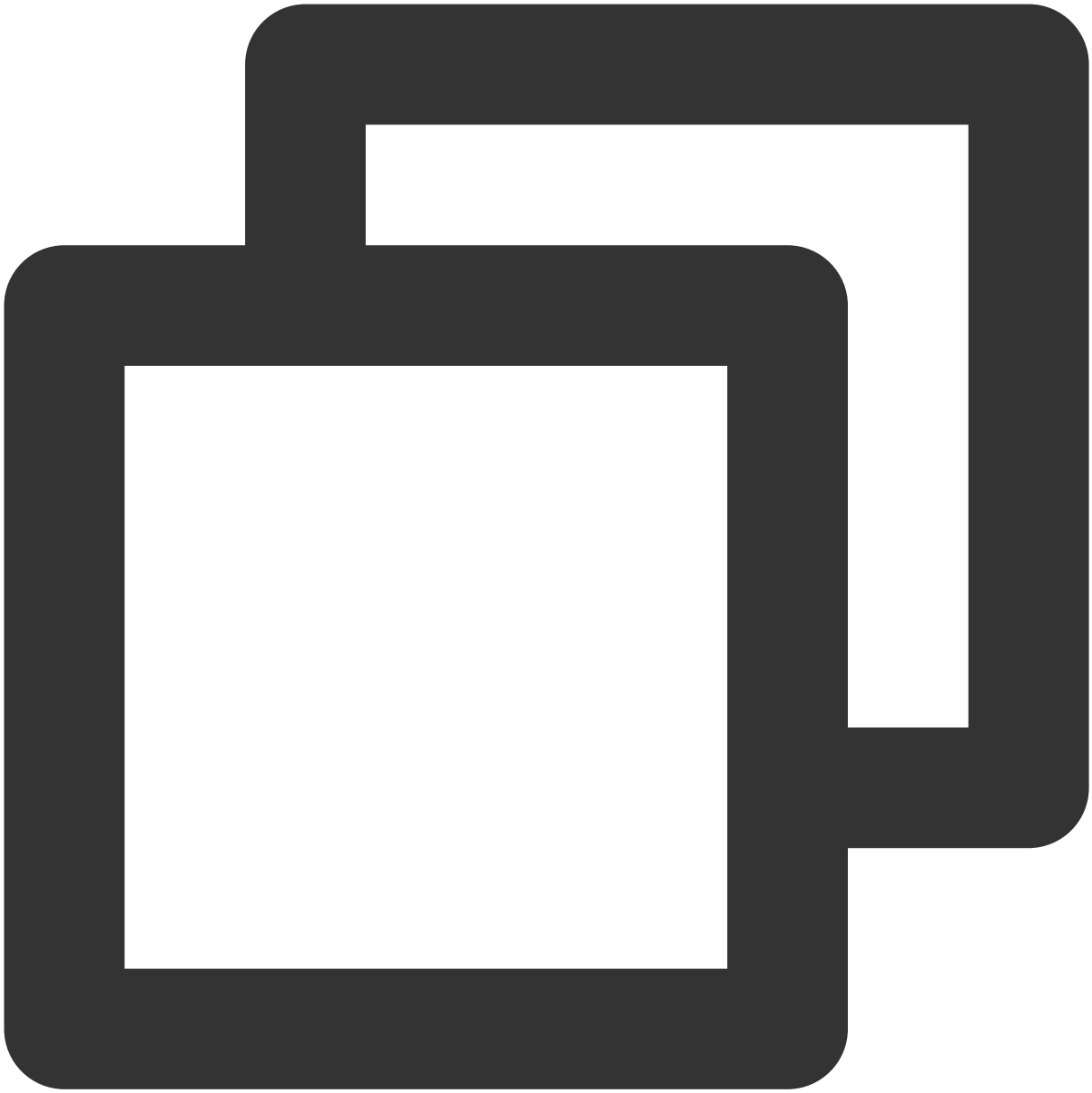
```
        unlinked: function(target) {
          // 每次被移除时执行，target是custom-ul节点实例对象，触发在detached生命周期之后
        }
      }
    }
  })
```

注意：

必须在两个组件定义中都加入 relations 定义，否则不会生效。

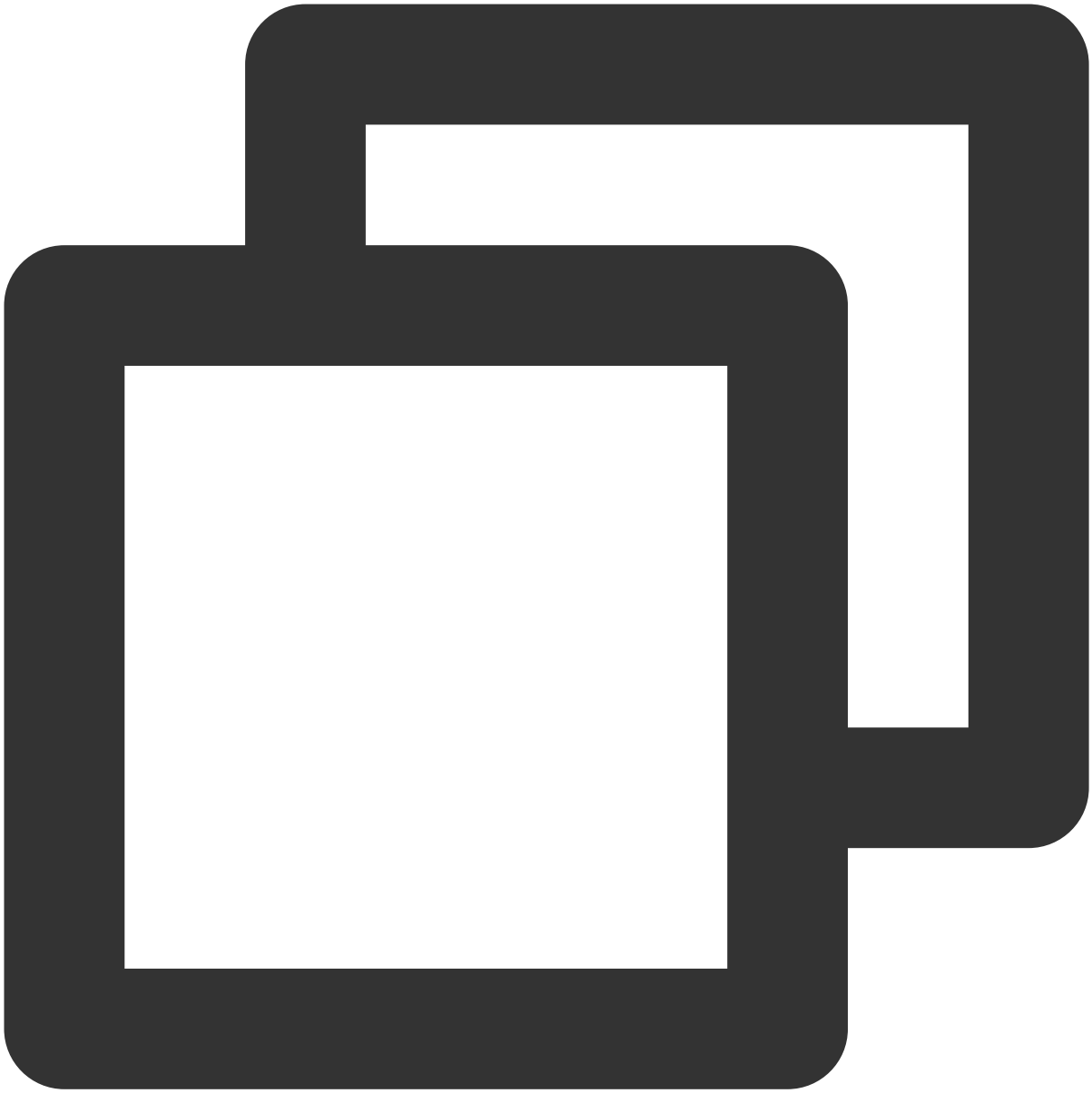
7.2 关联一类组件

有时，需要关联的是一类组件，如：



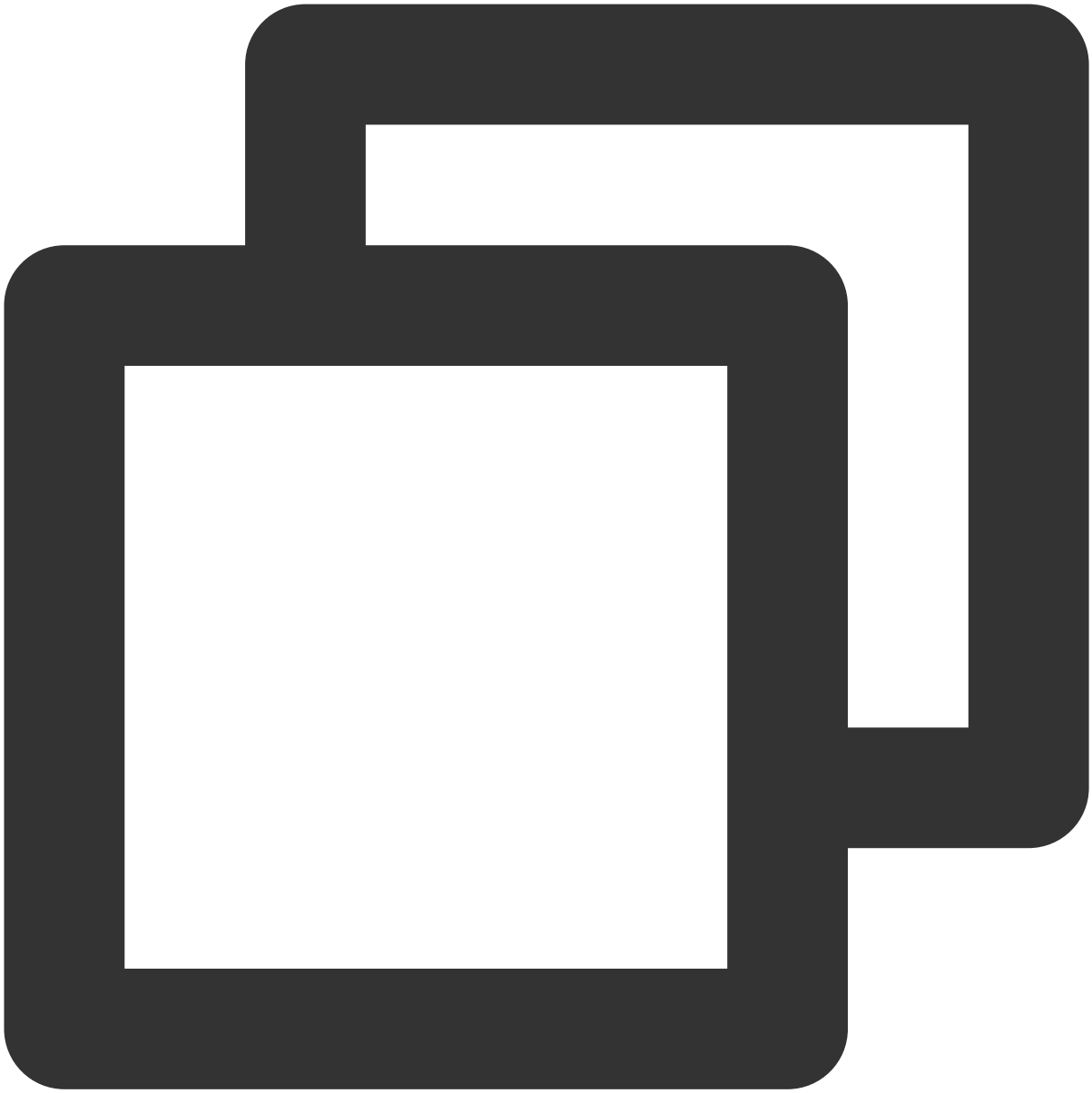
```
<custom-form>
  <view>
    input
    <custom-input></custom-input>
  </view>
  <custom-submit> submit </custom-submit>
</custom-form>
```

`custom-form` 组件想要关联 `custom-input` 和 `custom-submit` 两个组件。此时，如果这两个组件都有同一个 behavior：



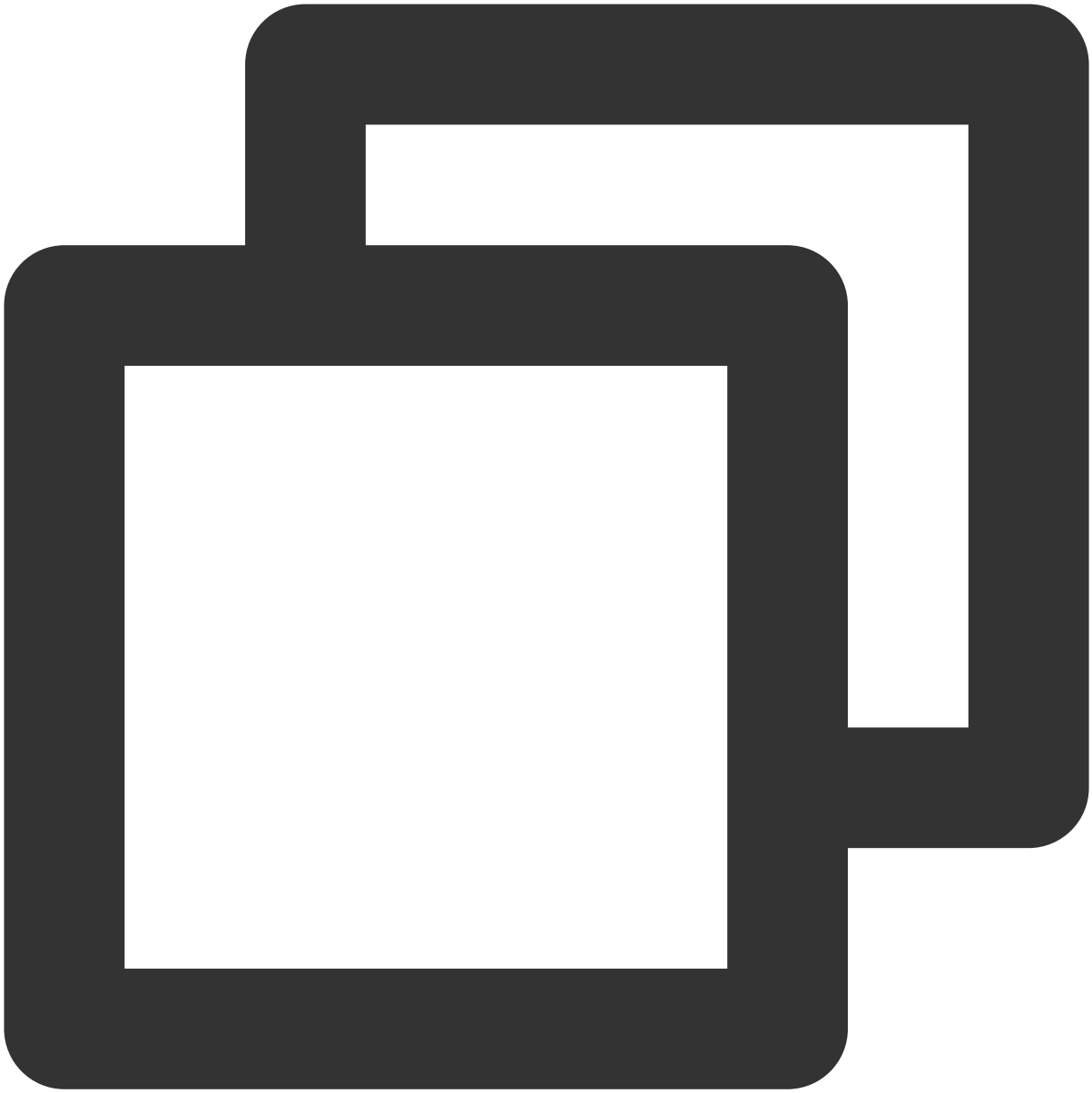
```
// path/to/custom-form-controls.js
module.exports = Behavior({
  // ...
})
```

版权所有：腾讯云计算（北京）有限责任公司



```
// path/to/custom-input.js
var customFormControls = require('./custom-form-controls')
Component({
  behaviors: [customFormControls],
  relations: {
    './custom-form': {
      type: 'ancestor', // 关联的目标节点应为祖先节点
    }
  }
})
```

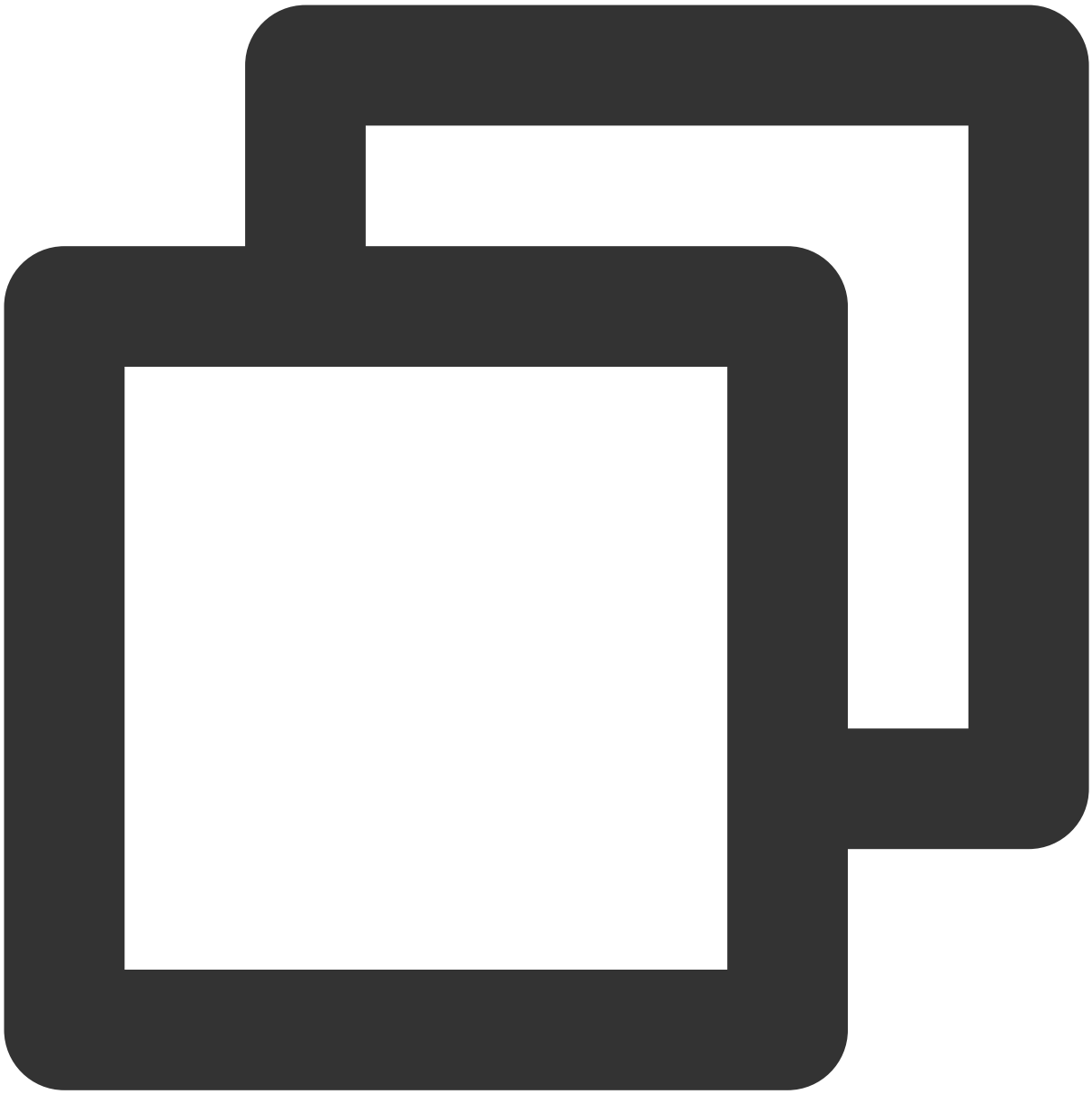
版权所有：腾讯云计算（北京）有限责任公司



```
// path/to/custom-submit.js
var customFormControls = require('./custom-form-controls')
Component({
  behaviors: [customFormControls],
  relations: {
    './custom-form': {
      type: 'ancestor', // 关联的目标节点应为祖先节点
    }
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

则在 `relations` 关系定义中，可使用这个 behavior 来代替组件路径作为关联的目标节点：



```
// path/to/custom-form.js
var customFormControls = require('./custom-form-controls')
Component({
  relations: {
    'customFormControls': {
      type: 'descendant', // 关联的目标节点应为子孙节点
      target: customFormControls
    }
  }
```

```
})
```

7.2 relations 定义段

`relations` 定义段包含目标组件路径及其对应选项，可包含的选项见下表。

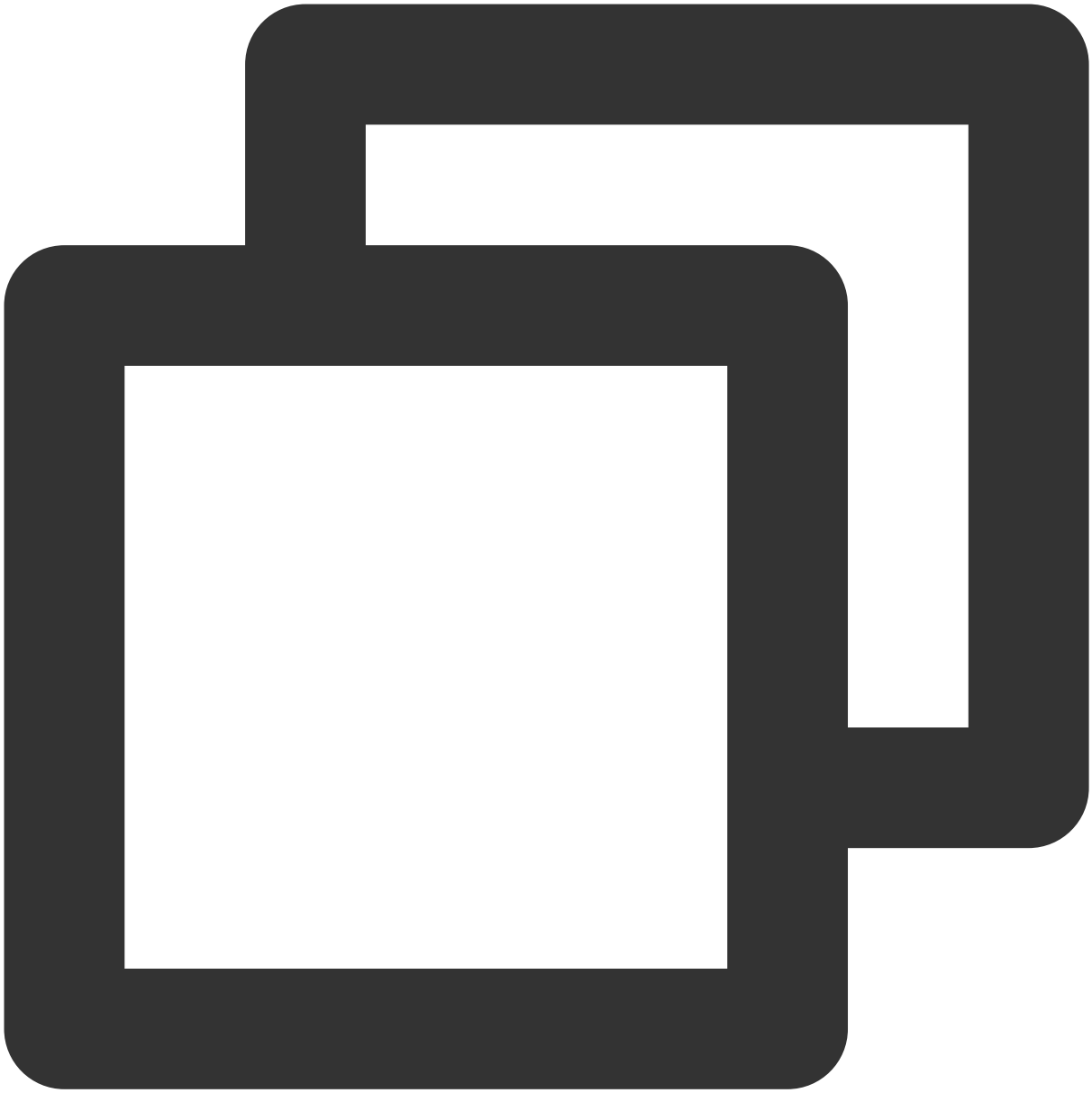
选项	类型	是否必填	描述
type	String	是	目标组件的相对关系，可选的值为 `parent` 、 `child` 、 `ancestor` 、 `descendant`
linked	Function	否	关系生命周期函数，当关系被建立在页面节点树中时触发，触发时机在组件 attached 生命周期之后
linkChanged	Function	否	关系生命周期函数，当关系在页面节点树中发生改变时触发，触发时机在组件 moved 生命周期之后
unlinked	Function	否	关系生命周期函数，当关系脱离页面节点树时触发，触发时机在组件 detached 生命周期之后
target	String	否	如果这一项被设置，则它表示关联的目标节点所应具有的behavior，所有拥有这一 behavior 的组件节点都会被关联

8. 数据监听器

数据监听器可以用于监听和响应任何属性和数据字段的变化。

8.1 使用数据监听器

有时，在一些数据字段被 setData 设置时，需要执行一些操作。
例如， `this.data.sum` 永远是 `this.data.numberA` 与 `this.data.numberB` 的和。此时，可以使用数据监听器进行如下实现。

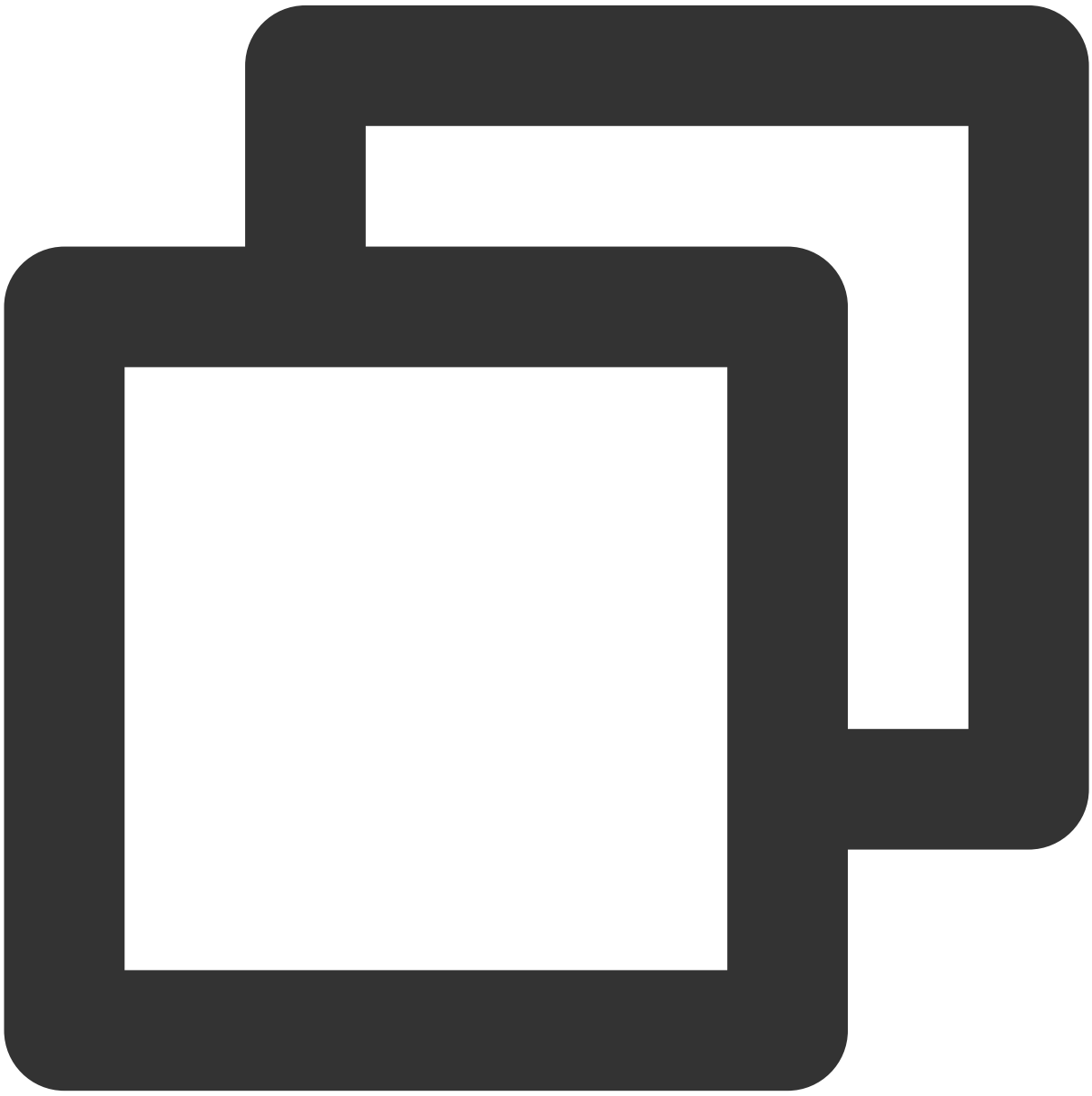


```
Component({
  attached: function() {
    this.setData({
      numberA: 1,
      numberB: 2,
    })
  },
  observers: {
    'numberA, numberB': function(numberA, numberB) {
      // 在 numberA 或者 numberB 被设置时，执行这个函数
      this.setData({
```

```
          sum: numberA + numberB
        })
      }
    }
})
```

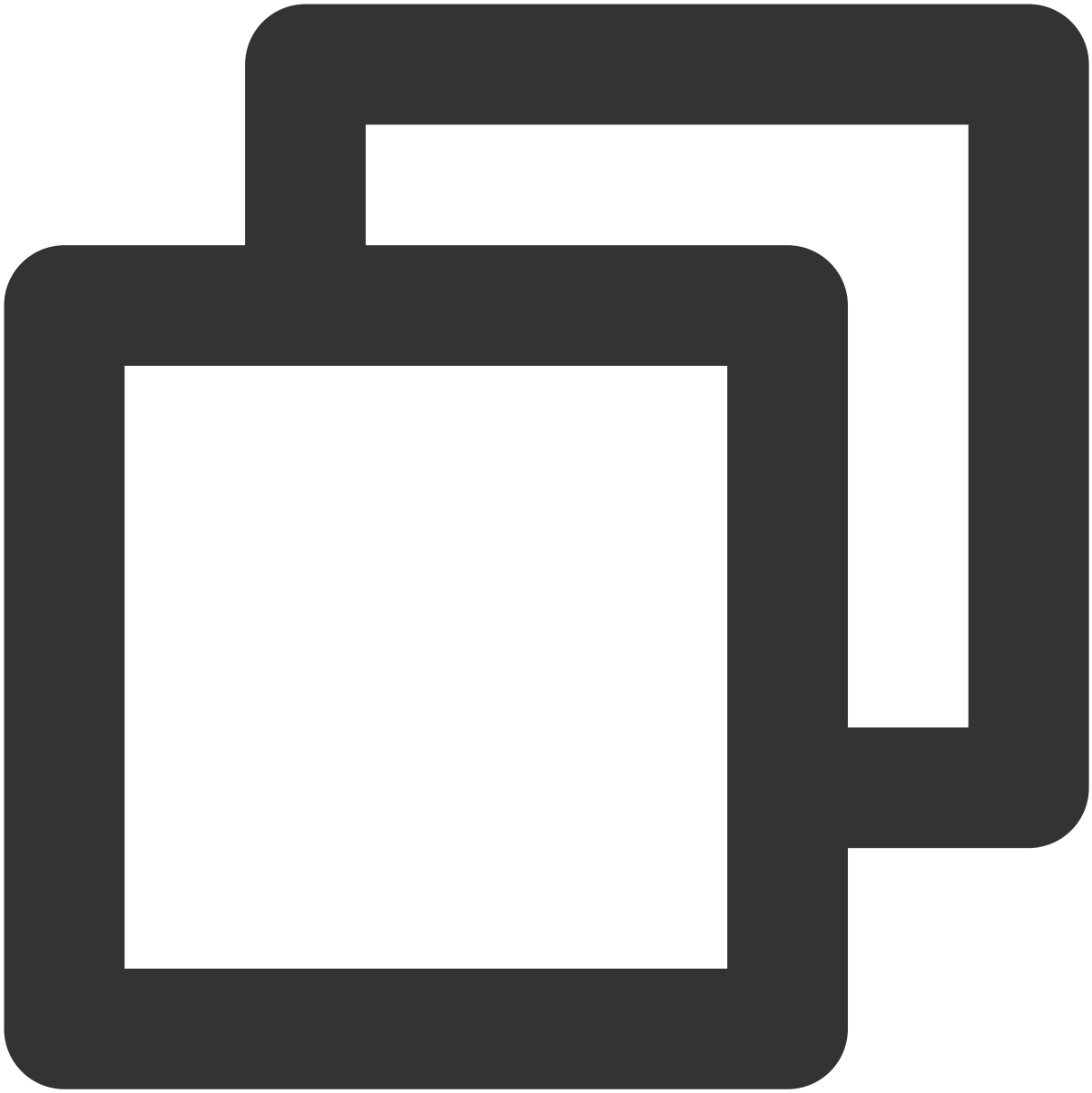
8.2 监听字段语法

数据监听器支持监听属性或内部数据的变化，可以同时监听多个。一次 setData 最多触发每个监听器一次。同时，监听器可以监听子数据字段，如下例所示。



```
Component({
  observers: {
    'some.subfield': function(subfield) {
      // 使用 setData 设置 this.data.some.subfield 时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
      subfield === this.data.some.subfield
    },
    'arr[12]': function(arr12) {
      // 使用 setData 设置 this.data.arr[12] 时触发
      // （除此以外，使用 setData 设置 this.data.arr 也会触发）
      arr12 === this.data.arr[12]
    },
  }
})
```

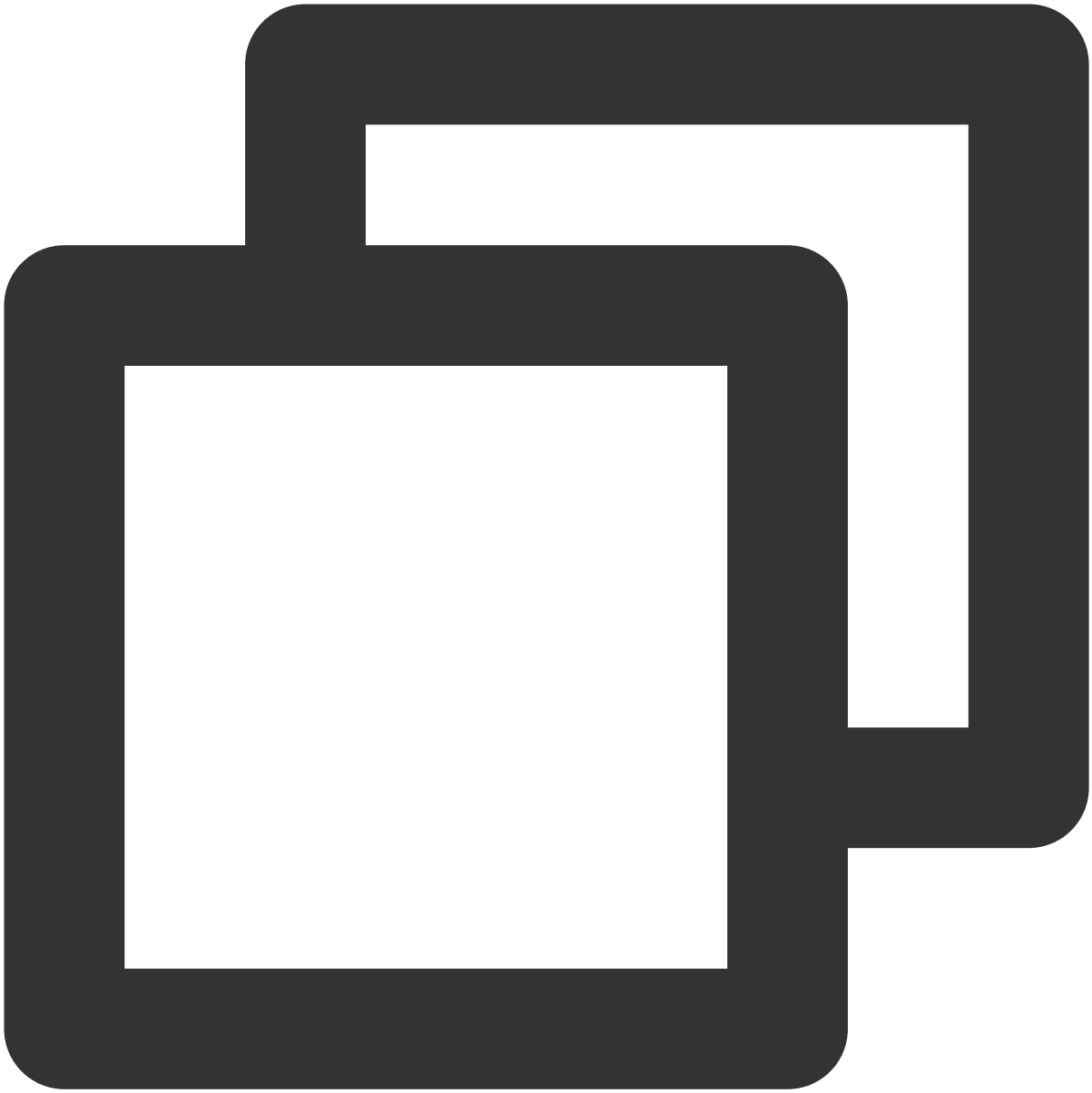
如果需要监听所有子数据字段的变化，可以使用通配符 `**` 。



```
Component({
  observers: {
    'some.field.**': function(field) {
      // 使用 setData 设置 this.data.some.field 本身或其下任何子数据字段时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
      field === this.data.some.field
    },
  },
  attached: function() {
    // 这样会触发上面的 observer
    this.setData({
```

```
        'some.field': { /* ... */ }
      })
      // 这样也会触发上面的 observer
      this.setData({
        'some.field.xxx': { /* ... */ }
      })
      // 这样还是会触发上面的 observer
      this.setData({
        'some': { /* ... */ }
      })
    }
  })
```

特别地，仅使用通配符 `**` 可以监听全部 setData。



```
Component({
  observers: {
    '**': function() {
      // 每次 setData 都触发
    },
  },
})
```

8.3 注意事项

版权所有：腾讯云计算（北京）有限责任公司

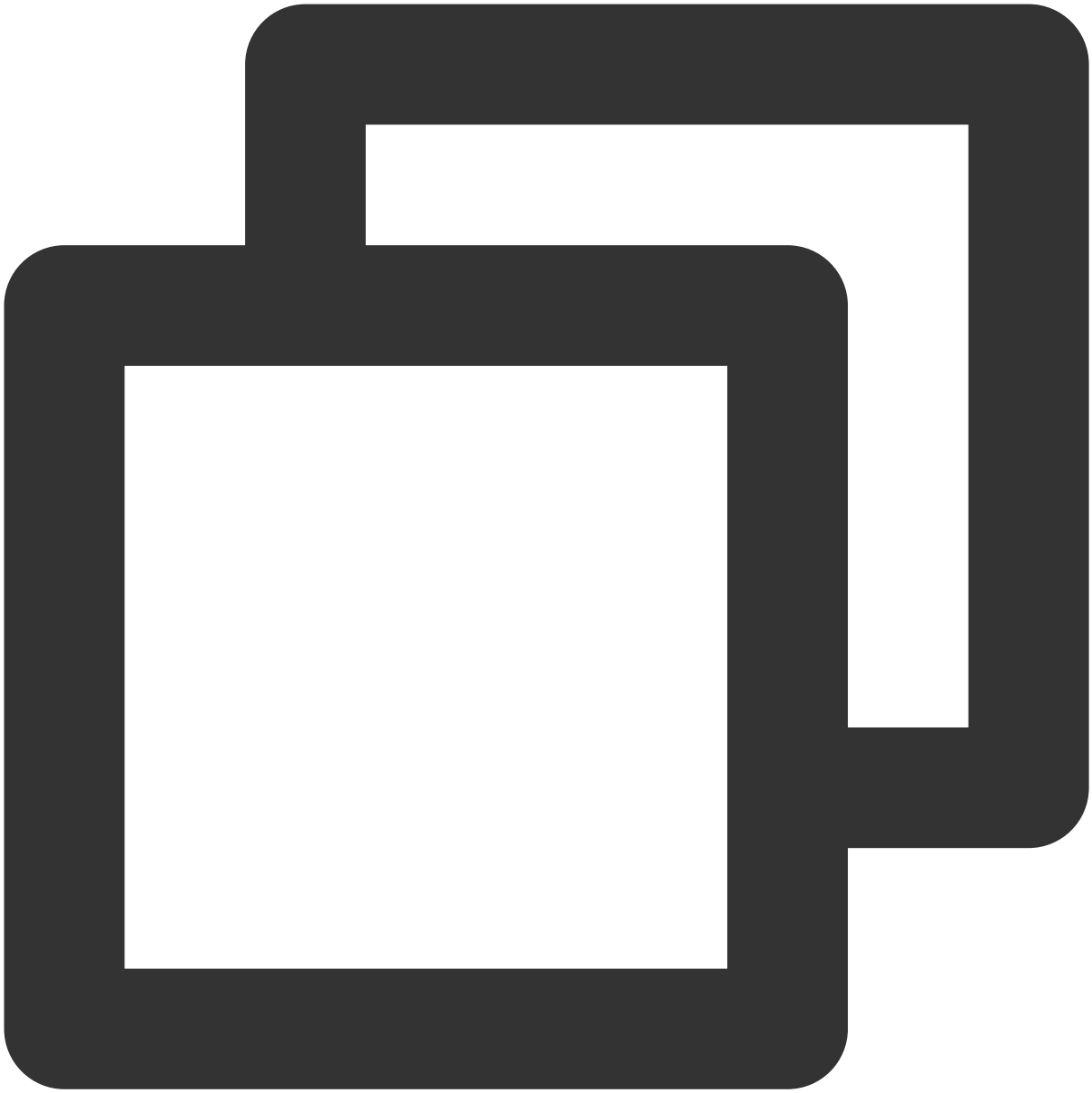
数据监听器监听的是 setData 涉及到的数据字段，即使这些数据字段的值没有发生变化，数据监听器依然会被触发；
如果在数据监听器函数中使用 setData 设置本身监听的数据字段，可能会导致死循环，需要特别留意；
数据监听器和属性的 observer 相比，数据监听器更强大且通常具有更好的性能。

9. 纯数据字段

纯数据字段是一些不用于界面渲染的 `data` 字段，可以用于提升页面更新性能。

9.1 组件数据中的纯数据字段

有些情况下，某些 `data` 中的字段（包括 `setData` 设置的字段）既不会展示在界面上，也不会传递给其他组件，仅仅在当前组件内部使用。
此时，可以指定这样的数据字段为“纯数据字段”，它们将仅仅被记录在 `this.data` 中，而不参与任何界面渲染过程，这样有助于提升页面更新性能。
指定“纯数据字段”的方法是在 `Component` 构造器的 `options` 定义段中指定 `pureDataPattern` 为一个正则表达式，字段名符合这个正则表达式的字段将成为纯数据字段。
代码示例：

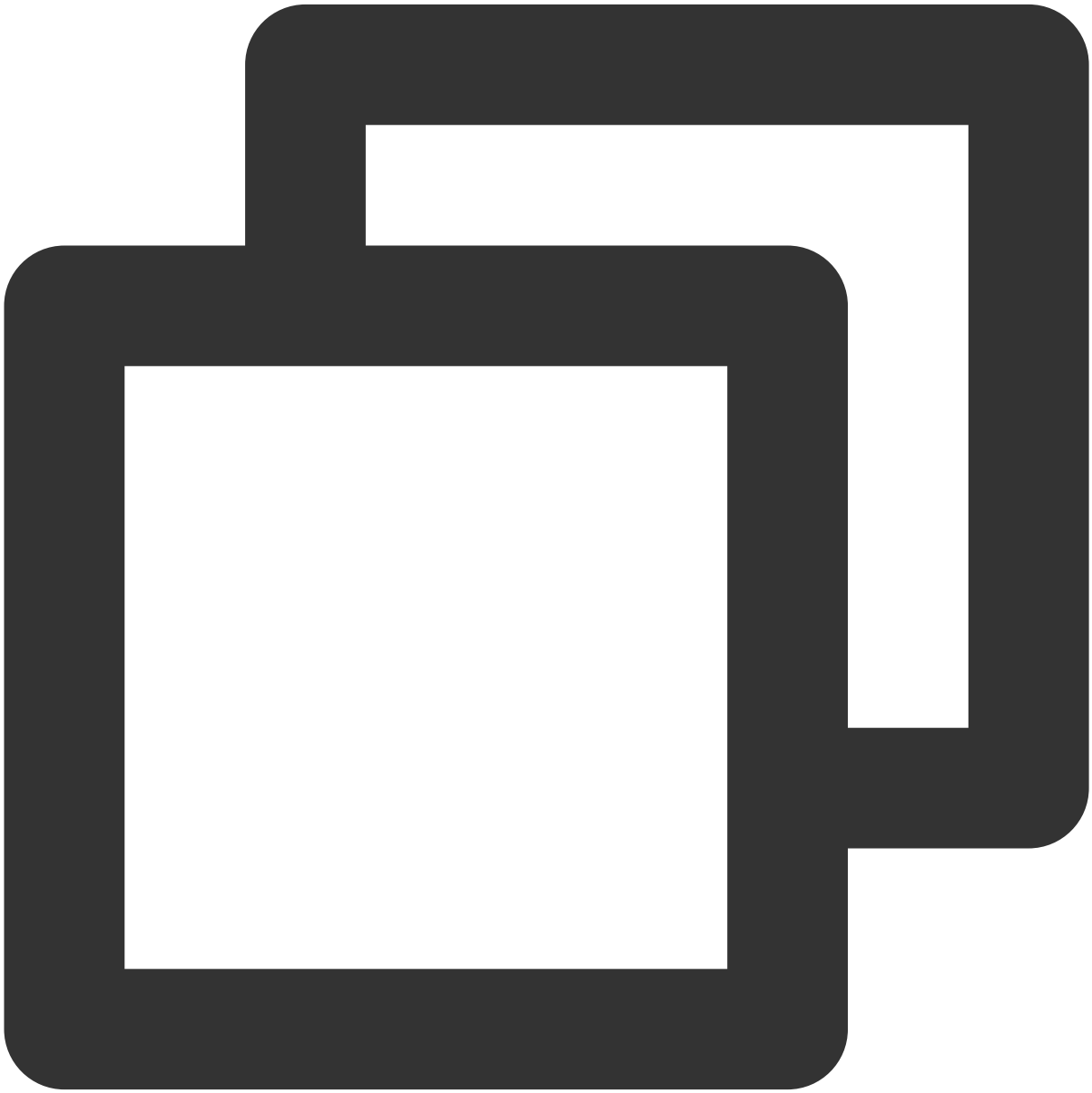


```
Component({
  options: {
    pureDataPattern: /^_/ // 指定所有 _ 开头的数据字段为纯数据字段
  },
  data: {
    a: true, // 普通数据字段
    _b: true, // 纯数据字段
  },
  methods: {
    myMethod() {
      this.data._b // 纯数据字段可以在 this.data 中获取
```

版权所有：腾讯云计算（北京）有限责任公司

```
      this.setData({
        c: true, // 普通数据字段
        _d: true, // 纯数据字段
      })
    }
  }
})
```

上述组件中的纯数据字段不会被应用到 WXML 上：



```
<view wx:if="{{a}}"> 这行会被展示 </view>
<view wx:if="{{_b}}"> 这行不会被展示 </view>
```

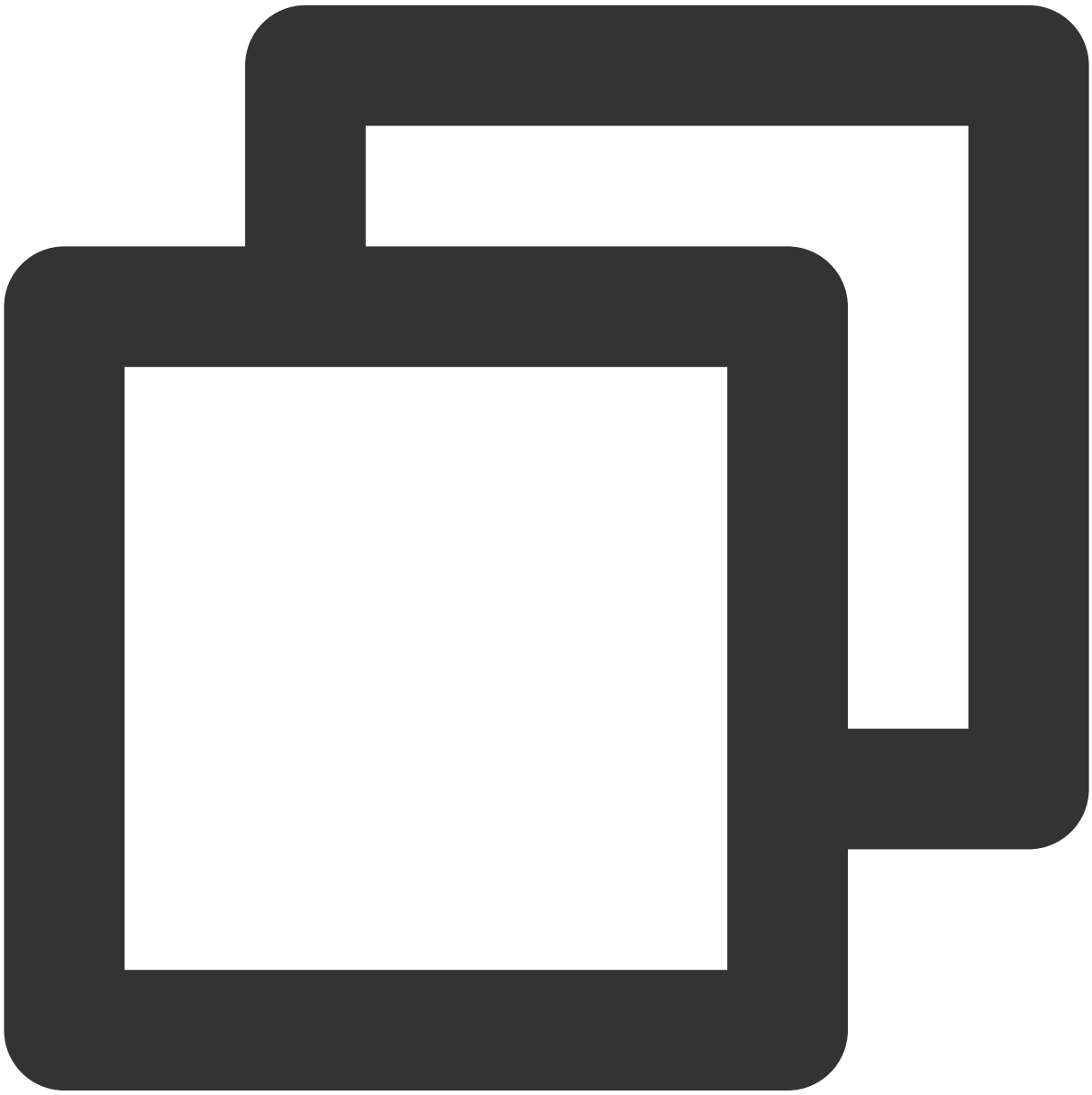
版权所有：腾讯云计算（北京）有限责任公司

9.2 组件属性中的纯数据字段

属性也可以被指定为纯数据字段（遵循 `pureDataPattern` 的正则表达式）。

属性中的纯数据字段可以像普通属性一样接收外部传入的属性值，但不能将它直接用于组件自身的 WXML 中。

代码示例：



```
Component({
  options: {
    pureDataPattern: /^_/
  },
```

```
  properties: {
    a: Boolean,
    _b: {
      type: Boolean,
      observer() {
        // 不要这样做！这个 observer 永远不会被触发
      }
    },
  }
})
```

注意：

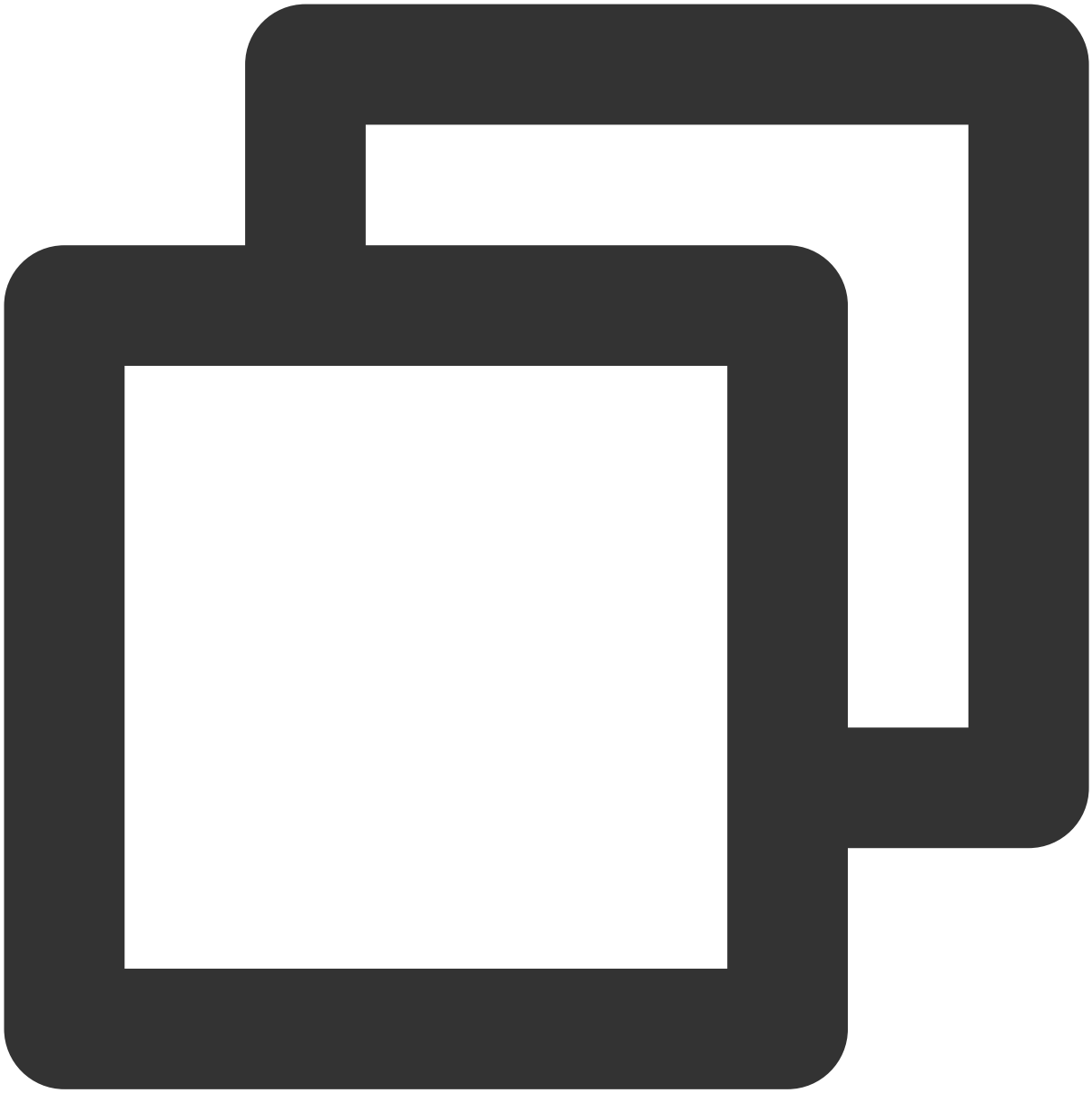
属性中的纯数据字段的属性 **observer** 永远**不会**触发！如果想要监听属性值变化，使用 数据监听器 代替。

9.3 使用数据监听器监听纯数据字段

数据监听器 可以用于监听纯数据字段（与普通数据字段一样）。这样，可以通过监听、响应纯数据字段的变化来改变界面。

下面的示例是一个将 JavaScript 时间戳转换为可读时间的自定义组件。

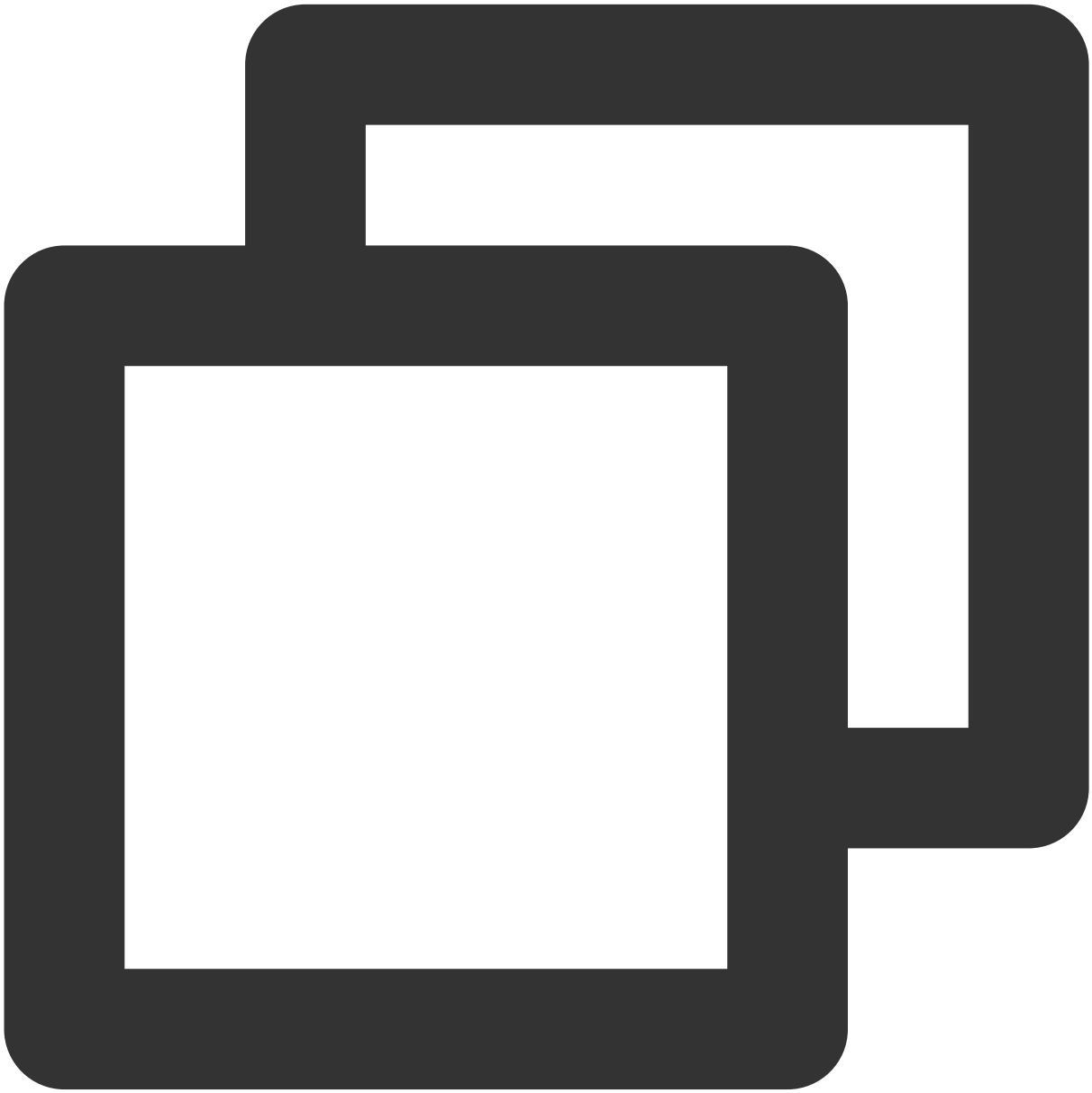
代码示例：



```
Component({
  options: {
    pureDataPattern: /^timestamp$/ // 将 timestamp 属性指定为纯数据字段
  },
  properties: {
    timestamp: Number,
  },
  observers: {
    timestamp: function () {
      // timestamp 被设置时，将它展示为可读时间字符串
      var timeString = new Date(this.data.timestamp).toLocaleString()
```

版权所有：腾讯云计算（北京）有限责任公司

```
        this.setData({
          timeString: timeString
        })
      }
    }
})
```



```
<view>{{timeString}}</view>
```

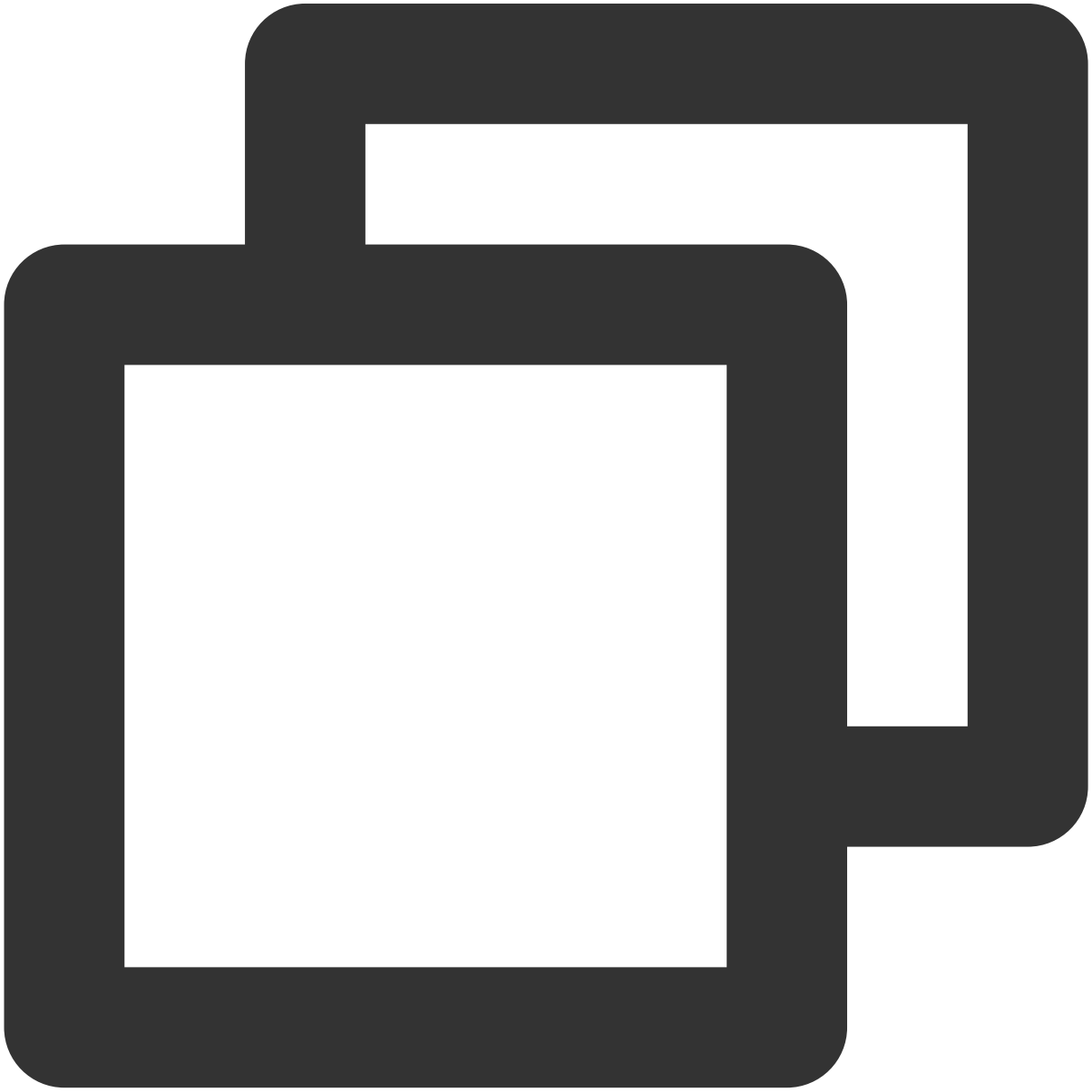
10. 抽象节点

10.1 在组件中使用抽象节点

有时，自定义组件模板中的一些节点，其对应的自定义组件不是由自定义组件本身确定的，而是自定义组件的调用者确定的。这时可以把这个节点声明为“抽象节点”。

例如，我们现在来实现一个“选框组”（selectable-group）组件，它其中可以放置单选框（custom-radio）或者复选框（custom-checkbox）。这个组件的 wxml 可以这样编写：

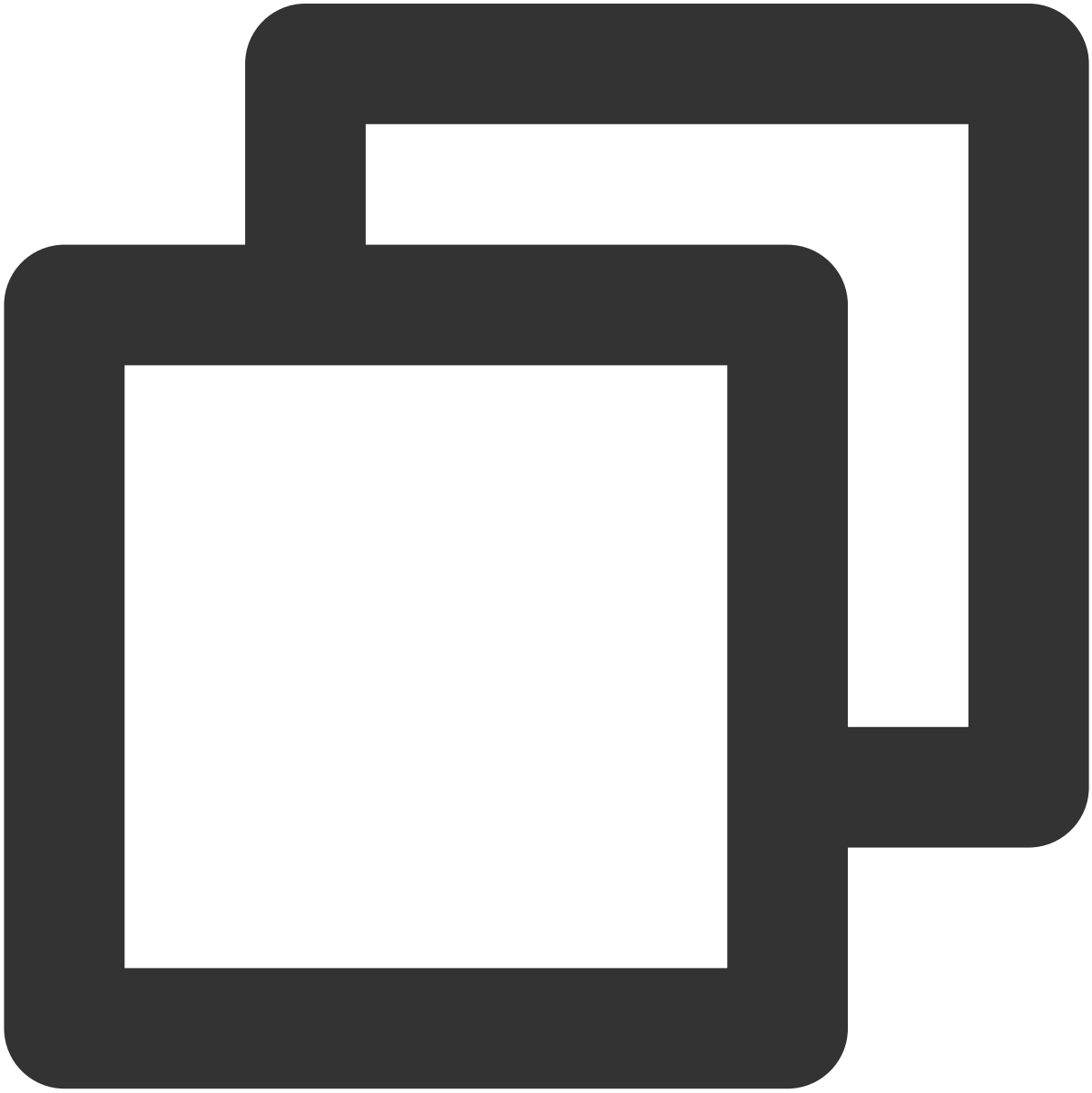
代码示例：



版权所有：腾讯云计算（北京）有限责任公司

```
<!-- selectable-group.wxml -->
<view wx:for="{{labels}}">
  <label>
    <selectable disabled="{{false}}"></selectable>
    {{item}}
  </label>
</view>
```

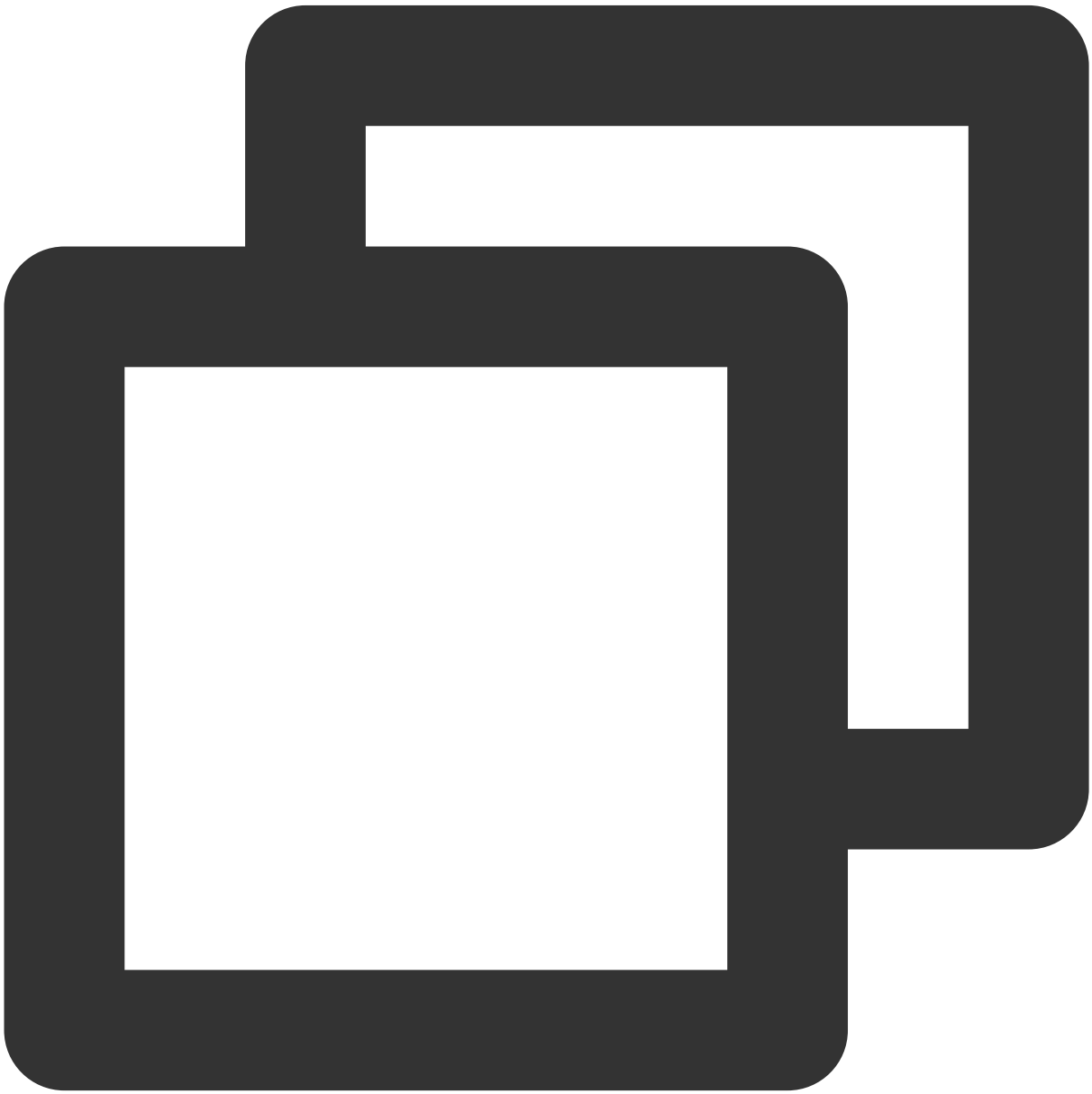
其中，“selectable”不是任何在 json 文件的 usingComponents 字段中声明的组件，而是一个抽象节点。它需要在 componentGenerics 字段中声明：



```
{
  "componentGenerics": {
    "selectable": true
  }
}
```

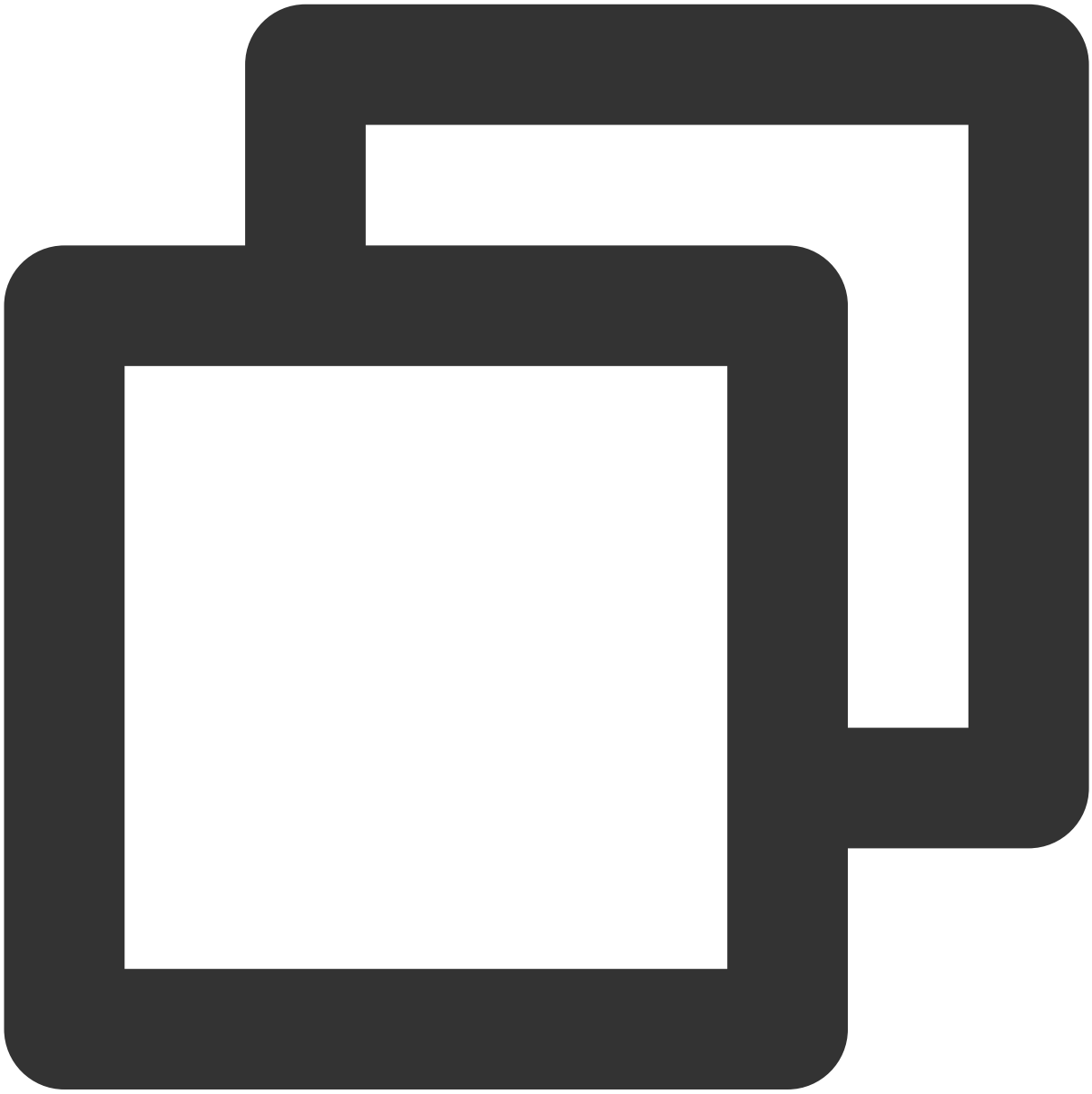
10.2 使用包含抽象节点的组件

在使用 selectable-group 组件时，必须指定 “selectable” 具体是哪个组件：



```
<selectable-group generic:selectable="custom-radio" />
```

这样，在生成这个 selectable-group 组件的实例时，“selectable” 节点会生成 “custom-radio” 组件实例。类似地，如果这样使用：

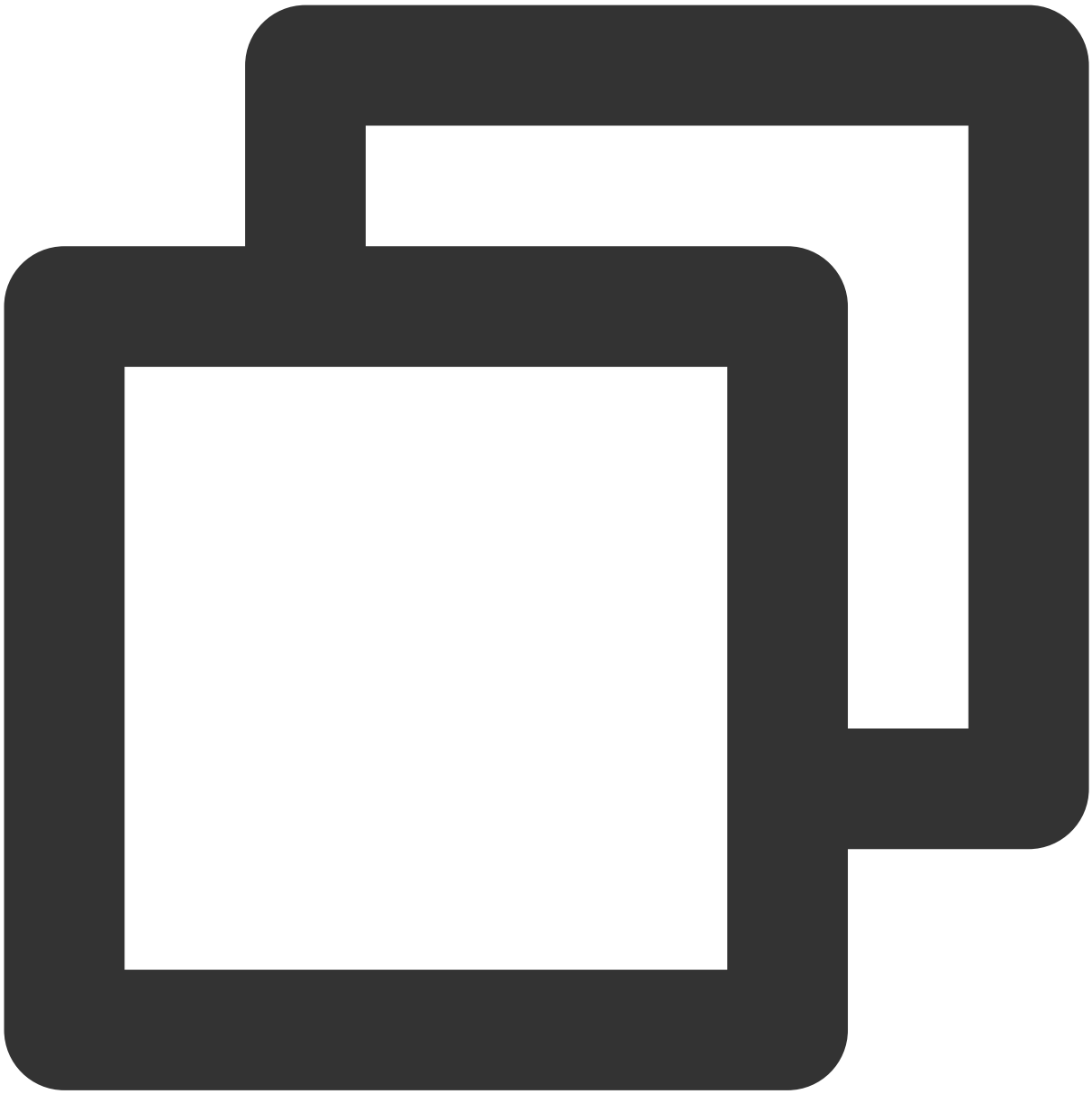


```
<selectable-group generic:selectable="custom-checkbox" />
```

“selectable” 节点则会生成 “custom-checkbox” 组件实例。

注意：

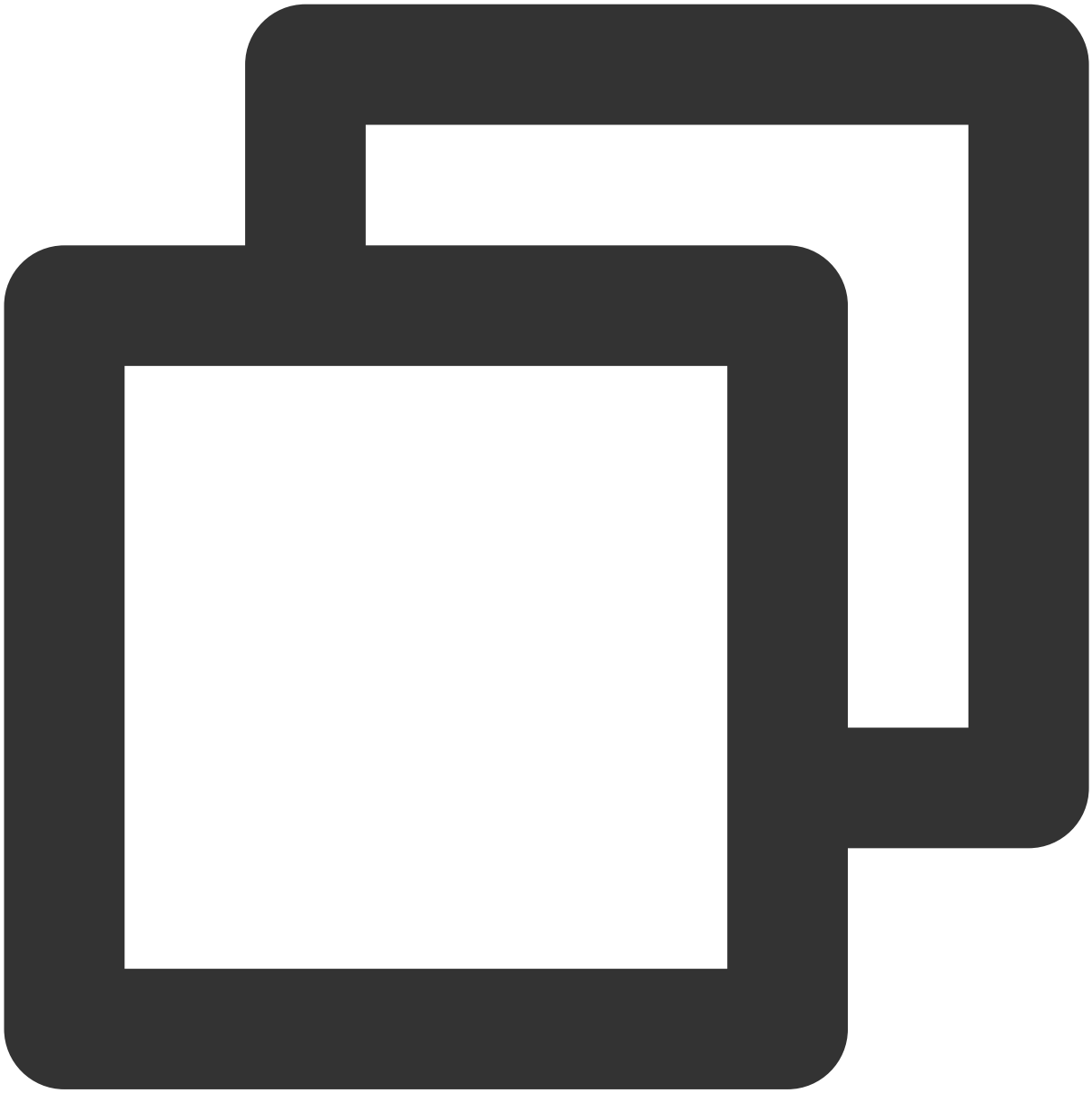
上述的 `custom-radio` 和 `custom-checkbox` 需要包含在这个 wxml 对应 json 文件的 `usingComponents` 定义段中。



```
{
  "usingComponents": {
    "custom-radio": "path/to/custom/radio",
    "custom-checkbox": "path/to/custom/checkbox"
  }
}
```

10.3 抽象节点的默认组件

抽象节点可以指定一个默认组件，当具体组件未被指定时，将创建默认组件的实例。默认组件可以在 `componentGenerics` 字段中指定：



```
{
  "componentGenerics": {
    "selectable": {
      "default": "path/to/default/component"
    }
  }
```

```
}
```

10.4 注意事项

节点的 generic 引用 `generic:xxx="yyy"` 中，值 `yyy` 只能是静态值，不能包含数据绑定。因而抽象节点特性并不适用于动态决定节点名的场景。

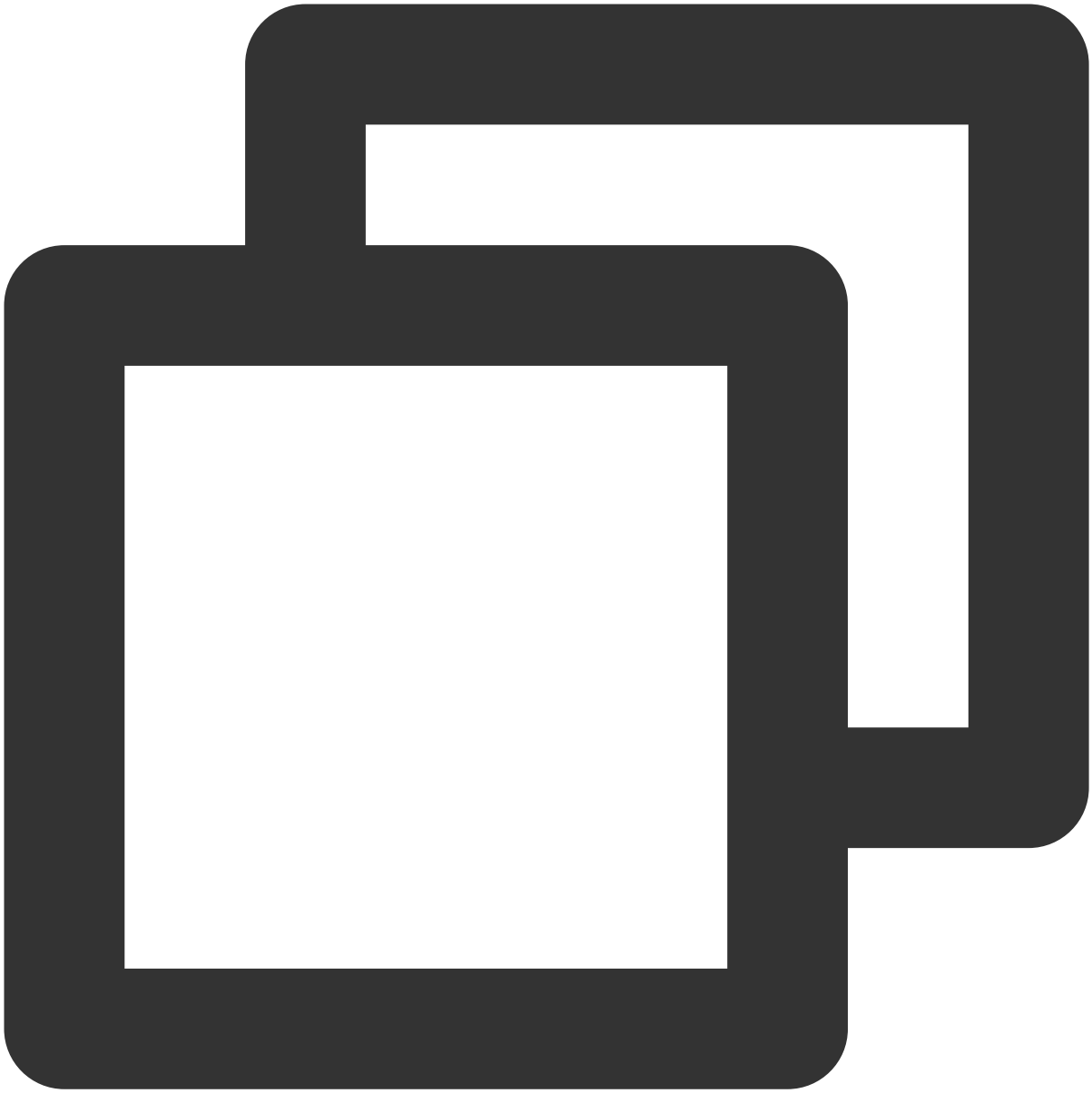
11. 自定义组件扩展

为了更好定制自定义组件的功能，可以使用自定义组件扩展机制。

11.1 自定义组件扩展

为了更好的理解扩展后的效果，先举一个例子：

版权所有：腾讯云计算（北京）有限责任公司



```
// behavior.js
module.exports = Behavior({
  definitionFilter(defFields) {
    defFields.data.from = 'behavior'
  },
})

// component.js
Component({
  data: {
    from: 'component'
```

```
  },
  behaviors: [require('behavior.js')],
  ready() {
    console.log(this.data.from) // 此处会发现输出 behavior 而不是 component
  }
})
```

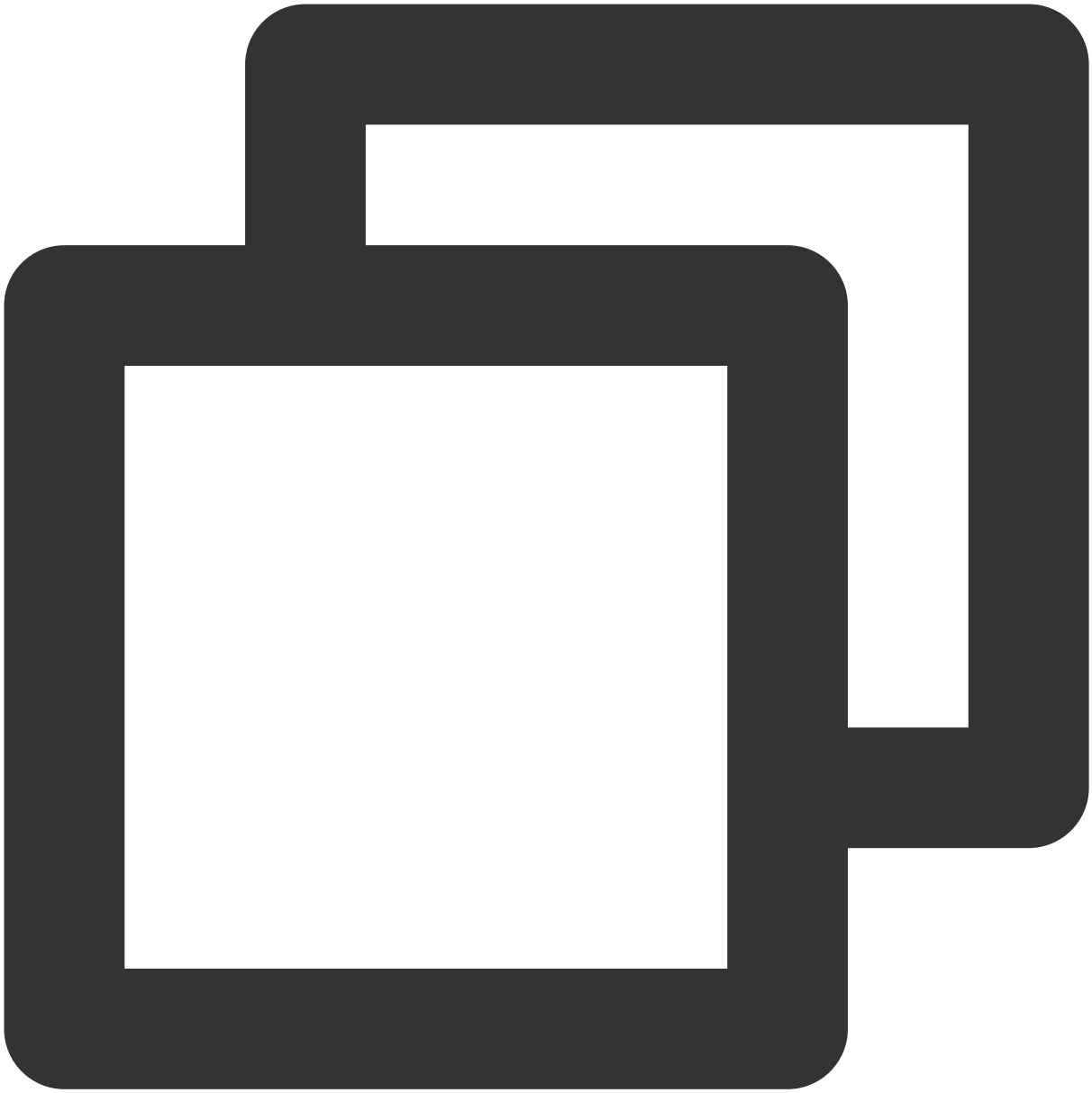
通过例子可以发现，自定义组件的扩展其实就是提供了修改自定义组件定义段的能力，上述例子就是修改了自定义组件中的 `data` 定义段里的内容。

11.2 使用扩展

`Behavior()` 构造器提供了新的定义段 `definitionFilter` ，用于支持自定义组件扩展。

`definitionFilter` 是一个函数，在被调用时会注入两个参数，第一个参数是使用该 behavior 的 component/behavior 的定义对象，第二个参数是该 behavior 所使用的 behavior 的 `definitionFilter` 函数列表。

以下举个例子来说明：



```
// behavior3.js
module.exports = Behavior({
  definitionFilter(defFields, definitionFilterArr) {},
})

// behavior2.js
module.exports = Behavior({
behaviors: [require('behavior3.js')],
definitionFilter(defFields, definitionFilterArr) {
  // definitionFilterArr[0](defFields)
},
```

版权所有：腾讯云计算（北京）有限责任公司

```
})

// behavior1.js
module.exports = Behavior({
behaviors: [require('behavior2.js')],
definitionFilter(defFields, definitionFilterArr) {},
})

// component.js
Component({
behaviors: [require('behavior1.js')],
})
```

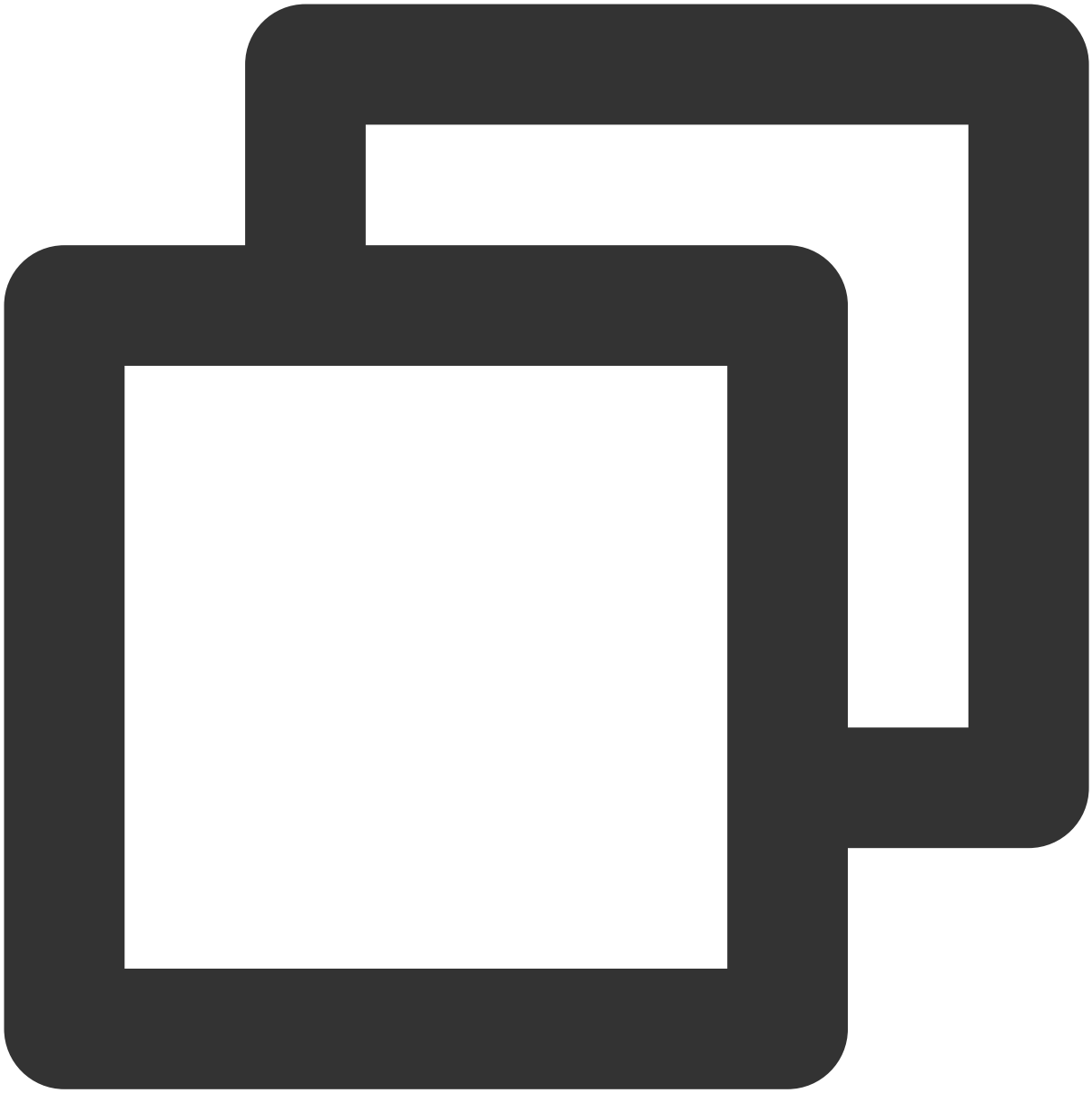
上述代码中声明了1个自定义组件和3个 behavior，每个 behavior 都使用了 definitionFilter 定义段。那么按照声明的顺序会有如下事情发生：

1. 当进行 behavior2 的声明时就会调用 behavior3 的 `definitionFilter` 函数，其中 `defFields` 参数是 behavior2 的定义段， `definitionFilterArr` 参数即为空数组，因为 behavior3 没有使用其他的 behavior ；
2. 当进行 behavior1 的声明时就会调用 behavior2 的 `definitionFilter` 函数，其中 `defFields` 参数是 behavior1 的定义段， `definitionFilterArr` 参数是一个长度为1的数组， `definitionFilterArr[0]` 即为 behavior3 的 `definitionFilter` 函数，因为 behavior2 使用了 behavior3。用户在此处可以自行决定在进行 behavior1 的声明时要不要调用 behavior3 的 `definitionFilter` 函数，如果需要调用，在此处补充代码 `definitionFilterArr[0](defFields)` 即可， `definitionFilterArr` 参数会由基础库补充传入；
3. 同理，在进行 component 的声明时就会调用 behavior1 的 `definitionFilter` 函数。

简单概括， `definitionFilter` 函数可以理解为当 A 使用了 B 时，A 声明就会调用 B 的 `definitionFilter` 函数并传入 A 的定义对象让 B 去过滤。此时如果 B 还使用了 C 和 D ，那么 B 可以自行决定要不要调用 C 和 D 的 `definitionFilter` 函数去过滤 A 的定义对象。

代码示例：

下面利用扩展简单实现自定义组件的计算属性功能:



```
// behavior.js
module.exports = Behavior({
  lifetimes: {
    created() {
      this._originalSetData = this.setData // 原始 setData
      this.setData = this._setData // 封装后的 setData
    }
  },
  definitionFilter(defFields) {
    const computed = defFields.computed || {}
    const computedKeys = Object.keys(computed)
```

版权所有：腾讯云计算（北京）有限责任公司

```
    const computedCache = {}


    // 计算 computed
    const calcComputed = (scope, insertToData) => {
      const needUpdate = {}
      const data = defFields.data = defFields.data || {}

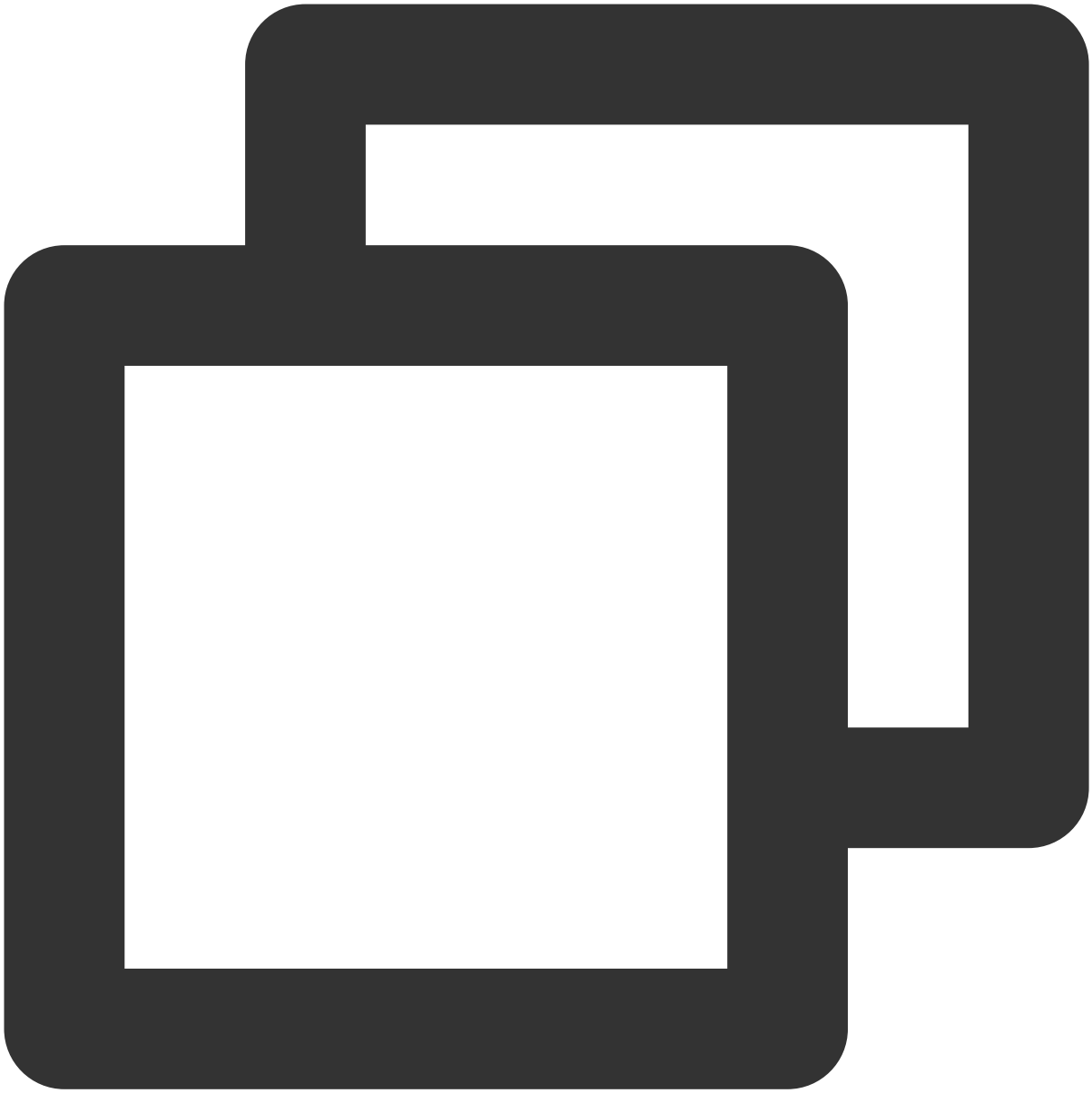

      for (let key of computedKeys) {
        const value = computed[key].call(scope) // 计算新值
        if (computedCache[key] !== value) needUpdate[key] = computedCache[key] = va
        if (insertToData) data[key] = needUpdate[key] // 直接插入到 data 中，初始化时才
      }


      return needUpdate
    }


    // 重写 setData 方法
    defFields.methods = defFields.methods || {}
    defFields.methods._setData = function (data, callback) {
      const originalSetData = this._originalSetData // 原始 setData
      originalSetData.call(this, data, callback) // 做 data 的 setData
      const needUpdate = calcComputed(this) // 计算 computed
      originalSetData.call(this, needUpdate) // 做 computed 的 setData
    }


    // 初始化 computed
    calcComputed(defFields, true) // 计算 computed
  }
})
```

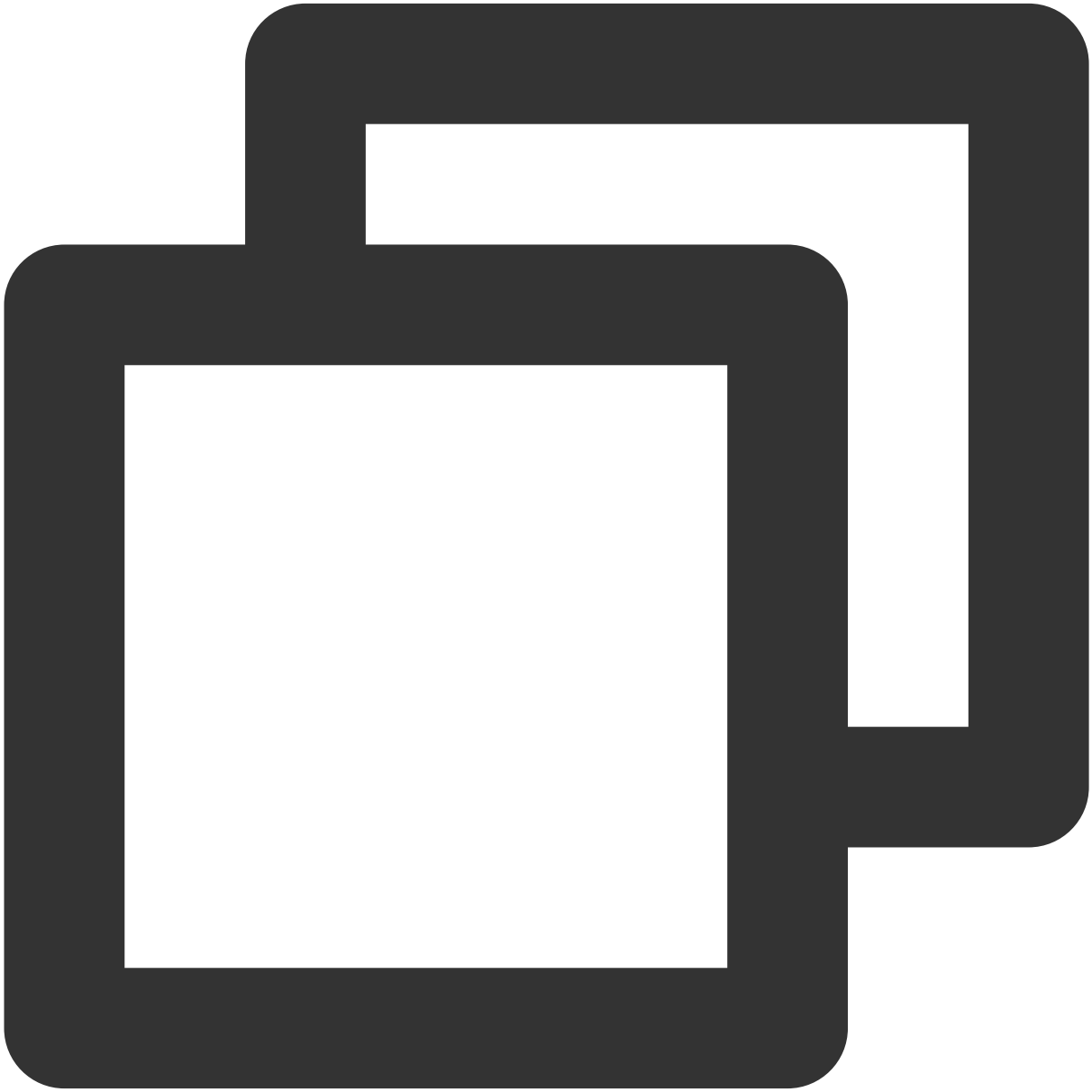
在组件中使用：



```
const beh = require('./behavior.js')
Component({
  behaviors: [beh],
  data: {
    a: 0,
  },
  computed: {
    b() {
      return this.data.a + 100
    },
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  methods: {
    onTap() {
      this.setData({
        a: ++this.data.a,
      })
    }
  }
})
```



```
<view>data: {{a}}</view>
<view>computed: {{b}}</view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
<button bindtap="onTap">click</button>
```

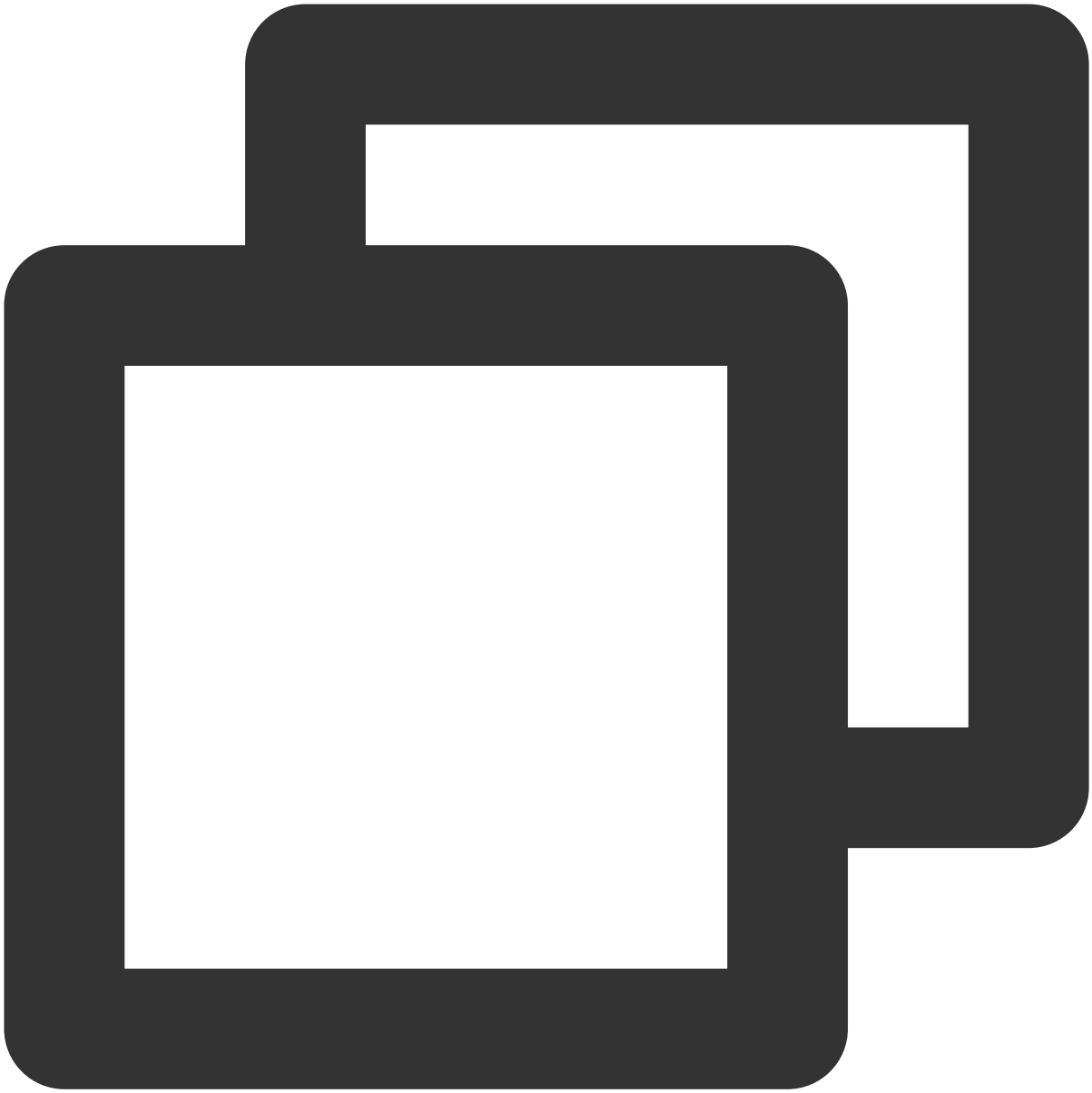
实现原理很简单，对已有的 setData 进行二次封装，在每次 setData 的时候计算出 computed 里各字段的值，然后设到 data 中，以达到计算属性的效果。

12. 占位组件

在使用如 分包异步化 特性时，自定义组件所引用的其他自定义组件，在刚开始进行渲染时可能处于不可用的状态。此时，为了使渲染过程不被阻塞，不可用的自定义组件需要一个「占位组件」（Component placeholder）。基础库会用占位组件替代不可用组件进行渲染，在该组件可用后再将占位组件替换回该组件。
一个自定义组件的占位组件可以是另一个自定义组件、或一个内置组件。

12.1 配置

页面或自定义组件对应的 JSON 配置中的 `componentPlaceholder` 字段用于指定占位组件，如：



```
{
  "usingComponents": {
    "comp-a": "../comp/compA",
    "comp-b": "../comp/compB",
    "comp-c": "../comp/compC"
  },
  "componentPlaceholder": {
    "comp-a": "view",
    "comp-b": "comp-c"
  }
}
```

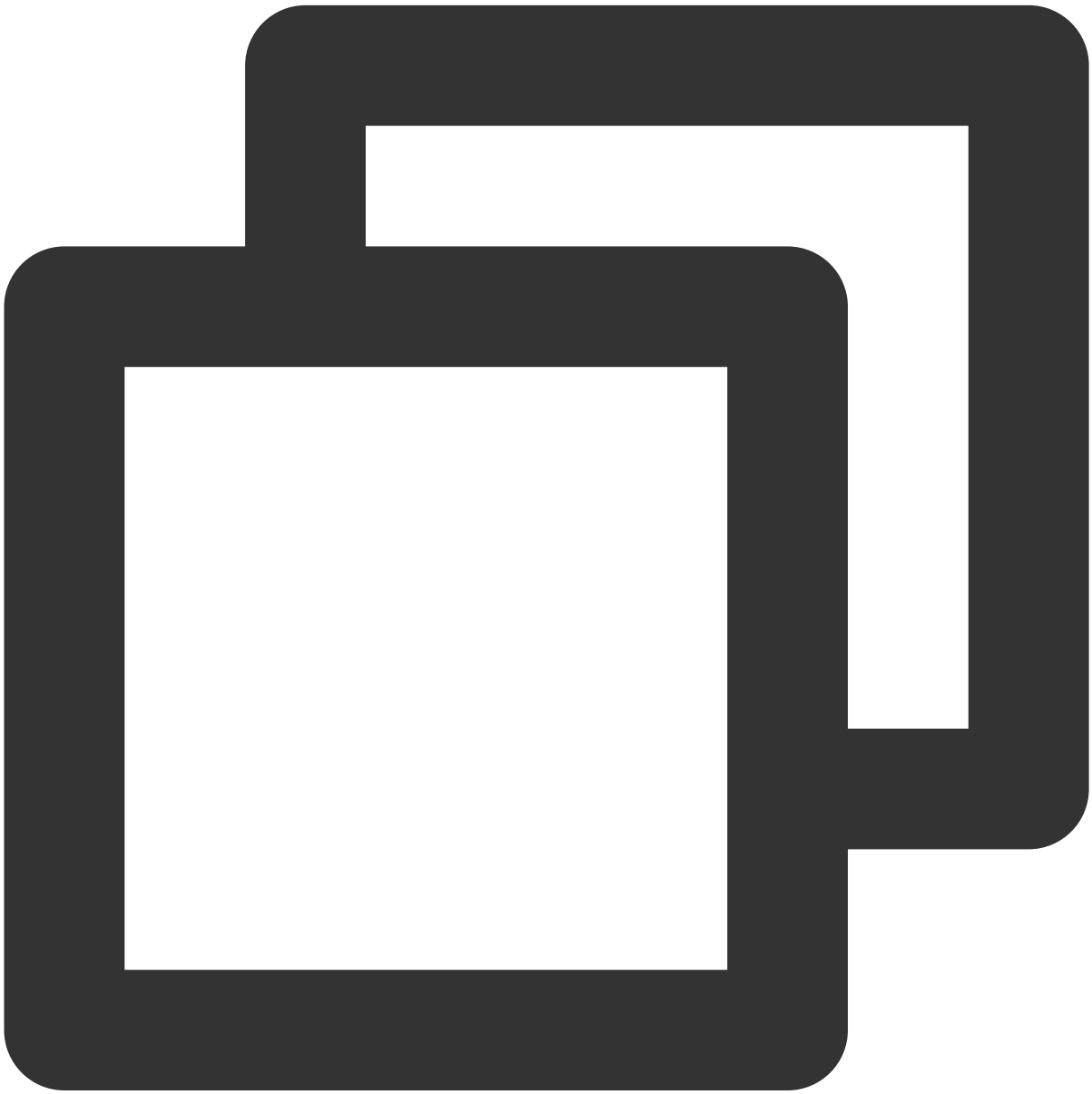
版权所有：腾讯云计算（北京）有限责任公司

该配置表示：

组件 `comp-a` 的占位组件为内置组件 `view`

组件 `comp-b` 的占位组件为自定义组件 `comp-c` （其路径在 `usingComponents` 中配置）

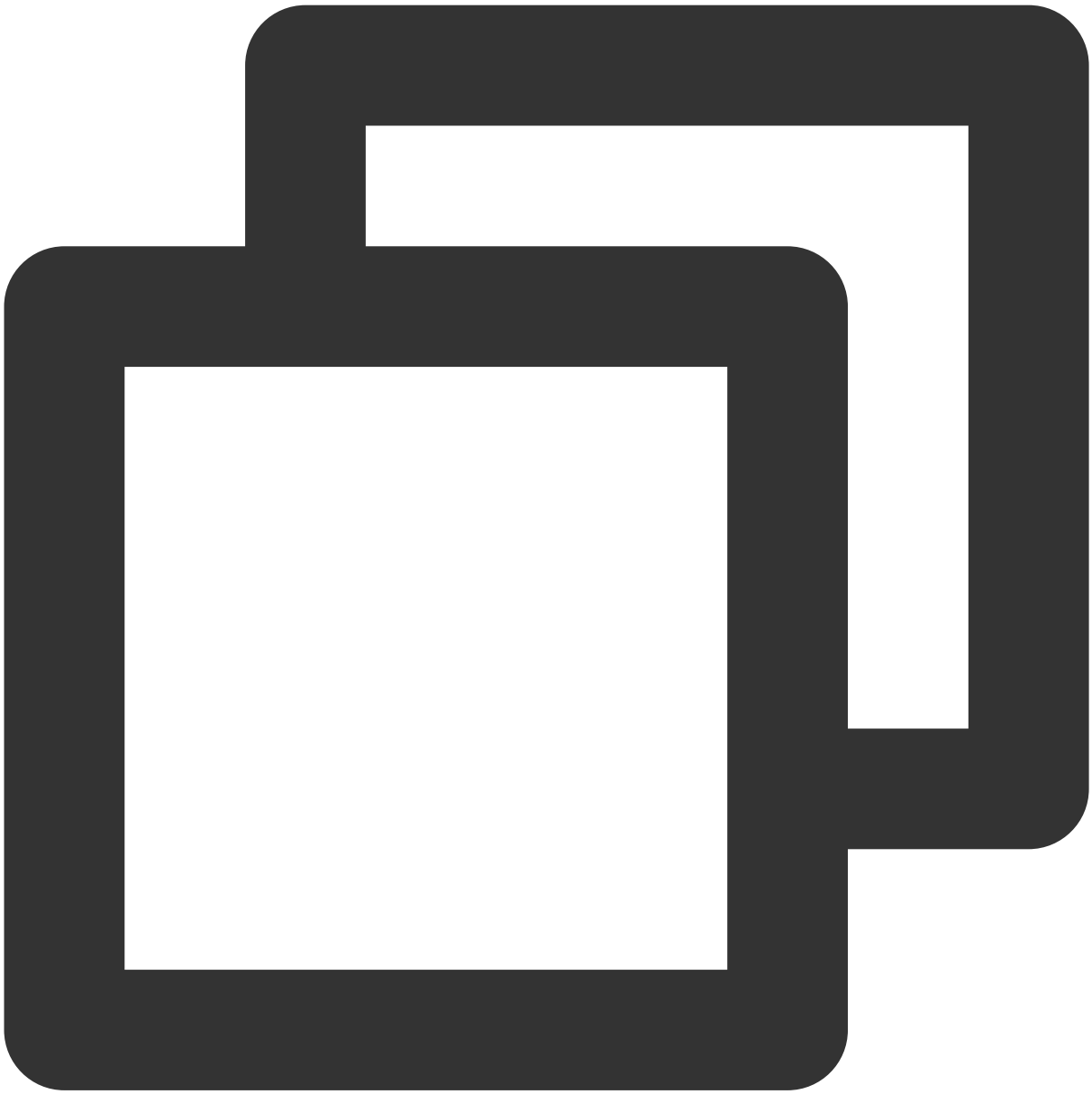
假设该配置对应的模板如下：



```
<button ontap="onTap">显示组件</button>
<comp-a wx-if="{{ visible }}">
  <comp-b prop="{{ p }}">text in slot</comp-b>
</comp-a>
```

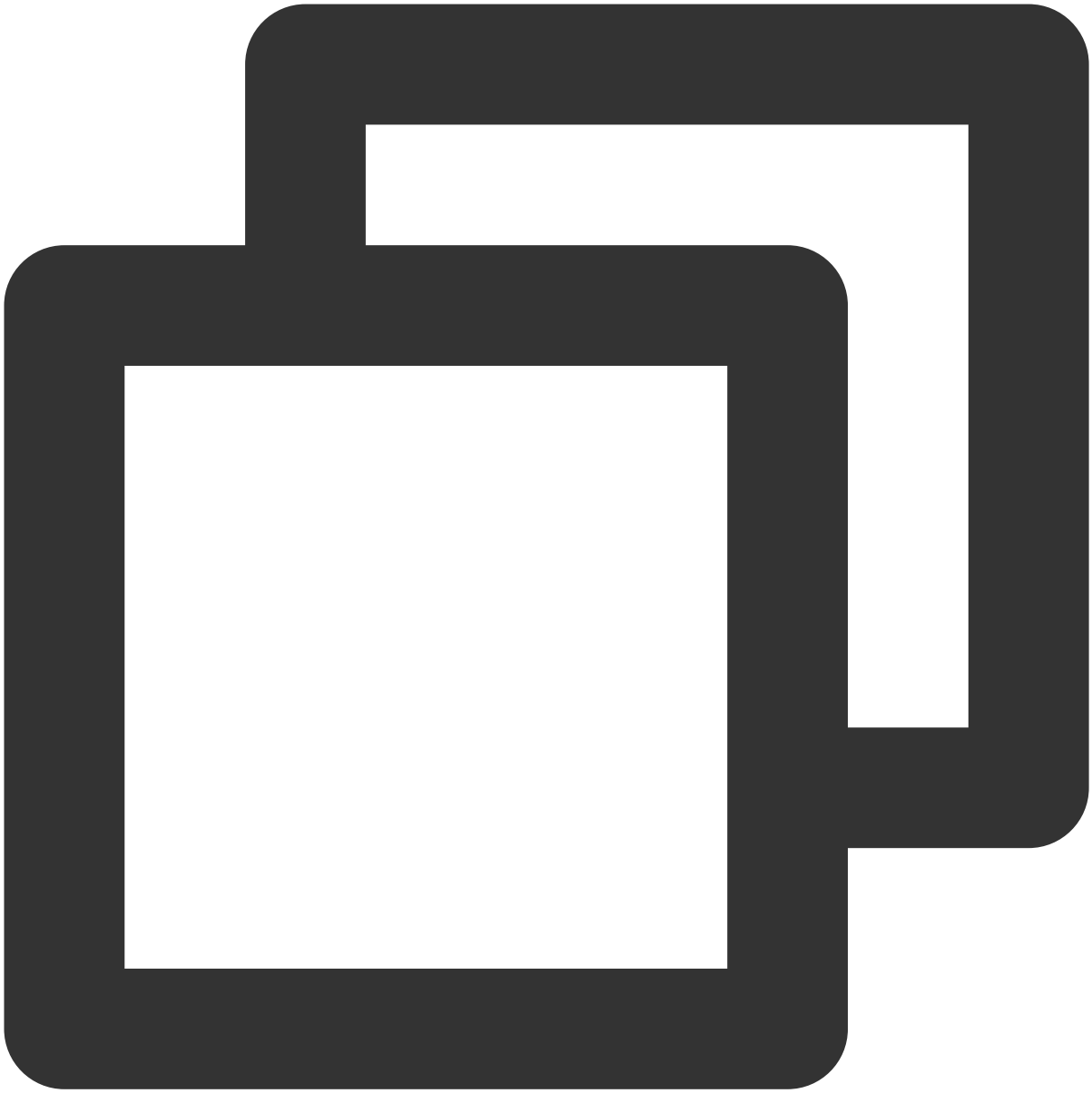
版权所有：腾讯云计算（北京）有限责任公司

小程序启动时 `visible` 为 `false`，那么只有 `button` 会被渲染；点击按钮后，`this.setData({ visible: true })` 被执行，此时如果 `comp-a`，`comp-b` 均不可用，则页面将被渲染为：



```
<button>显示组件</button>
<view>
  <comp-c prop="{{ p }}">text in slot</comp-c>
</view>
```

`comp-a` 与 `comp-b` 准备完成后，页面被替换为：



```
<button>显示组件</button>
<comp-a>
  <comp-b prop="{{ p }}">text in slot</comp-b>
</comp-a>
```

12.2 注意事项

1. 当一个组件被指定为占位组件时（如上例中的 `comp-c` ），为其指定占位组件是无效的。可以理解为如果一个组件需要作为其他组件的占位组件，则它**必须**在一开始就是**可用**的；
2. 目前自定义组件不可用的情况包括：

版权所有：腾讯云计算（北京）有限责任公司

2.1 使用分包异步化特性的情况下，引用了其他分包的组件，而对应分包还未下载；
2.2 使用时注入特性的情况下，该组件还未注入；
3. 如果一个组件不可用，且其占位组件不存在，则渲染时会报错并抛出；
4. 如果一个组件不存在，但为其指定了可用的占位组件，则占位组件可以被正常渲染，但后续尝试准备替换时会报错并抛出。

附：有占位组件参与的渲染流程

基础库尝试渲染一个组件时，会首先递归检查 `usingComponents`，收集其将使用到的所有组件的信息；在这个过程中，如果某个被使用到的组件不可用，基础库会先检查其是否有对应的占位组件。如果没有，基础库会中断渲染并抛出错误；如果有，则会标记并在后续渲染流程中使用占位组件替换该不可用的组件进行渲染。不可用的组件会在当前渲染流程结束后尝试准备（下载分包或注入代码等）；等到准备过程完成后，再尝试渲染该组件（实际上也是在执行这个流程），并替换掉之前渲染的占位组件。

版权所有：腾讯云计算（北京）有限责任公司

基础能力

网络

最近更新时间：2024-07-12 17:43:16

说明：

在小程序中使用网络相关的 API 时，需要注意下列问题，请开发者提前了解。

服务器域名配置

每个小程序需要事先设置一个通讯域名，小程序**只可以跟指定的域名与进行网络通信**。包括普通 HTTPS 请求（wx.request）、上传文件（wx.uploadFile）、下载文件（wx.downloadFile）和 Websocket通信（wx.connectSocket）、UDP通信（wx.createUDPSocket）、TCP通信（wx.createTCPSocket）。

网络接口只允许与同个局域网内的**非本机 IP** 以及配置过的服务器域名通信。

配置流程

服务器域名请在小程序后台 > 小程序管理 > 开发管理 > 域名设置 中进行配置，配置时需要注意：

域名支持 `https` （wx.request、wx.uploadFile、wx.downloadFile）协议、 `wss` （wx.connectSocket）协议及 tcp 协议和 udp 协议；

域名不能使用 IP 地址或 localhost；

不可以配置端口，如 http://miniApp.com，但请求的 URL 中可以包含端口，例如 http://miniApp.com:8080。

对于 `wss` 域名，无需配置端口，默认允许请求该域名下所有端口。

不支持配置父域名，使用子域名。

网络请求

超时时间

默认超时时间和最大超时时间都是60s。

超时时间可以在 app.json 中配置。

也可以在接口调用时指定超时时间，如 `wx.request({ timeout: 5000 })` ，单位为ms。接口调用的timeout 配置优先级高于 `app.json` 中的配置。

使用限制

网络请求的 referer header 不可设置。其格式固定为 https://appservice.wx.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为0表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本。

wx.request、wx.uploadFile、wx.downloadFile的最大并发限制是 **10** 个。

wx.connectSocket 的最大并发限制是 **5** 个；

版权所有：腾讯云计算（北京）有限责任公司

小程序进入后台运行后，如果 **5s** 内网络请求没有结束，会回调错误信息 `fail interrupted` ；在回到前台之前，网络请求接口调用都会无法调用。

返回值编码

建议服务器返回值使用 **UTF-8** 编码。对于非 UTF-8 编码，小程序会尝试进行转换，但是会有转换失败的可能。
小程序会自动对 BOM 头进行过滤（只过滤一个 BOM 头）。

回调函数

只要成功接收到服务器返回，无论 `statusCode` 是多少，都会进入 `success` 回调。请开发者根据业务逻辑对返回值进行判断。

常见问题

HTTPS 证书

小程序必须使用 HTTPS/WSS 发起网络请求。请求时系统会对服务器域名使用的 HTTPS 证书进行校验，如果校验失败，则请求不能成功发起。由于系统限制，不同平台对于证书要求的严格程度不同。为了保证小程序的兼容性，建议开发者按照最高标准进行证书配置，并使用相关工具检查现有证书是否符合要求。
对证书要求如下：
HTTPS 证书必须有效；
证书必须被系统信任，即根证书被已系统内置。
部署 SSL 证书的网站域名必须与证书颁发的域名一致。
证书必须在有效期内。
证书的信任链必需完整（需要服务器配置）。
`iOS` 不支持自签名证书。
`iOS` 下证书必须满足苹果 App Transport Security (ATS) 的要求。
TLS 必须支持 1.2 及以上版本。部分旧 `Android` 机型还未支持 TLS 1.2，请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本。
部分 CA 可能不被操作系统信任，请开发者在选择证书时注意小程序和各系统的相关通告。
说明：
证书有效性可以使用 `openssl s_client -connect example.com:443` 命令验证，也可以使用其他 在线工具。
除了网络请求 API 外，小程序中其他 HTTPS 请求如果出现异常，也请按上述流程进行检查。如 https 的图片无法加载、音视频无法播放等。

跳过域名校验

在开发者工具中，可以临时开启 `开发环境不校验请求域名、TLS版本及 HTTPS 证书` 选项，跳过服务器域名的校验。此时，在开发者工具中及手机开启调试模式时，不会进行服务器域名的校验。

版权所有：腾讯云计算（北京）有限责任公司

在服务器域名配置成功后，建议开发者关闭此选项进行开发，并在各平台下进行测试，以确认服务器域名配置正确。

说明：

如果手机上出现 “打开调试模式可以发出请求，关闭调试模式无法发出请求” 的现象，请确认是否跳过了域名校验，并确认服务器域名和证书配置是否正确。

如遇到“服务器内部错误”，请尝试以下办法自查 ①更换浏览器 ②刷新网页 ③排查文件是否放在正确的服务器根目录下。 排查办法：按照以下规则拼接网址，并且自行尝试访问是否成功：https://m.da9c.cn/文件名。

版权所有：腾讯云计算（北京）有限责任公司

存储

最近更新时间：2024-07-12 17:42:09

每个小程序都可以有自己的本地缓存，可以通过 wx.setStorage/wx.setStorageSync、wx.getStorage/wx.getStorageSync、wx.clearStorage/wx.clearStorageSync，wx.removeStorage/wx.removeStorageSync 对本地缓存进行读写和清理。

隔离策略

storage 以用户维度隔离，同一台设备上，A 用户无法读取到 B 用户的数据；不同小程序之间也无法互相读写数据。同一个用户，同一个小程序 storage 上限为 10MB。

清理策略

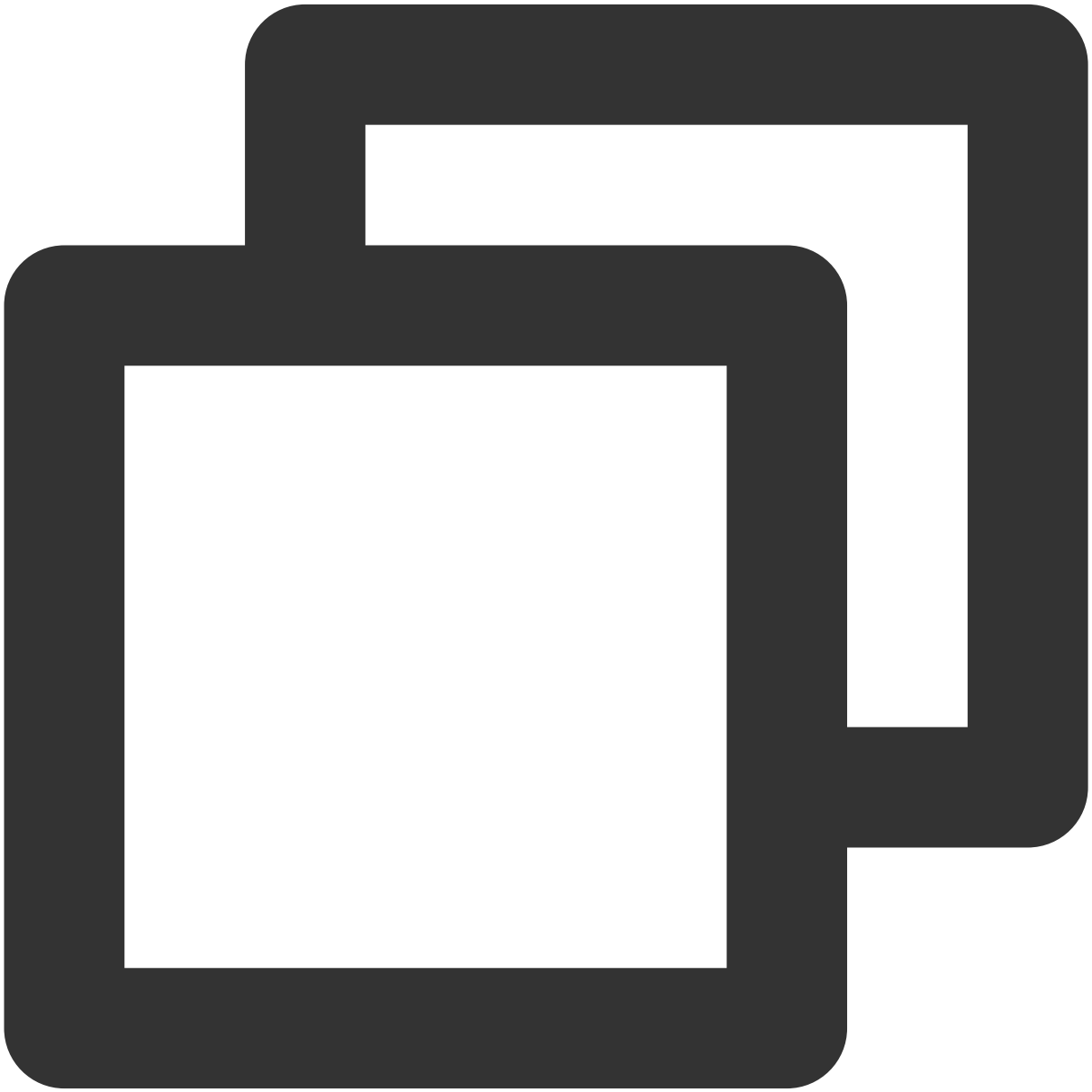
本地缓存的清理时机跟代码包一样，只有在代码包被清理的时候本地缓存才会被清理。

版权所有：腾讯云计算（北京）有限责任公司

文件系统

最近更新时间：2024-07-12 17:41:49

文件系统是小程序提供的一套以小程序和用户维度隔离的存储以及一套相应的管理接口。通过 wx.getFileSystemManager() 可以获取到全局唯一的文件系统管理器，所有文件系统的管理操作通过 FileSystemManager 来调用。



```
var fs = wx.getFileSystemManager()
```

版权所有：腾讯云计算（北京）有限责任公司

文件主要分为两大类：
代码包文件：代码包文件指的是在项目目录中添加的文件。
本地文件：通过调用接口本地产生，或通过网络下载下来，存储到本地的文件。
其中本地文件又分为三种：
本地临时文件：临时产生，随时会被回收的文件。运行时最多存储 4GB，结束运行后，如果已使用超过 2GB，会以文件为维度按照最近使用时间从远到近进行清理至少于2GB。
本地缓存文件：小程序通过接口把本地临时文件缓存后产生的文件，不能自定义目录和文件名。跟本地用户文件共计，小程序（含小游戏）最多可存储 200MB。
本地用户文件：小程序通过接口把本地临时文件缓存后产生的文件，允许自定义目录和文件名。跟本地缓存文件共计，小程序（含小游戏）最多可存储 200MB。

1.代码包文件

由于代码包文件大小限制，代码包文件适用于放置首次加载时需要的文件，对于内容较大或需要动态替换的文件，不推到代码包中，推荐在小程序启动之后再用下载接口下载到本地。

1.1 访问代码包文件

代码包文件的访问方式是从项目根目录开始写文件路径，不支持相对路径的写法。如：`/a/b/c`、`a/b/c` 都是合法的，`./a/b/c`、`../a/b/c` 则不合法。

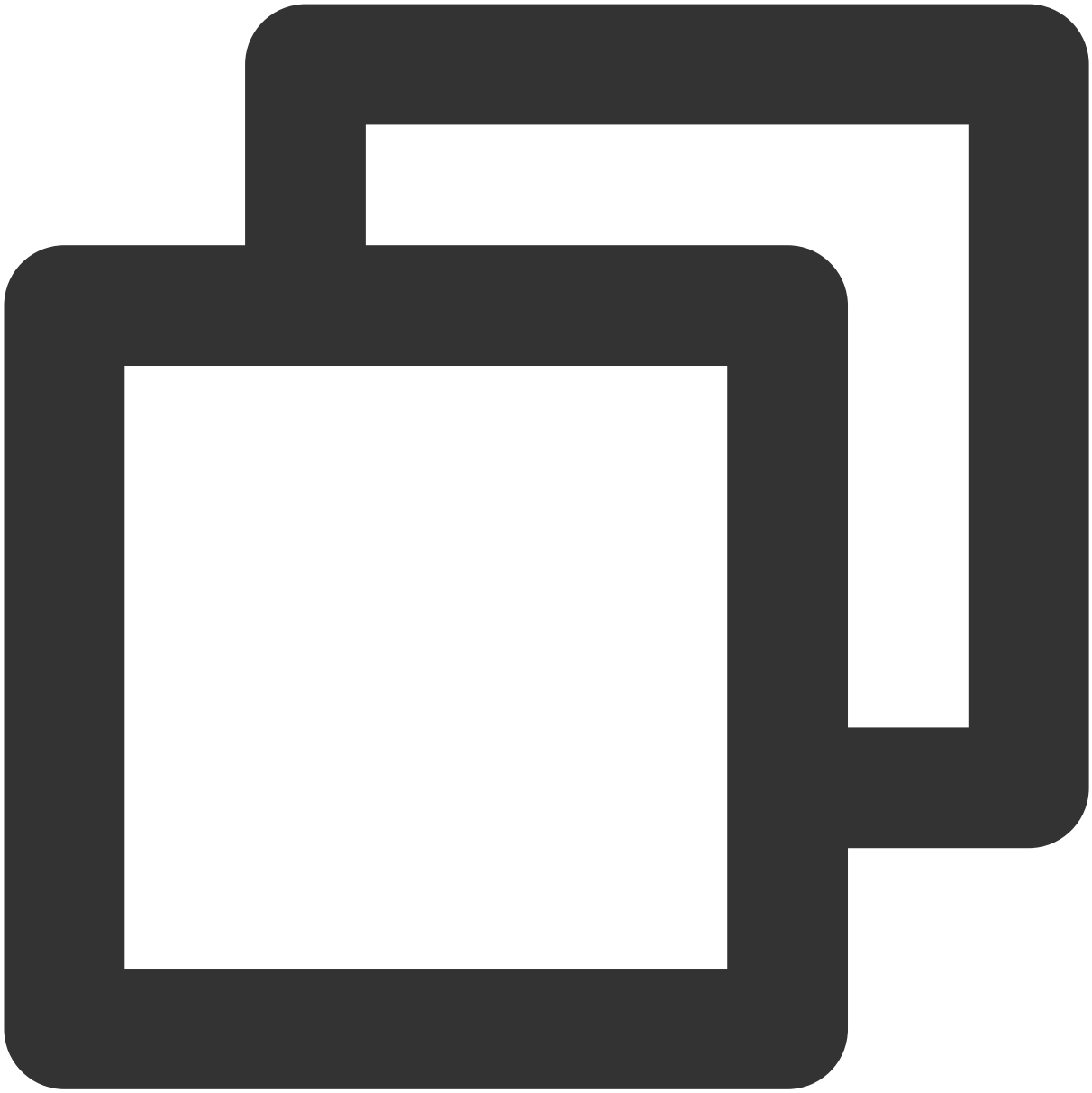
1.2 修改代码包文件

代码包内的文件无法在运行后动态修改或删除，修改代码包文件需要重新发布版本。

2.本地文件

本地文件指的是小程序被用户添加到手机后，会有一块独立的文件存储区域，以用户维度隔离。即同一台手机，每个用户不能访问到其他登录用户的文件，同一个用户不同 appId 之间的文件也不能互相访问。
本地文件的文件路径均为以下格式：

版权所有：腾讯云计算（北京）有限责任公司



```
{{协议名}}://文件路径
```

说明：

其中，协议名在 iOS/Android 客户端为 "wxfile"，在开发者工具上为 "http"，开发者无需关注这个差异，也不应在代码中去硬编码完整文件路径。

2.1 本地临时文件

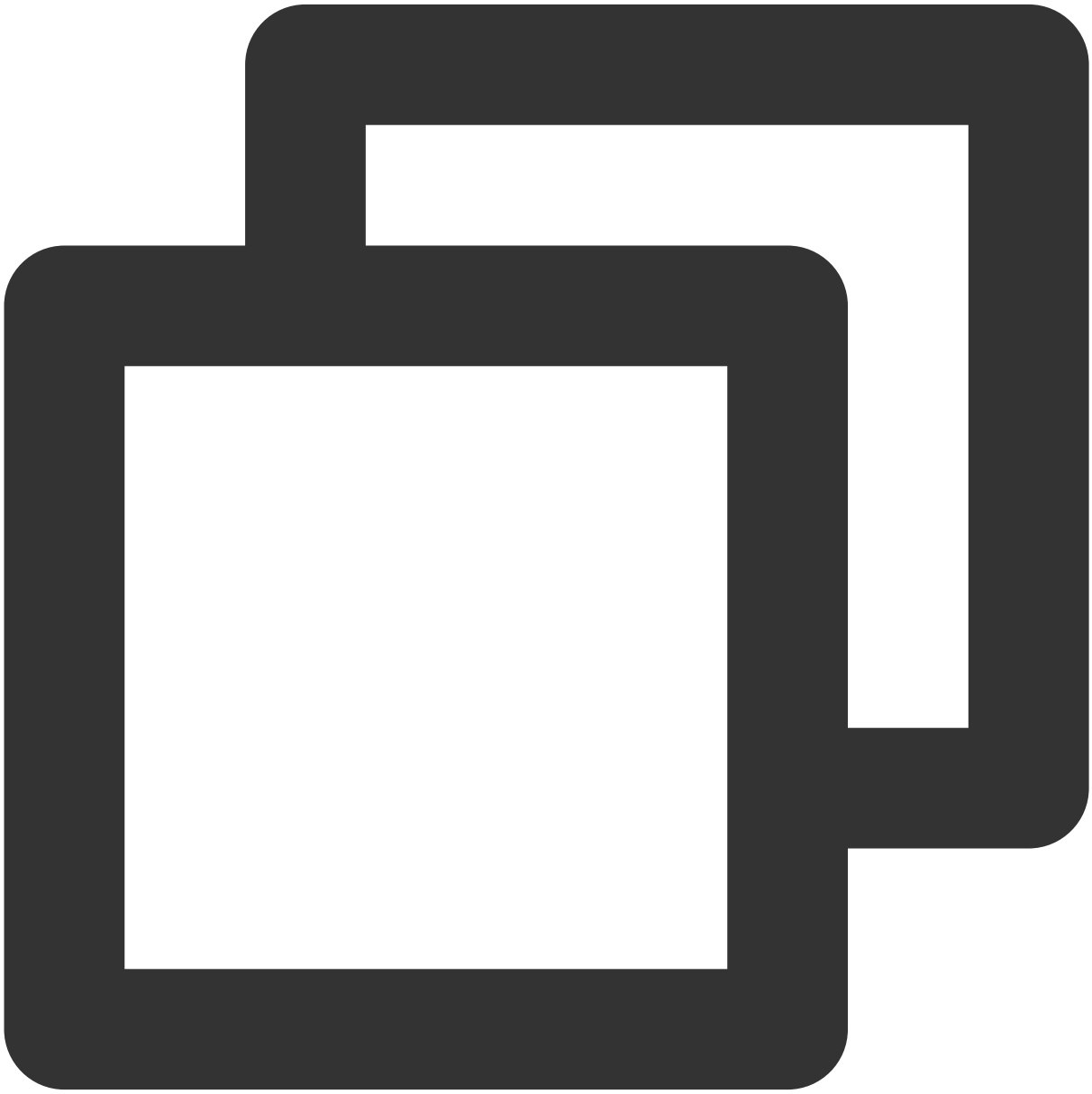
本地临时文件只能通过调用特定接口产生，不能直接写入内容。本地临时文件产生后，仅在当前生命周期内保证有效，重启之后不一定可用。如果需要保证在下次启动时无需下载，可通过 FileSystemManager.saveFile() 或

FileSystemManager.copyFile() 接口把本地临时文件转换成本地缓存文件或本地用户文件。

临时文件的清理策略为：小程序退出后系统会检查该小程序的临时文件占用，若不超过2GB则不进行清理，超过上限则以文件为维度按照最近使用时间从远到近进行清理。同时也会检查所有小程序的临时文件占用，若超过6GB则以小程序为维度进行清理。

因此，开发者在下载临时文件时，可先通过FileSystemManager.access()检查该文件是否存在，减少重复文件下载，提升用户体验。

代码示例：



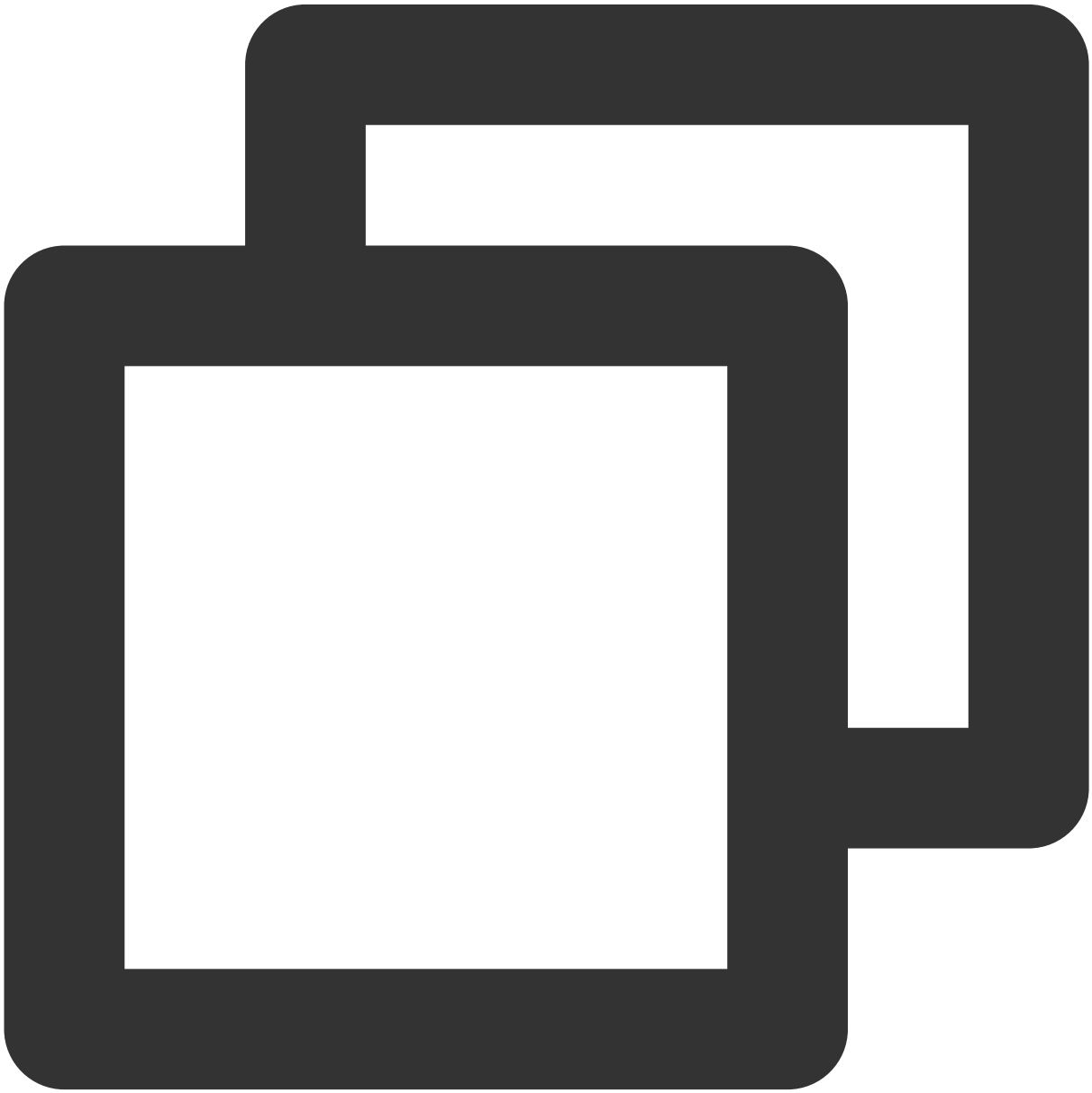
```
wx.chooseImage({
  success: function (res) {
```

```
    var tempFilePaths = res.tempFilePaths // tempFilePaths 的每一项是一个本地临时文件路径
  }
})
```

2.2 本地缓存文件

本地缓存文件只能通过调用特定接口产生，不能直接写入内容。本地缓存文件产生后，重启之后仍可用。本地缓存文件只能通过 FileSystemManager.saveFile() 接口将本地临时文件保存获得。

代码示例：



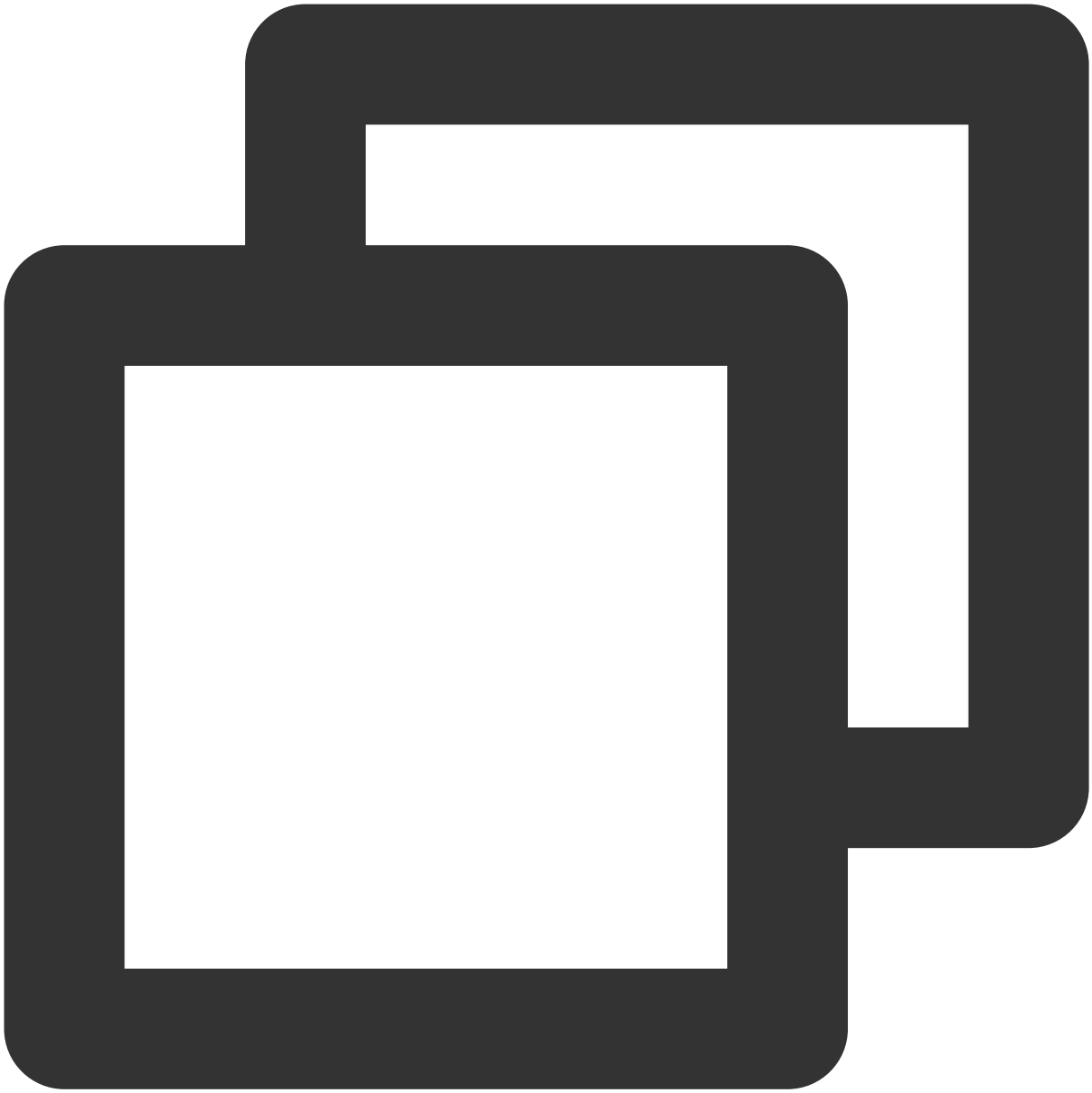
```
fs.saveFile({
```

```
  tempFilePath: '', // 传入一个本地临时文件路径
  success(res) {
    console.log(res.savedFilePath) // res.savedFilePath 为一个本地缓存文件路径
  }
})
```

2.3 本地用户文件

我们提供了一个用户文件目录给开发者，开发者对这个目录有完全自由的读写权限。通过 wx.env.USER_DATA_PATH 可以获取到这个目录的路径。

代码示例：



```
// 在本地用户文件目录下创建一个文件 hello.txt, 写入内容 "hello, world"
const fs = wx.getFileSystemManager()
fs.writeFileSync(`${wx.env.USER_DATA_PATH}/hello.txt`,  'hello, world', 'utf8')
```

2.4 读写权限

接口、组件	读	写
代码包文件	有	无

版权所有：腾讯云计算（北京）有限责任公司

本地临时文件	有	无
本地缓存文件	有	无
本地用户文件	有	有

2.5 清理策略

本地临时文件只保证在小程序当前生命周期内，一旦小程序被关闭就可能被清理，即下次冷启动不保证可用。
本地缓存文件和本地用户文件的清理时机跟代码包一样，只有在代码包被清理的时候才会一同被清理。

版权所有：腾讯云计算（北京）有限责任公司

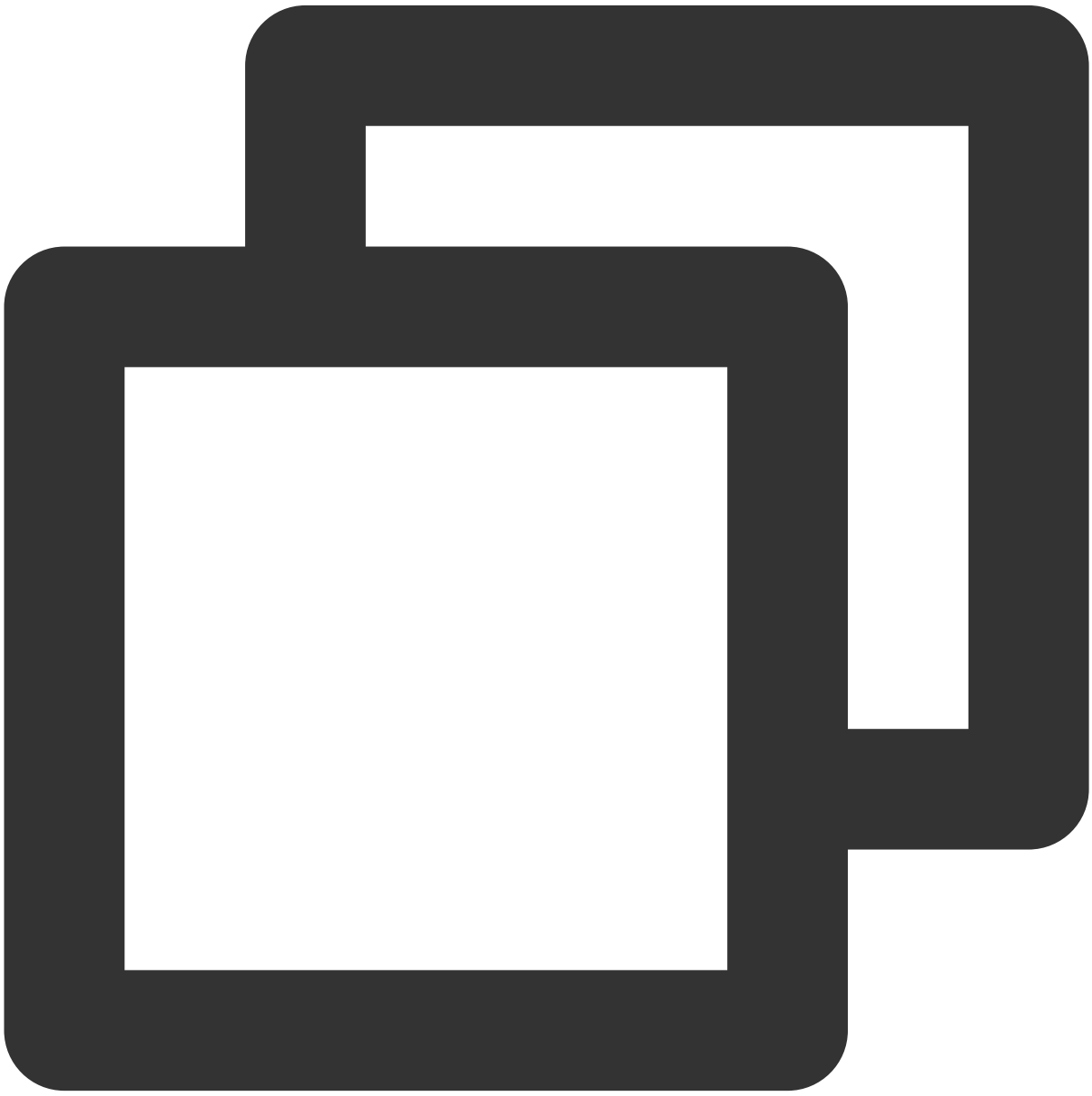
分包加载

最近更新时间：2024-08-09 17:54:21

1. 使用分包

1.1 配置方法

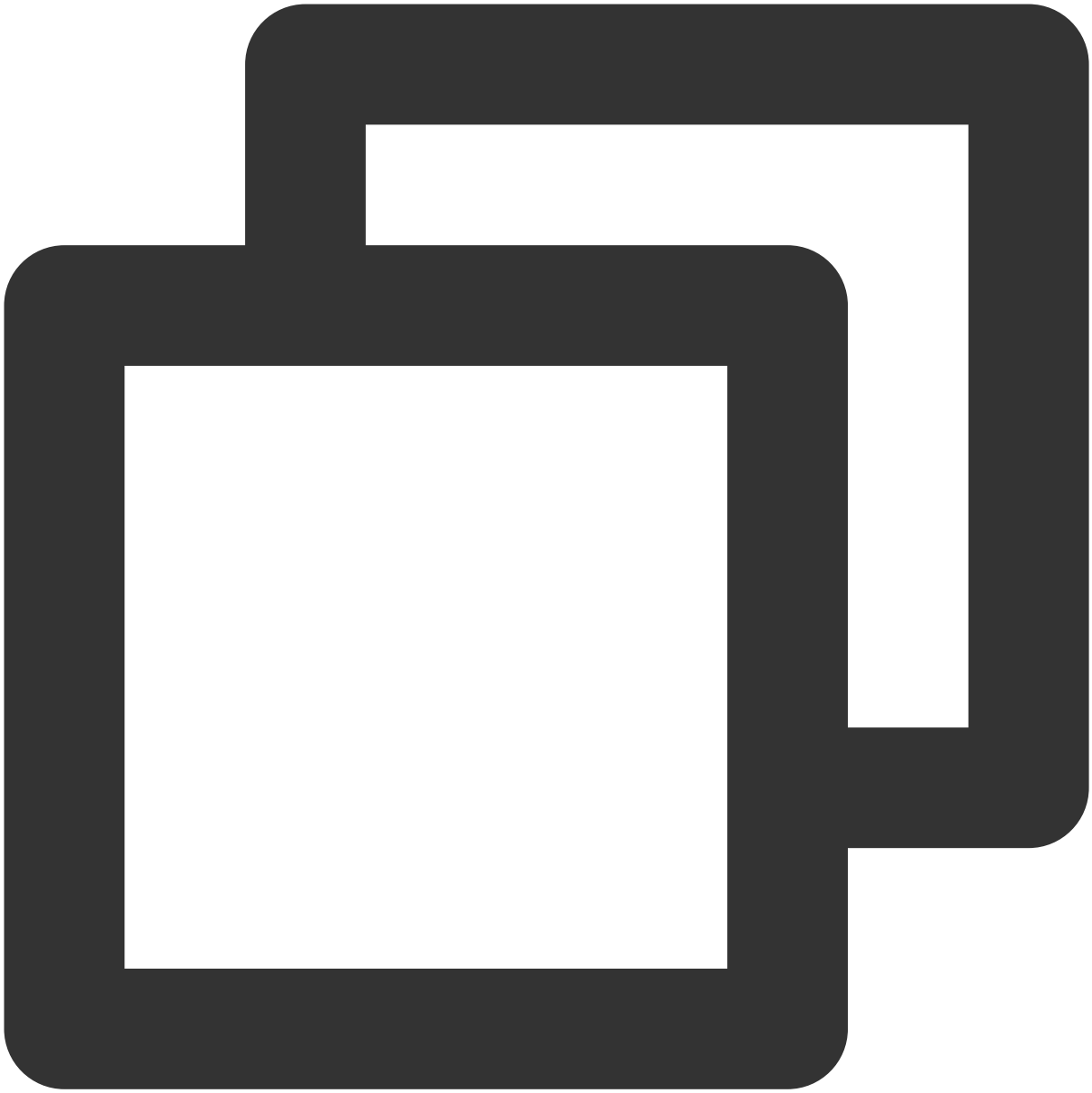
假设支持分包的小程序目录结构如下：



```
├── app.js
├── app.json
├── app.wxss
├── packageA
│   └── pages
│       ├── cat
│       └── dog
├── packageB
│   └── pages
│       ├── apple
│       └── banana
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 app.json `subpackages` 字段声明项目分包结构：

> 写成 subPackages 也支持。



```
{
  "pages":[
    "pages/index",
    "pages/logs"
  ],
  "subpackages": [
    {
      "root": "packageA",
      "pages": [
        "pages/cat",
        "pages/dog"
```

版权所有：腾讯云计算（北京）有限责任公司

```
      ]
    }, {
      "root": "packageB",
      "name": "pack2",
      "pages": [
        "pages/apple",
        "pages/banana"
      ]
    }
  ]
}
```

`subpackages` 中，每个分包的配置有以下几项：

字段	类型	说明
root	String	分包根目录
name	String	分包别名，分包预下载时可以使用
pages	StringArray	分包页面路径，相对于分包根目录
independent	Boolean	分包是否是独立分包

1.2 打包原则

声明 `subpackages` 后，将按 `subpackages` 配置路径进行打包，`subpackages` 配置路径外的目录将被打包到主包中。

主包也可以有自己的 pages，即最外层的 pages 字段。

`subpackage` 的根目录不能是另外一个 `subpackage` 内的子目录。

`tabBar` 页面必须在主包内。

1.3 引用原则

`packageA` 无法 require `packageB` JS 文件，但可以 require 主包、`packageA` 内的 JS 文件；使用 分包异步化 时不受此条限制。

`packageA` 无法 import `packageB` 的 template，但可以 require 主包、`packageA` 内的 template。

`packageA` 无法使用 `packageB` 的资源，但可以使用主包、`packageA` 内的资源。

2. 独立分包

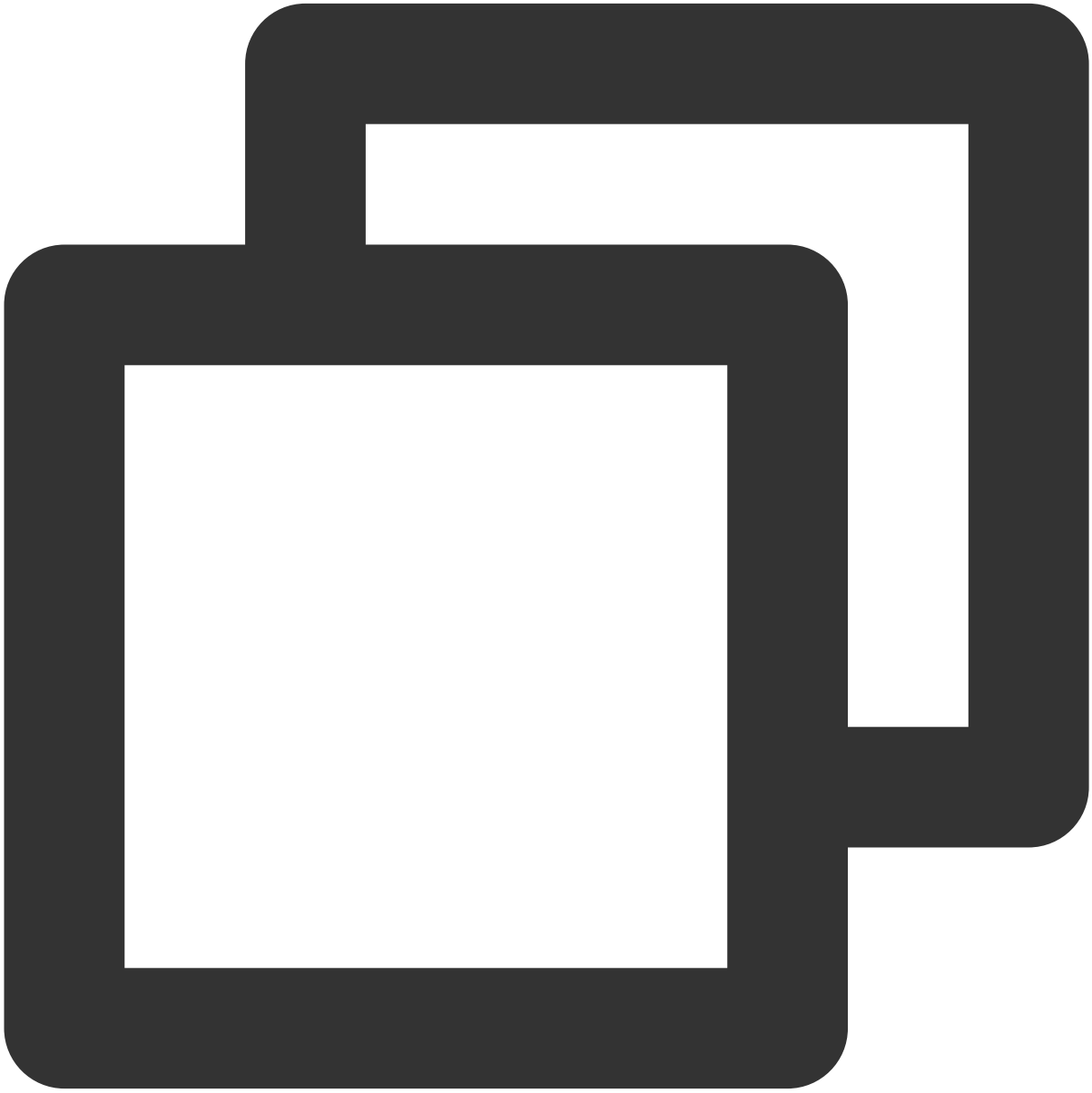
独立分包是小程序中一种特殊类型的分包，可以独立于主包和其他分包运行。从独立分包中页面进入小程序时，不需要下载主包。当用户进入普通分包或主包内页面时，主包才会被下载。

版权所有：腾讯云计算（北京）有限责任公司

开发者可以按需将某些具有一定功能独立性的页面配置到独立分包中。当小程序从普通的分包页面启动时，需要首先下载主包；而独立分包不依赖主包即可运行，可以很大程度上提升分包页面的启动速度。
一个小程序中可以有多个独立分包。

2.1 配置方法

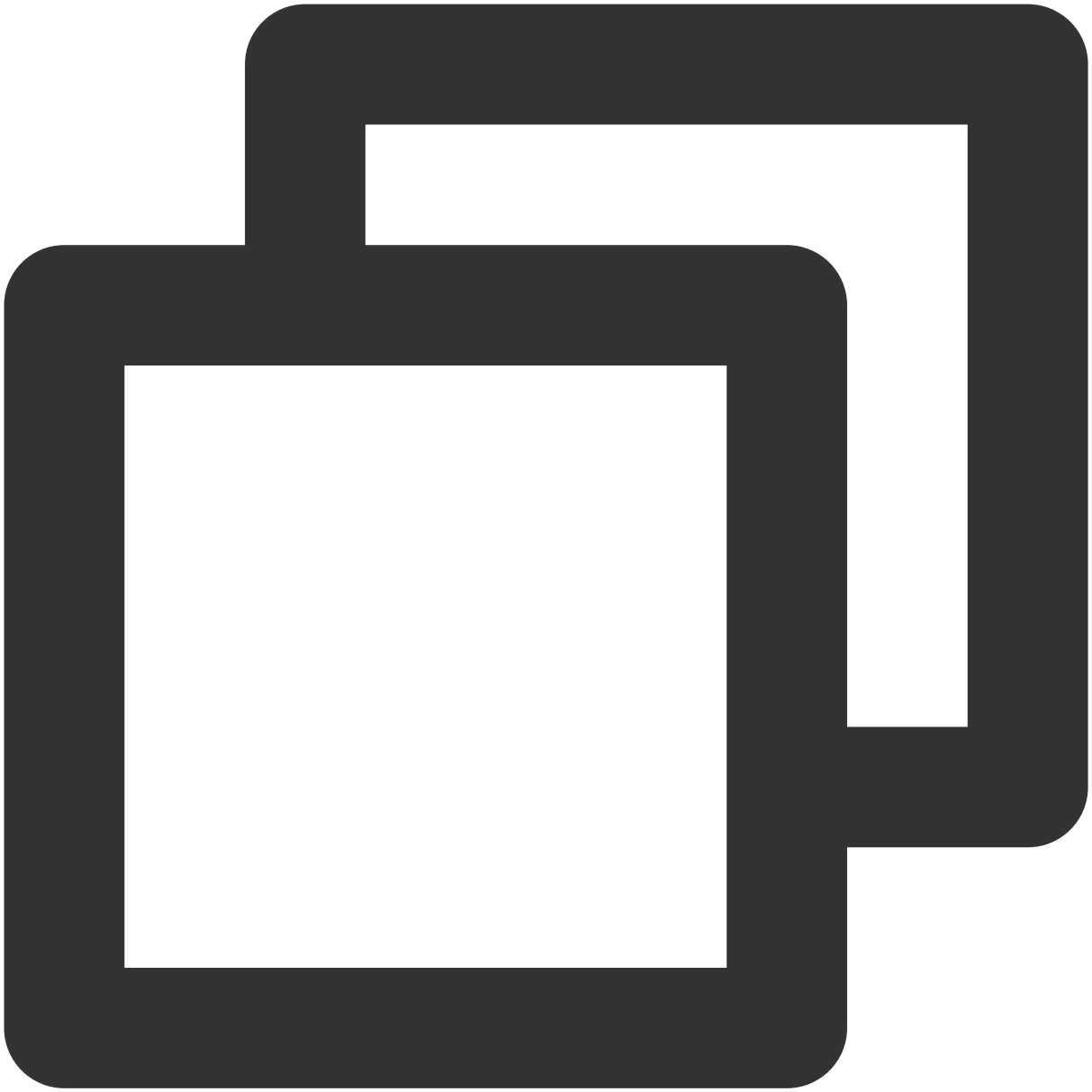
假设小程序目录结构如下：



```
├── app.js
├── app.json
├── app.wxss
```

```
├── moduleA
│   └── pages
│       ├── rabbit
│       └── squirrel
├── moduleB
│   └── pages
│       ├── pear
│       └── pineapple
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 `app.json` 的 `subpackages` 字段中对应的分包配置项中定义 `independent` 字段声明对应分包为独立分包。



```
{
  "pages": [
    "pages/index",
    "pages/logs"
  ],
  "subpackages": [
    {
      "root": "moduleA",
      "pages": [
        "pages/rabbit",
        "pages/squirrel"
```

版权所有：腾讯云计算（北京）有限责任公司

```
      ]
    }, {
      "root": "moduleB",
      "pages": [
        "pages/pear",
        "pages/pineapple"
      ],
      "independent": true
    }
  ]
}
```

2.2 限制

独立分包属于分包的一种。普通分包的所有限制都对独立分包有效。独立分包中自定义组件的处理方式同普通分包。

此外，使用独立分包时要注意：

独立分包中不能依赖主包和其他分包中的内容，包括 js 文件、template、wxss、自定义组件等（使用 分包异步化 时 js 文件、自定义组件不受此条限制）；

主包中的 app.wxss 对独立分包无效，应避免在独立分包页面中使用 app.wxss 中的样式；

App 只能在主包内定义，独立分包中不能定义 App，会造成无法预期的行为；

2.3 注意事项

2.3.1 关于 `getApp()`

与普通分包不同，独立分包运行时，`App` 并不一定被注册，因此 `getApp()` 也不一定可以获得 `App` 对象：

当用户从独立分包页面启动小程序时，主包不存在，`App` 也不存在，此时调用 `getApp()` 获取到的是 `undefined`。当用户进入普通分包或主包内页面时，主包才会被下载，`App` 才会被注册。

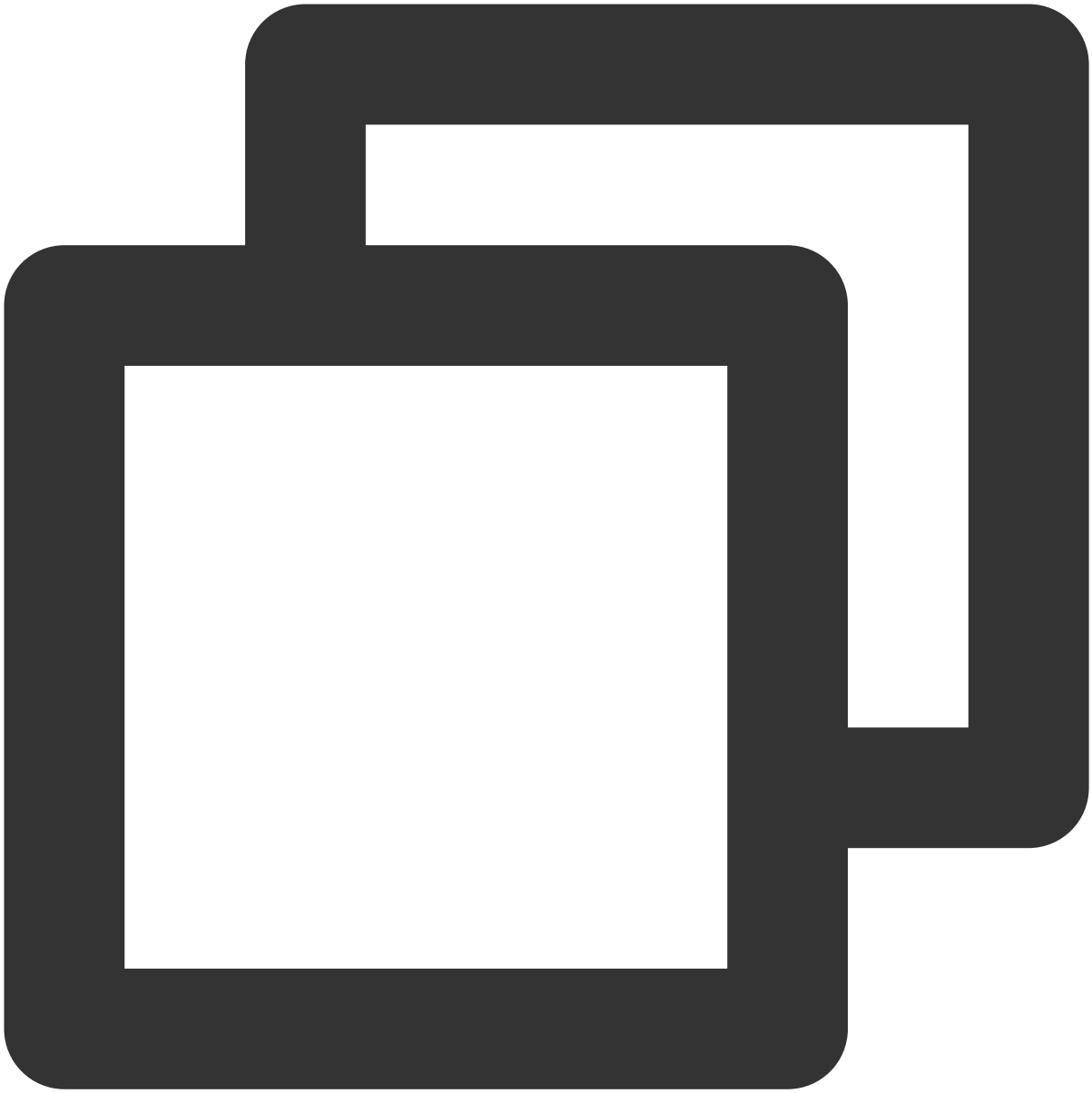
当用户是从普通分包或主包内页面跳转到独立分包页面时，主包已经存在，此时调用 `getApp()` 可以获取到真正的 `App`。

由于这一限制，开发者无法通过 `App` 对象实现独立分包和小程序其他部分的全局变量共享。

为了在独立分包中满足这一需求，`getApp` 支持 [`allowDefault`] 参数，在 `App` 未定义时返回一个默认实现。当主包加载，`App` 被注册时，默认实现中定义的属性会被覆盖合并到真正的 `App` 中。

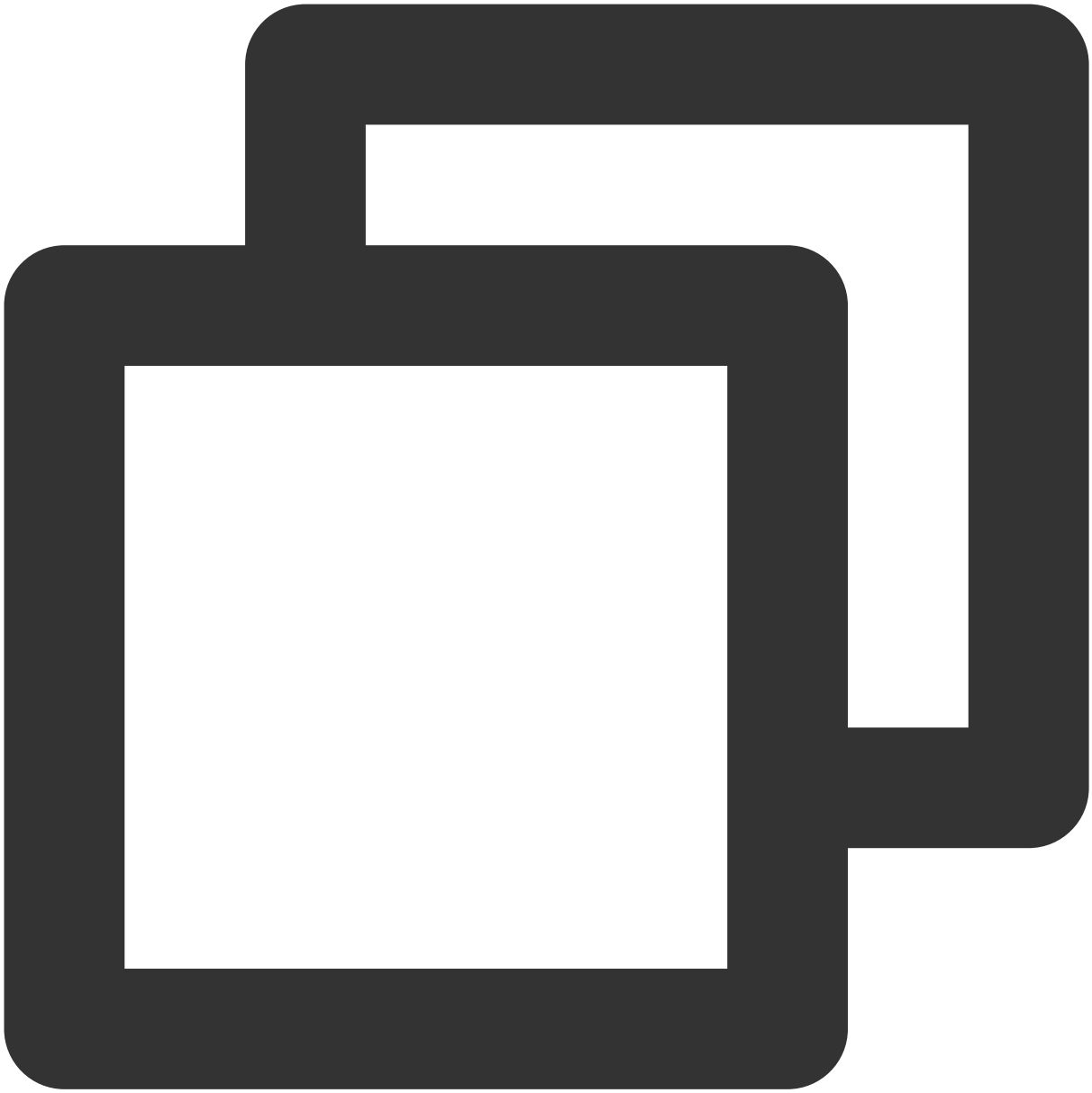
示例代码：

独立分包中



```
const app = getApp({allowDefault: true}) // {}
app.data = 456
app.global = {}
```

app.js中



```
App({
  data: 123,
  other: 'hello'
})

console.log(getApp()) // {global: {}, data: 456, other: 'hello'}
```

2.3.2 关于 App 生命周期

当从独立分包启动小程序时，主包中 `App` 的 `onLaunch` 和首次 `onShow` 会在从独立分包页面首次进入主包或其他普通分包页面时调用。

由于独立分包中无法定义 `App`，小程序生命周期的监听可以使用 wx.onAppShow，wx.onAppHide 完成。`App` 上的其他事件可以使用 wx.onError，wx.onPageNotFound 监听。

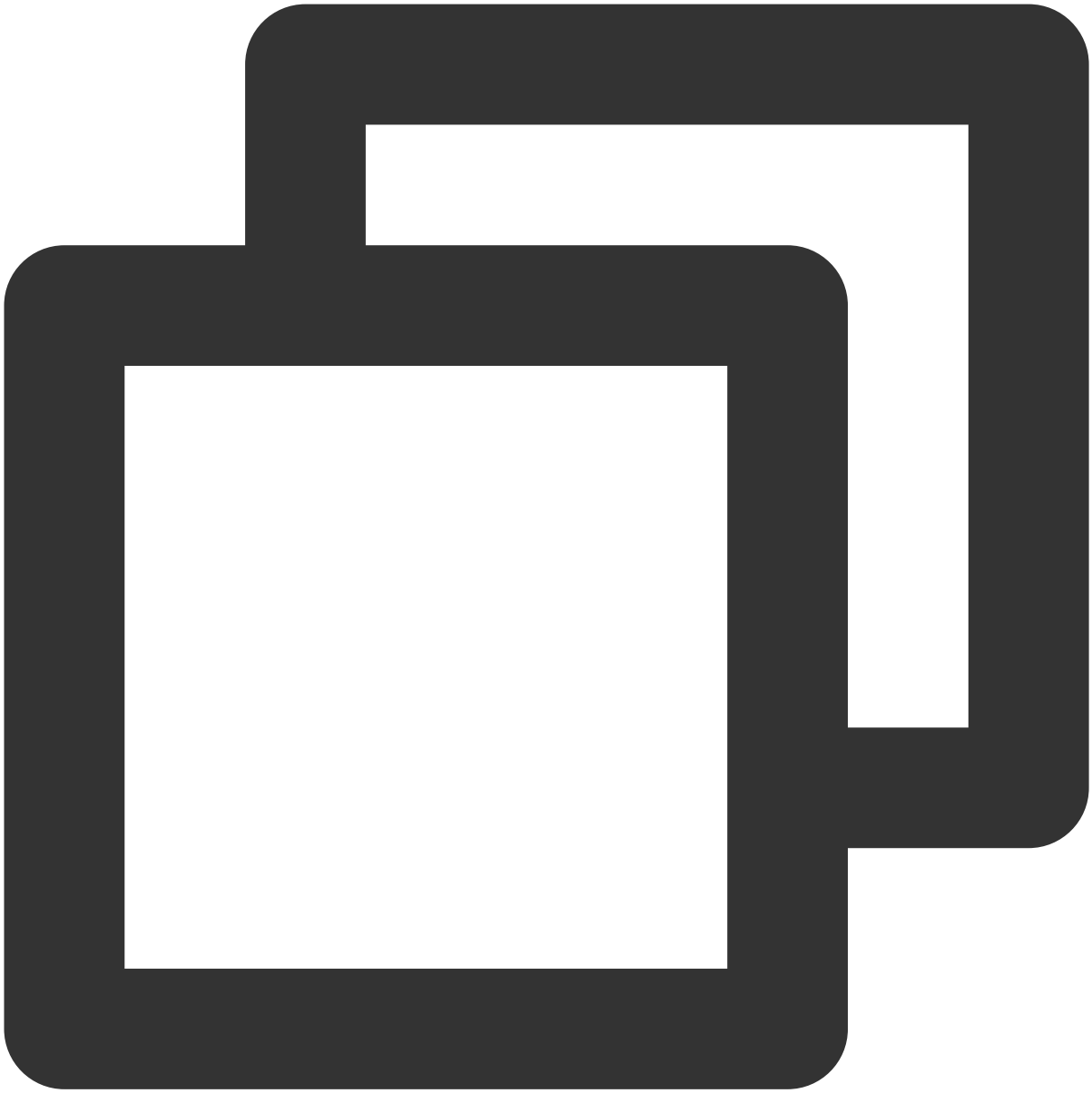
3. 分包预下载

开发者可以通过配置，在进入小程序某个页面时，由框架自动预下载可能需要的分包，提升进入后续分包页面时的启动速度。对于 独立分包，也可以预下载主包。
分包预下载目前只支持通过配置方式使用，暂不支持通过调用 API 完成。

> vConsole 里有preloadSubpackages开头的日志信息，可以用来验证预下载的情况。

3.1 配置方法

预下载分包行为在进入某个页面时触发，通过在 `app.json` 增加 `preloadRule` 配置来控制。



```
{
  "pages": ["pages/index"],
  "subpackages": [
    {
      "root": "important",
      "pages": ["index"],
    },
    {
      "root": "sub1",
      "pages": ["index"],
    },
```

版权所有：腾讯云计算（北京）有限责任公司

```
    {
      "name": "hello",
      "root": "path/to",
      "pages": ["index"]
    },
    {
      "root": "sub3",
      "pages": ["index"]
    },
    {
      "root": "indep",
      "pages": ["index"],
      "independent": true
    }
  ],
  "preloadRule": {
    "pages/index": {
      "network": "all",
      "packages": ["important"]
    },
    "sub1/index": {
      "packages": ["hello", "sub3"]
    },
    "sub3/index": {
      "packages": ["path/to"]
    },
    "indep/index": {
      "packages": ["__APP__"]
    }
  }
}
```

`preloadRule` 中，`key` 是页面路径，`value` 是进入此页面的预下载配置，每个配置有以下几项：

字段	类型	必填	默认值	说明
packages	StringArray	是	无	进入页面后预下载分包的 root 或 name，__APP__ 表示主包。
network	String	否	wifi	在指定网络下预下载，可选值为： all: 不限网络； wifi: 仅wifi下预下载。

3.2 限制

同一个分包中的页面享有共同的预下载大小限额2M，限额会在工具中打包时校验。

如，页面 A 和 B 都在同一个分包中，A 中预下载总大小0.5M的分包，B中最多只能预下载总大小1.5M的分包。

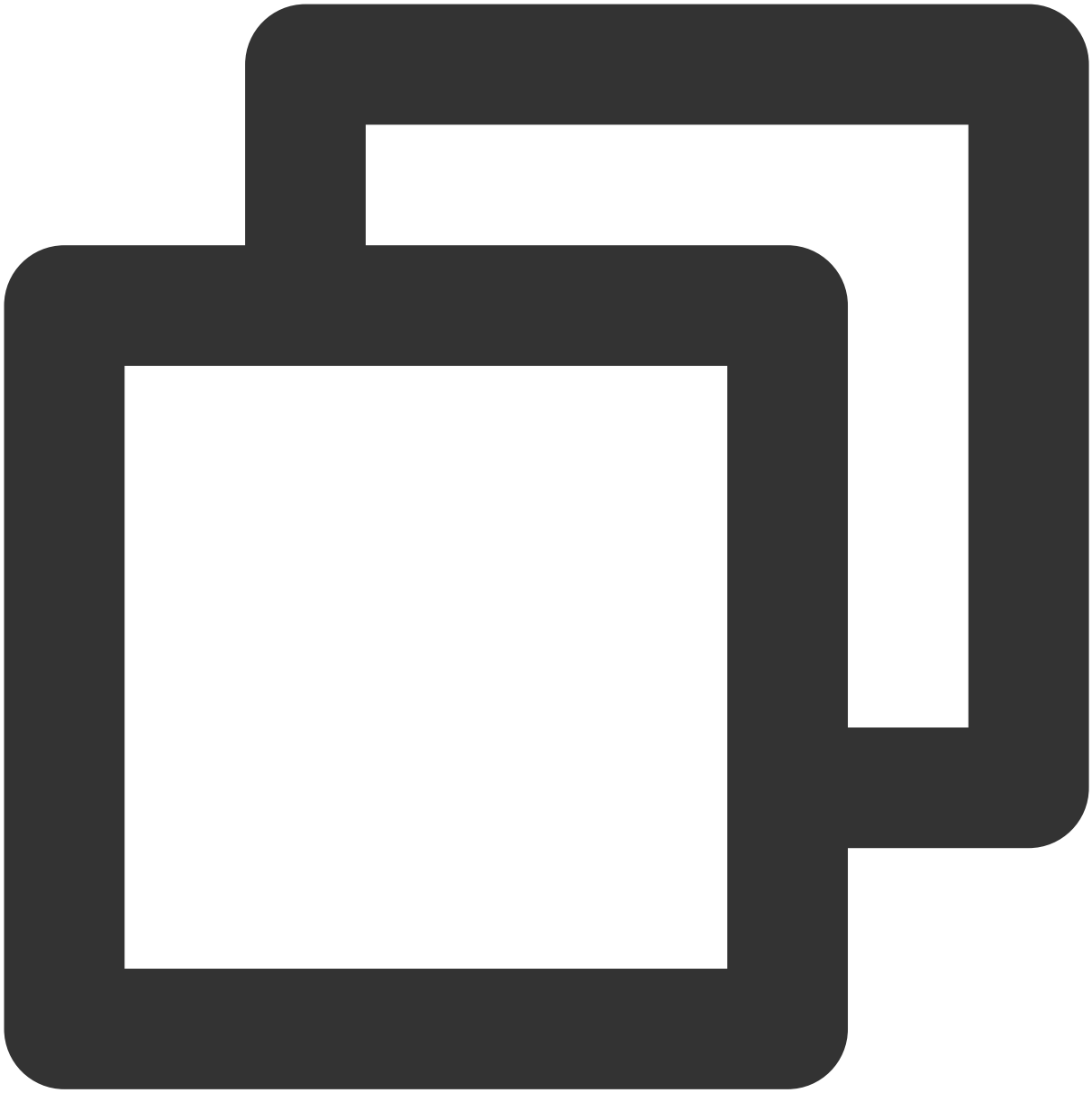
4. 分包异步化

在小程序中，不同的分包对应不同的下载单元；因此，除了非独立分包可以依赖主包外，分包之间不能互相使用自定义组件或进行 require。「分包异步化」特性将允许通过一些配置和新的接口，使部分跨分包的内容可以等待下载后异步使用，从而一定程度上解决这个限制。

4.1 跨分包自定义组件引用

一个分包使用其他分包的自定义组件时，由于其他分包还未下载或注入，其他分包的组件处于不可用的状态。通过为其他分包的自定义组件设置 占位组件，我们可以先渲染占位组件作为替代，在分包下载完成后再进行替换。例如：

版权所有：腾讯云计算（北京）有限责任公司



```
// subPackageA/pages/index.json
{
  "usingComponents": {
    "button": "../../commonPackage/components/button",
    "list": "../../subPackageB/components/full-list",
    "simple-list": "../components/simple-list",
    "plugin-comp": "plugin://pluginInSubPackageB/comp"
  },
  "componentPlaceholder": {
    "button": "view",
    "list": "simple-list",
```

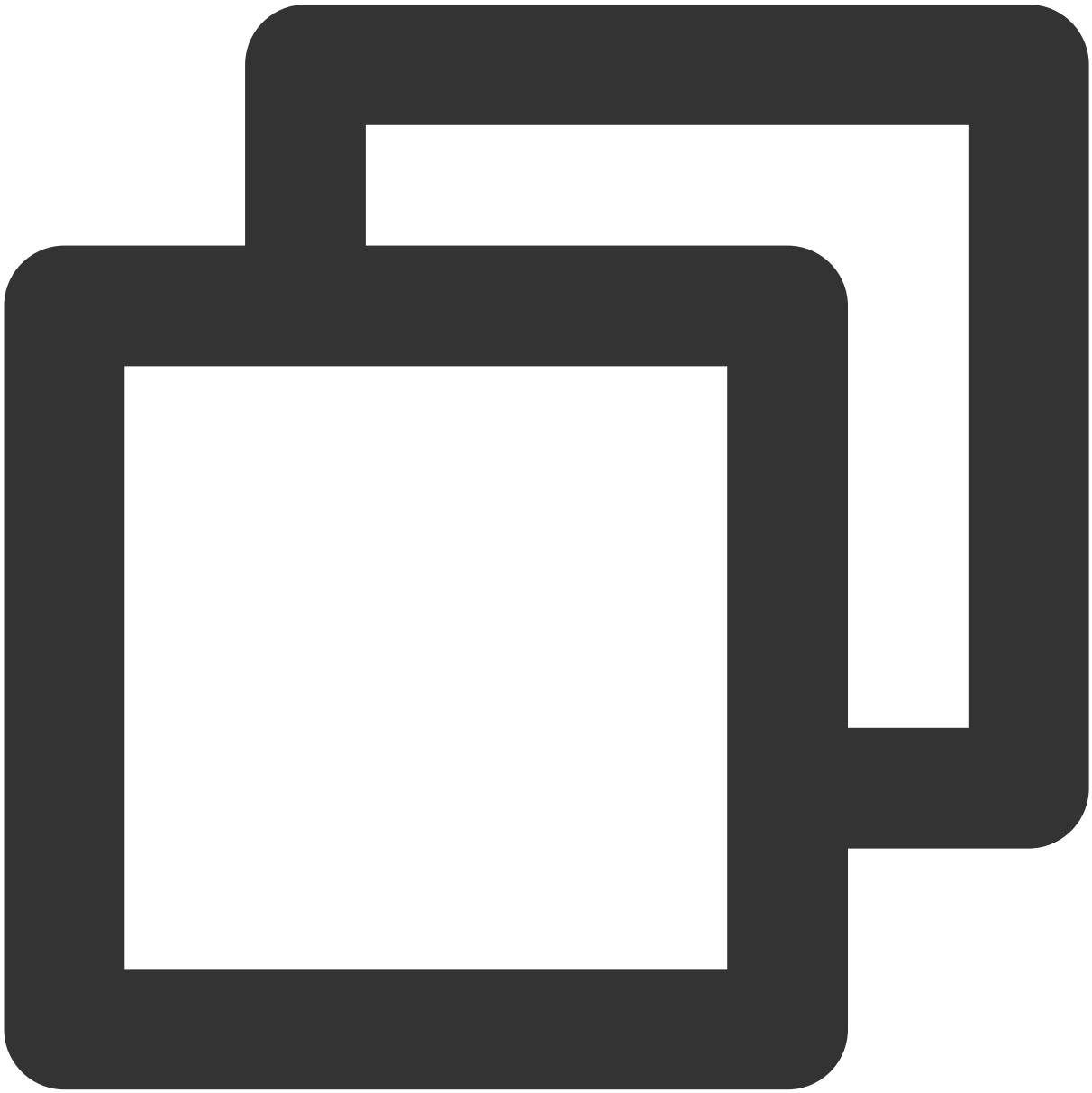
版权所有：腾讯云计算（北京）有限责任公司

```
      "plugin-comp": "view"
    }
  }
```

在这个配置中，`button` 和 `list` 两个自定义组件是跨分包引用组件，其中 `button` 在渲染时会使用内置组件 `view` 作为替代，list 会使用当前分包内的自定义组件 `simple-list` 作为替代进行渲染；在这两个分包下载完成后，占位组件就会被替换为对应的跨分包组件。

4.2 跨分包 JS 代码引用

一个分包中的代码引用其它分包的代码时，为了不让下载阻塞代码运行，我们需要异步获取引用的结果。如：



```
// subPackageA/index.js
// 使用回调函数风格的调用
require('../subPackageB/utils.js', utils => {
  console.log(utils.whoami) // Wechat MiniProgram
}, ({mod, errMsg}) => {
  console.error(`path: ${mod}, ${errMsg}`)
})
// 或者使用 Promise 风格的调用
require.async('../commonPackage/index.js').then(pkg => {
  pkg.getPackageName() // 'common'
}).catch(({mod, errMsg}) => {
  console.error(`path: ${mod}, ${errMsg}`)
})
```

详情可参考 模块化 - require 章节。

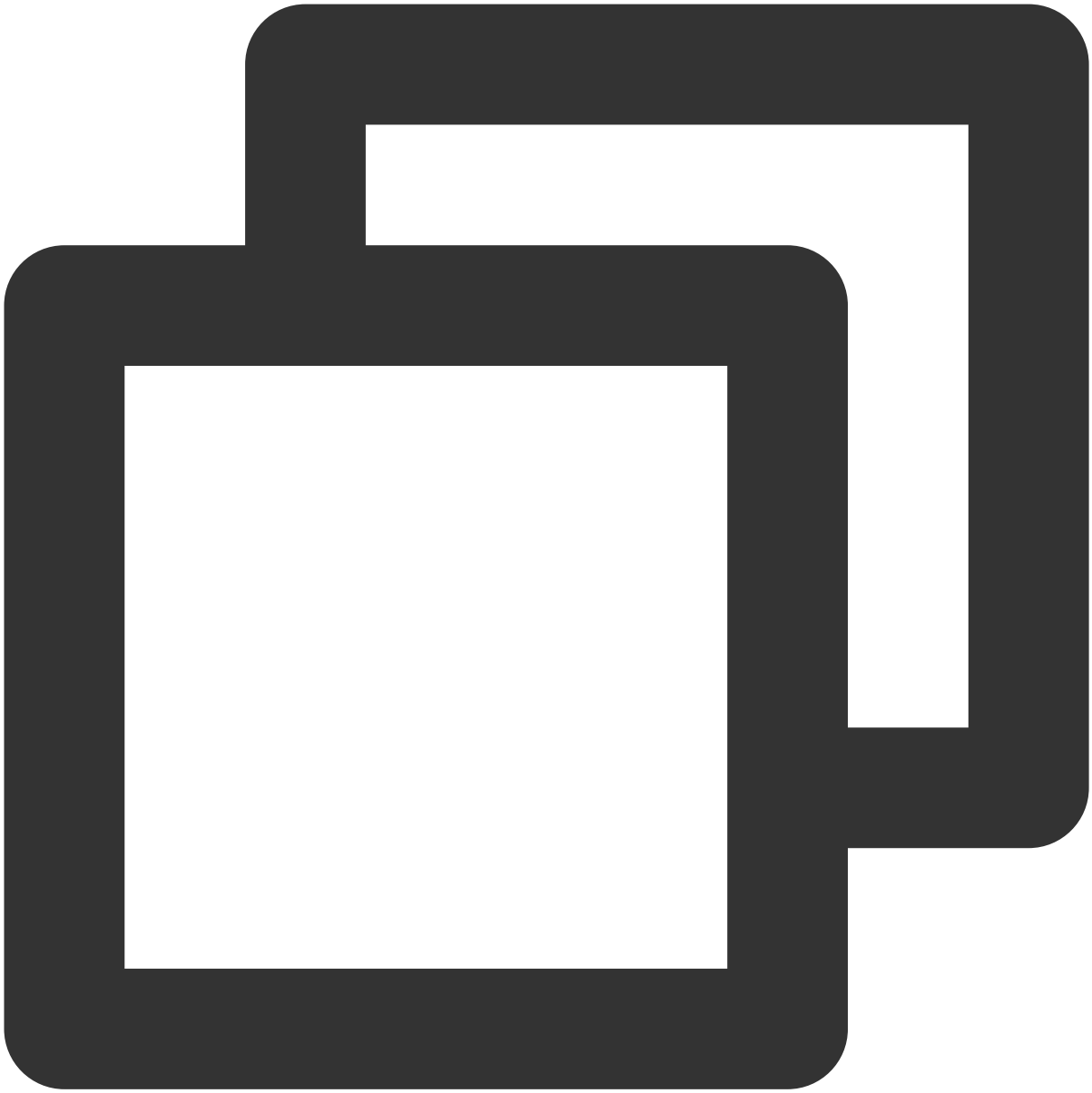
画布

最近更新时间：2024-07-12 17:37:59

Canvas 画布 提供了绘制界面，可以在之上进行任意绘制。

1. 基础使用

第一步：在 WXML 中添加 canvas 组件



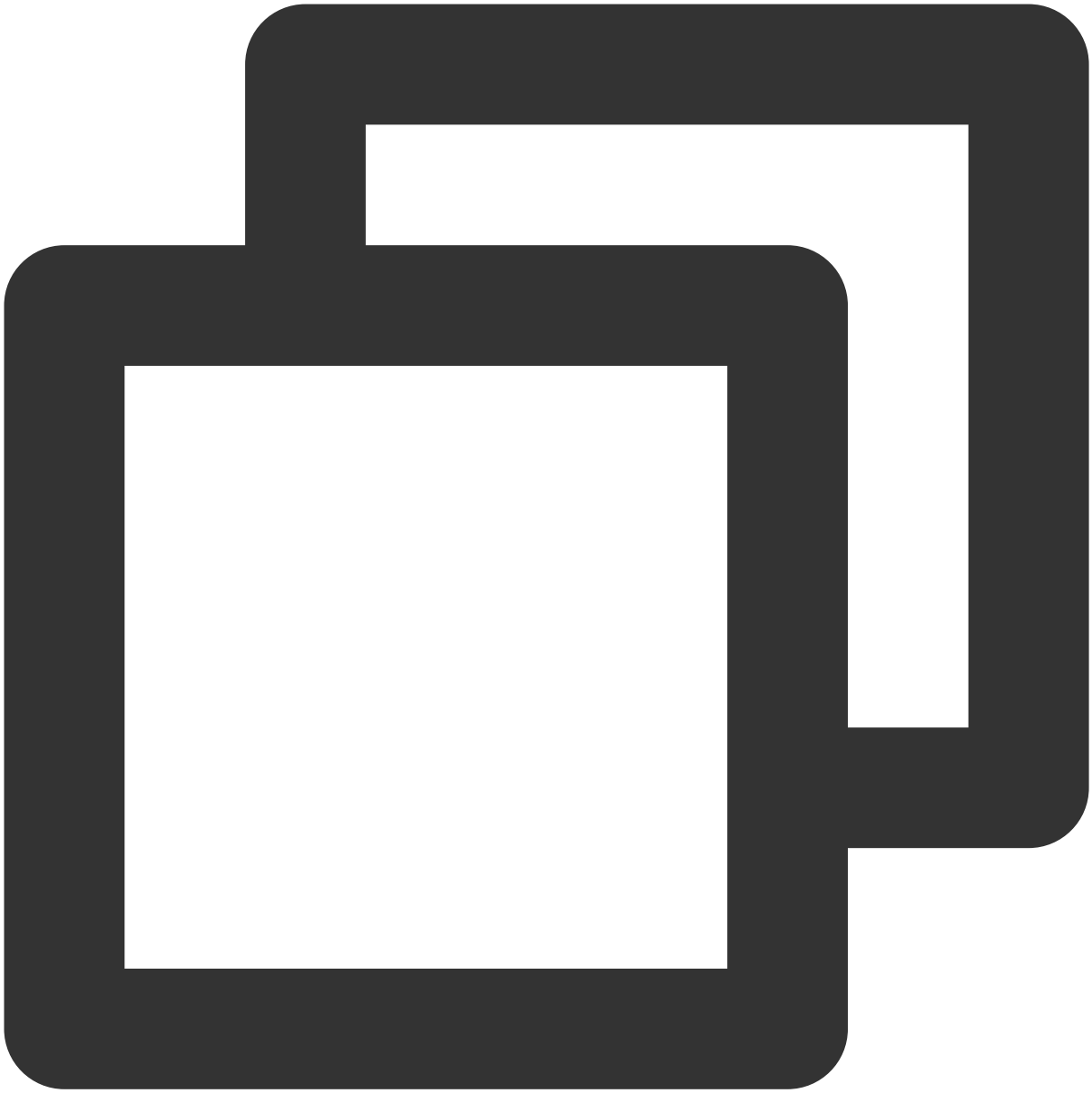
```
<!-- 2d 类型的 canvas -->
<canvas id="myCanvas" type="2d" style="border: 1px solid; width: 300px; height: 150
```

首先需要在 WXML 中添加 Canvas 画布。

指定 `id="myCanvas"` 唯一标识一个 canvas，用于后续获取 Canvas 对象。

指定 `type` 用于定义画布类型，本例子使用 `type="2d"` 示例。

第二步：获取 Canvas 对象和渲染上下文



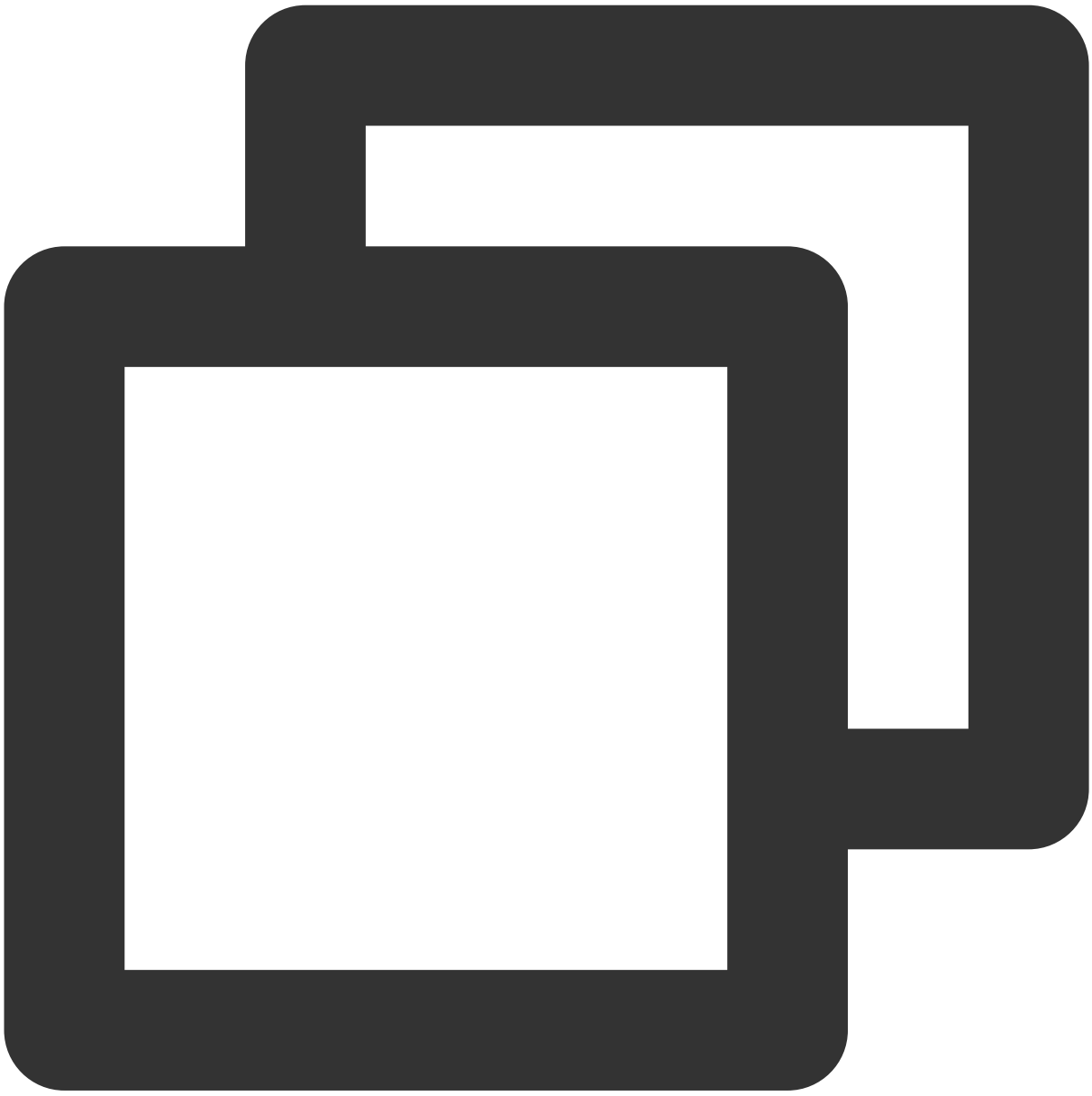
```
wx.createSelectorQuery()
    .select('#myCanvas') // 在 WXML 中填入的 id
    .fields({ node: true, size: true })
    .exec((res) => {
        // Canvas 对象
        const canvas = res[0].node
        // 渲染上下文
        const ctx = canvas.getContext('2d')
    })
```

通过 SelectorQuery 选择上一步的 canvas，可以获取到 Canvas。

版权所有：腾讯云计算（北京）有限责任公司

再通过 Canvas.getContext，我们可以获取到 渲染上下文 RenderingContext。
后续的画布操作与渲染操作，都需要通过这两个对象来实现。

第三步：初始化 Canvas



```
wx.createSelectorQuery()
    .select('#myCanvas') // 在 WXML 中填入的 id
    .fields({ node: true, size: true })
    .exec((res) => {
        // Canvas 对象
        const canvas = res[0].node
```

```
        // 渲染上下文
        const ctx = canvas.getContext('2d')


        // Canvas 画布的实际绘制宽高
        const width = res[0].width
        const height = res[0].height


        // 初始化画布大小
        const dpr = wx.getWindowInfo().pixelRatio
        canvas.width = width * dpr
        canvas.height = height * dpr
        ctx.scale(dpr, dpr)
    })
```

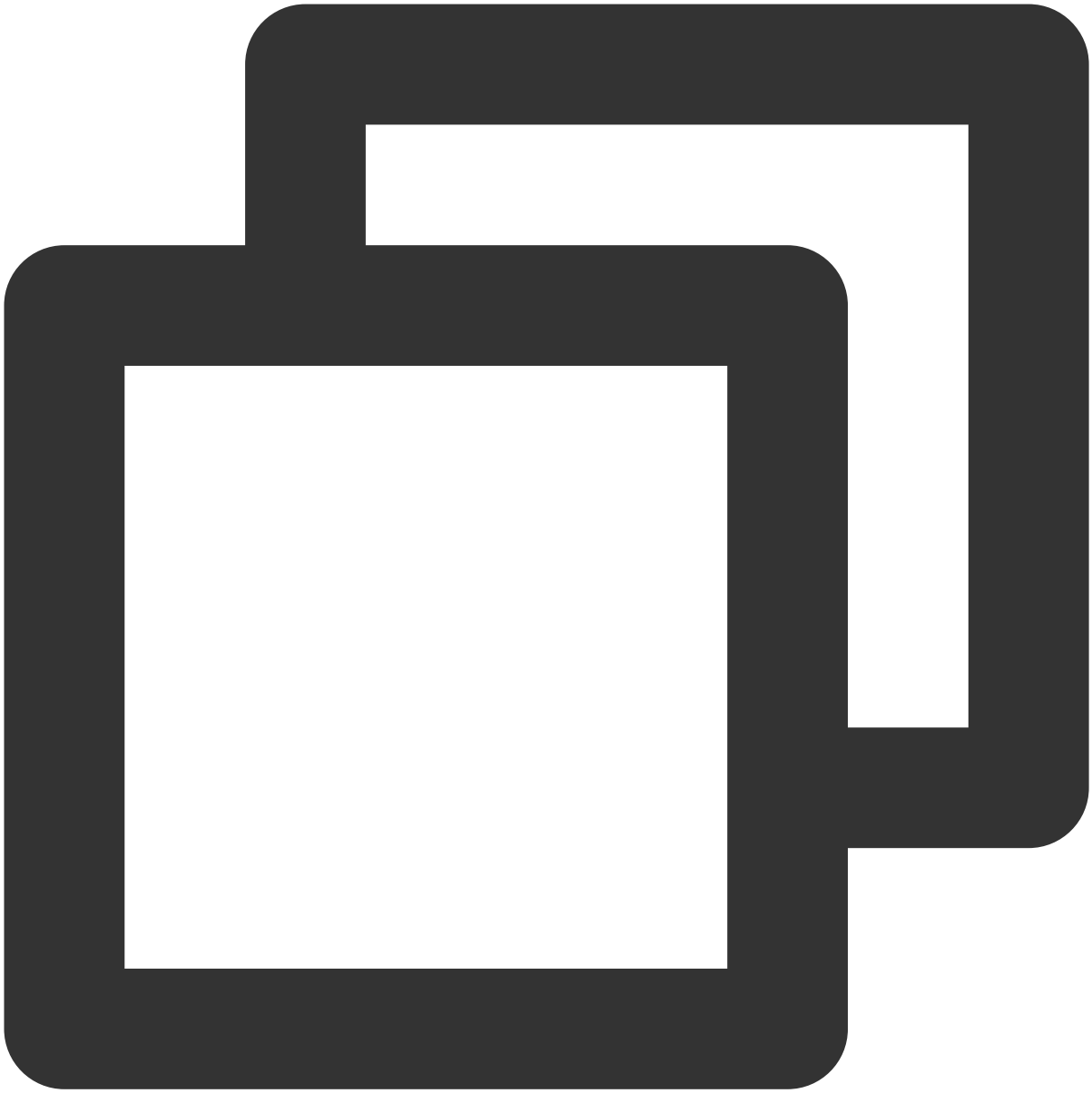
canvas 的宽高分为渲染宽高和逻辑宽高：

渲染宽高为 canvas 画布在页面中所实际占用的宽高大小，即通过对节点进行 boundingClientRect 请求获取到的大小。

逻辑宽高为 canvas 在渲染过程中的逻辑宽高大小，如绘制一个长方形与逻辑宽高相同，最终长方形会占满整个画布。逻辑宽高默认为 `300 * 150` 。

不同的设备上，存在物理像素和逻辑像素不相等的情况，所以一般我们需要用 wx.getWindowInfo 获取设备的像素比，乘上 canvas 的渲染大小，作为画布的逻辑大小。

第四步：进行绘制



```
// 省略上面初始化步骤，已经获取到 canvas 对象和 ctx 渲染上下文

// 清空画布
ctx.clearRect(0, 0, width, height)

// 绘制红色正方形
ctx.fillStyle = 'rgb(200, 0, 0)';
ctx.fillRect(10, 10, 50, 50);

// 绘制蓝色半透明正方形
ctx.fillStyle = 'rgba(0, 0, 200, 0.5)';
```

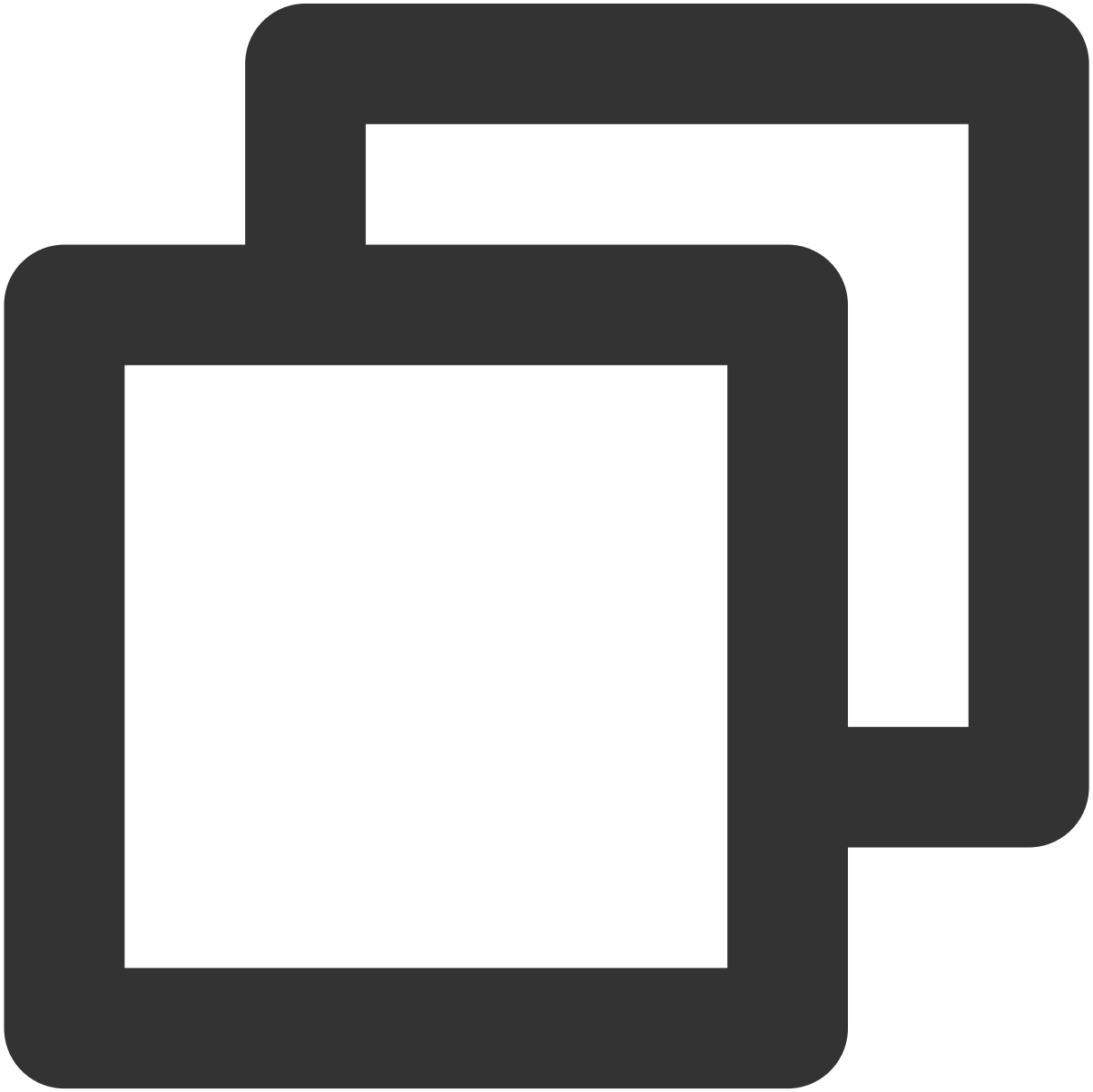
版权所有：腾讯云计算（北京）有限责任公司

```
ctx.fillRect(30, 30, 50, 50);
```

通过 渲染上下文 上的绘图 api，我们可以在画布上进行任意的绘制。

2. 进阶使用

2.1 绘制图片

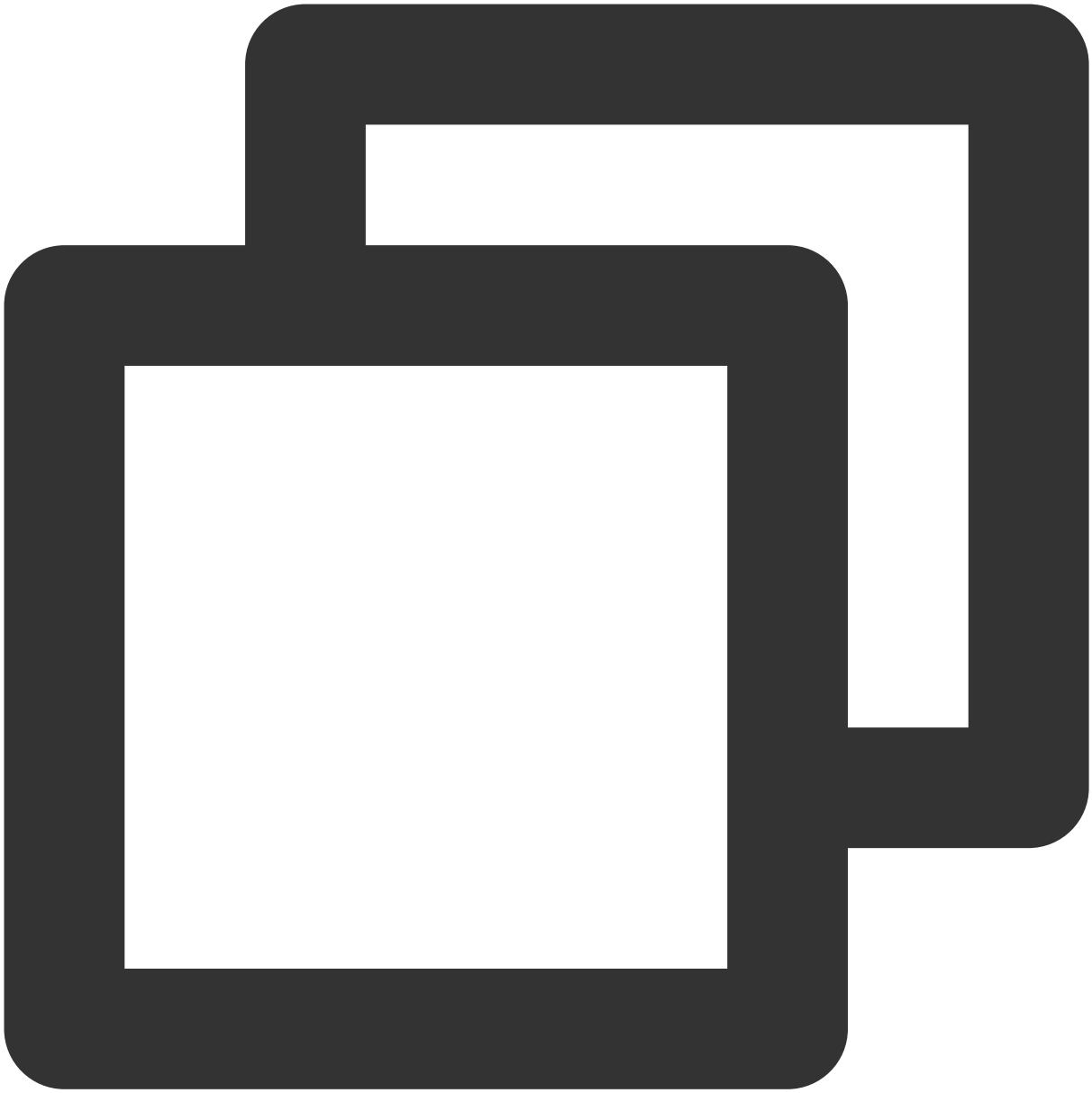


```
// 省略上面初始化步骤，已经获取到 canvas 对象和 ctx 渲染上下文
```

```
// 图片对象
const image = canvas.createImage()
// 图片加载完成回调
image.onload = () => {
    // 将图片绘制到 canvas 上
    ctx.drawImage(image, 0, 0)
}
// 设置图片src
image.src = 'https://open.weixin.qq.com/zh_CN/htmledition/res/assets/res-design-dow
```

通过 Canvas.createImage 我们可以创建图片对象并加载图片。当图片加载完成触发 onload 回调之后，使用 ctx.drawImage 即可将图片绘制到 canvas 上。

2.2 生成图片



```
// 省略上面初始化步骤，已经获取到 canvas 对象和 ctx 渲染上下文

// 绘制红色正方形
ctx.fillStyle = 'rgb(200, 0, 0)';
ctx.fillRect(10, 10, 50, 50);

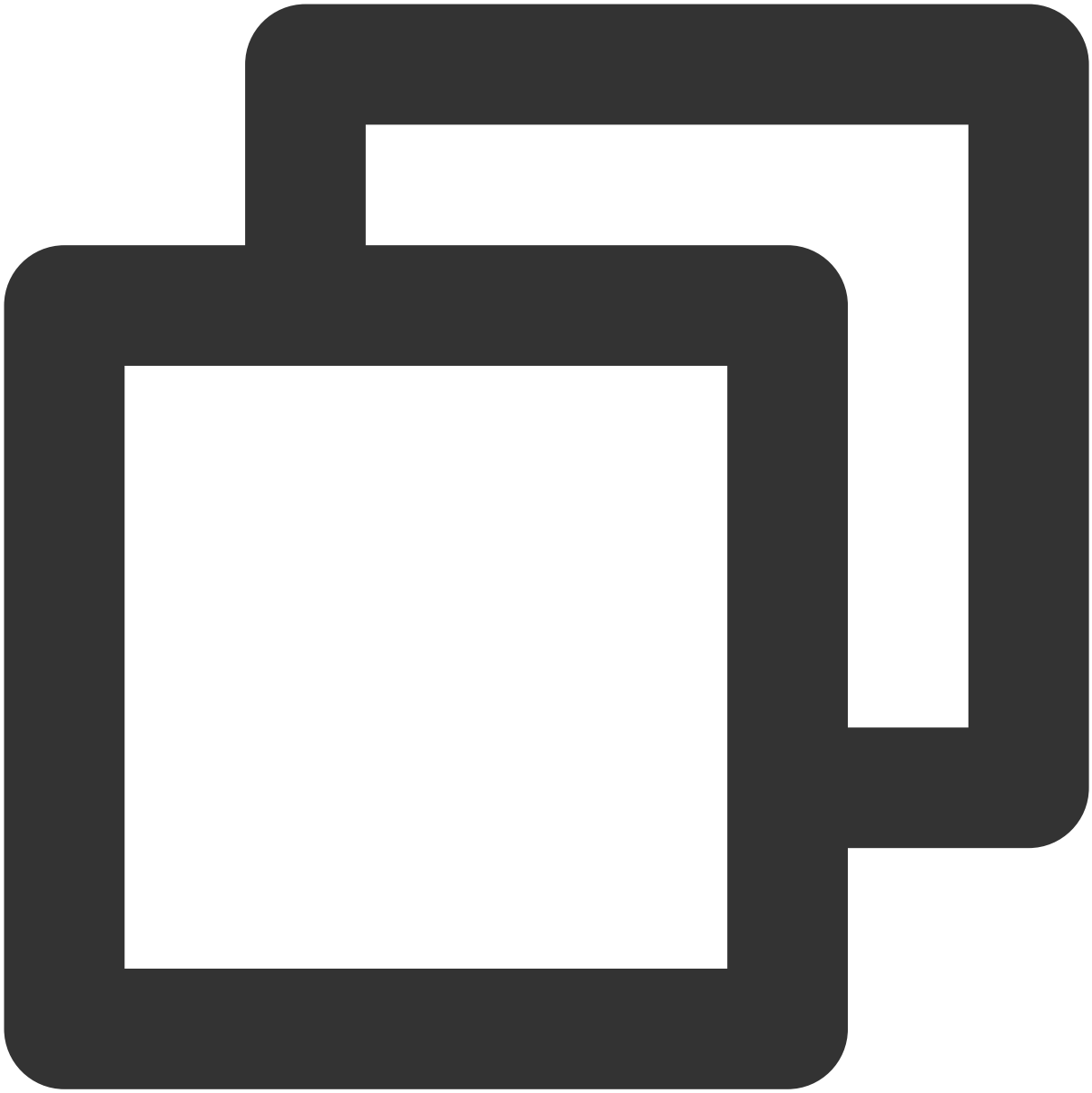
// 绘制蓝色半透明正方形
ctx.fillStyle = 'rgba(0, 0, 200, 0.5)';
ctx.fillRect(30, 30, 50, 50);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 生成图片
wx.canvasToTempFilePath({
    canvas,
    success: res => {
        // 生成的图片临时文件路径
        const tempFilePath = res.tempFilePath
    },
})
```

通过 wx.canvasToTempFilePath 接口，可以将 canvas 上的内容生成图片临时文件。

2.3 帧动画



```
// 省略上面初始化步骤，已经获取到 canvas 对象和 ctx 渲染上下文

const startTime = Date.now()

// 帧渲染回调
const draw = () => {
  const time = Date.now()
  // 计算经过的时间
  const elapsed = time - startTime
```

```
  // 计算动画位置
  const n = Math.floor(elapsed / 3000)
  const m = elapsed % 3000
  const dx = (n % 2 ? 0 : 1) + (n % 2 ? 1 : -1) * (m < 2500 ? easeOutBounce(m / 250
  const x = (width - 50) * dx


  // 渲染
  ctx.clearRect(0, 0, width, height)
  ctx.fillStyle = 'rgb(200, 0, 0)';
  ctx.fillRect(x, height / 2 - 25, 50, 50);


  // 注册下一帧渲染
  canvas.requestAnimationFrame(draw)
}

draw()
```

通过 Canvas.requestAnimationFrame 可以注册动画帧回调，在回调内进行动画的逐帧绘制。

2.4 自定义字体

通过 wx.loadFontFace 可以为 Canvas 加载自定义字体。

自定义 tabBar

最近更新时间：2024-07-12 17:24:10

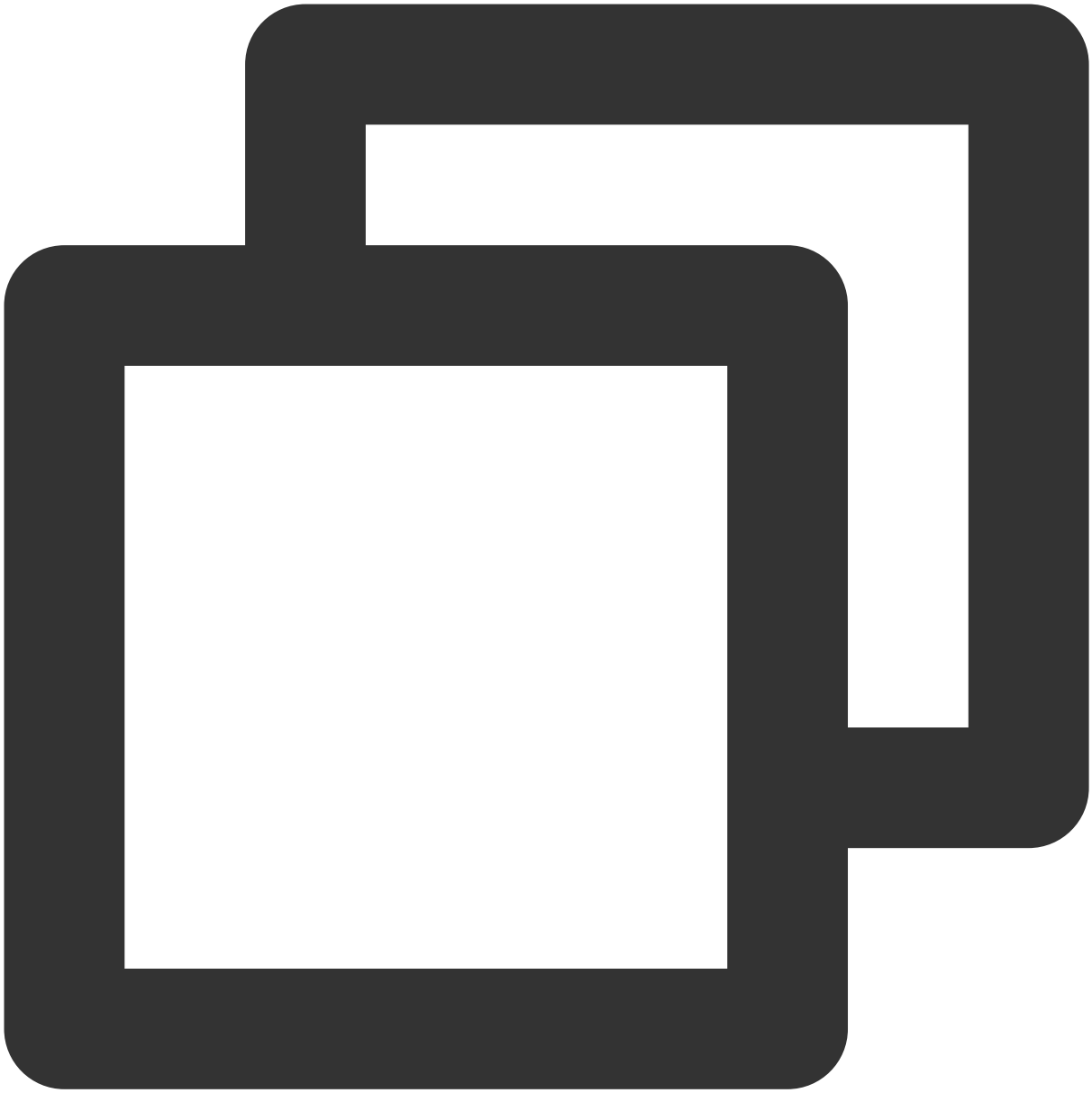
自定义 tabBar 可以让开发者更加灵活地设置 tabBar 样式，以满足更多个性化的场景。
在自定义 tabBar 模式下
为了保证低版本兼容以及区分哪些页面是 tab 页，tabBar 的相关配置项需完整声明，但这些字段不会作用于自定义 tabBar 的渲染。
此时需要开发者提供一个自定义组件来渲染 tabBar，所有 tabBar 的样式都由该自定义组件渲染。推荐用 fixed 在底部的 cover-view + cover-image 组件渲染样式，以保证 tabBar 层级相对较高。
与 tabBar 样式相关的接口，如 wx.setTabBarItem 等将失效。
每个 tab 页下的自定义 tabBar 组件实例是不同的，可通过自定义组件下的 getTabBar 接口，获取当前页面的自定义 tabBar 组件实例。
注意：
如需实现 tab 选中态，要在当前页面下，通过 `getTabBar` 接口获取组件实例，并调用 setData 更新选中态。可参考底部的代码示例。

使用流程

配置信息

在 `app.json` 中的 `tabBar` 项指定 `custom` 字段，同时其余 `tabBar` 相关配置也补充完整；
所有 tab 页的 json 里需声明 `usingComponents` 项，也可以在 `app.json` 全局开启。
代码示例：

版权所有：腾讯云计算（北京）有限责任公司

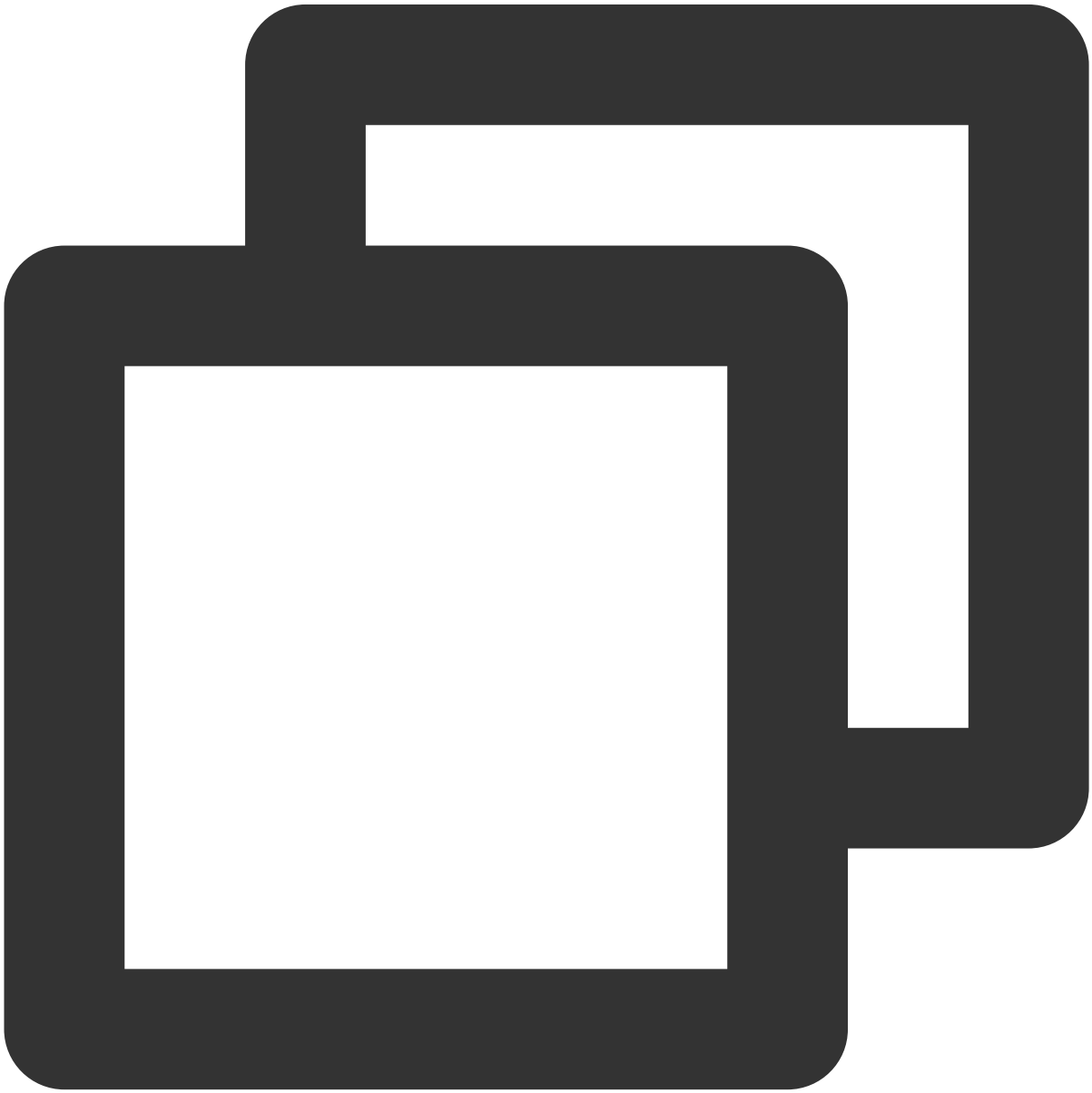


```
{
  "tabBar": {
    "custom": true,
    "color": "#000000",
    "selectedColor": "#000000",
    "backgroundColor": "#000000",
    "list": [{
      "pagePath": "page/component/index",
      "text": "组件"
    }, {
      "pagePath": "page/API/index",
```

```
      "text": "接口"
    }]
  },
  "usingComponents": {}
}
```

添加 tabBar 代码文件

在代码根目录下添加入口文件:

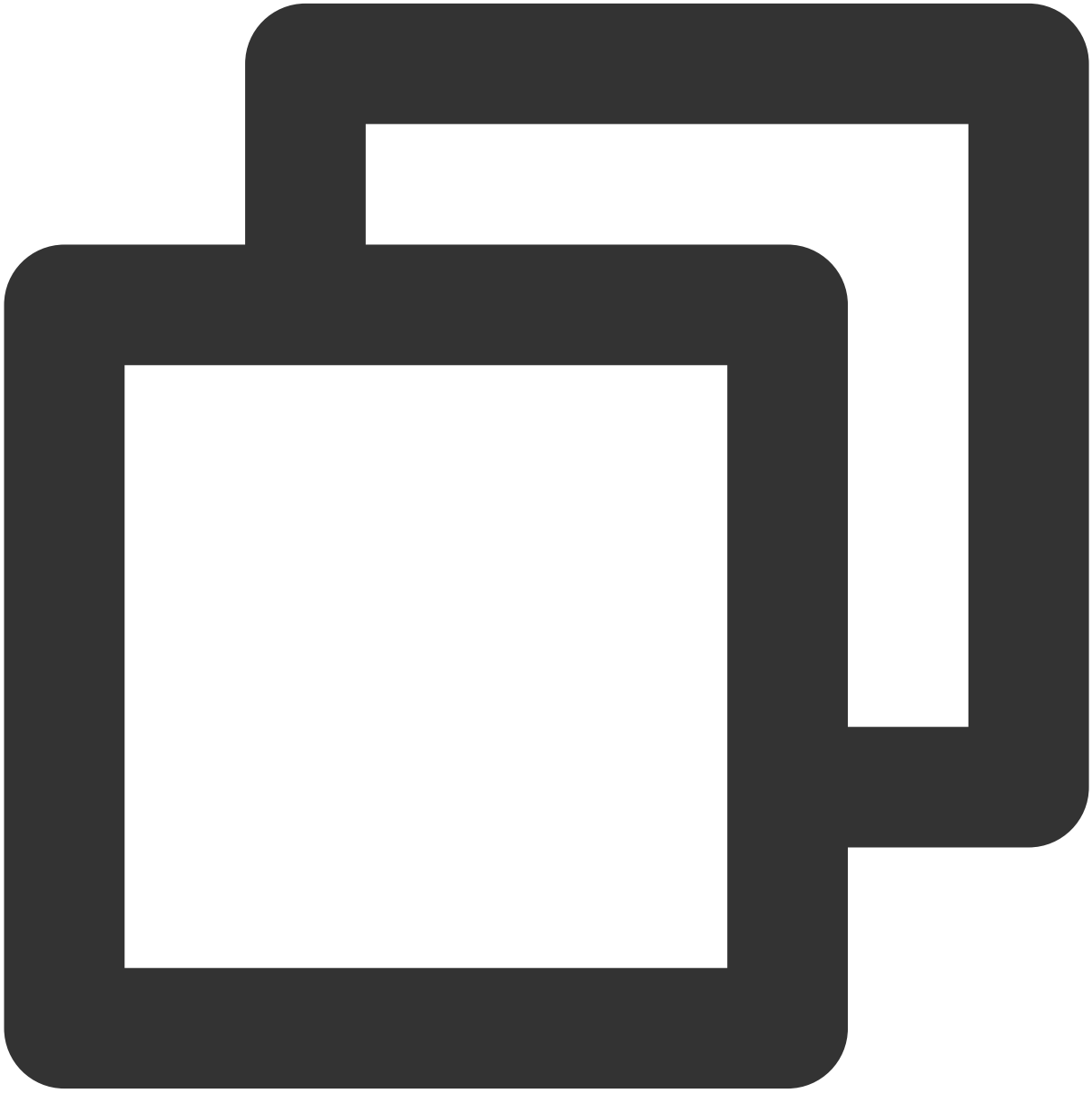


```
custom-tab-bar/index.js
```

```
custom-tab-bar/index.json
custom-tab-bar/index.wxml
custom-tab-bar/index.wxss
```

编写 tabBar 代码

用自定义组件的方式编写即可，该自定义组件完全接管 tabBar 的渲染。另外，自定义组件新增 `getTabBar` 接口，可获取当前页面下的自定义 tabBar 组件实例。



```
Component({
  pageLifetimes: {
```

```
    show() {
      if (typeof this.getTabBar === 'function' &&
        this.getTabBar()) {
        this.getTabBar().setData({
          selected: 0
        })
      }
    }
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

多进程 Worker

最近更新时间：2024-07-12 18:29:16

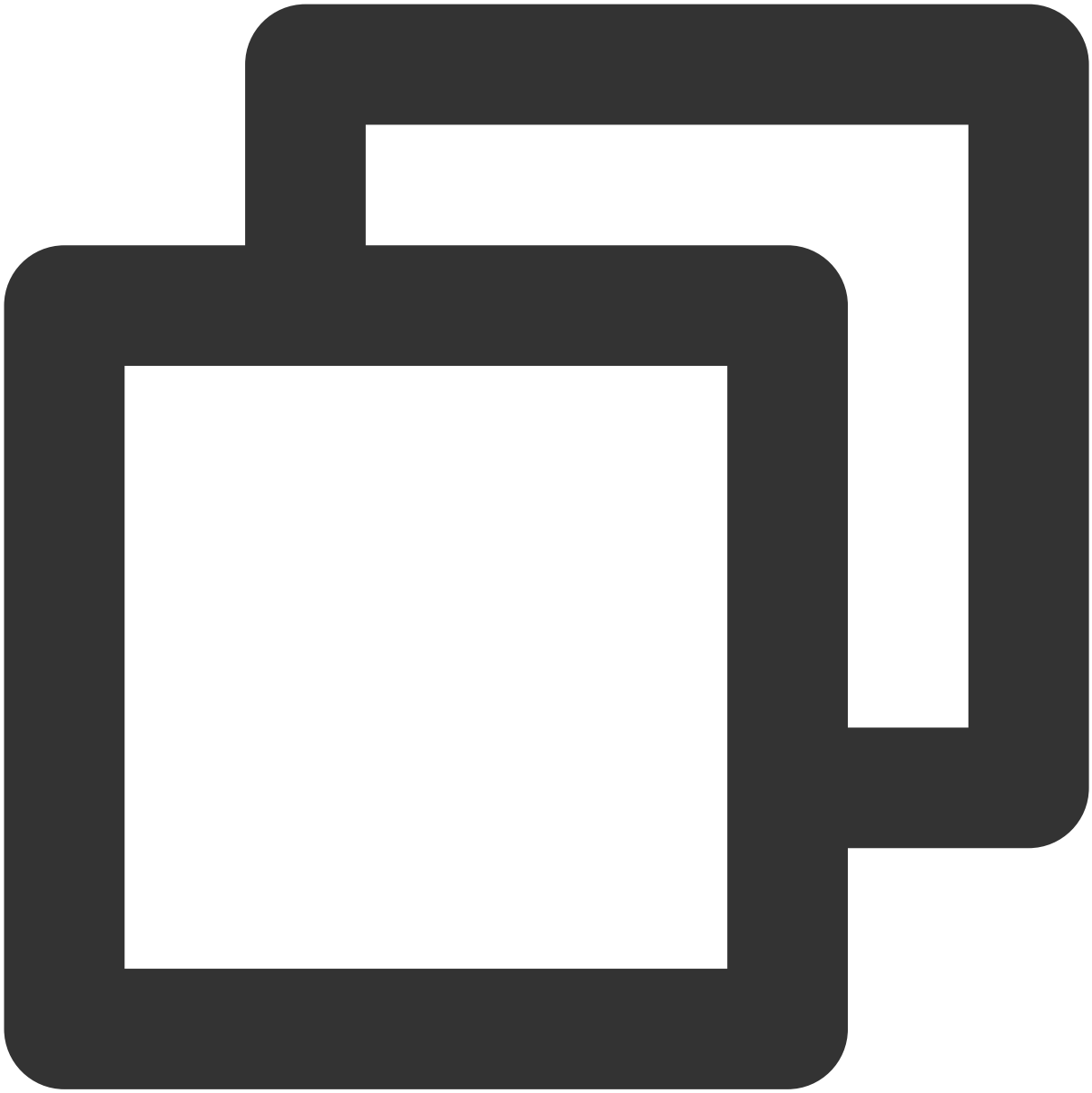
一些异步处理的任务，可以放置于 Worker 中运行，待运行结束后，再把结果返回到小程序主线程。Worker 运行于一个单独的全局上下文与线程中，不能直接调用主线程的方法。
Worker 与主线程之间的数据传输，双方使用 Worker.postMessage() 来发送数据，Worker.onMessage() 来接收数据，传输的数据并不是直接共享，而是被复制的。

使用流程

配置 Worker 信息

在 `app.json` 中可配置 Worker 代码放置的目录，目录下的所有 JS 代码最终将被打包成一个 JS 文件：
配置示例：

版权所有：腾讯云计算（北京）有限责任公司

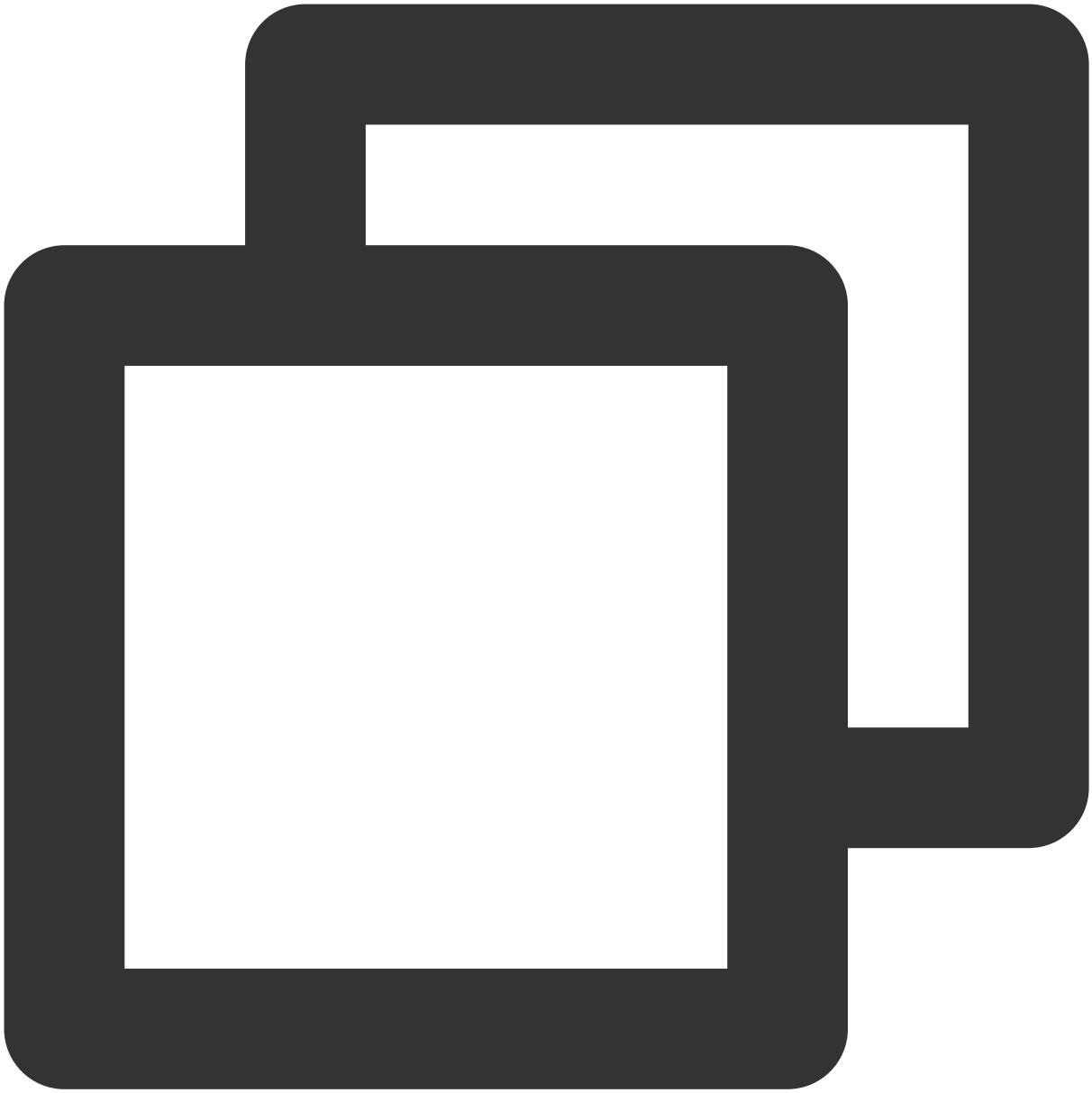


```
{
  "workers": "workers"
}
```

通过以上方式配置，workers 目录下的所有 JS 文件会被打包为一个 JS 文件，并作为小程序首包的一部分。

添加 Worker 代码文件

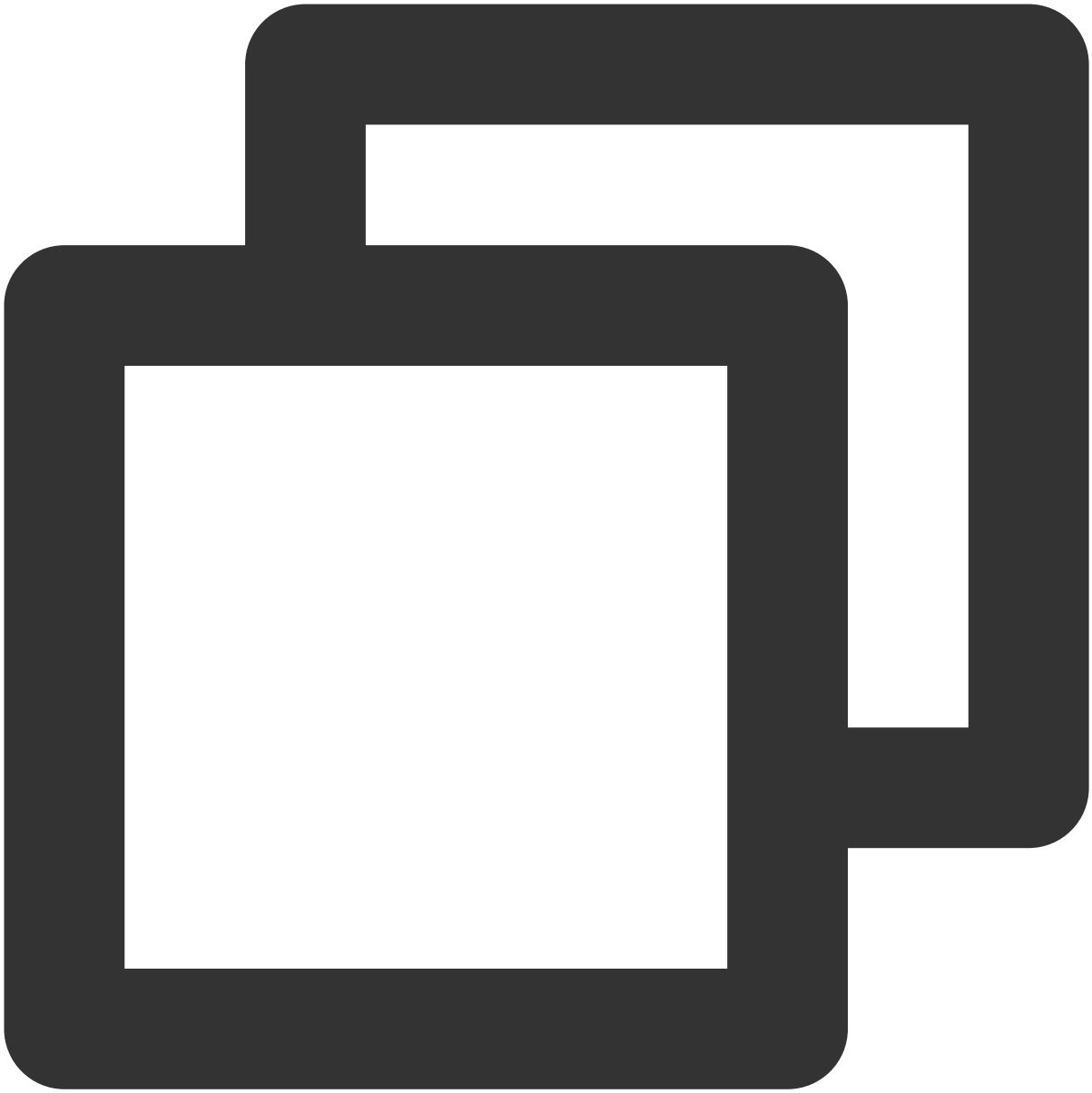
根据步骤1中的配置，在代码目录下新建以下两个入口文件：



```
workers/request/index.js
workers/request/utils.js
workers/response/index.js
```

添加后，目录结构如下：

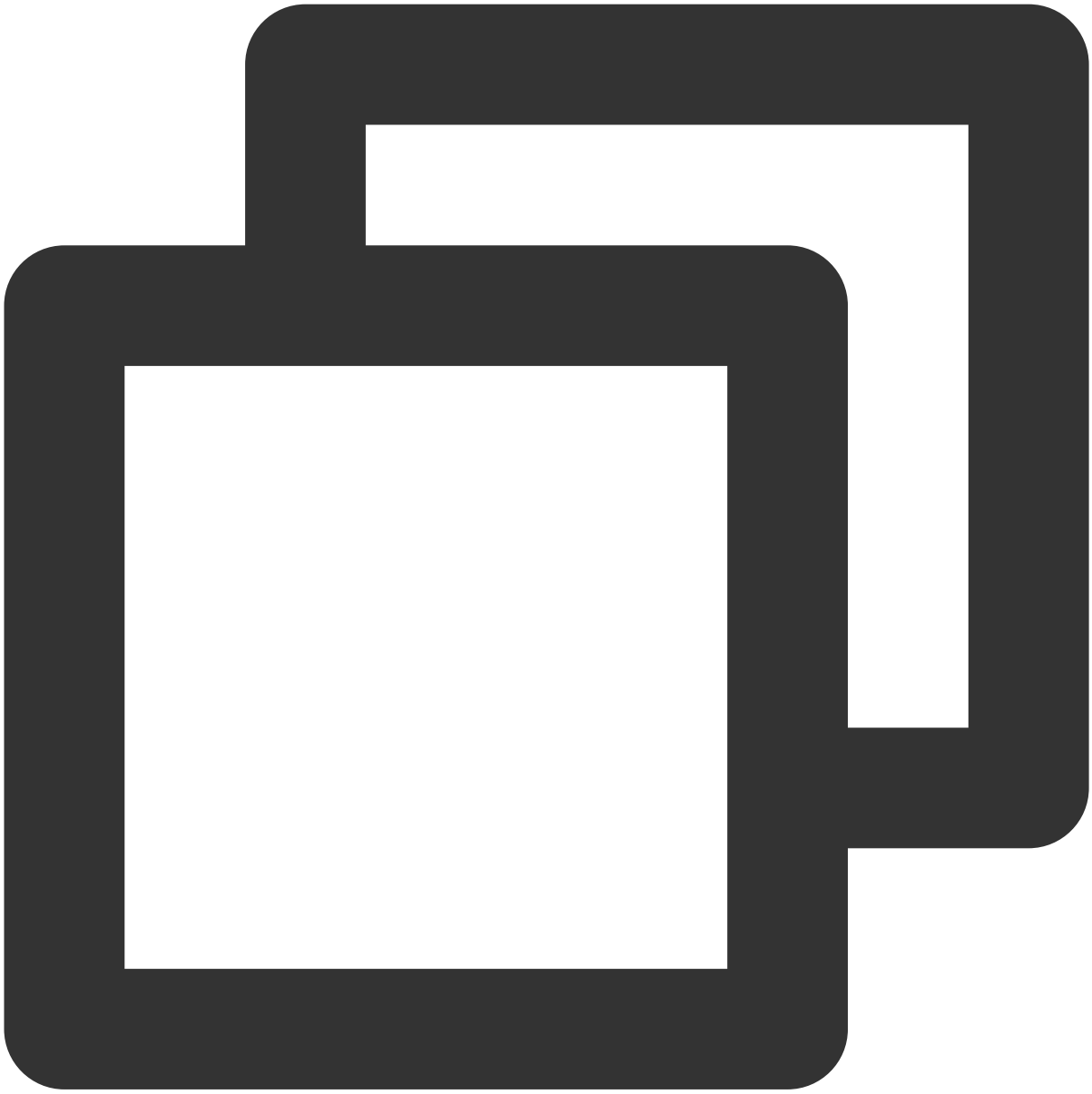
版权所有：腾讯云计算（北京）有限责任公司



```
├── app.js
├── app.json
├── project.config.json
└── workers
    ├── request
    │   ├── index.js
    │   └── utils.js
    └── response
        └── index.js
```

编写 Worker 代码

在 `workers/request/index.js` 编写 Worker 响应代码

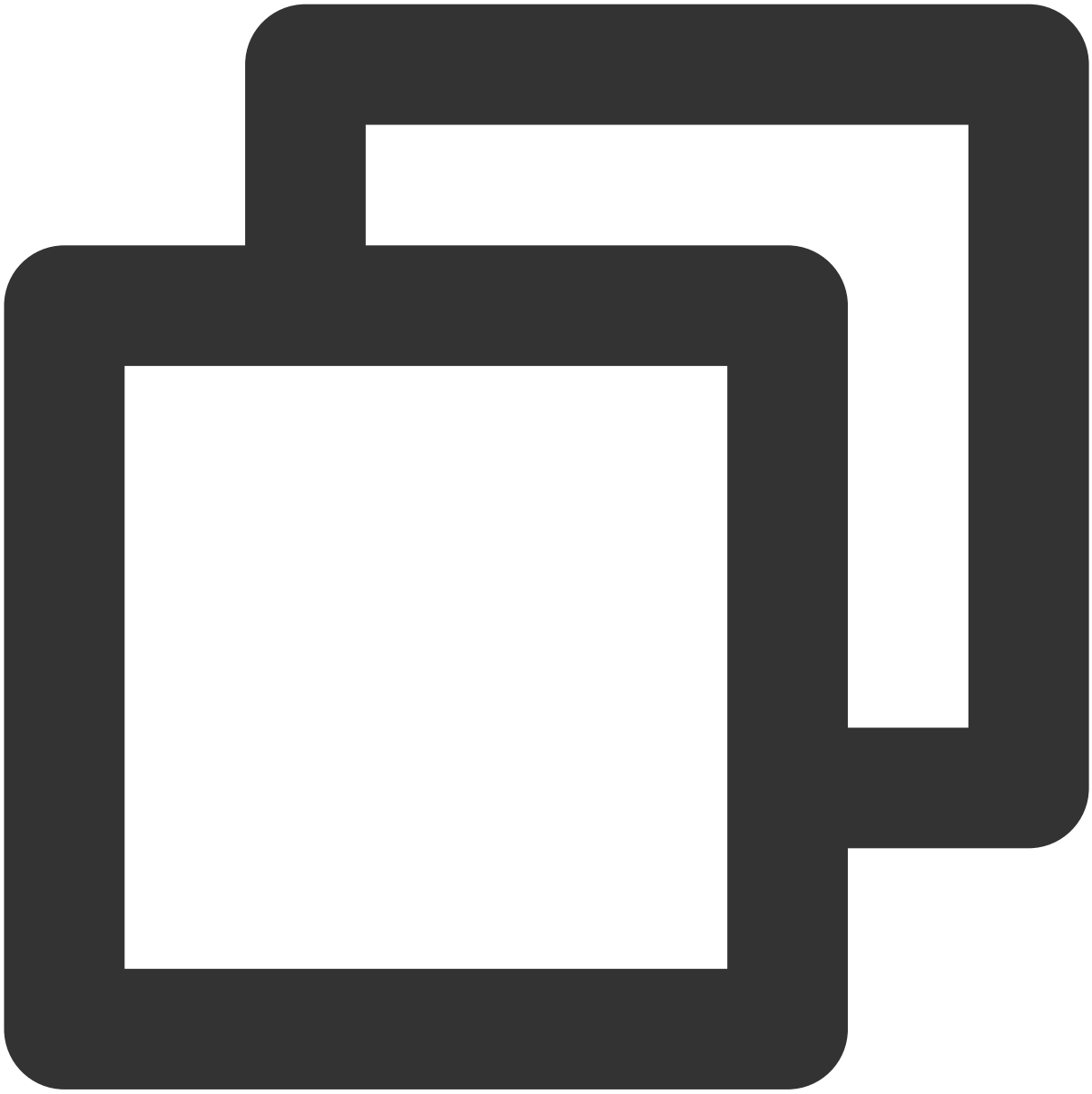


```
const utils = require('./utils')

// 在 Worker 线程执行上下文会全局暴露一个 worker 对象，直接调用 worker.onMessage/postMessa
worker.onMessage(function (res) {
  console.log(res)
})
```

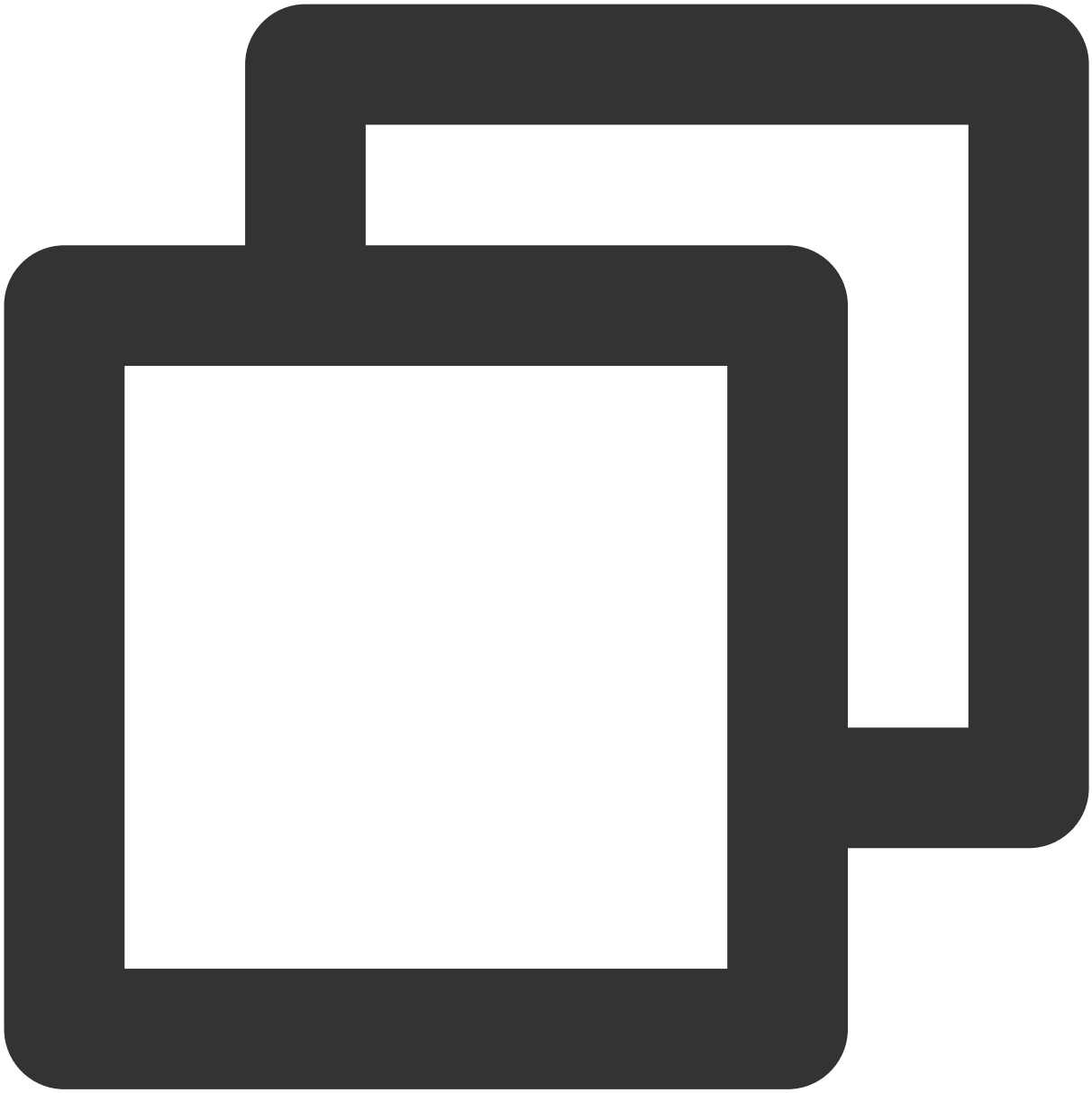
在主线程中初始化 Worker

在主线程的代码 app.js 中初始化 Worker



```
const worker = wx.createWorker('workers/request/index.js') // 文件名指定 worker 的入口
```

主线程向 Worker 发送消息



```
worker.postMessage({
  msg: 'hello worker'
})
```

注意事项

Worker 最大并发数量限制为 1 个，创建下一个前请用 Worker.terminate() 结束当前 Worker；
Worker 内代码只能 require 指定 Worker 路径内的文件，无法引用其它路径；

版权所有：腾讯云计算（北京）有限责任公司

Worker 的入口文件由 wx.createWorker() 时指定，开发者可动态指定 Worker 入口文件；

Worker 内不支持 `wx` 系列的 API；

Workers 之间不支持发送消息；

Worker 目录内只支持放置 JS 文件，其他类型的静态文件需要放在 Worker 目录外。

版权所有：腾讯云计算（北京）有限责任公司

DarkMode Adaptation Guide

最近更新时间：2024-07-12 17:18:38

1. Enable DarkMode

Configure darkmode as true in `app.json`, that means the current mini program has been adapted to DarkMode, all the basic components will display different default styles according to the system theme, and the navigation bar and tab bar will be automatically switched according to the following configuration.

2. Related Configuration

When `app.json` is configured with `darkmode` as `true`, some of the configuration items of mini program can be configured in the form of variables, defining the colour or icon under different themes in the variable configuration file, as follows:

1. Configure `themeLocation` in app.json to specify the path of variable configuration file theme.json, for example: add `theme.json` in the root directory, you need to configure `"themeLocation": "theme.json"`;
2. Define the relevant variables in theme.json;
3. Referencing variables in app.json starting with `@`;

Properties configured via variables are supported.

Globally configured window properties and properties under page configuration

navigationBarBackgroundColor;

navigationBarTextStyle;

backgroundColor;

backgroundTextStyle;

backgroundColourTop;

backgroundColorBottom.

Global configuration of window.tabBar properties

colour;

selectedColor;

backgroundColor;

borderStyle;

list;

iconPath;

selectedIconPath.

版权所有：腾讯云计算（北京）有限责任公司

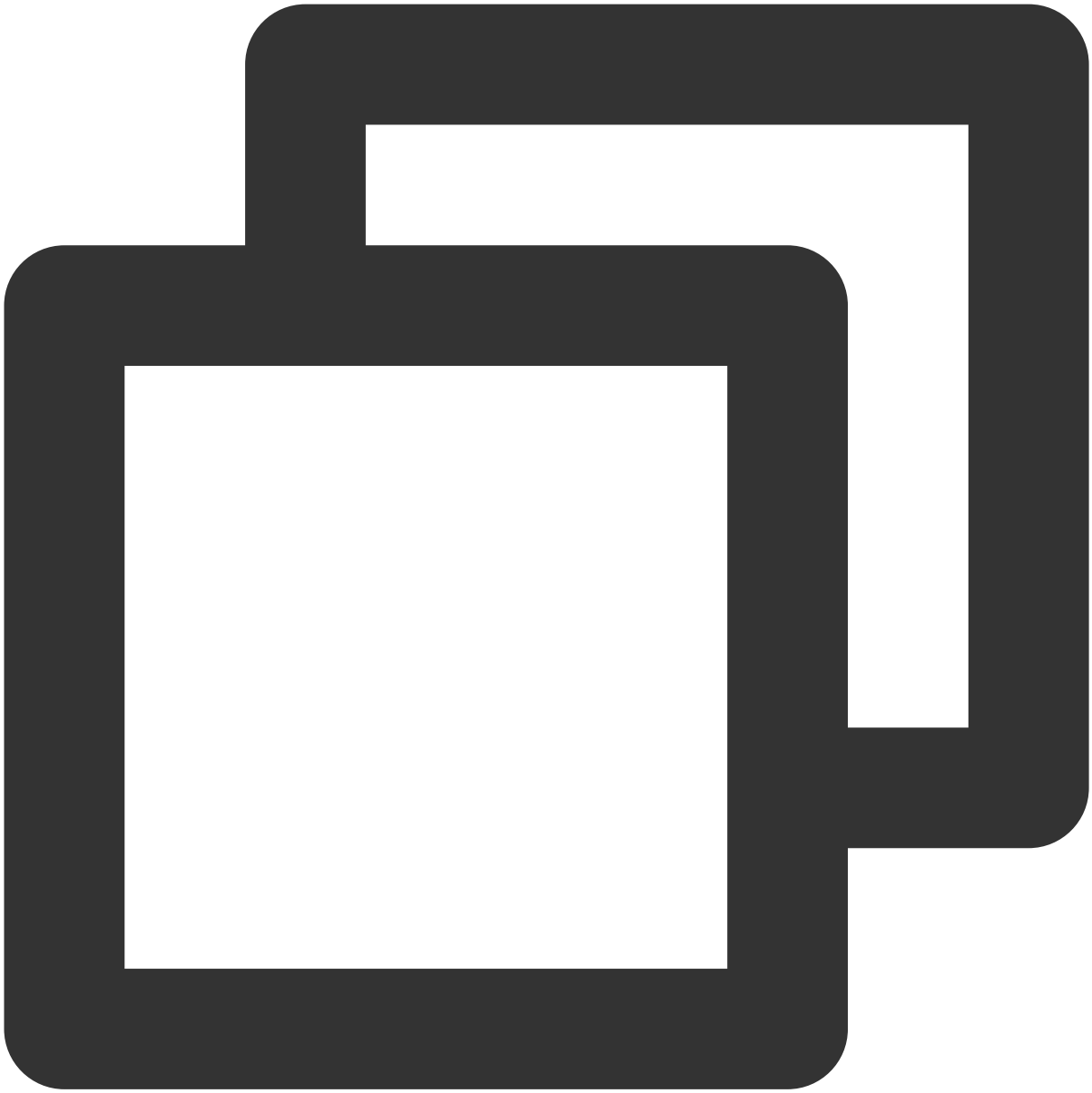
2.1 Variable configuration file theme.json

`theme.json` is used to define the variables related to the colour theme, you need to configure the path of `theme.json` in `themeLocation` first, otherwise you can't read the variable configuration.

The configuration file must contain the following properties:

Attributes	Type	Required	Descripción
light	object	yes	Definition of the variable in light mode
dark	object	yes	Definition of the variable in dark mode

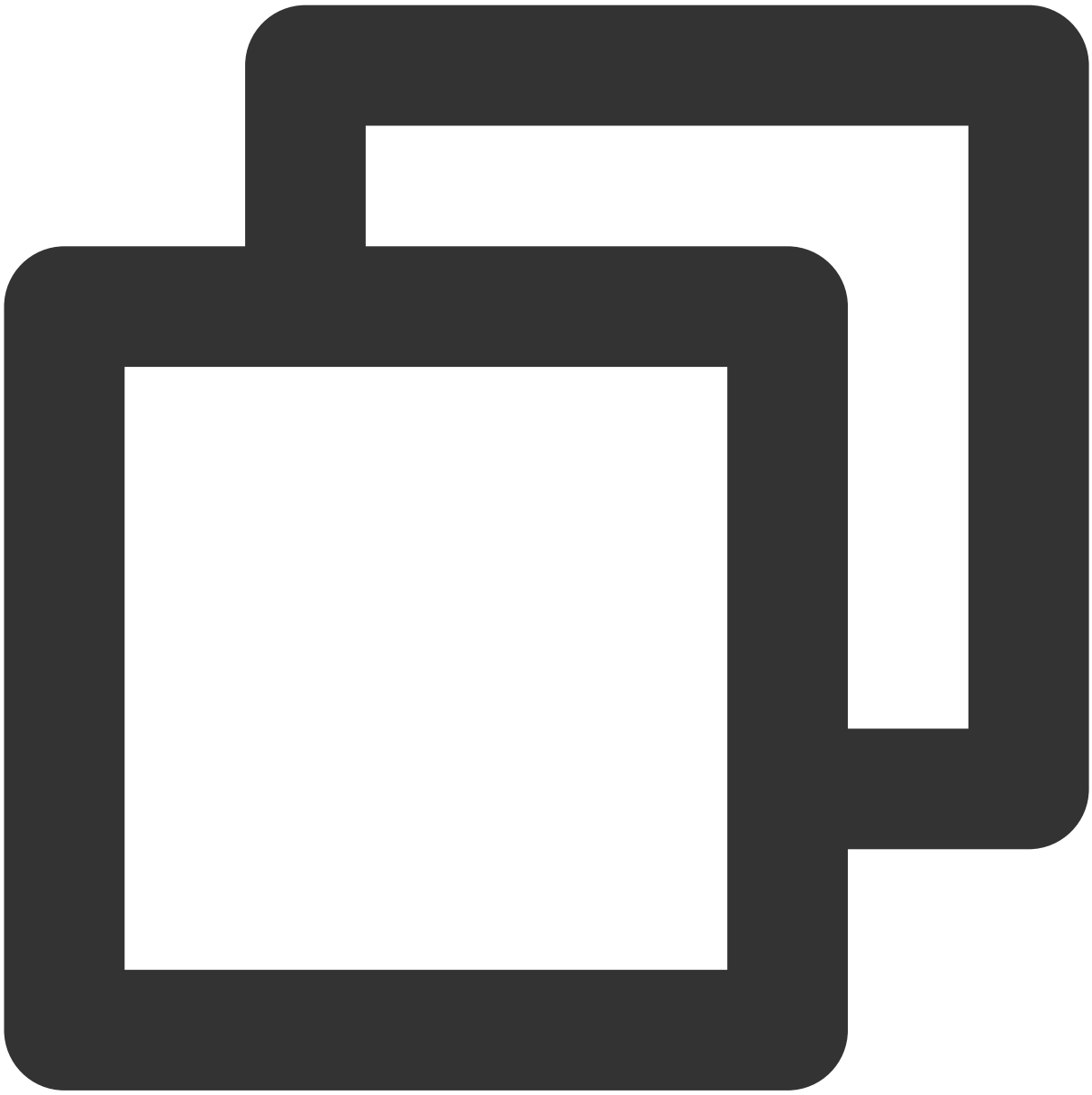
Both `light` and `dark` can define variable names and values as key: value, for example:



```
{
  "light": {
    "navBgColor": "#f6f6f6",
    "navTxtStyle": "black"
  },
  "dark": {
    "navBgColor": "#191919",
    "navTxtStyle": "white"
  }
}
```

版权所有：腾讯云计算（北京）有限责任公司

After completing the definition, it can be referenced in the global configuration or in the relevant properties of the page configuration starting with `@` , for example:



```
// Global Configuration
{
  "window": {
    "navigationBarBackgroundColor": "@navBgColor",
    "navigationBarTextStyle": "@navTxtStyle"
  }
}
```

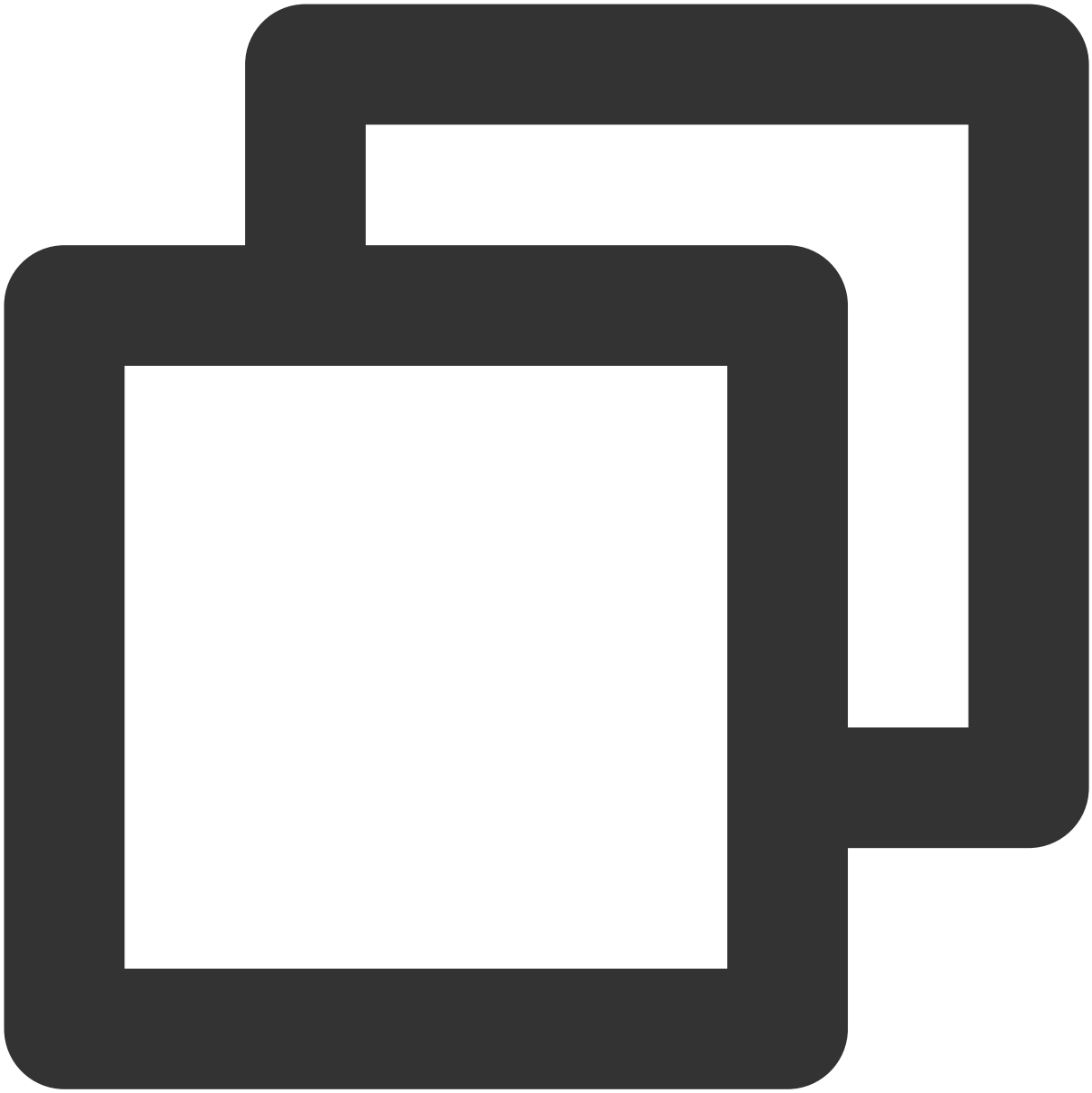
版权所有：腾讯云计算（北京）有限责任公司

```
// Page Configuration
{
  "navigationBarBackgroundColor": "@navBgColor",
  "navigationBarTextStyle": "@navTxtStyle"
}
```

After the configuration is completed, the mini program framework will automatically display the colour under the corresponding theme for the mini program according to the system theme.

2.2 Configuration Example

app.json (the example omits configuration items other than those related to the theme)

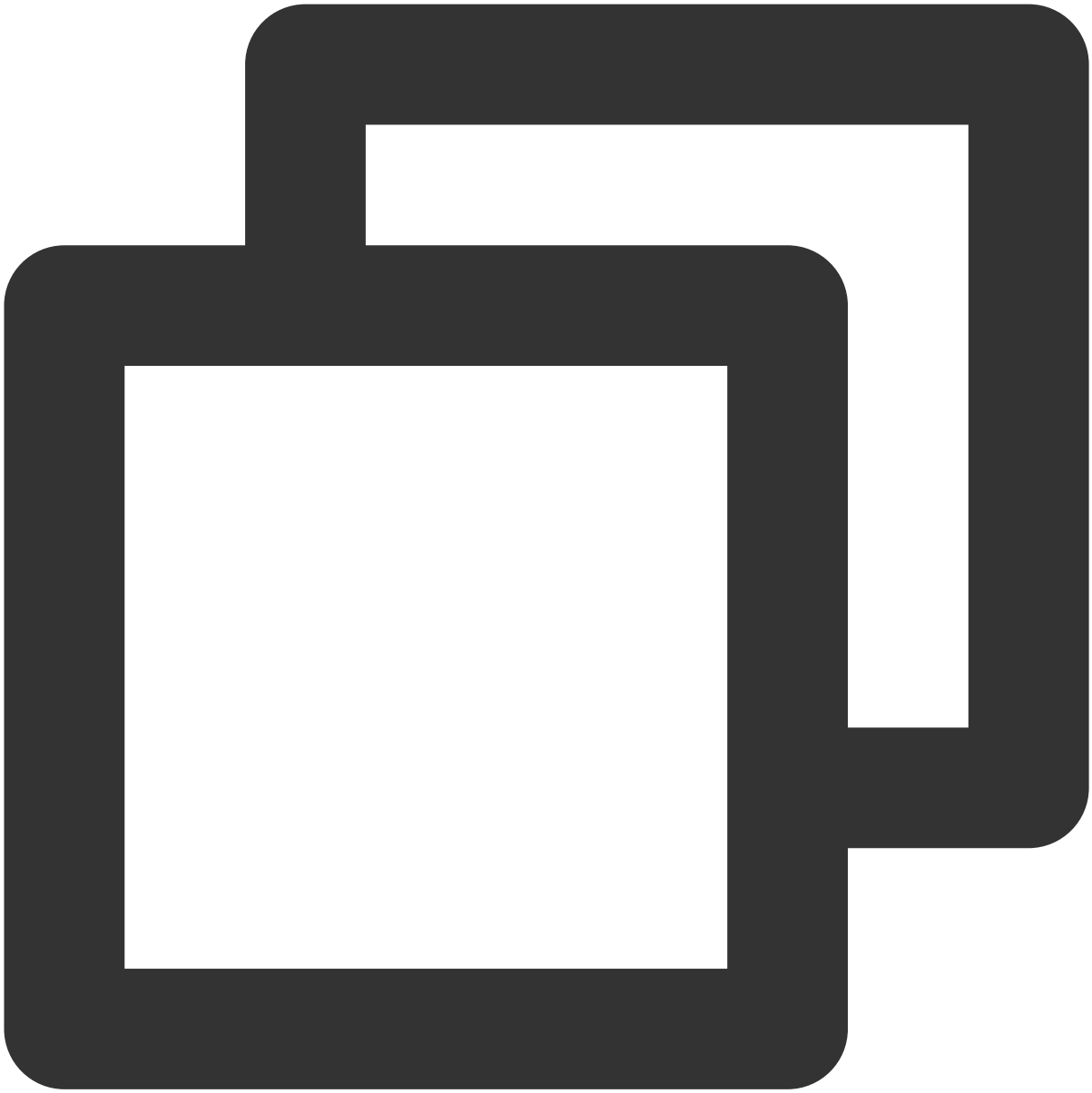


```
{
  "window": {
      "navigationBarBackgroundColor": "@navBgColor",
      "navigationBarTextStyle": "@navTxtStyle",
      "backgroundColor": "@bgColor",
      "backgroundTextStyle": "@bgTxtStyle",
      "backgroundColorTop": "@bgColorTop",
      "backgroundColorBottom": "@bgColorBottom"
  },
  "tabBar": {
      "color": "@tabFontColor",
```

版权所有：腾讯云计算（北京）有限责任公司

```
        "selectedColor": "@tabSelectedColor",
        "backgroundColor": "@tabBgColor",
        "borderStyle": "@tabBorderStyle",
        "list": [{
            "iconPath": "@iconPath1",
            "selectedIconPath": "@selectedIconPath1"
        }, {
            "iconPath": "@iconPath2",
            "selectedIconPath": "@selectedIconPath2"
        }]
    }
}
```

theme.json



```
{
  "light": {
      "navBgColor": "#f6f6f6",
      "navTxtStyle": "black",
      "bgColor": "#ffffff",
      "bgTxtStyle": "light",
      "bgColorTop": "#eeeeee",
      "bgColorBottom": "#efefef",
      "tabFontColor": "#000000",
      "tabSelectedColor": "#3cc51f",
      "tabBgColor": "#ffffff",
```

版权所有：腾讯云计算（北京）有限责任公司

```
        "tabBorderStyle": "black",
        "iconPath1": "image/icon1_light.png",
        "selectedIconPath1": "image/selected_icon1_light.png",
        "iconPath2": "image/icon2_light.png",
        "selectedIconPath2": "image/selected_icon2_light.png",
    },
    "dark": {
        "navBgColor": "#191919",
        "navTxtStyle": "white",
        "bgColor": "#1f1f1f",
        "bgTxtStyle": "dark",
        "bgColorTop": "#191919",
        "bgColorBottom": "#1f1f1f",
        "tabFontColor": "#ffffff",
        "tabSelectedColor": "#51a937",
        "tabBgColor": "#191919",
        "tabBorderStyle": "white",
        "iconPath1": "image/icon1_dark.png",
        "selectedIconPath1": "image/selected_icon1_dark.png",
        "iconPath2": "image/icon2_dark.png",
        "selectedIconPath2": "image/selected_icon2_dark.png",
    }
}
```

3. Get current system theme

If app.json declares `"darkmode": true`, `wx.getSystemInfo` or `wx.getSystemInfoSync` will return a `theme` attribute with a value of light or dark.

If app.json does not declare `"darkmode": true`, the theme property is not available (i.e. `theme` is `undefined`).

4. Listening for theme switching events

There are 2 ways to do this:

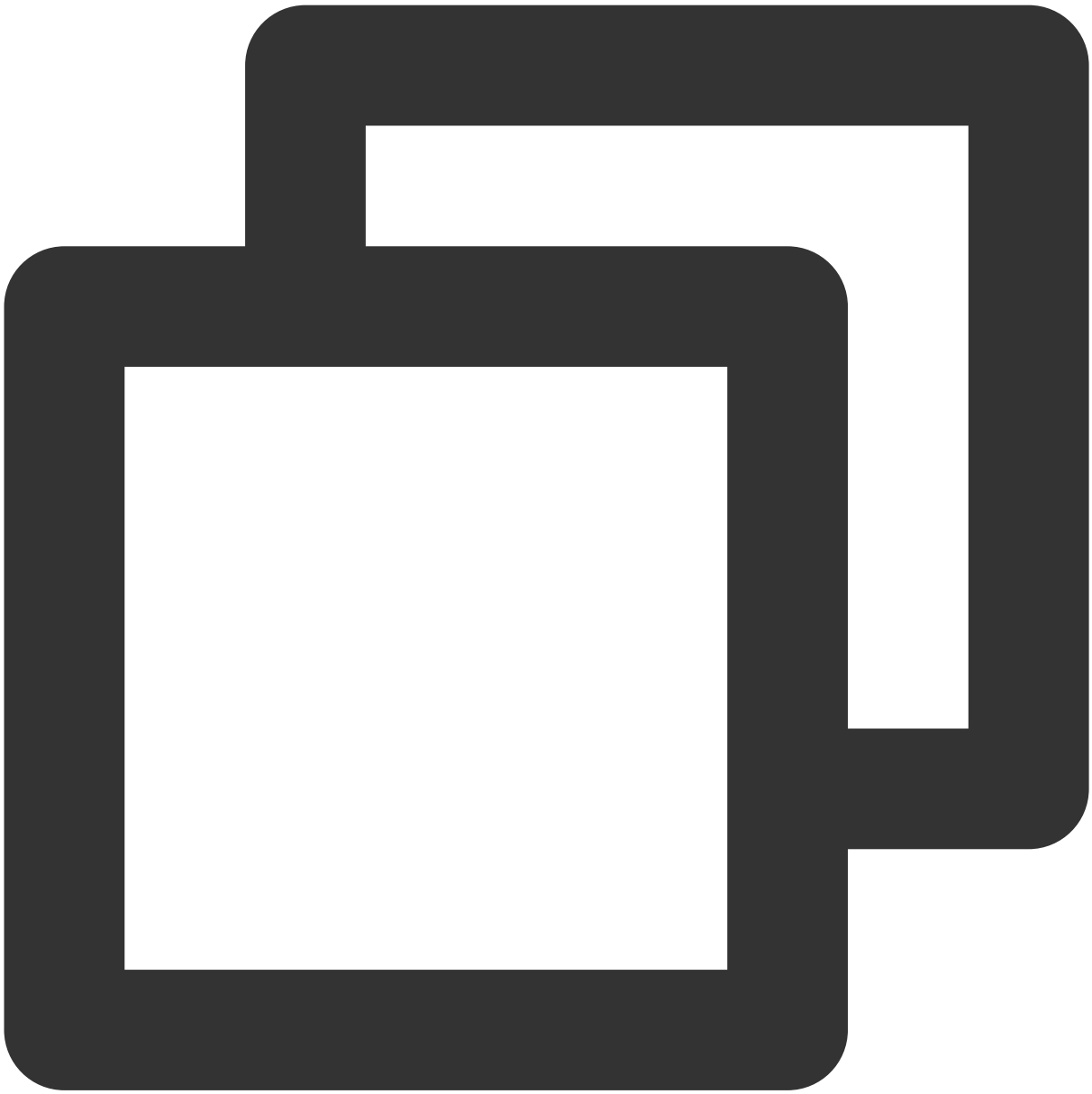
Pass onThemeChange callback method in App(), the callback will be triggered when the theme is switched;

Listen to theme change by wx.onThemeChange, wx.offThemeChange cancels the listening.

5. WXSS Adaptation

版权所有：腾讯云计算（北京）有限责任公司

WXSS supports adapting different themes via prefers-color-scheme media query, which is the same as the web adapting method, for example:



```
/* General style begin */
.some-background {
  background: white;
}
.some-text {
  color: black;
}
/* General style end */
```

```
@media (prefers-color-scheme: dark) {
  /* Style start under DarkMode */
  .some-background {
      background: #1b1b1b;
  }
  .some-text {
      color: #ffffff;
  }
  /* Style end under DarkMode */
}
```

6. Tips

1. Note that the media query in WXSS is not affected by the darkmode switch in app.json. As long as the client supports DarkMode, the media query will take effect when the system is switched to DarkMode, no matter "darkmode": true is configured or not;
2. Theme switch event needs to be triggered when "darkmode": true is configured.

版权所有：腾讯云计算（北京）有限责任公司

框架
小程序配置
项目配置

最近更新时间：2024-07-12 18:30:00

项目根目录中的 `project.config.json` 文件可以对项目进行配置。

`project.config.json` 文件在小程序中主要用于配置小程序开发工具的行为和特性。这个文件允许开发者调整开发工具的一些默认设置，以便更好地适应开发流程和环境。

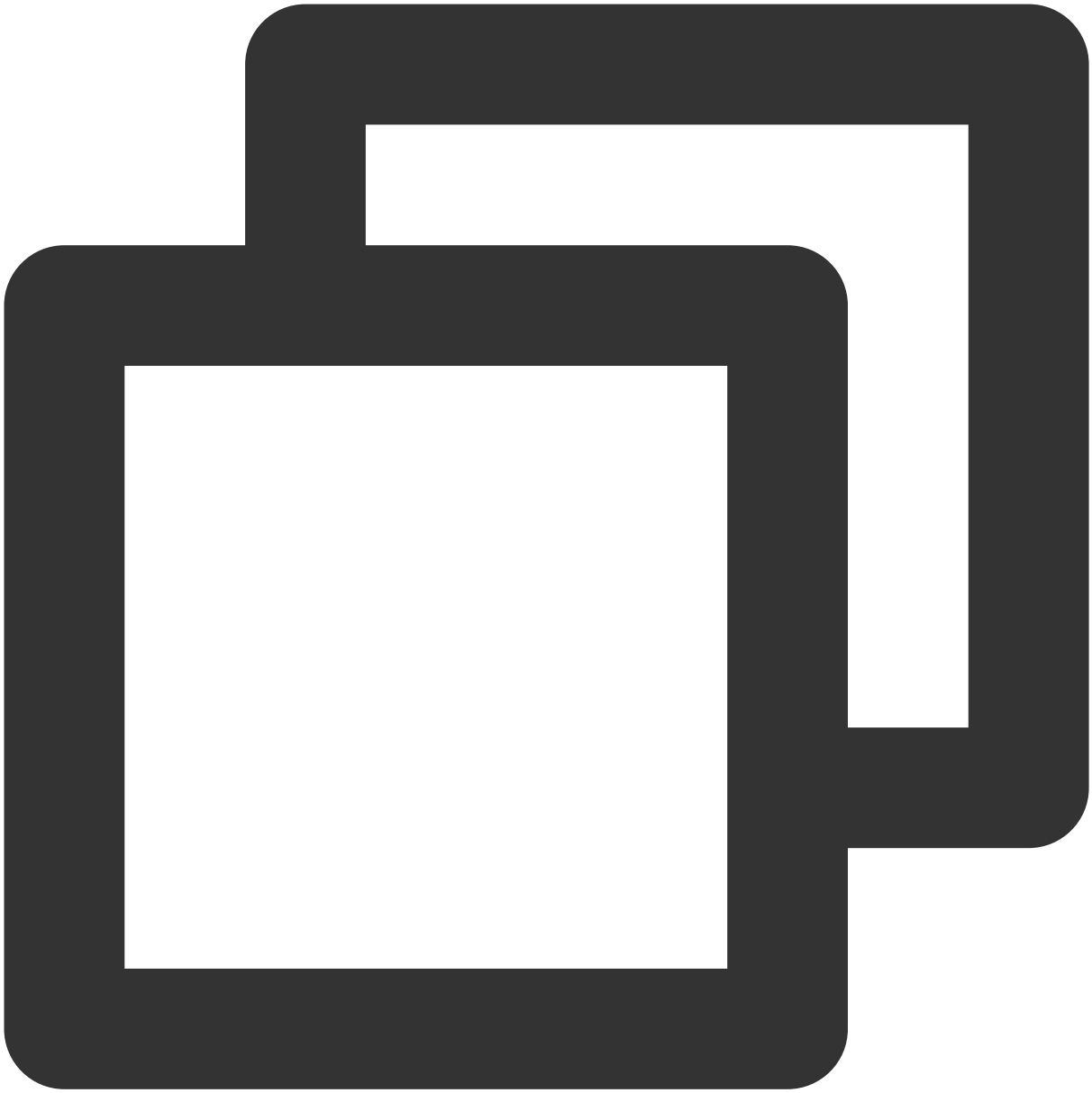
配置项

属性	类型	描述
miniprogramRoot	Path String	指定小程序源码的目录(需为相对路径)
compileType	String	编译类型，默认值为 miniprogram
TCMPPappid	String	项目的 appid
projectname	String	项目名字
setting	Object	项目设置
libVersion	String	基础库版本
condition	Object	条件编译配置
mockConfig	Object	IDE mockAPI配置

miniprogramRoot

小程序源码的目录(需为相对路径)。

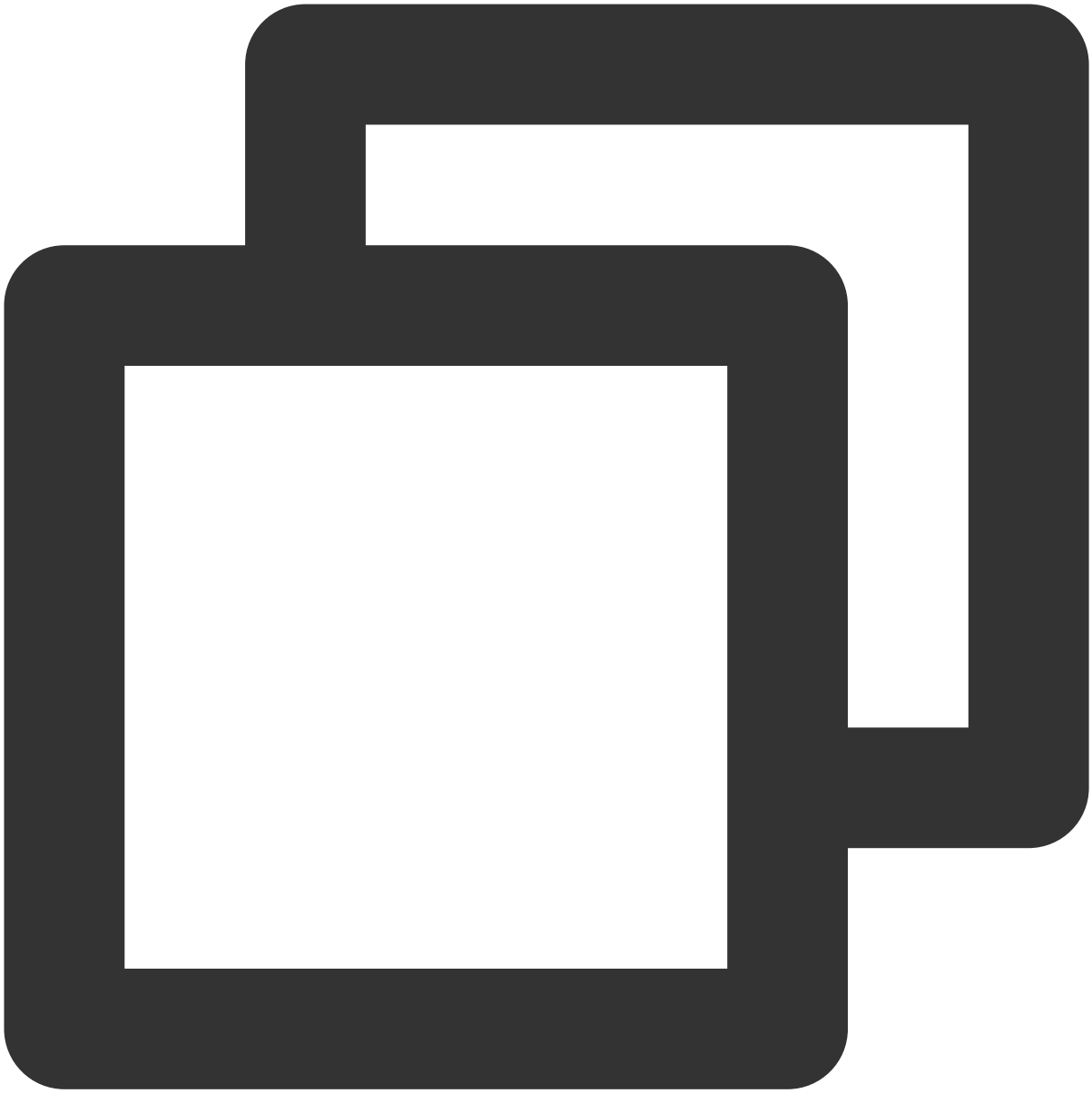
版权所有：腾讯云计算（北京）有限责任公司



```
{
  "miniprogramRoot": "./src"
}
```

compileType

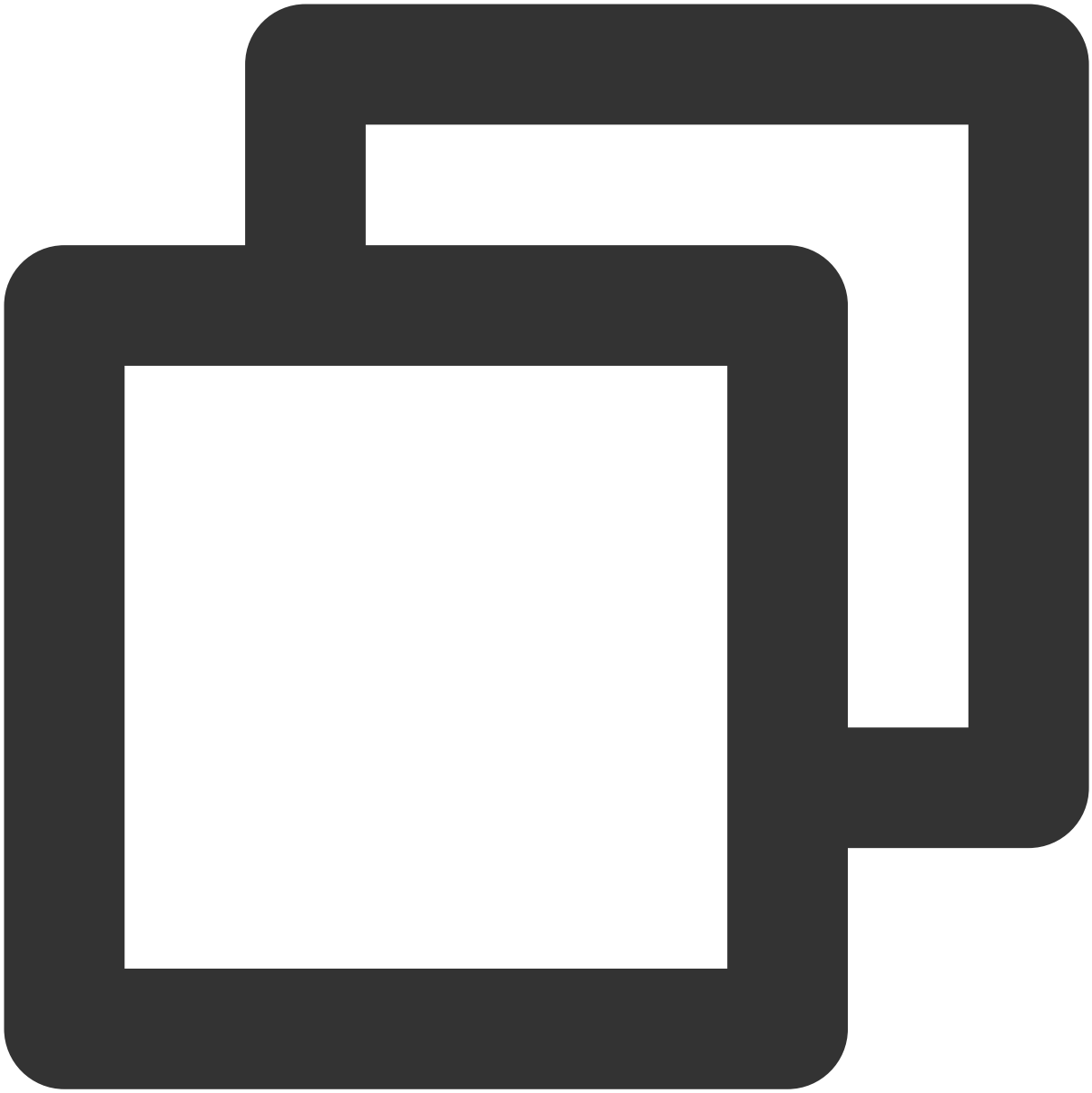
编译类型，默认值为 miniprogram。



```
{
  "compileType": "miniprogram"
}
```

TCMPPappid

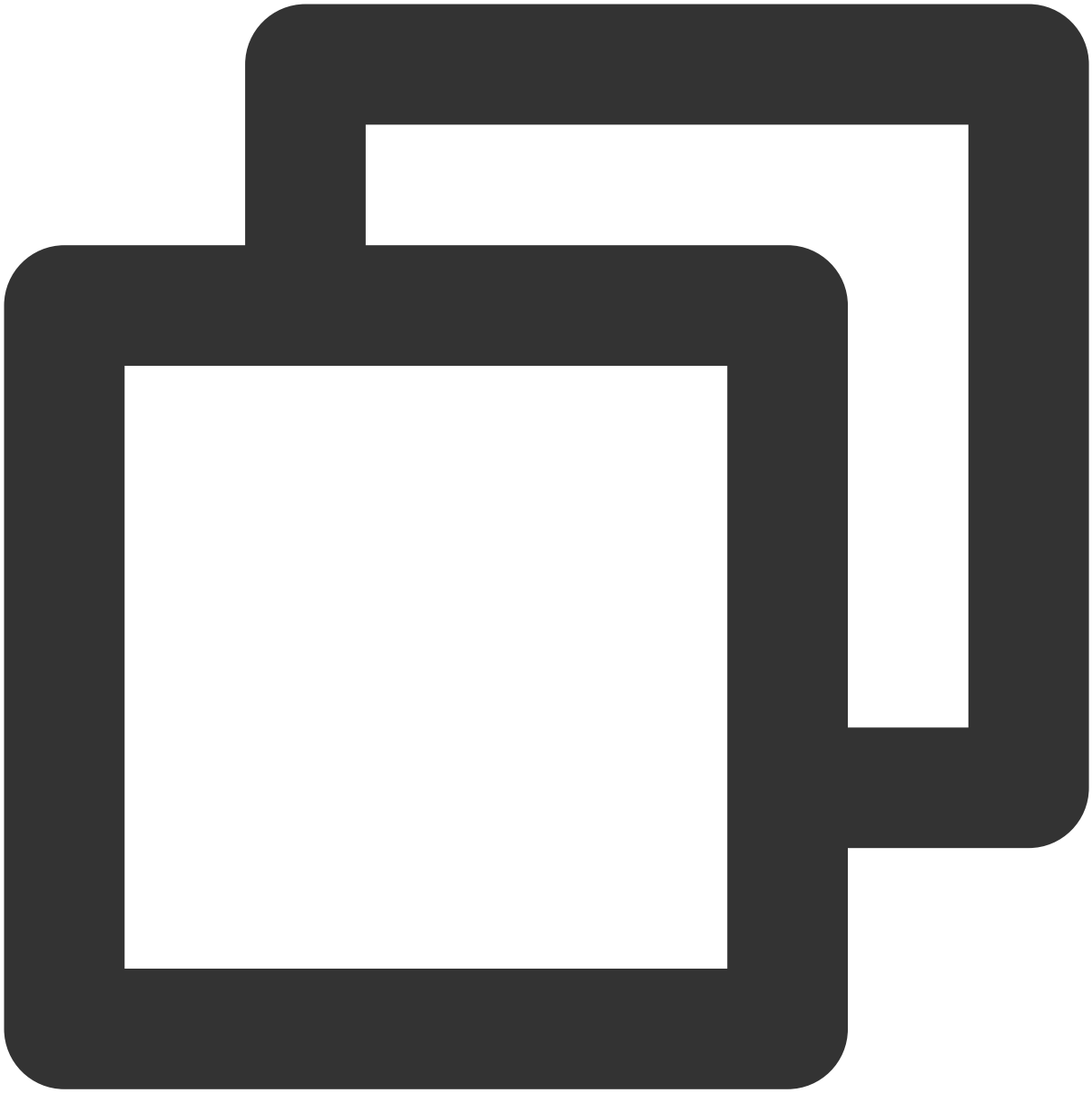
小程序在TCMPP控制台上对应的 appId。



```
{
  "TCMPPappid": "mpz5g4hpzd5p3k2l"
}
```

projectname

项目名称。



```
{
  "projectname": "main-demo"
}
```

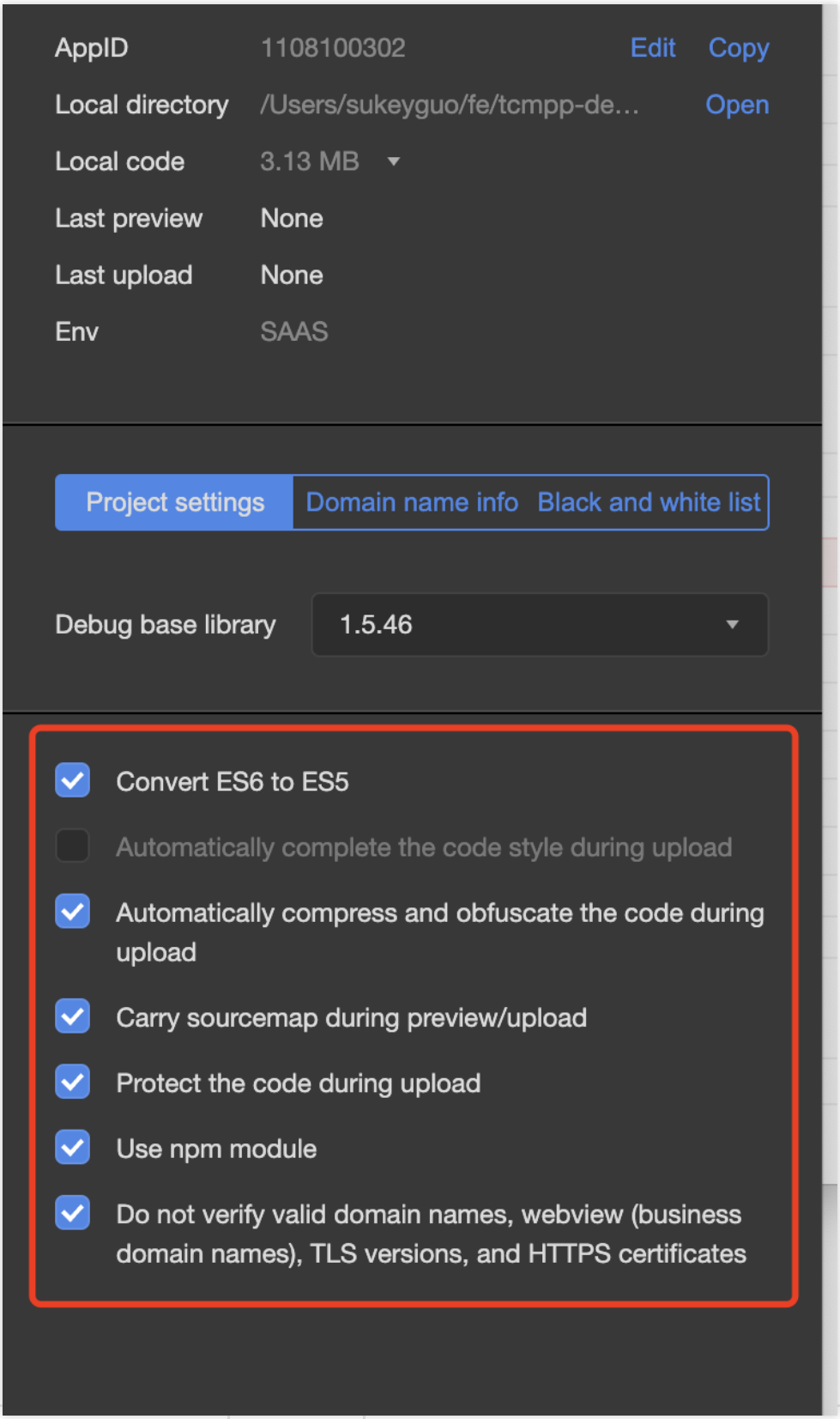
setting

项目的编译设置，可以指定以下设置。

以下字段可以开发者工具 > 详情 > 本地设置可以对应上。

版权所有：腾讯云计算（北京）有限责任公司


AppID
1108100302
编辑
复制
本地目录
/Users/sukeyguo/fe/tcmpp-de...
打开
本地代码
3.13 MB
上次预览
无
上次上传
无
当前环境
SAAS
项目设置
域名信息
黑/白名单
调试基础库
1.5.46
ES6 转 ES5
上传代码时样式自动补全
上传代码时自动压缩混淆
预览/上传时携带sourcemap
上传时进行代码保护
使用 npm 模块
不校验合法域名、web-view（业务域名）、TLS 版本以及 HTTPS 证书



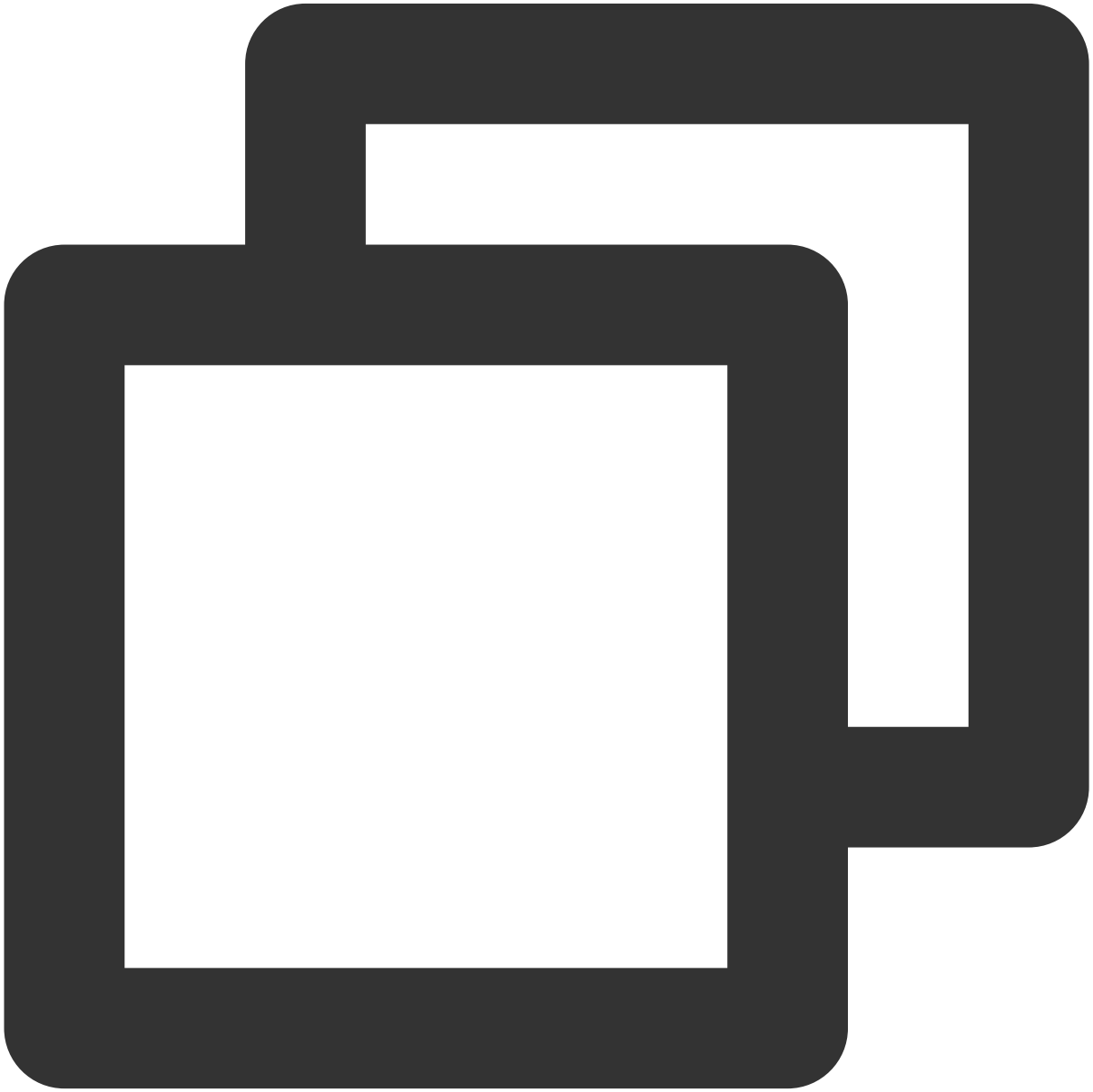
字段名	类型	说明
es6	Boolean	是否启用 es6 转 es5
postcss	Boolean	上传代码时样式是否自动补全

版权所有：腾讯云计算（北京）有限责任公司

minified	Boolean	上传代码时是否自动压缩脚本文件
uglifyFileName	Boolean	上传时进行代码保护
urlCheck	Boolean	是否检查安全域名和 TLS 版本以及HTTPS证书

libVersion

当前使用的基础库版本号。

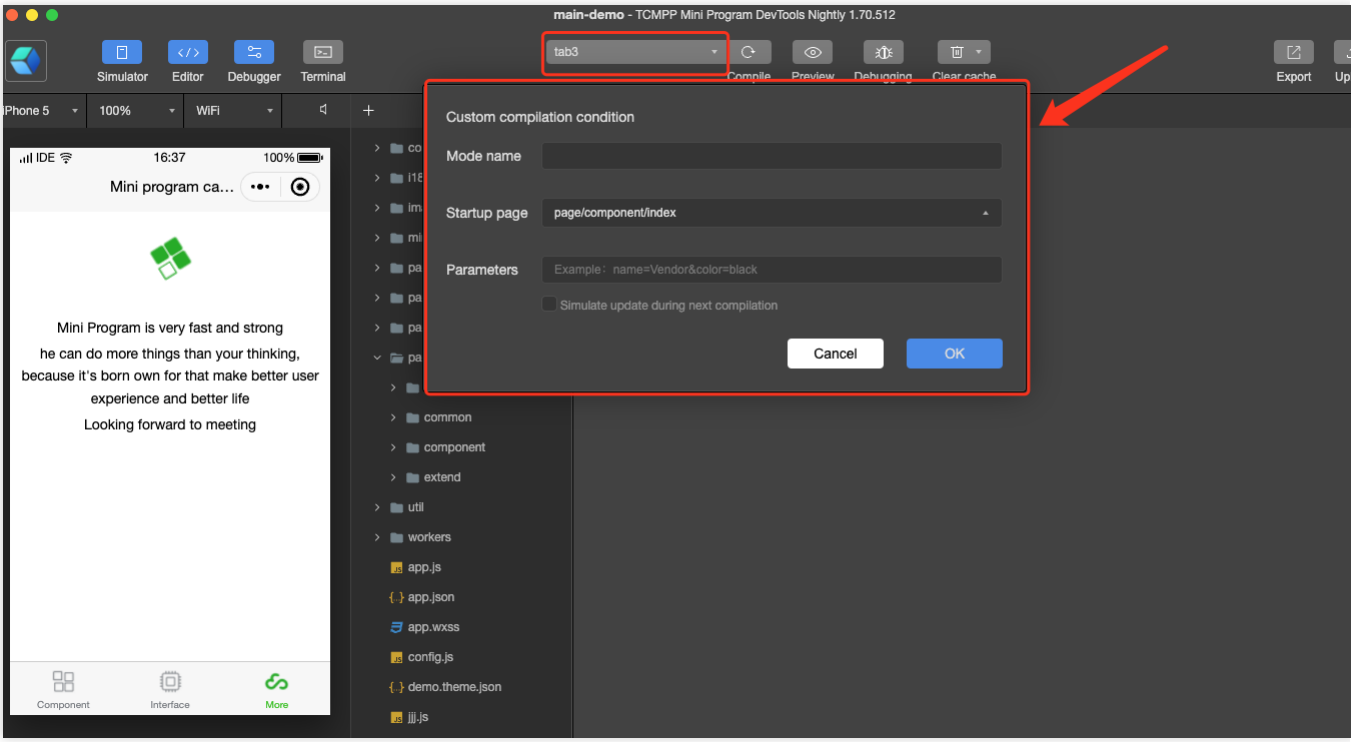


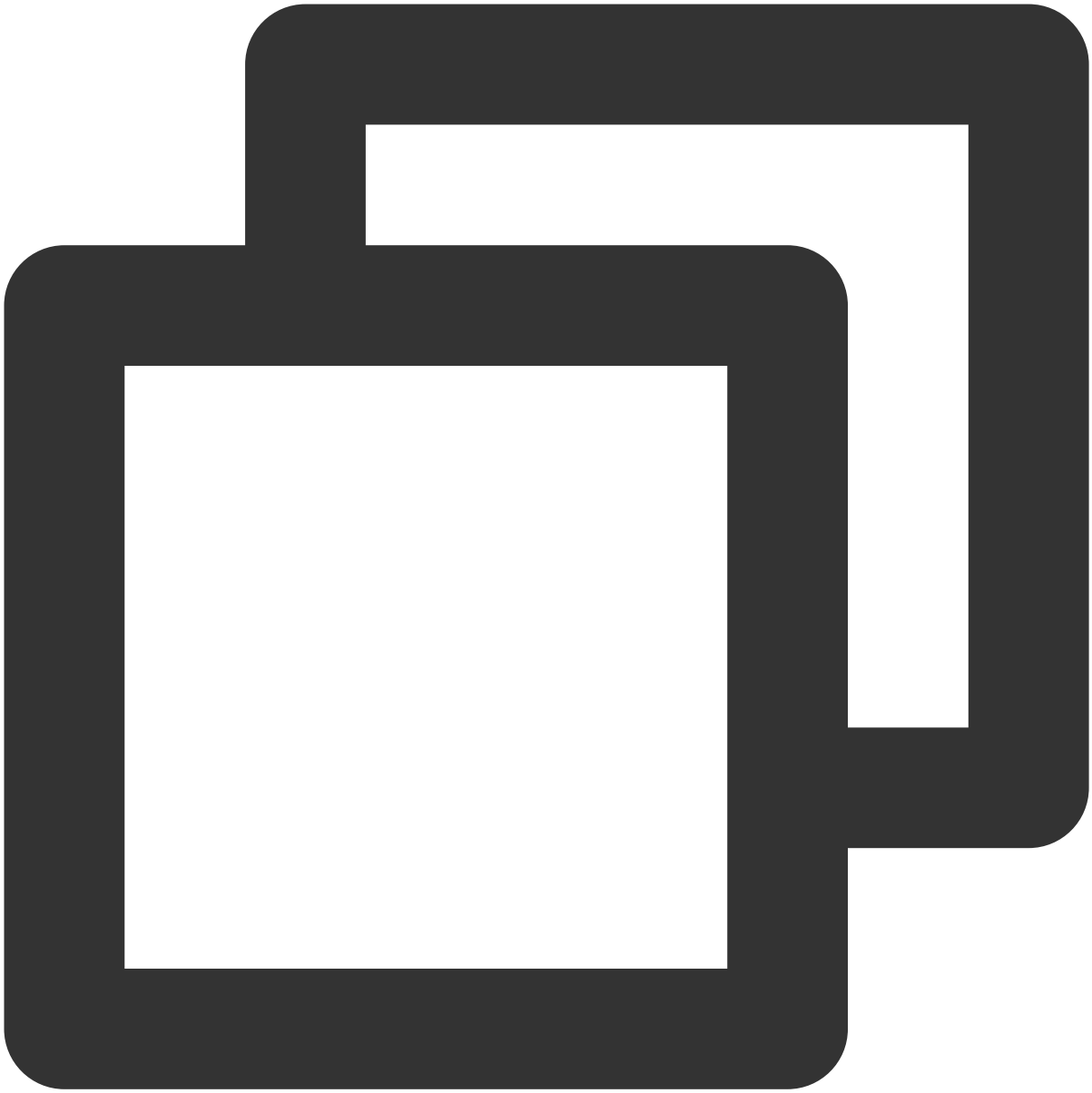
```
{
```

```
  "libVersion": "1.5.0"
}
```

condition

对应在 IDE 上设置的条件编译信息。



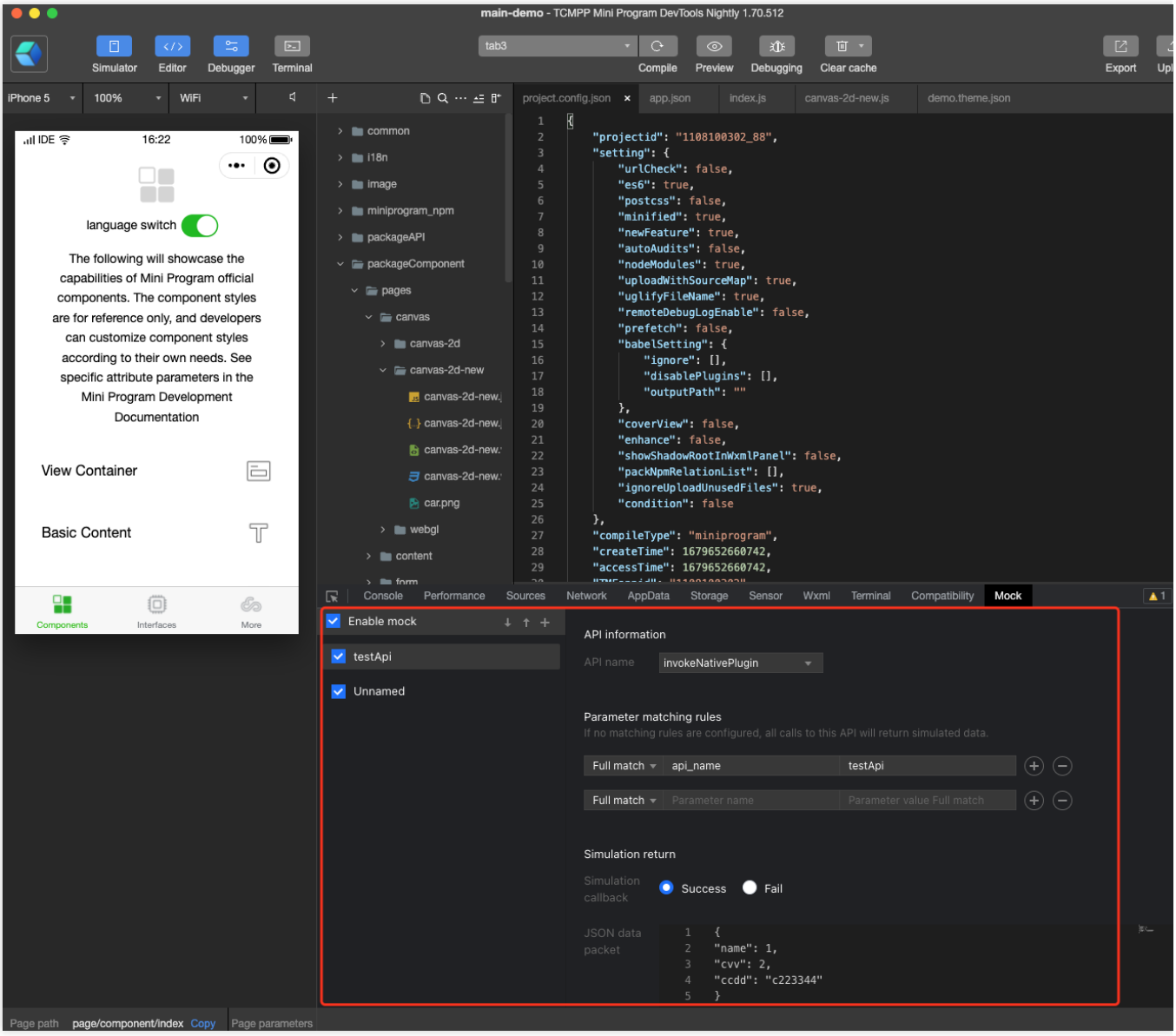


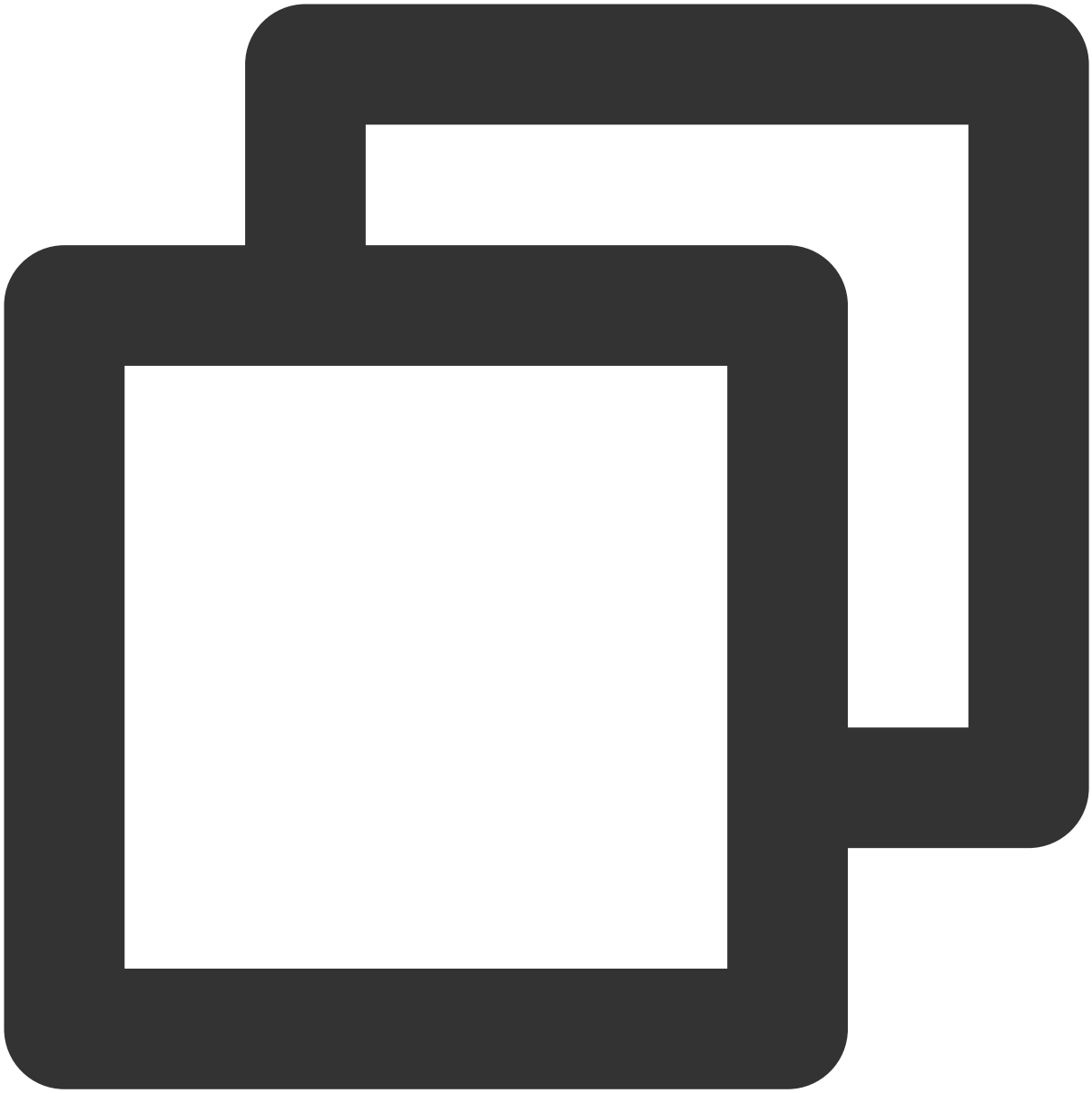

```
{
  "condition": {
    "miniprogram": {
      "current": 0,
      "list": [
        {
          "id": 0,
          "name": "tab3",
          "pathName": "page/extend/index",
          "query": "",
          "scene": null
```

```
        },
        {
          "id": -1,
          "name": "tab4",
          "pathName": "page/extend/index",
          "query": "name=1",
          "scene": null
        }
      ]
    }
  }
}
```

mockConfig

对应项目在 IDE 调试面板上对 API 进行 mock 时自动生成的配置信息。


main-demo - TCMPP Mini Program DevTools Nightly 1.70.512
Simulator
Editor
Debugger
Terminal
tab3
Compile
Preview
Debugging
Clear cache
Export
iPhone 5
100%
WiFi
language switch
The following will showcase the capabilities of Mini Program official components. The component styles are for reference only, and developers can customize component styles according to their own needs. See specific attribute parameters in the Mini Program Development Documentation
View Container
Basic Content
Components
Interfaces
More
project.config.json
app.json
index.js
canvas-2d-new.js
demo.theme.json
Console
Performance
Sources
Network
AppData
Storage
Sensor
Wxml
Terminal
Compatibility
Mock
Enable mock
testApi
Unnamed
API information
API name
invokeNativePlugin
Parameter matching rules
If no matching rules are configured, all calls to this API will return simulated data.
Full match
api_name
testApi
Parameter name
Parameter value Full match
Simulation return
Simulation callback
Success
Fail
JSON data packet
Page path
page/component/index
Copy
Page parameters




```
{
  "globalOpen": true,
  "rules": [
    {
      "ruleName": "testApi",
      "ruleId": "48b83a70-84fd-400d-b53d-097518cfa515",
      "apiName": "invokeNativePlugin",
      "enable": true,
      "mockType": "API",
      "returnType": "succ",
      "resStr": "{\\n\\"name\\": 1,\\n\\"cvv\\": 2,\\n\\"ccdd\\": \\"c223344\\"\\n}
```

```
      "filterList": [
        {
          "propName": "api_name",
          "propRegString": "testApi",
          "matchType": "string"
        },
        {
          "propName": "",
          "propRegString": "",
          "matchType": "string"
        }
      ],
      "verifyFail": false
    }
  ]
}
```

版权所有：腾讯云计算（北京）有限责任公司

全局配置

最近更新时间：2024-07-24 10:08:45

小程序根目录下的 `app.json` 文件用来对小程序进行全局配置。文件内容为一个 JSON 对象，有以下属性：

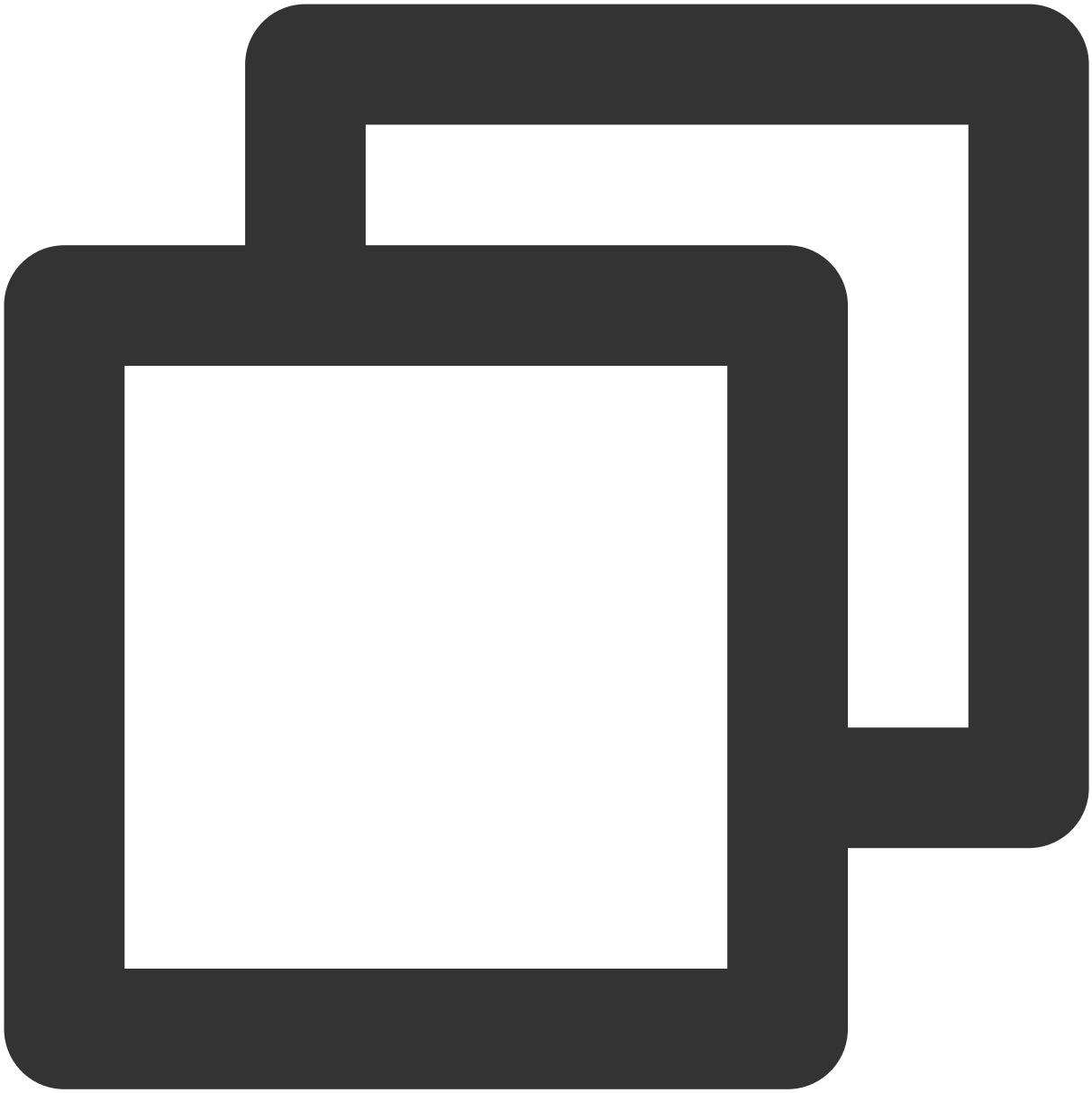
配置项

属性	类型	必填	描述
entryPagePath	string	否	小程序默认启动首页
pages	string[]	是	页面路径列表
window	Object	否	全局的默认窗口表现
tabBar	Object	否	底部 tab 栏的表现
debug	boolean	否	是否开启 debug 模式，默认关闭
subpackages	Object[]	否	分包结构配置
workers	string	否	Worker 代码放置的目录
navigateToMiniProgramAppIdList	string[]	否	需要跳转的小程序列表，详见 wx.navigateToMiniProgram
preloadRule	Object	否	分包预下载规则
usingComponents	Object	否	全局自定义组件配置
permission	Object	否	小程序接口权限相关设置
darkmode	boolean	否	小程序支持 DarkMode
themeLocation	string	否	指明 theme.json 的位置，darkmode 为 true 为必填

entryPagePath

指定小程序的默认启动路径（首页）。如果不填，将默认为 pages 列表的第一项。不支持带页面路径参数。

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "entryPagePath": "pages/index/index"
}
```

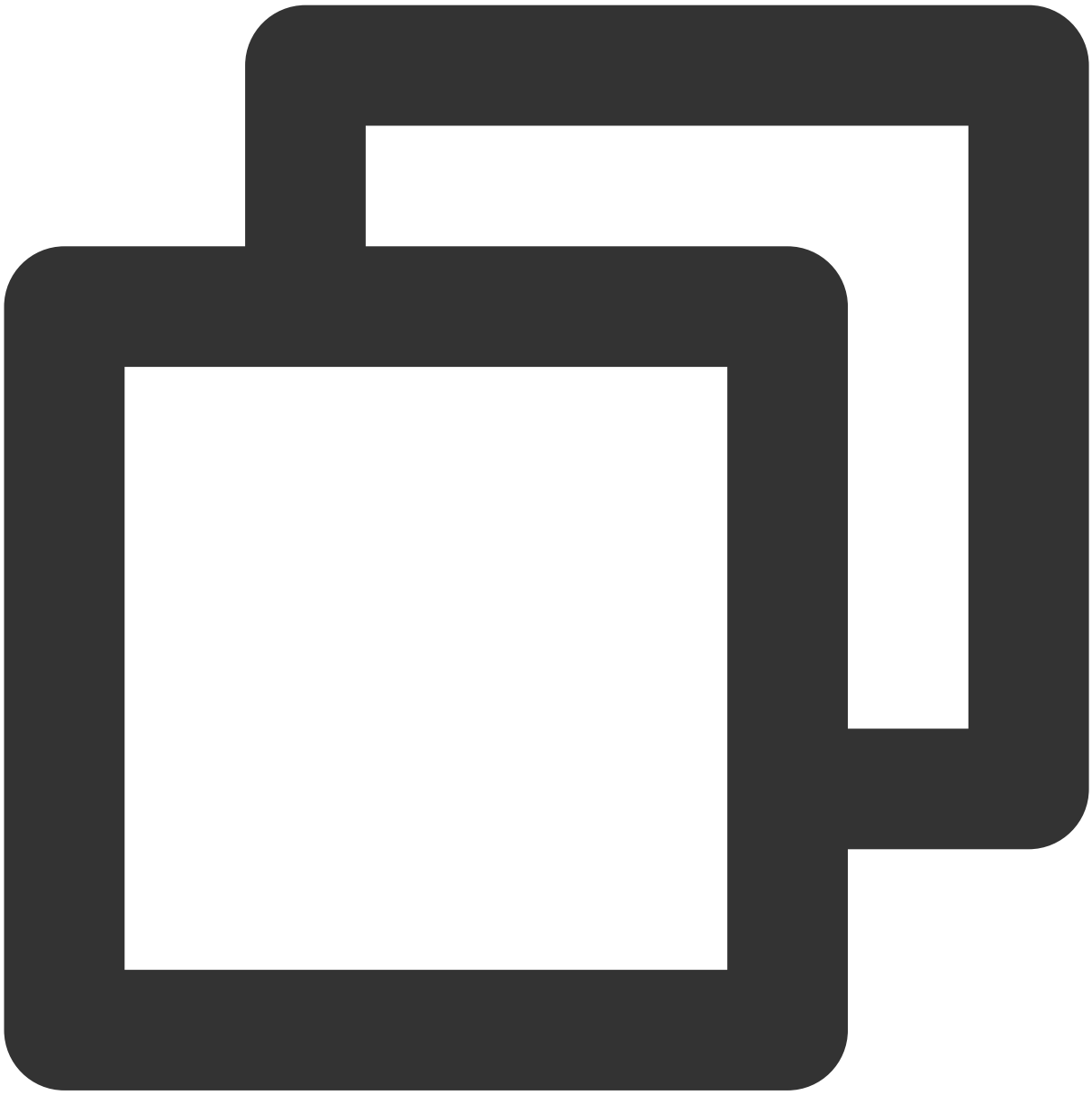
pages

用于指定小程序由哪些页面组成，每一项都对应一个页面的 路径（含文件名） 信息。文件名不需要写文件后缀，框架会自动去寻找对应位置的 .json，.js，.wxml，.wxss 四个文件进行处理。
未指定 entryPagePath 时，数组的第一项代表小程序的初始页面（首页）。

版权所有：腾讯云计算（北京）有限责任公司

小程序中新增/减少页面，都需要对 pages 数组进行修改。

如开发目录为：

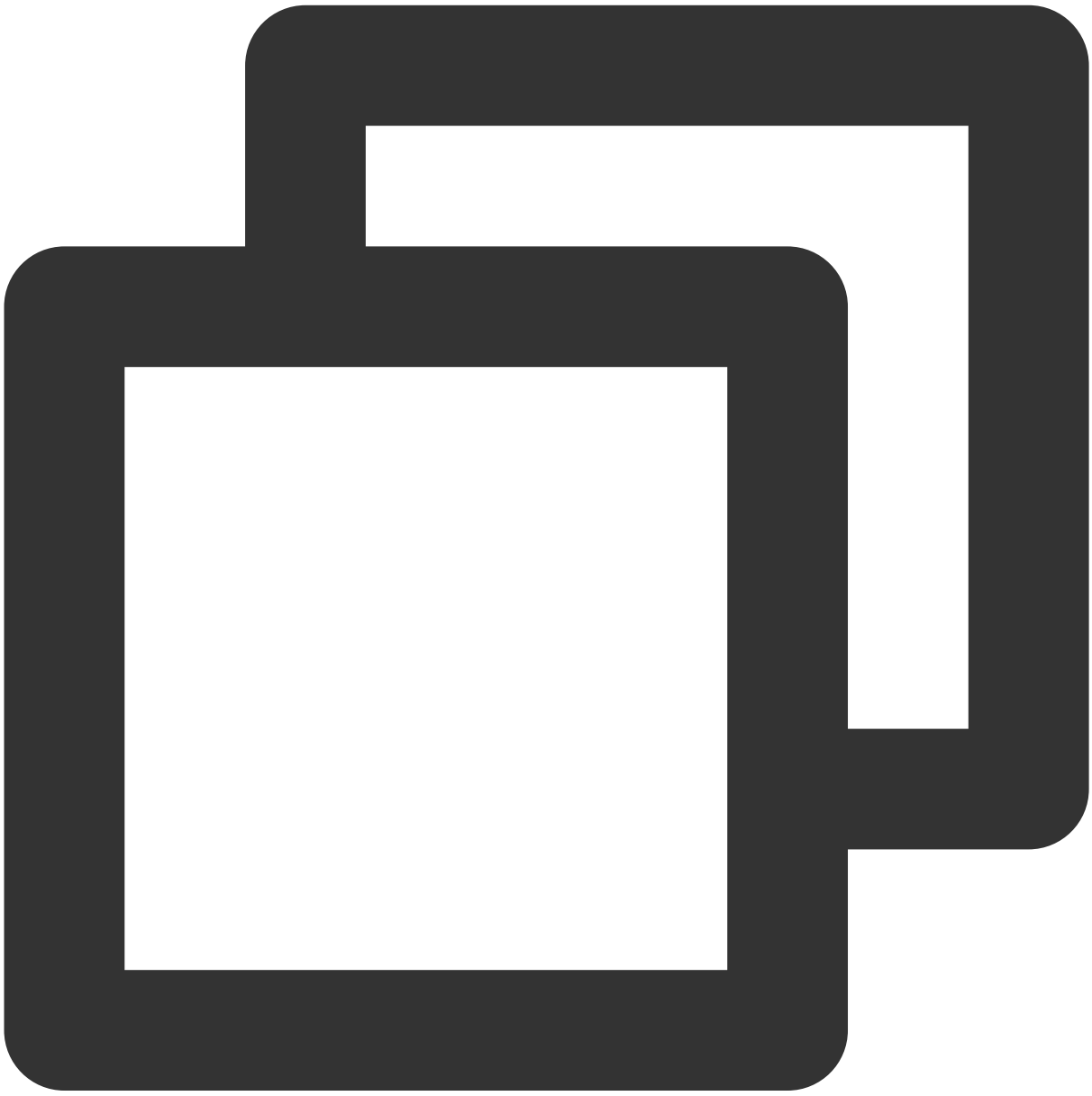


```
├── app.js
├── app.json
├── app.wxss
├── pages
│   └── index
│       ├── index.wxml
│       ├── index.js
│       ├── index.json
```

版权所有：腾讯云计算（北京）有限责任公司

```
|   |   └── index.wxss
|   └── logs
|       ├── logs.wxml
|       └── logs.js
└── utils
```

则需要在 app.json 中写：



```
{
  "pages": ["pages/index/index", "pages/logs/logs"]
}
```

window

用于设置小程序的状态栏、导航条、标题、窗口背景色。

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如 #000000
navigationBarTextStyle	string	white	导航栏标题、状态栏颜色，仅支持 black / white
navigationBarTitleText	string	-	导航栏标题文字内容
navigationStyle	string	default	导航栏样式，仅支持以下值： default 默认样式 custom 自定义导航栏，只保留右上角胶囊按钮 hide 自定义导航栏，可以支持隐私导航栏和胶囊按钮。 参见表格后紧接的注 2。
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	string	dark	下拉 loading 的样式，仅支持 dark / light
backgroundColorTop	string	#ffffff	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	string	#ffffff	底部窗口的背景色，仅 iOS 支持
enablePullDownRefresh	boolean	false	是否开启全局的下拉刷新。 详见 Page.onPullDownRefresh
onReachBottomDistance	number	50	页面上拉触底事件触发时距页面底部距离，单位为 px。 详见 Page.onReachBottom
customNavigateBack	Boolean	false	是否需要拦截页面的默认返回（单击 tabBar 返回或侧滑或 back 键返回）结合 Page.onCustomBack 使用
pageOrientation	string	portrait	屏幕旋转设置，支持 auto / portrait / landscape 详见 响应显示区域变化

注 1：HexColor（十六进制颜色值），如"#ff00ff"

注 2：关于 `navigationStyle`

`navigationStyle` 只在 app.json 中生效。

`navigationStyle` : custom 对 web-view 组件无效

tabBar

版权所有：腾讯云计算（北京）有限责任公司

如果小程序是一个多 tab 应用（客户端窗口的底部或顶部有 tab 栏可以切换页面），可以通过 tabBar 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页面。

属性	类型	必填	默认值	描述
color	HexColor	是	-	tab 上的文字默认颜色，仅支持十六进制颜色
selectedColor	HexColor	是	-	tab 上的文字选中时的颜色，仅支持十六进制颜色
backgroundColor	HexColor	是	-	tab 的背景色，仅支持十六进制颜色
borderStyle	string	否	black	tabbar 上边框的颜色， 仅支持 black / white
list	Array	是	-	tab 的列表，详见 list 属性说明，最少 2 个、最多 5 个 tab
position	string	否	bottom	tabBar 的位置，仅支持 bottom / top
custom	boolean	否	false	自定义 tabBar，详情请参见 自定义 tabBar

其中 list 接受一个数组，**只能配置最少2个、最多5个 tab**。tab 按数组的顺序排序，每个项都是一个对象，其属性值如下：

属性	类型	必填	说明
pagePath	string	是	页面路径，必须在 pages 中先定义
text	string	是	tab 上按钮文字
iconPath	string	否	图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，不支持网络图片。 **当 position 为 top 时，不显示 icon**。
selectedIconPath	string	否	选中时的图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，不支持网络图片。 **当 position 为 top 时，不显示 icon**。

版权所有：腾讯云计算（北京）有限责任公司



debug

可以在开发者工具中开启 `debug` 模式，在开发者工具的控制台面板，调试信息以 `info` 的形式给出，其信息有 Page 的注册，页面路由，数据更新，事件触发等。可以帮助开发者快速定位一些常见的问题。

subpackages

启用 分包加载 时，声明项目分包结构。

说明：

写成 subPackages 也支持。

workers

使用 Worker 处理多线程任务时，设置 Worker 代码放置的目录。

navigateToMiniProgramAppIdList

当小程序需要使用 navigateToMiniProgram 接口跳转到其他小程序时，需要先在配置文件中声明需要跳转的小程序 appId 列表，最多允许填写10个。

preloadRule

声明 分包预下载 的规则。

usingComponents

在 `app.json` 中声明的自定义组件视为全局自定义组件，在小程序内的页面或自定义组件中可以直接使用而无需再声明。建议仅在此声明几乎所有页面都会用到的自定义组件。

注意：

全局自定义组件会视为被所有页面依赖，会在所有页面启动时进行初始化，影响启动性能且会占用主包大小。只被个别页面或分包引用的自定义组件应尽量在页面配置中声明。

permission

小程序接口权限相关设置。字段类型为 `Object`，结构为：

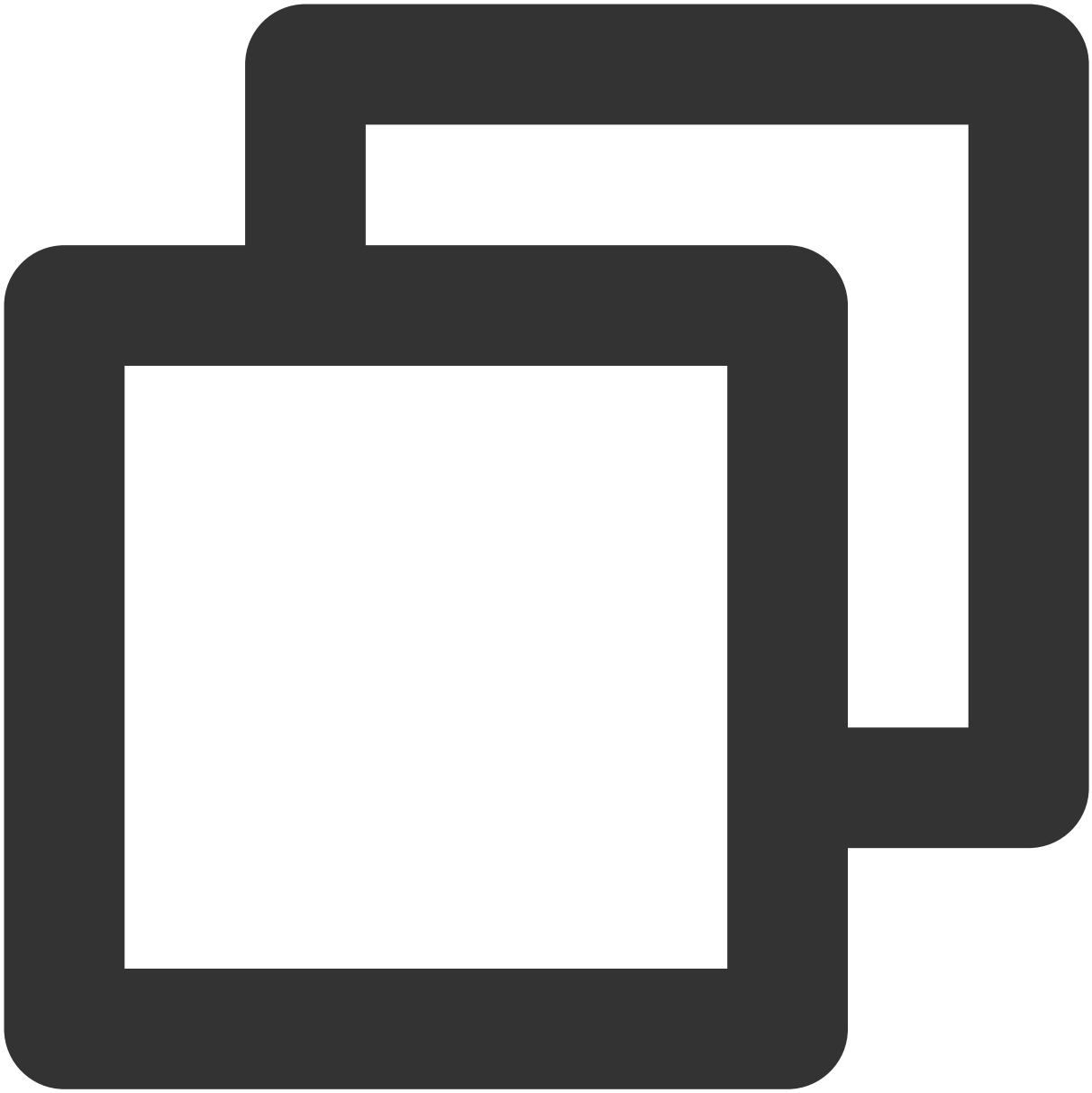
属性	类型	必填	默认值	描述
scope.userLocation	PermissionObject	否	-	位置相关权限声明

PermissionObject 结构

属性	类型	必填	默认值	说明
desc	string	是	-	小程序获取权限时展示的接口中文用途说明。最长80个字符
desc-en	string	否	-	小程序获取权限时展示的接口英文用途说明。最长80个字符
desc-adesc-ar	string	否	-	小程序获取权限时展示的接口阿语用途说明。最长80个字符

如：

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "pages": ["pages/index/index"],
  "permission": {
    "scope.userLocation": {
      "desc": "你的位置信息将用于小程序位置接口的效果展示" // 高速公路行驶持续后台定位
    }
  }
}
```

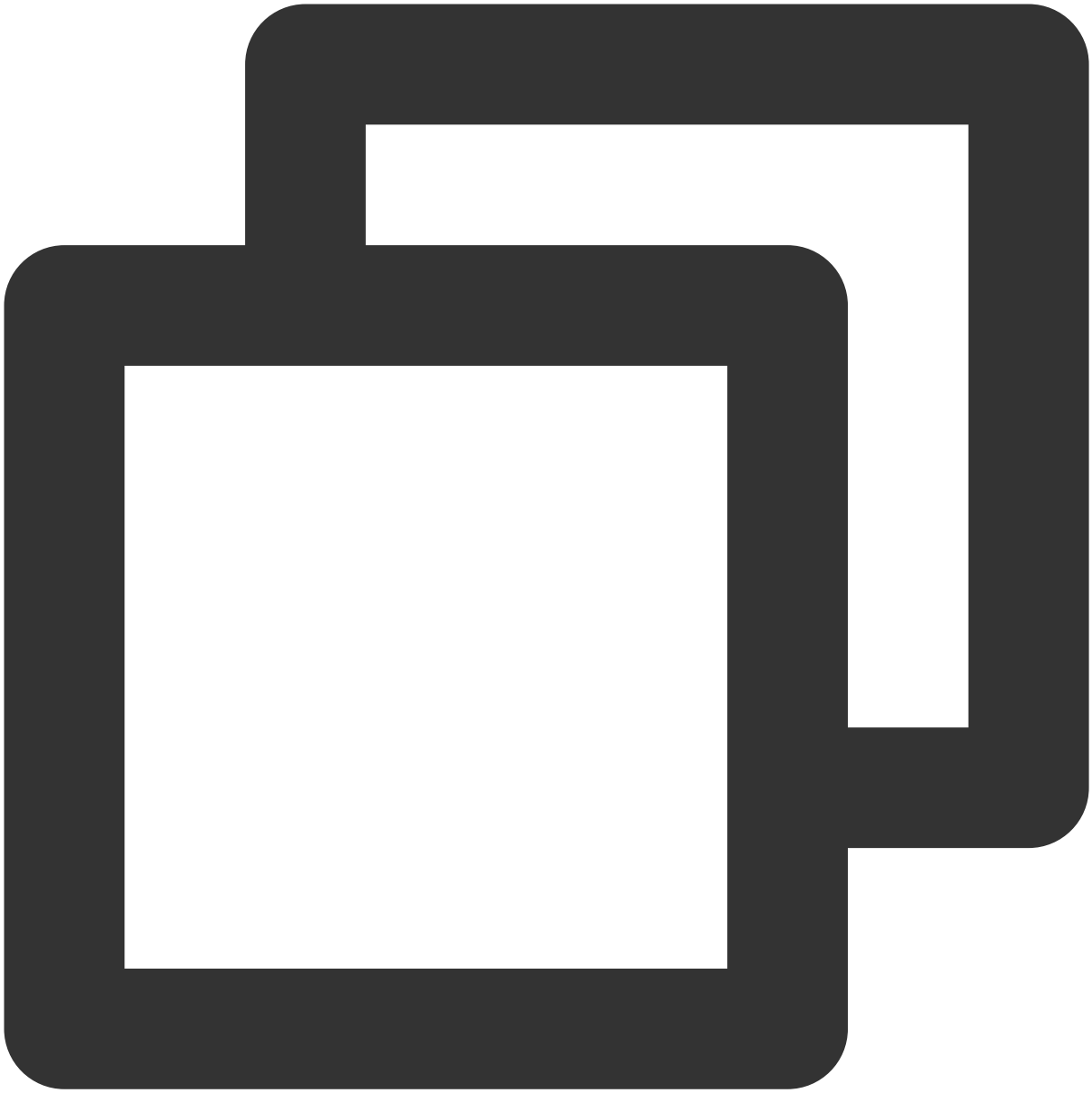
scope列表

scope	对应接口	描述
scope.userinfo	wx.getUserInfo	用户信息
scope.userLocation	wx.getLocation	获取地理位置信息
scope.userFuzzyLocation	wx.getFuzzyLocation	获取模糊地理位置信息
scope.record	live-pusher 组件 or wx.startRecord	直播或者录音
scope.camera	camera 组件	相机组件
scope.addPhoneCalendar	wx.addPhoneCalendar	添加日志
scope.writePhotosAlbum	wx.saveImageToPhotosAlbum	保存图片到相册
scope.bluetooth	wx.openBluetoothAdapter	蓝牙

darkmode

可通过配置 `"darkmode": true` 表示当前小程序可适配 DarkMode，所有基础组件均会根据系统主题展示不同的默认样式，navigation bar 和 tab bar 也会根据开发者的配置自动切换。
配置后，请根据DarkMode 适配指南自行完成基础样式以外的适配工作。

版权所有：腾讯云计算（北京）有限责任公司



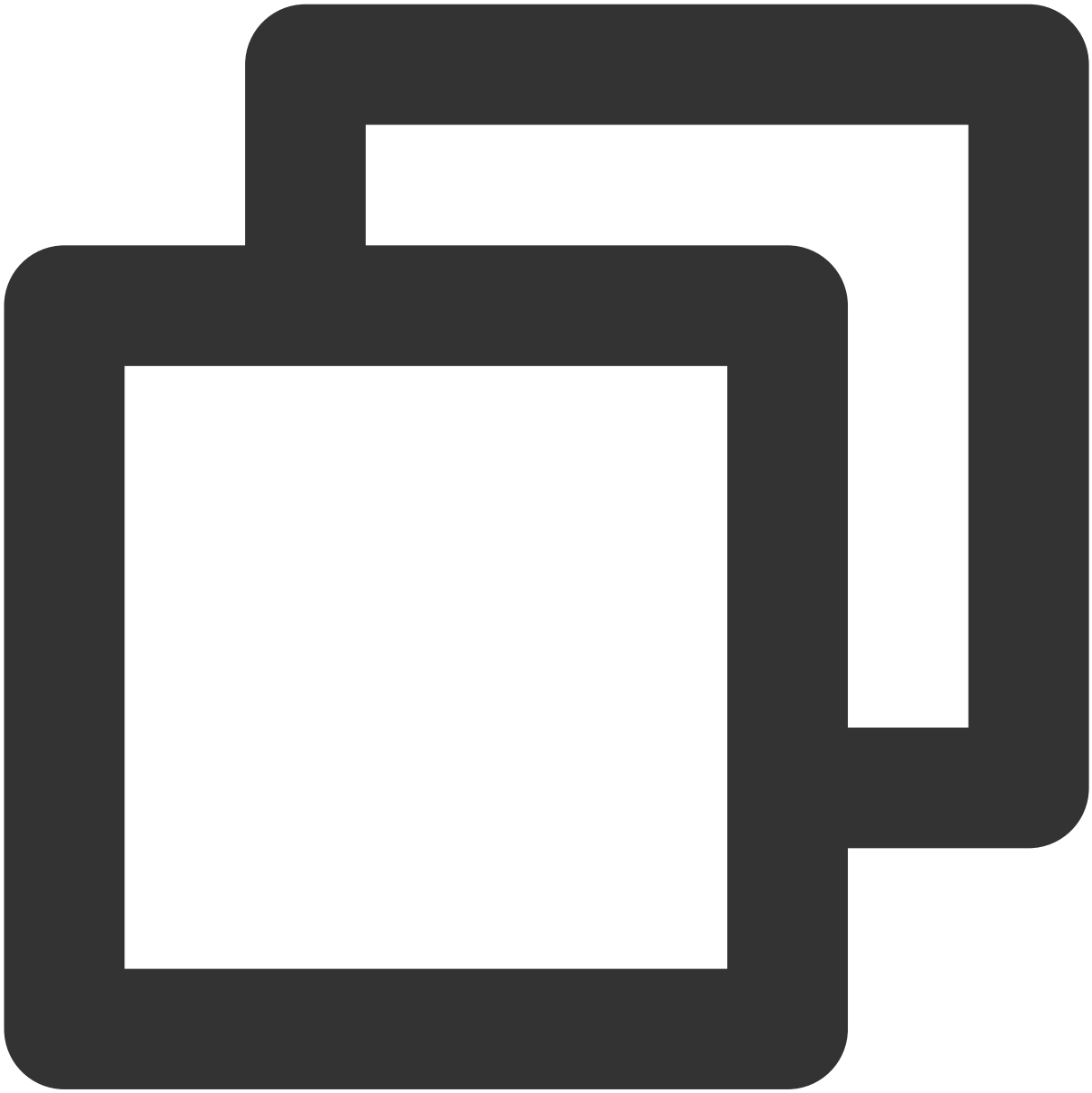
```
{
  "darkmode": true
}
```

themeLocation

自定义 theme.json 的路径，当配置 `"darkmode":true` 时，当前配置文件为必填项。

版权所有：腾讯云计算（北京）有限责任公司

配置示例



```
{
  "pages": ["pages/index/index", "pages/logs/index"],
  "window": {
    "navigationBarTitleText": "Demo"
  },
  "tabBar": {
    "list": [
      {
```

版权所有：腾讯云计算（北京）有限责任公司

```
        "pagePath": "pages/index/index",
        "text": "首页"
      },
      {
        "pagePath": "pages/logs/logs",
        "text": "日志"
      }
    ]
  },
  "debug": true
}
```

版权所有：腾讯云计算（北京）有限责任公司

页面配置

最近更新时间：2024-07-12 18:34:51

app.json 中的部分配置，也支持对单个页面进行配置，可以在页面对应的 `.json` 文件来对本页面的表现进行配置。
页面中配置项在当前页面会覆盖 app.json 中相同的配置项（样式相关的配置项属于 app.json 中的 window 属性，但这里不需要额外指定 window 字段），具体的取值和含义可参考 全局配置 中说明。
文件内容为一个 JSON 对象，有以下属性：

配置项

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如 #000000
navigationBarTextStyle	string	white	导航栏标题、状态栏颜色，仅支持 black / white
navigationBarTitleText	string		导航栏标题文字内容
navigationStyle	string	default	导航栏样式，仅支持以下值： default 默认样式 custom 自定义导航栏，只保留右上角胶囊按钮 hide 自定义导航栏，可以支持隐私导航栏和胶囊按钮。
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	string	dark	下拉 loading 的样式，仅支持 dark / light
backgroundColorTop	string	#ffffff	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	string	#ffffff	底部窗口的背景色，仅 iOS 支持
enablePullDownRefresh	boolean	false	是否开启当前页面下拉刷新。 详见 Page.onPullDownRefresh
onReachBottomDistance	number	50	页面上拉触底事件触发时距页面底部距离，单位为px。 详见 Page.onReachBottom
pageOrientation	string	portrait	屏幕旋转设置，支持 auto / portrait /

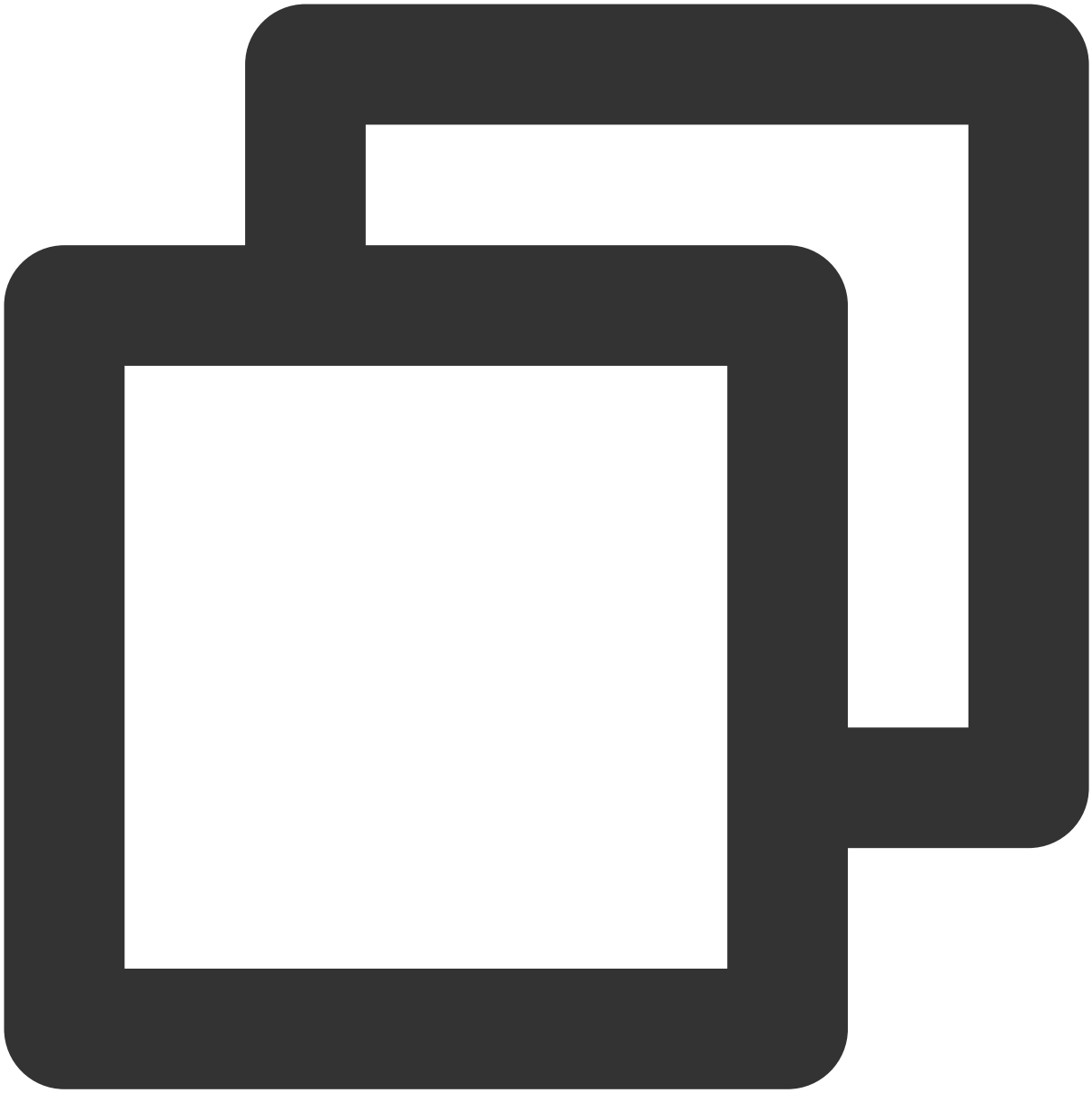
版权所有：腾讯云计算（北京）有限责任公司

			landscape 详见 响应显示区域变化
disableScroll	boolean	false	设置为 true 则页面整体不能上下滚动。 只在页面配置中有效，无法在 app.json 中设置
usingComponents	Object	否	页面自定义组件配置

注意：

并不是所有 app.json 中的配置都可以在页面覆盖或单独指定，仅限于本文档包含的选项。

配置示例



```
{
  "navigationBarBackgroundColor": "#ffffff",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "接口功能演示",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
}
```

版权所有：腾讯云计算（北京）有限责任公司

框架接口

App

最近更新时间：2024-07-12 18:40:38

App(Object object)

注册小程序。接受一个 Object 参数，其指定小程序的生命周期回调等。

App() 必须在 app.js 中调用，必须调用且只能调用一次。不然会出现无法预期的后果。

参数

Object object

属性	类型	默认值	必填	说明
onLaunch	function	-	否	生命周期回调——监听小程序初始化。
onShow	function	-	否	生命周期回调——监听小程序启动或切前台。
onHide	function	-	否	生命周期回调——监听小程序切后台。
onError	function	-	否	错误监听函数。
onPageNotFound	function	-	否	页面不存在监听函数。
onUnhandledRejection	function	-	否	未处理的 Promise 拒绝事件监听函数。
onThemeChange	function	-	否	监听系统主题变化
其他	any	-	否	开发者可以添加任意的函数或数据变量到 Object 参数中，用 this 可以访问

> 关于小程序前后台的定义和小程序的运行机制，请参考小程序运行机制章节。

onLaunch(Object object)

小程序初始化完成时触发，全局只触发一次。参数也可以使用 wx.getLaunchOptionsSync 获取。

参数：与 wx.getLaunchOptionsSync 一致。

onShow(Object object)

版权所有：腾讯云计算（北京）有限责任公司

小程序启动，或从后台进入前台显示时触发。也可以使用 wx.onAppShow 绑定监听。

参数：与 wx.onAppShow 一致。

onHide()

小程序从前台进入后台时触发。也可以使用 wx.onAppHide 绑定监听。

onError(String error)

小程序发生脚本错误或 API 调用报错时触发。也可以使用 wx.onError 绑定监听。

参数：与 wx.onError 一致。

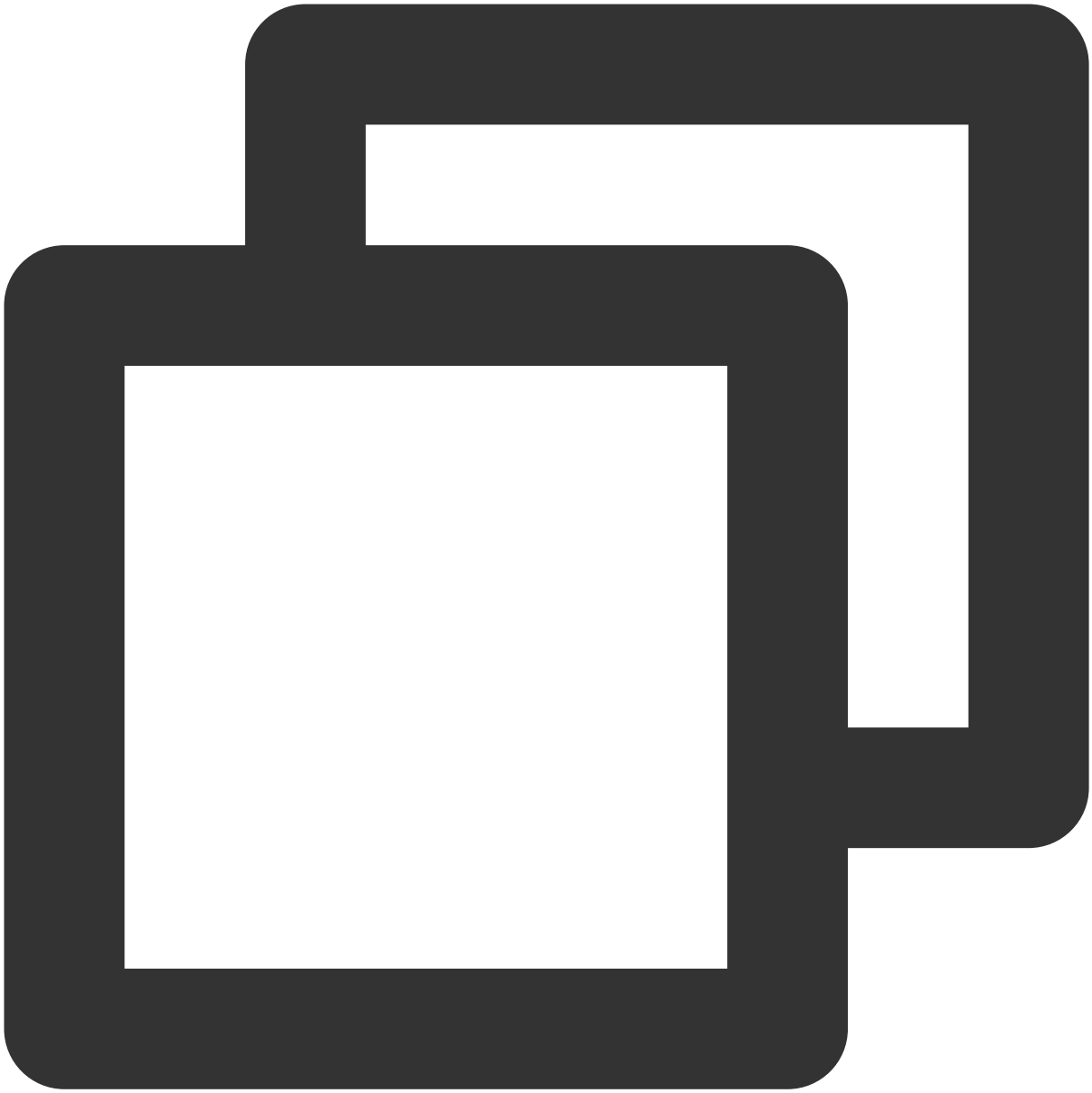
onPageNotFound(Object object)

小程序要打开的页面不存在时触发。也可以使用 wx.onPageNotFound 绑定监听。注意事项请参考 wx.onPageNotFound。

参数：与 wx.onPageNotFound 一致。

示例代码：

版权所有：腾讯云计算（北京）有限责任公司



```
App({
  onPageNotFound(res) {
    wx.redirectTo({
      url: 'pages/...'
    }) // 如果是 tabbar 页面，请使用 wx.switchTab
  }
})
```

onUnhandledRejection(Object object)

小程序有未处理的 Promise 拒绝时触发。也可以使用 wx.onUnhandledRejection 绑定监听。注意事项请参考 wx.onUnhandledRejection。

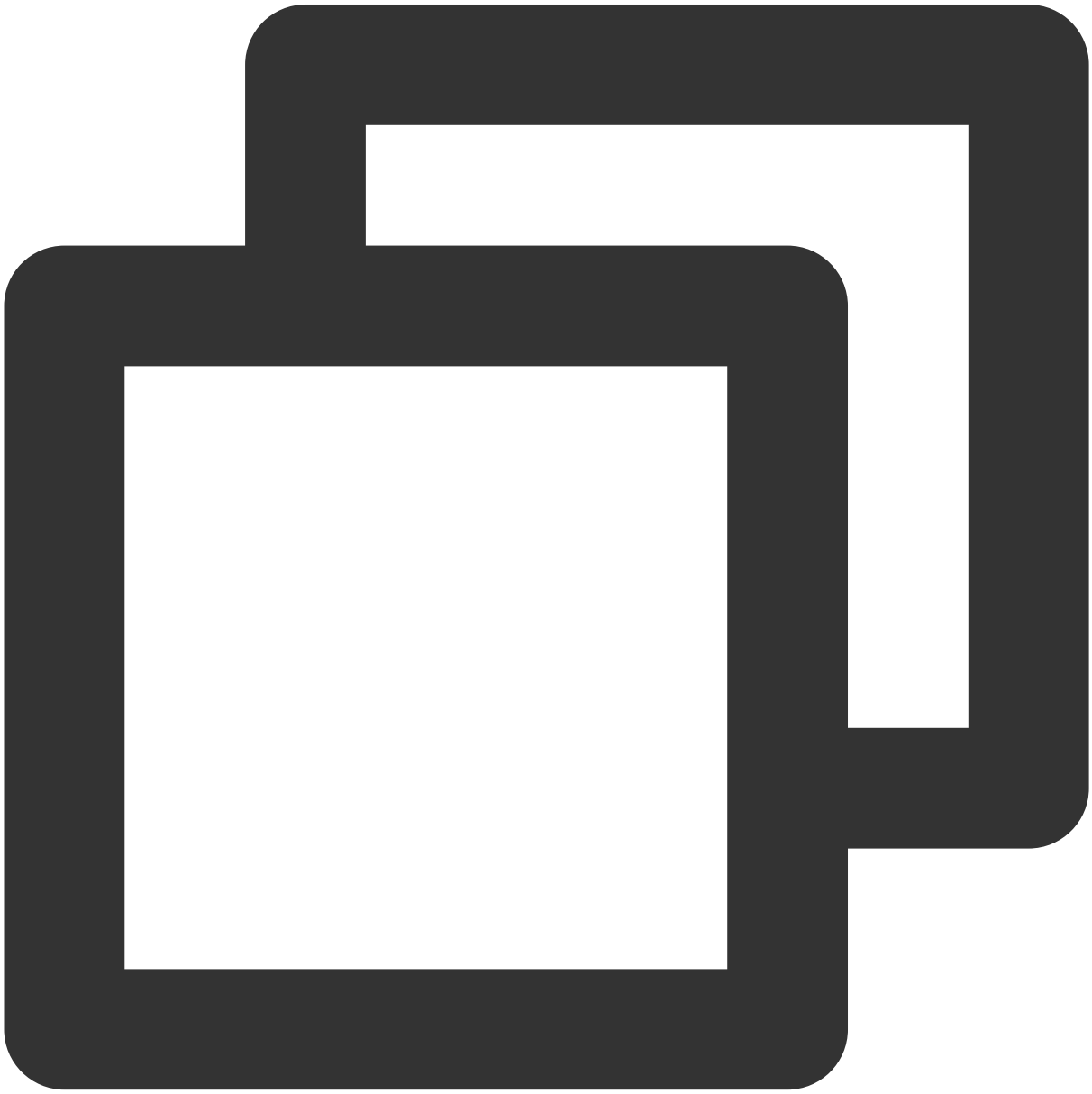
参数：与 wx.onUnhandledRejection 一致。

onThemeChange(Object object)

系统切换主题时触发。也可以使用 wx.onThemeChange 绑定监听。

参数：与 wx.onThemeChange 一致。

示例代码



```
App({
  onLaunch (options) {
    // Do something initial when launch.
  },
  onShow (options) {
    // Do something when show.
  },
  onHide () {
    // Do something when hide.
  },
  onError (msg) {
    console.log(msg)
  },
  globalData: 'I am global data'
})
```

AppObject getApp(Object object)

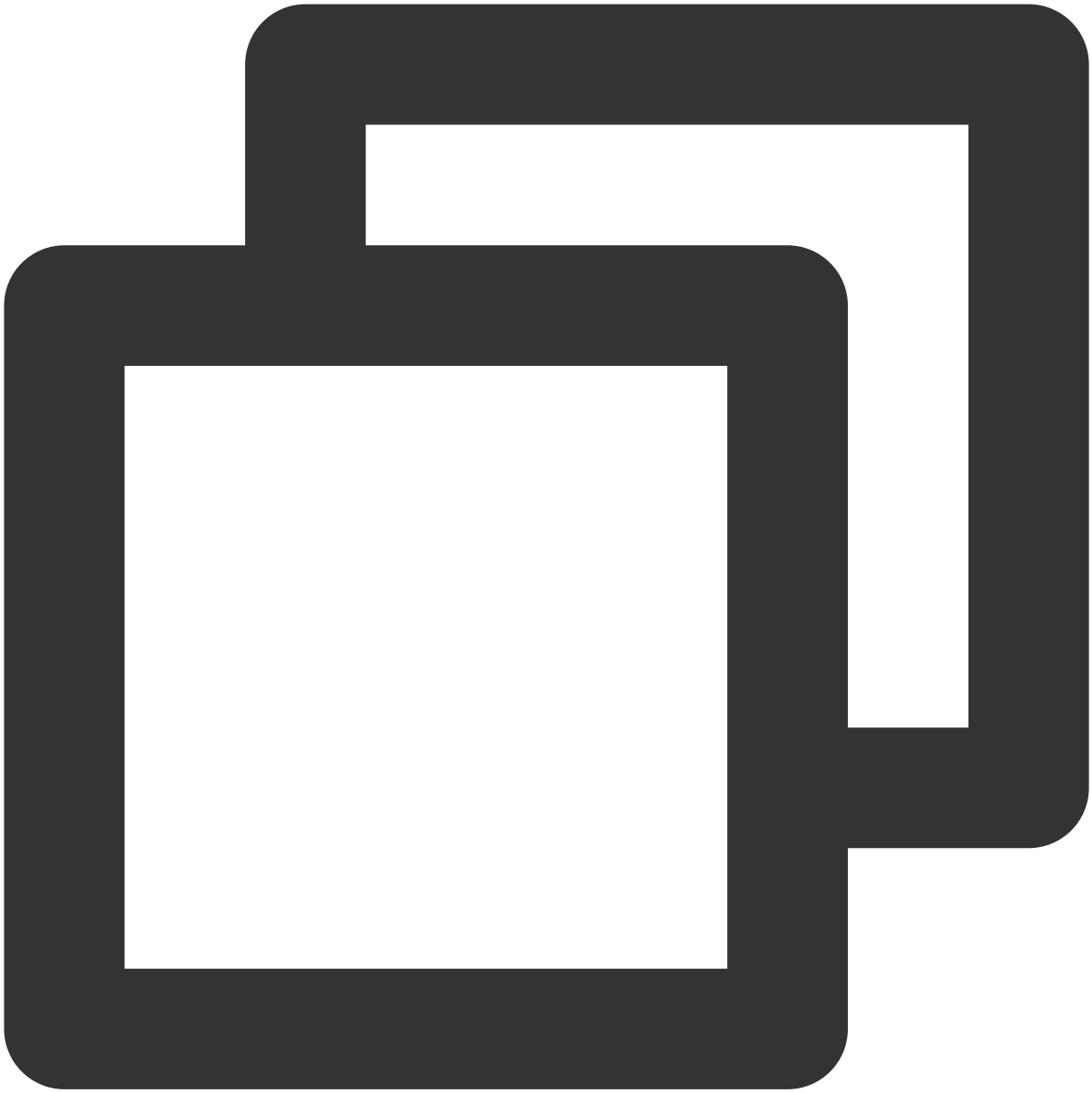
获取到小程序全局唯一的 App 实例。

参数

Object object

属性	类型	默认值	必填	说明
allowDefault	boolean	false	否	在 App 未定义时返回默认实现。当 App 被调用时，默认实现中定义的属性会被覆盖合并到 App 中。一般用于 独立分包。

示例代码



```
// other.js
var appInstance = getApp()
console.log(appInstance.globalData) // I am global data
```

注意事项

不要在定义于 `App()` 内的函数中，或调用 `App` 前调用 `getApp()` ，使用 `this` 就可以拿到 app 实例；通过 `getApp()` 获取实例之后，不要私自调用生命周期函数。

版权所有：腾讯云计算（北京）有限责任公司

页面

最近更新时间：2024-07-12 18:51:28

Page(Object object)

注册小程序中的一个页面。接受一个 `Object` 类型参数，其指定页面的初始数据、生命周期回调、事件处理函数等。

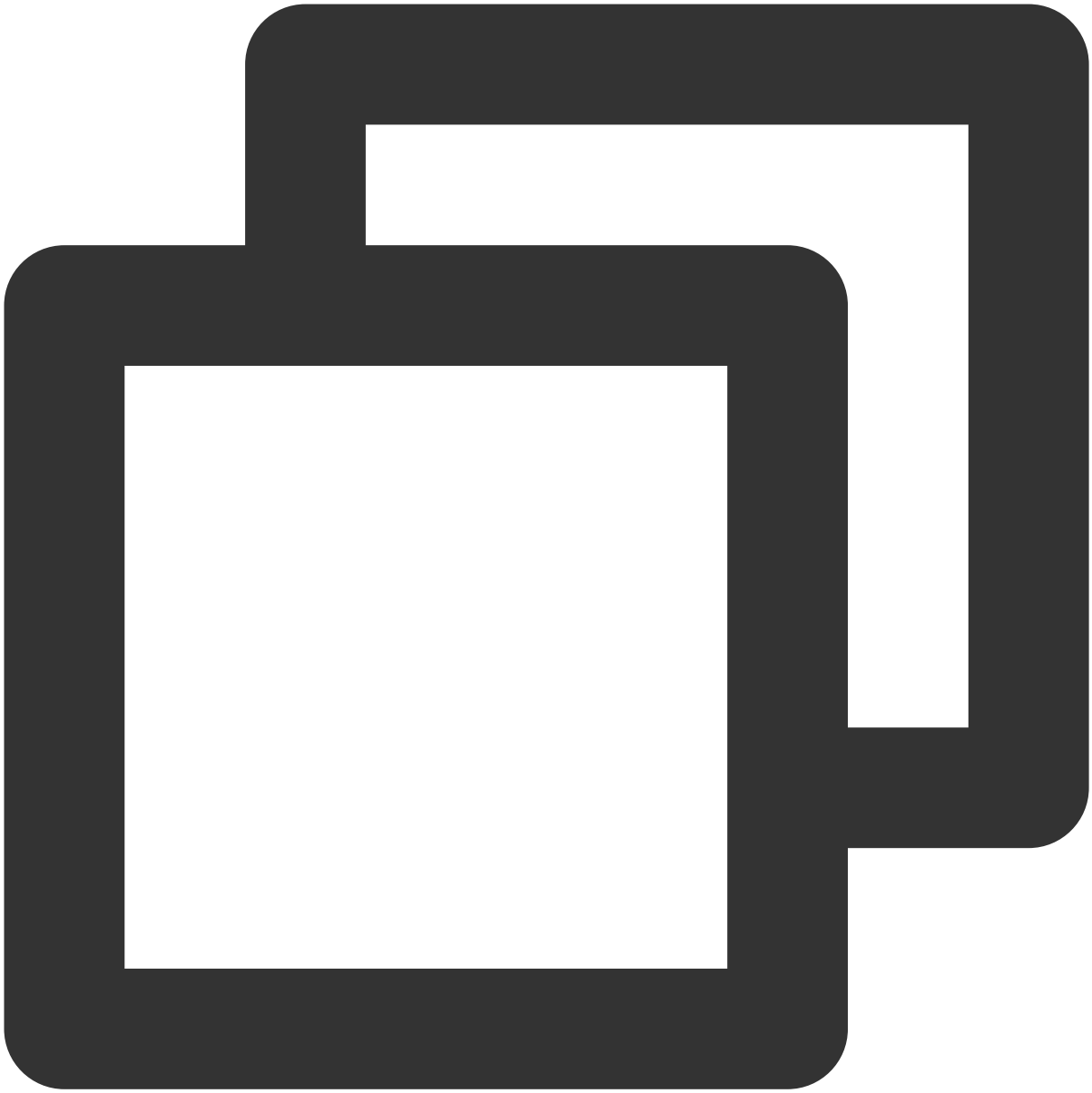
参数

Object object

属性	类型	默认值	必填	说明
data	Object	-	-	页面的初始数据
onLoad	function	-	-	生命周期回调—监听页面加载
onShow	function	-	-	生命周期回调—监听页面显示
onReady	function	-	-	生命周期回调—监听页面初次渲染完成
onHide	function	-	-	生命周期回调—监听页面隐藏
onUnload	function	-	-	生命周期回调—监听页面卸载
onPullDownRefresh	function	-	-	监听用户下拉动作
onReachBottom	function	-	-	页面上拉触底事件的处理函数
onShareAppMessage	function	-	-	用户单击右上角转发
onPageScroll	function	-	-	页面滚动触发事件的处理函数
onCustomBack	function	-	-	开发者拦截页面的默认返回时回调
onTabItemTap	function	-	-	当前是 tab 页时，单击 tab 时触发
其他	any	-	-	开发者可以添加任意的函数或数据到 Object 参数中，在页面的函数中用 this 可以访问。这部分属性会在页面实例创建时进行一次深拷贝。

示例代码：

版权所有：腾讯云计算（北京）有限责任公司



```
//index.js
Page({
  data: {
    text: "This is page data."
  },
  onLoad: function(options) {
    // Do some initialize when page load.
  },
  onShow: function() {
    // Do something when page show.
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  onReady: function() {
    // Do something when page ready.
  },
  onHide: function() {
    // Do something when page hide.
  },
  onUnload: function() {
    // Do something when page close.
  },
  onPullDownRefresh: function() {
    // Do something when pull down.
  },
  onReachBottom: function() {
    // Do something when page reach bottom.
  },
  onShareAppMessage: function () {
    // return custom share data when user share.
  },
  onPageScroll: function() {
    // Do something when page scroll
  },
  onResize: function() {
    // Do something when page resize
  },
  onTabItemTap(item) {
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  },
  // Event handler.
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    }, function() {
      // this is setData callback
    })
  },
  customData: {
    hi: 'MINA'
  }
})
```

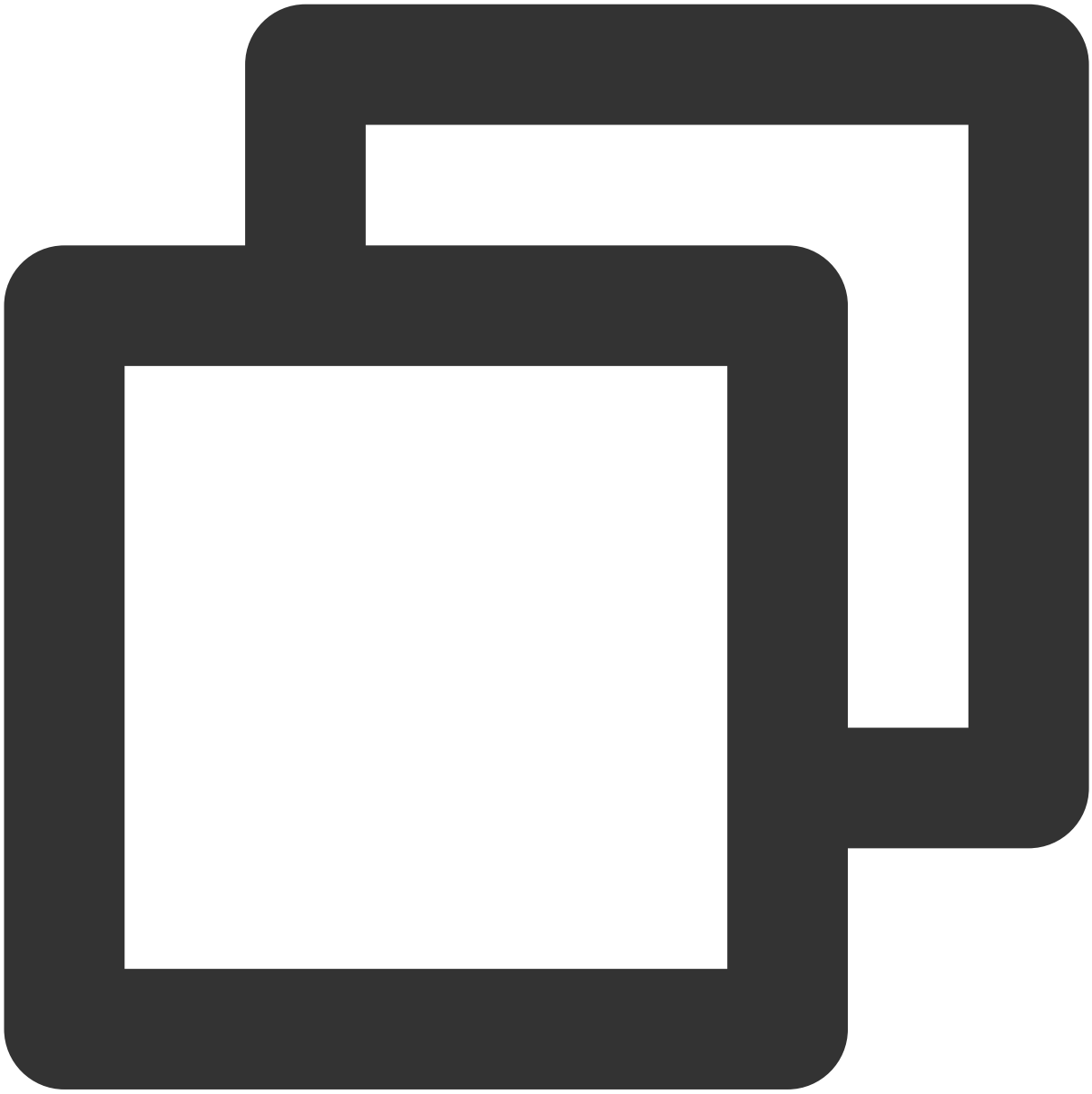
data

`data` 是页面第一次渲染使用的**初始数据**。

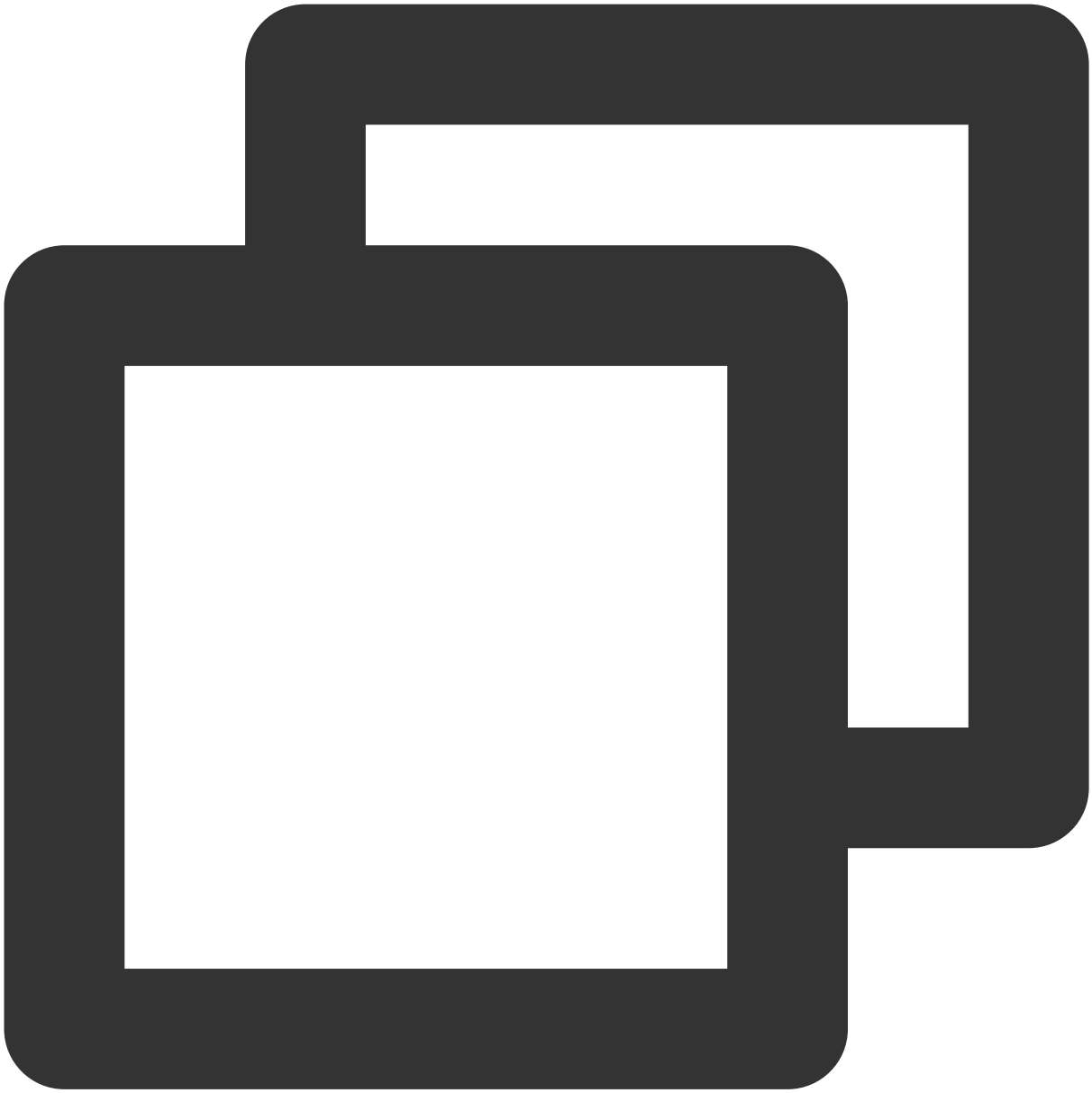
页面加载时， `data` 将会以 `JSON` 字符串的形式由逻辑层传至渲染层，因此 `data` 中的数据必须是可以转成 `JSON` 的类型：字符串，数字，布尔值，对象，数组。

渲染层可以通过 WXML 对数据进行绑定。

示例代码：



```
<view>{{text}}</view>
<view>{{array[0].msg}}</view>
```



```
Page({
  data: {
    text: 'init data',
    array: [{msg: '1'}, {msg: '2'}]
  }
})
```

生命周期回调函数

生命周期的触发以及页面的路由方式详见 逻辑层。

onLoad(Object query)

页面加载时触发。一个页面只会调用一次，可以在 onLoad 的参数中获取打开当前页面路径中的参数。

参数：

名称	类型	说明
query	Object	打开当前页面路径中的参数

onShow()

页面显示/切入前台时触发。

onReady()

页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。

注意：对界面内容进行设置的 API 如wx.setNavigationBarTitle，请在onReady之后进行。详见生命周期。

onHide()

页面隐藏/切入后台时触发。 如 wx.navigateTo 或底部 tab 切换到其他页面，小程序切入后台等。

onUnload()

页面卸载时触发。如wx.redirectTo 或 wx.navigateBack 到其他页面时。

页面事件处理函数

onPullDownRefresh()

监听用户下拉刷新事件。

需要在app.json的 window 选项中或 页面配置 中开启enablePullDownRefresh；

可以通过wx.startPullDownRefresh触发下拉刷新，调用后触发下拉刷新动画，效果与用户手动下拉刷新一致；

当处理完数据刷新后，wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。

onReachBottom()

监听用户上拉触底事件。

可以在app.json的 window 选项中或 页面配置 中设置触发距离onReachBottomDistance；

在触发距离内滑动期间，本事件只会被触发一次。

onPageScroll(Object object)

监听用户滑动页面事件。

参数 Object object：

属性	类型	说明
scrollTop	Number	页面在垂直方向已滚动的距离（单位px）

版权所有：腾讯云计算（北京）有限责任公司

注意：

请只在需要的时候才在 page 中定义此方法，不要定义空方法。以减少不必要的事件派发对渲染层-逻辑层通信的影响。

请避免在 onPageScroll 中过于频繁的执行 setData 等引起逻辑层-渲染层通信的操作。尤其是每次传输大量数据，会影响通信耗时。

onShareAppMessage(Object object)

监听用户单击页面内转发按钮（button 组件 open-type="share"）或右上角菜单“转发”按钮的行为，并自定义转发内容。

注意：

只有定义了此事件处理函数，右上角菜单才会显示“转发”按钮。

参数 Object object:

参数	类型	说明
from	String	转发事件来源 button：页面内转发按钮； menu：右上角转发菜单。
target	Object	如果 from 值是 button，则 target 是触发这次转发事件的 button，否则为 undefined
webViewUrl	String	页面中包含web-view组件时，返回当前web-view的url
shareTarget	string	用户单击分享到哪里。

此事件处理函数需要 return 一个 Object，用于自定义转发内容，返回内容如下：

字段	类型	说明	默认值
title	string	转发标题	当前小程序名称
path	string	转发路径	当前页面 path ，必须是以 / 开头的完整路径
imageUrl	string	自定义图片路径，可以是本地文件路径、代码包文件路径或者网络图片路径。支持 PNG 及 JPG。显示图片长宽比是 5:4。	使用默认截图
entryDataHash	string	监听用户单击页面内转发按钮，只有带上该参数，才支持快速分享	-
shareType	string	指定分享的类型	"miniapp"

版权所有：腾讯云计算（北京）有限责任公司

shareTarget参数说明：

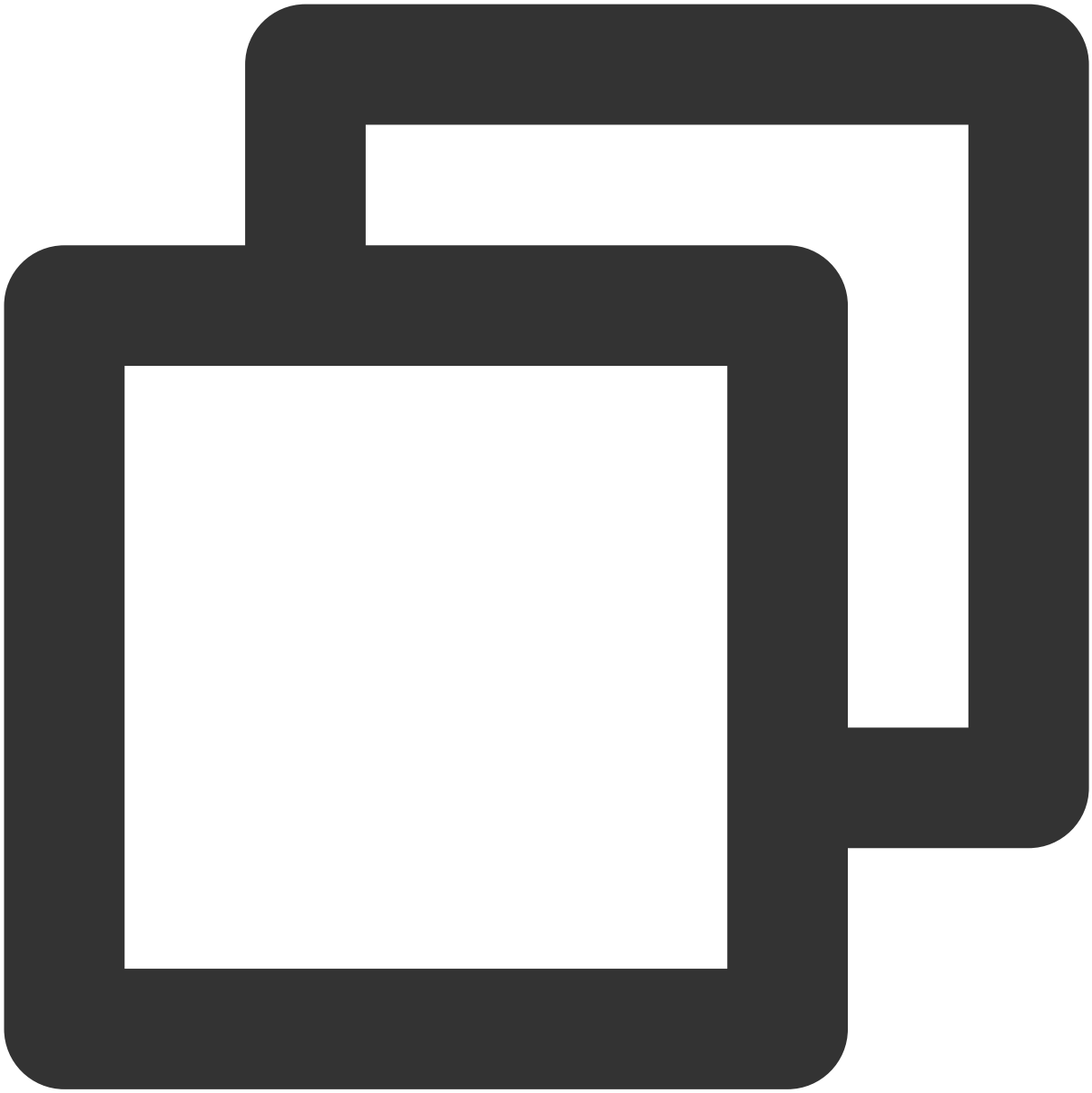
取值	说明
0	分享到 QQ 好友
1	分享到 QQ 空间
2	从当前聊天窗口打开，快速分享到当前聊天窗口
3	分享到微信好友
4	分享到微信朋友圈
5	分享面板分享到最近联系人
6	分享到快捷分享好友列表

shareType 参数说明
：
用来指定分享的类型，目前分享有两种，一种是分享小程序，另外一种是分享一张图片。

值	说明
"miniapp"	以小程序的形式分享，title、path、imageUrl、entryDataHash 参数会生效
"picture"	以图片的形式分享，imageUrl、entryDataHash 参数会生效

示例代码：

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  onShareAppMessage(res) {
    if (res.from === 'button') {
      // 来自页面内转发按钮
      console.log(res.target)
    }
    return {
      title: '自定义转发标题',
      path: '/page/user?id=123'
    }
  }
```

版权所有：腾讯云计算（北京）有限责任公司

```
})
```

onCustomBack()

监听用户返回上一页事件。

需要在app.json的 window 选项中或 页面配置 中开启customNavigateBack。

单击 tabBar 返回或手势侧滑或 back 键返回均会触发该事件。

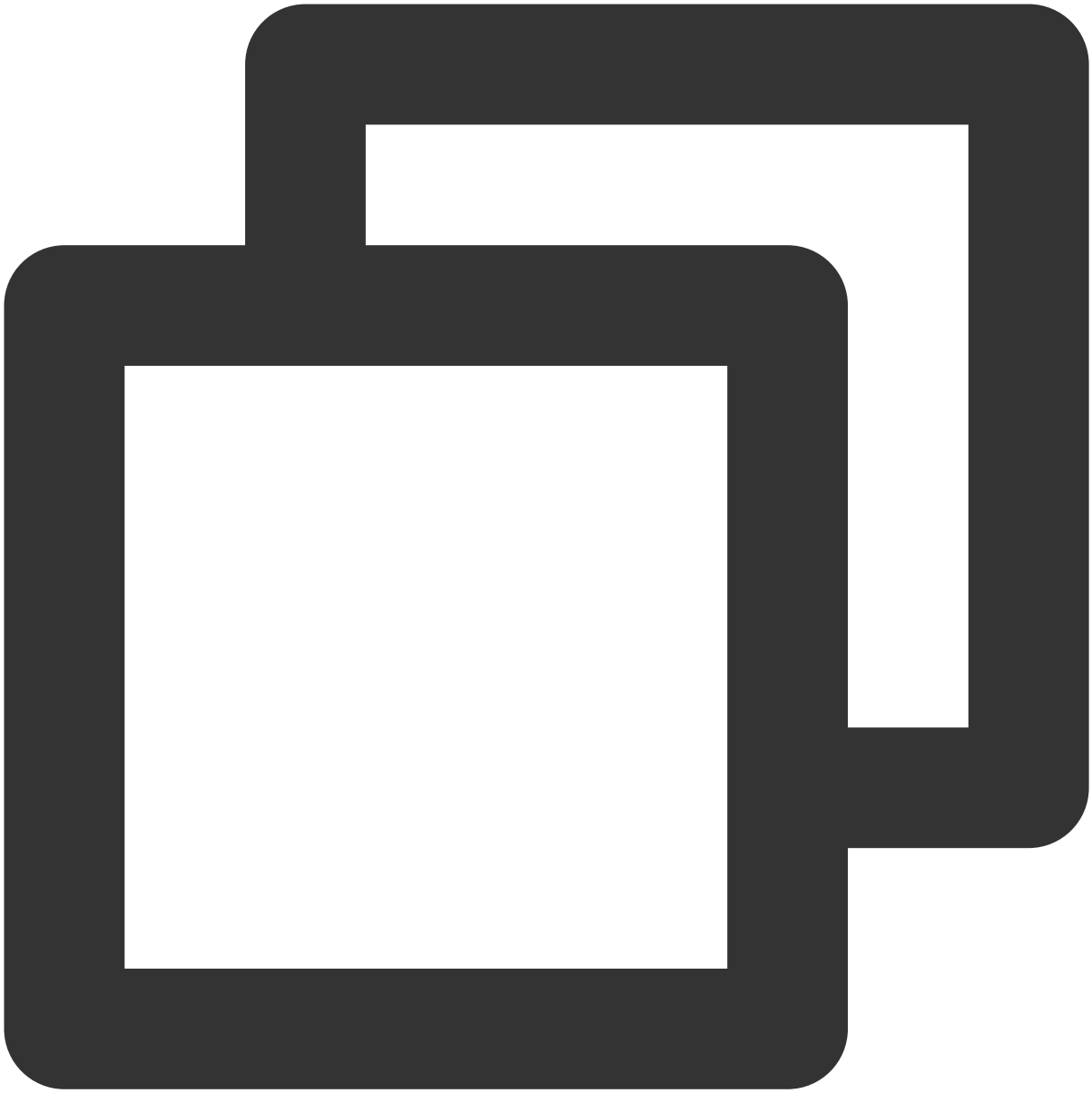
onTabItemTap(Object)

单击 tab 时触发。

Object 参数说明：

参数	类型	说明
index	String	被单击 tabItem 的序号，从0开始
pagePath	String	被单击 tabItem 的页面路径
text	String	被单击 tabItem 的按钮文字

示例代码：

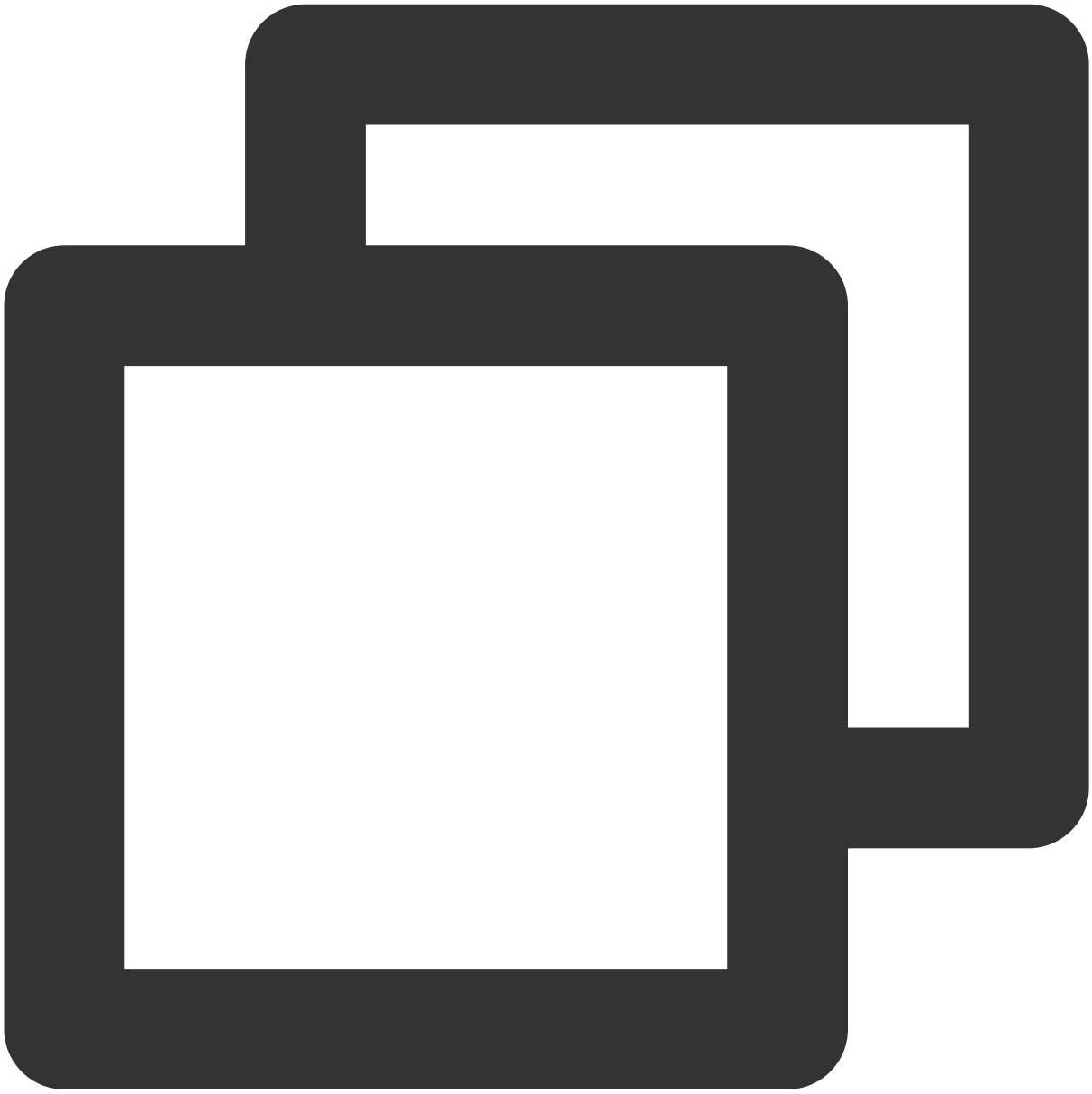


```
Page({
  onTabItemTap(item) {
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  }
})
```

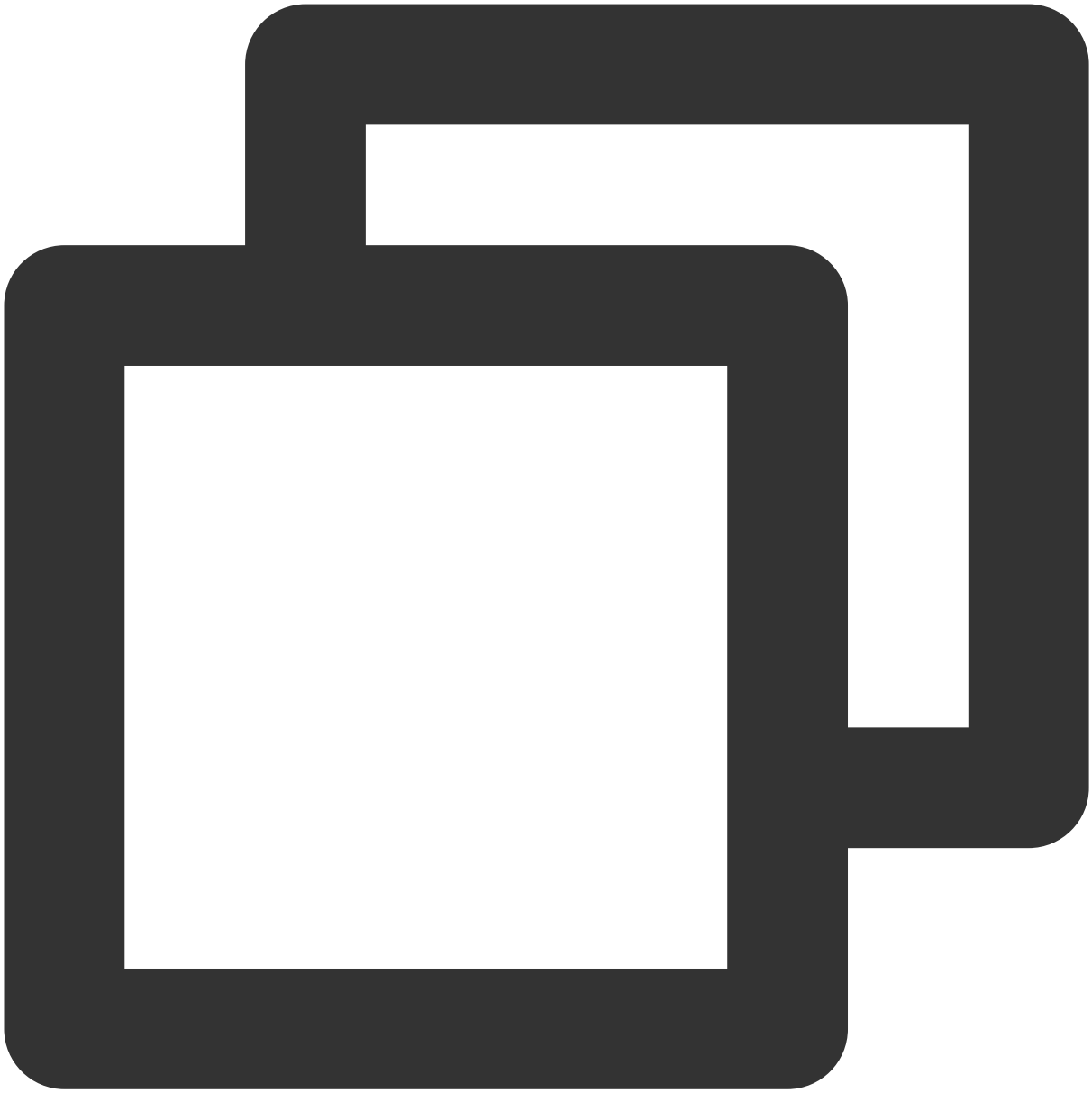
组件事件处理函数

`Page` 中还可以定义组件事件处理函数。在渲染层的组件中加入 事件绑定，当事件被触发时，就会执行 Page 中定义的事件处理函数。

示例代码：



```
<view bindtap="viewTap"> click me </view>
```

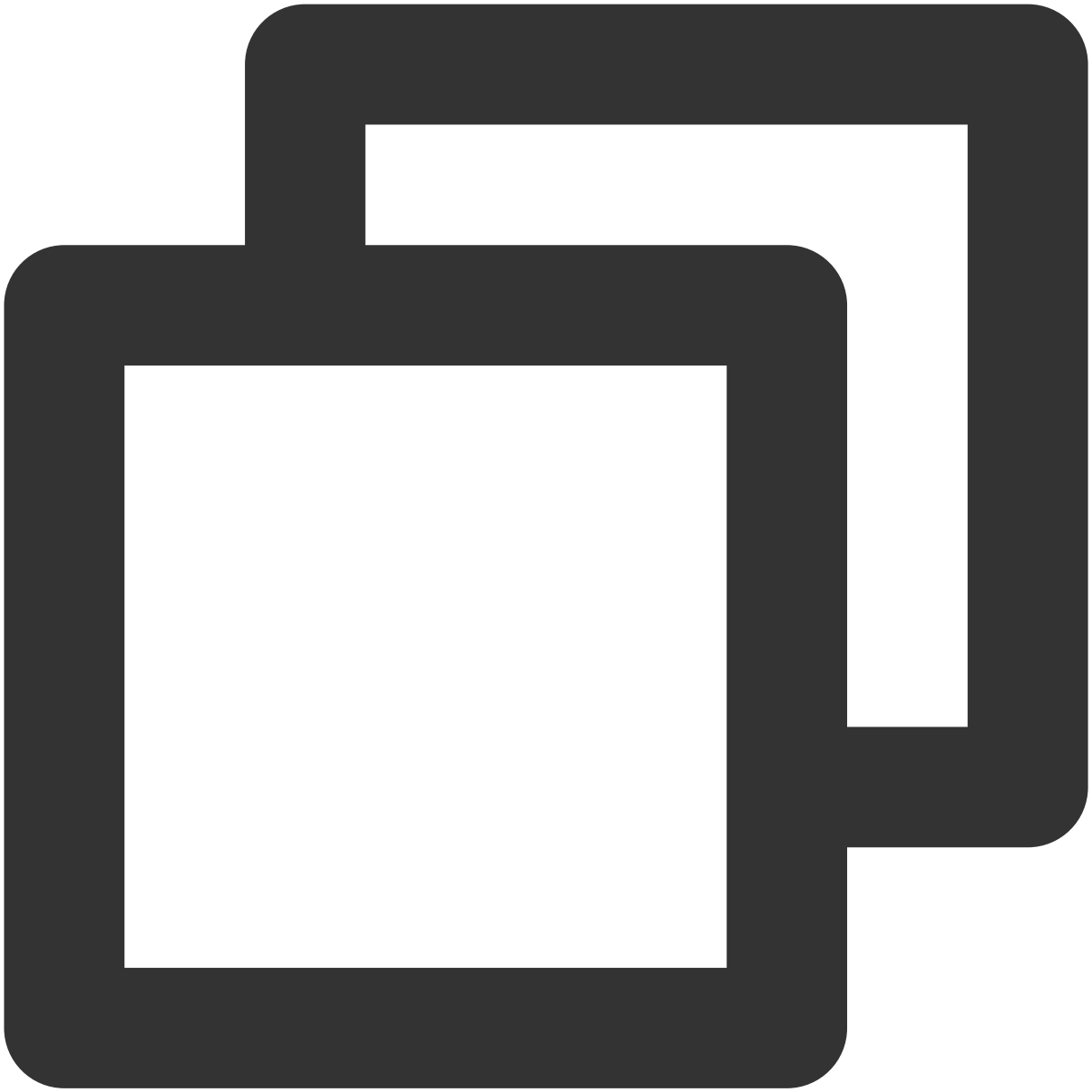


```
Page({
  viewTap: function() {
    console.log('view tap')
  }
})
```

Page.route

到当前页面的路径，类型为 String。

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  viewTap: function() {
    console.log('view tap')
  }
})
```

Page.prototype.setData(Object data, Function callback)

`setData` 函数用于将数据从逻辑层发送到渲染层（异步），同时改变对应的 `this.data` 的值（同步）。

参数:

字段	类型	必填	描述
data	Object	是	此次要改变的数据
callback	Function	否	setData 引起的界面更新渲染完毕后的回调函数

`Object` 以 `key: value` 的形式表示，将 `this.data` 中的 `key` 对应的值改变成 `value` 。

其中 `key` 可以以数据路径的形式给出，支持改变数组中的某一项或对象的某个属性，如 `array[2].message` ， `a.b.c.d` ，并且不需要在 this.data 中预先定义。

注意：

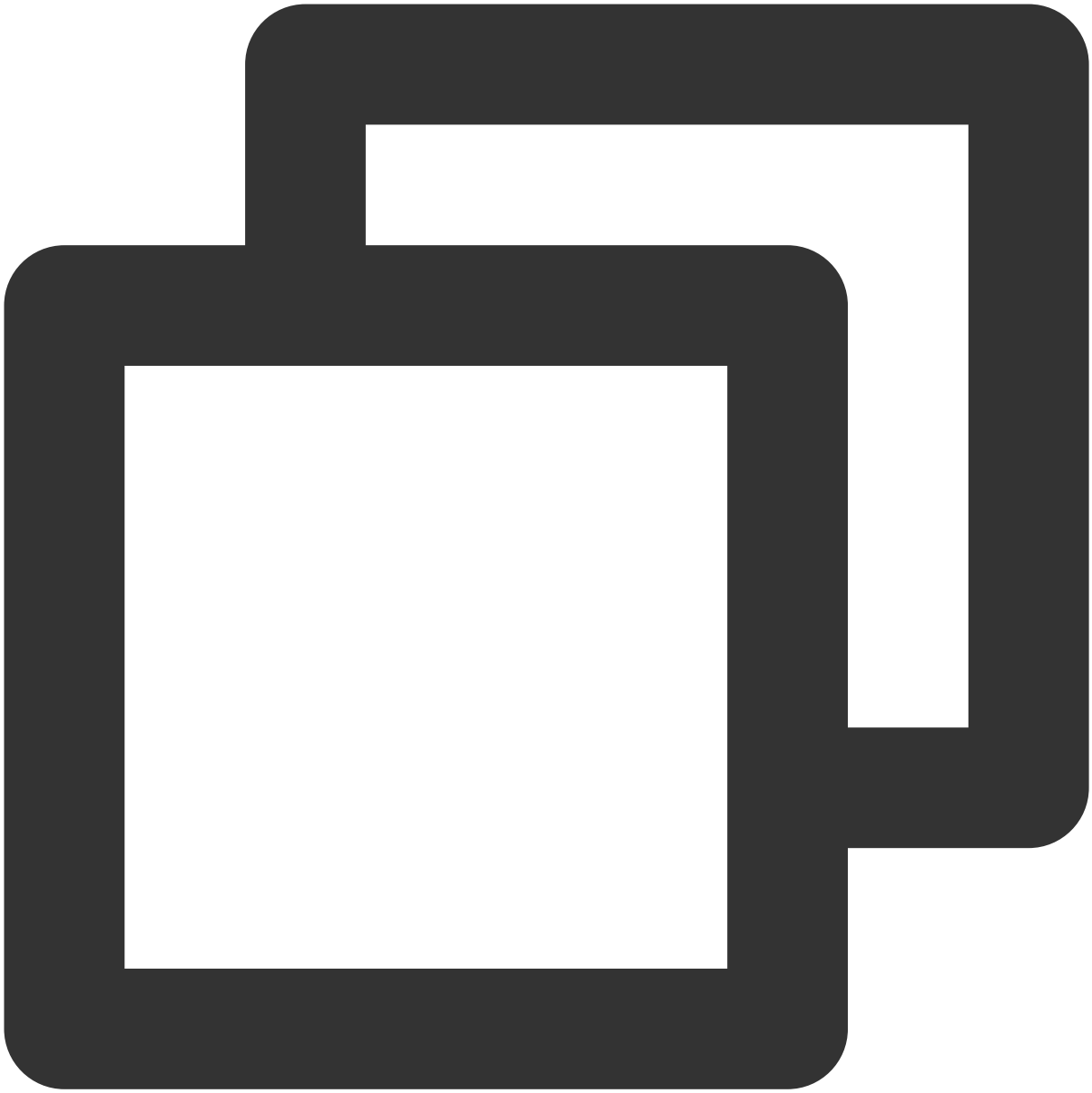
直接修改 this.data 而不调用 this.setData 是无法改变页面的状态的，还会造成数据不一致。

仅支持设置可 JSON 化的数据。

单次设置的数据不能超过1024kB，请尽量避免一次设置过多的数据。

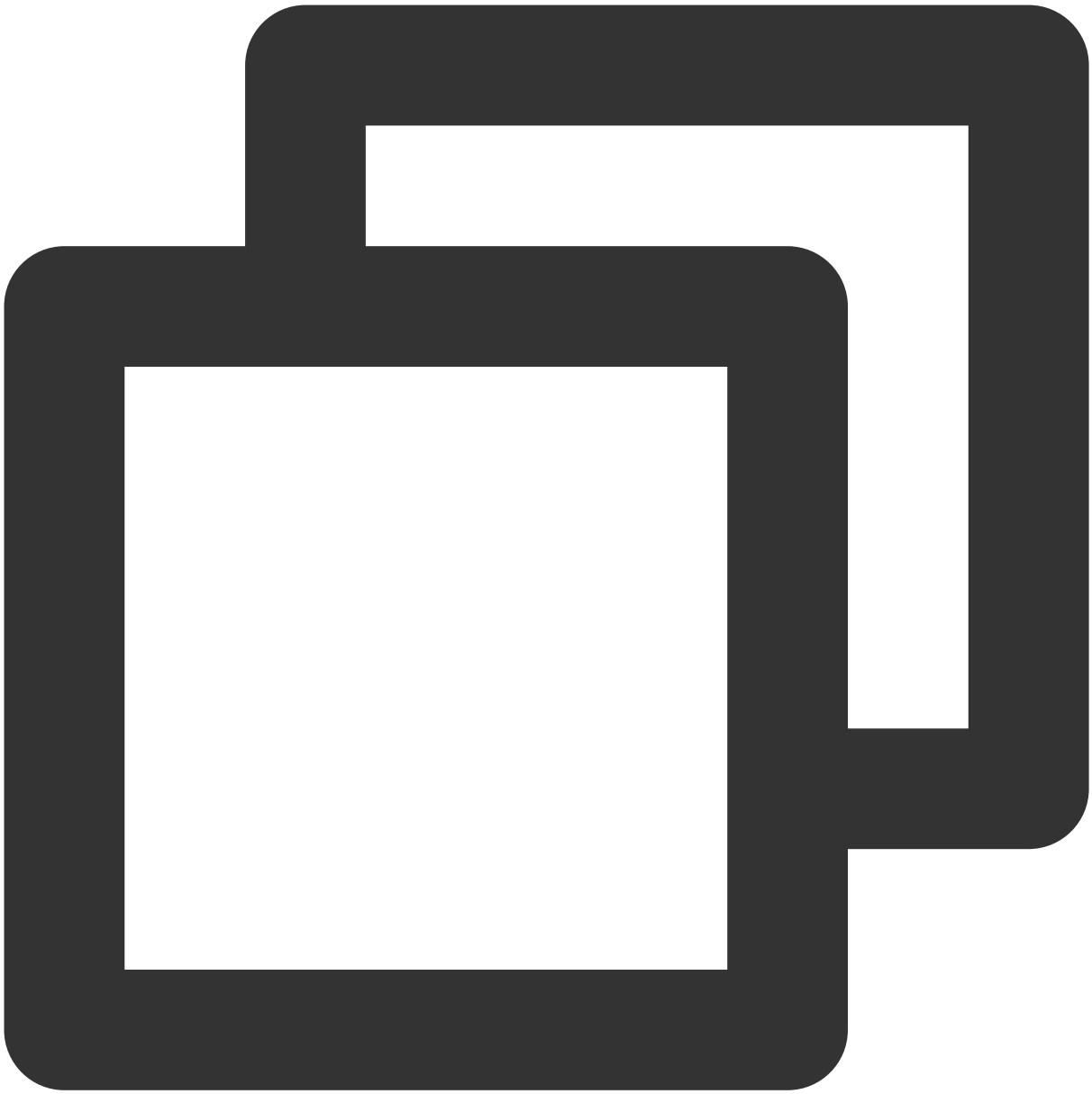
请不要把 data 中任何一项的 value 设为 `undefined` ，否则这一项将不被设置并可能遗留一些潜在问题。

版权所有：腾讯云计算（北京）有限责任公司



```
<!--index.wxml-->
<view>{{text}}</view>
<button bindtap="changeText"> Change normal data </button>
<view>{{num}}</view>
<button bindtap="changeNum"> Change normal num </button>
<view>{{array[0].text}}</view>
<button bindtap="changeItemInArray"> Change Array data </button>
<view>{{object.text}}</view>
<button bindtap="changeItemInObject"> Change Object data </button>
<view>{{newField.text}}</view>
<button bindtap="addNewField"> Add new data </button>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// index.js
Page({
  data: {
    text: 'init data',
    num: 0,
    array: [{text: 'init data'}],
    object: {
      text: 'init data'
    }
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  changeText: function() {
    // this.data.text = 'changed data' // 不要直接修改 this.data
    // 应该使用 setData
    this.setData({
      text: 'changed data'
    })
  },
  changeNum: function() {
    // 或者，可以修改 this.data 之后马上用 setData 设置一下修改了的字段
    this.data.num = 1
    this.setData({
      num: this.data.num
    })
  },
  changeItemInArray: function() {
    // 对于对象或数组字段，可以直接修改一个其下的子字段，这样做通常比修改整个对象或数组更好
    this.setData({
      'array[0].text':'changed data'
    })
  },
  changeItemInObject: function(){
    this.setData({
      'object.text': 'changed data'
    });
  },
  addNewField: function() {
    this.setData({
      'newField.text': 'new data'
    })
  }
})
```

PageObject[] getCurrentPages()

获取当前页面栈。数组中第一个元素为首页，最后一个元素为当前页面。

注意事项

不要尝试修改页面栈，会导致路由以及页面状态错误。

不要在 `App.onLaunch` 的时候调用 `getCurrentPages()`，此时 `page` 还没有生成。

自定义组件

最近更新时间：2024-07-12 18:38:20

Component

创建自定义组件，接受一个 Object 类型的参数。

参数

Object object

定义段	类型	是否必填	描述
properties	Object Map	否	组件的对外属性，是属性名到属性设置的映射表
data	Object	否	组件的内部数据，和 properties 一同用于组件的模板渲染
observers	Object	否	组件数据字段监听器，用于监听 properties 和 data 的变化，参见 数据监听器
methods	Object	否	组件的方法，包括事件响应函数和任意的自定义方法，关于事件响应函数的使用，参见 组件间通信与事件
behaviors	String Array	否	类似于 mixins 和 traits 的组件间代码复用机制，参见 behaviors
created	Function	否	组件生命周期函数-在组件实例刚刚被创建时执行，注意此时不能调用 setData
attached	Function	否	组件生命周期函数-在组件实例进入页面节点树时执行
ready	Function	否	组件生命周期函数-在组件布局完成后执行
detached	Function	否	组件生命周期函数-在组件实例被从页面节点树移除时执行
relations	Object	否	组件间关系定义，参见 组件间关系
externalClasses	String Array	否	组件接受的外部样式类，参见 外部样式类
options	Object Map	否	一些选项（文档中介绍相关特性时会涉及具体的选项设置，这里暂不列举）

版权所有：腾讯云计算（北京）有限责任公司

lifetimes	Object	否	组件生命周期声明对象，参见 组件生命周期
pageLifetimes	Object	否	组件所在页面的生命周期声明对象，参见 组件生命周期

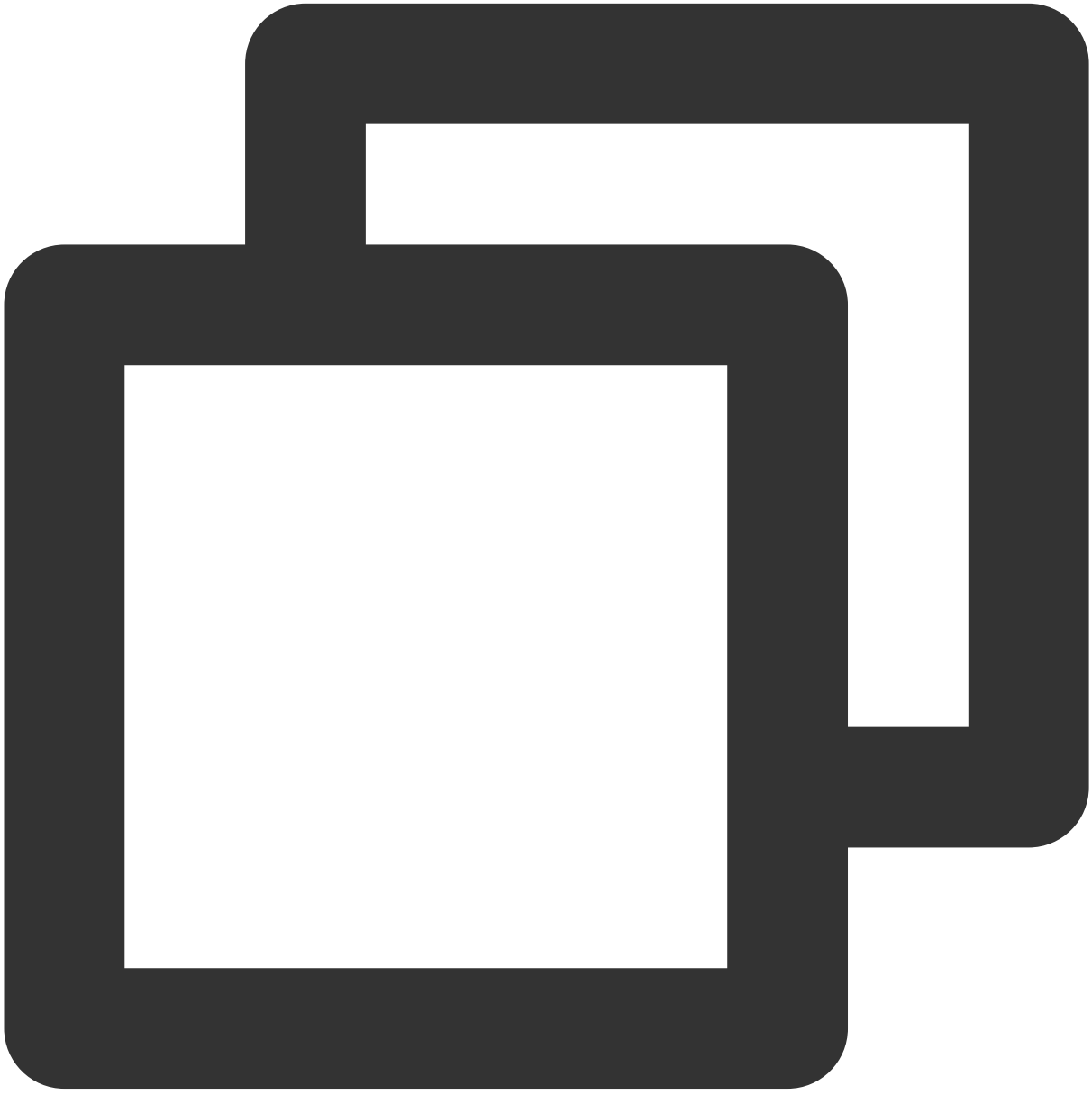
生成的组件实例可以在组件的方法、生命周期函数和属性 `observer` 中通过 `this` 访问。组件包含一些通用属性和方法。

属性名	类型	描述
is	String	组件的文件路径
id	String	节点 id
dataset	String	节点 dataset
data	Object	组件数据，**包括内部数据和属性值**
properties	Object	组件数据，**包括内部数据和属性值**（与 `data` 一致）

方法名	参数	描述
setData	Object `newData`	设置 data 并执行视图层渲染
triggerEvent	String `name`，Object `detail`，Object `options`	触发事件，参见 组件间通信与事件
createSelectorQuery	-	创建一个 SelectorQuery 对象，选择器选取范围为这个组件实例内
createIntersectionObserver	-	创建一个 IntersectionObserver 对象，选择器选取范围为这个组件实例内
selectComponent	String `selector`	使用选择器选择组件实例节点，返回匹配到的第一个组件实例对象（会被 `wx://component-export` 影响）
selectAllComponents	String `selector`	使用选择器选择组件实例节点，返回匹配到的全部组件实例对象组成的数组（会被 `wx://component-export` 影响）
selectownerComponent	String `selector`	使用选择器选择组件实例节点，返回匹配到的全部组件实例对象组成的数组（会被 `wx://component-export` 影响）
getRelationNodes	String	获取这个关系所对应的所有关联节点，参见 组件间关

	`relationKey`	系
hasBehavior	Object `behavior`	检查组件是否具有 `behavior` （检查时会递归检查被直接或间接引入的所有behavior）

示例代码：



```
Component({
  behaviors: [],

  // 属性定义（详情参见下文）
```

```
  properties: {
    myProperty: { // 属性名
      type: String,
      value: ''
    },
    myProperty2: String // 简化的定义方式
  },

  data: {}, // 私有数据，可用于模板渲染

  lifetimes: {
    // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
    attached: function () { },
    detached: function () { },
  },

  // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
  attached: function () { }, // 此处attached的声明会被lifetimes字段中的声明覆盖
  ready: function() { },

  pageLifetimes: {
    // 组件所在页面的生命周期函数
    show: function () { },
    hide: function () { },
  },

  methods: {
    onMyButtonTap: function(){
      this.setData({
        // 更新属性和数据的方法与更新页面数据的方法类似
      })
    },
    // 内部方法建议以下划线开头
    _myPrivateMethod: function(){
      // 这里将 data.A[0].B 设为 'myPrivateData'
      this.setData({
        'A[0].B': 'myPrivateData'
      })
    },
    _propertyChange: function(newVal, oldVal) {

    }
  }
})
```

properties

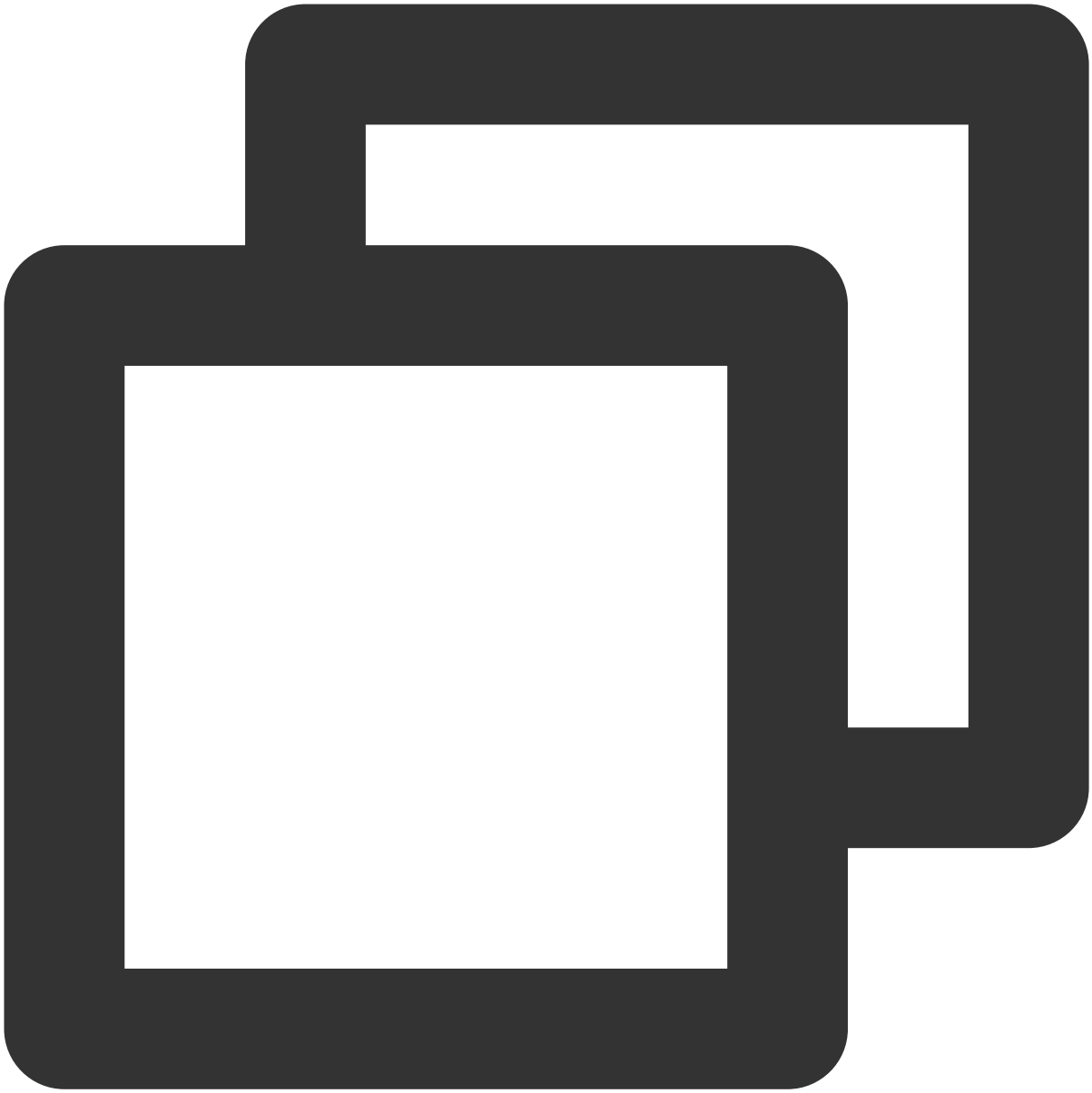
定义段	类型	是否必填	描述
type	-	是	属性的类型
value	-	否	属性的初始值
observer	Function	否	属性值变化时的回调函数

属性值的改变情况可以使用 observer 来监听。目前，在新版本基础库中不推荐使用这个字段，而是使用 Component 构造器的 `observers` 字段代替，它更加强大且性能更好。

注意：

定义段中的 `type` 字段为 **必填** 项。

示例代码：



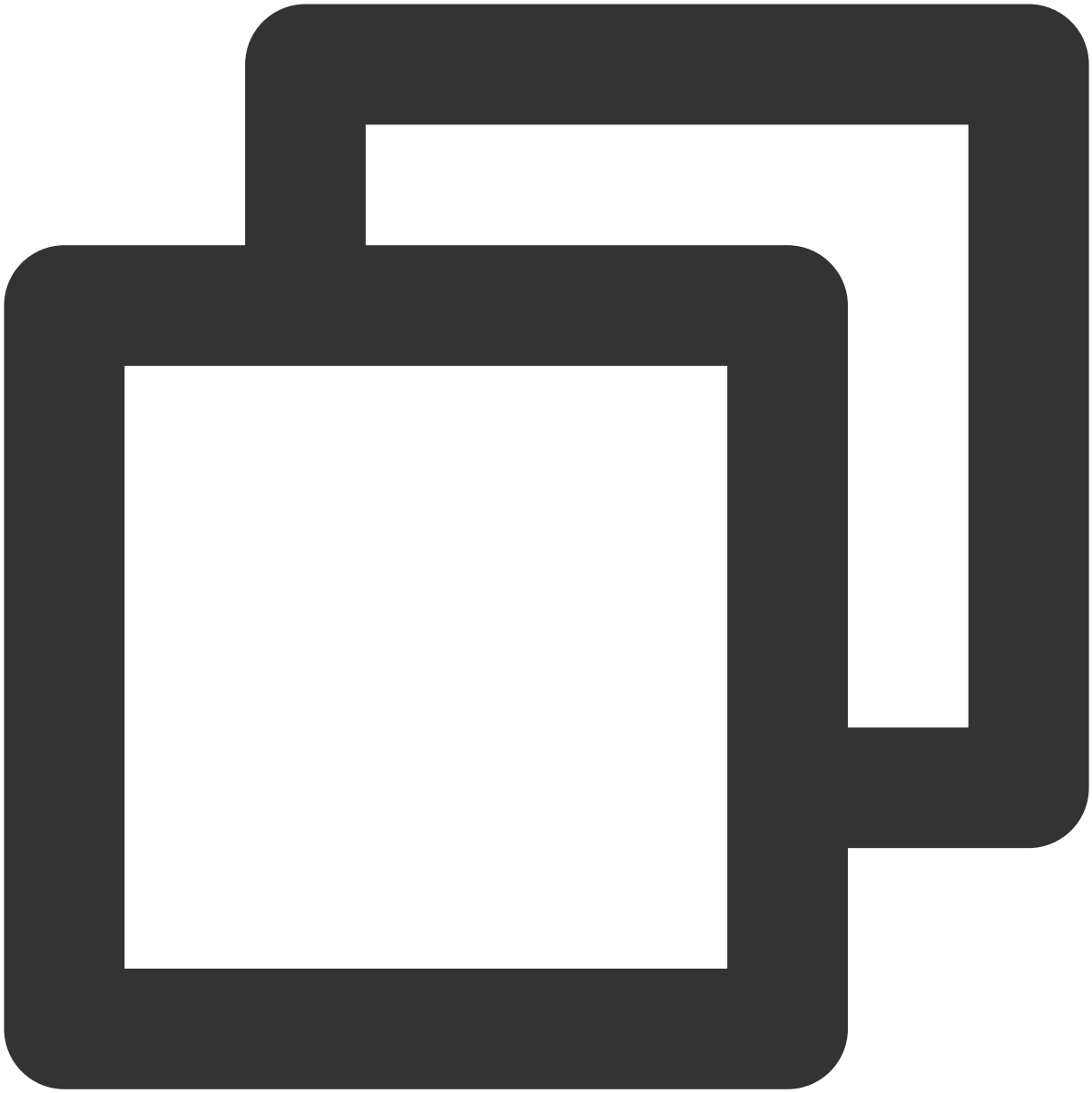
```
Component({
  properties: {
    min: {
      type: Number,
      value: 0
    },
    max: {
      type: Number,
      value: 0,
      observer: function(newVal, oldVal) {
        // 属性值变化时执行
```

版权所有：腾讯云计算（北京）有限责任公司

```
        }
      },
      lastLeaf: {
        // 这个属性可以是 Number 、 String 、 Boolean 三种类型中的一种
        type: Number,
        optionalTypes: [String, Object],
        value: 0
      }
    }
  })
```

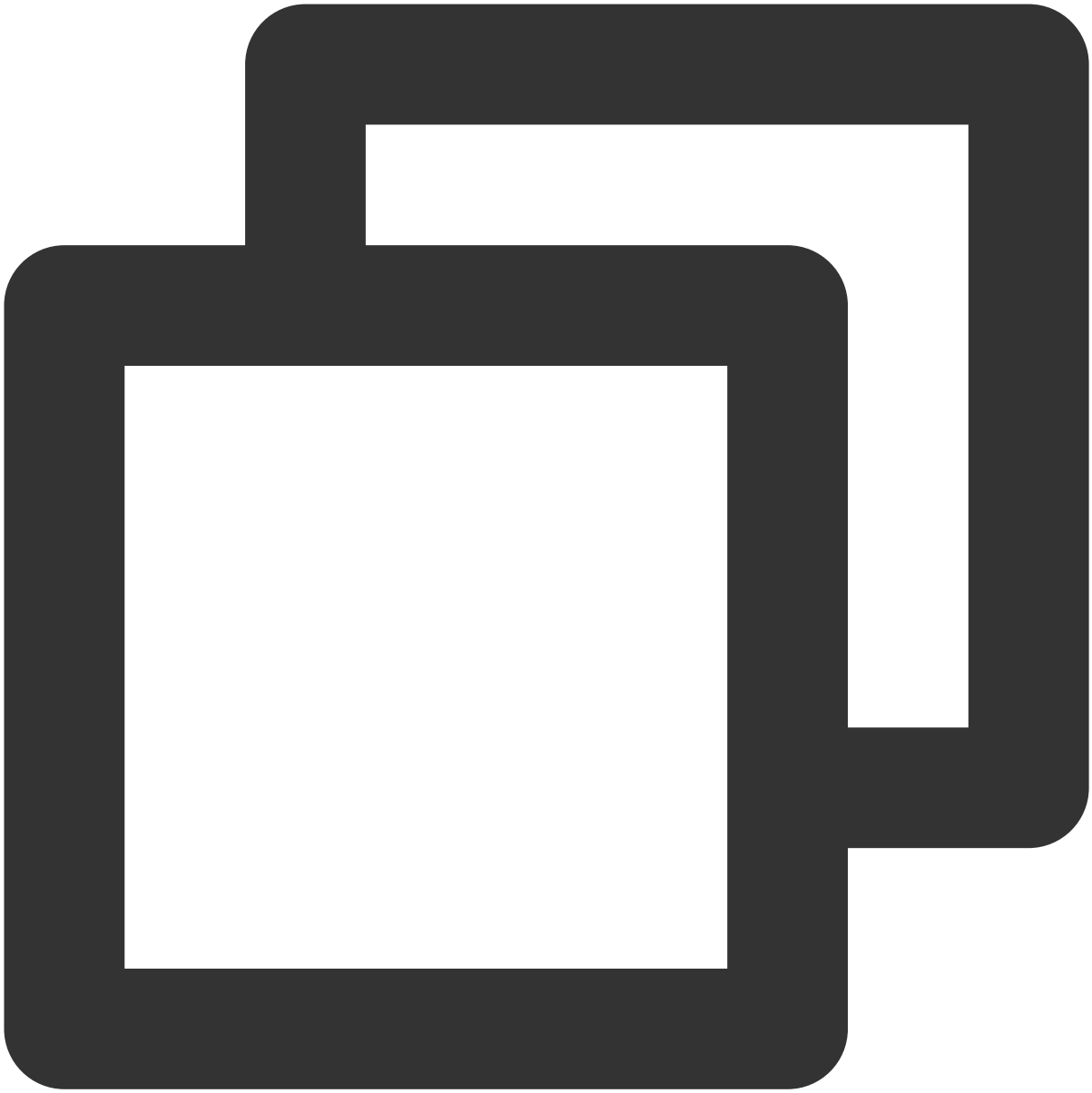
属性的类型可以为 `String` `Number` `Boolean` `Object` `Array` 其一，也可以为 `null` 表示不限制类型。

多数情况下，属性最好指定一个确切的类型。这样，在 WXML 中以字面量指定属性值时，值可以获得一个确切的类型，如：



```
<custom-comp min="1" max="5" />
```

此时，由于自定义组件的对应属性被规定为 `Number` 类型， `min` 和 `max` 会被赋值为 `1` 和 `5` ，而非 `"1"` 和 `"5"` ，即：



```
this.data.min === 1 // true
this.data.max === 5 // true
```

tips

使用 `this.data` 可以获取内部数据和属性值；但直接修改它不会将变更应用到界面上，应使用 `setData` 修改；

生命周期函数无法在组件方法中通过 `this` 访问到；

属性名应避免以 `data` 开头，即不要命名成 `dataXyz` 这样的形式，因为在 WXML 中， `data-xyz=""` 会被作为节点 `dataset` 来处理，而不是组件属性；
在一个组件的定义和使用时，组件的属性名和 `data` 字段相互间都不能冲突（尽管它们位于不同的定义段中）。

Behavior

注册一个 `behavior` ，接受一个 `Object` 类型的参数。

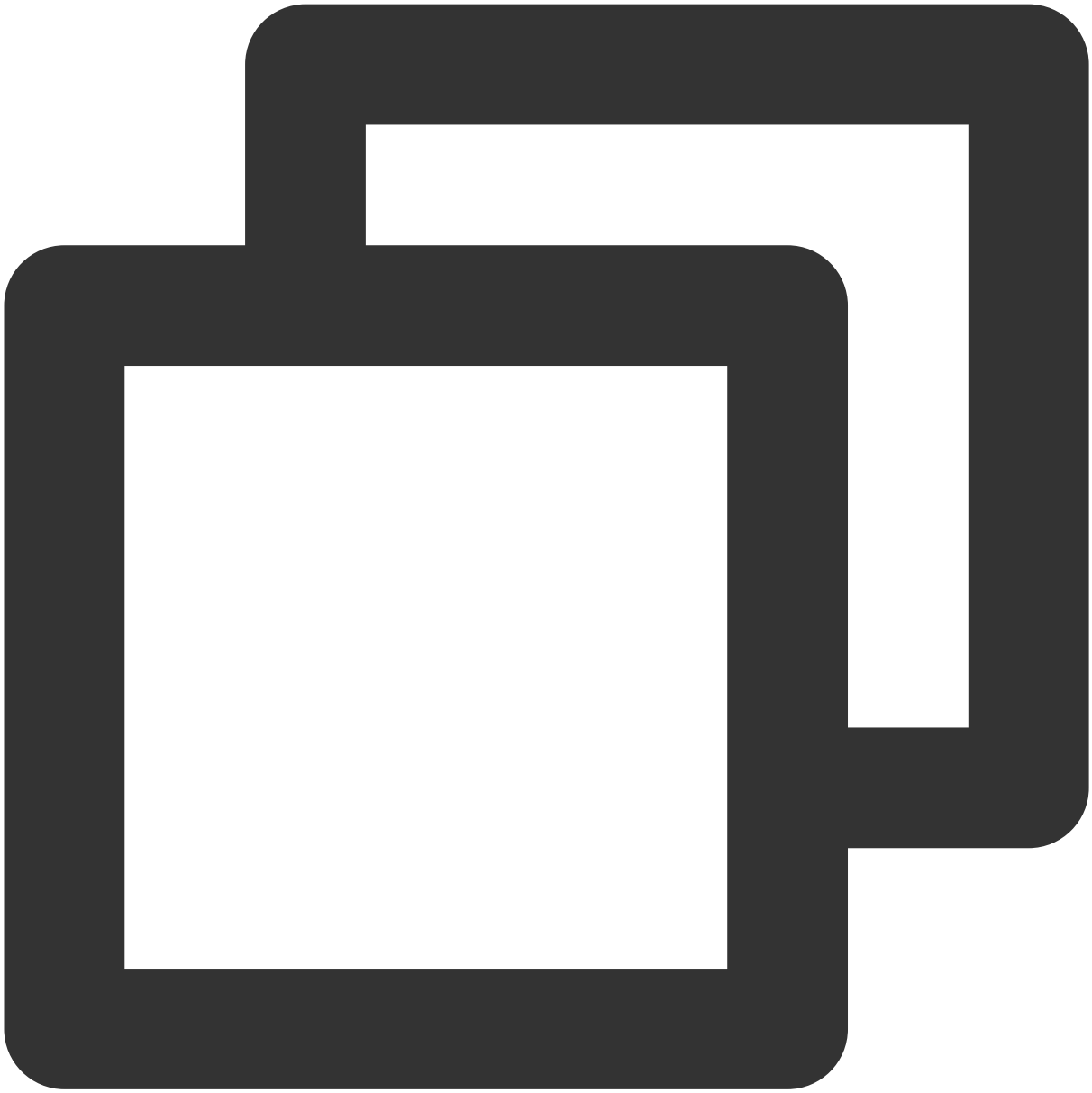
参数

Object object

定义段	类型	是否必填	描述
properties	Object Map	否	同组件的属性
data	Object	否	同组件的数据
methods	Object	否	同自定义组件的方法
behaviors	String Array	否	类似于 mixins 和 traits 的组件间代码复用机制，参见 behaviors
created	Function	否	组件生命周期函数-在组件实例刚刚被创建时执行，注意此时不能调用 `setData`
attached	Function	否	组件生命周期函数-在组件实例进入页面节点树时执行
ready	Function	否	组件生命周期函数-在组件布局完成后执行
detached	Function	否	组件生命周期函数-在组件实例被从页面节点树移除时执行
relations	Object	否	组件间关系定义，参见 组件间关系
lifetimes	Object	否	组件生命周期声明对象，参见 组件生命周期
pageLifetimes	Object	否	组件所在页面的生命周期声明对象，参见 组件生命周期

代码示例：

版权所有：腾讯云计算（北京）有限责任公司



```
// my-behavior.js
module.exports = Behavior({
  behaviors: [],
  properties: {
    myBehaviorProperty: {
      type: String
    }
  },
  data: {
    myBehaviorData: {}
  },
```

版权所有：腾讯云计算（北京）有限责任公司

```
  attached: function(){},
  methods: {
    myBehaviorMethod: function(){}
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

模块化

最近更新时间：2024-07-12 17:09:23

require

引入模块。返回模块通过 module.exports 或 `exports` 暴露的接口。
需要引入其他分包的模块的时候，可以通过配置 `callback` 回调函数来异步获取指定模块。异步获取失败的时候，将会触发 `error` 回调函数。

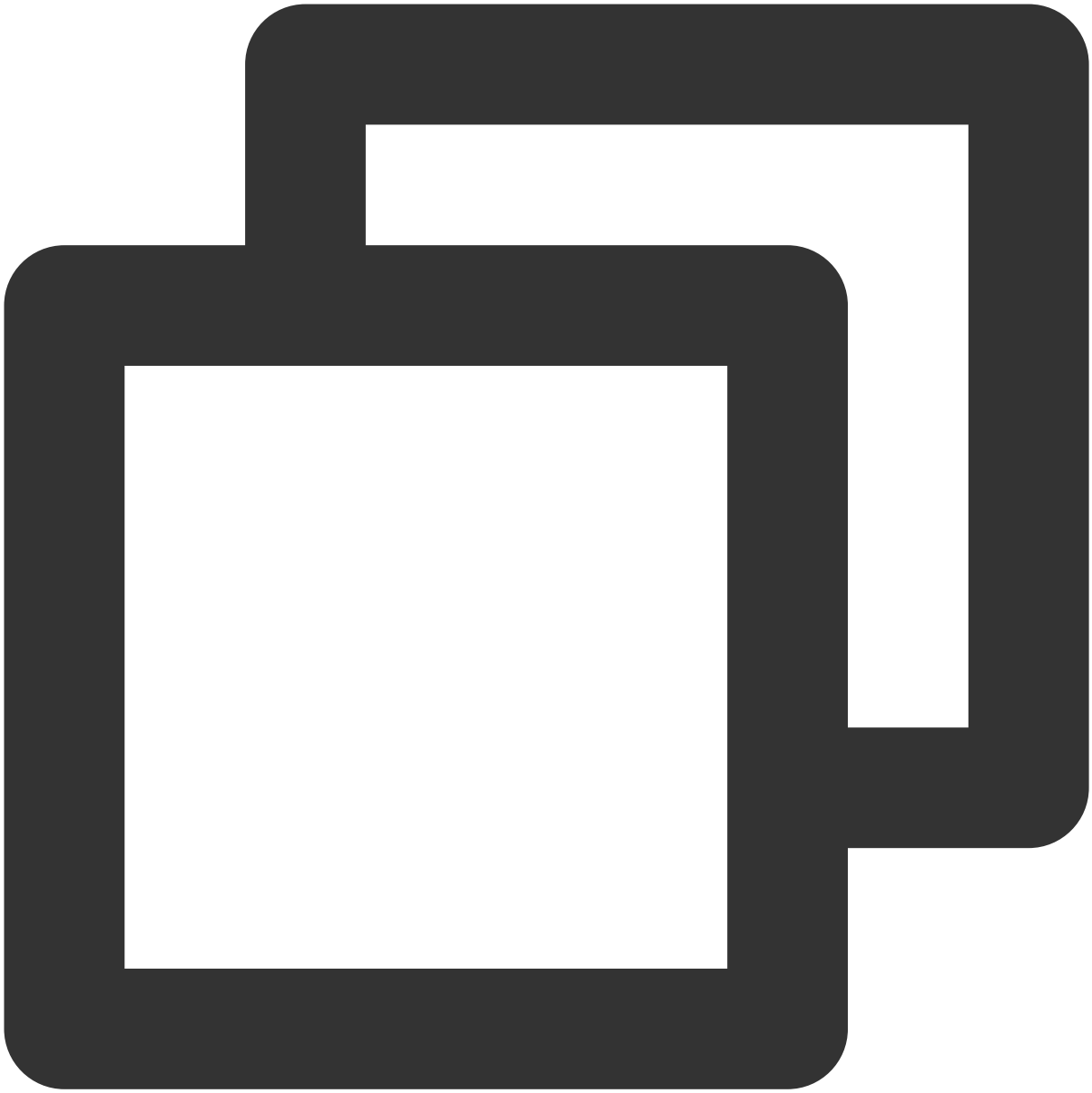
参数

名称	类型	必填	说明
path	string	是	需要引入模块文件相对于当前文件的相对路径，或npm模块名，或npm模块路径。默认不支持绝对路径，可通过配置 `resolveAlias` 自定义路径映射。
callback	function	否	异步加载成功回调函数，该回调函数参数为成功加载的模块。
error	function	否	异步加载失败回调函数，该回调函数参数为错误信息和模块名。

require.async 链式调用

可以通过链式调用的方式使用。

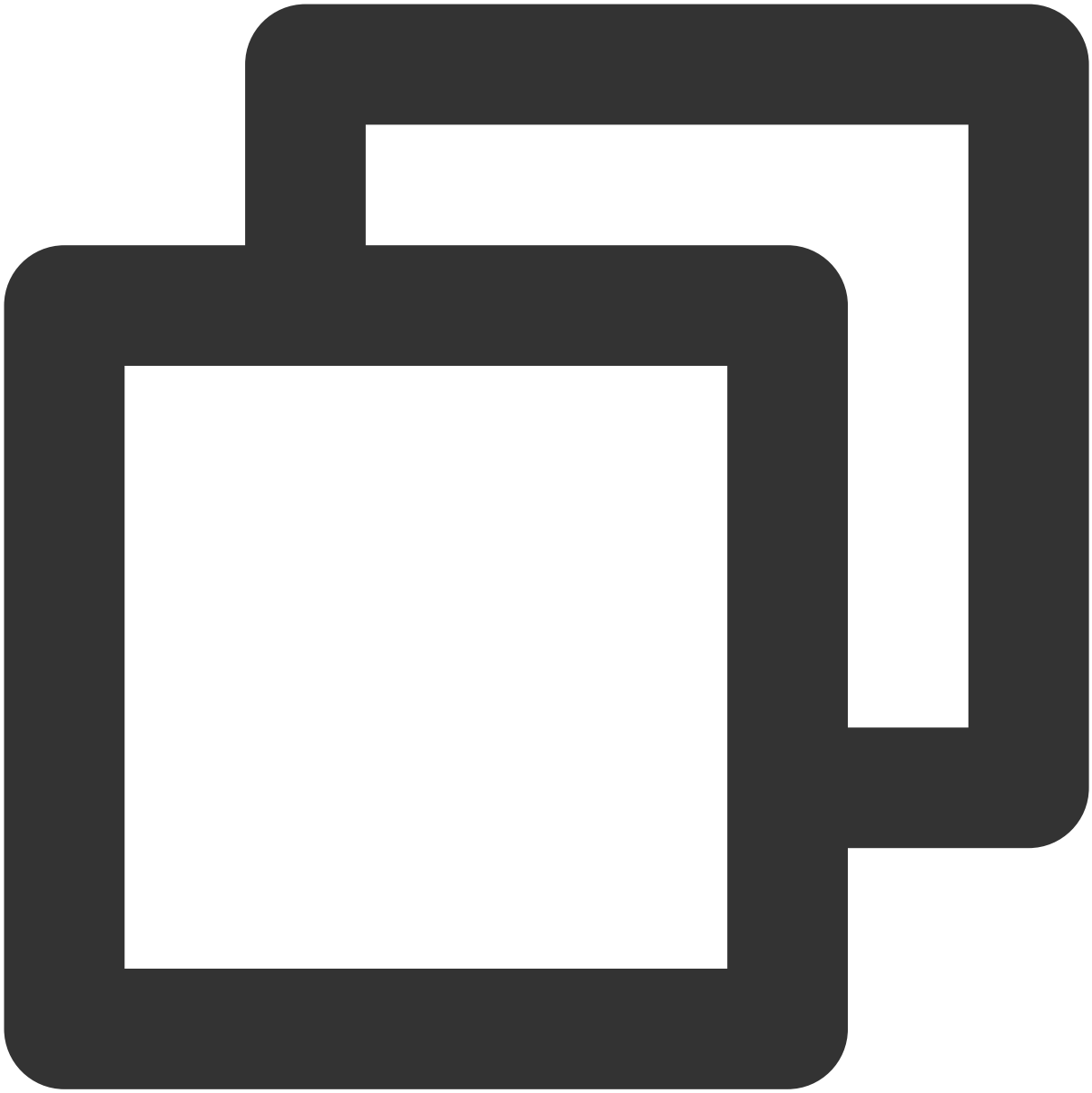
版权所有：腾讯云计算（北京）有限责任公司



```
require
    .async('path/to/mod')
    .then((mod) => {
        console.log(mod)
    })
    .catch(({ errMsg, mod }) => {
        console.error(`path: ${mod}, ${errMsg}`)
    })
```

示例代码

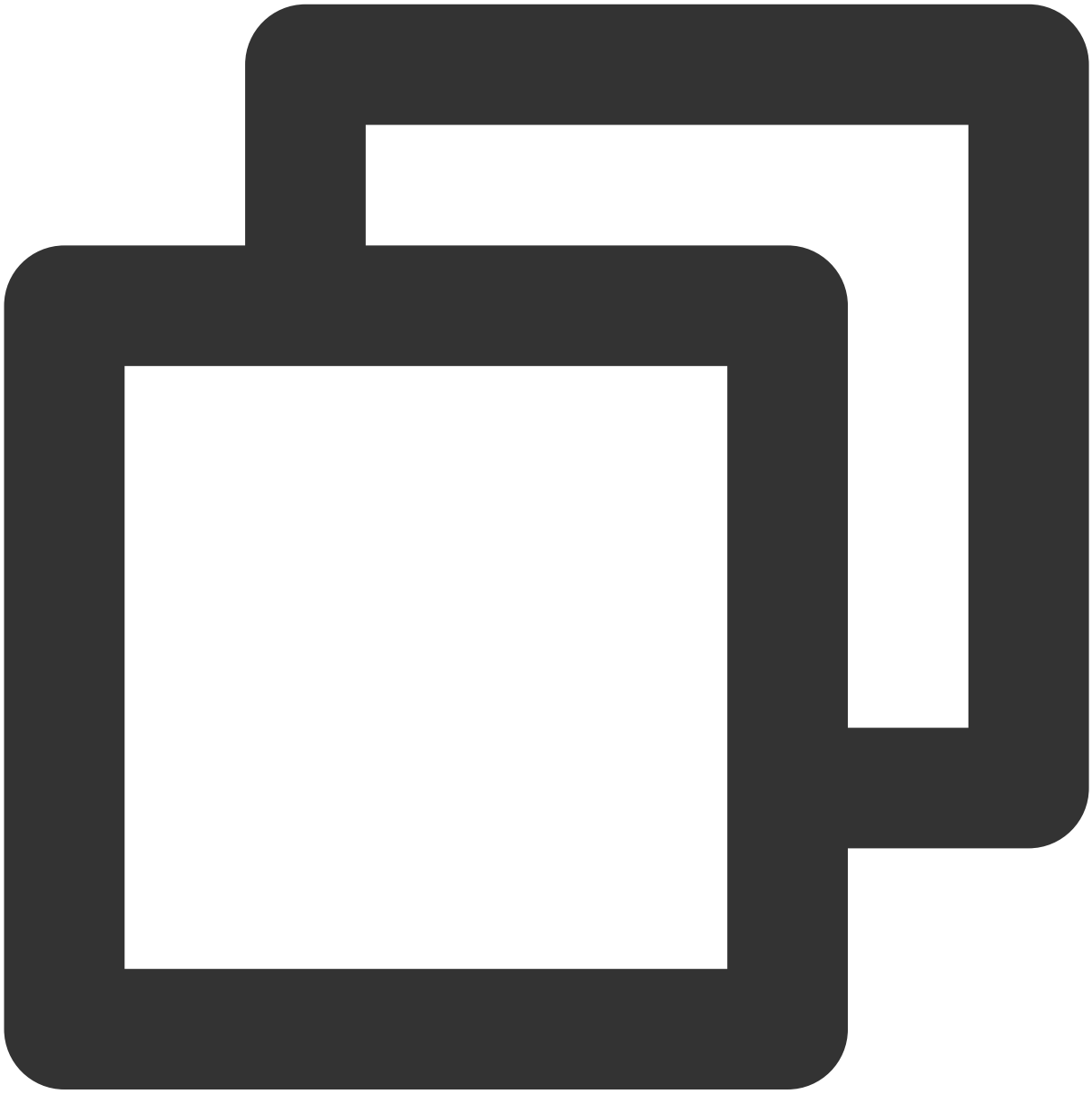
同一包内调用



```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

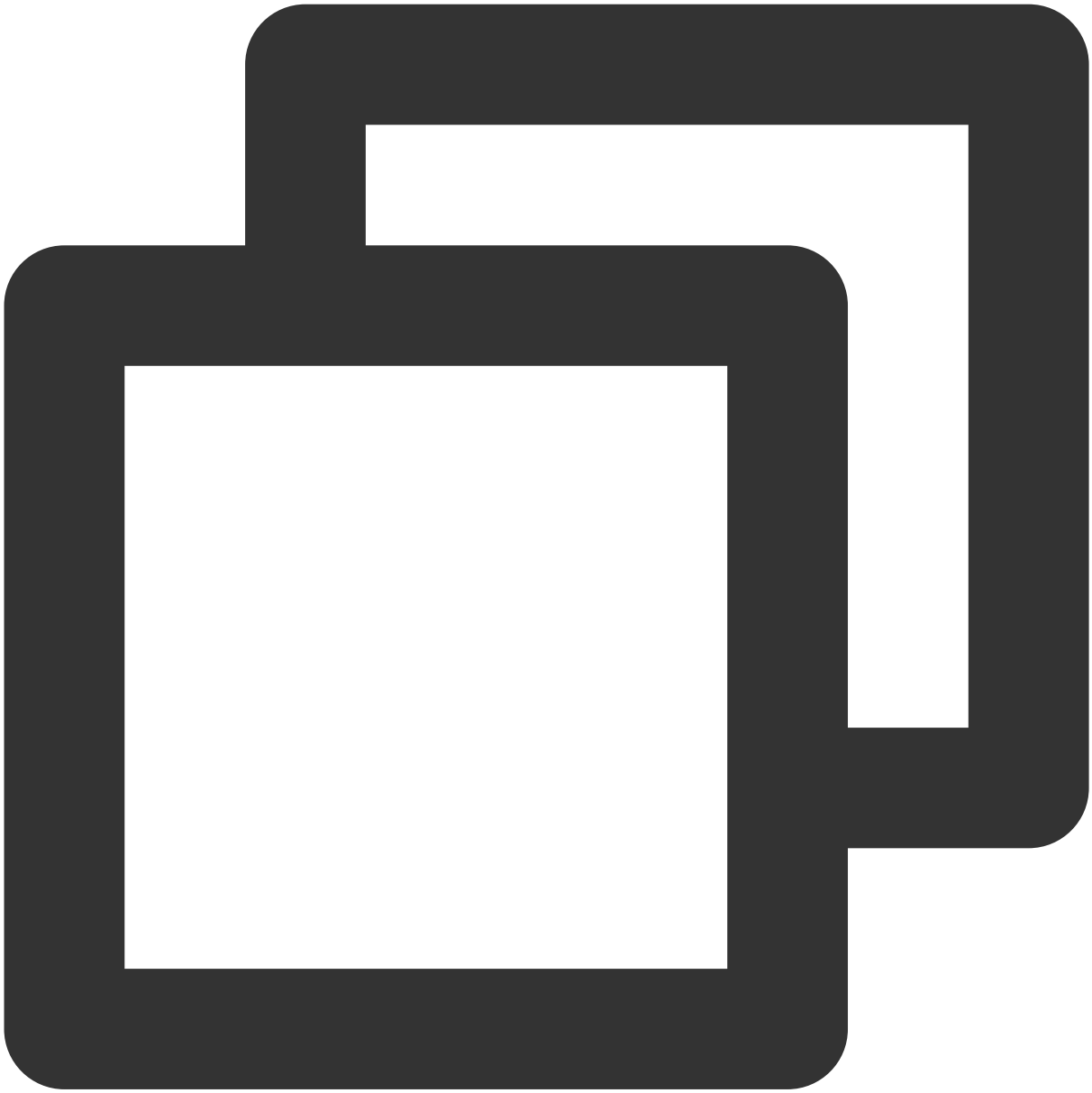


```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
```

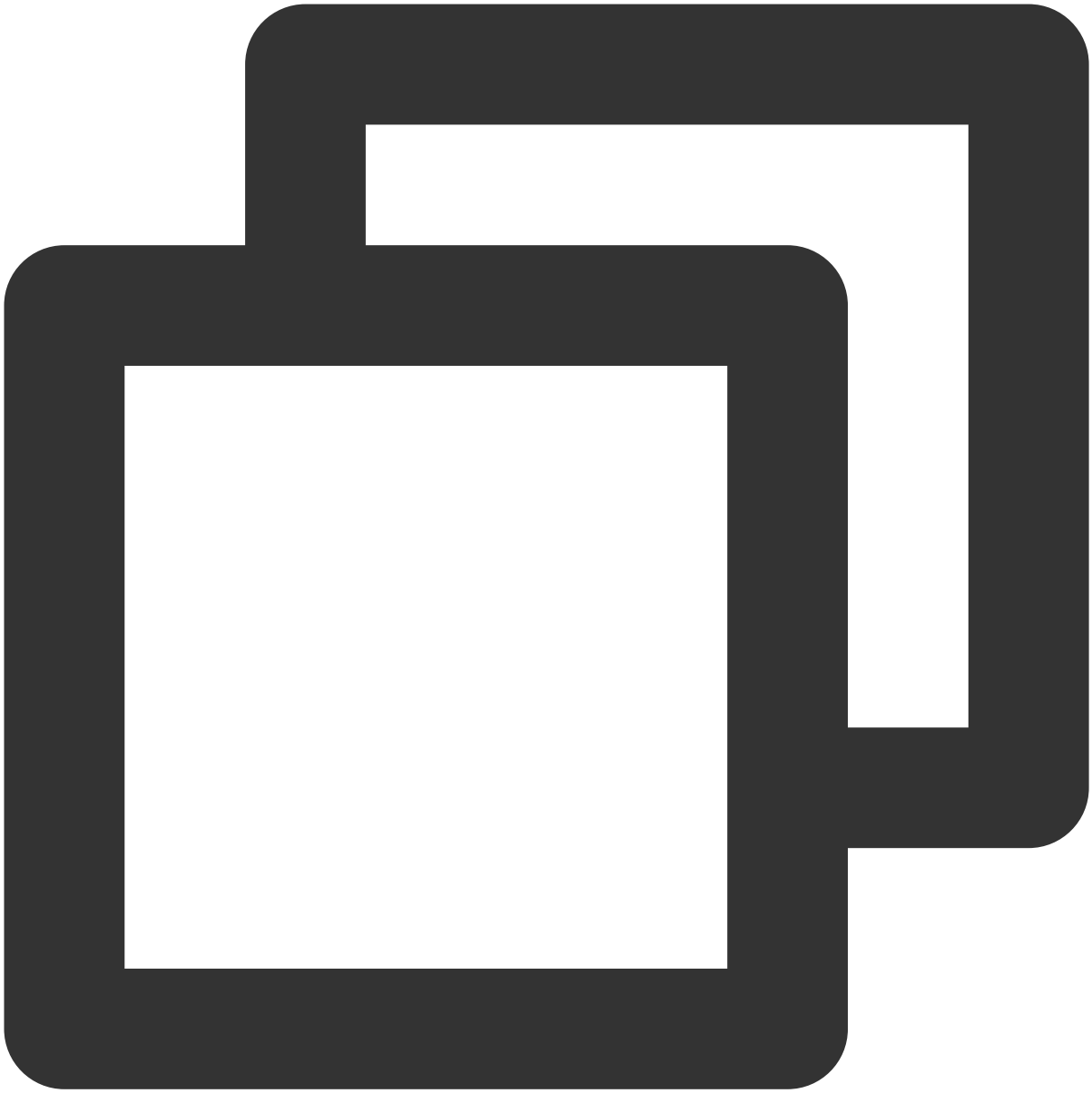
版权所有：腾讯云计算（北京）有限责任公司

```
})
```

跨分包异步调用



```
// subpackage/common.js 分包 common 文件
export const sayHello = () => console.log("hello")
```



```
// pages/index.js 主包页面
let common;
require('../../subpackage/common.js', (mod) => {
    common = mod
}, ({ errMsg, mod }) => {
    console.error(`path: ${mod}, ${errMsg}`)
})

Page({
    sayHello() {
        common && common.sayHello()
```

版权所有：腾讯云计算（北京）有限责任公司

```
    }
})
```

module

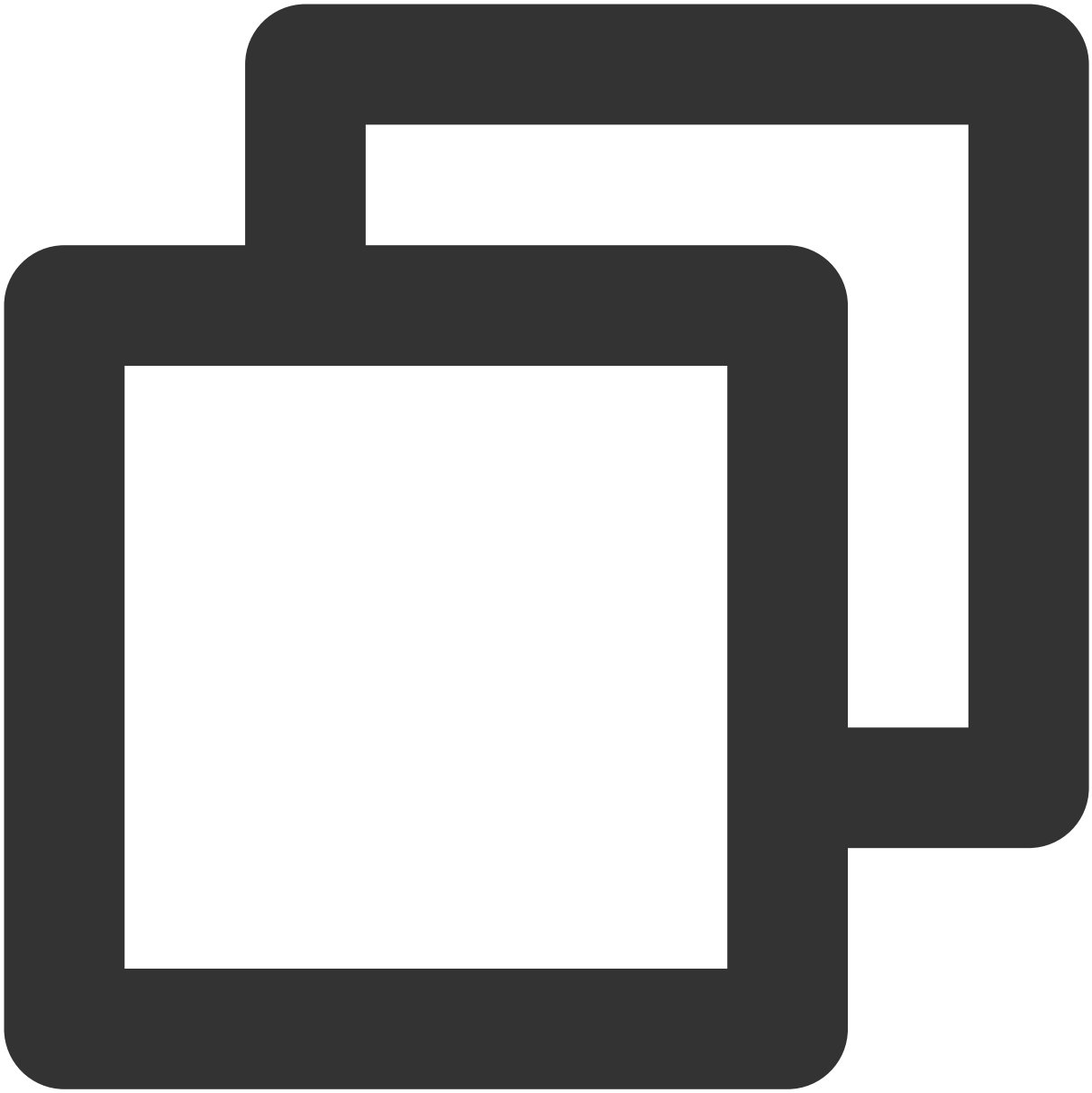
Object module

当前模块对象

属性

属性	类型	说明
exports	Object	模块向外暴露的对象，使用 `require` 引用该模块时可以获取

示例代码



```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}


module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

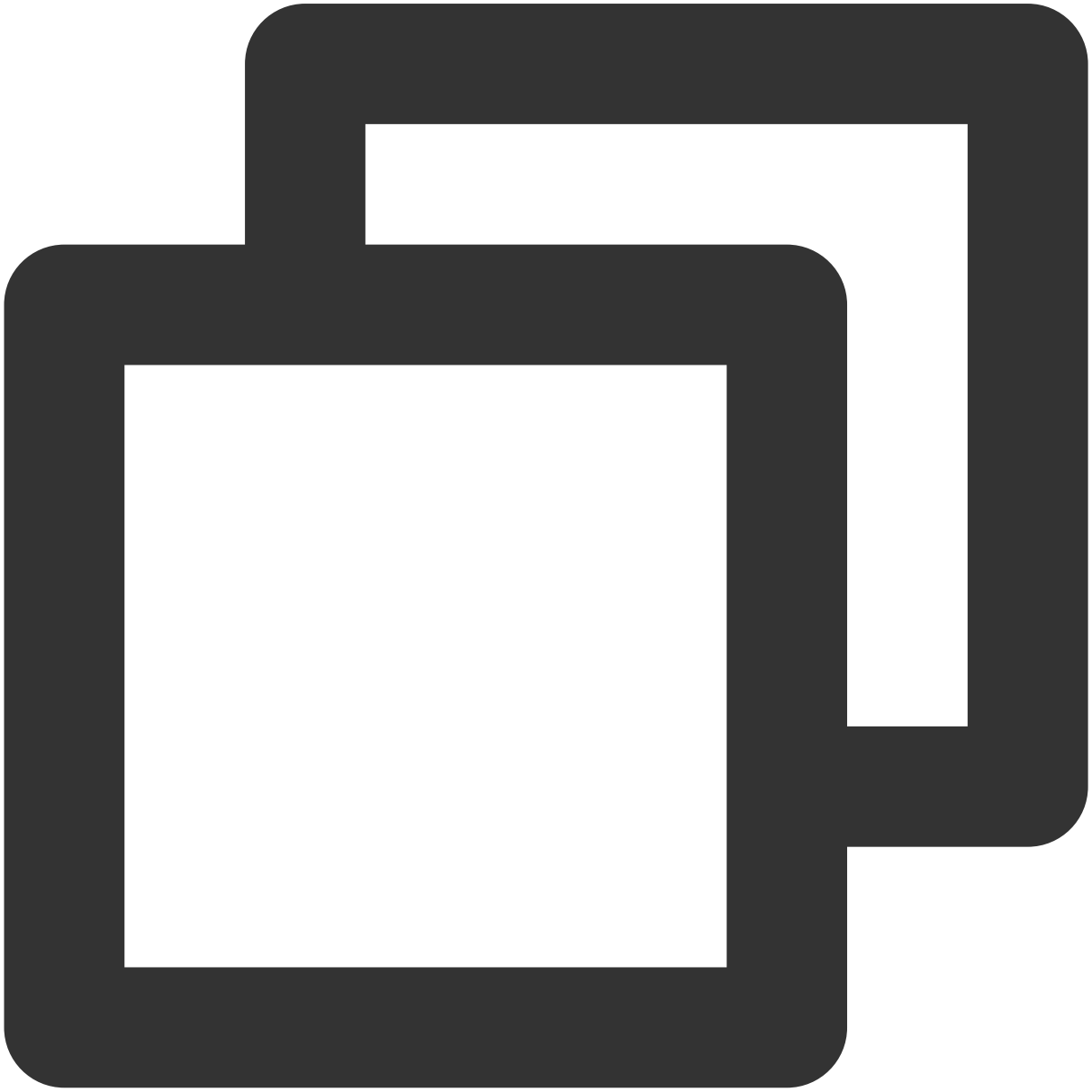
版权所有：腾讯云计算（北京）有限责任公司

exports

Object exports

module.exports 的引用。

示例代码



```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}


module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

版权所有：腾讯云计算（北京）有限责任公司

基础能力

最近更新时间：2024-07-12 17:09:02

console

向调试面板中打印日志。console 是一个全局对象，可以直接访问。在客户端中，向 vConsole 中输出日志。

方法

console.debug()
向调试面板中打印 debug 日志
console.log()
向调试面板中打印 log 日志
console.info()
向调试面板中打印 info 日志
console.warn()
向调试面板中打印 warn 日志
console.error()
向调试面板中打印 error 日志
console.group(string label)
在调试面板中创建一个新的分组。随后输出的内容都会被添加一个缩进，表示该内容属于当前分组。调用 console.groupEnd之后分组结束。
console.groupEnd()
结束由 console.group 创建的分组

注意事项

由于 vConsole 功能有限，以及不同客户端对 console 方法的支持情况有差异，建议开发者在小程序中只使用本文档中提供的方法。

定时器

setTimeout

该 API 使用方法为 number setTimeout(function callback, number delay, any rest)

版权所有：腾讯云计算（北京）有限责任公司

功能说明：设定一个定时器。在定时到期以后执行注册的回调函数。
参数及说明：
function callback，回调函数；
number delay，延迟的时间，函数的调用会在该延迟之后发生，单位 ms；
any rest，param1, param2, ..., paramN 等附加参数，它们会作为参数传递给回调函数。
返回值：number，定时器的编号，这个值可以传递给 clearTimeout 来取消该定时。

clearTimeout

> 该 API 使用方法为 clearTimeout(number timeoutID)

功能说明：取消由 setTimeout 设置的定时器。
参数及说明：number timeoutID，要取消的定时器的 ID。

setInterval

> 该 API 使用方法为 number setInterval(function callback, number delay, any rest)

功能说明：设定一个定时器。按照指定的周期（以毫秒计）来执行注册的回调函数。
参数及说明：
function callback，回调函数；
number delay，执行回调函数之间的时间间隔，单位 ms；
any rest，param1, param2, ..., paramN 等附加参数，它们会作为参数传递给回调函数。
返回值：number，定时器的编号，这个值可以传递给 clearInterval 来取消该定时。

clearInterval

> 该 API 使用方法为 clearInterval(number intervalID)

功能说明：取消由 setInterval 设置的定时器。
参数及说明：number intervalID，要取消的定时器的 ID。

版权所有：腾讯云计算（北京）有限责任公司

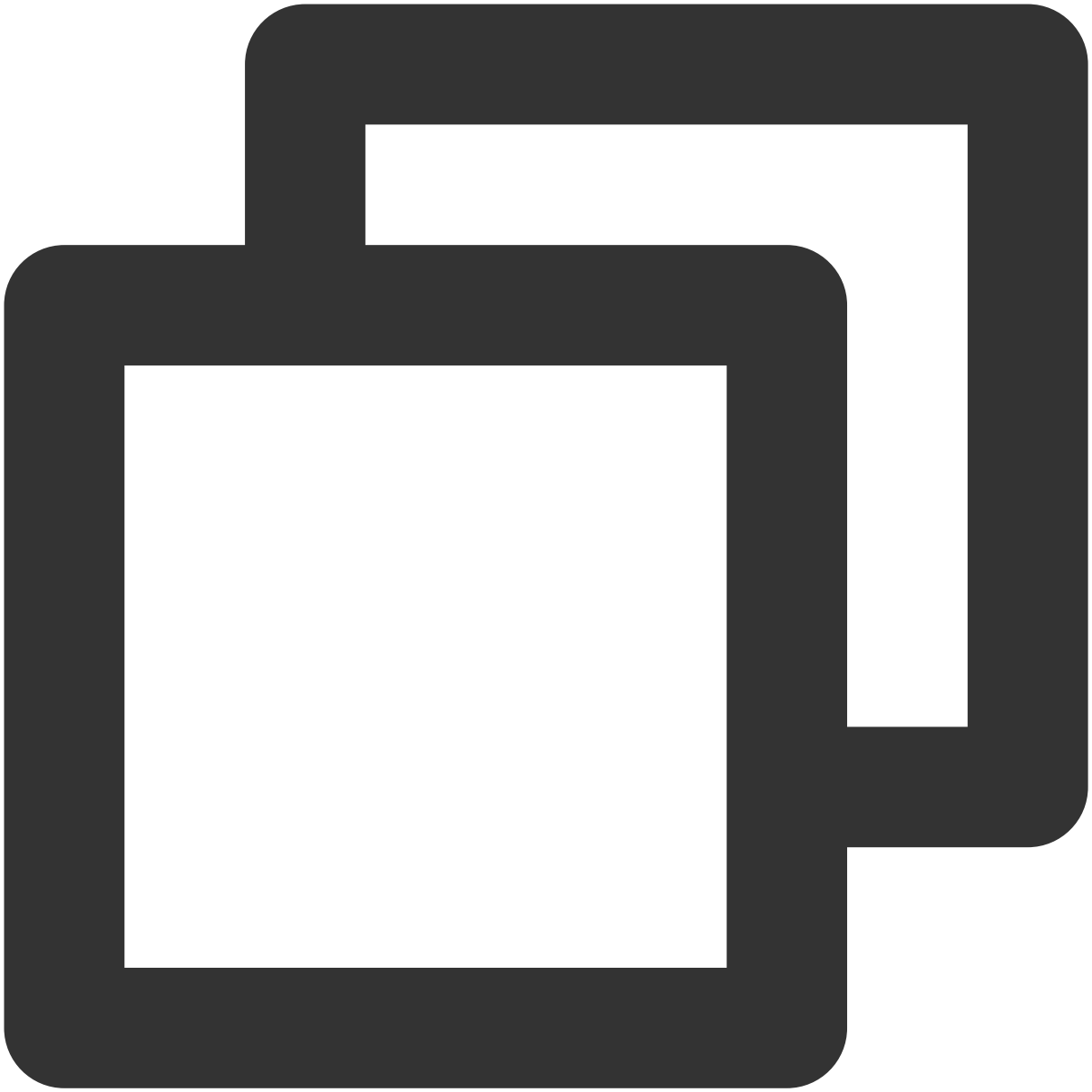
WXML 语法参考

介绍

最近更新时间：2024-07-12 18:53:23

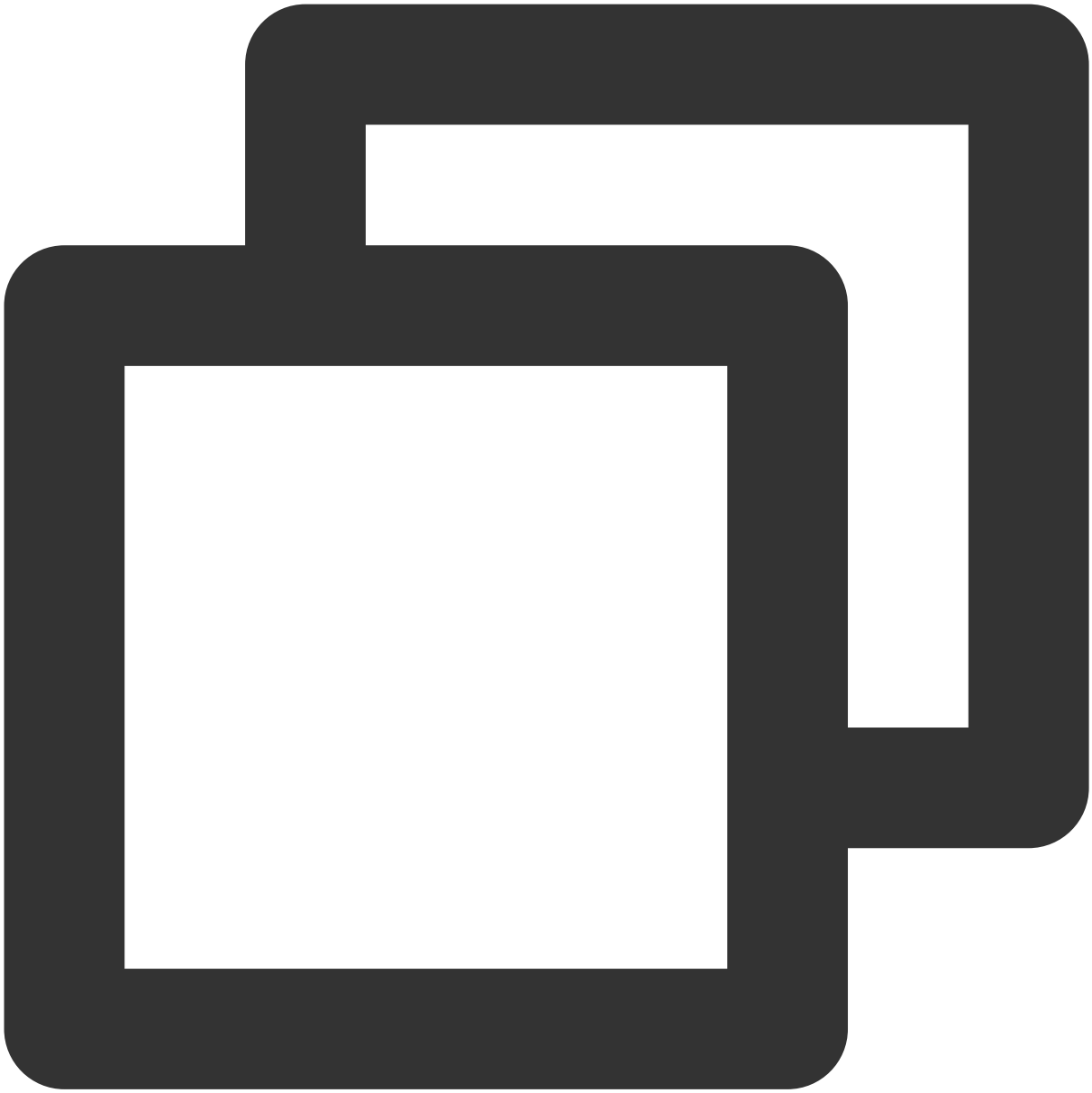
WXML 是框架设计的一套标签语言，结合 API 能力、事件系统，可以构建出页面的结构。
用以下一些简单的例子来看看 WXML 具有什么能力：

数据绑定



```
<!--wxml-->
<view> {{message}} </view>
```

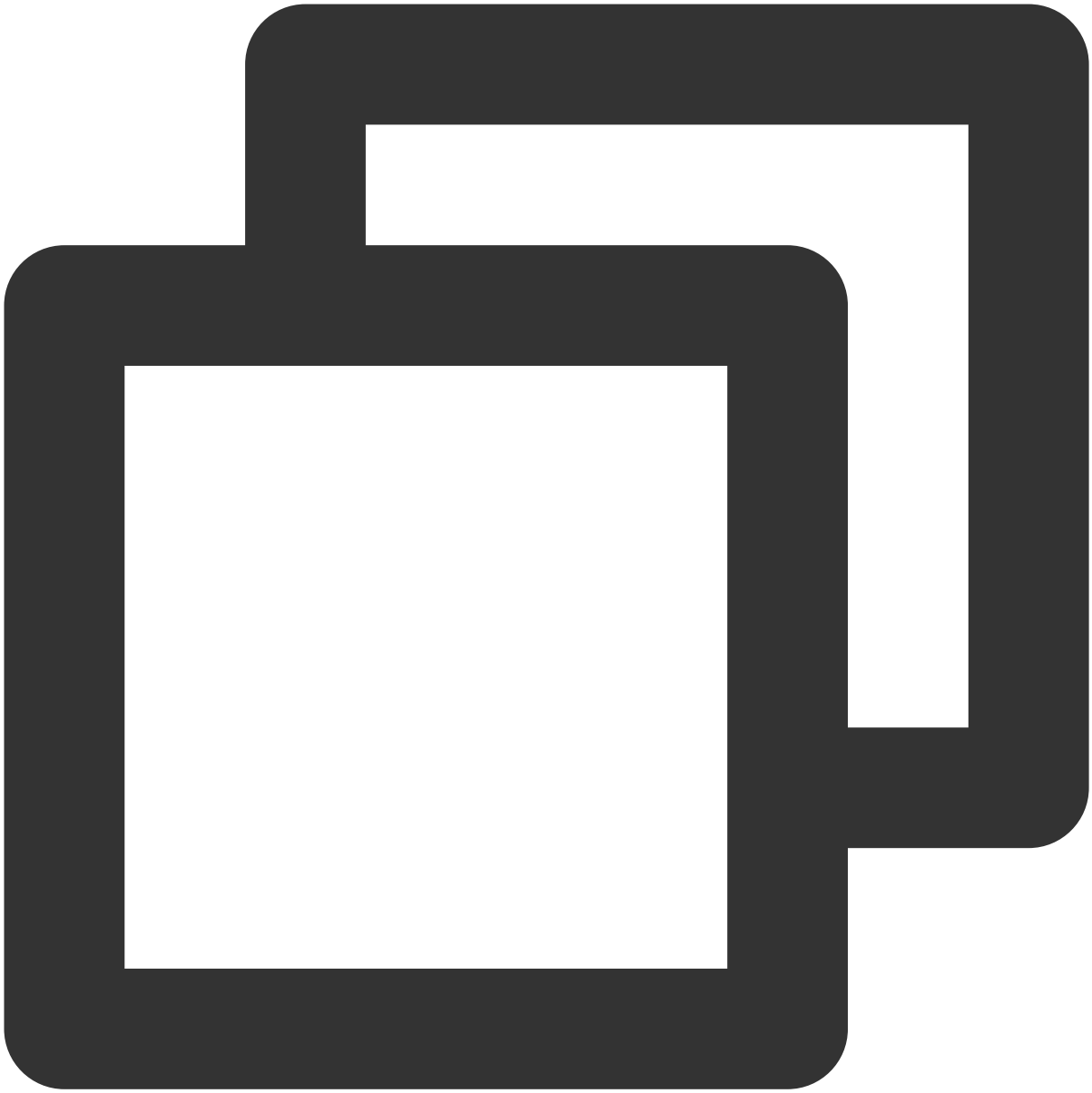
版权所有：腾讯云计算（北京）有限责任公司



```
// page.js
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

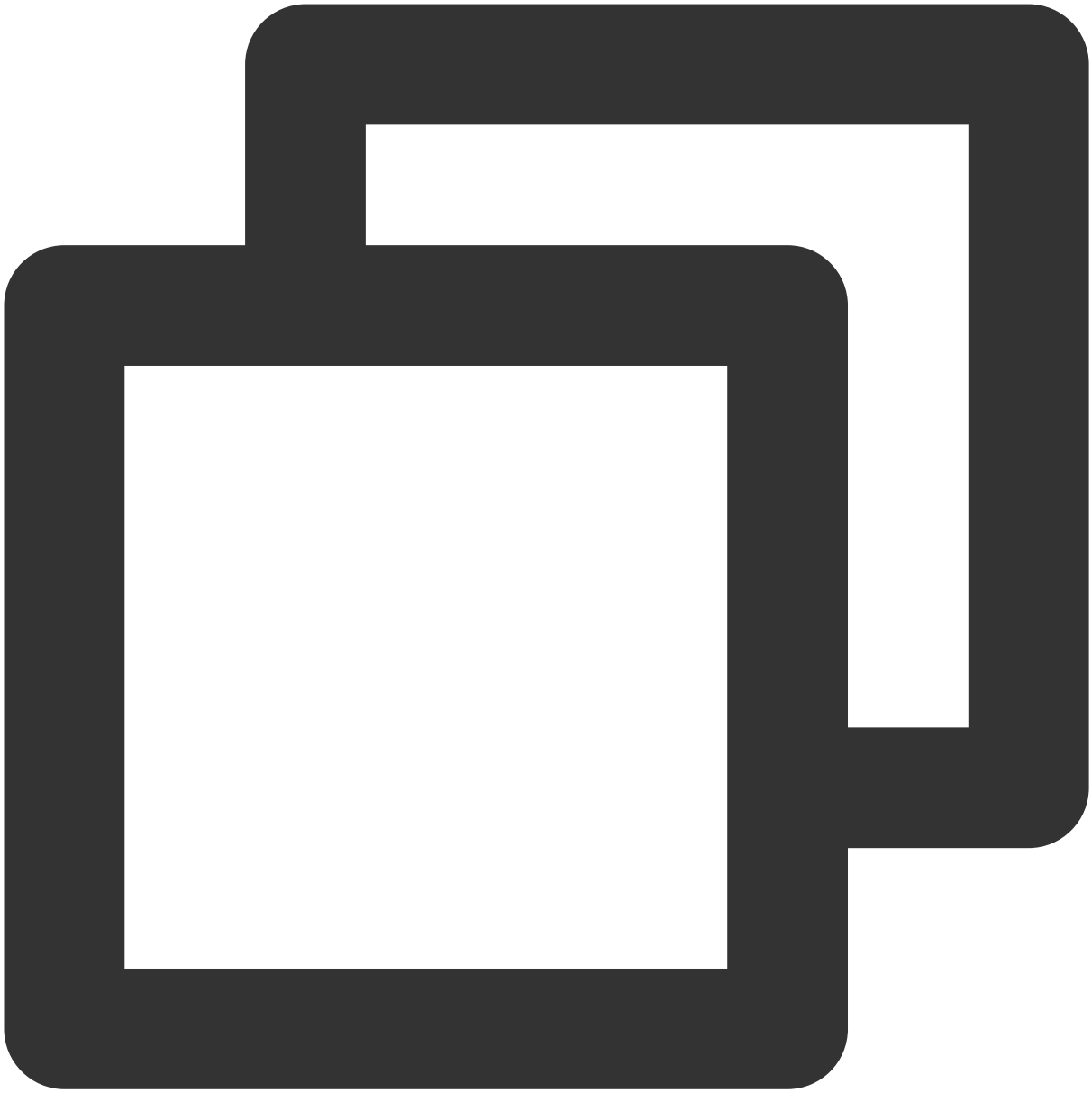
列表渲染

版权所有：腾讯云计算（北京）有限责任公司



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

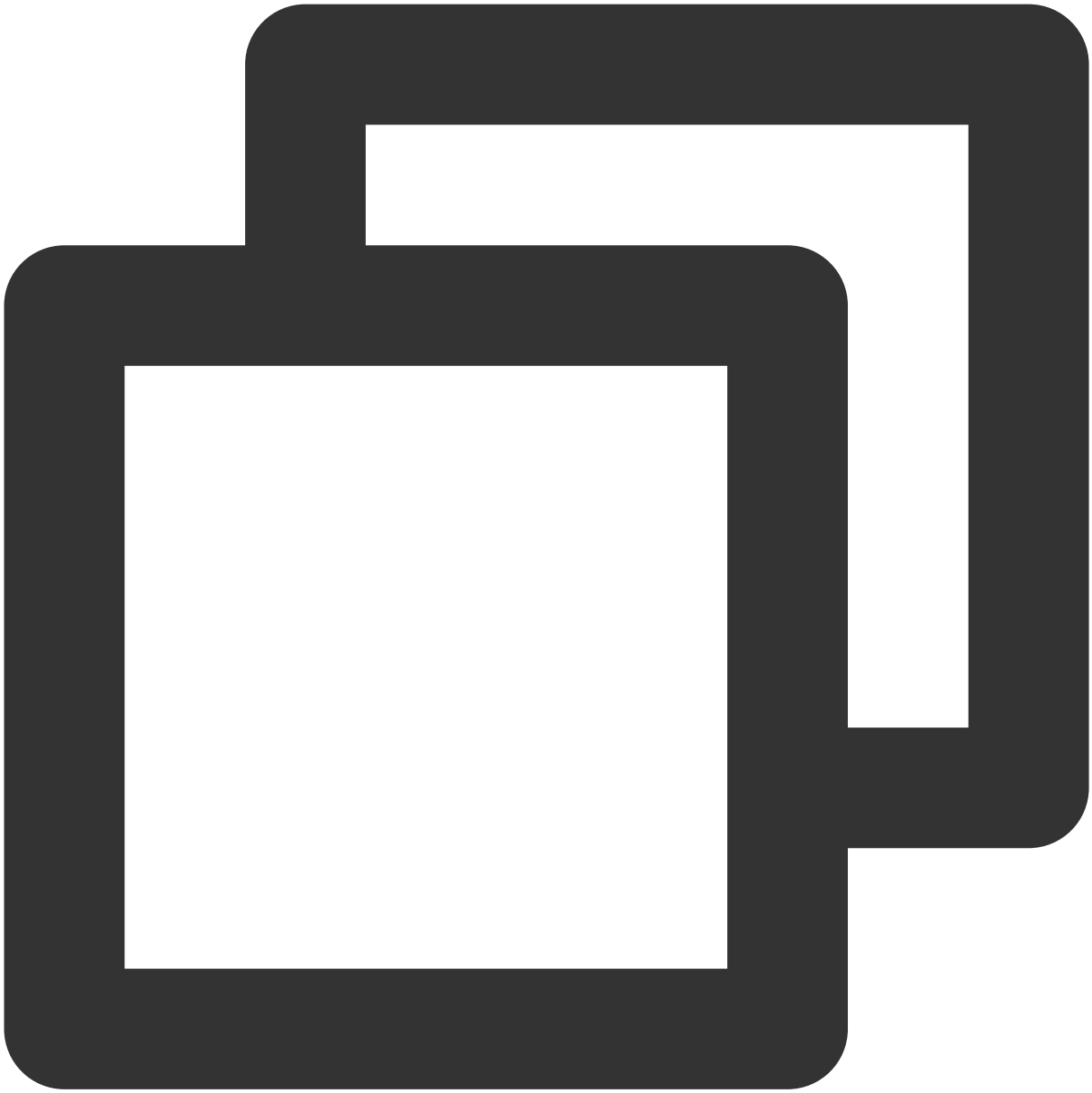
版权所有：腾讯云计算（北京）有限责任公司



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

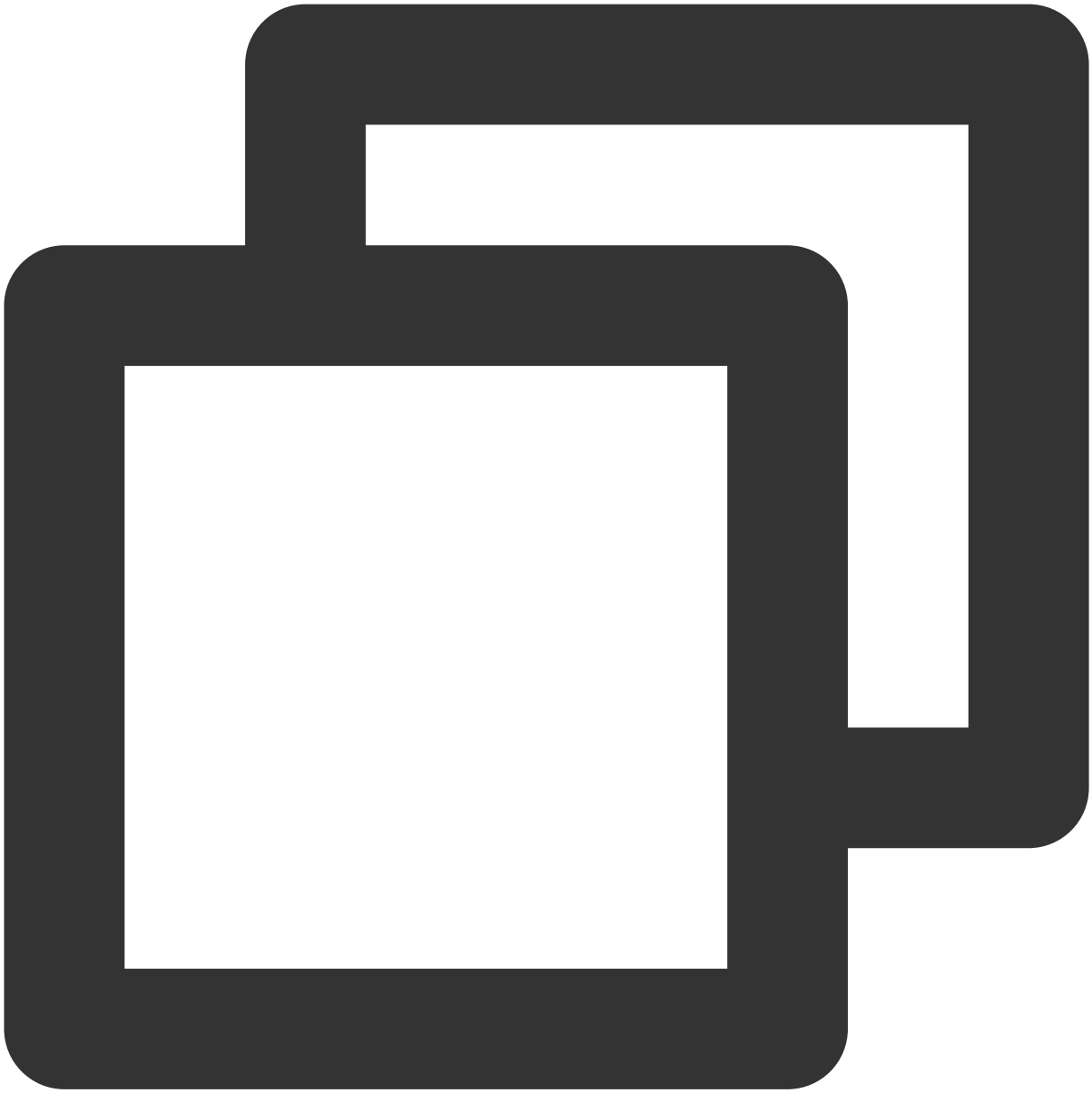
条件渲染

版权所有：腾讯云计算（北京）有限责任公司



```
<!--wxml-->
<view wx:if="{{view == 'WEBVIEW'}}"> WEBVIEW </view>
<view wx:elif="{{view == 'APP'}}"> APP </view>
<view wx:else="{{view == 'MINA'}}"> MINA </view>
```

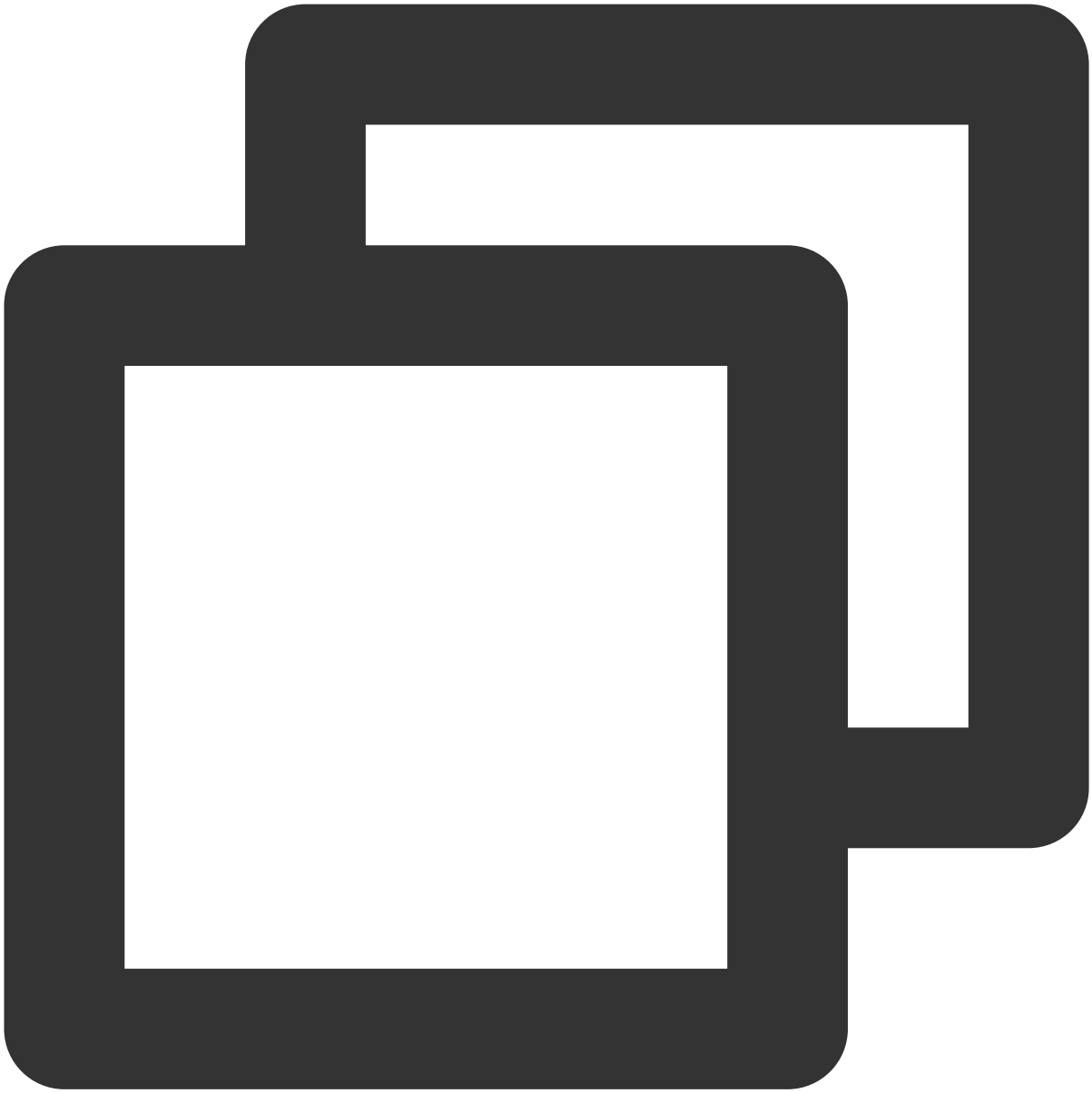
版权所有：腾讯云计算（北京）有限责任公司



```
// page.js
Page({
  data: {
    view: 'MINA'
  }
})
```

模板

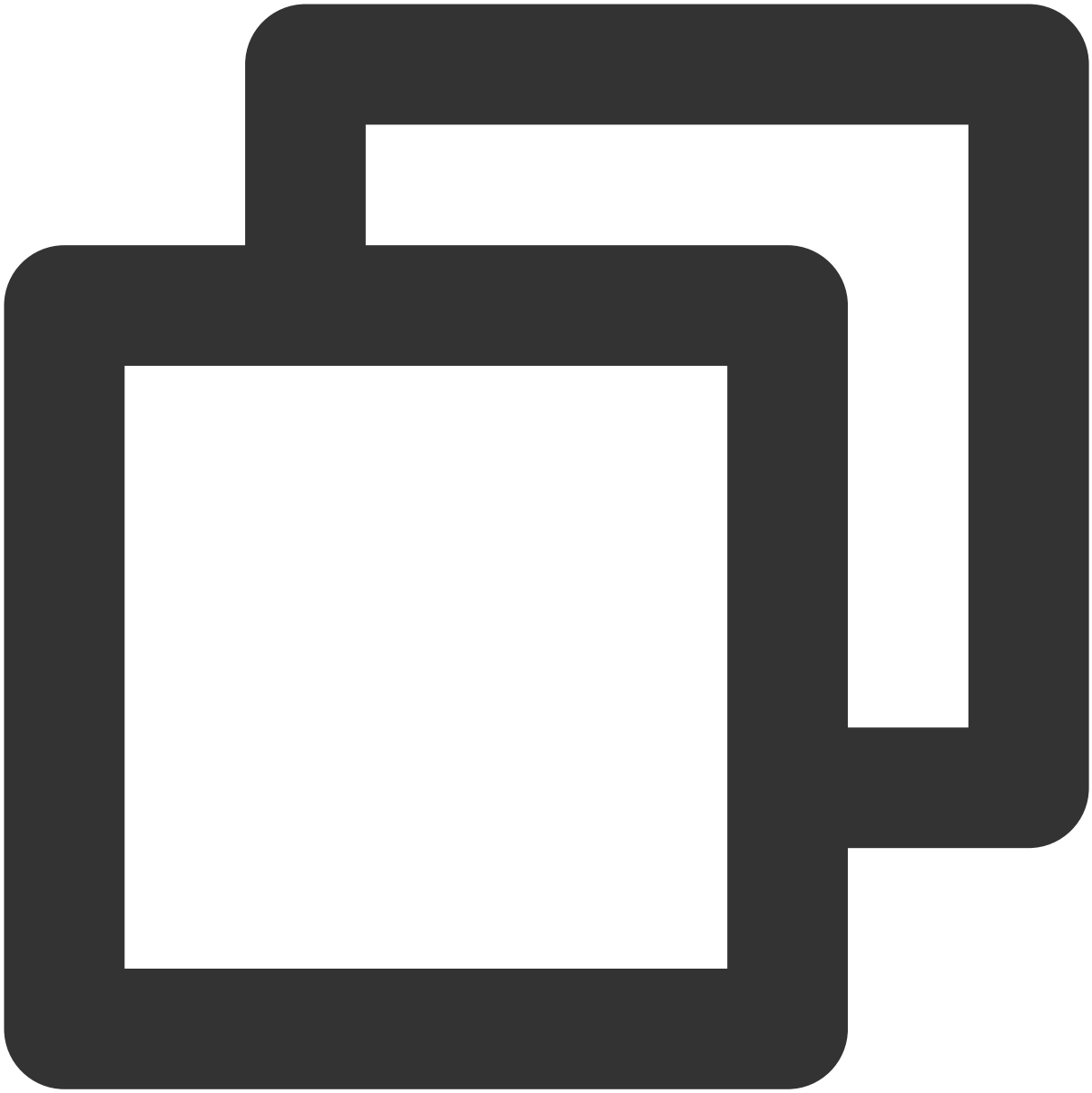
版权所有：腾讯云计算（北京）有限责任公司



```
<!--wxml-->
<template name="staffName">
  <view>
    FirstName: {{firstName}}, LastName: {{lastName}}
  </view>
</template>

<template is="staffName" data="{{...staffA}}"></template>
<template is="staffName" data="{{...staffB}}"></template>
<template is="staffName" data="{{...staffC}}"></template>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// page.js
Page({
  data: {
    staffA: {firstName: 'Hulk', lastName: 'Hu'},
    staffB: {firstName: 'Shang', lastName: 'You'},
    staffC: {firstName: 'Gideon', lastName: 'Lin'}
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

数据绑定

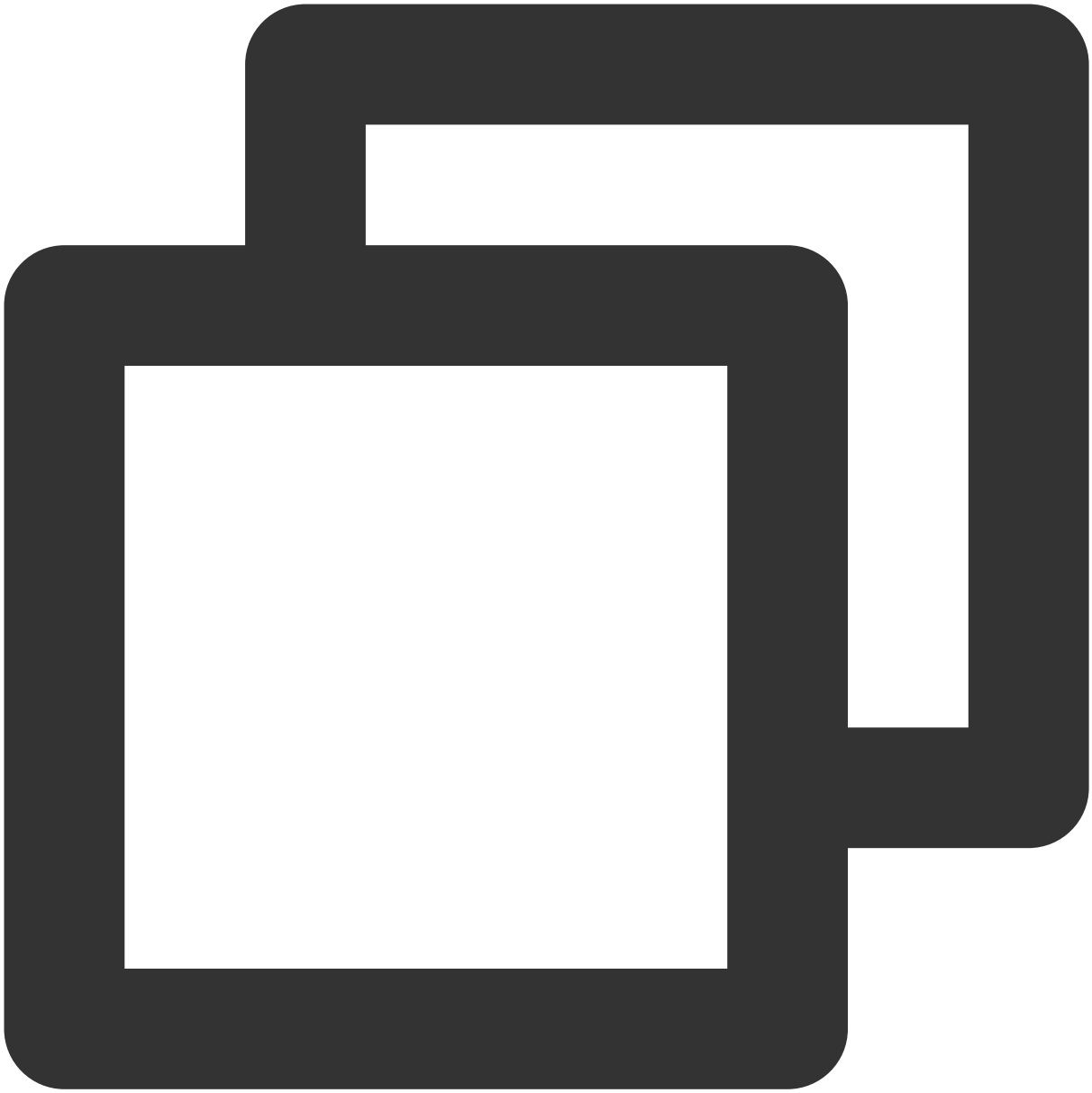
最近更新时间：2024-07-12 17:06:27

WXML 中的动态数据均来自对应 Page 的 data。

简单绑定

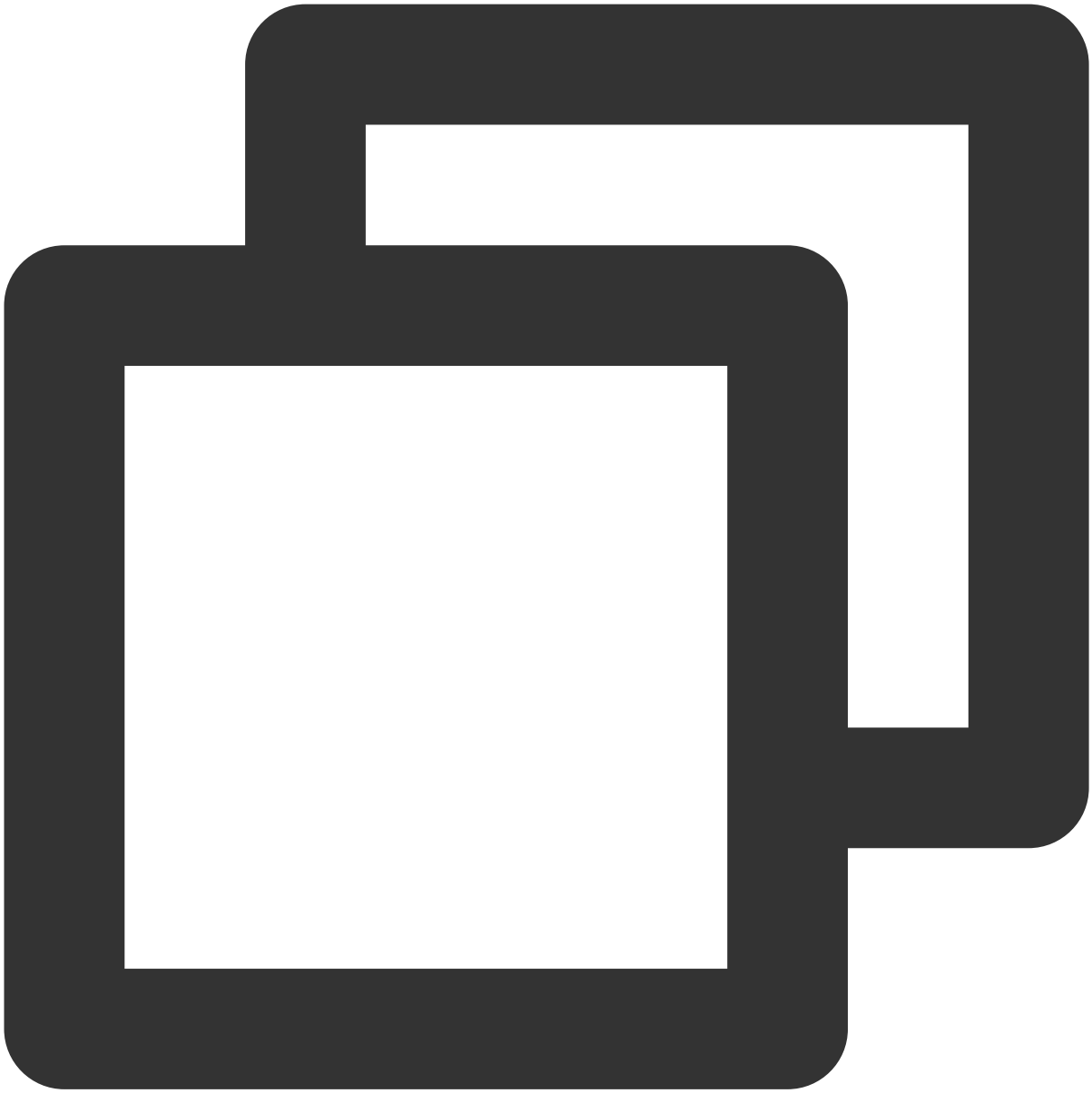
数据绑定使用 Mustache 语法（双大括号）将变量包起来，可以作用于：

内容



```
<view> {{ message }} </view>
```

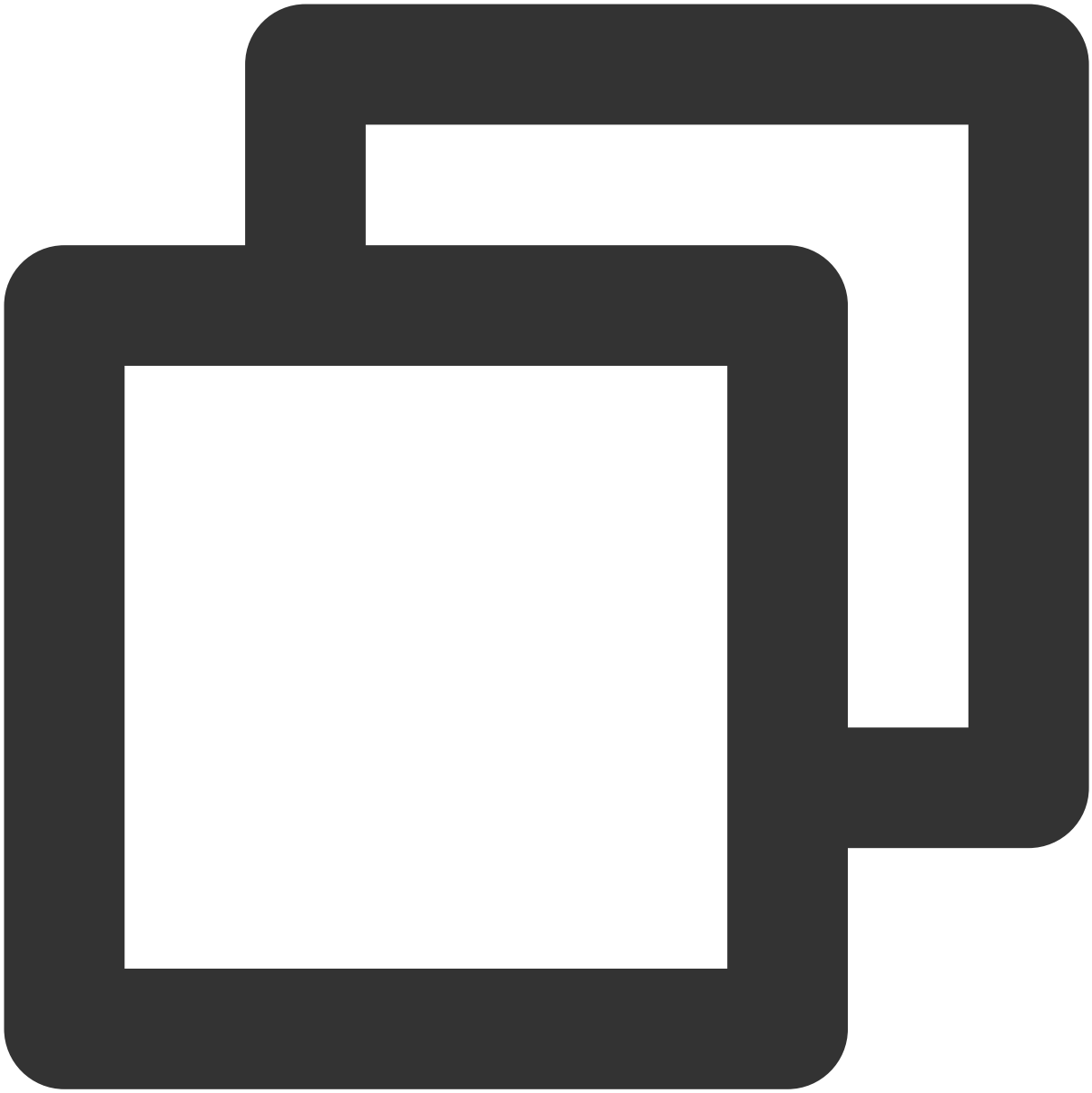
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

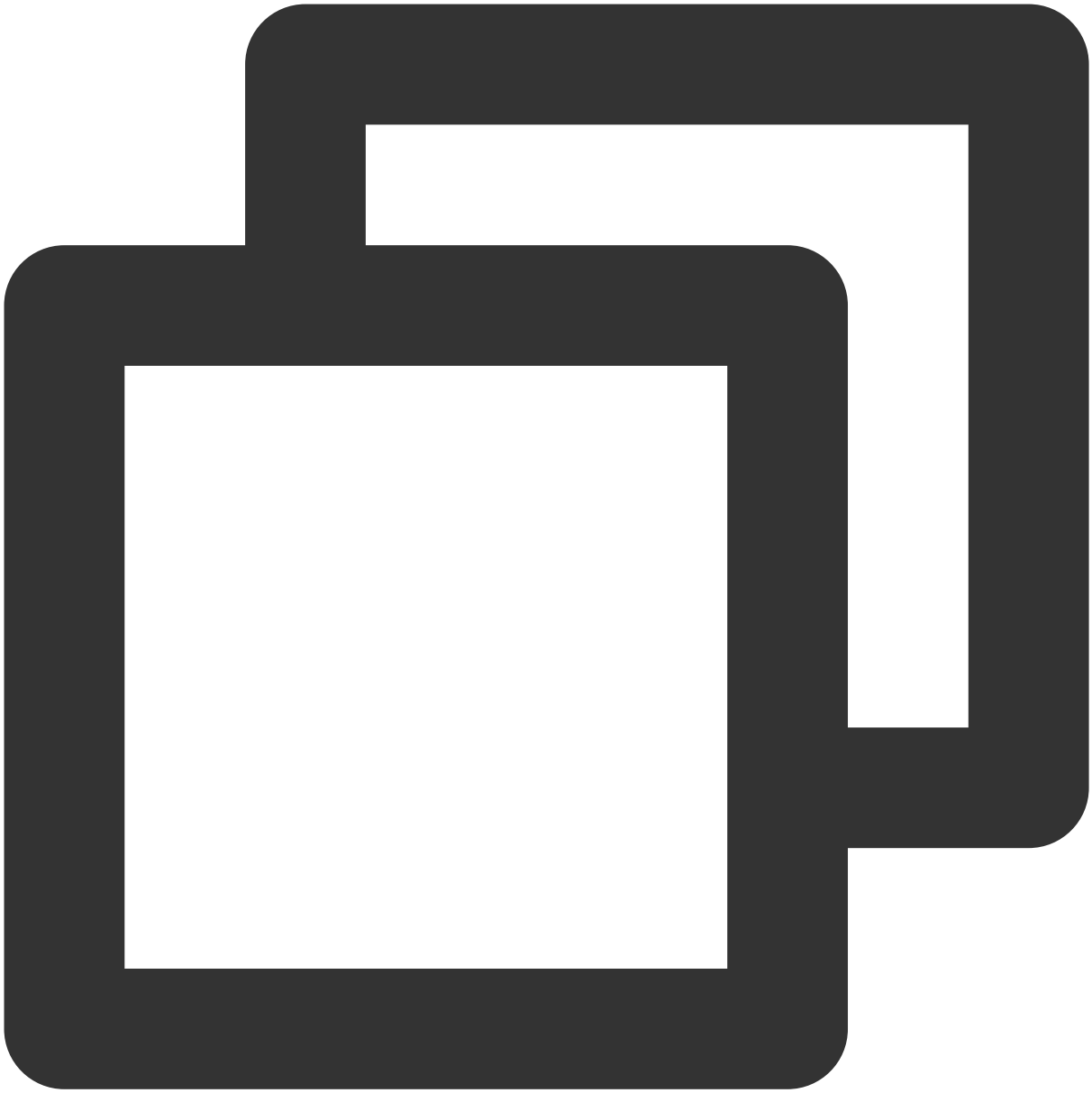
组件属性（需要在双引号之内）

版权所有：腾讯云计算（北京）有限责任公司



```
<view id="item-{{id}}"> </view>
```

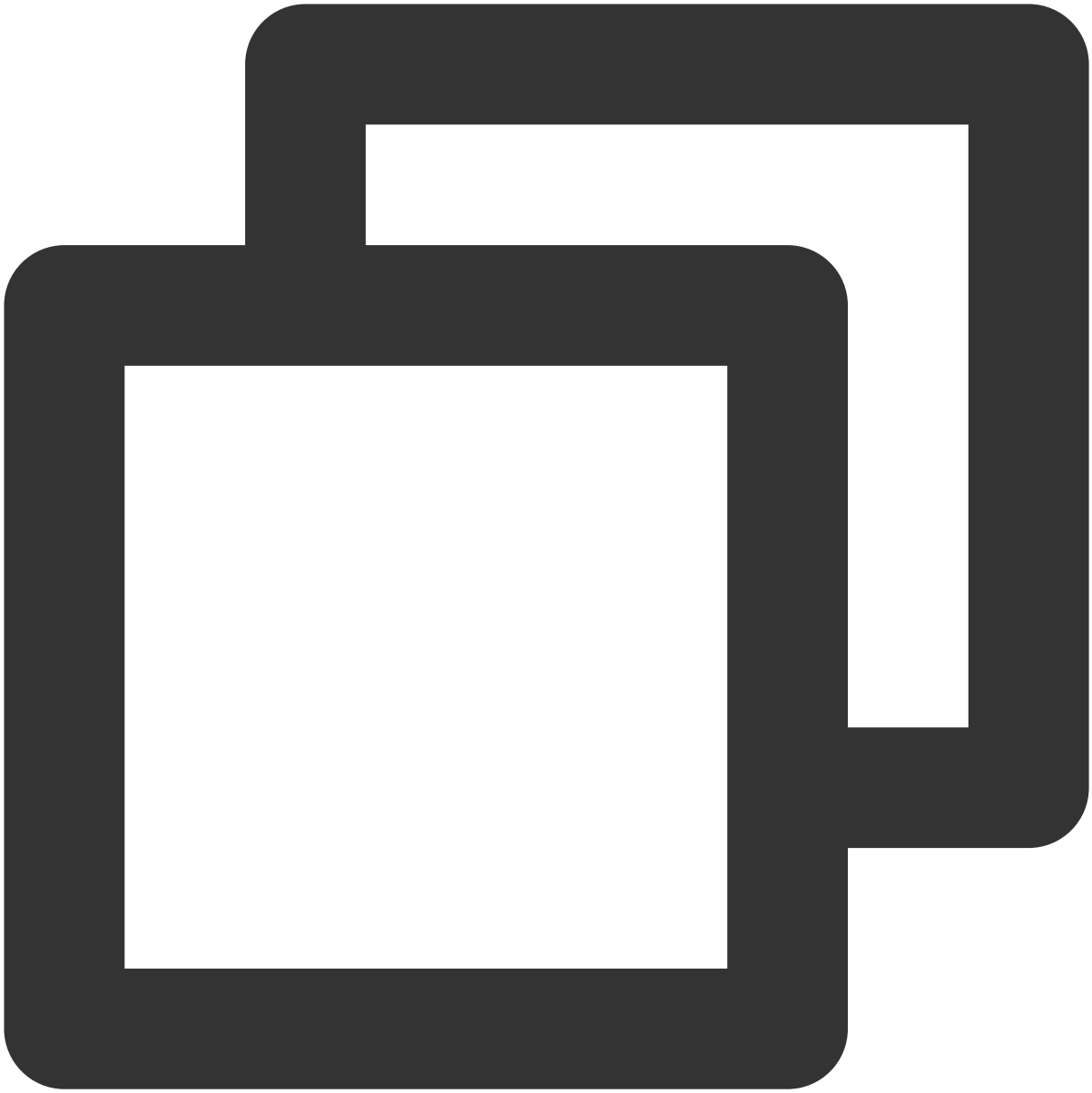
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    id: 0
  }
})
```

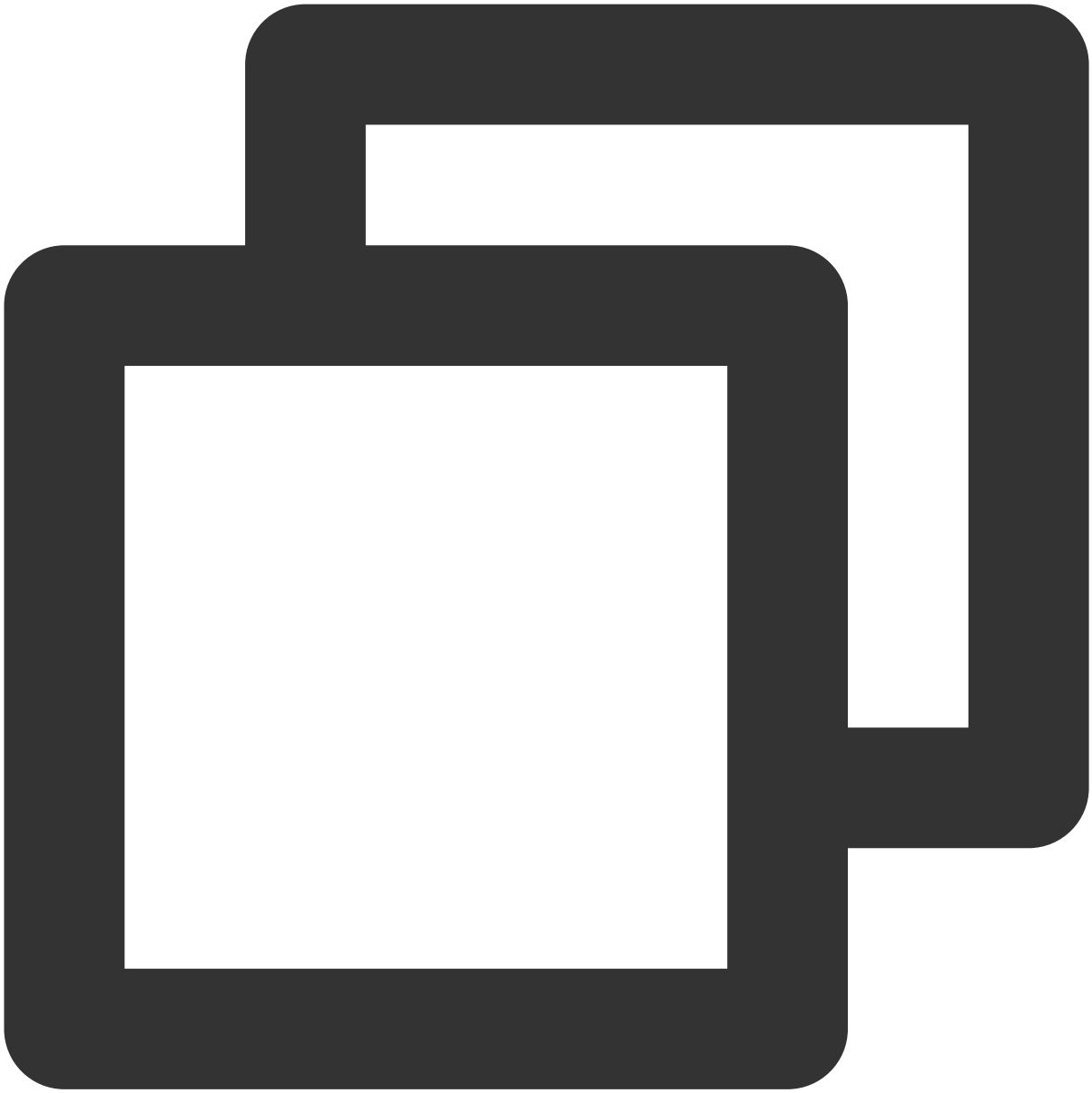
控制属性（需要在双引号之内）

版权所有：腾讯云计算（北京）有限责任公司



```
<view wx:if="{{condition}}"> </view>
```

版权所有：腾讯云计算（北京）有限责任公司

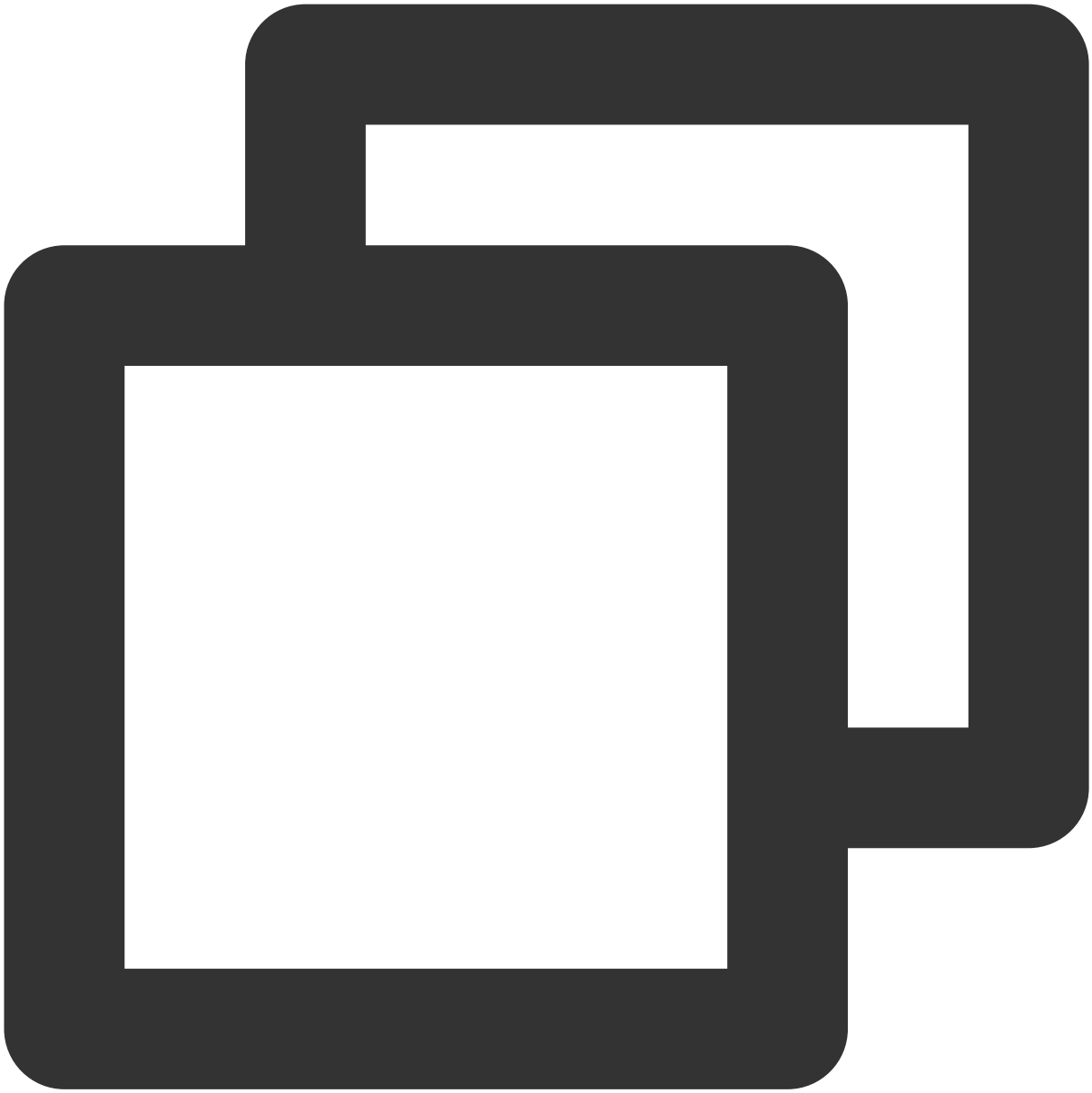


```
Page({
  data: {
    condition: true
  }
})
```

关键字（需要在双引号之内）

`true` ：boolean 类型的 true，代表真值。

`false` ： boolean 类型的 false，代表假值。



```
<checkbox checked="{{false}}"> </checkbox>
```

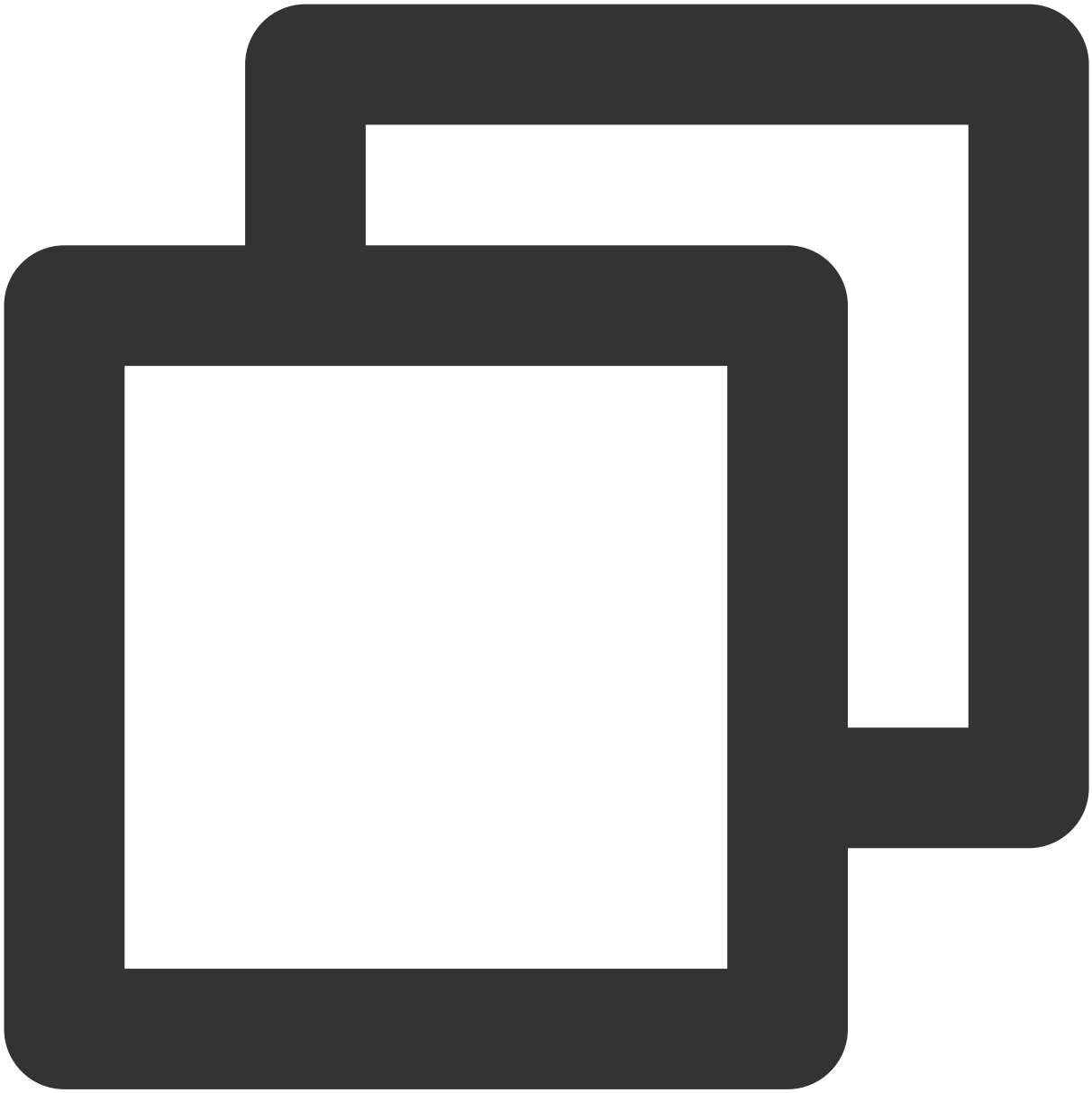
注意：

不要直接写 checked="false"，其计算结果是一个字符串，转成 boolean 类型后代表真值。

运算

可以在 `{{}}` 内进行简单的运算，支持的有如下几种方式：

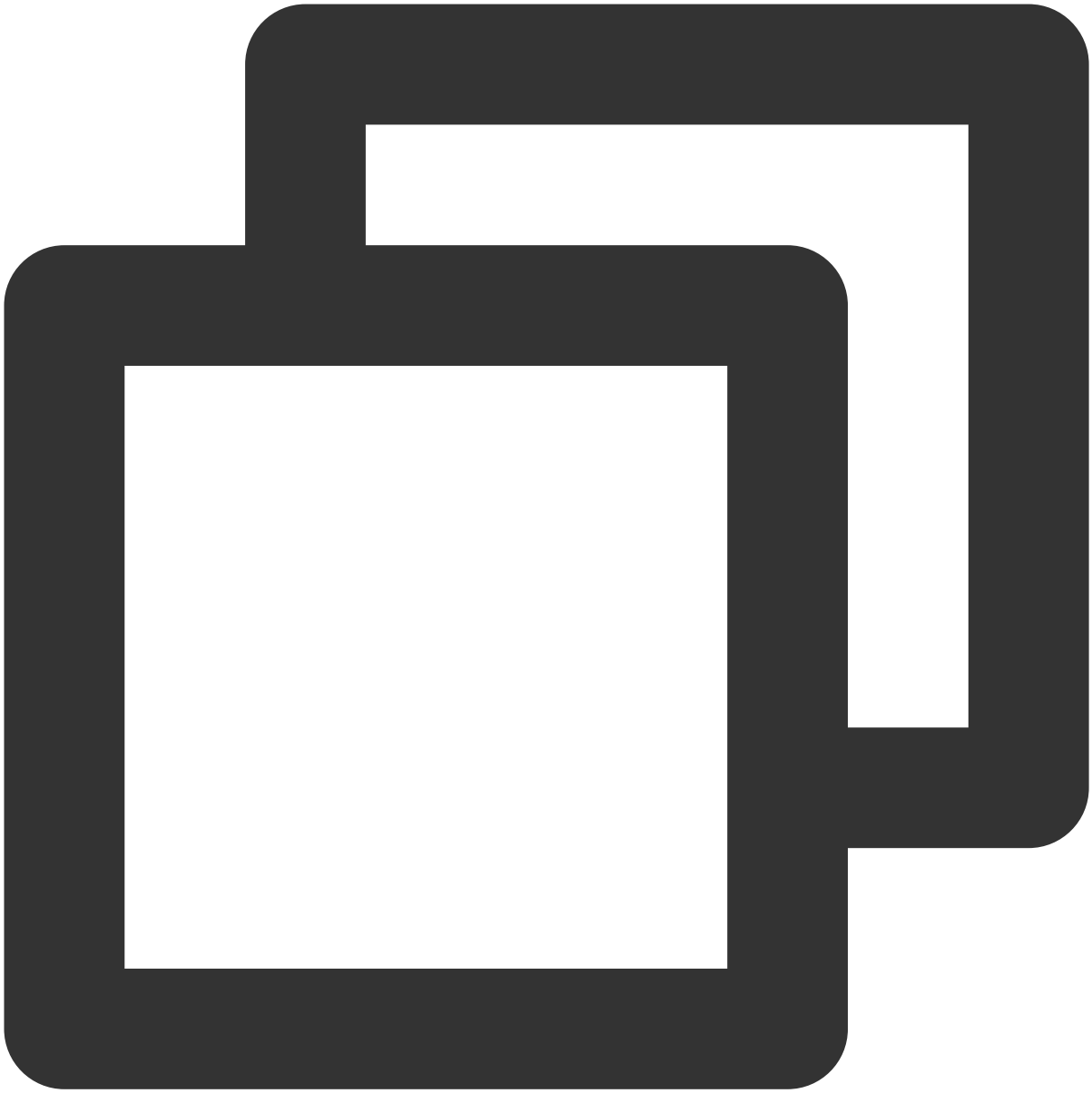
三元运算



```
<view hidden="{{flag ? true : false}}"> Hidden </view>
```

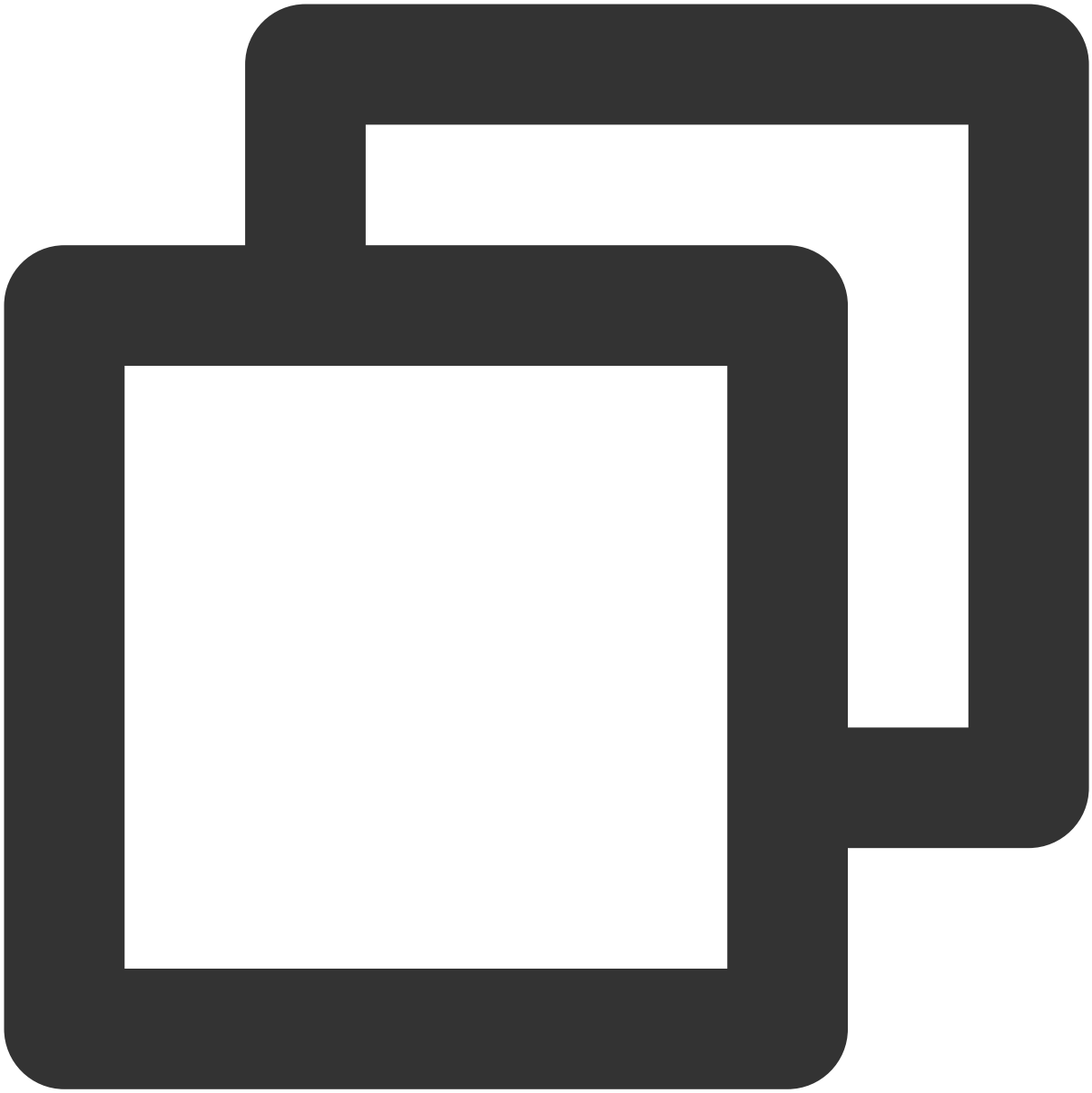
算数运算

版权所有：腾讯云计算（北京）有限责任公司



```
<view> {{a + b}} + {{c}} + d </view>
```

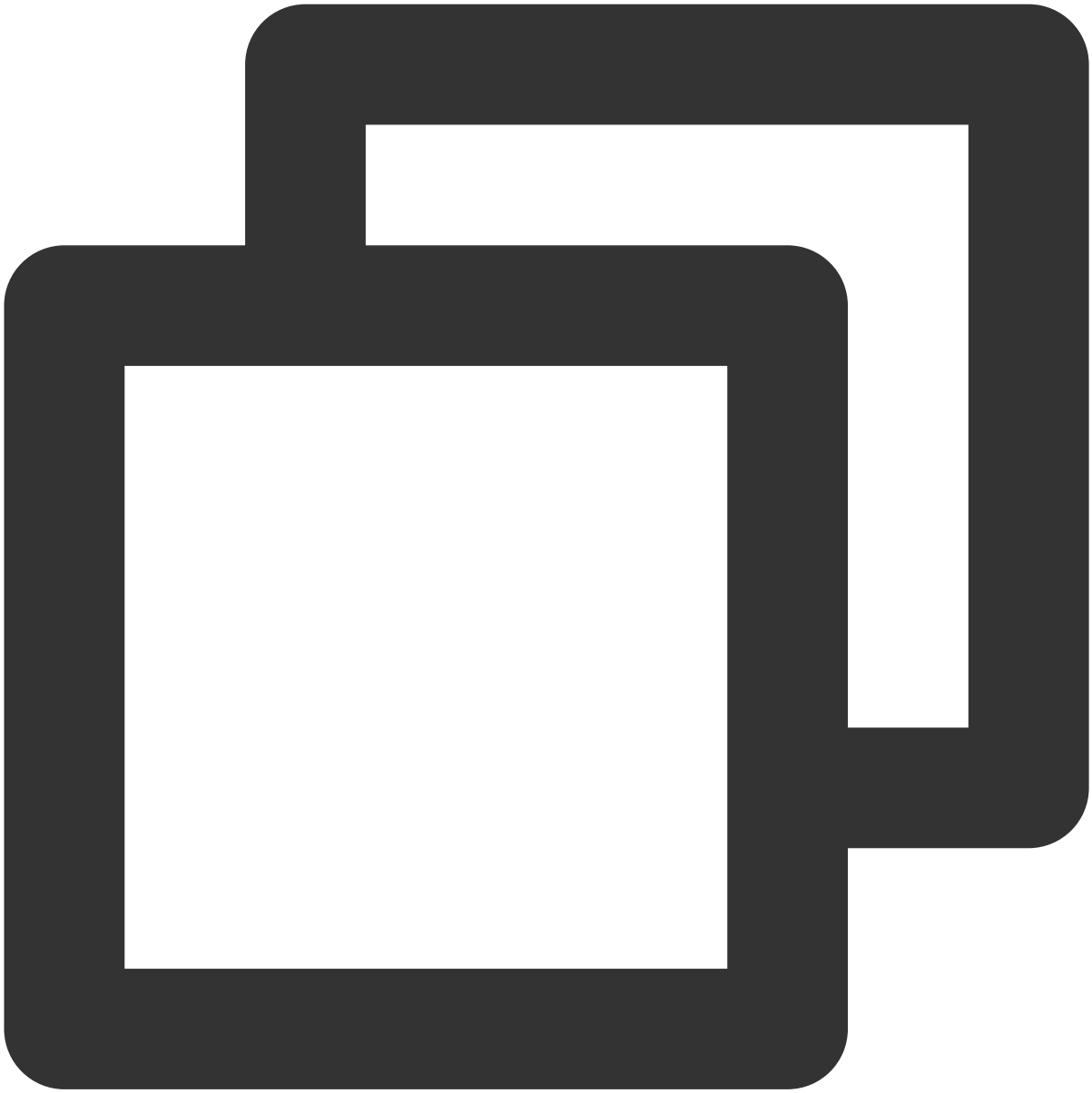
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    a: 1,
    b: 2,
    c: 3
  }
})
```

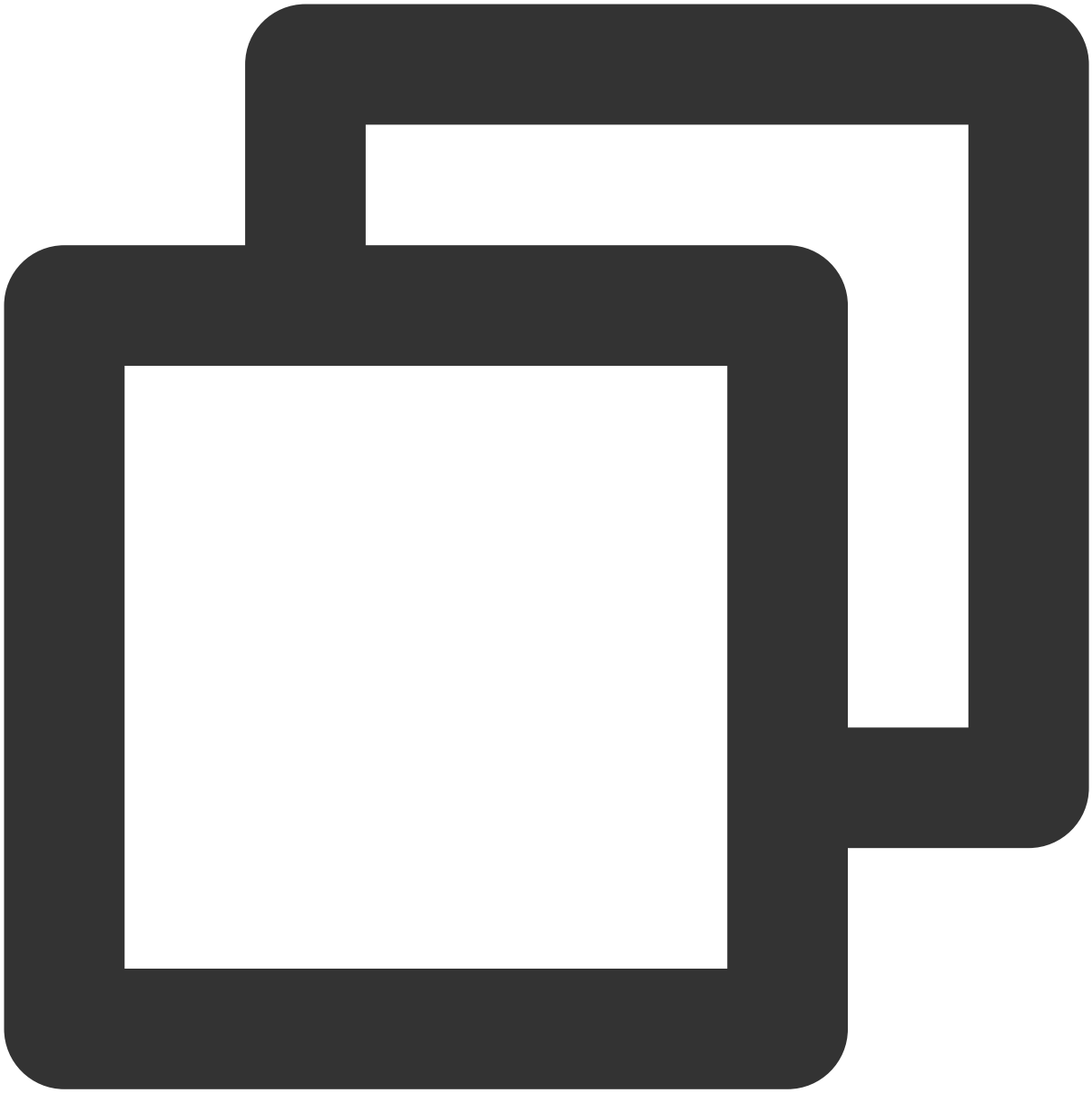
view 中的内容为 `3 + 3 + d` 。

逻辑判断



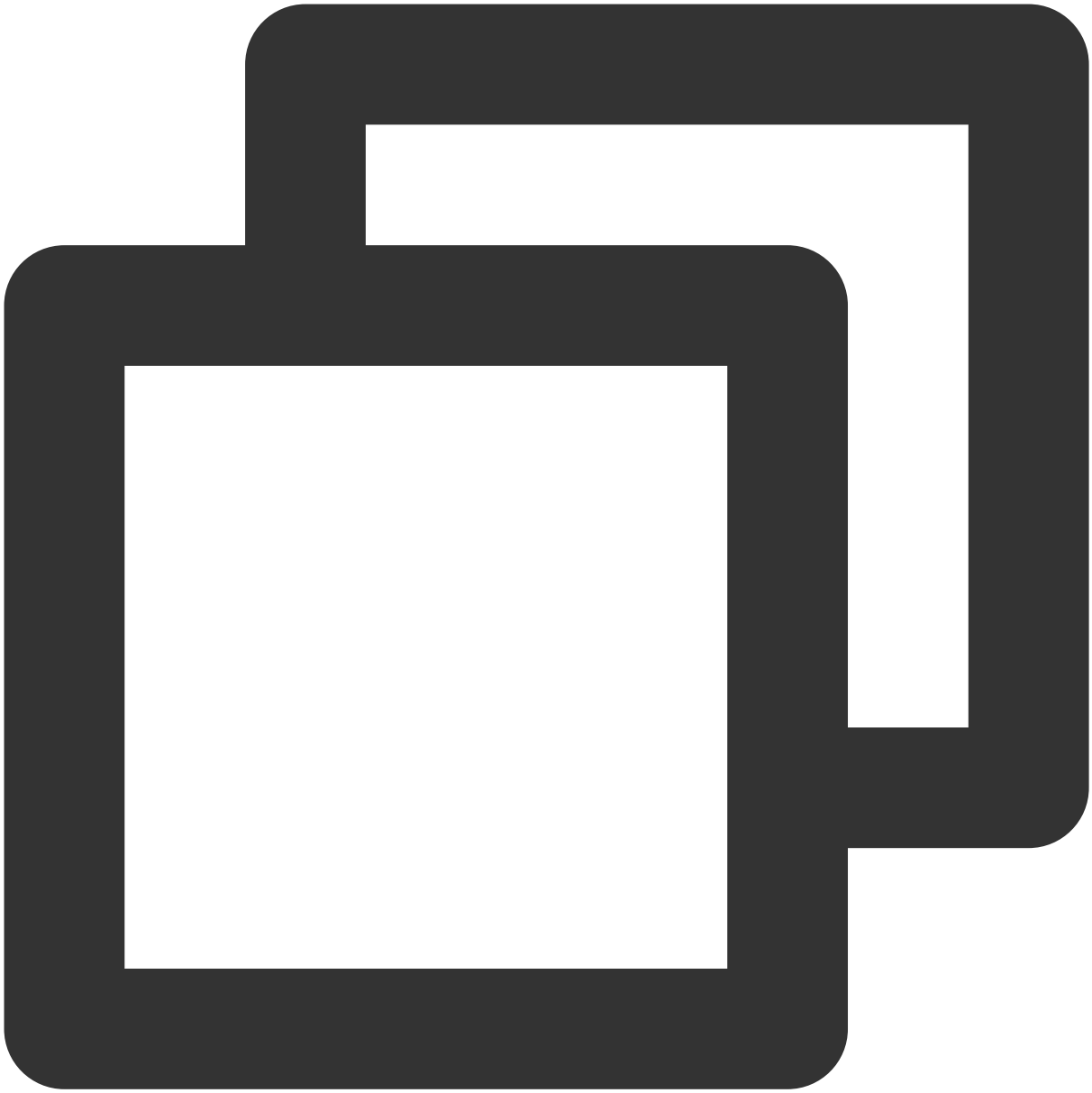
```
<view wx:if="{{length > 5}}"> </view>
```

字符串运算



```
<view>{{"hello" + name}}</view>
```

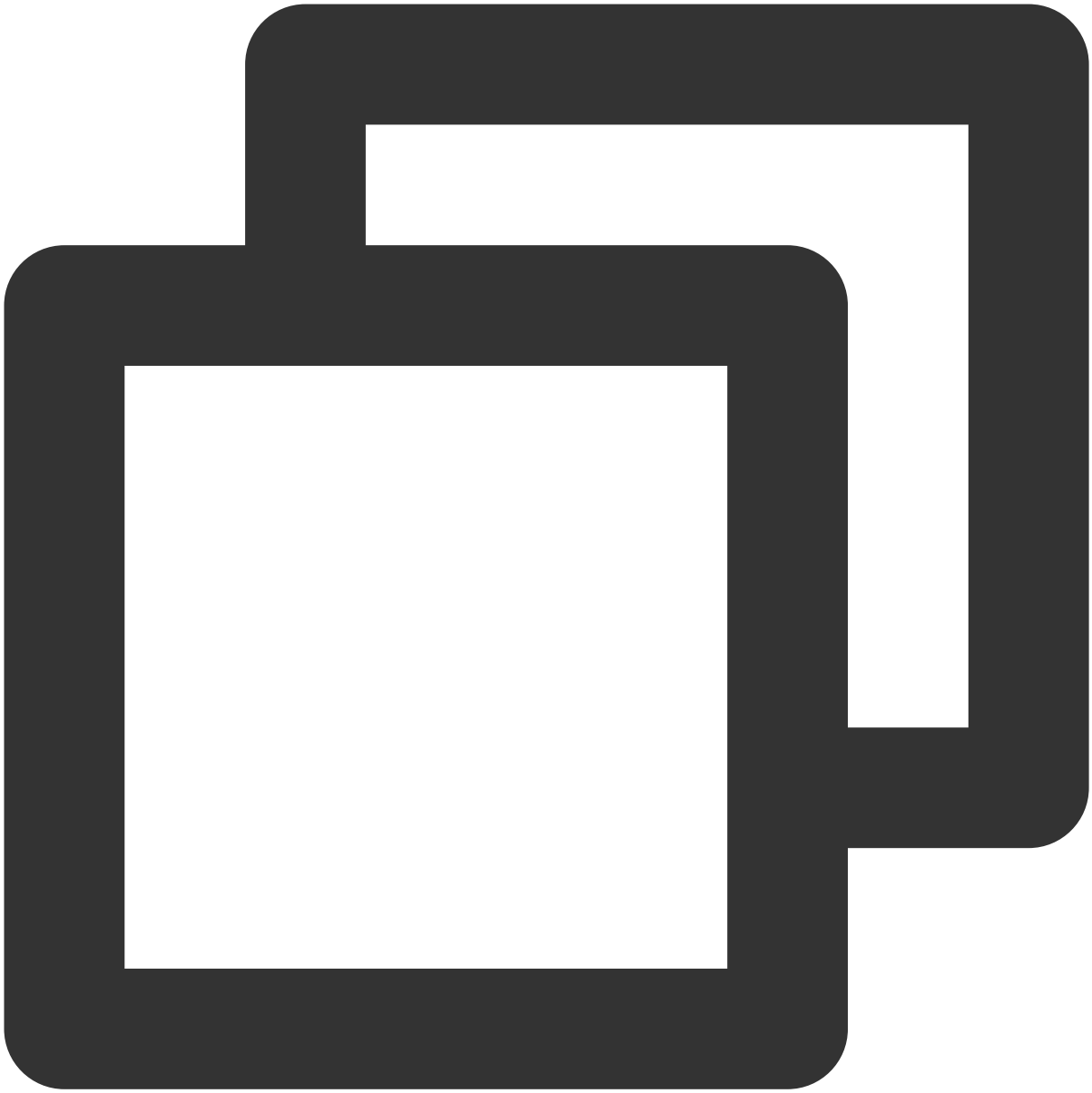
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data:{
    name: 'MINA'
  }
})
```

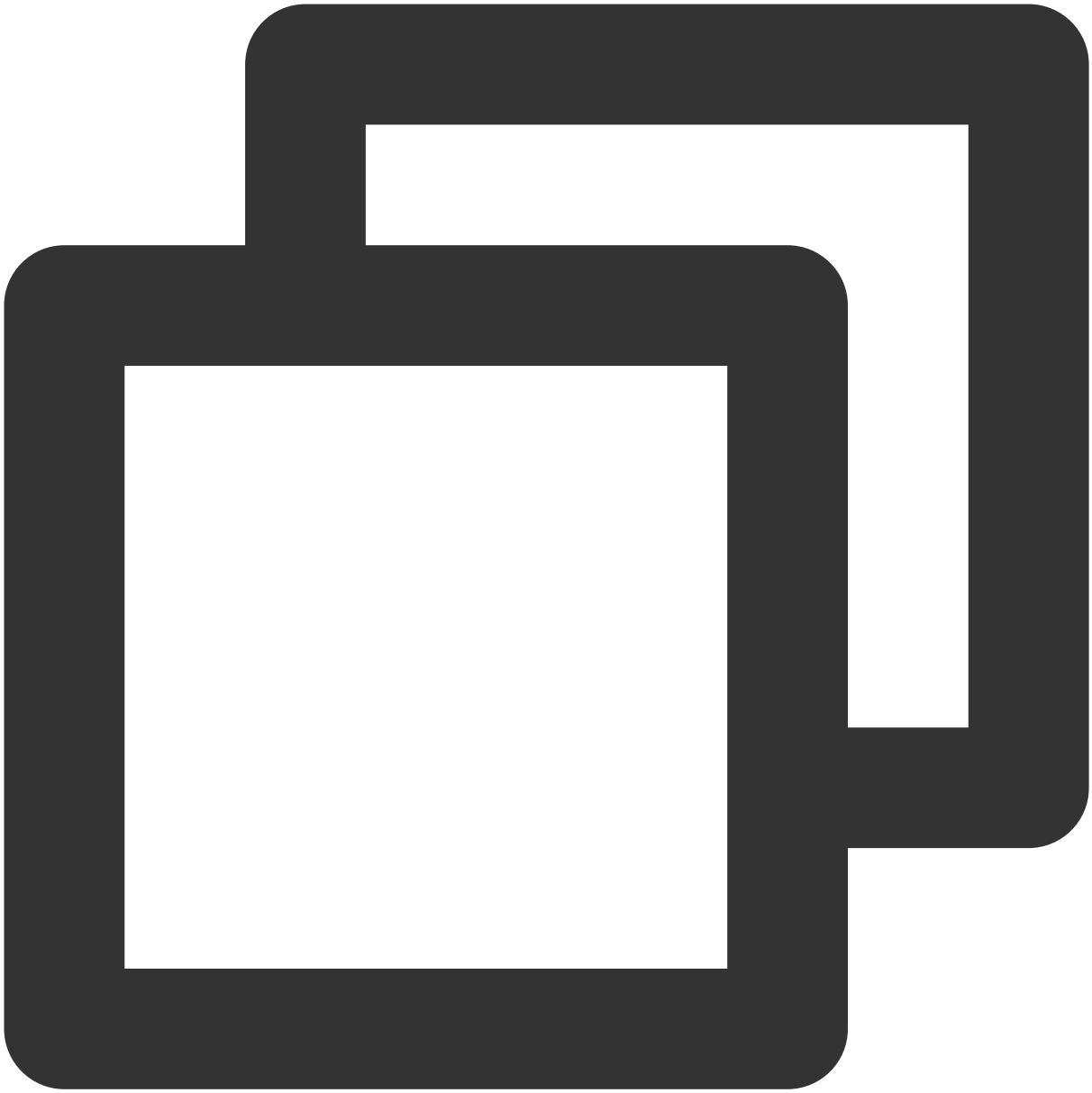
数据路径运算

版权所有：腾讯云计算（北京）有限责任公司



```
<view>{{object.key}} {{array[0]}}</view>
```

版权所有：腾讯云计算（北京）有限责任公司



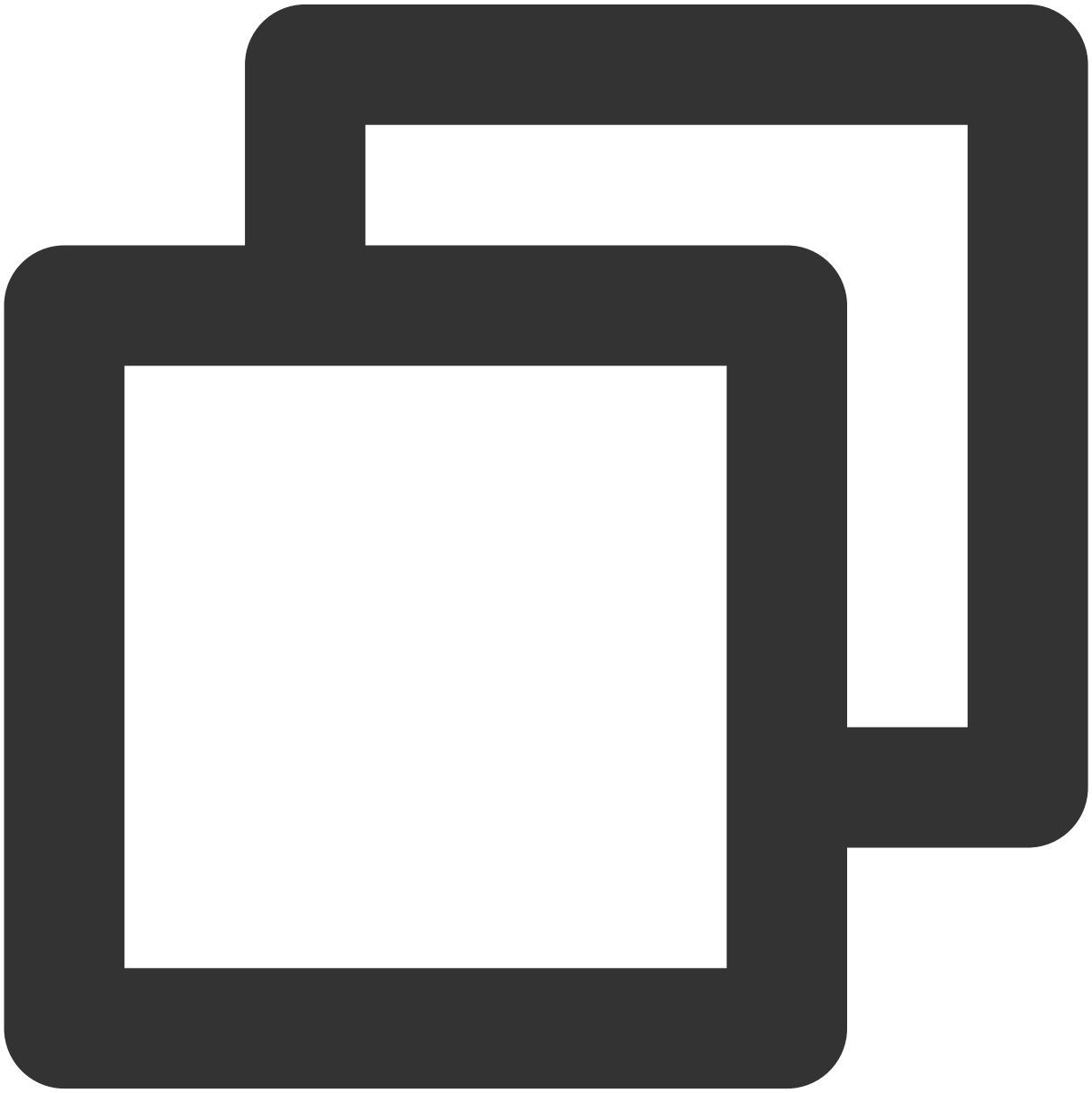
```
Page({
  data: {
    object: {
      key: 'Hello '
    },
    array: ['MINA']
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

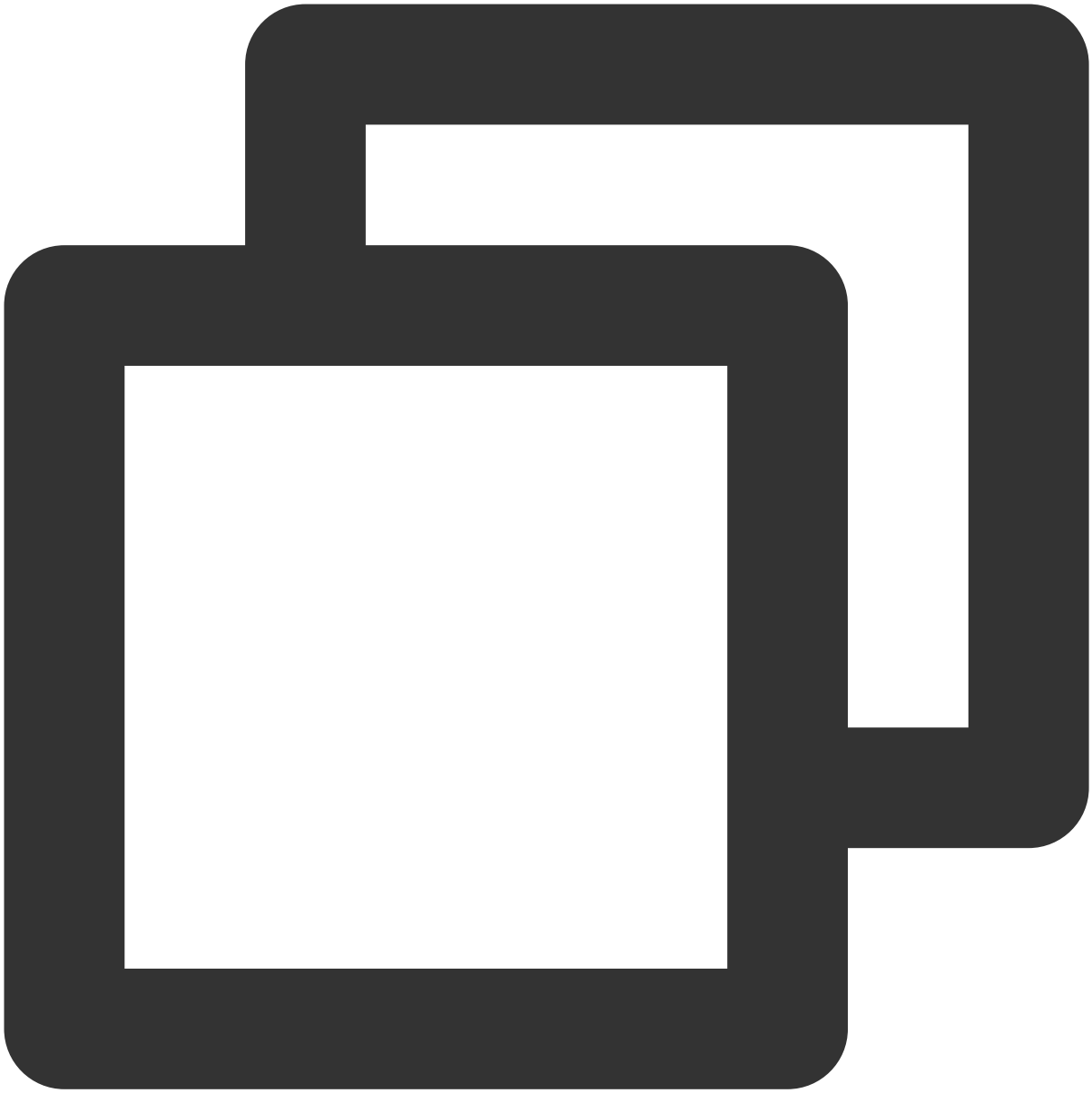
组合

也可以在 Mustache 内直接进行组合，构成新的对象或者数组。

数组



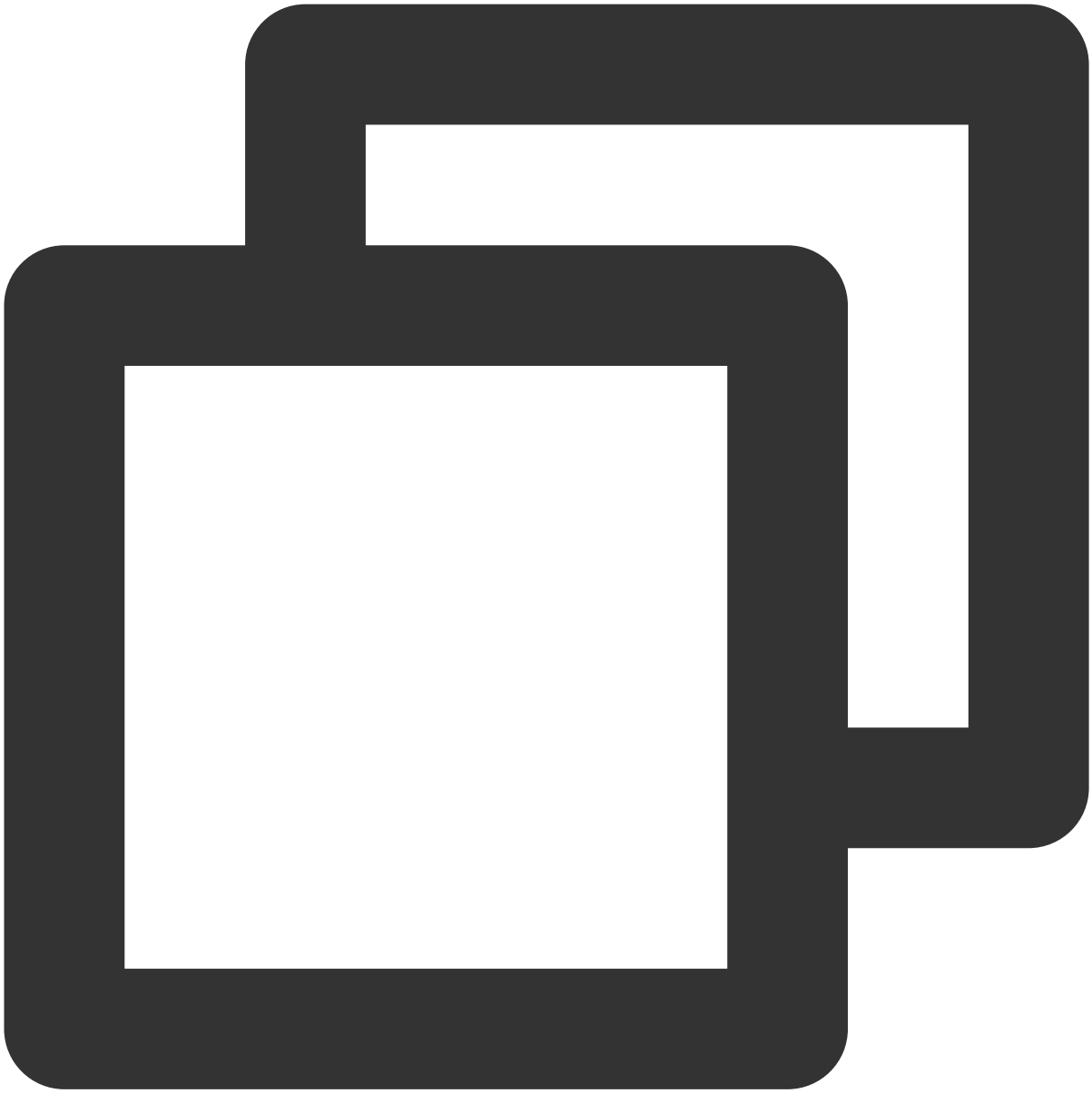
```
<view wx:for="{{[zero, 1, 2, 3, 4]}}"> {{item}} </view>
```



```
Page({
  data: {
    zero: 0
  }
})
```

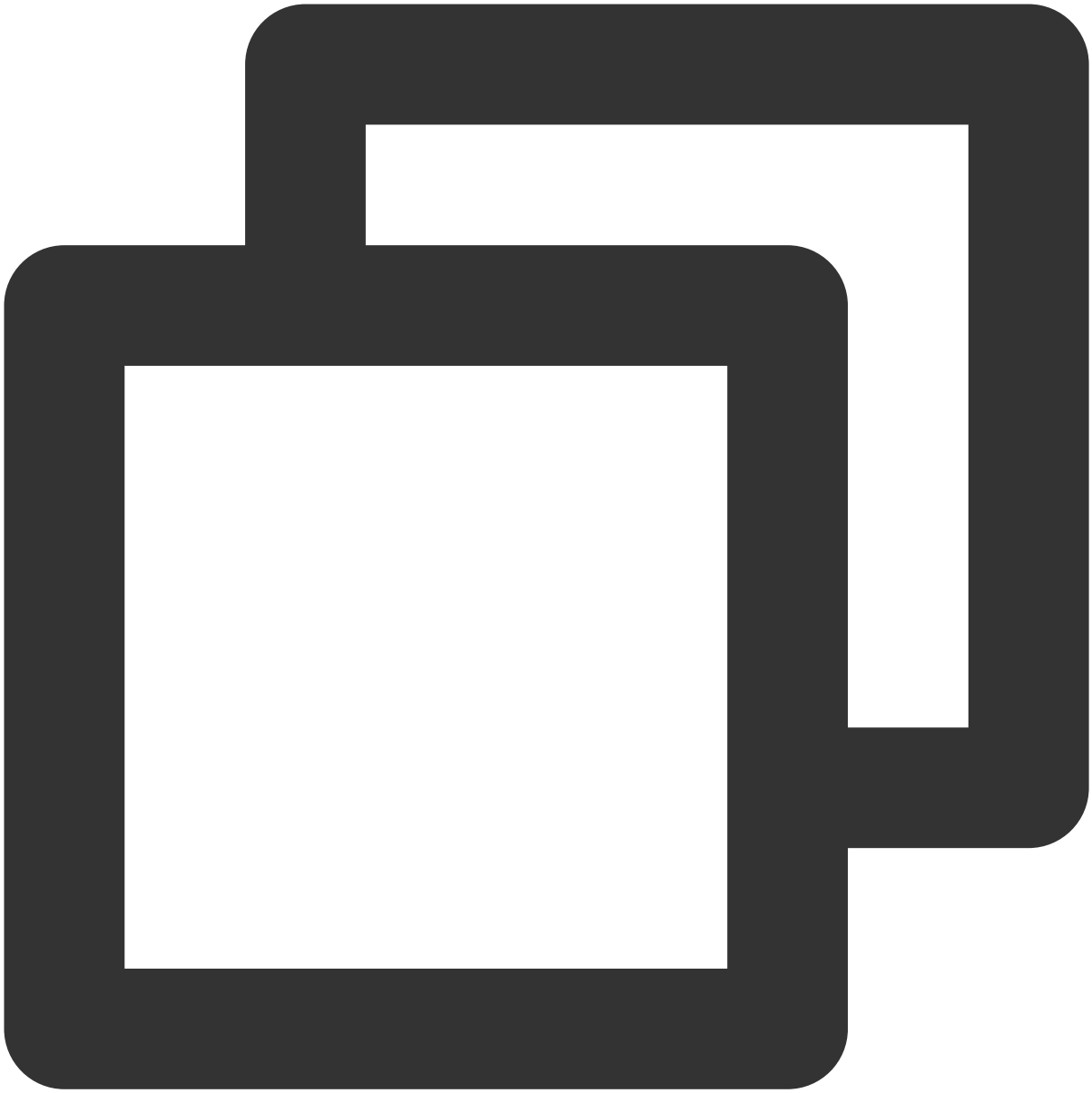
最终组合成数组 `[0, 1, 2, 3, 4]` 。

对象



```
<template is="objectCombine" data="{{for: a, bar: b}}"></template>
```

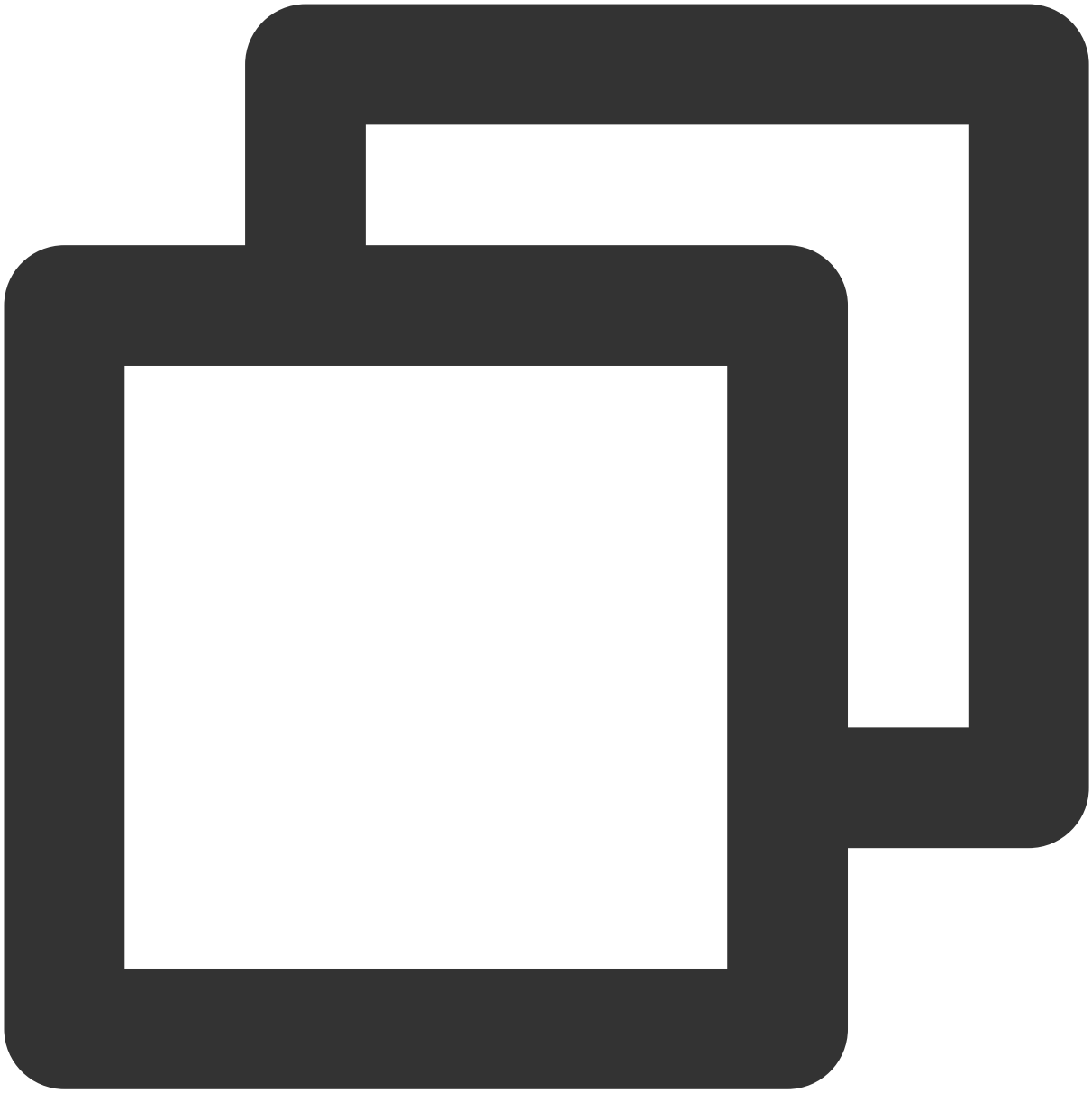
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    a: 1,
    b: 2
  }
})
```

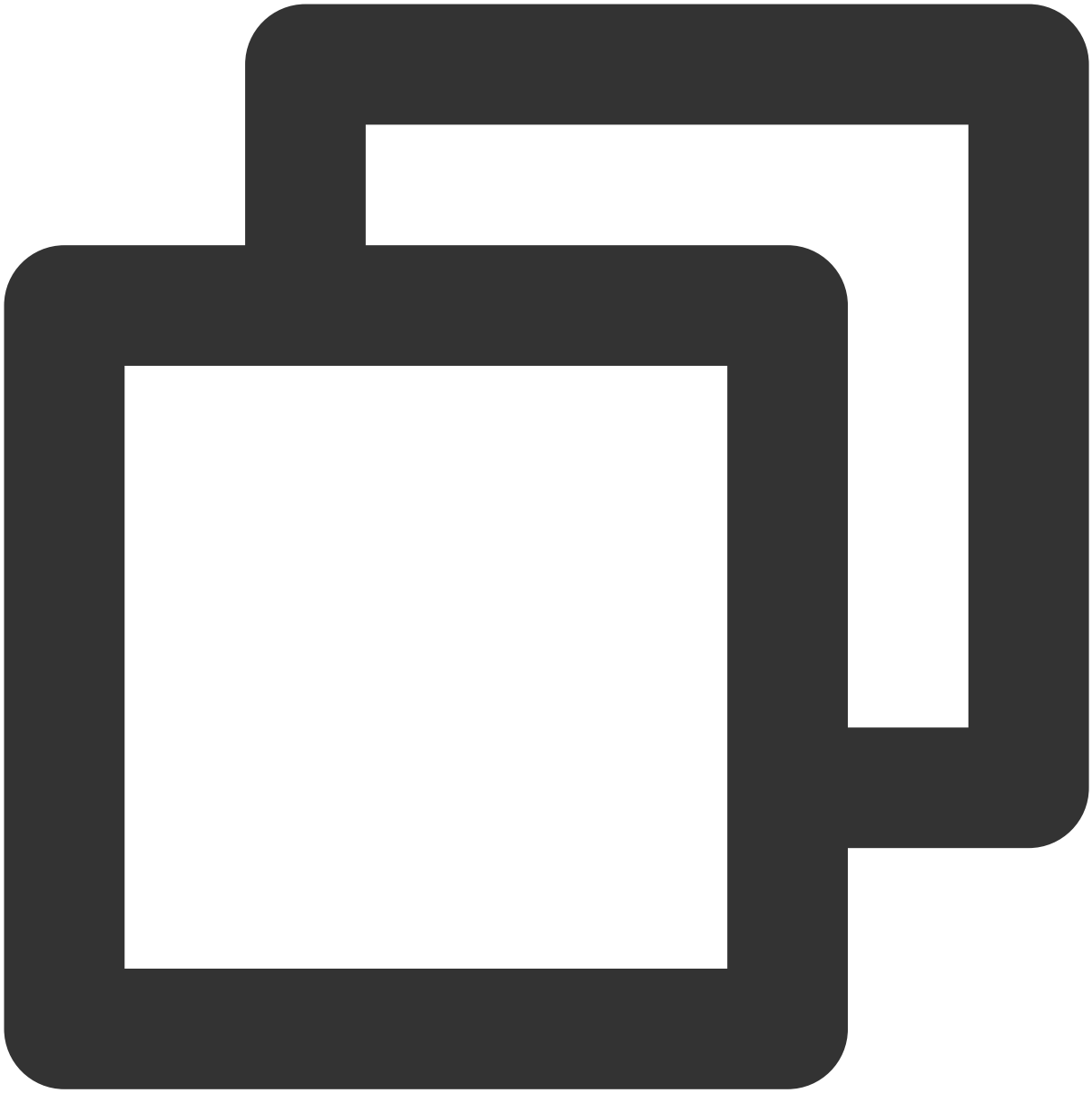
最终组合成的对象是 `{for: 1, bar: 2}`

也可以用扩展运算符 `...` 来将一个对象展开



```
<template is="objectCombine" data="{{...obj1, ...obj2, e: 5}}"></template>
```

版权所有：腾讯云计算（北京）有限责任公司



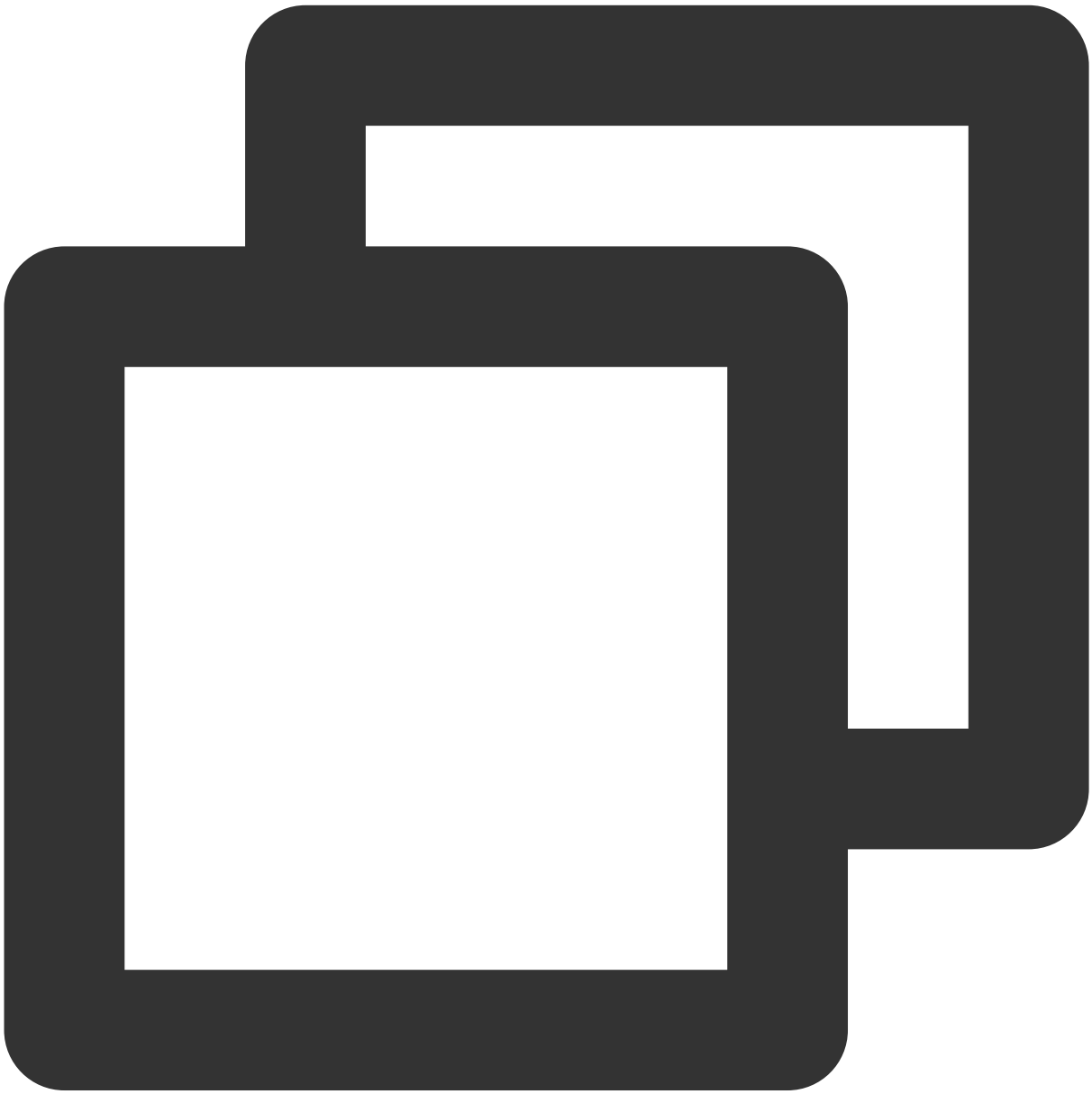
```
Page({
  data: {
    obj1: {
      a: 1,
      b: 2
    },
    obj2: {
      c: 3,
      d: 4
    }
  }
```

版权所有：腾讯云计算（北京）有限责任公司

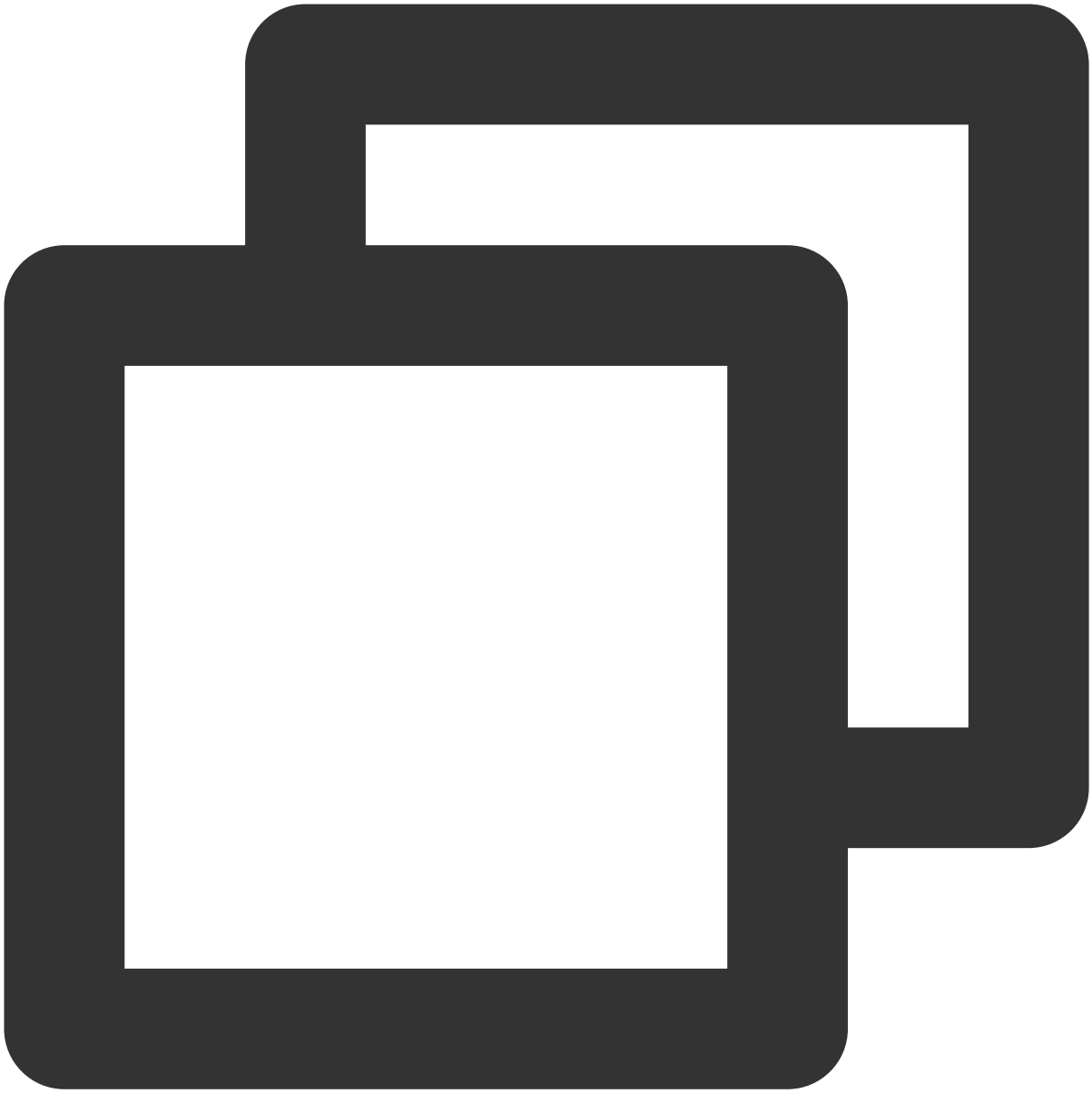
```
})
```

最终组合成的对象是 `{a: 1, b: 2, c: 3, d: 4, e: 5}` 。

如果对象的 `key` 和 `value` 相同，也可以间接地表达。



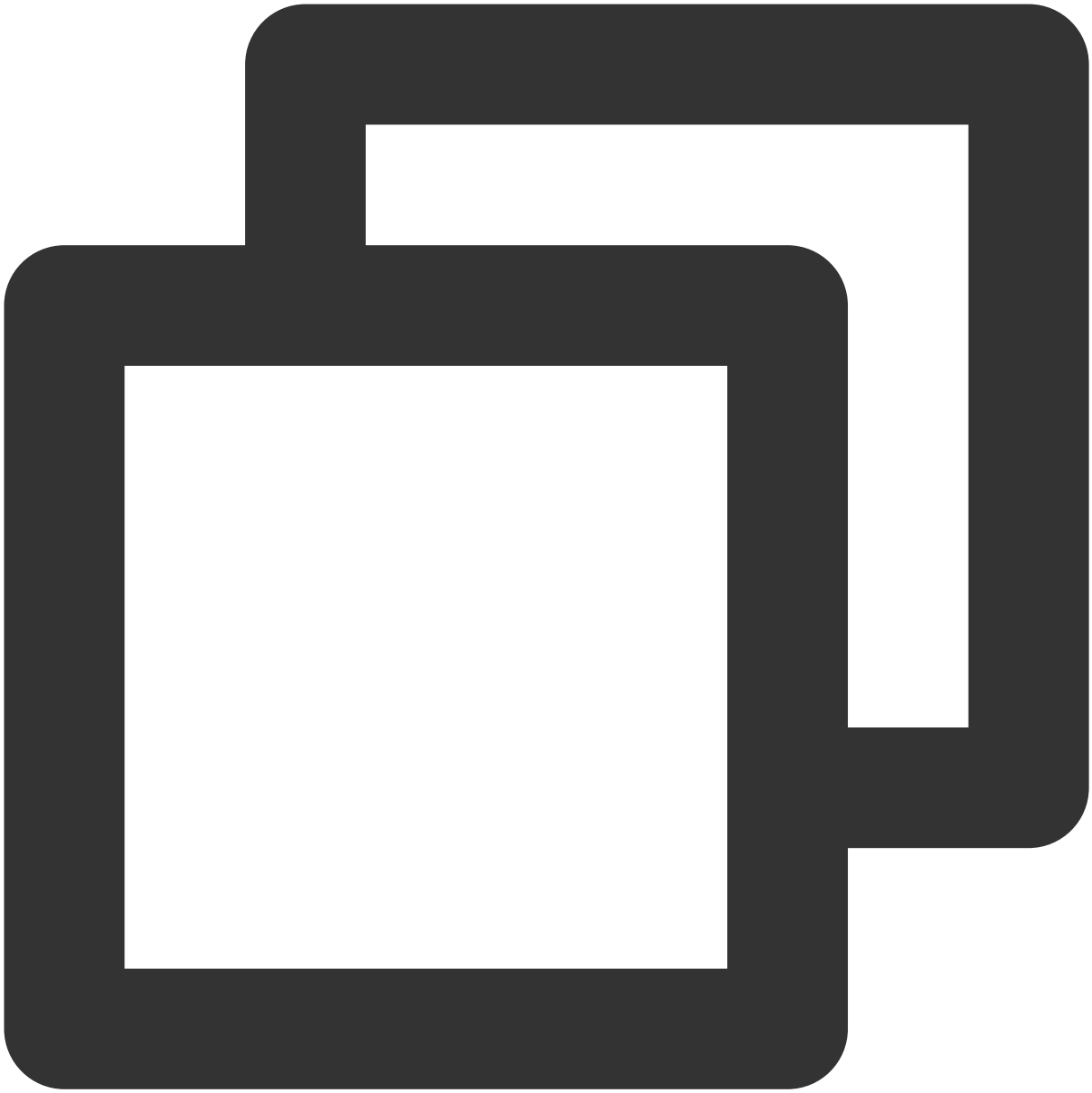
```
<template is="objectCombine" data="{{foo, bar}}"></template>
```



```
Page({
  data: {
    foo: 'my-foo',
    bar: 'my-bar'
  }
})
```

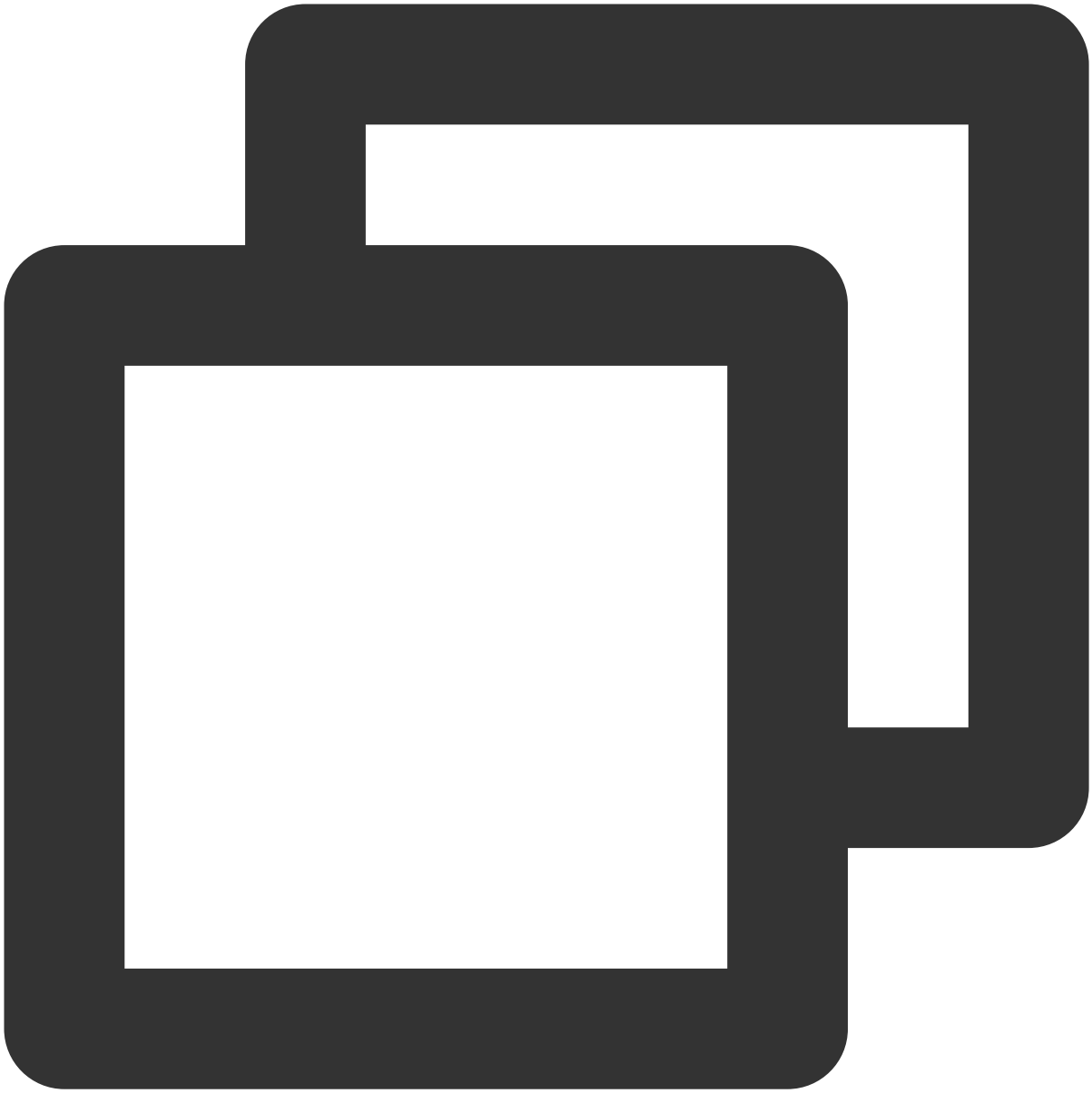
最终组合成的对象是 `{foo: 'my-foo', bar:'my-bar'}` 。

上述方式可以随意组合，但是如有存在变量名相同的情况，后边的会覆盖前面，如：



```
<template is="objectCombine" data="{{...obj1, ...obj2, a, c: 6}}"></template>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    obj1: {
      a: 1,
      b: 2
    },
    obj2: {
      b: 3,
      c: 4
    },
    a: 5
```

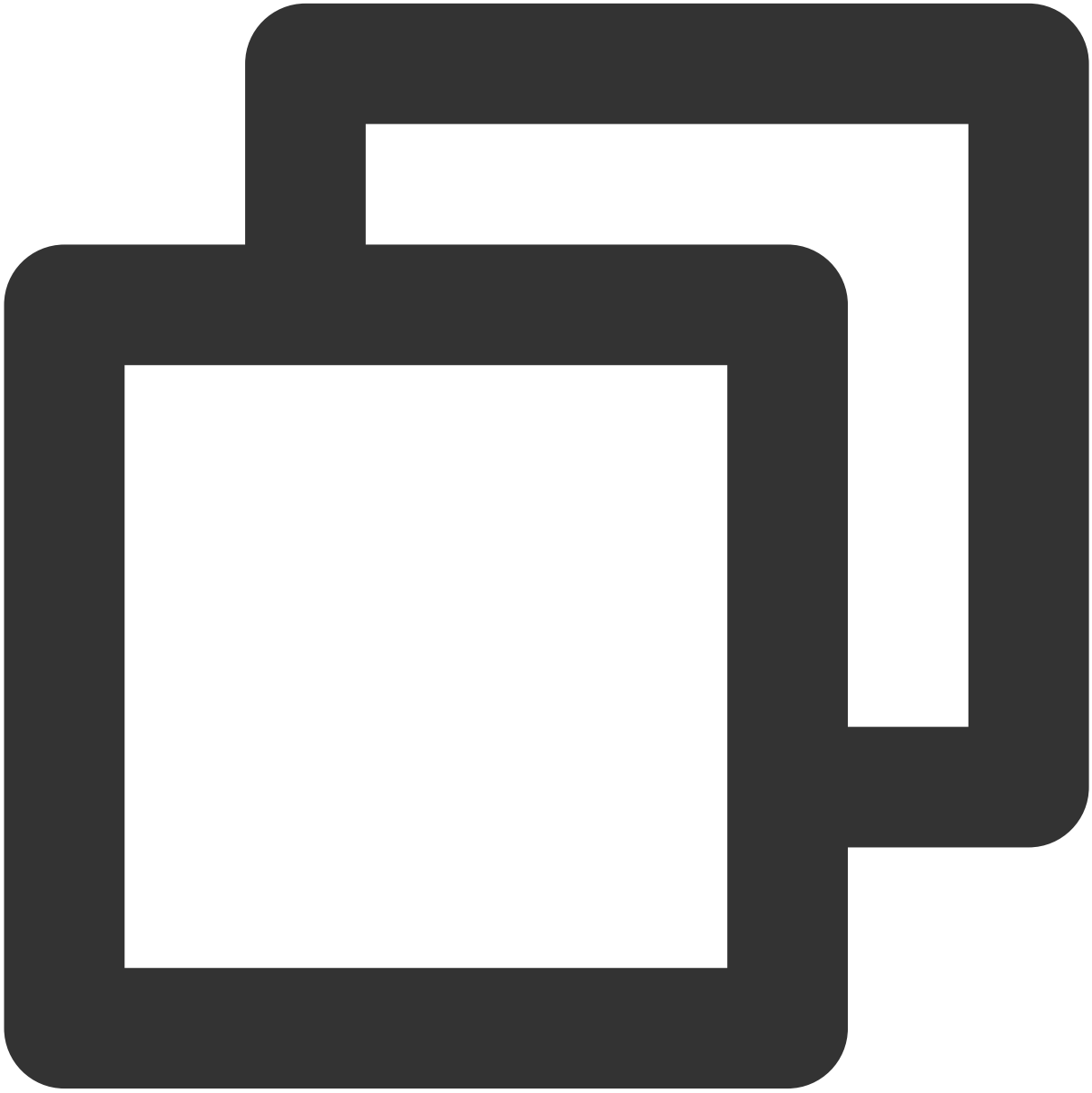
版权所有：腾讯云计算（北京）有限责任公司

```
  }
})
```

最终组合成的对象是 `{a: 5, b: 3, c: 6}` 。

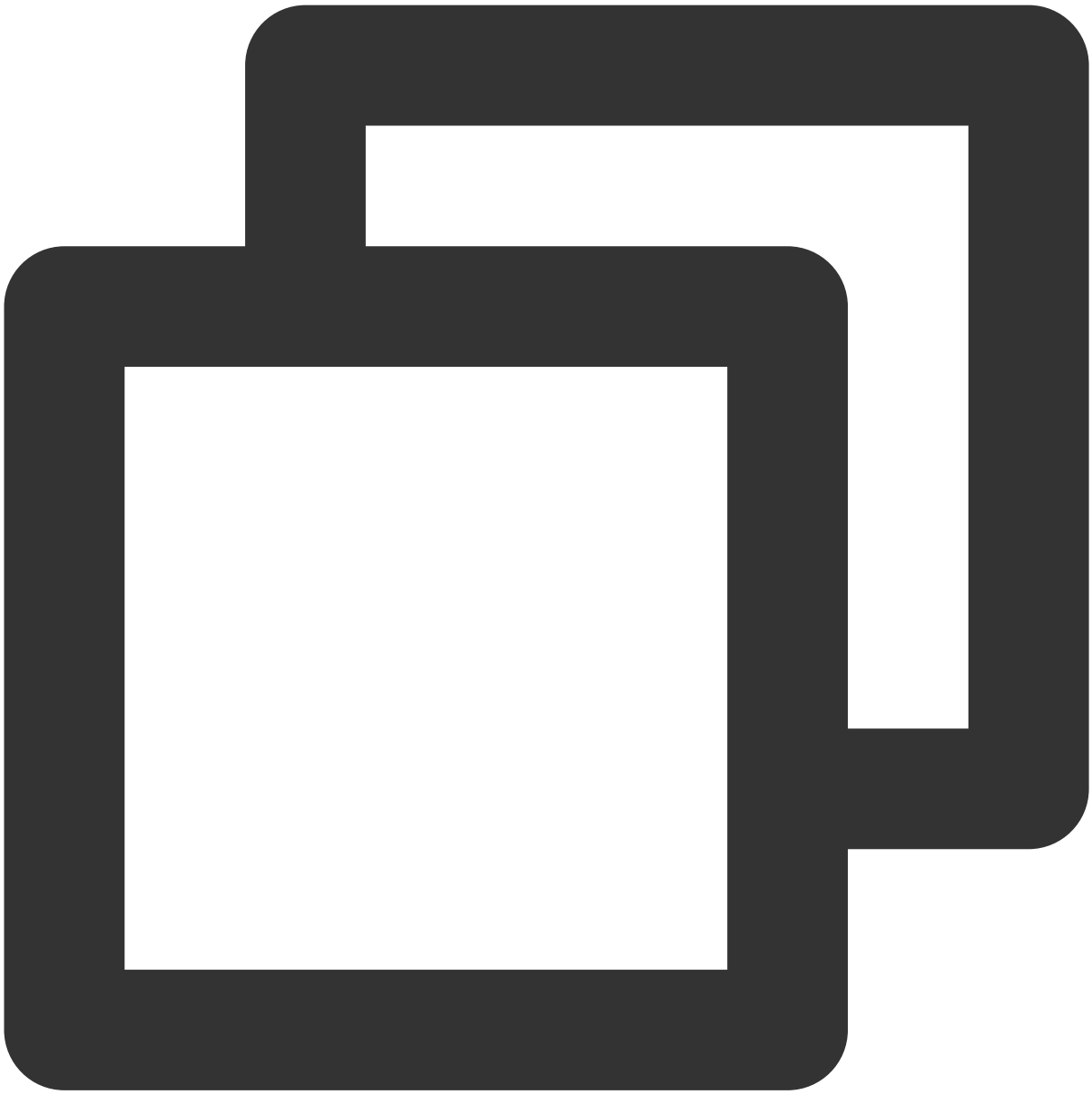
说明：

花括号和引号之间如果有空格，将最终被解析成为字符串。



```
<view wx:for="{{[1,2,3]}} ">
  {{item}}
</view>
```

等同于



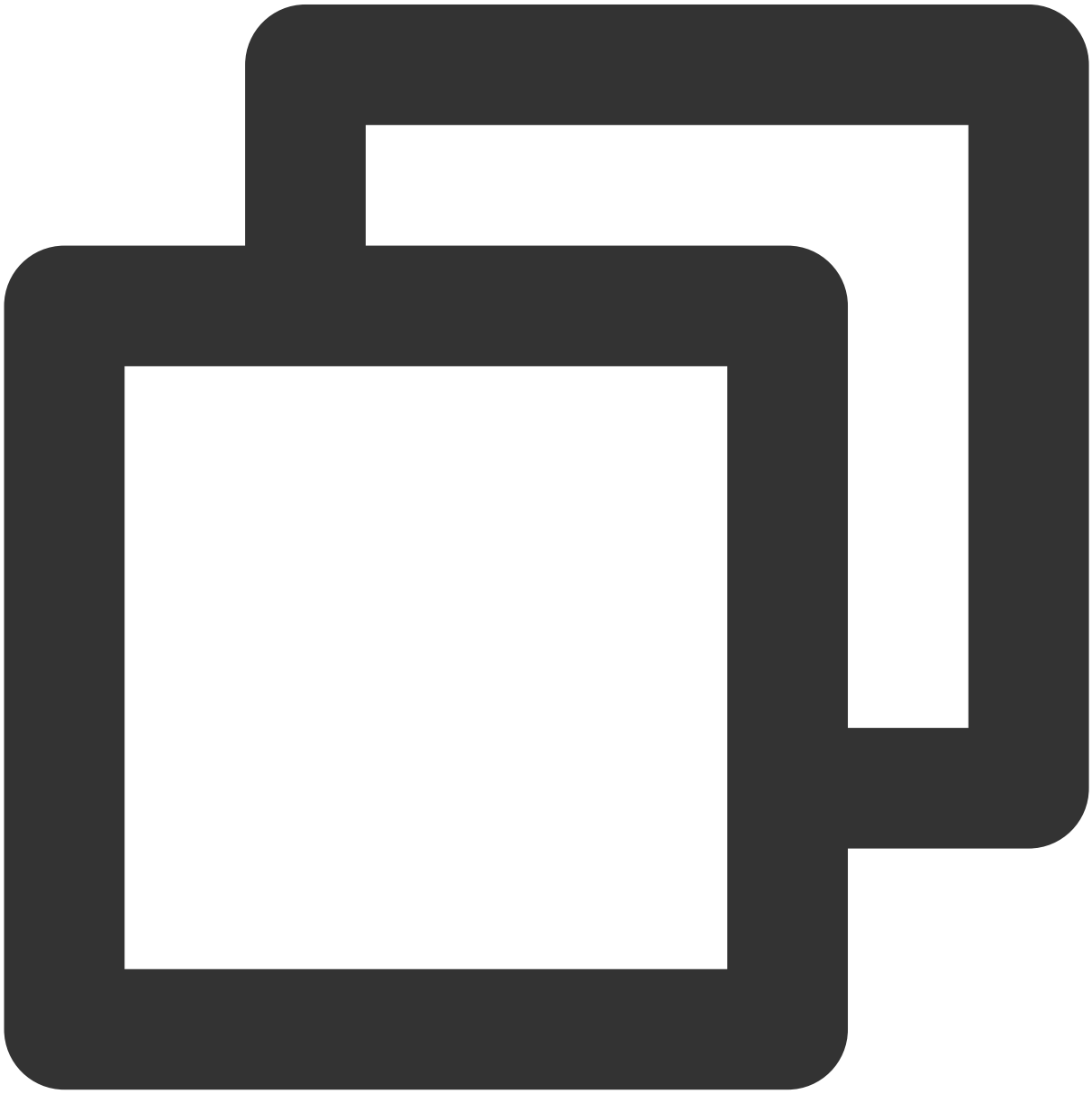
```
<view wx:for="{{[1,2,3] + ' '}}">
  {{item}}
</view>
```

列表渲染

最近更新时间：2024-07-12 17:05:25

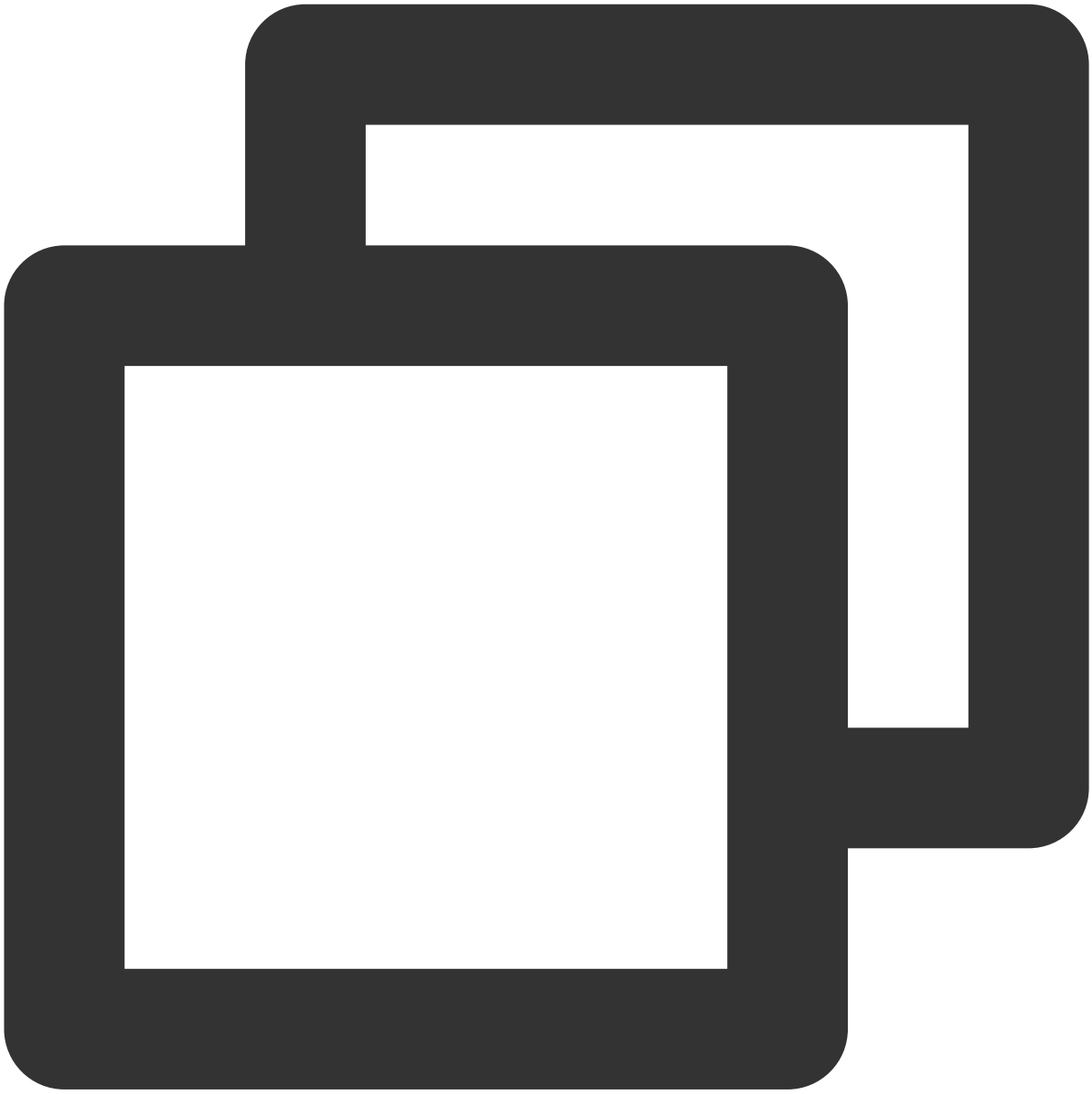
wx:for

在组件上使用 `wx:for` 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。
默认数组的当前项的下标变量名默认为 `index` ，数组当前项的变量名默认为 `item`



版权所有：腾讯云计算（北京）有限责任公司

```
<view wx:for="{{array}}">
  {{index}}: {{item.message}}
</view>
```

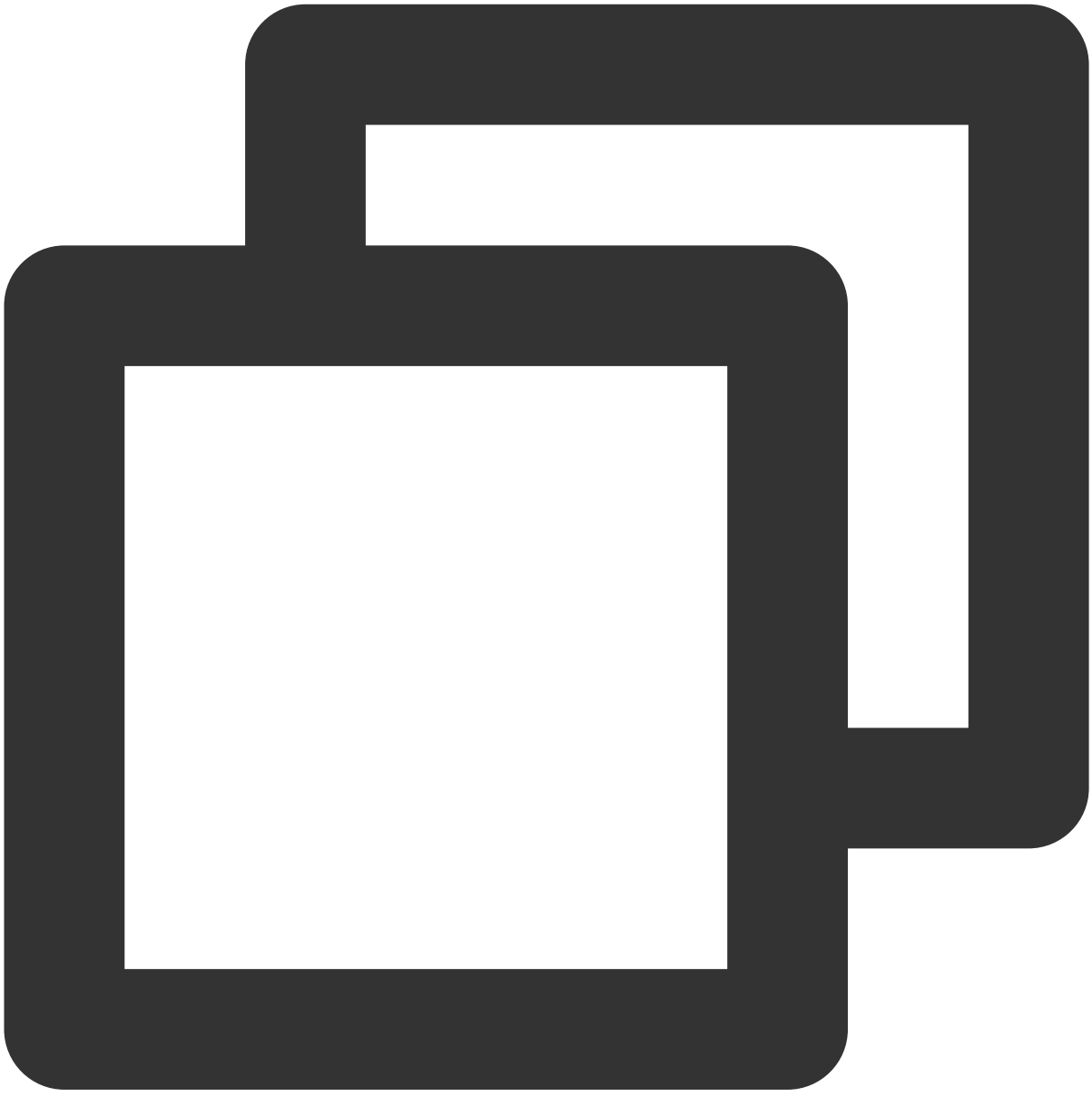


```
Page({
  data: {
    array: [{
      message: 'foo',
    }, {
      message: 'bar'
```

```
    }]
  }
})
```

使用 `wx:for-item` 可以指定数组当前元素的变量名,

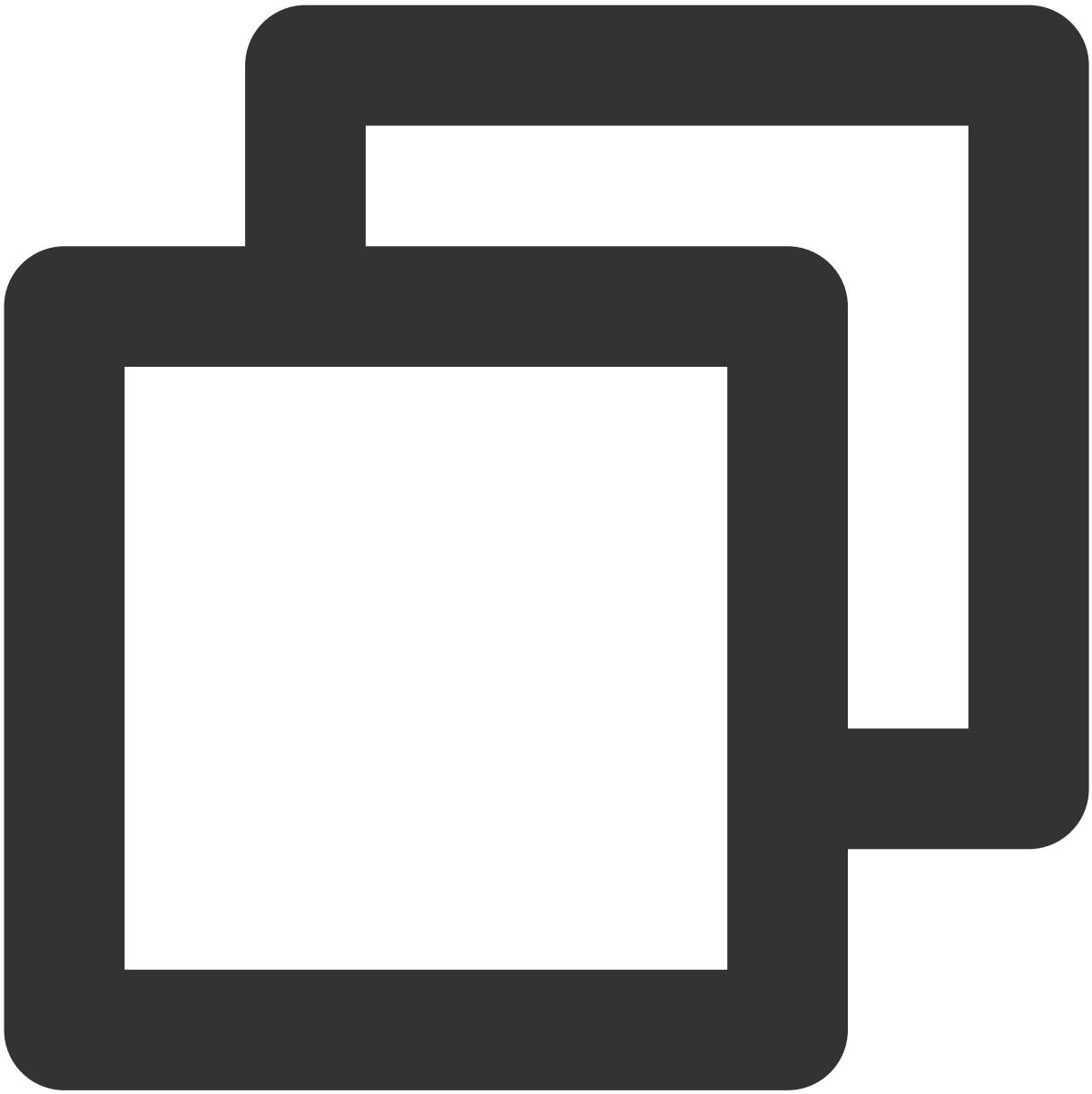
使用 `wx:for-index` 可以指定数组当前下标的变量名：



```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
</view>
```

版权所有：腾讯云计算（北京）有限责任公司

`wx:for` 也可以嵌套，下边是一个九九乘法表。

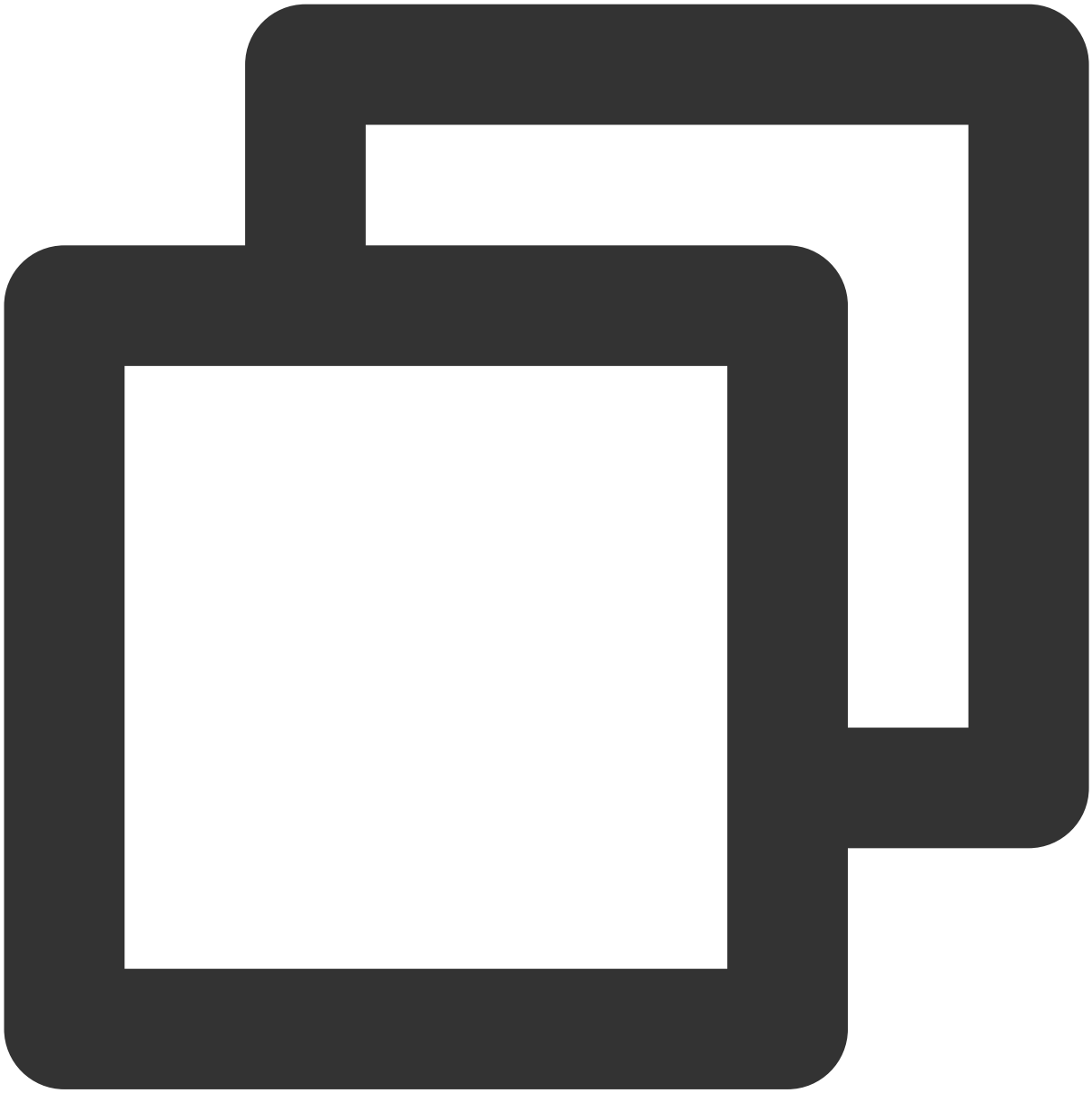


```
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    </view>
  </view>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司

block wx:for

类似 `block wx:if`，也可以将 `wx:for` 用在 `<block/>` 标签上，以渲染一个包含多节点的结构块。例如：



```
<block wx:for="{{[1, 2, 3]}}">
  <view> {{index}}: </view>
  <view> {{item}} </view>
</block>
```

版权所有：腾讯云计算（北京）有限责任公司

wx:key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 input 中的输入内容，switch 的选中状态），需要使用 `wx:key` 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供：

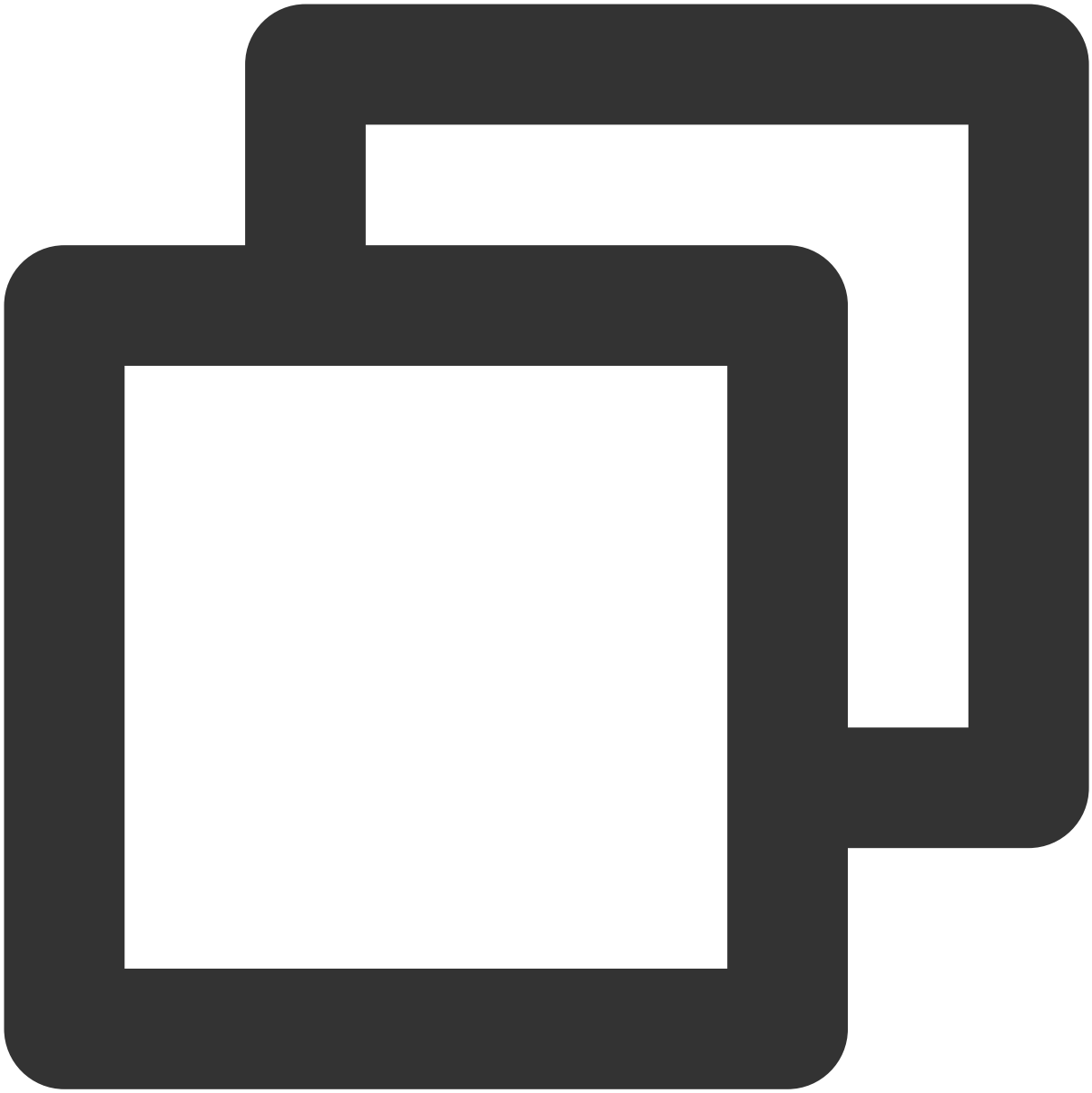
字符串，代表在 `for` 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。

保留关键字 `*this` 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字。

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。

如不提供 `wx:key`，会报一个 `warning`， 如果明确知道该列表是静态，或者不必关注其顺序，可以选择忽略。

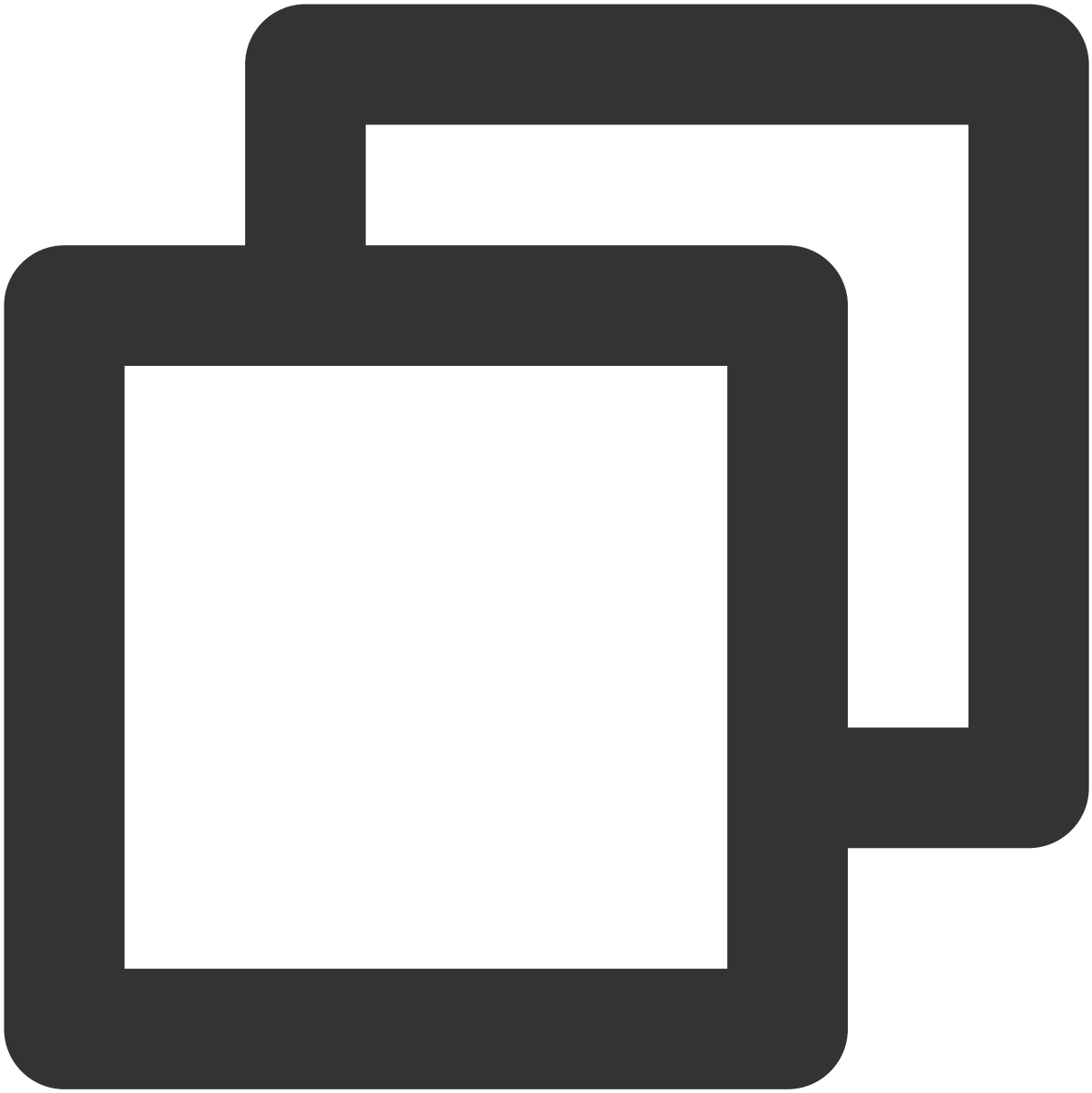
示例代码



```
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;"> {{item.id
<button bindtap="switch"> Switch </button>
<button bindtap="addToFront"> Add to the front </button>

<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;"> {{item}} <
<button bindtap="addNumberToFront"> Add to the front </button>
```

版权所有：腾讯云计算（北京）有限责任公司



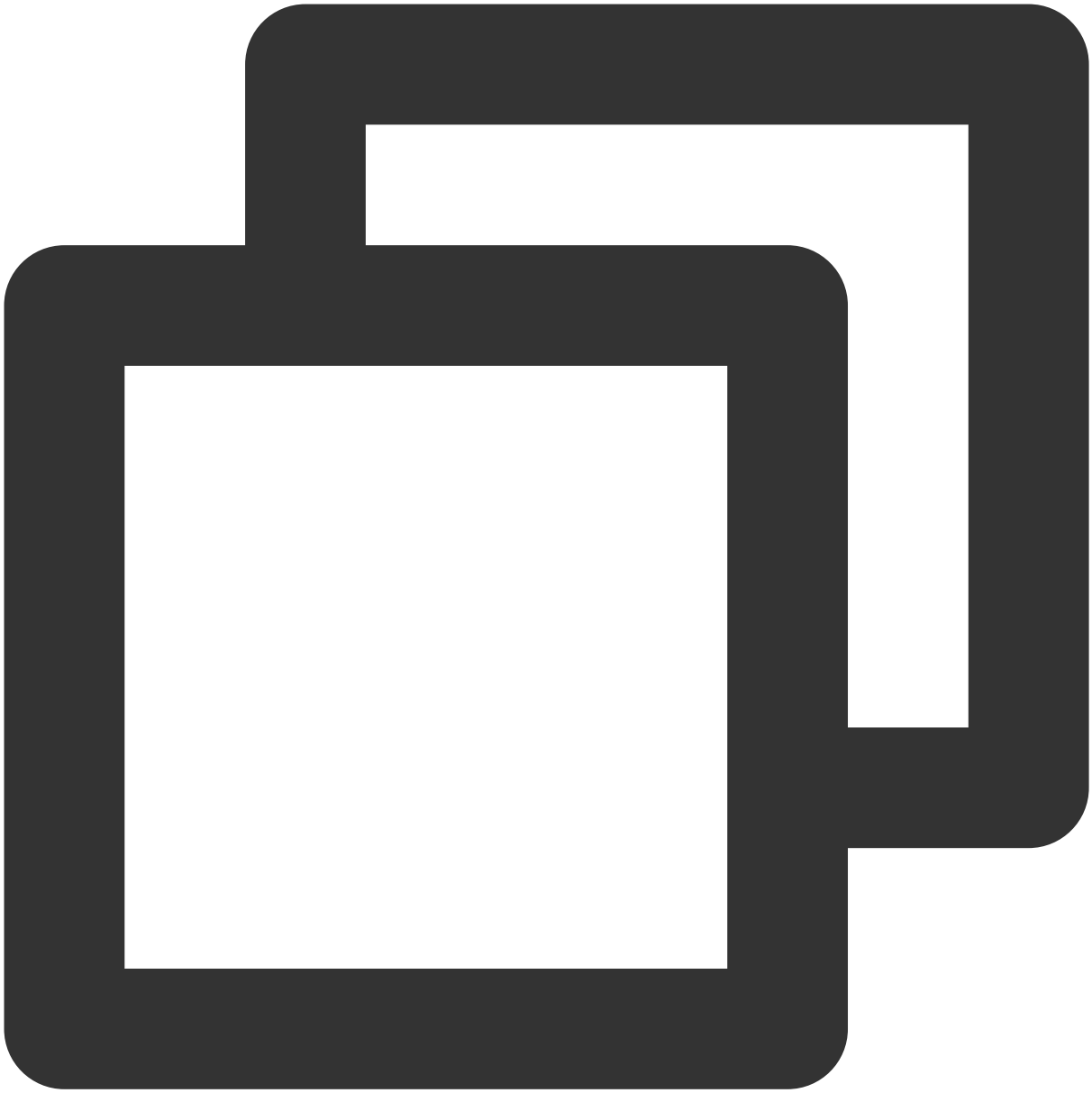
```
Page({
  data: {
    objectArray: [
      {id: 5, unique: 'unique_5'},
      {id: 4, unique: 'unique_4'},
      {id: 3, unique: 'unique_3'},
      {id: 2, unique: 'unique_2'},
      {id: 1, unique: 'unique_1'},
      {id: 0, unique: 'unique_0'},
    ],
    numberArray: [1, 2, 3, 4]
```

版权所有：腾讯云计算（北京）有限责任公司

```
  },
  switch: function(e) {
    const length = this.data.objectArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addToFront: function(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' + length}].concat(this.
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1 ].concat(this.data.n
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})
```

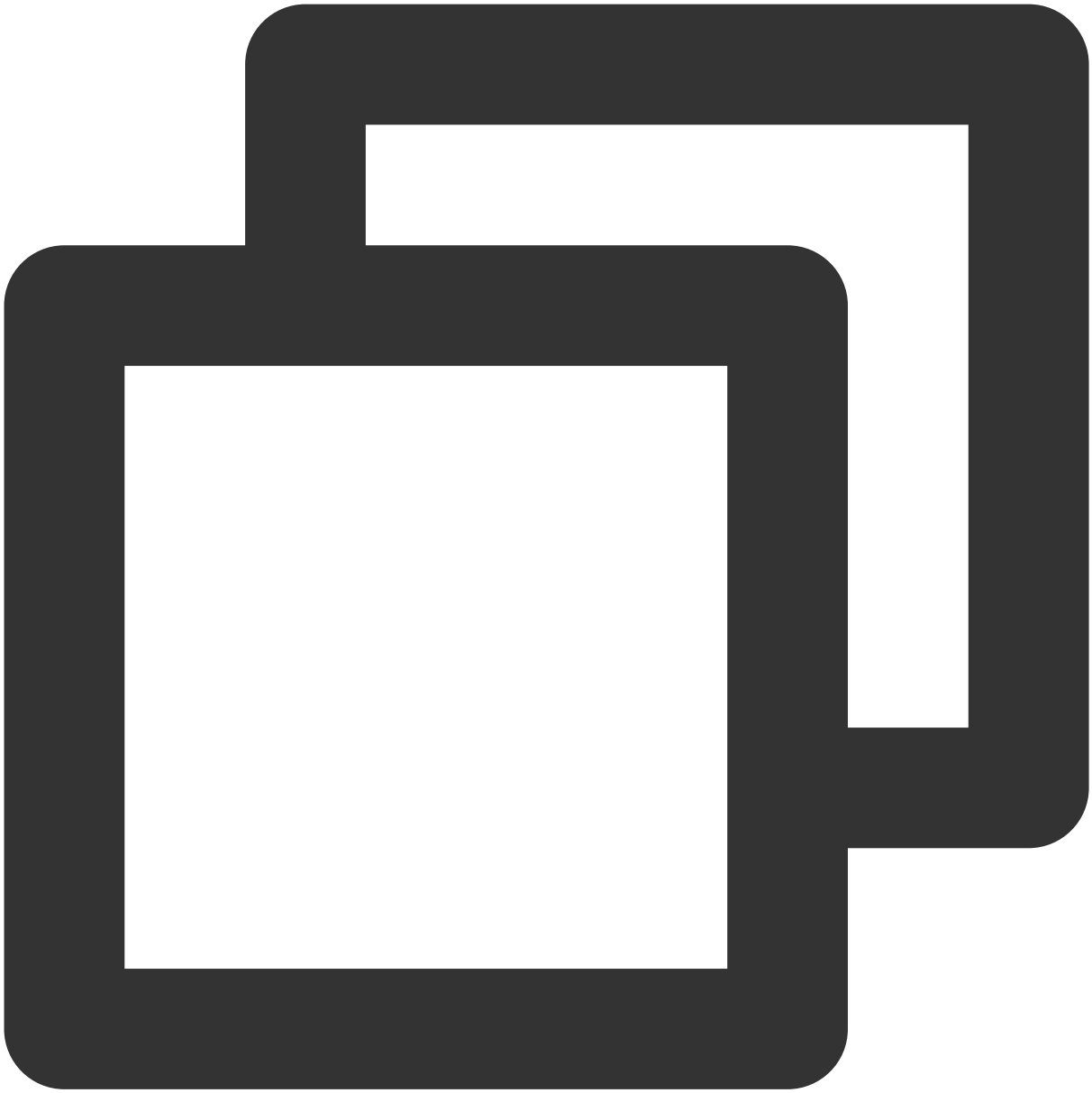
注意事项

当 `wx:for` 的值为字符串时，会将字符串解析成字符串数组。



```
<view wx:for="array">
  {{item}}
</view>
```

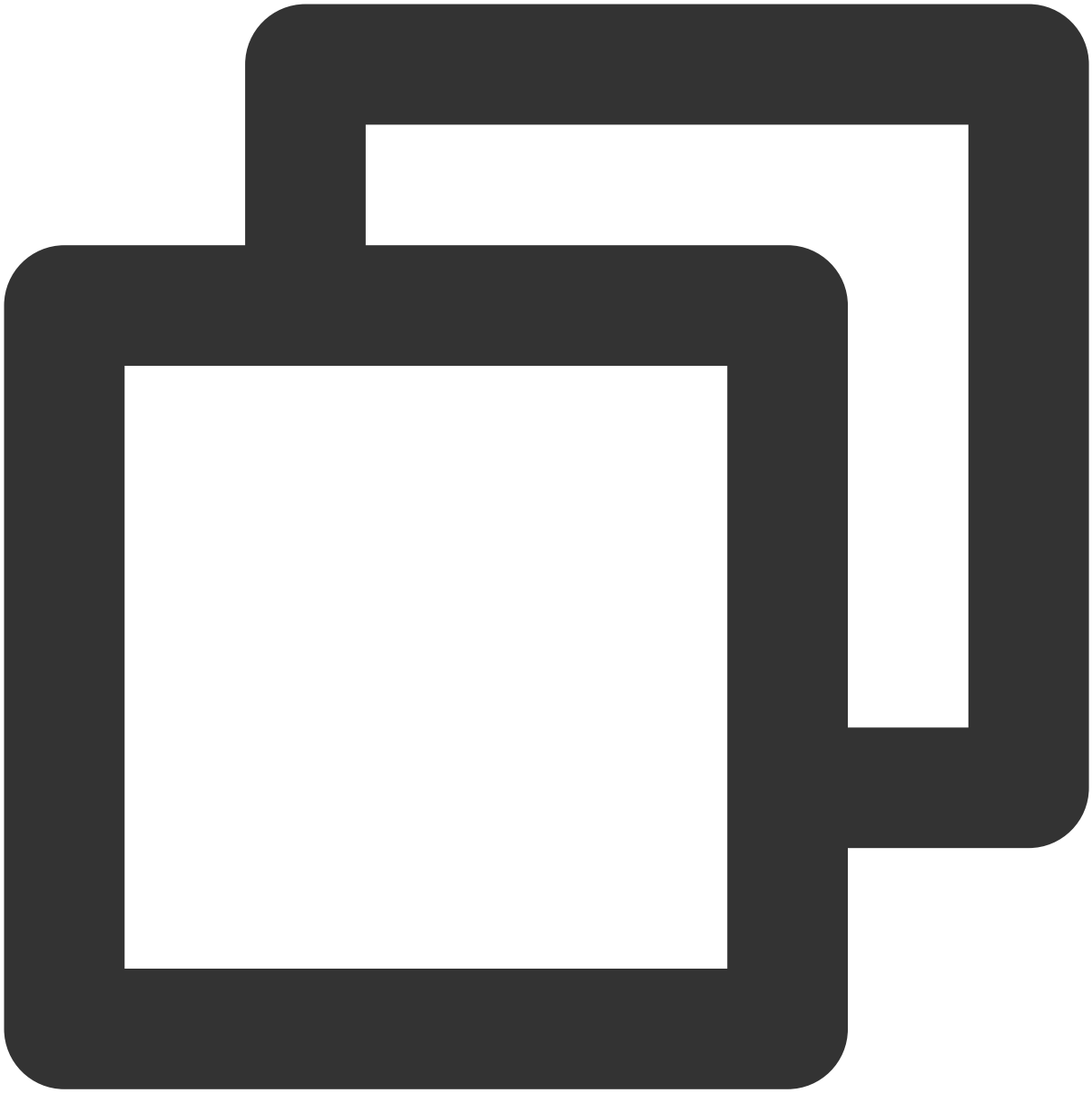
等同于



```
<view wx:for="{{['a','r','r','a','y']}}">
  {{item}}
</view>
```

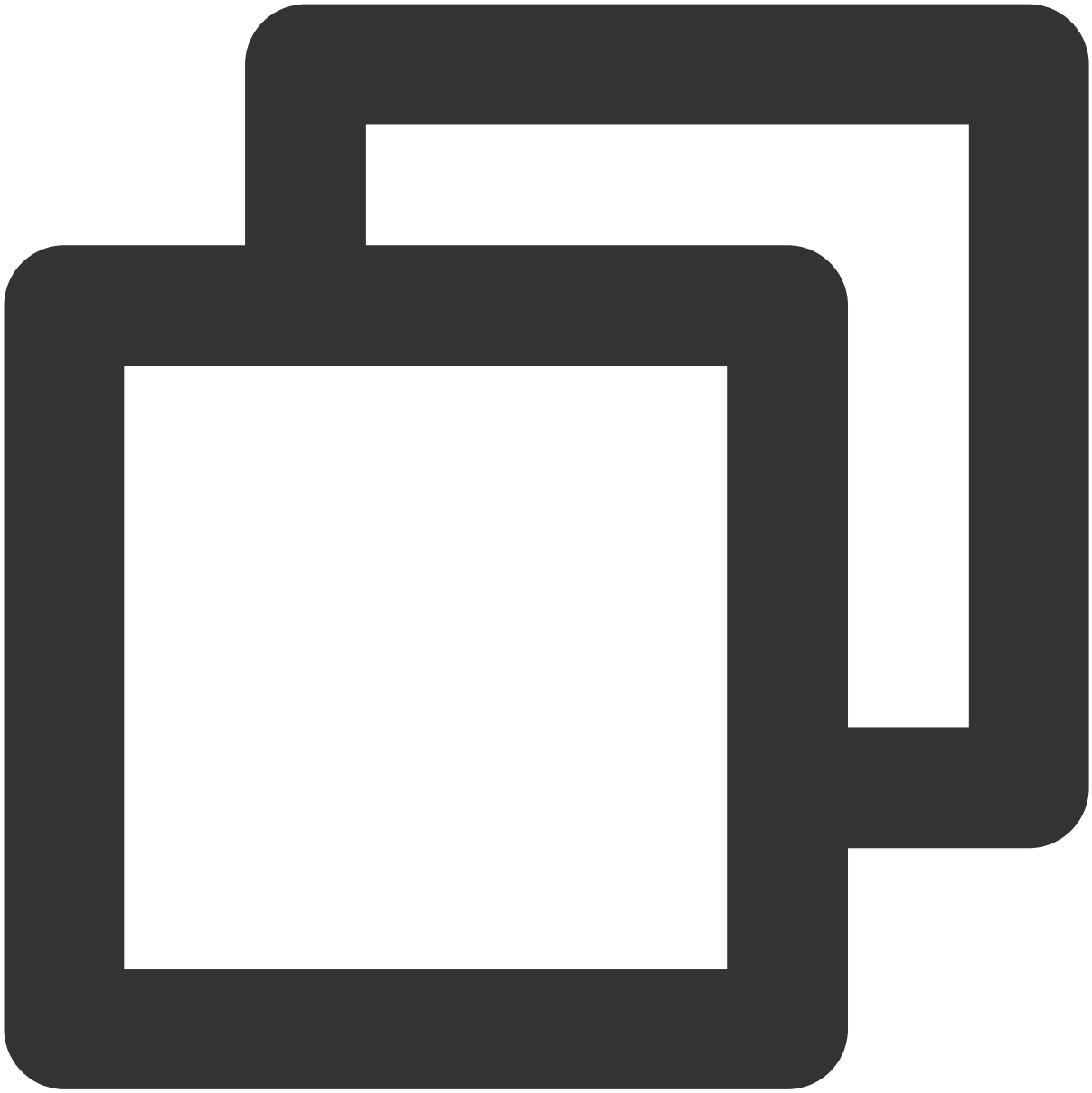
说明：

花括号和引号之间如果有空格，将最终被解析成为字符串。



```
<view wx:for="{{[1,2,3]}} ">
  {{item}}
</view>
```

等同于



```
<view wx:for="{{[1,2,3] + ' '}}" >
  {{item}}
</view>
```

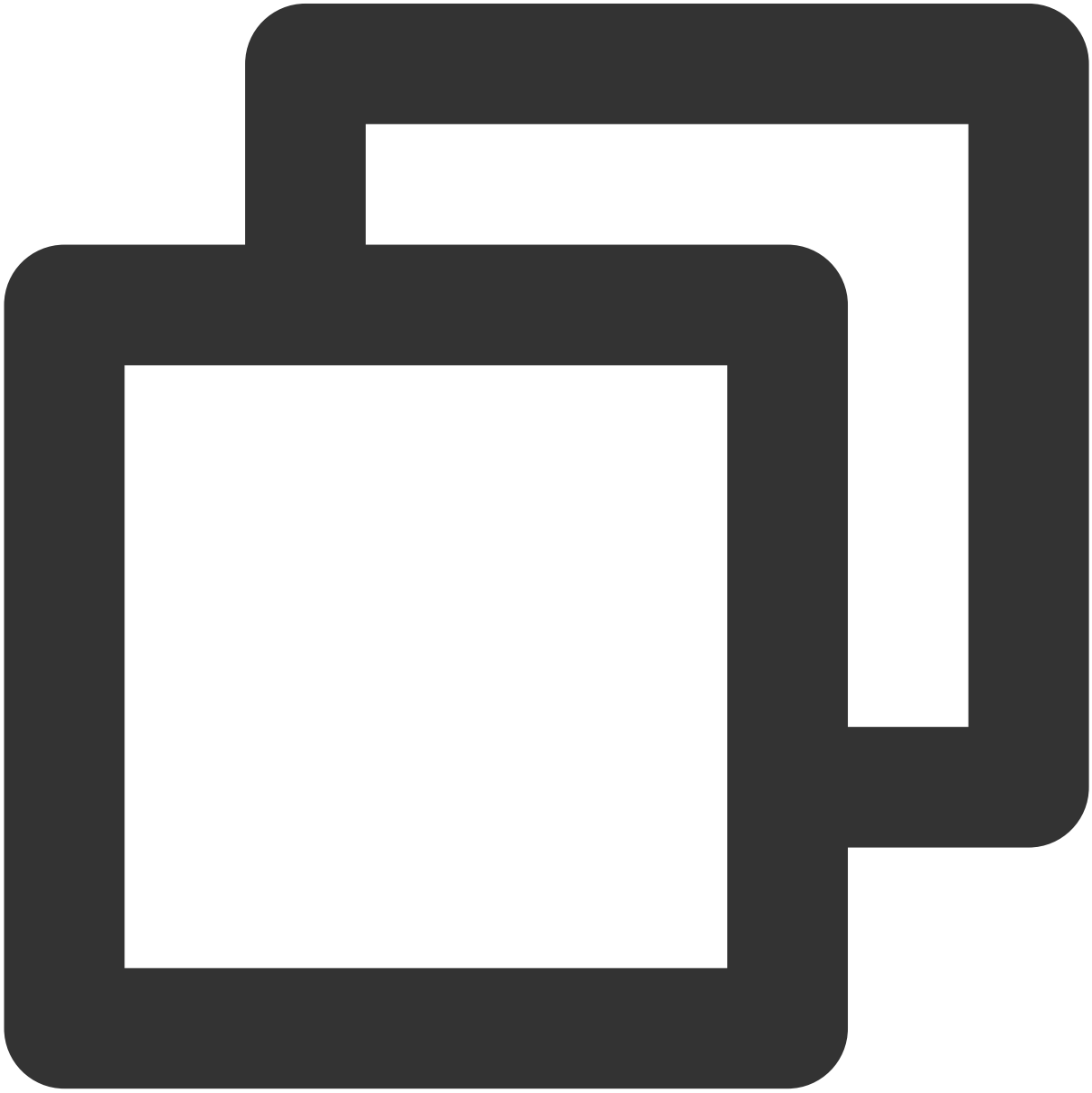
版权所有：腾讯云计算（北京）有限责任公司

条件渲染

最近更新时间：2024-07-12 17:02:55

wx:if

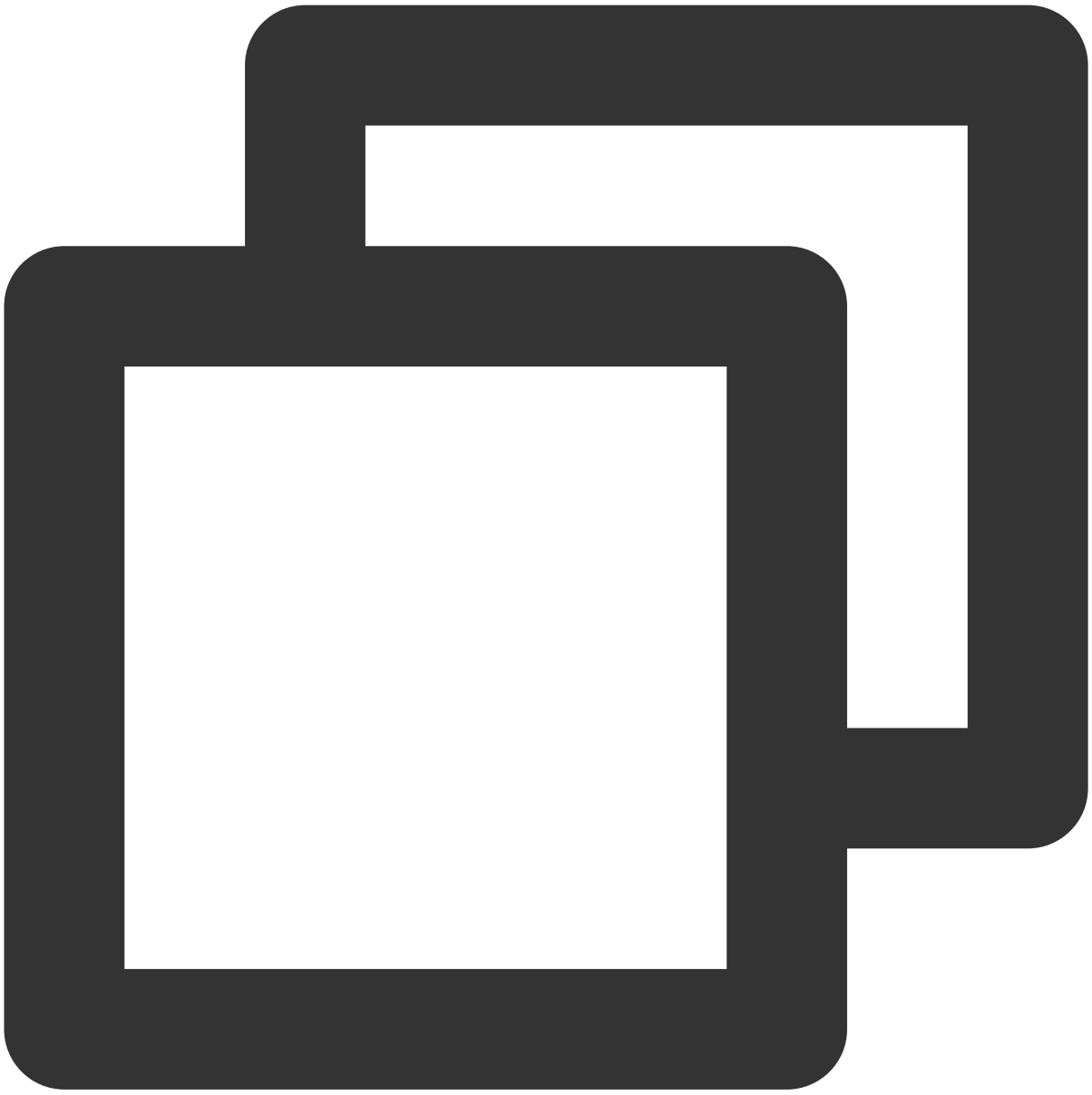
在框架中，使用 `wx:if=""` 来判断是否需要渲染该代码块：



```
<view wx:if="{{condition}}"> True </view>
```

版权所有：腾讯云计算（北京）有限责任公司

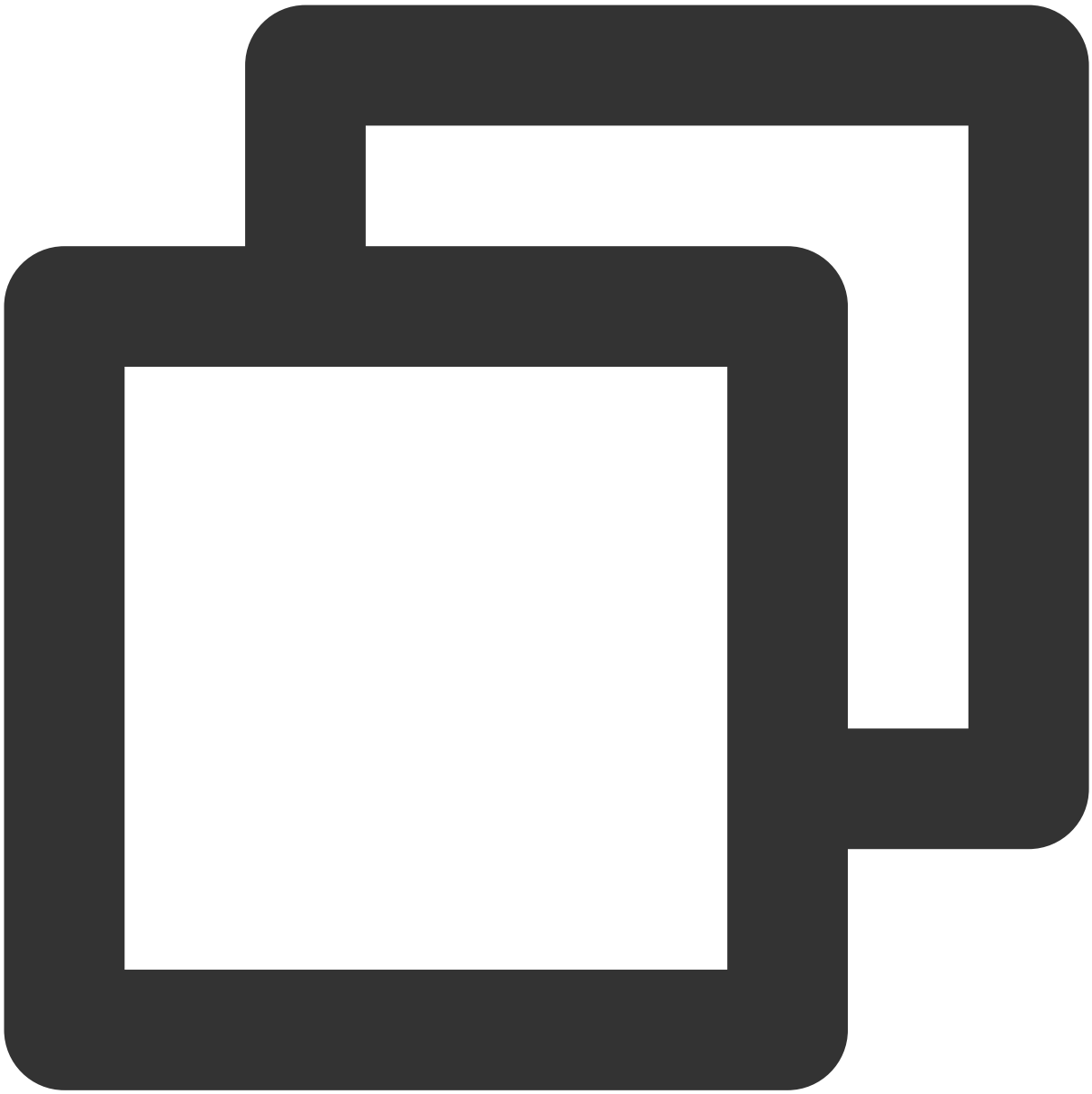
也可以用 `wx:elif` 和 `wx:else` 来添加一个 `else` 块：



```
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>
```

block wx:if

因为 `wx:if` 是一个控制属性，需要将它添加到一个标签上。如果要一次性判断多个组件标签，可以使用一个 `<block/>` 标签将多个组件包装起来，并在上边使用 `wx:if` 控制属性。



```
<block wx:if="{{true}}">
  <view> view1 </view>
  <view> view2 </view>
</block>
```

注意：

`<block/>` 并不是一个组件，它仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

版权所有：腾讯云计算（北京）有限责任公司

wx:if vs hidden

因为 `wx:if` 之中的模板也可能包含数据绑定，所以当 `wx:if` 的条件值切换时，框架有一个局部渲染的过程，因为它会确保条件块在切换时销毁或重新渲染。

同时 `wx:if` 也是**惰性的**，如果在初始渲染条件为 `false`，框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下，`hidden` 就简单的多，组件始终会被渲染，只是简单的控制显示与隐藏。

一般来说，`wx:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，如果需要频繁切换的情景下，用 `hidden` 更好，如果在运行时条件不大可能改变则 `wx:if` 较好。

版权所有：腾讯云计算（北京）有限责任公司

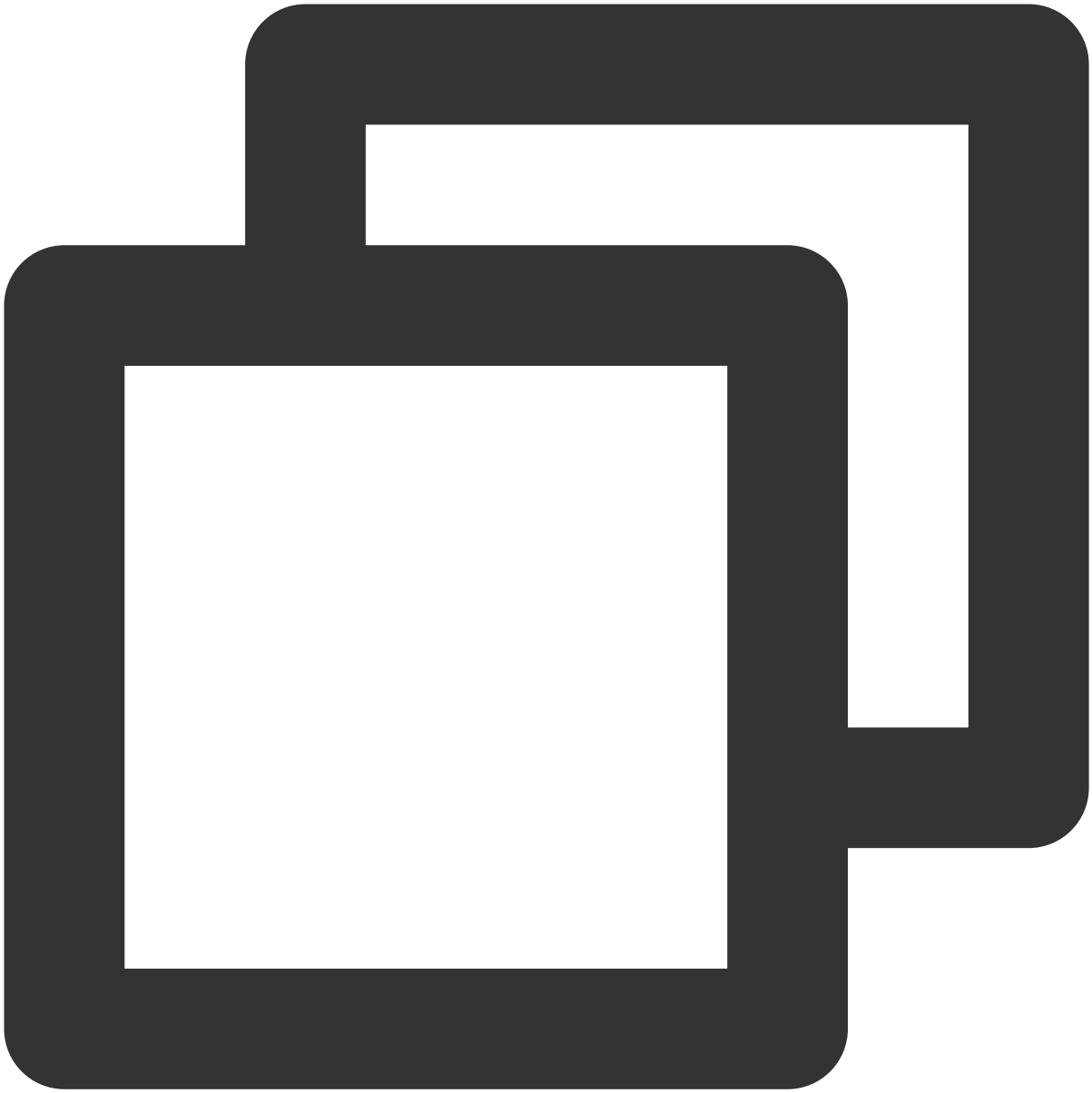
模板

最近更新时间：2024-07-12 17:00:43

WXML 提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。

定义模板

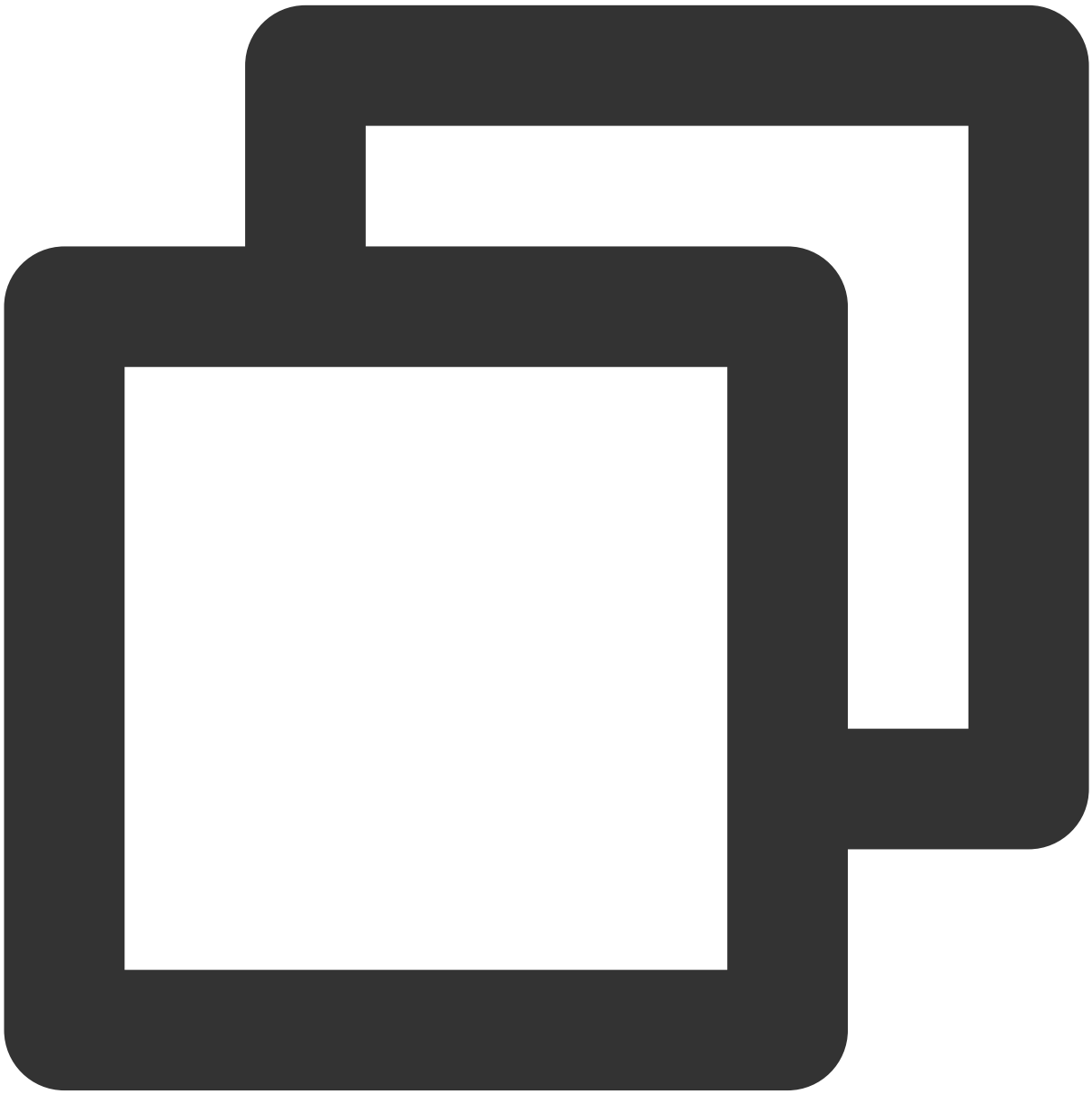
使用 name 属性，作为模板的名字。然后在 `<template/>` 内定义代码片段，如：



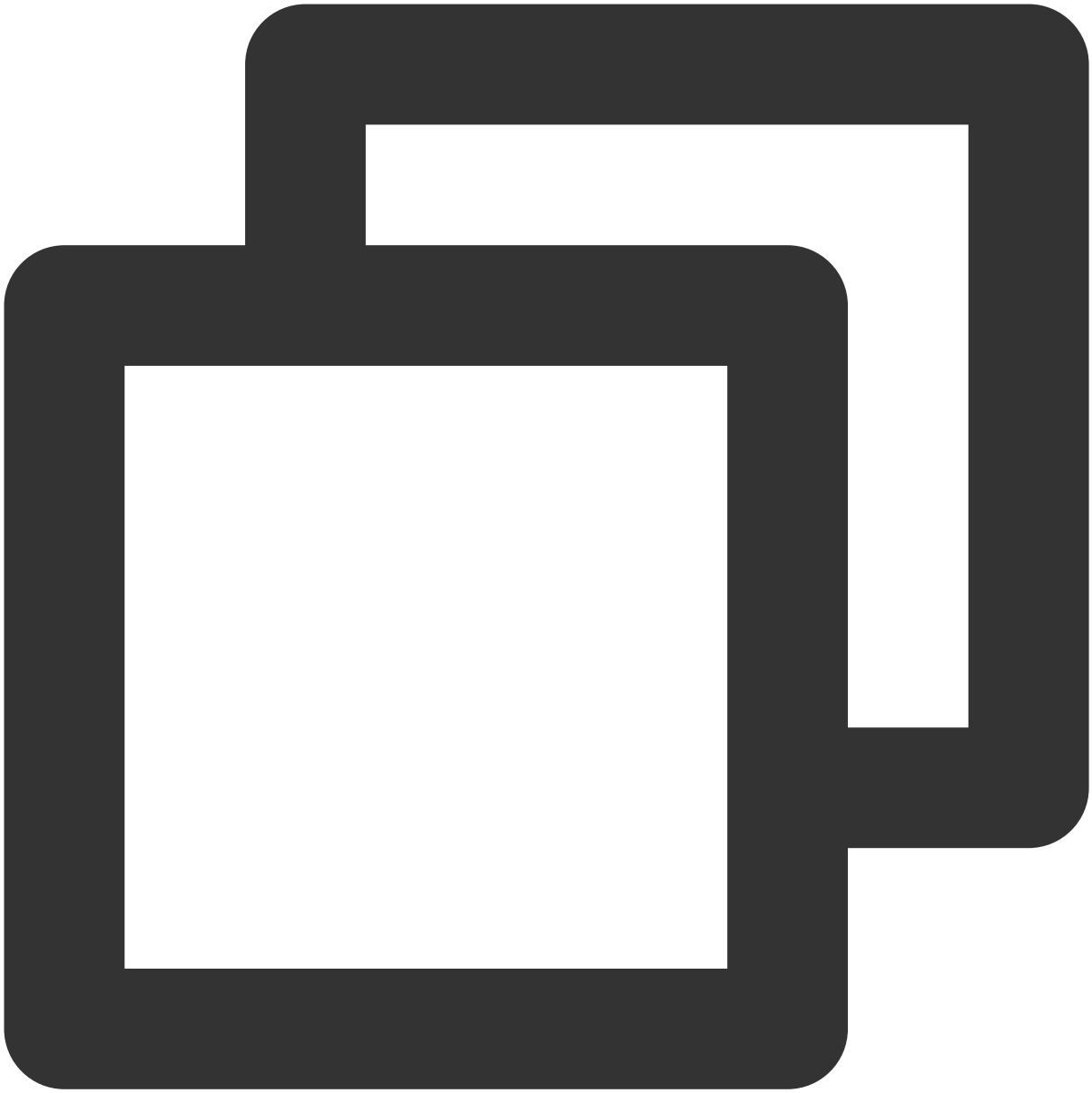
```
<!--
  index: int
  msg: string
  time: string
-->
<template name="msgItem">
  <view>
    <text> {{index}}: {{msg}} </text>
    <text> Time: {{time}} </text>
  </view>
</template>
```

使用模板

使用 is 属性，声明需要的使用的模板，然后将模板所需要的 data 传入，如：



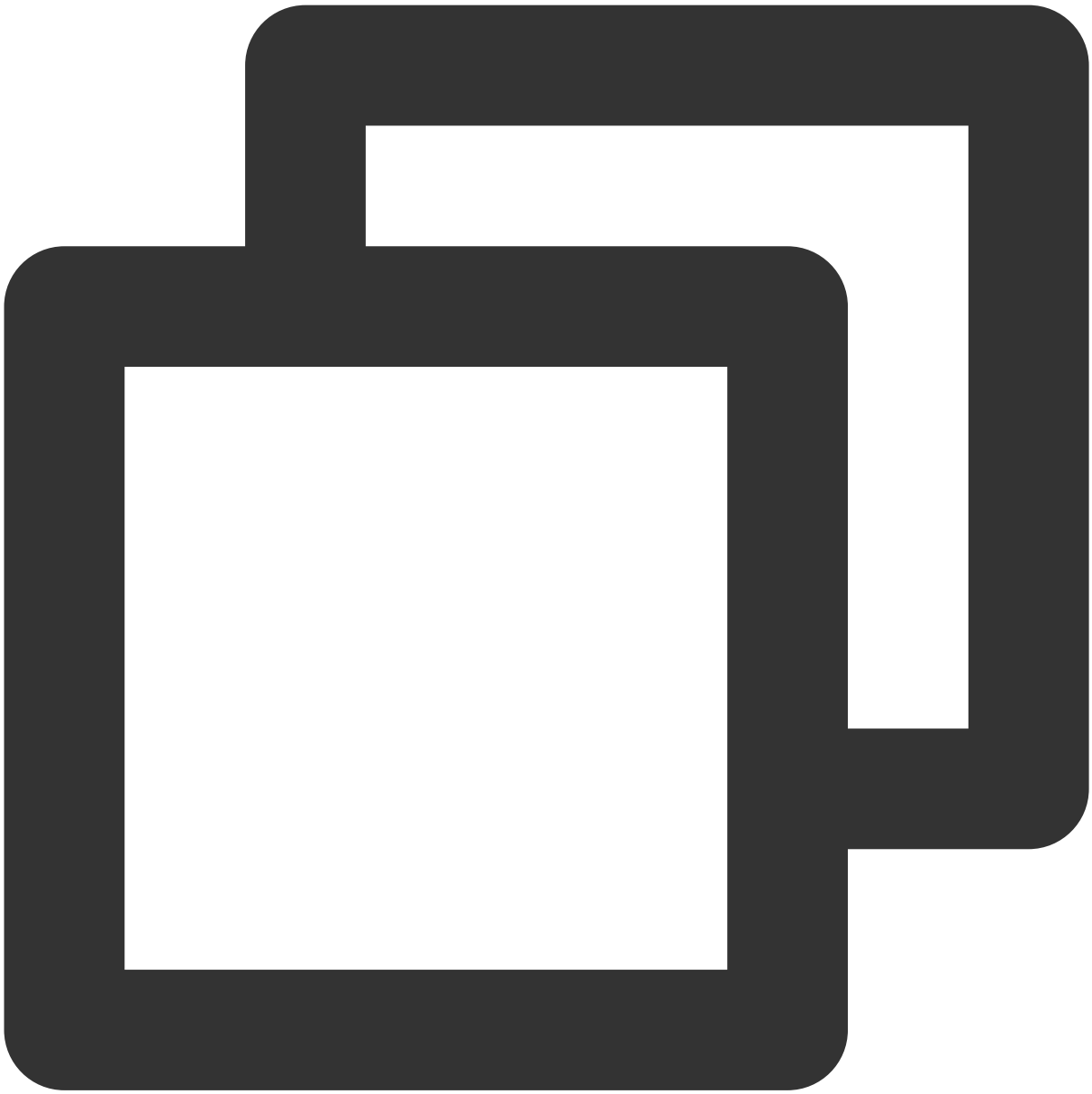
```
<template is="msgItem" data="{{...item}}"/>
```



```
Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2016-09-15'
    }
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

`is` 属性可以使用 Mustache 语法，来动态决定具体需要渲染哪个模板：



```
<template name="odd">
  <view> odd </view>
</template>
<template name="even">
  <view> even </view>
</template>

<block wx:for="{{[1, 2, 3, 4, 5]}}">
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}"/>
```

```
</block>
```

模板的作用域

模板拥有自己的作用域，只能使用 data 传入的数据以及模板定义文件中定义的 `<wxs />` 模块。

版权所有：腾讯云计算（北京）有限责任公司

引用

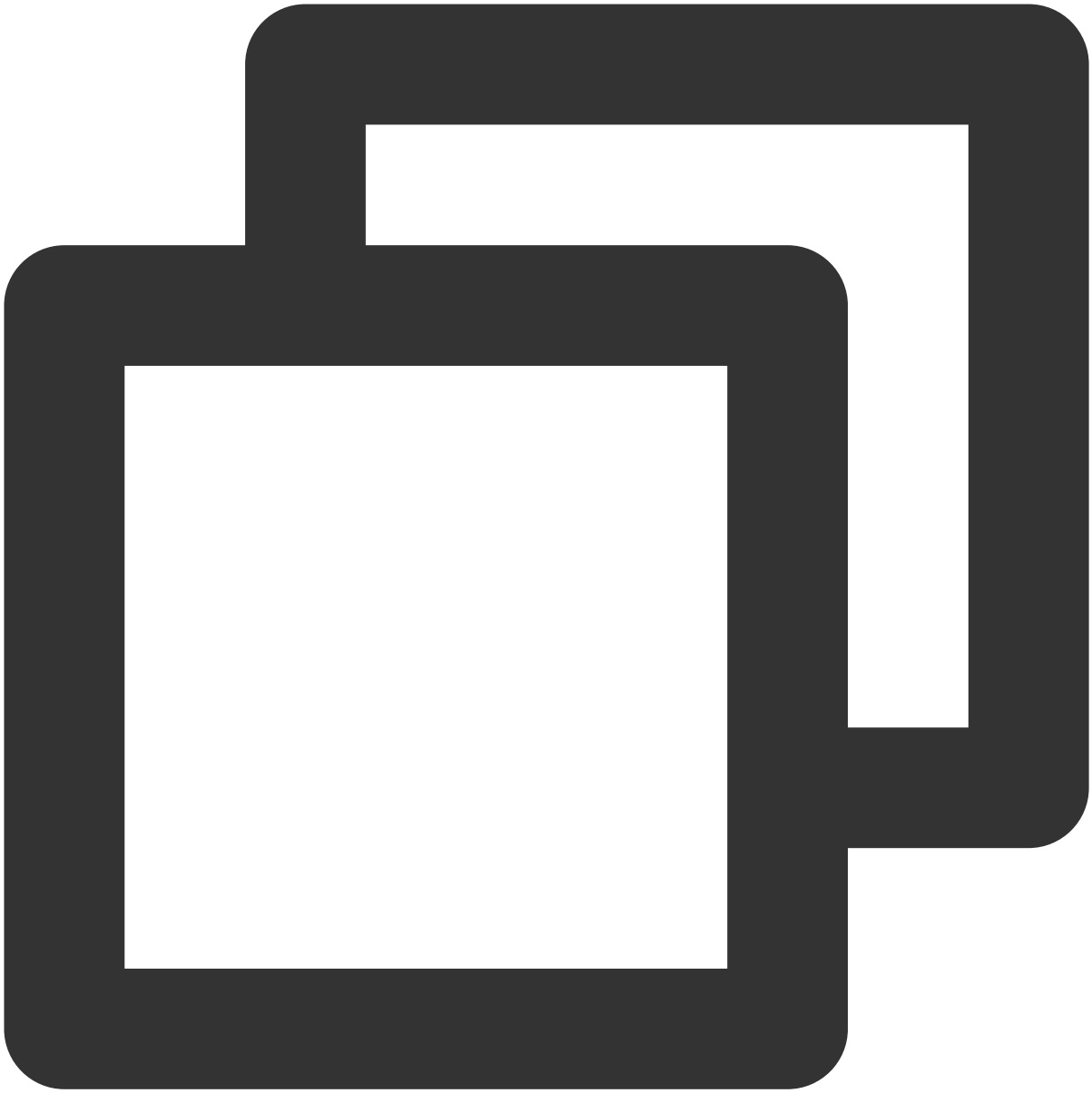
最近更新时间：2024-07-12 17:00:16

WXML 提供两种文件引用方式 `import` 和 `include`。

import

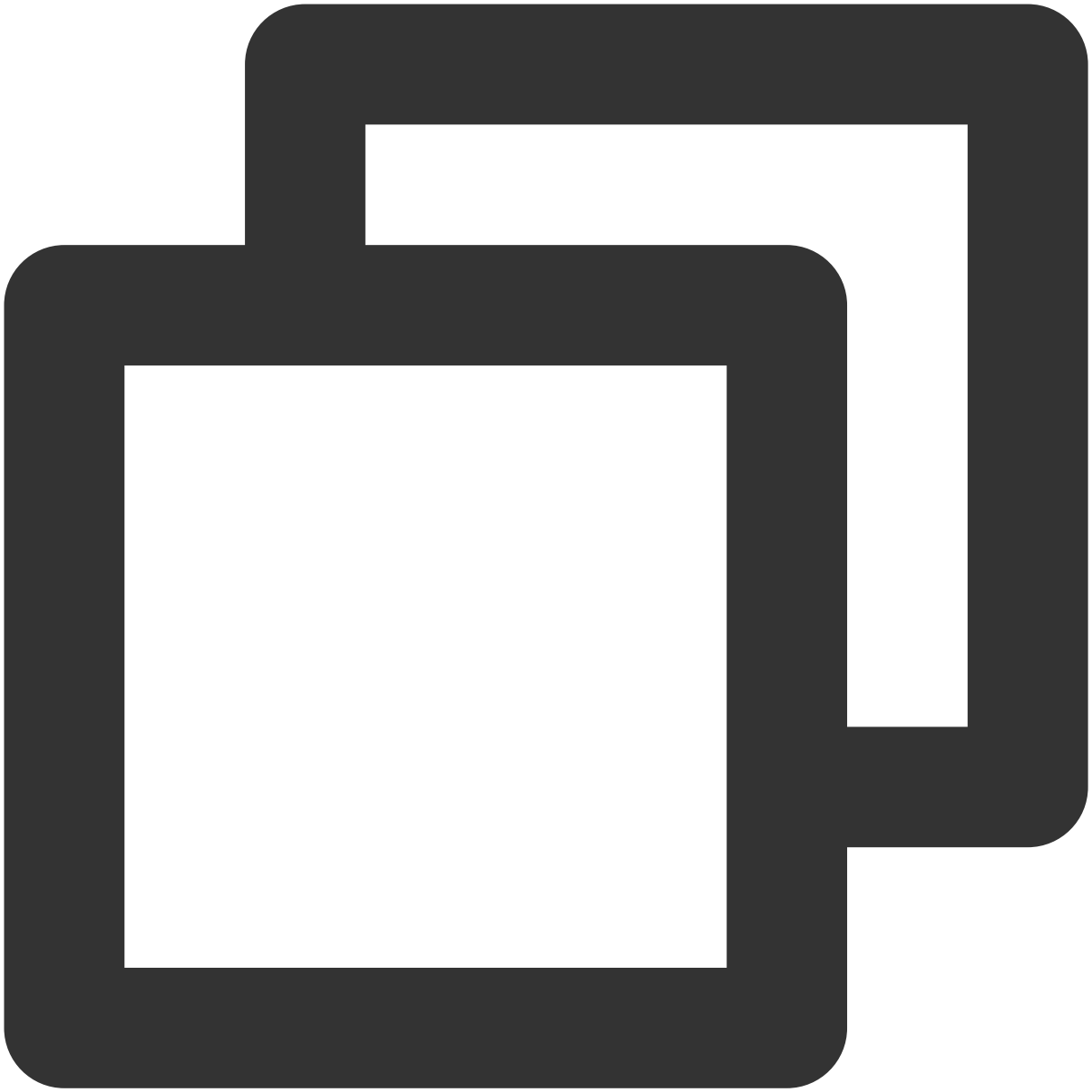
`import` 可以在该文件中使用目标文件定义的 `template`，如：
在 item.wxml 中定义了一个叫 `item` 的 `template` ：

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 index.wxml 中引用了 item.wxml，就可以使用 `item` 模板：

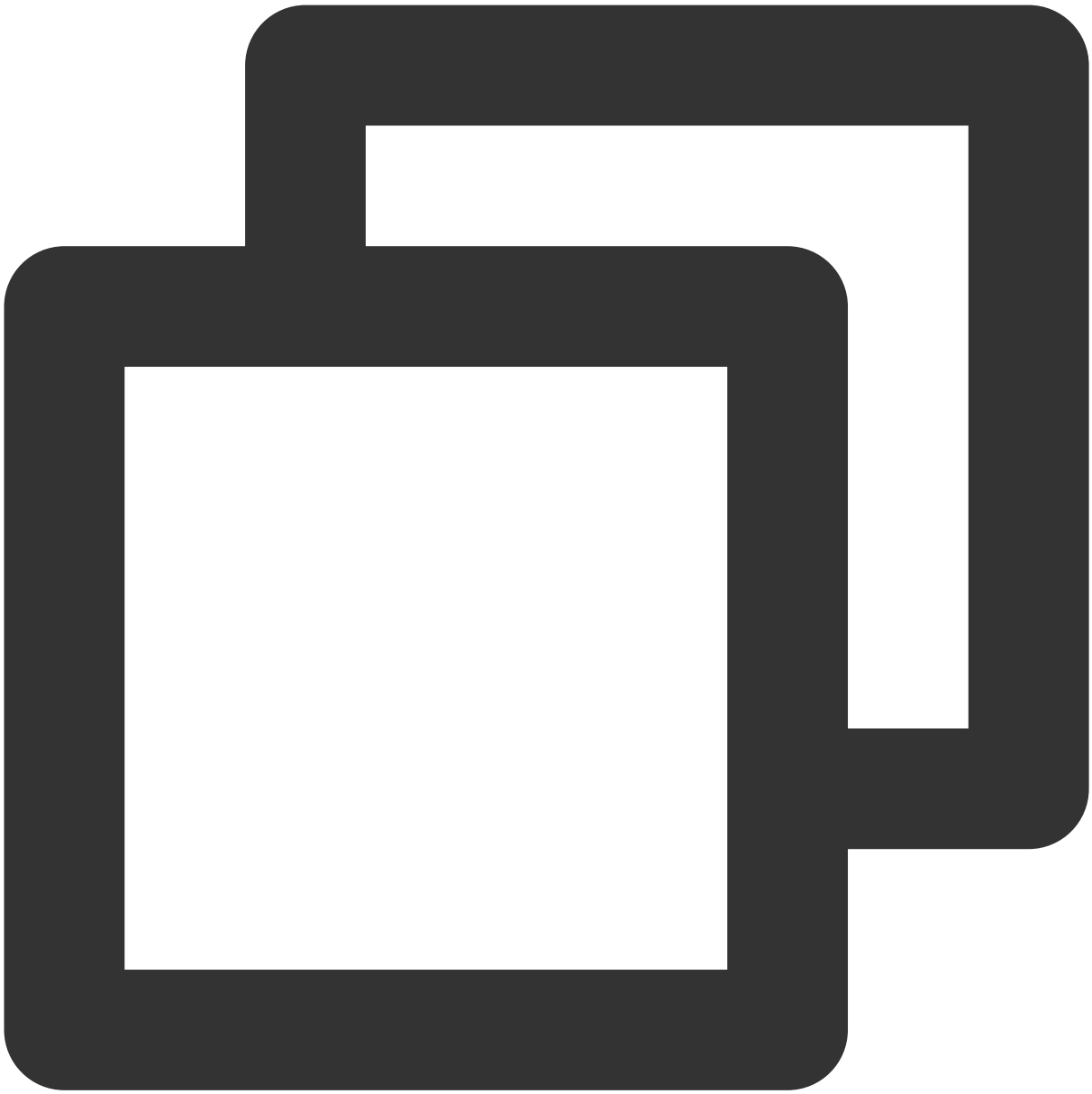


```
<import src="item.wxml"/>
<template is="item" data="{{text: 'forbar'}}"/>
```

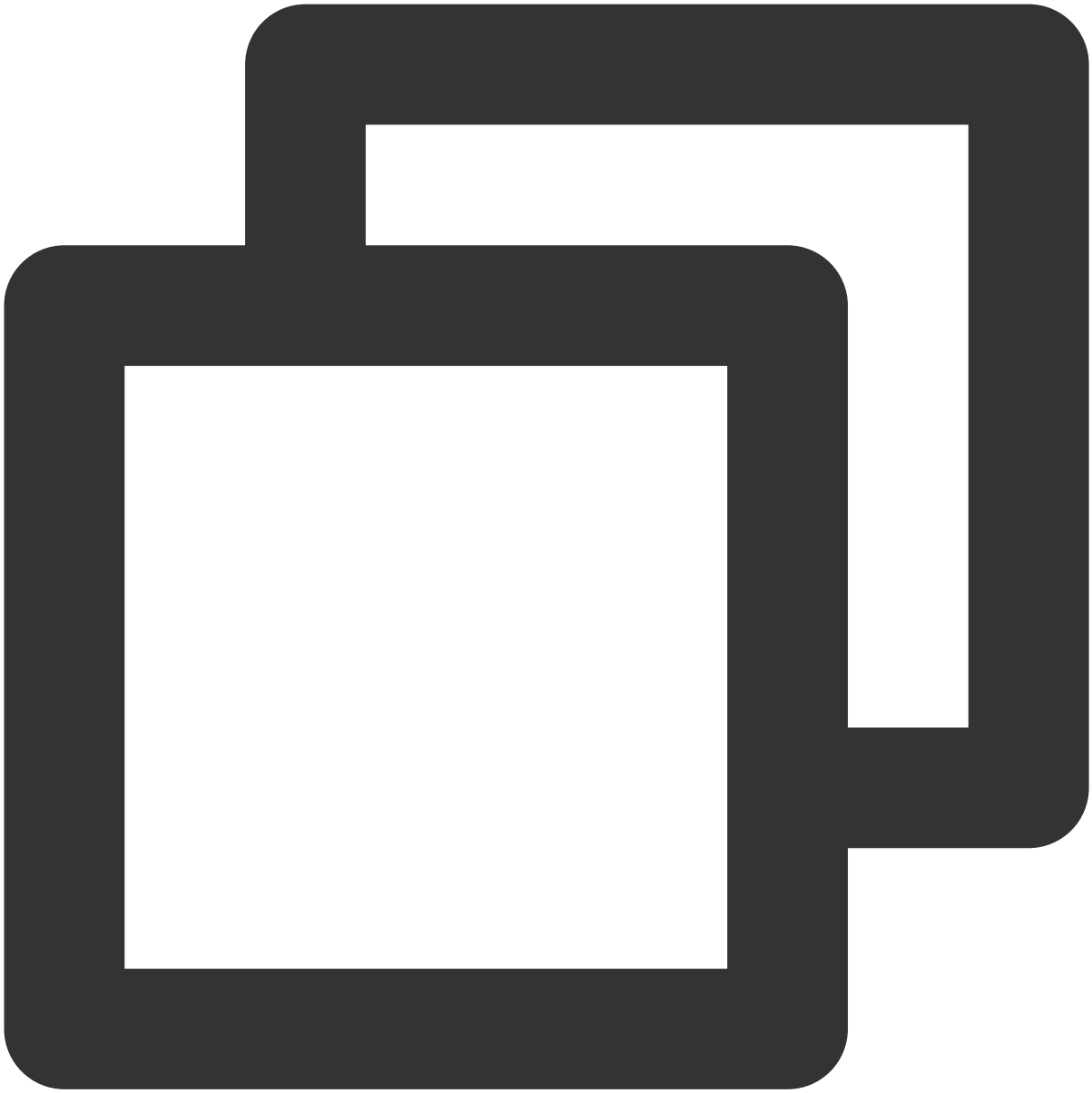
import 的作用域

import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件 import 的 template。

如：C import B，B import A，在C中可以使用B定义的 `template` ，在B中可以使用A定义的 `template` ，但是C不能使用A定义的 `template` 。

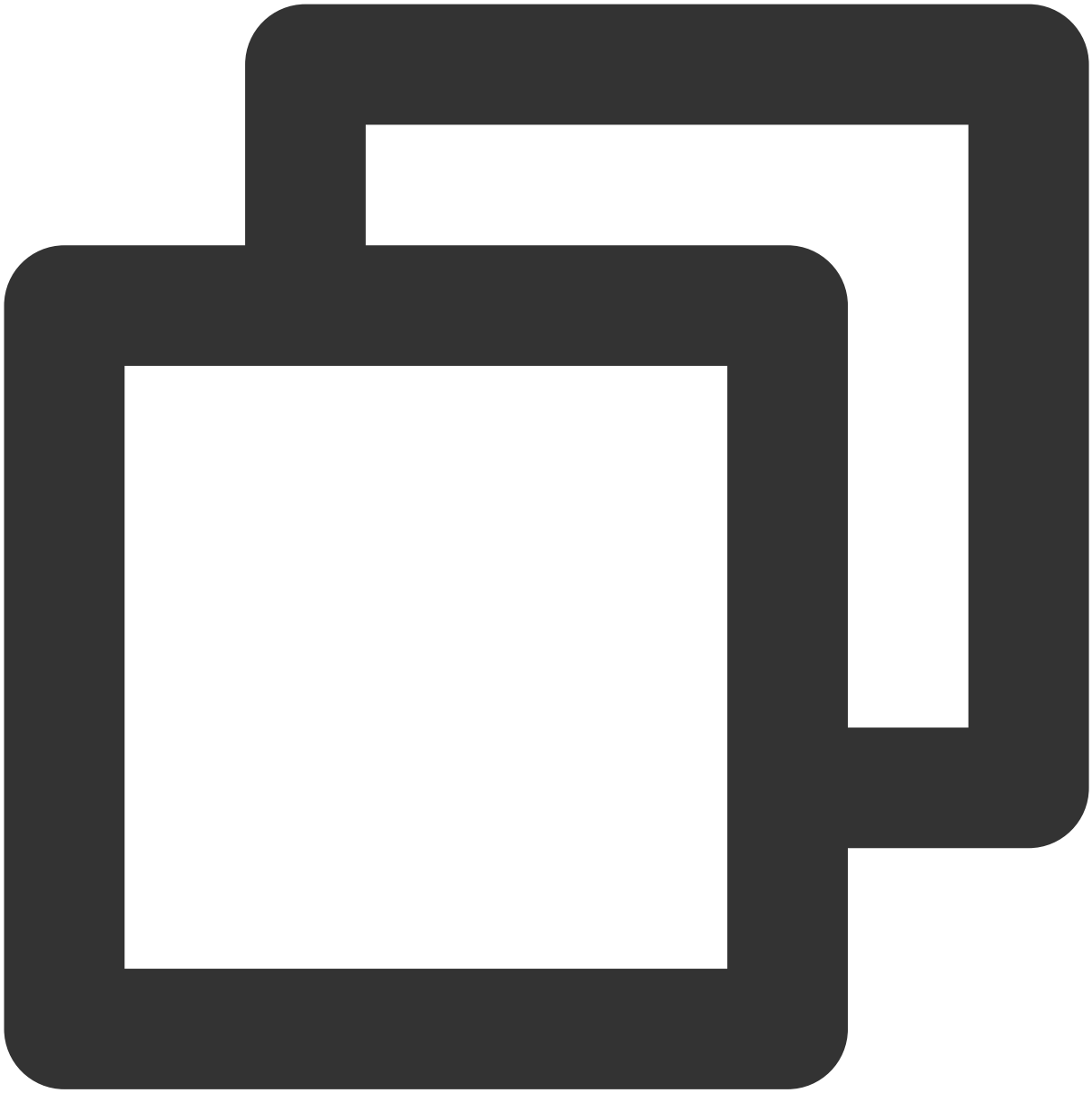


```
<!-- A.wxml -->
<template name="A">
  <text> A template </text>
</template>
```



```
<!-- B.wxml -->
<import src="a.wxml"/>
<template name="B">
  <text> B template </text>
</template>
```

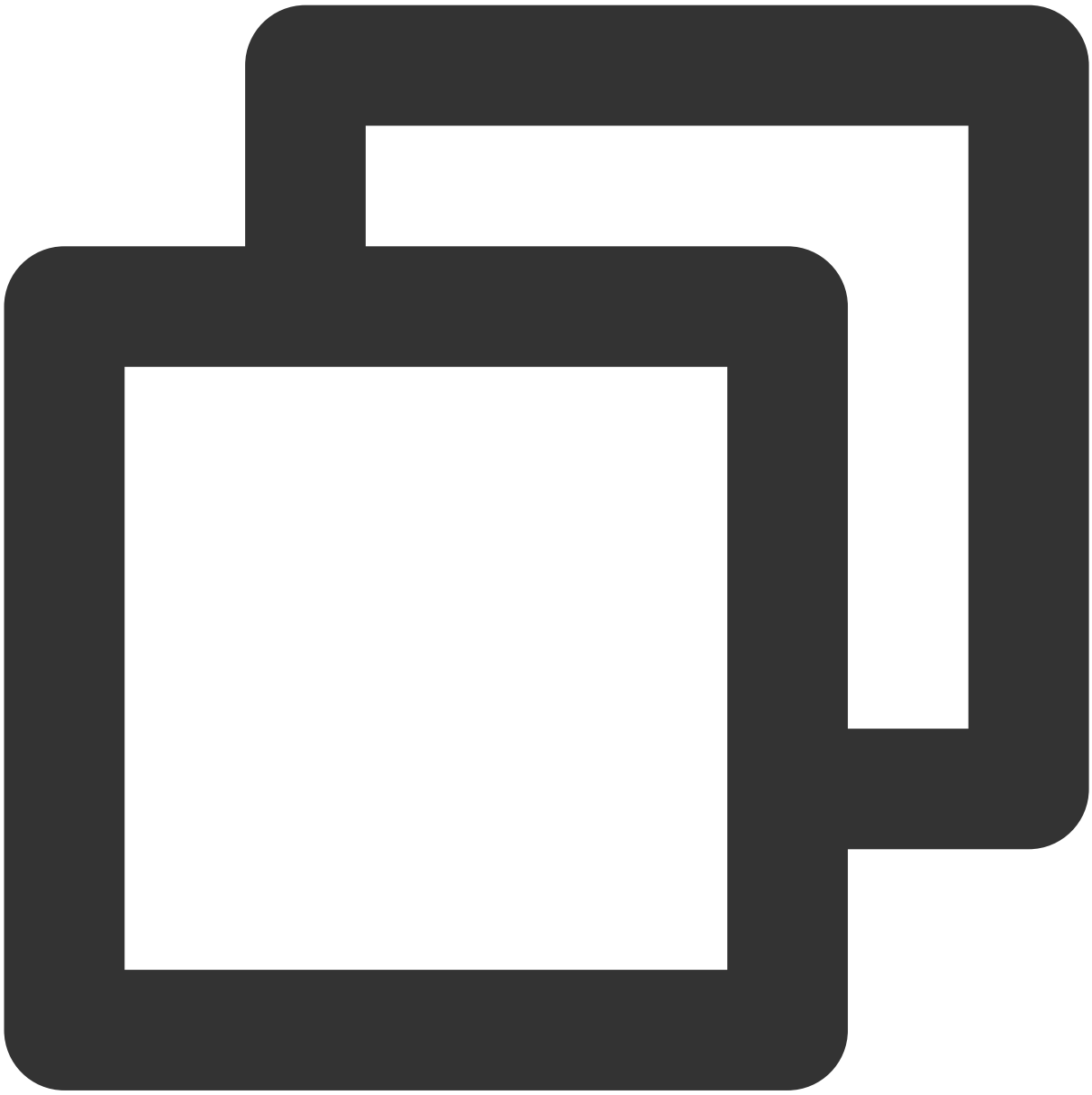
版权所有：腾讯云计算（北京）有限责任公司



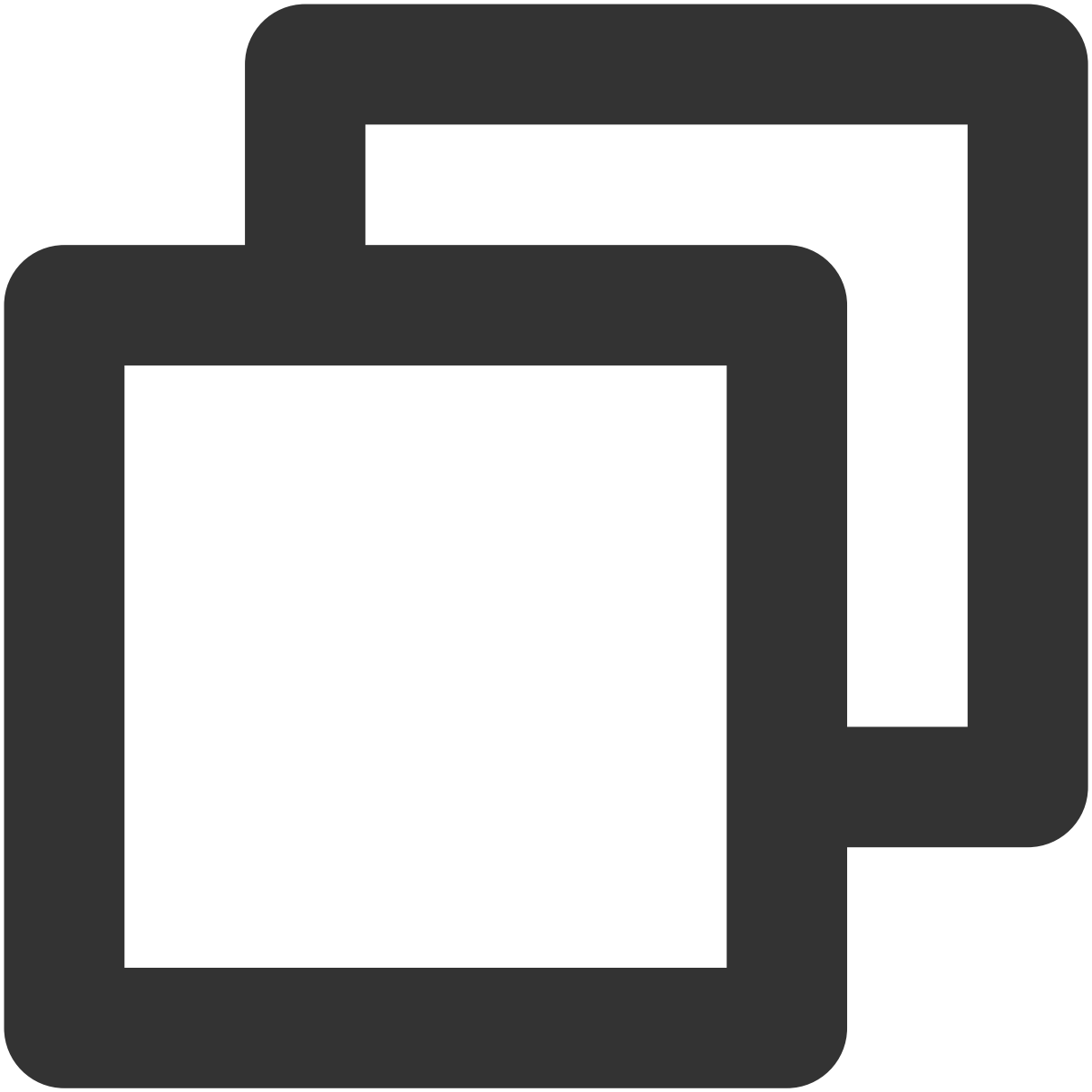
```
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/>  <!-- Error! Can not use tempalte when not import A. -->
<template is="B"/>
```

include

`include` 可以将目标文件**除了** `<template/>` `<wxs/>` 外的整个代码引入，相当于是拷贝到 `include` 位置，如：

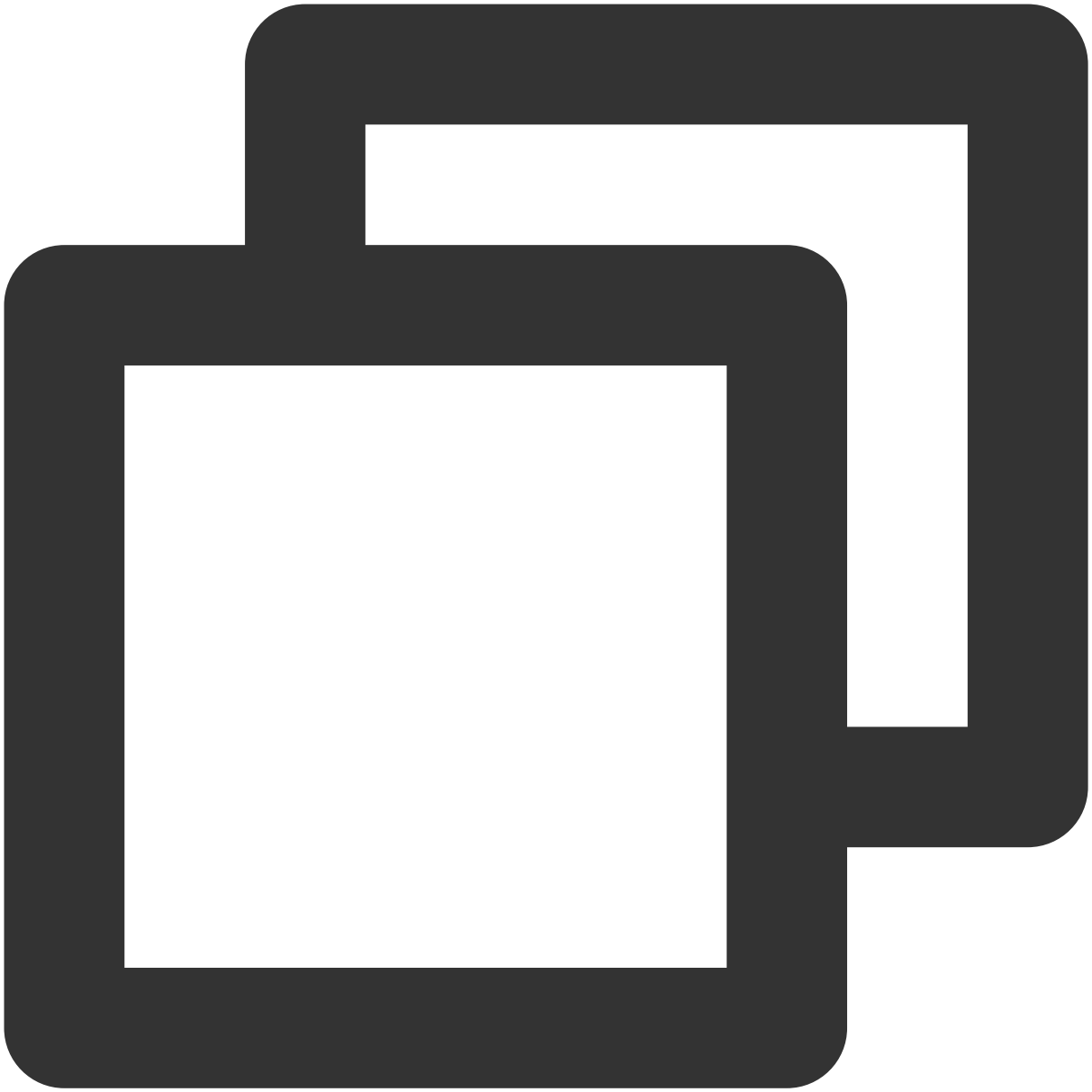


```
<!-- index.wxml -->
<include src="header.wxml"/>
<view> body </view>
<include src="footer.wxml"/>
```



```
<!-- header.wxml -->
<view> header </view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
<!-- footer.wxml -->
<view> footer </view>
```

版权所有：腾讯云计算（北京）有限责任公司

WXS 语法参考

介绍

最近更新时间：2024-07-12 16:59:16

WXS 是小程序的一套脚本语言，结合 `WXML` ，可以构建出页面的结构。

WXS 与 JavaScript 是不同的语言，有自己的语法，并不和 JavaScript 一致。

WXS 模块

变量

注释

运算符

语句

数据类型

基础类库

版权所有：腾讯云计算（北京）有限责任公司

WXS 模块

最近更新时间：2024-07-12 16:58:18

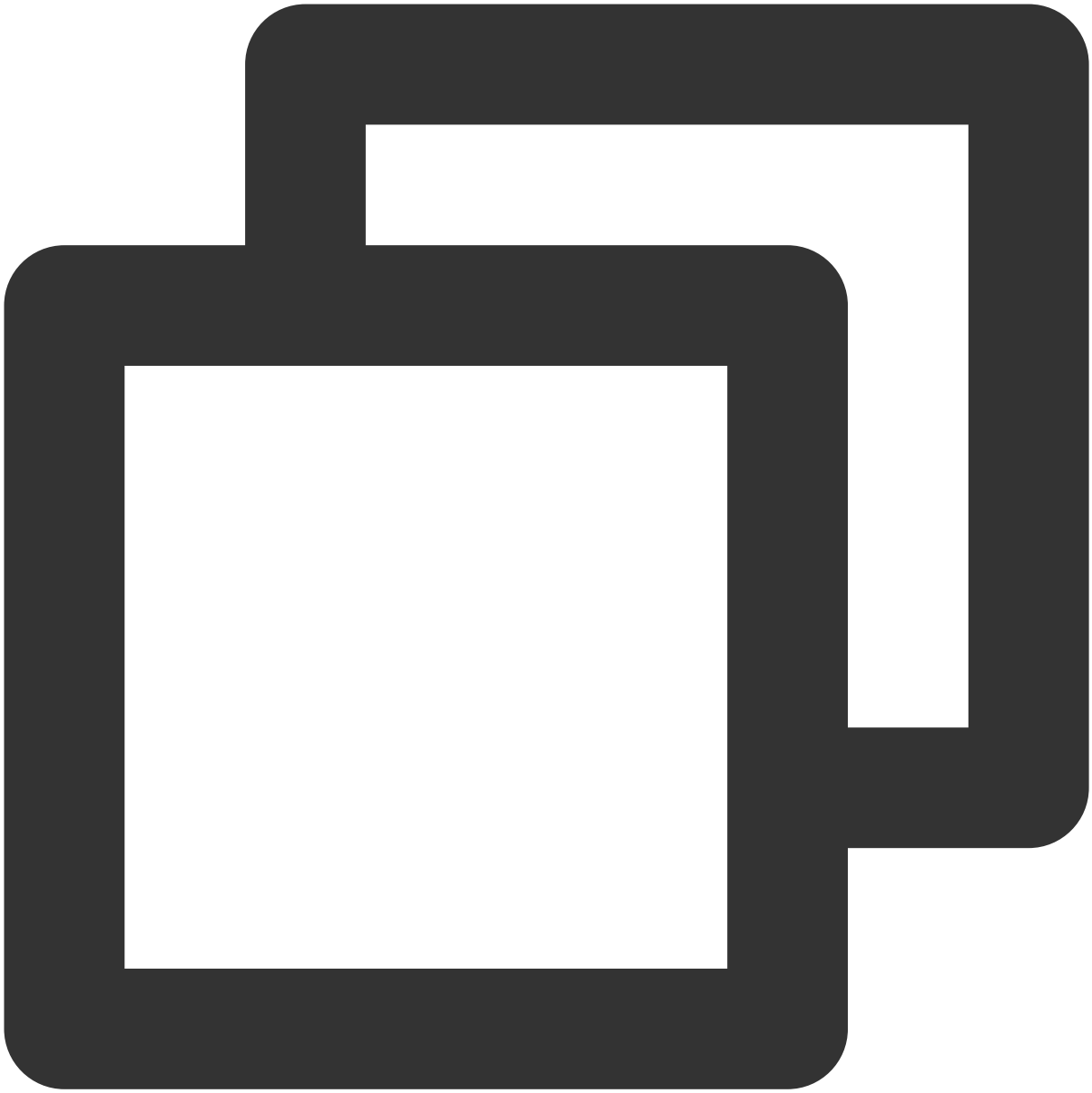
WXS 代码可以编写在 wxml 文件中的 `<wxs>` 标签内，或以 `.wxs` 为后缀名的文件内。

模块

每一个 `.wxs` 文件和 `<wxs>` 标签都是一个单独的模块。
每个模块都有自己独立的作用域。即在一个模块里面定义的变量与函数，默认为私有的，对其他模块不可见。
一个模块要想对外暴露其内部的私有变量与函数，只能通过 `module.exports` 实现。

.wxs 文件

在开发者工具里面，右键可以直接创建 `.wxs` 文件，在其中直接编写 WXS 脚本。
示例代码：



```
// /pages/comm.wxs

var foo = "'hello world' from comm.wxs";
var bar = function(d) {
  return d;
}
module.exports = {
  foo: foo,
  bar: bar
};
```

版权所有：腾讯云计算（北京）有限责任公司

上述例子在 `/pages/comm.wxs` 的文件里面编写了 WXS 代码。该 `.wxs` 文件可以被其他的 `.wxs` 文件 或 WXML 中的 `<wxs>` 标签引用。

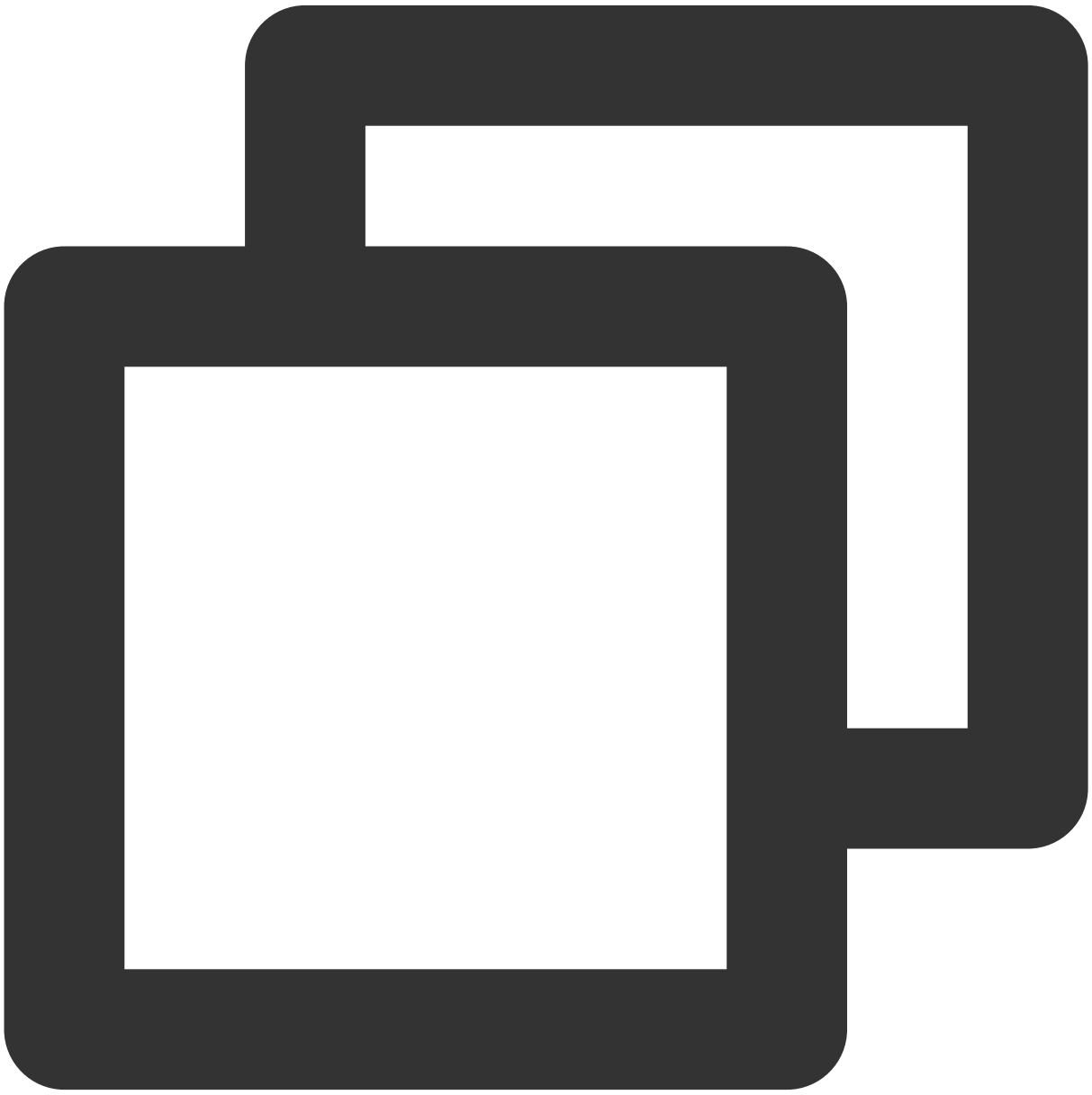
module 对象

每个 `wxs` 模块均有一个内置的 `module` 对象。

属性

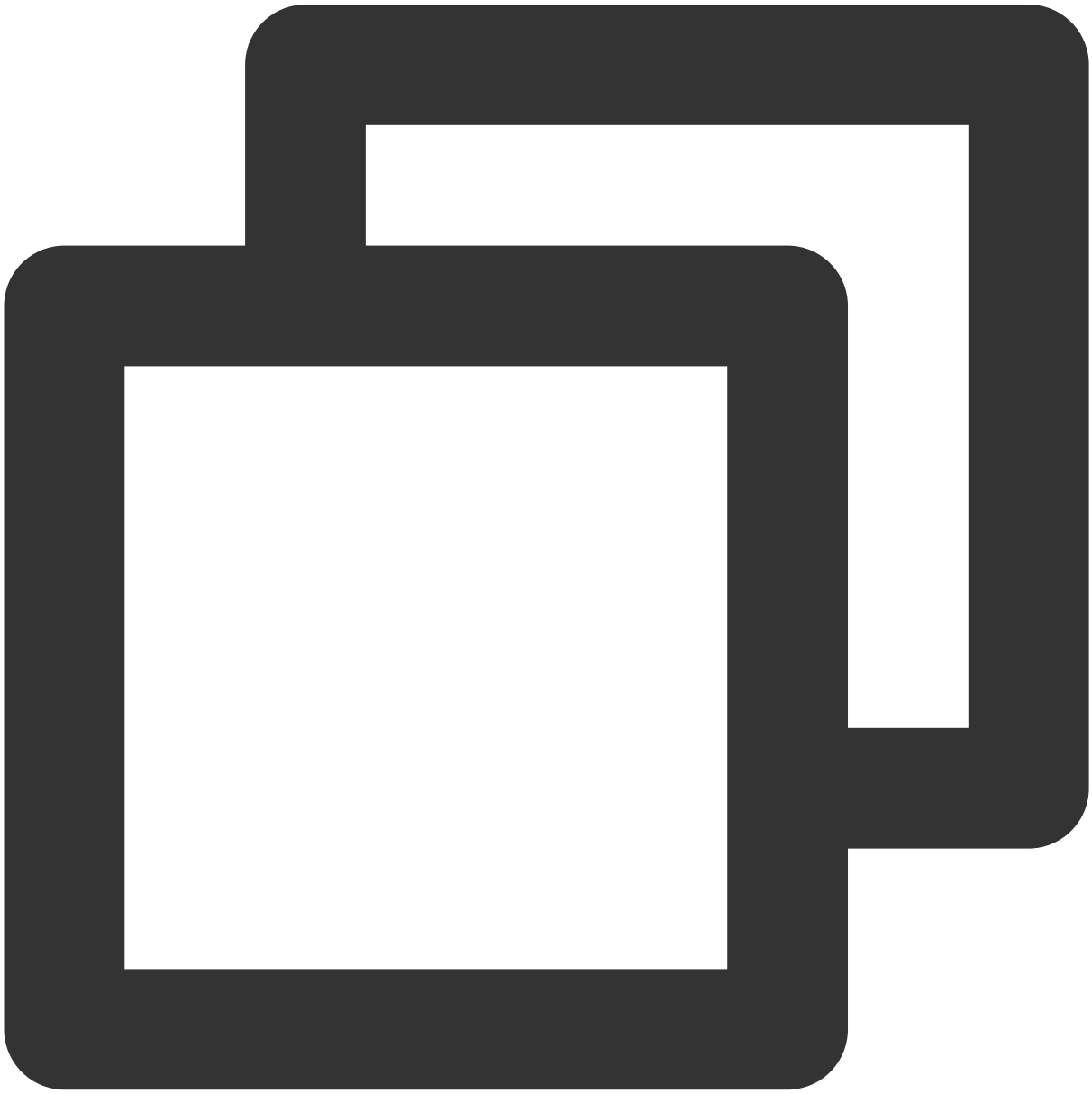
`exports`：通过该属性，可以对外共享本模块的私有变量与函数。

示例代码：



```
// /pages/tools.wxs

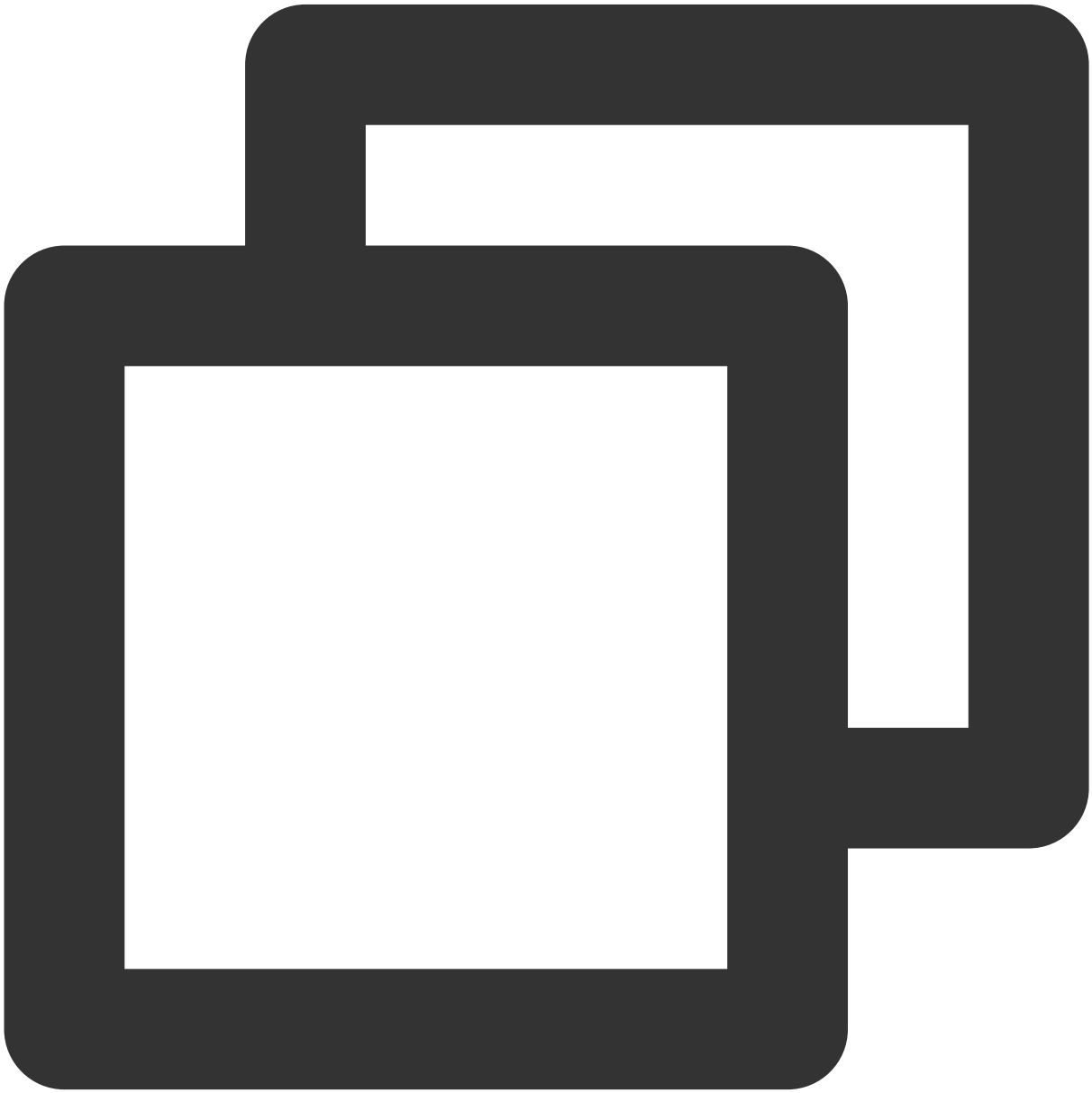
var foo = "'hello world' from tools.wxs";
var bar = function (d) {
  return d;
}
module.exports = {
  FOO: foo,
  bar: bar,
};
module.exports.msg = "some msg";
```



```
<!-- page/index/index.wxml -->

<wxs src="./../tools.wxs" module="tools" />
<view> {{tools.msg}} </view>
<view> {{tools.bar(tools.FOO)}} </view>
```

页面输出：



```
some msg
'hello world' from tools.wxs
```

require 函数

在 `.wxs` 模块中引用其他 `wxs` 文件模块，可以使用 `require` 函数。

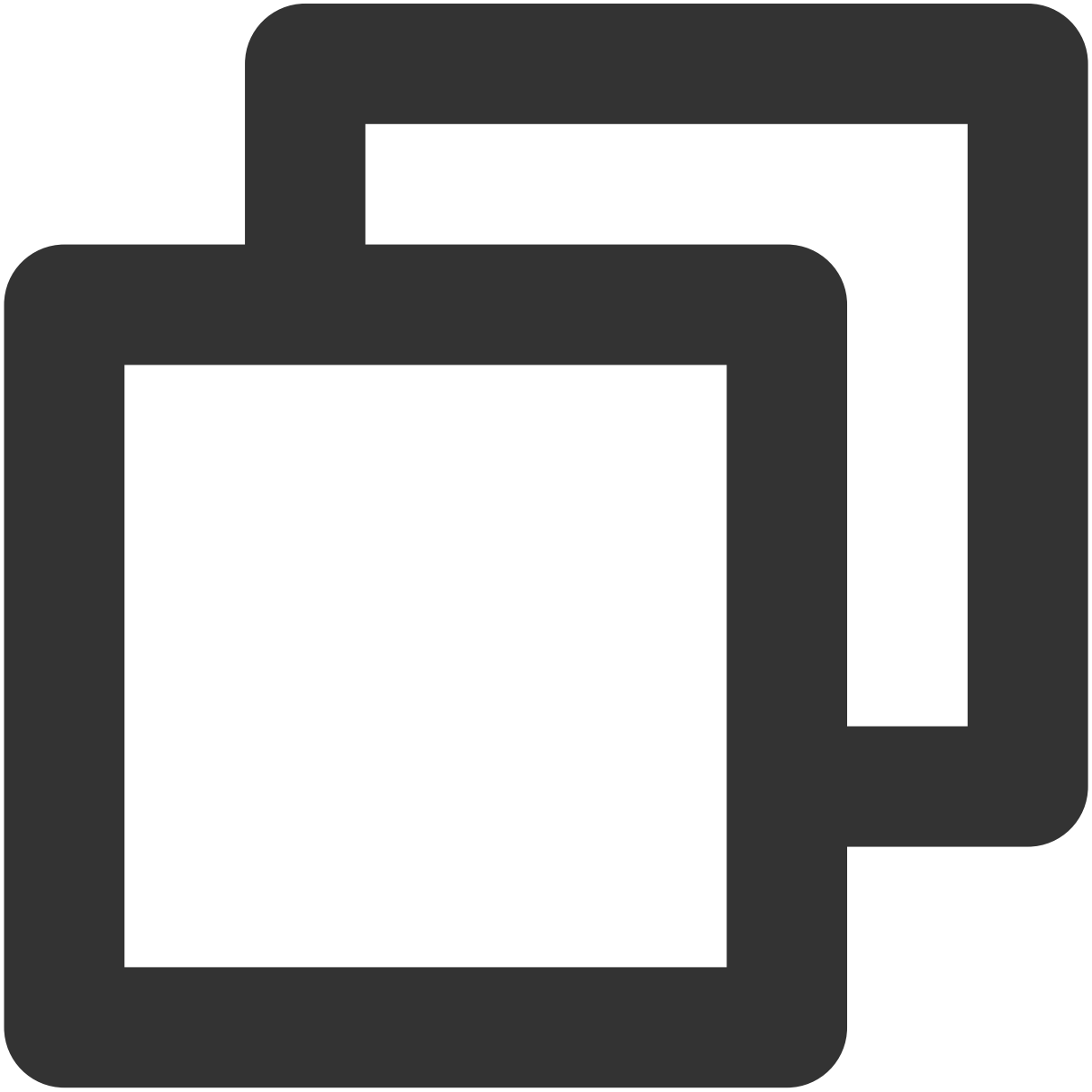
引用的时候，要注意如下几点：

只能引用 `.wxs` 文件模块，且必须使用相对路径。

`wxs` 模块均为单例， `wxs` 模块在第一次被引用时，会自动初始化为单例对象。多个页面，多个地方，多次引用，使用的都是同一个 `wxs` 模块对象。

如果一个 `wxs` 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

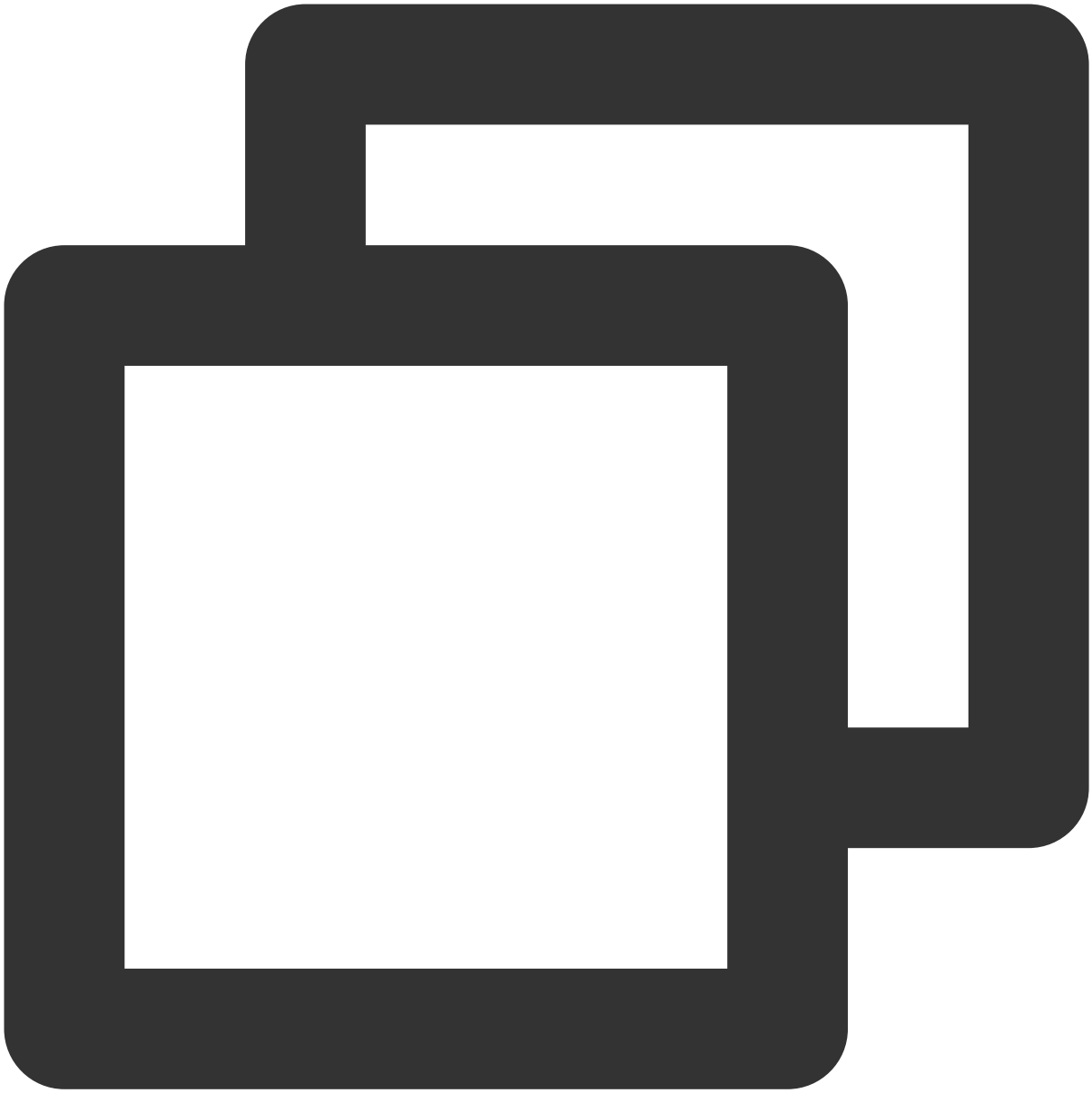
示例代码：



```
// /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";
```

版权所有：腾讯云计算（北京）有限责任公司

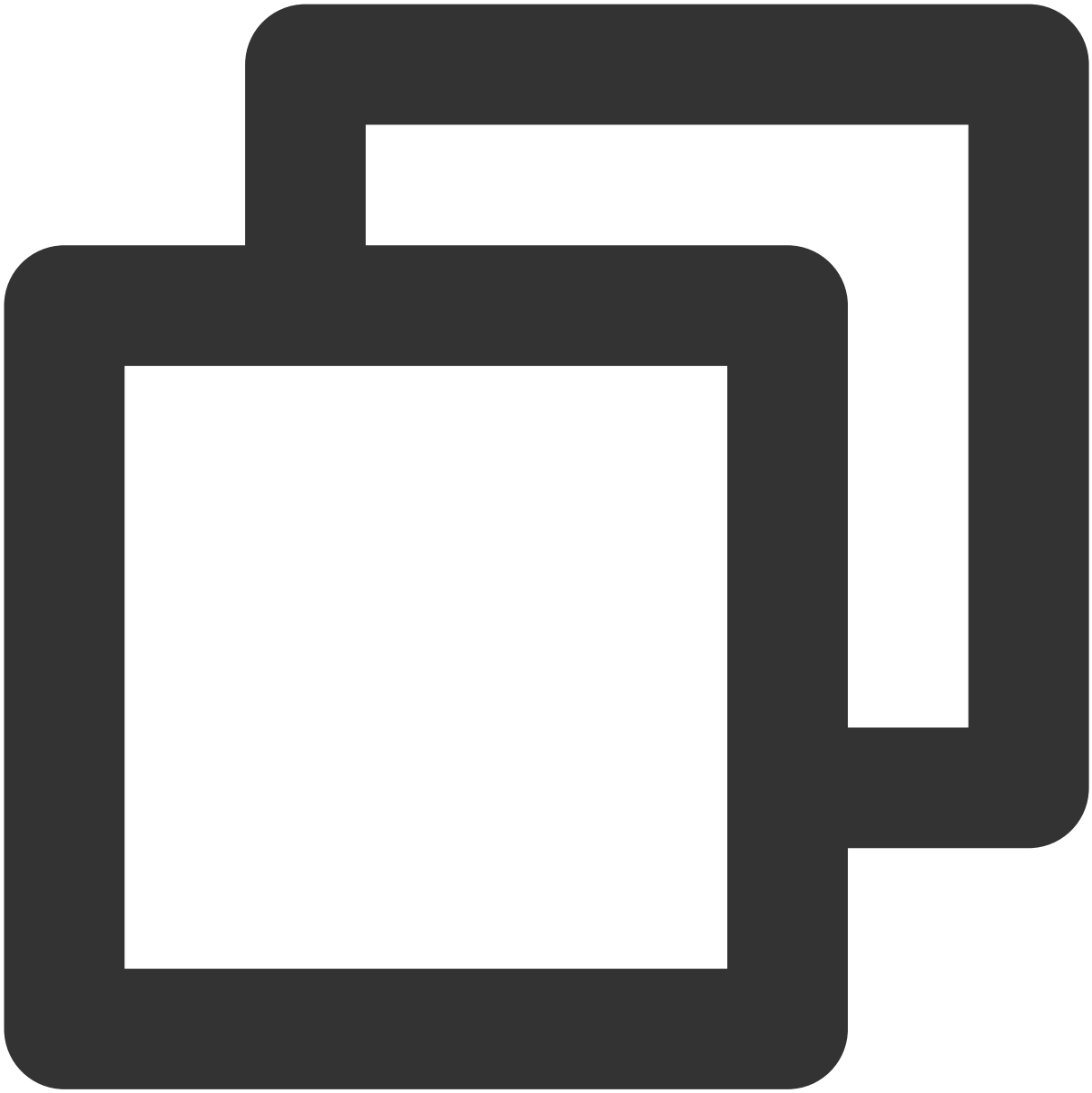


```
// /pages/logic.wxs

var tools = require("./tools.wxs");

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

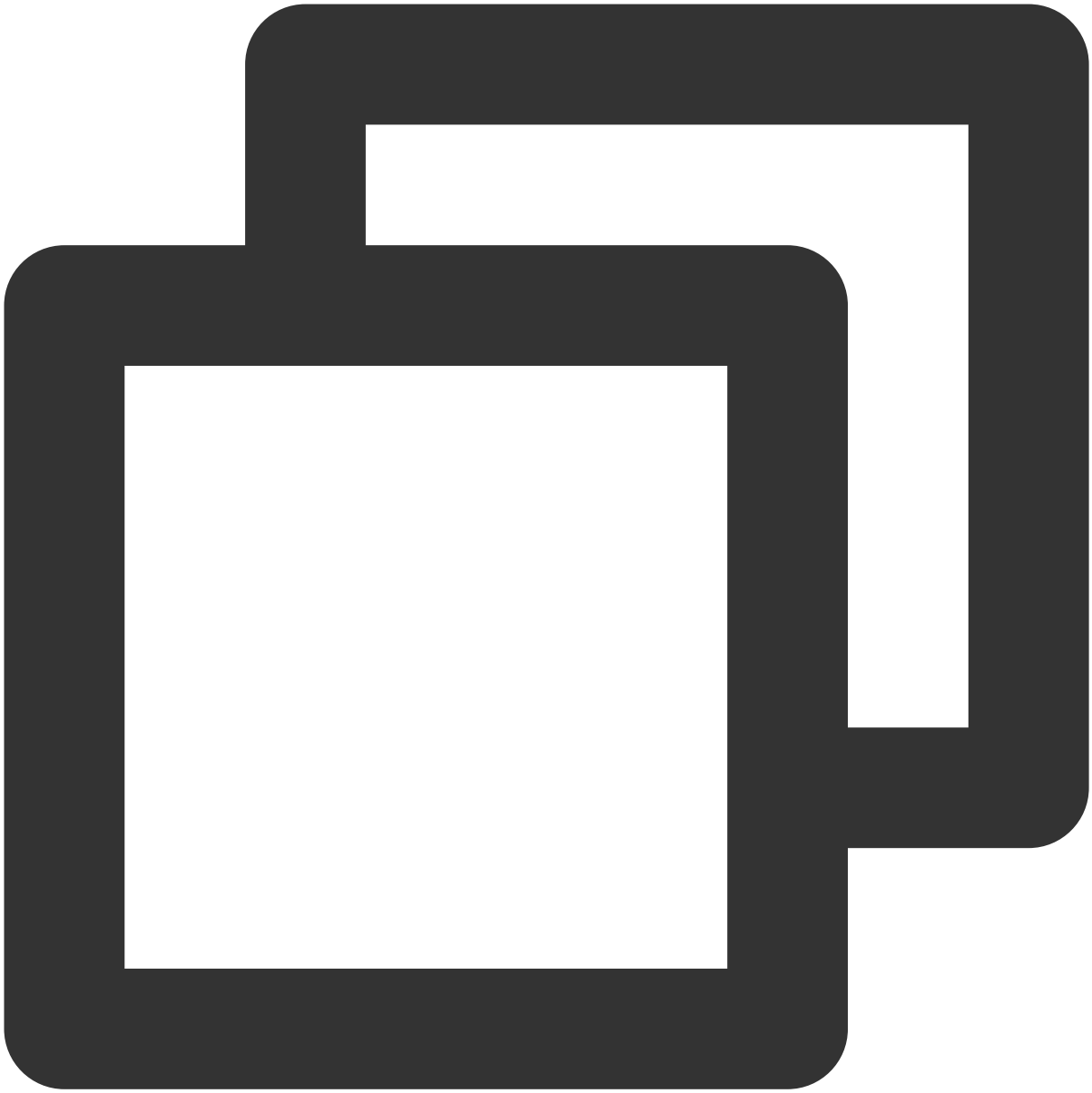
版权所有：腾讯云计算（北京）有限责任公司



```
<!-- /page/index/index.wxml -->

<wxs src="./../logic.wxs" module="logic" />
```

控制台输出：



```
'hello world' from tools.wxs
logic.wxs
some msg
```

<wxs> 标签

属性名	类型	默认值	说明
module	String	-	当前 `<wxs>` 标签的模块名。必填字段。

src	String	-	引用 .wxs 文件的相对路径。仅当本标签为**单闭合标签**或**标签的内容**为空时有效。

module 属性

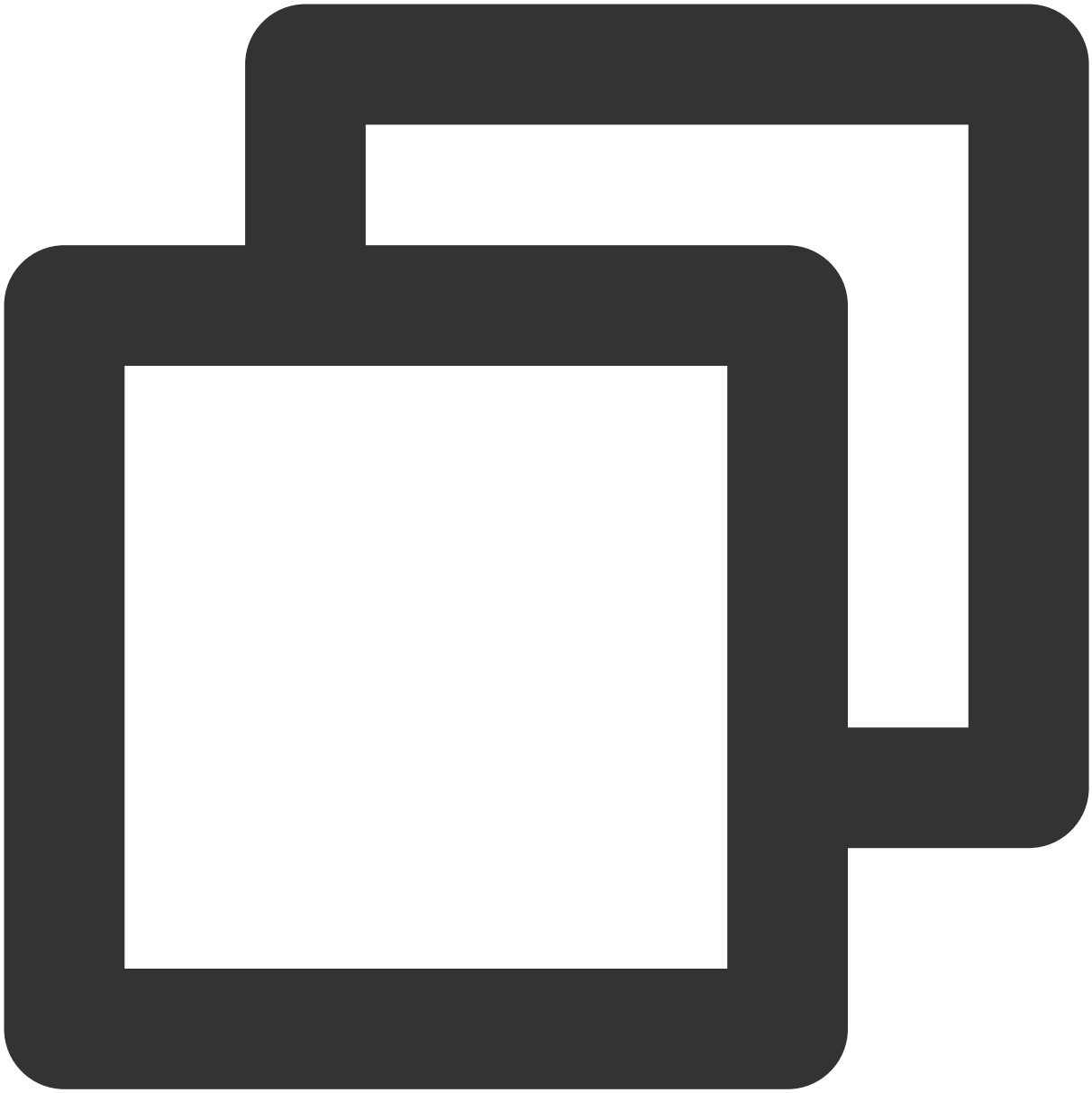
module 属性是当前 `<wxs>` 标签的模块名。在单个 wxml 文件内，建议其值唯一。有重复模块名则按照先后顺序覆盖（后者覆盖前者）。不同文件之间的 wxs 模块名不会相互覆盖。

module 属性值的命名必须符合下面两个规则：

首字符必须是：字母（a-zA-Z），下划线（_）；

剩余字符可以是：字母（a-zA-Z），下划线（_），数字（0-9）。

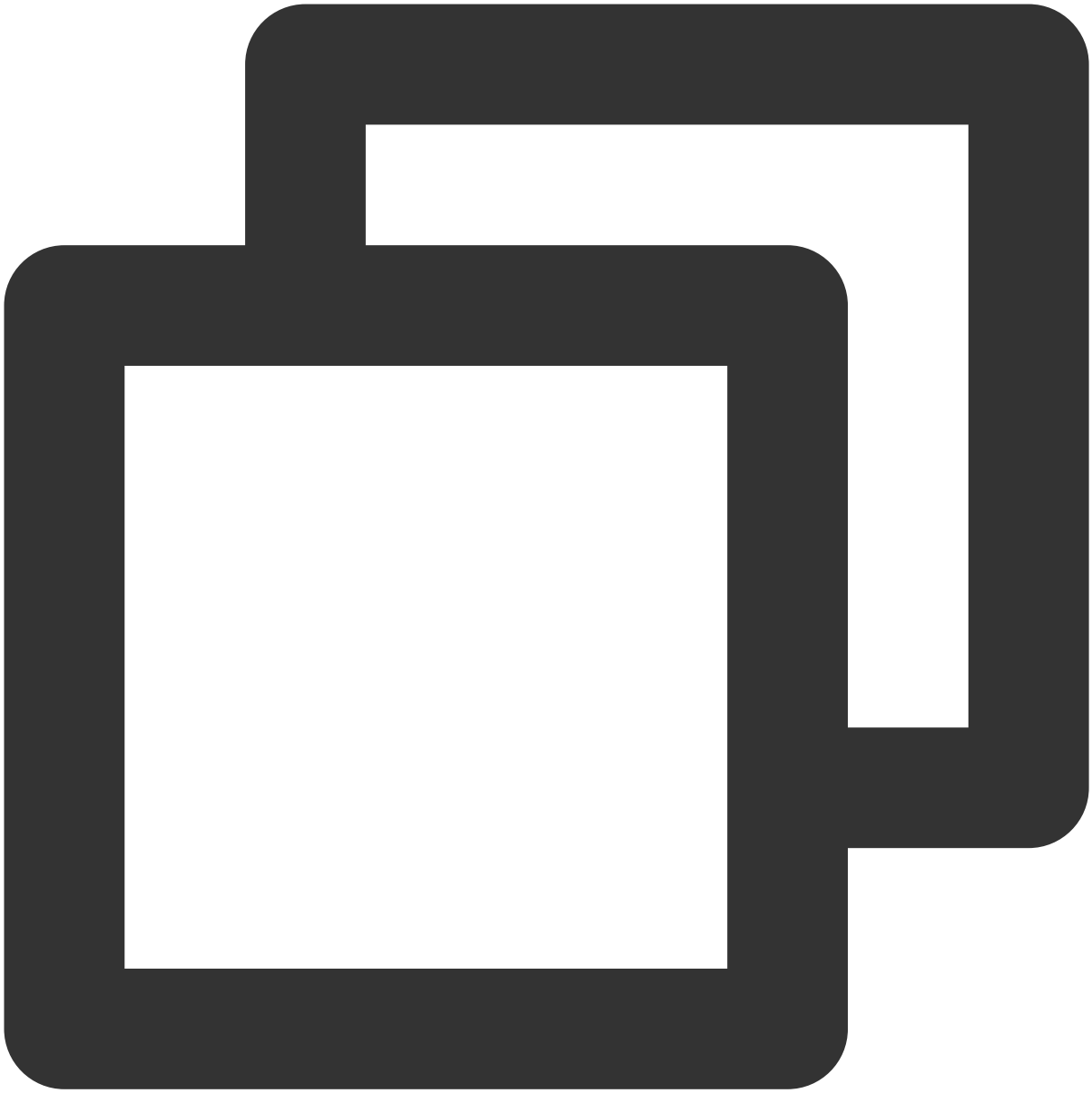
示例代码：



```
<!--wxml-->

<wxs module="foo">
var some_msg = "hello world";
module.exports = {
  msg : some_msg,
}
</wxs>
<view> {{foo.msg}} </view>
```

页面输出：



```
hello world
```

上面例子声明了一个名字为 `foo` 的模块，将 `some_msg` 变量暴露出来，供当前页面使用。

src 属性

src 属性可以用来引用其他的 `wxs` 文件模块。

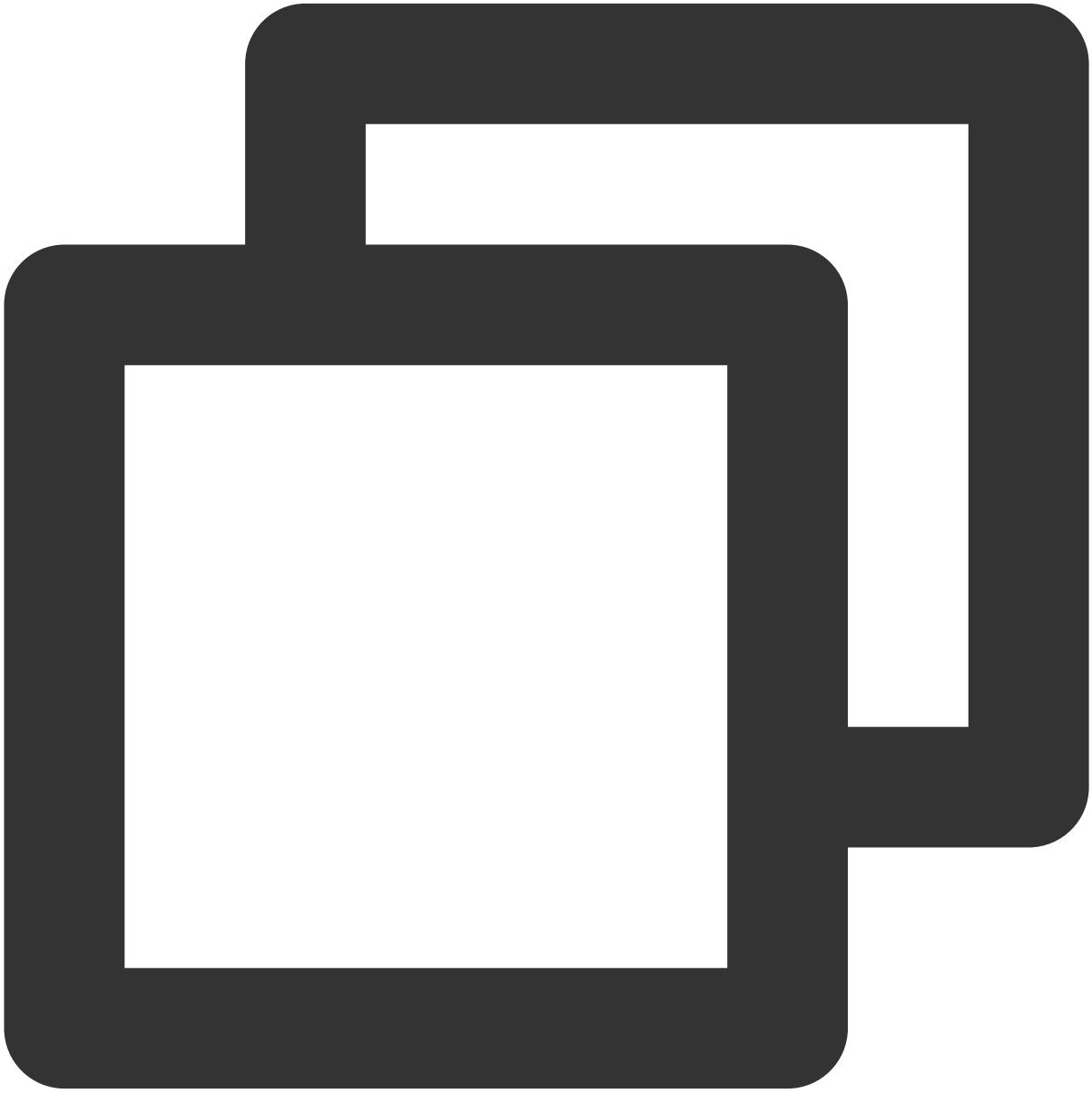
引用的时候，要注意如下几点：

只能引用 `.wxs` 文件模块，且必须使用相对路径；

版权所有：腾讯云计算（北京）有限责任公司

`wxs` 模块均为单例， `wxs` 模块在第一次被引用时，会自动初始化为单例对象。多个页面，多个地方，多次引用，使用的都是同一个 `wxs` 模块对象；
如果一个 `wxs` 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

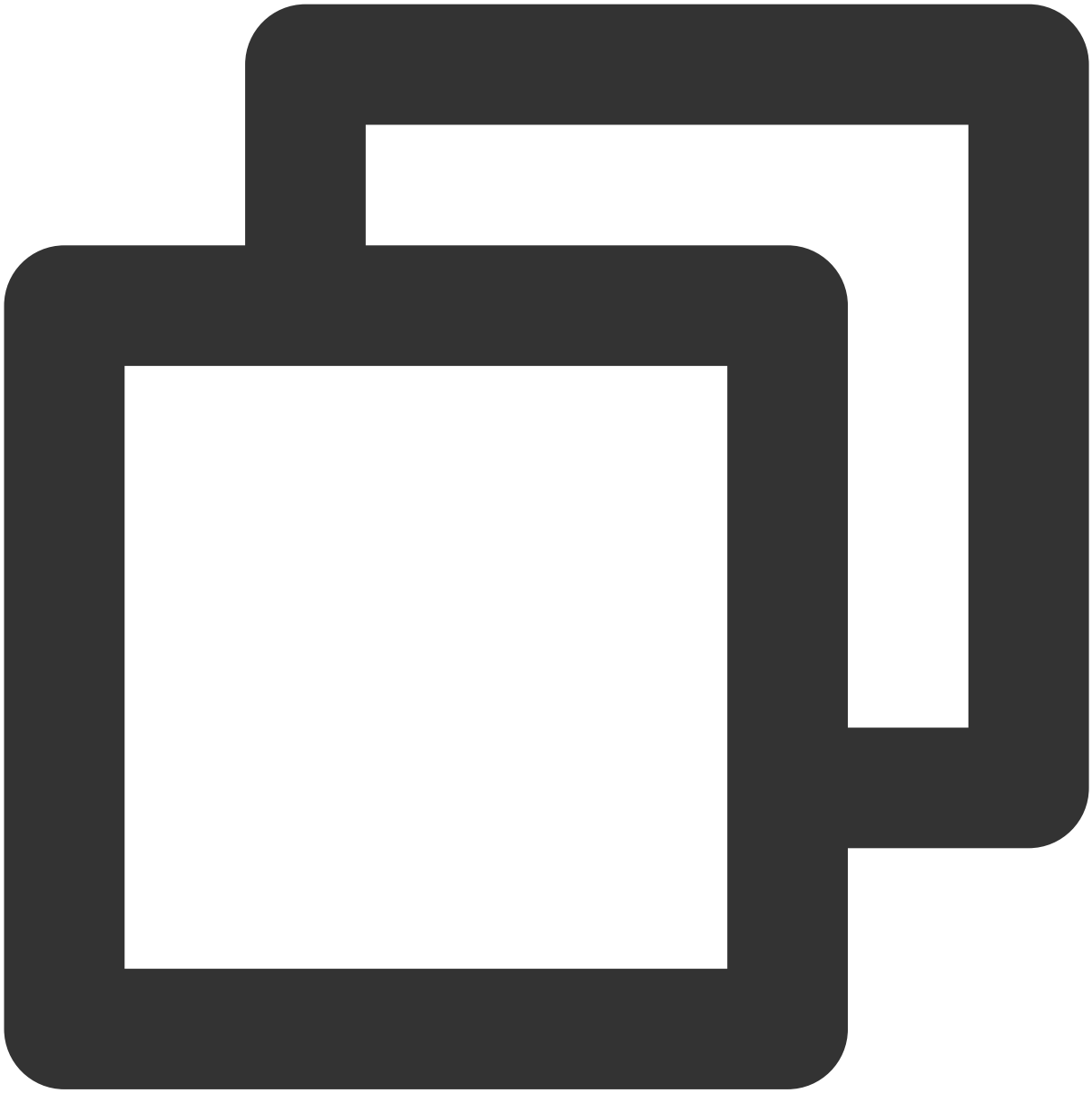
示例代码：



```
// /pages/index/index.js

Page({
  data: {
    msg: "'hello wrold' from js",
```

```
  }
})
```



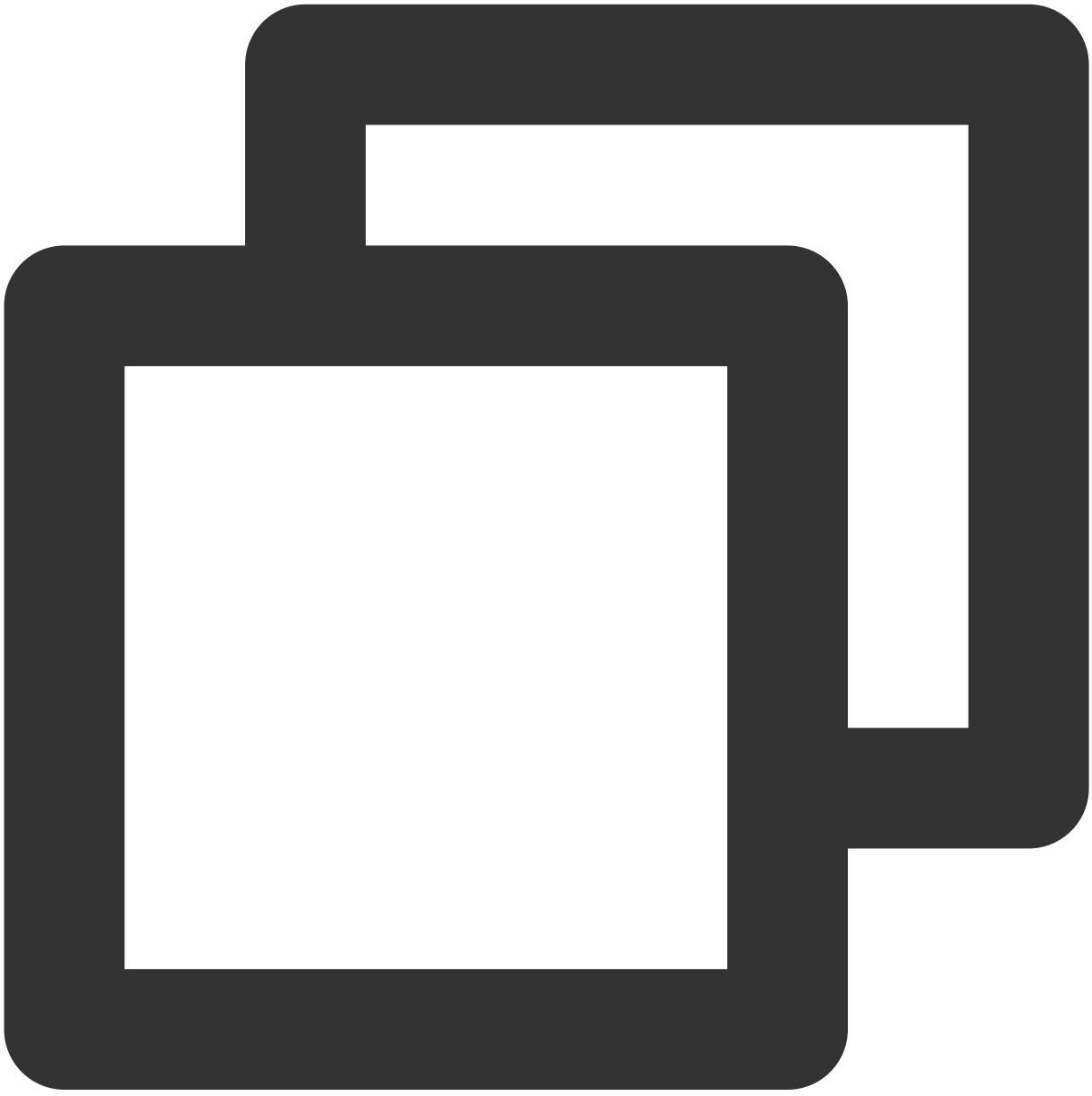
```
<!-- /pages/index/index.wxml -->

<wxs src="./../comm.wxs" module="some_comms"></wxs>
<!-- 也可以直接使用单标签闭合的写法
<wxs src="./../comm.wxs" module="some_comms" />
-->

<!-- 调用 some_comms 模块里面的 bar 函数，且参数为 some_comms 模块里面的 foo -->
```

```
<view> {{some_comms.bar(some_comms.foo)}} </view>
<!-- 调用 some_comms 模块里面的 bar 函数，且参数为 page/index/index.js 里面的 msg -->
<view> {{some_comms.bar(msg)}} </view>
```

页面输出：



```
'hello world' from comm.wxs
'hello wrold' from js
```

上述例子在文件 `/page/index/index.wxml` 中通过 `<wxs>` 标签引用了 `/page/comm.wxs` 模块。

版权所有：腾讯云计算（北京）有限责任公司

注意事项

`<wxs>` 模块只能在定义模块的 WXML 文件中被访问到。使用 `<include>` 或 `<import>` 时，`<wxs>` 模块不会被引入到对应的 WXML 文件中；

`<template>` 标签中，只能使用定义该 `<template>` 的 WXML 文件中定义的 `<wxs>` 模块。

版权所有：腾讯云计算（北京）有限责任公司

变量

最近更新时间：2024-07-12 19:05:37

概念

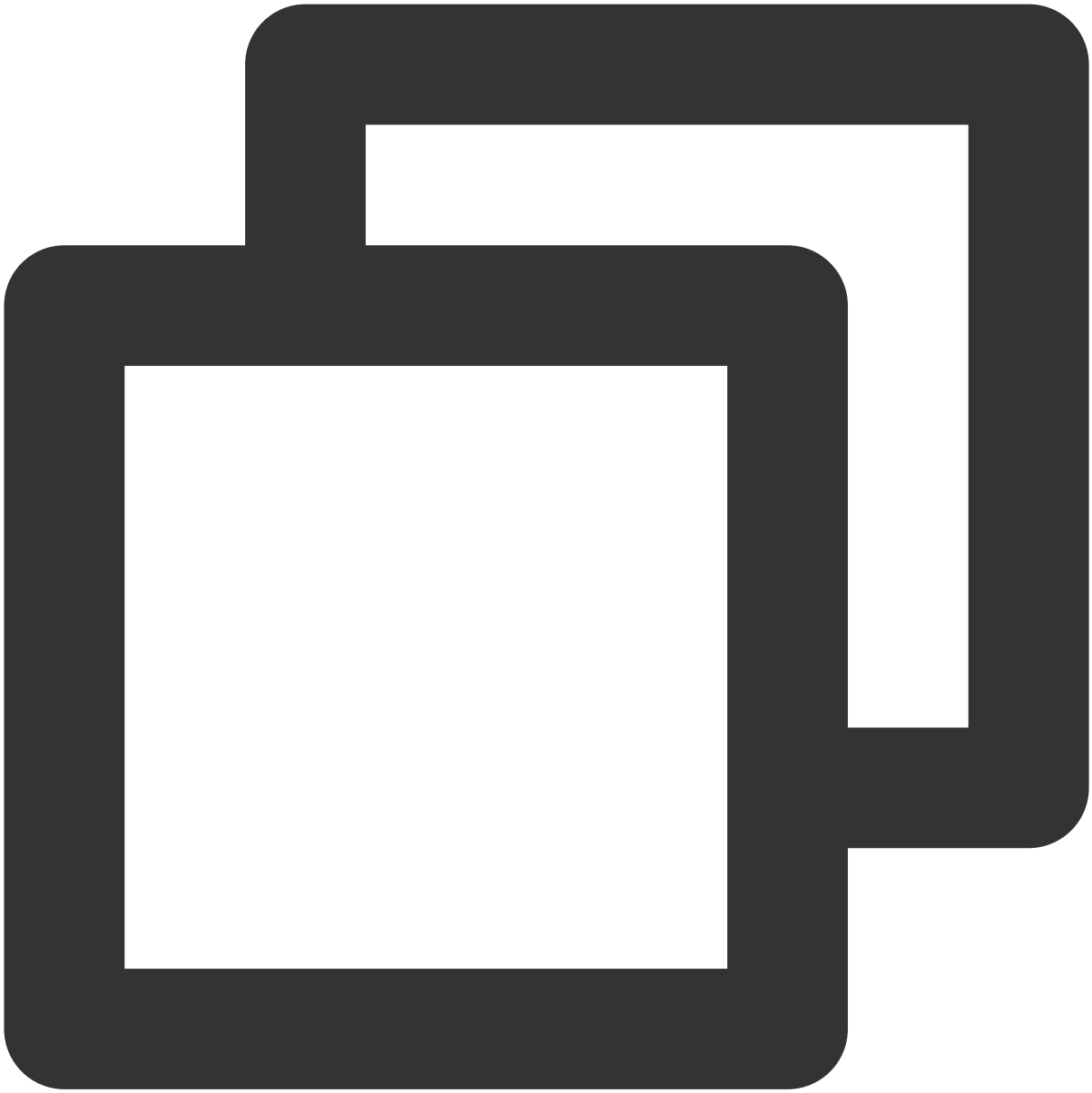
WXS 中的变量均为值的引用。

没有声明的变量直接赋值使用，会被定义为全局变量。

如果只声明变量而不赋值，则默认值为 `undefined` 。

var 表现与 javascript 一致，会有变量提升。

版权所有：腾讯云计算（北京）有限责任公司



```
var foo = 1;
var bar = "hello world";
var i; // i === undefined
```

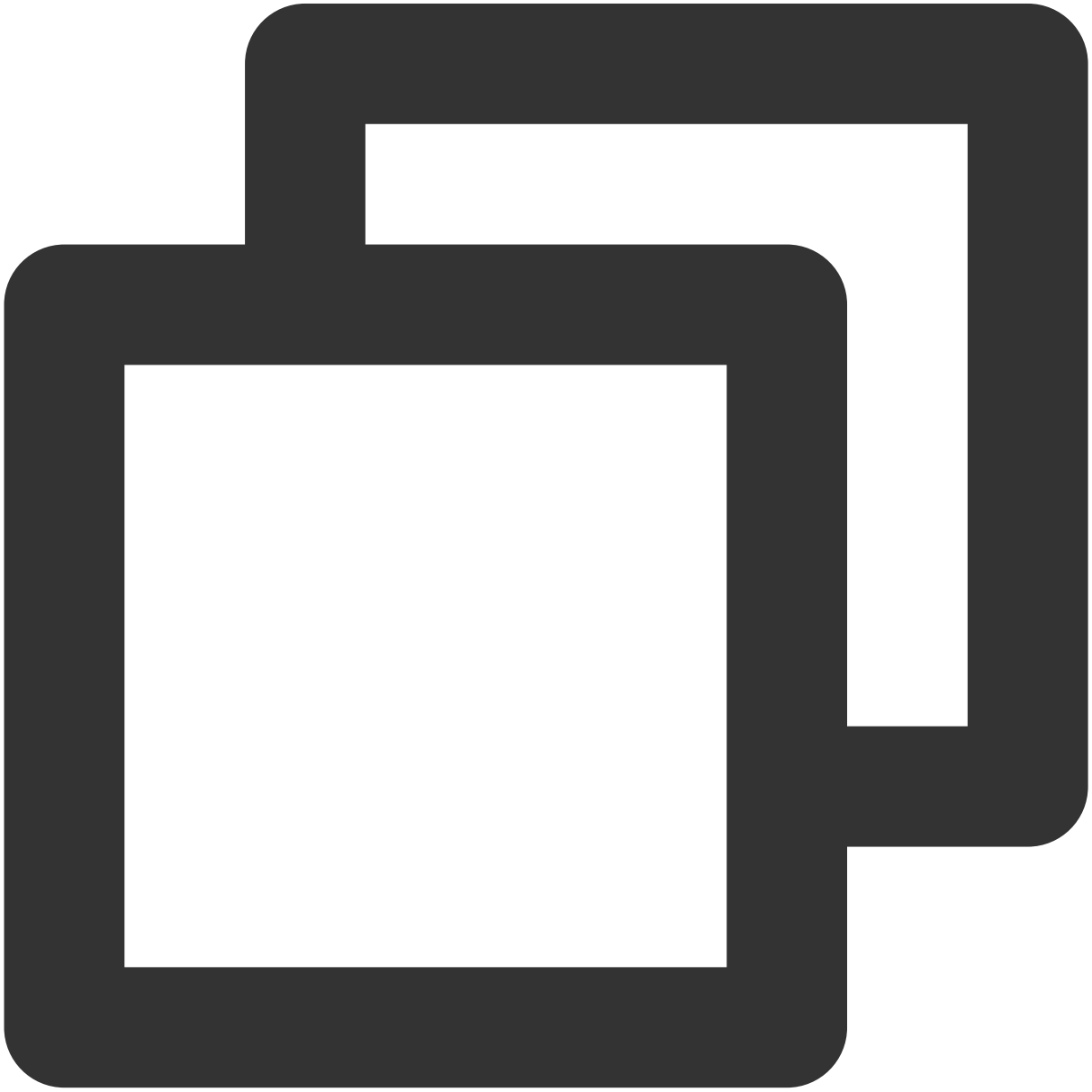
上面代码，分别声明了 `foo` 、 `bar` 、 `i` 三个变量。然后， `foo` 赋值为数值 1 ， `bar` 赋值为字符串 `"hello world"` 。

变量名

变量命名必须符合下面两个规则：
首字符必须是：字母（a-zA-Z），下划线（_）；
剩余字符可以是：字母（a-zA-Z），下划线（_），数字（0-9）。

保留标识符

以下标识符不能作为变量名：



```
delete
```

```
void
typeof

null
undefined
NaN
Infinity
var

if
else

true
false

require

this
function
arguments
return

for
while
do
break
continue
switch
case
default
```

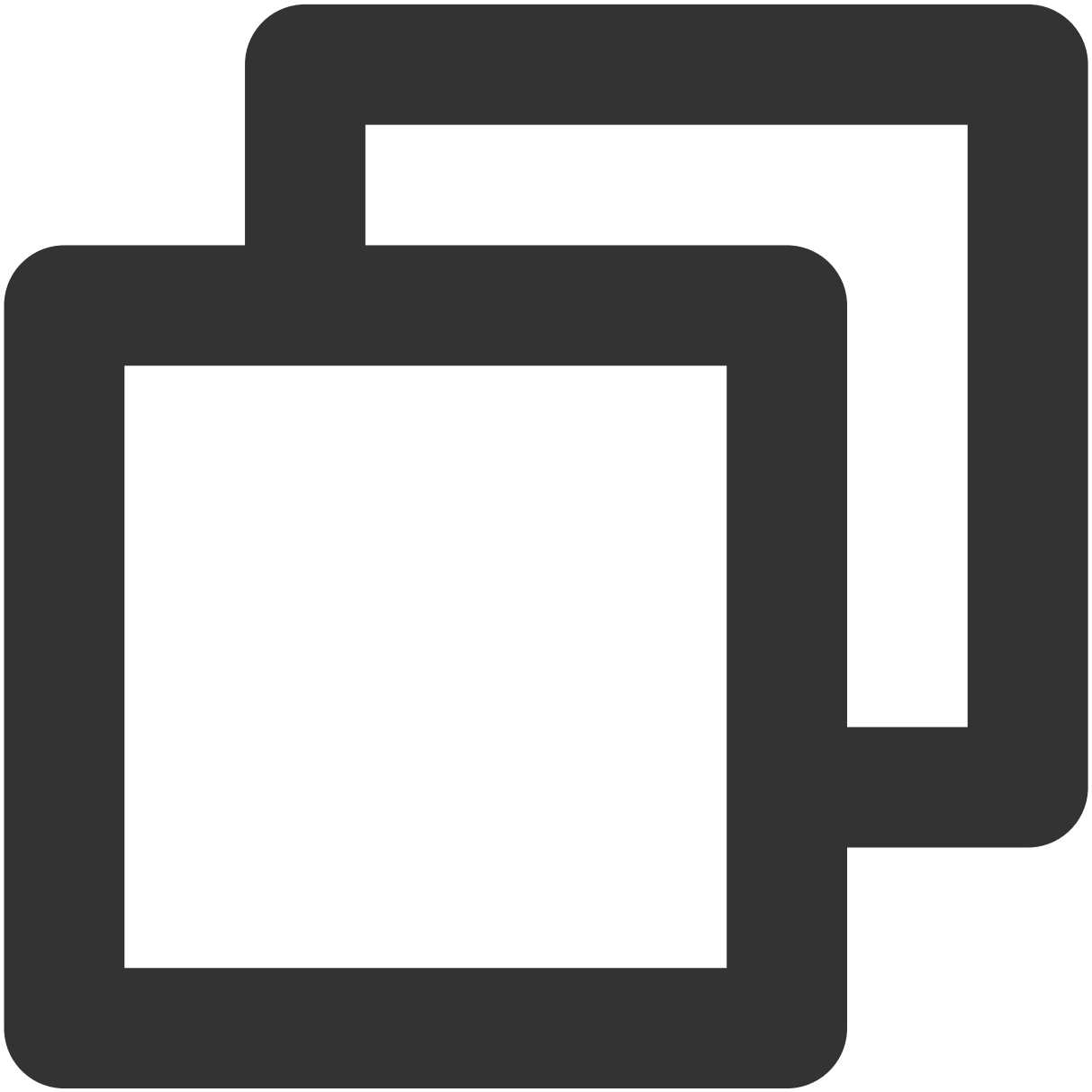
版权所有：腾讯云计算（北京）有限责任公司

注释

最近更新时间：2024-07-12 16:57:22

WXS 主要有 3 种注释的方法。

示例代码：



```
<!-- wxml -->
<wxs module="sample">
// 方法一：单行注释
```

版权所有：腾讯云计算（北京）有限责任公司

```
/*
方法二：多行注释
*/

/*
方法三：结尾注释。即从 /* 开始往后的所有 wxs 代码均被注释

var a = 1;
var b = 2;
var c = "fake";

</wxs>
```

上述例子中，所有 WXS 代码均被注释掉了。

说明：

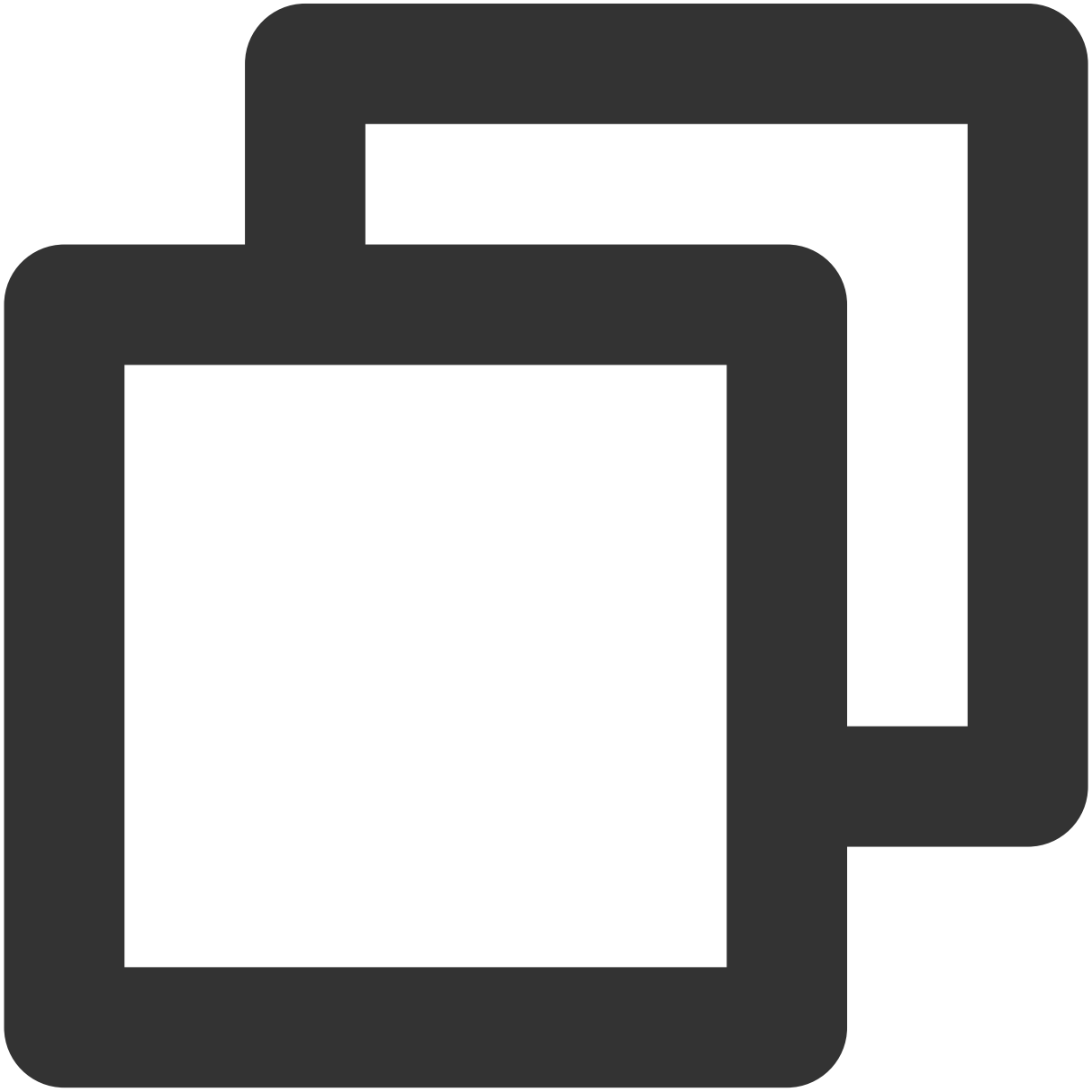
方法三和方法二的唯一区别是，没有 */ 结束符。

运算符

最近更新时间：2024-07-12 16:56:42

基础运算符

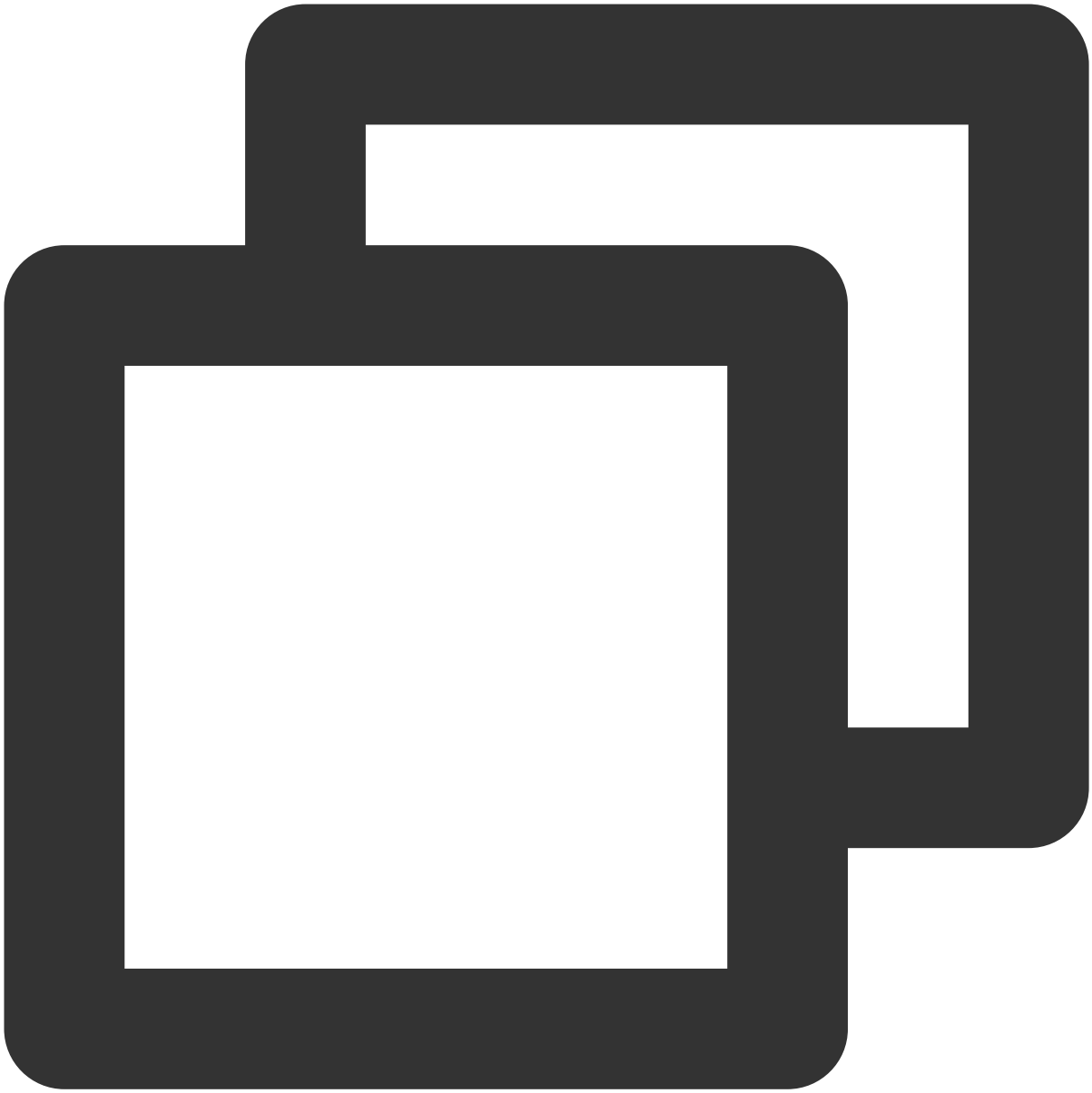
示例代码：



```
var a = 10, b = 20;
```

```
// 加法运算
console.log(30 === a + b);
// 减法运算
console.log(-10 === a - b);
// 乘法运算
console.log(200 === a * b);
// 除法运算
console.log(0.5 === a / b);
// 取余运算
console.log(10 === a % b);
```

加法运算 `+` 也可以用作字符串的拼接。

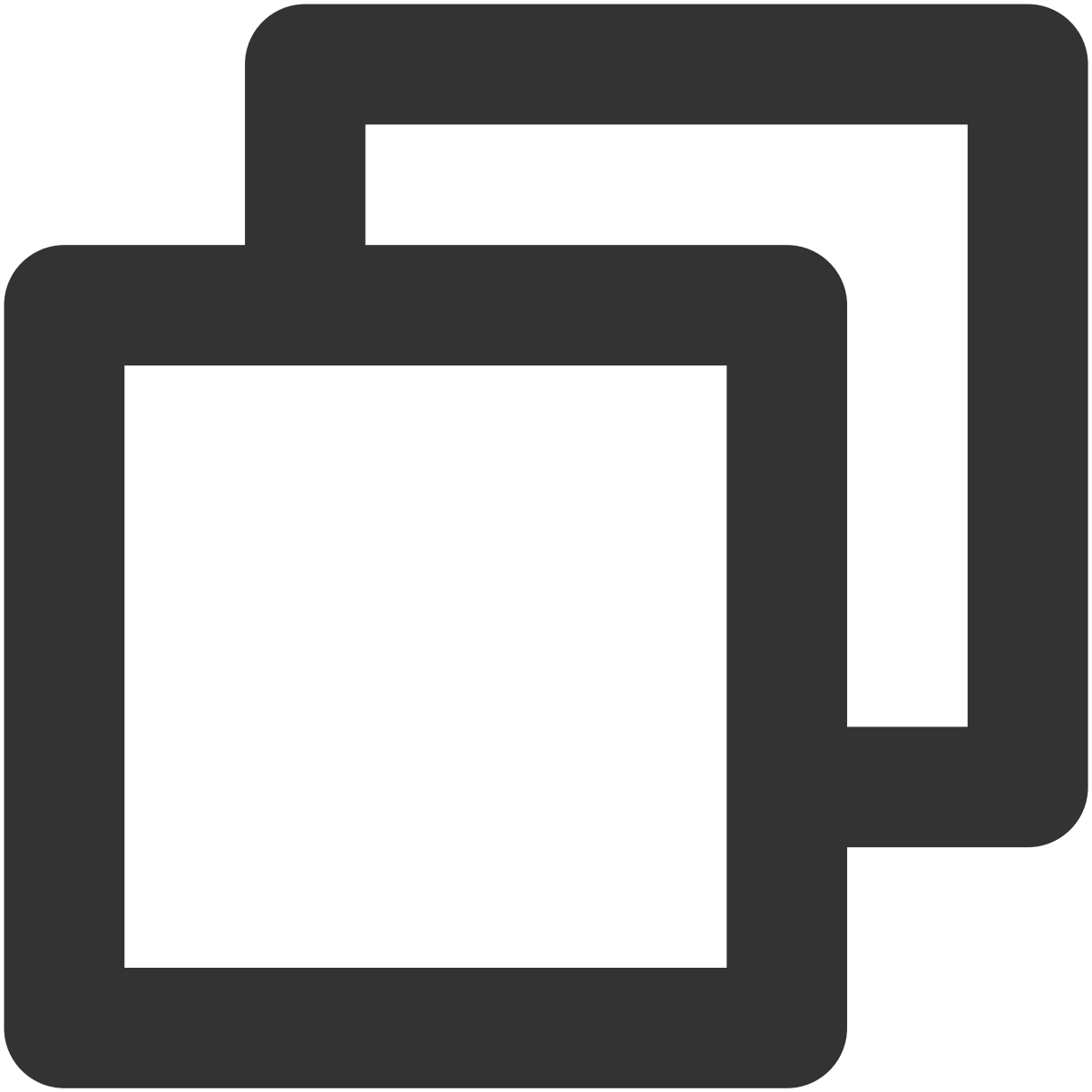


```
var a = '.w' , b = 'xs';

// 字符串拼接
console.log('.wxs' === a + b);
```

一元运算符

示例代码：



```
var a = 10, b = 20;

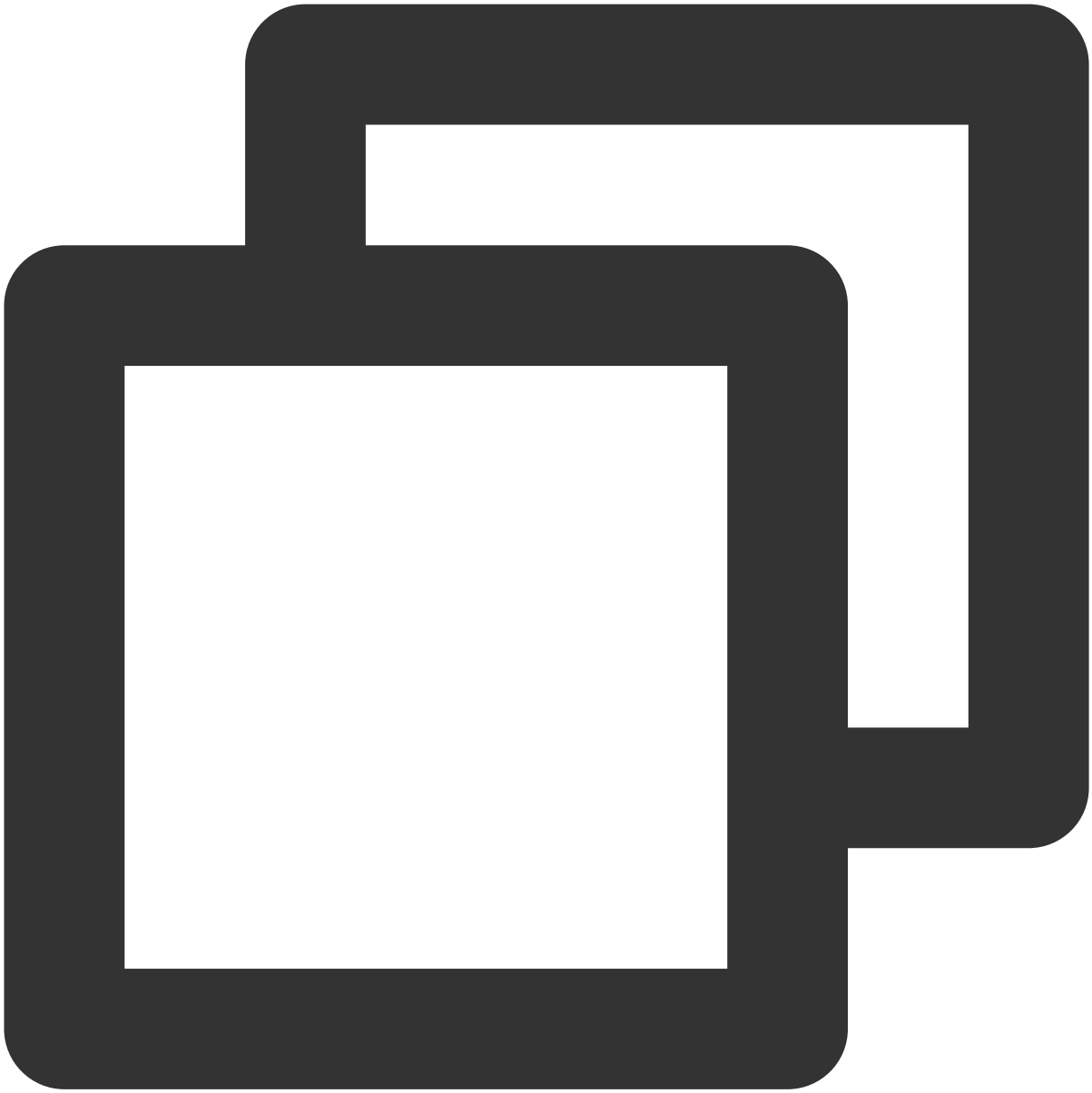
// 自增运算
console.log(10 === a++);
console.log(12 === ++a);
// 自减运算
console.log(12 === a--);
console.log(10 === --a);
// 正值运算
console.log(10 === +a);
// 负值运算
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(0-10 === -a);
// 否运算
console.log(-11 === ~a);
// 取反运算
console.log(false === !a);
// delete 运算
console.log(true === delete a.fake);
// void 运算
console.log(undefined === void a);
// typeof 运算
console.log("number" === typeof a);
```

位运算符

示例代码：



```
var a = 10, b = 20;

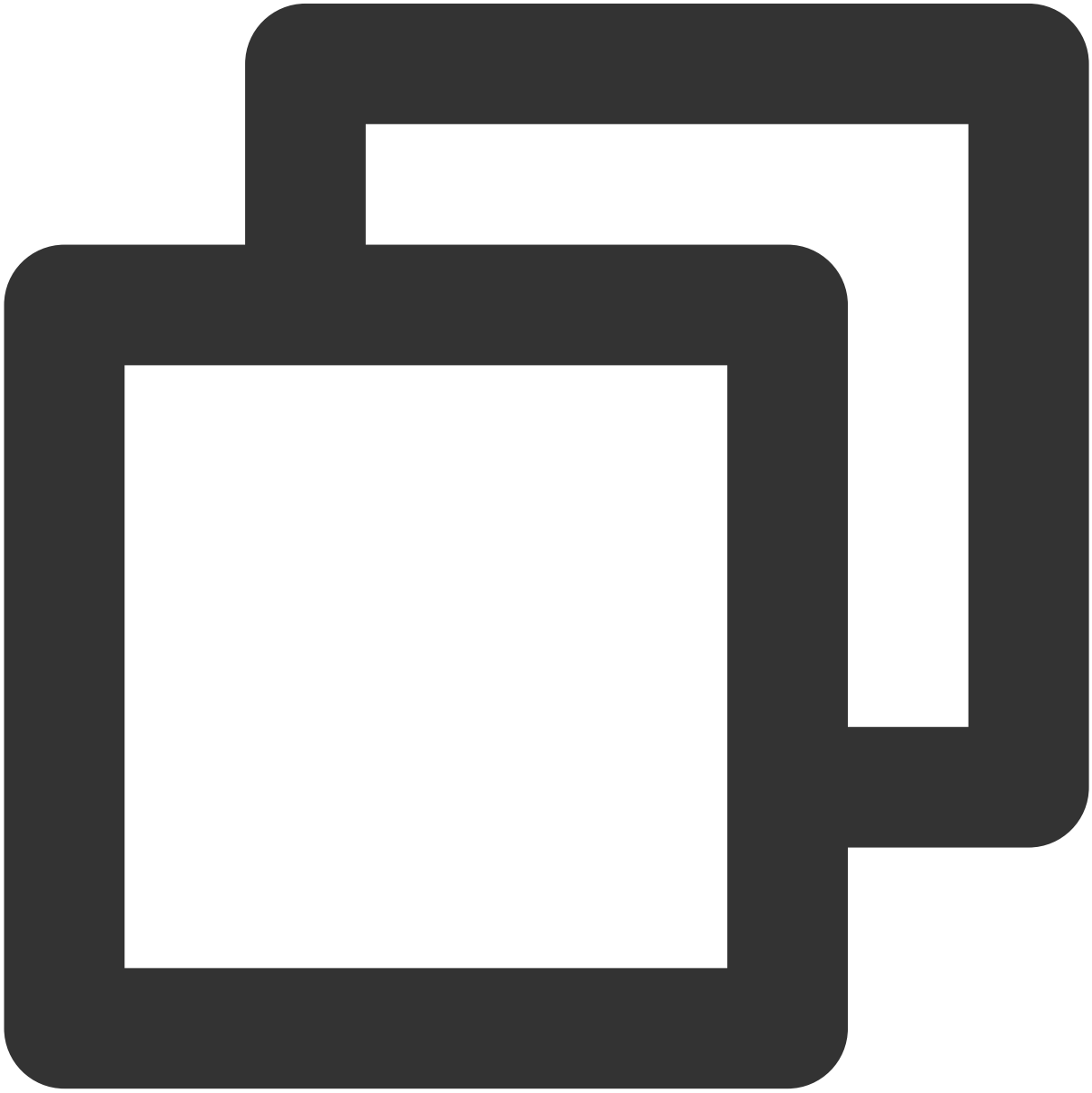
// 左移运算
console.log(80 === (a << 3));
// 带符号右移运算
console.log(2 === (a >> 2));
// 无符号右移运算
console.log(2 === (a >>> 2));
// 与运算
console.log(2 === (a & 3));
// 异或运算
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(9 === (a ^ 3));
// 或运算
console.log(11 === (a | 3));
```

比较运算符

示例代码：

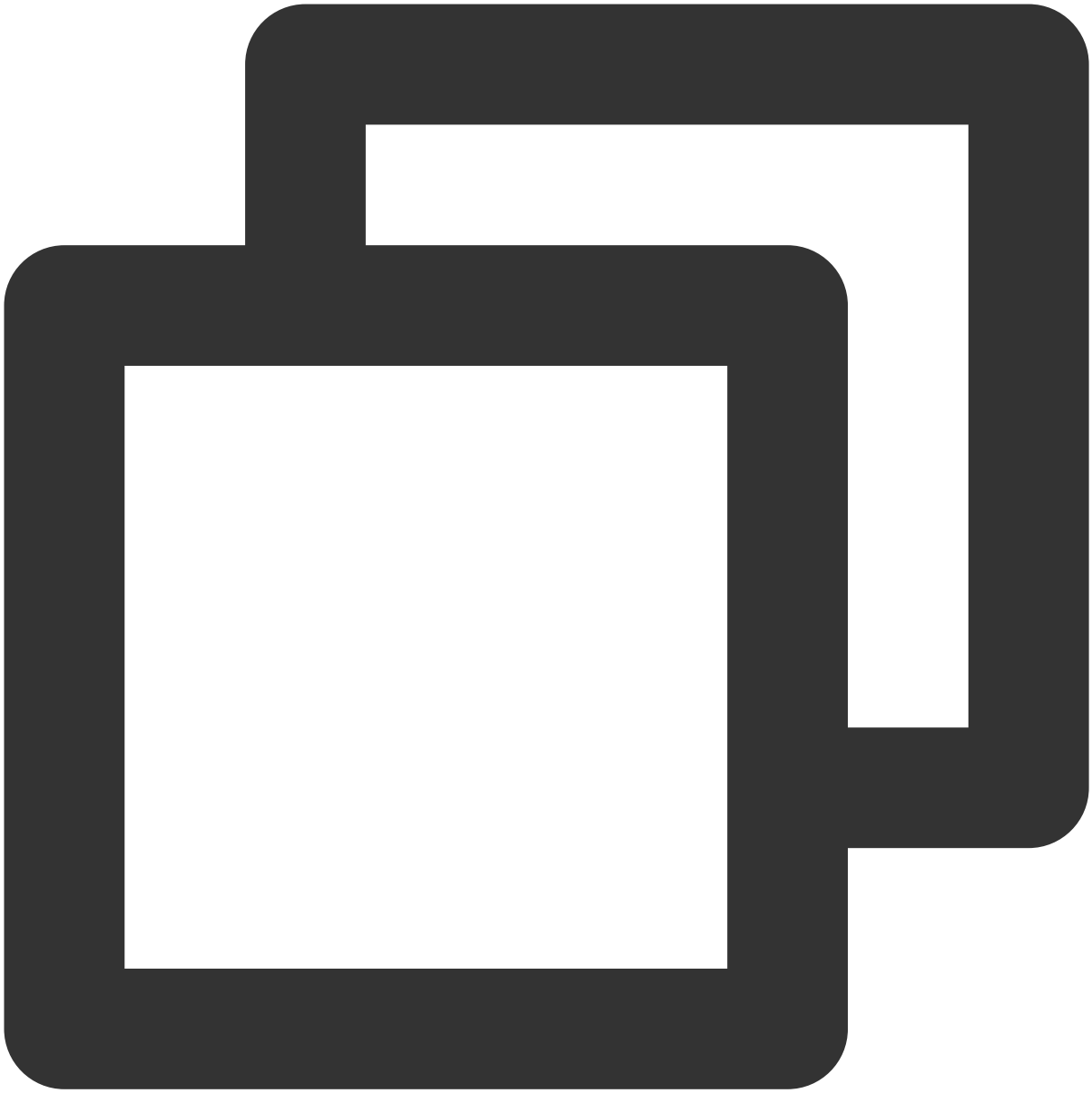


```
var a = 10, b = 20;
```

```
// 小于
console.log(true === (a < b));
// 大于
console.log(false === (a > b));
// 小于等于
console.log(true === (a <= b));
// 大于等于
console.log(false === (a >= b));
```

等值运算符

示例代码：



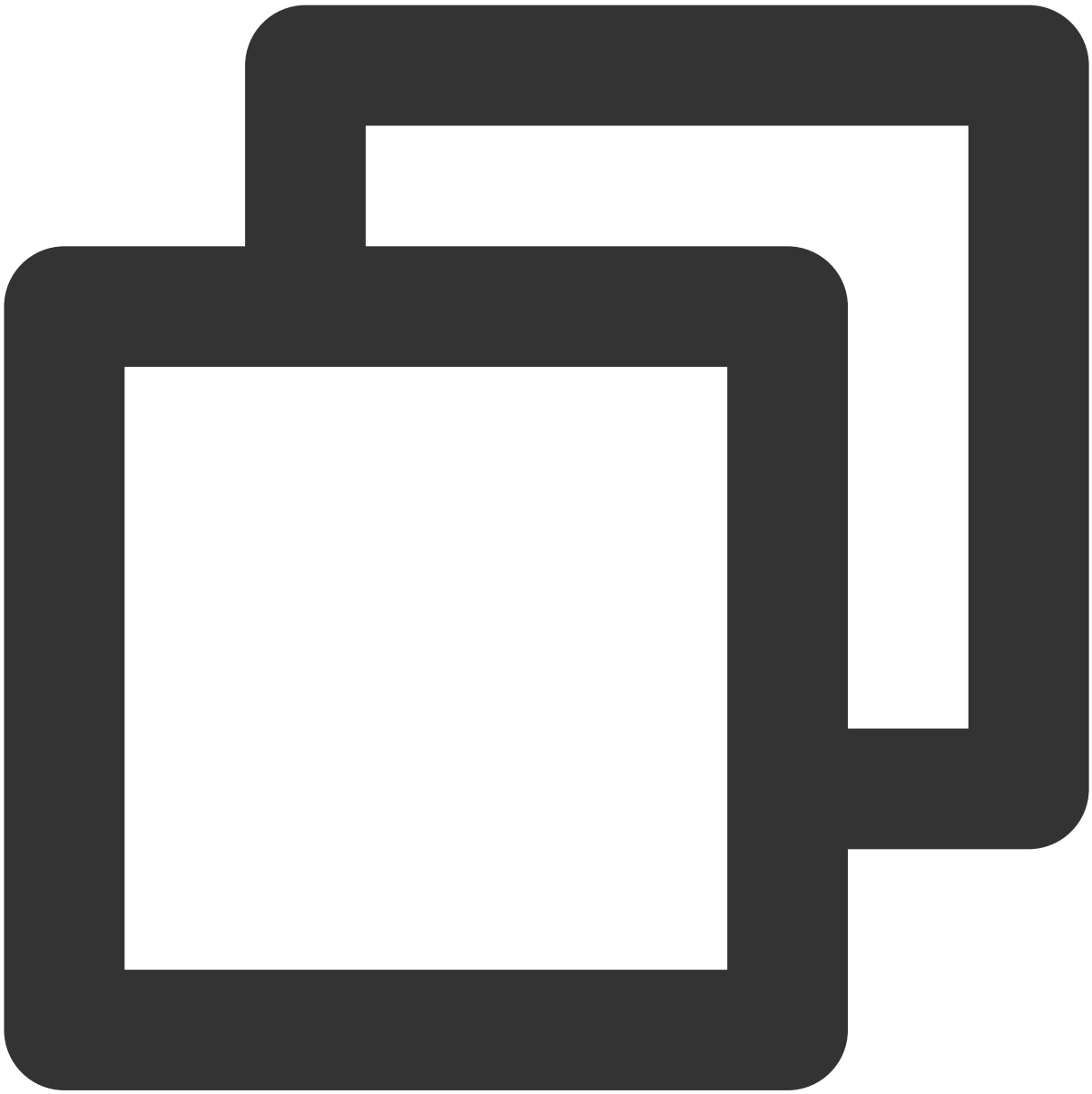
```
var a = 10, b = 20;

// 等号
console.log(false === (a == b));
// 非等号
console.log(true === (a != b));
// 全等号
console.log(false === (a === b));
// 非全等号
console.log(true === (a !== b));
```

版权所有：腾讯云计算（北京）有限责任公司

赋值运算符

示例代码：



```
var a = 10;

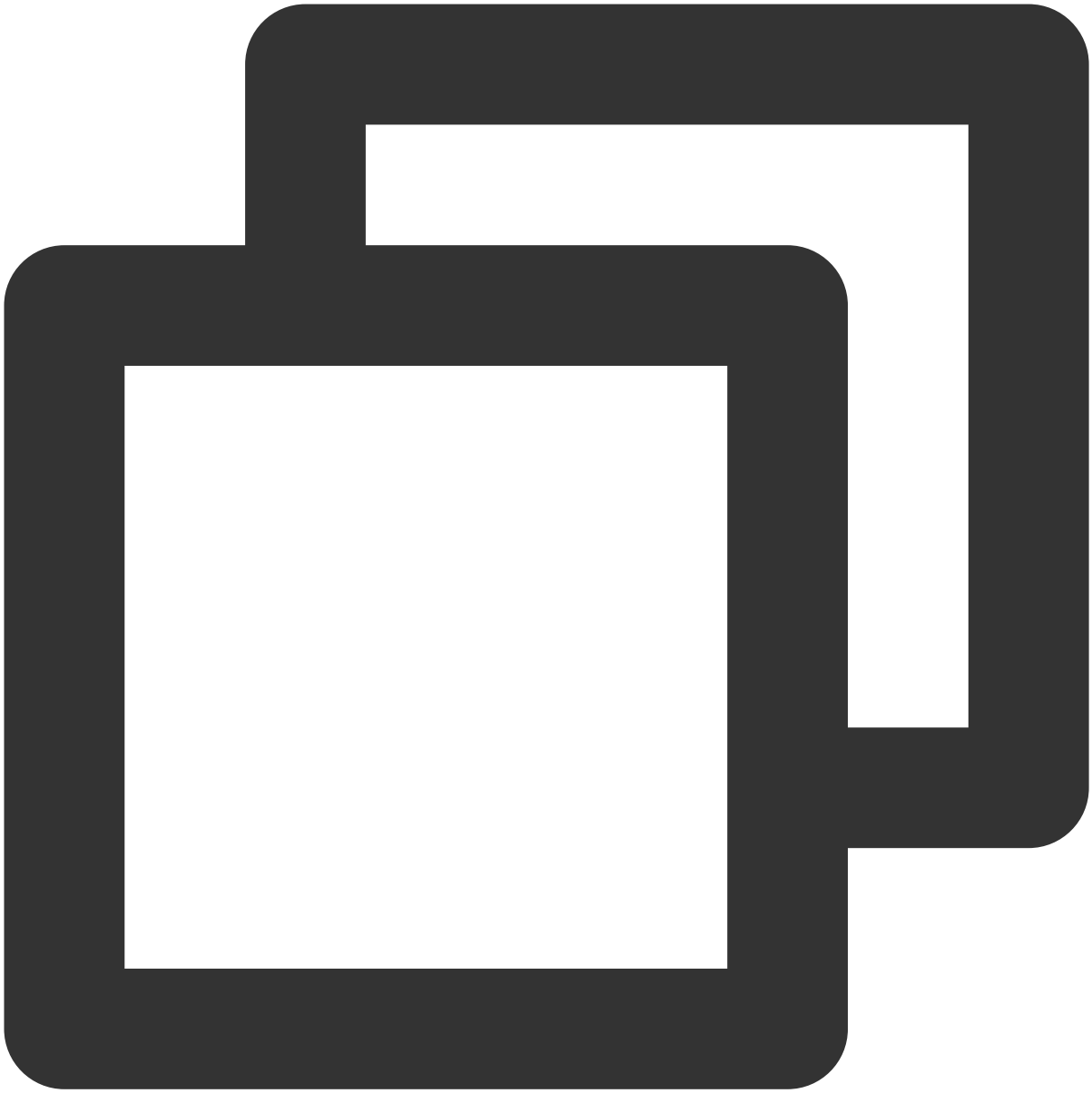
a = 10; a *= 10;
console.log(100 === a);
a = 10; a /= 5;
console.log(2 === a);
a = 10; a %= 7;
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(3 === a);
a = 10; a += 5;
console.log(15 === a);
a = 10; a -= 11;
console.log(-1 === a);
a = 10; a <<= 10;
console.log(10240 === a);
a = 10; a >>= 2;
console.log(2 === a);
a = 10; a >>>= 2;
console.log(2 === a);
a = 10; a &= 3;
console.log(2 === a);
a = 10; a ^= 3;
console.log(9 === a);
a = 10; a |= 3;
console.log(11 === a);
```

二元逻辑运算符

示例代码：

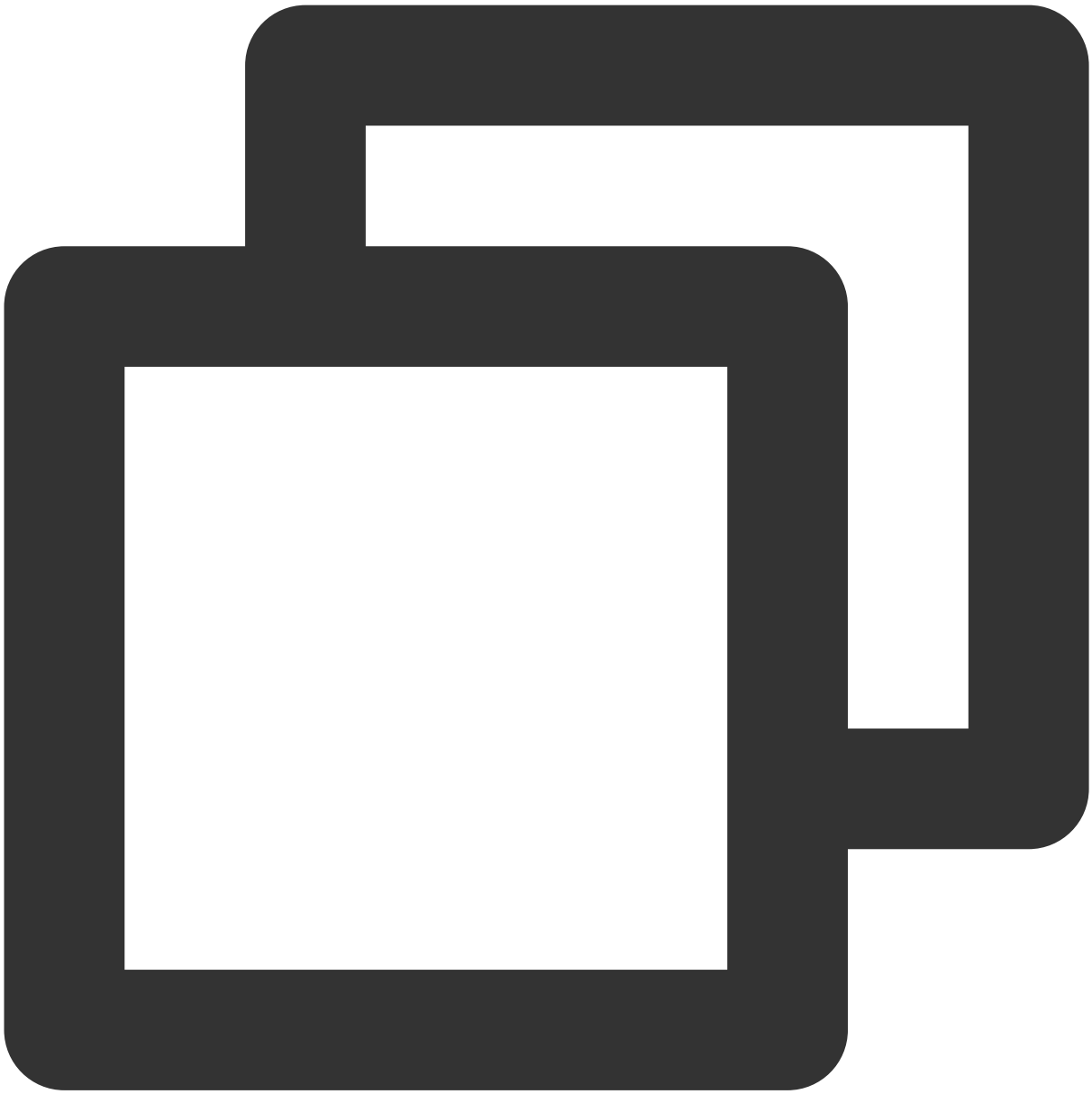


```
var a = 10, b = 20;

// 逻辑与
console.log(20 === (a && b));
// 逻辑或
console.log(10 === (a || b));
```

其他运算符

示例代码：



```
var a = 10, b = 20;

//条件运算符
console.log(20 === (a >= 10 ? a + 10 : b + 10));
//逗号运算符
console.log(20 === (a, b));
```

运算符优先级

优先级	运算符	说明	结合性
20	(...)	括号	n/a
19	... .	成员访问	从左到右
	... [...]	成员访问	从左到右
	... (...)	函数调用	从左到右
17	... ++	后置递增	n/a
	... --	后置递减	n/a
16	! ...	逻辑非	从右到左
	~ ...	按位非	从右到左
	+ ...	一元加法	从右到左
	- ...	一元减法	从右到左
	++ ...	前置递增	从右到左
	-- ...	前置递减	从右到左
	typeof ...	typeof	从右到左
	void ...	void	从右到左
	delete ...	delete	从右到左
14	... * ...	乘法	从左到右
	... / ...	除法	从左到右
	... % ...	取模	从左到右
13	... + ...	加法	从左到右
	... - ...	减法	从左到右
12	... << ...	按位左移	从左到右
	... >> ...	按位右移	从左到右
	... >>> ...	无符号右移	从左到右

版权所有：腾讯云计算（北京）有限责任公司

11	... < ...	小于	从左到右
	... <= ...	小于等于	从左到右
	... > ...	大于	从左到右
	... >= ...	大于等于	从左到右
10	... == ...	等号	从左到右
	... != ...	非等号	从左到右
	... === ...	全等号	从左到右
	... !== ...	非全等号	从左到右
9	... & ...	按位与	从左到右
8	... ^ ...	按位异或	从左到右
7	... | ...	按位或	从左到右
6	... && ...	逻辑与	从左到右
5	... || ...	逻辑或	从左到右
4	... ? ... : ...	条件运算符	从右到左
3	... = ...	赋值	从右到左
	... += ...	赋值	从右到左
	... -= ...	赋值	从右到左
	... *= ...	赋值	从右到左
	... /= ...	赋值	从右到左
	... %= ...	赋值	从右到左
	... <<= ...	赋值	从右到左
	... >>= ...	赋值	从右到左
	... >>>= ...	赋值	从右到左
	... &= ...	赋值	从右到左
	... ^= ...	赋值	从右到左

版权所有：腾讯云计算（北京）有限责任公司

	... | = ...	赋值	从右到左
0	... , ...	逗号	从左到右

版权所有：腾讯云计算（北京）有限责任公司

语句

最近更新时间：2024-07-12 16:54:52

if 语句

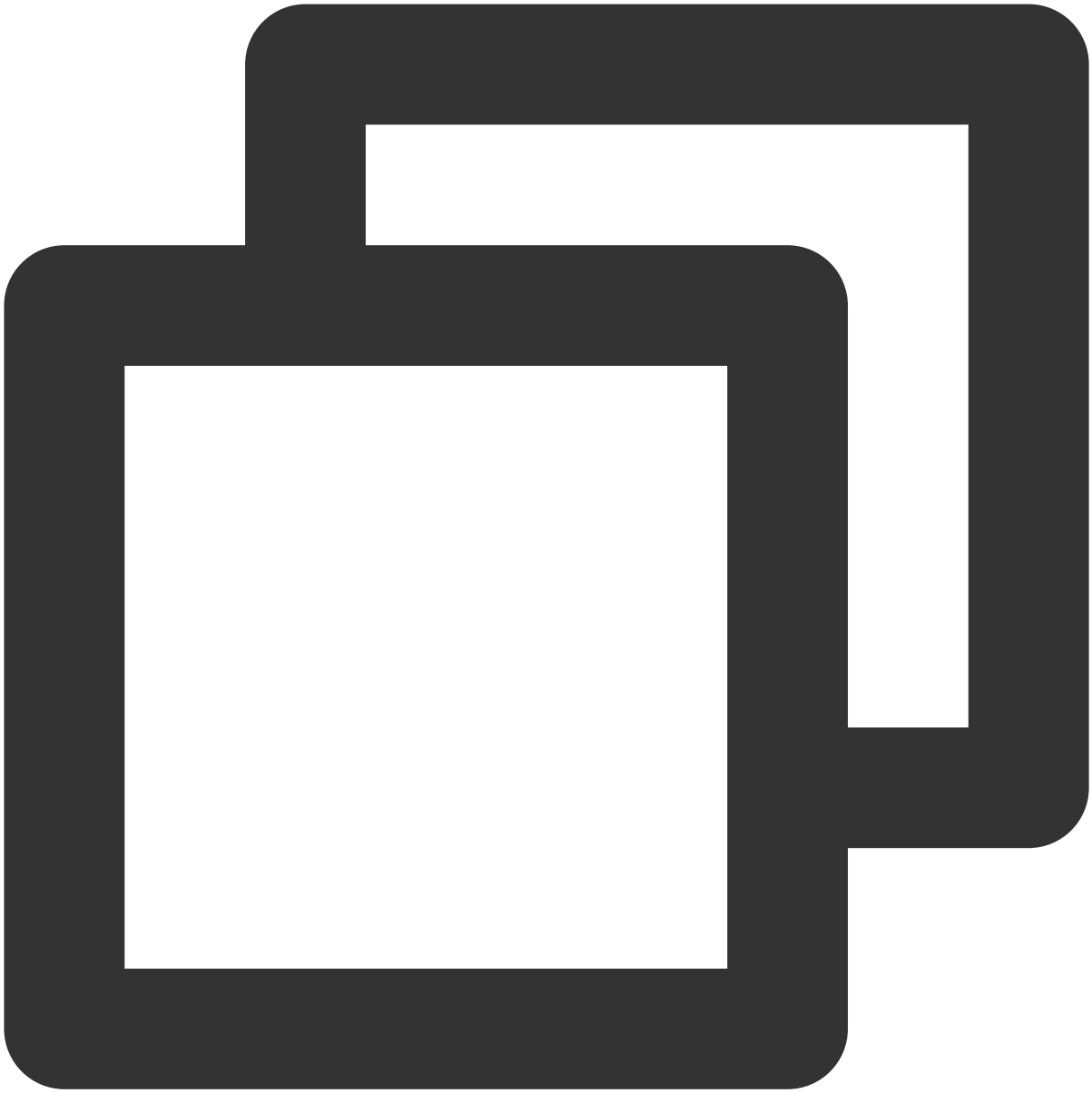
在 WXS 中，可以使用以下格式的 `if` 语句：

`if (expression) statement` ：当 `expression` 为 truthy 时，执行 `statement` ；

`if (expression) statement1 else statement2` ：当 `expression` 为 truthy 时，执行 `statement1` 。否则，执行 `statement2` ；

`if ... else if ... else statementN` 通过该句型，可以在 `statement1 ~ statementN` 之间选其中一个执行。

示例语句:



```
// if ...

if (表达式) 语句;

if (表达式)
  语句;

if (表达式) {
  代码块;
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
// if ... else

if (表达式) 语句;
else 语句;

if (表达式)
  语句;
else
  语句;

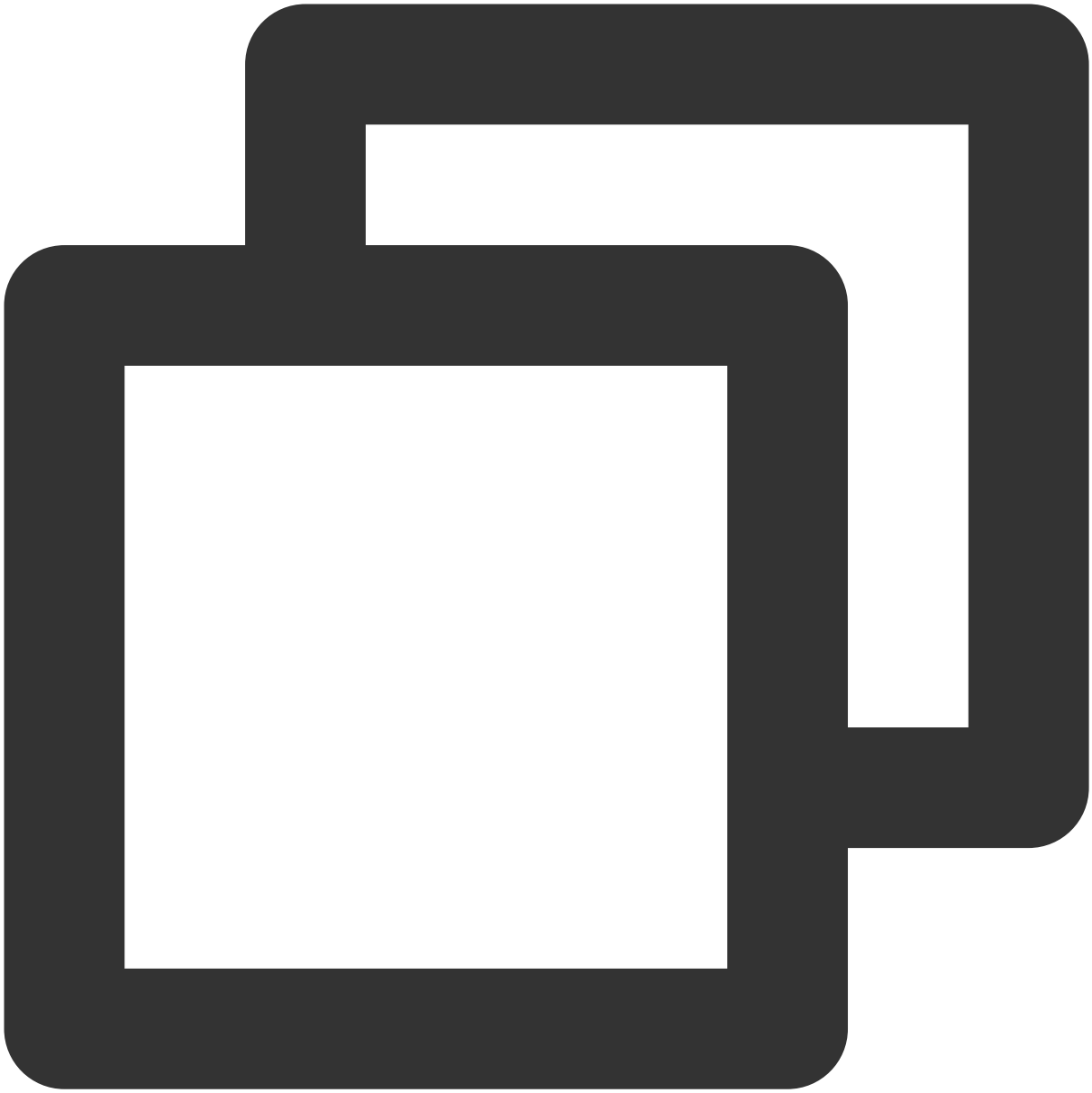
if (表达式) {
  代码块;
} else {
  代码块;
}

// if ... else if ... else ...

if (表达式) {
  代码块;
} else if (表达式) {
  代码块;
} else if (表达式) {
  代码块;
} else {
  代码块;
}
```

switch 语句

示例语法：



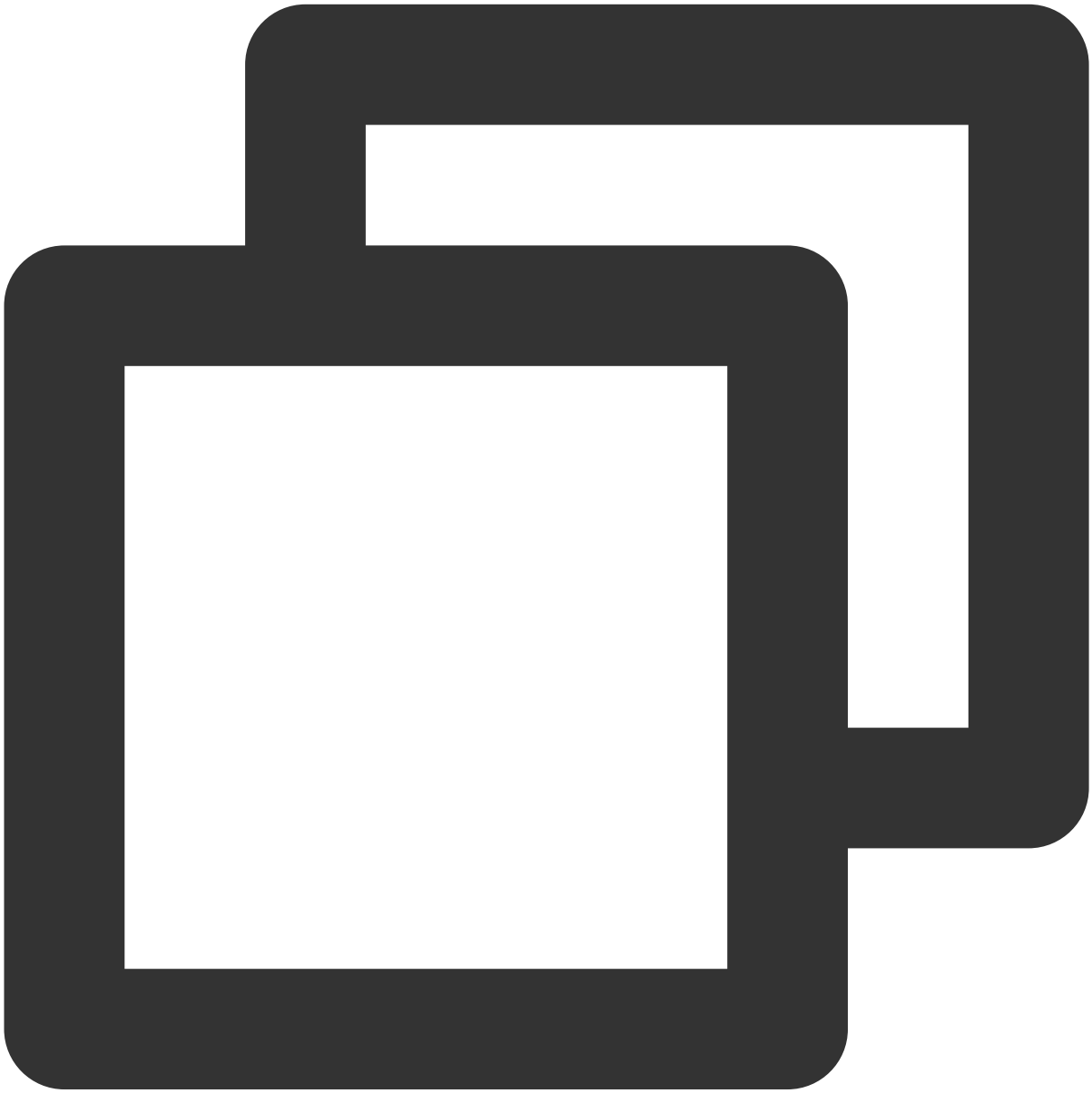
```
switch (表达式) {
  case 变量:
    语句;
  case 数字:
    语句;
    break;
  case 字符串:
    语句;
  default:
    语句;
}
```

版权所有：腾讯云计算（北京）有限责任公司

default 分支可以省略不写；

case 关键词后面只能使用： `变量` ， `数字` ， `字符串` 。

示例代码：

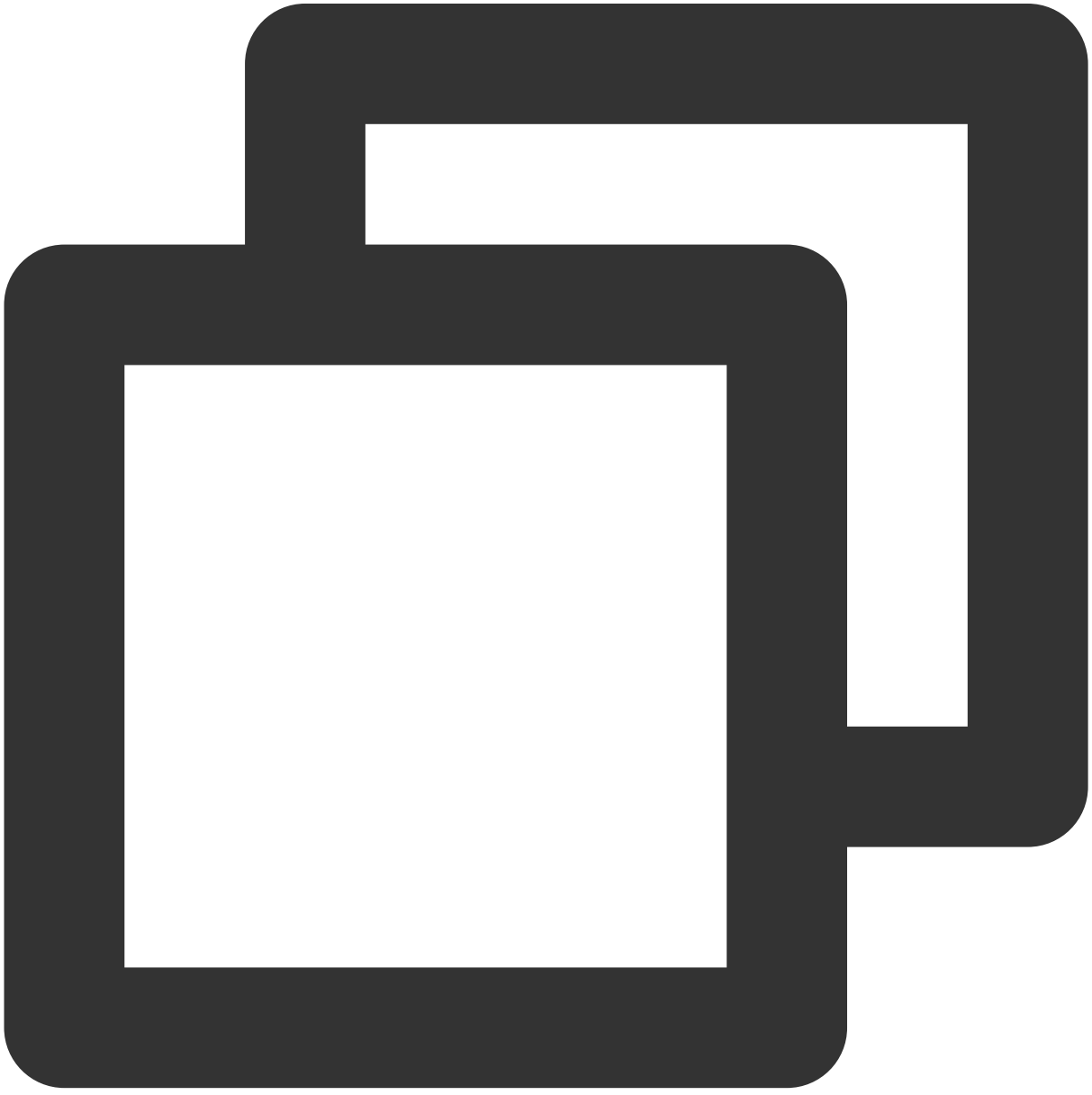


```
var exp = 10;

switch ( exp ) {
case "10":
  console.log("string 10");
  break;
```

```
case 10:
  console.log("number 10");
  break;
case exp:
  console.log("var exp");
  break;
default:
  console.log("default");
}
```

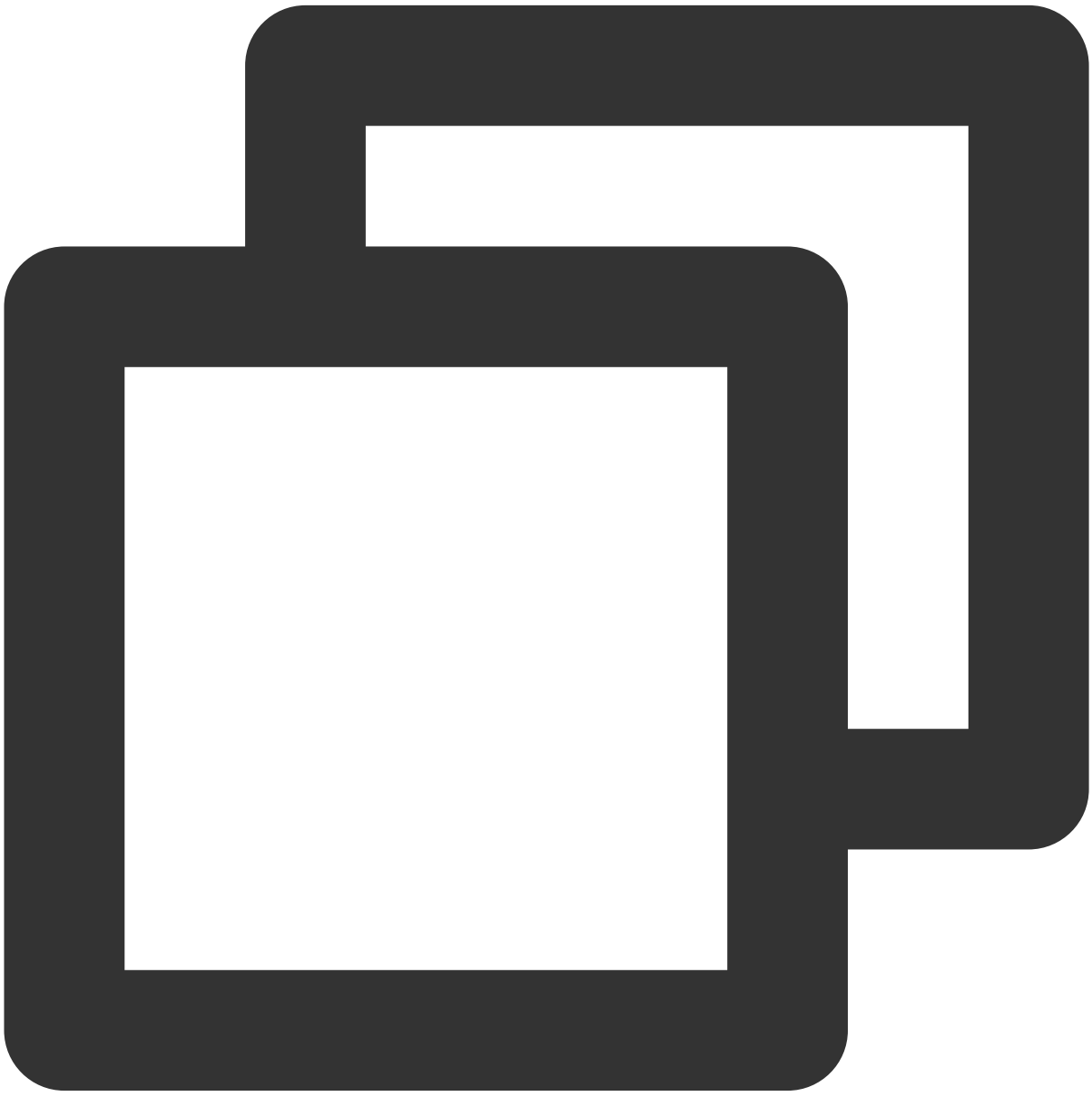
输出：



```
number 10
```

for 语句

示例语法：

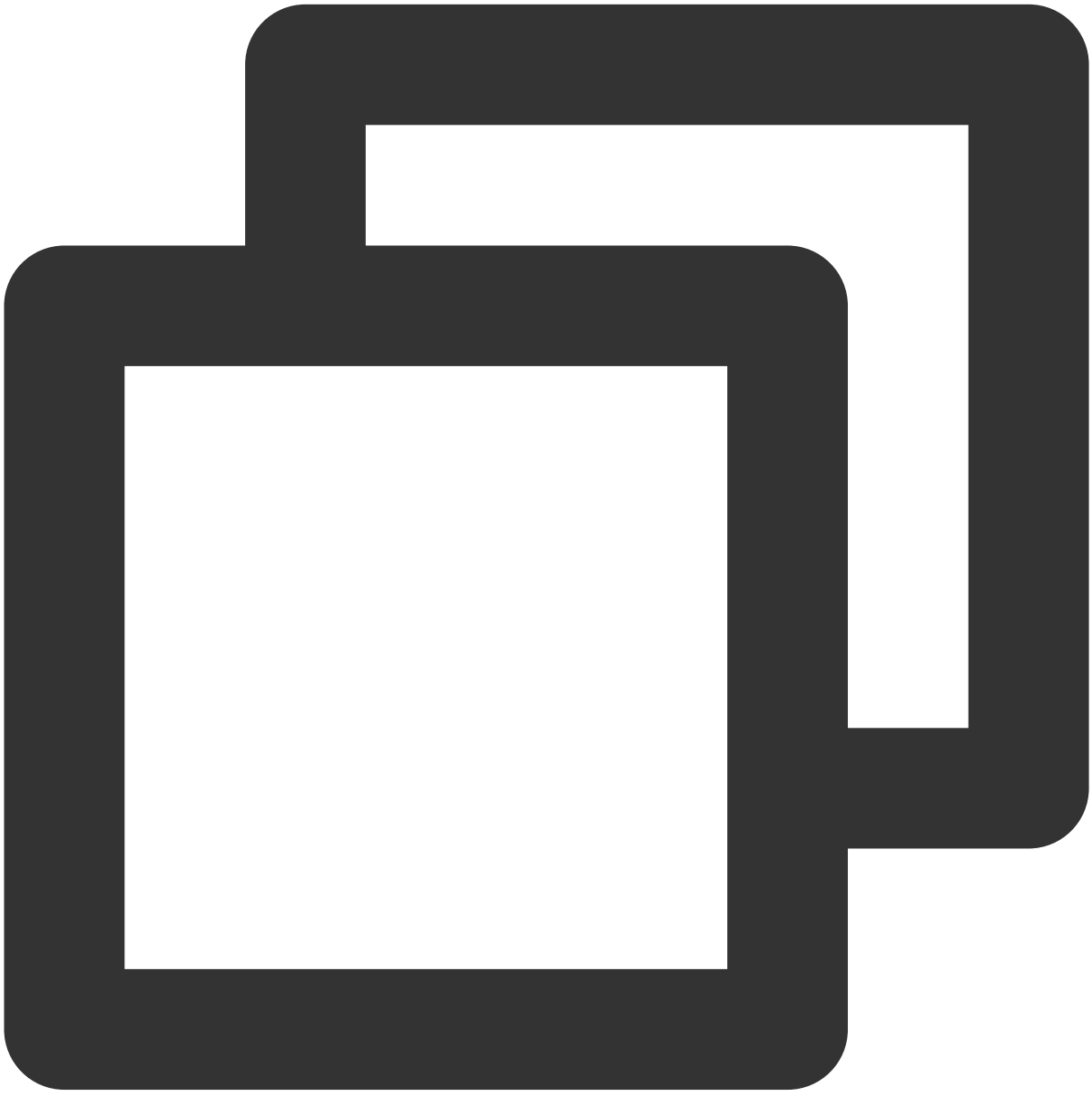


```
for (语句; 语句; 语句)
  语句;
```

```
for (语句; 语句; 语句) {
  代码块;
}
```

支持使用 `break`，`continue` 关键词。

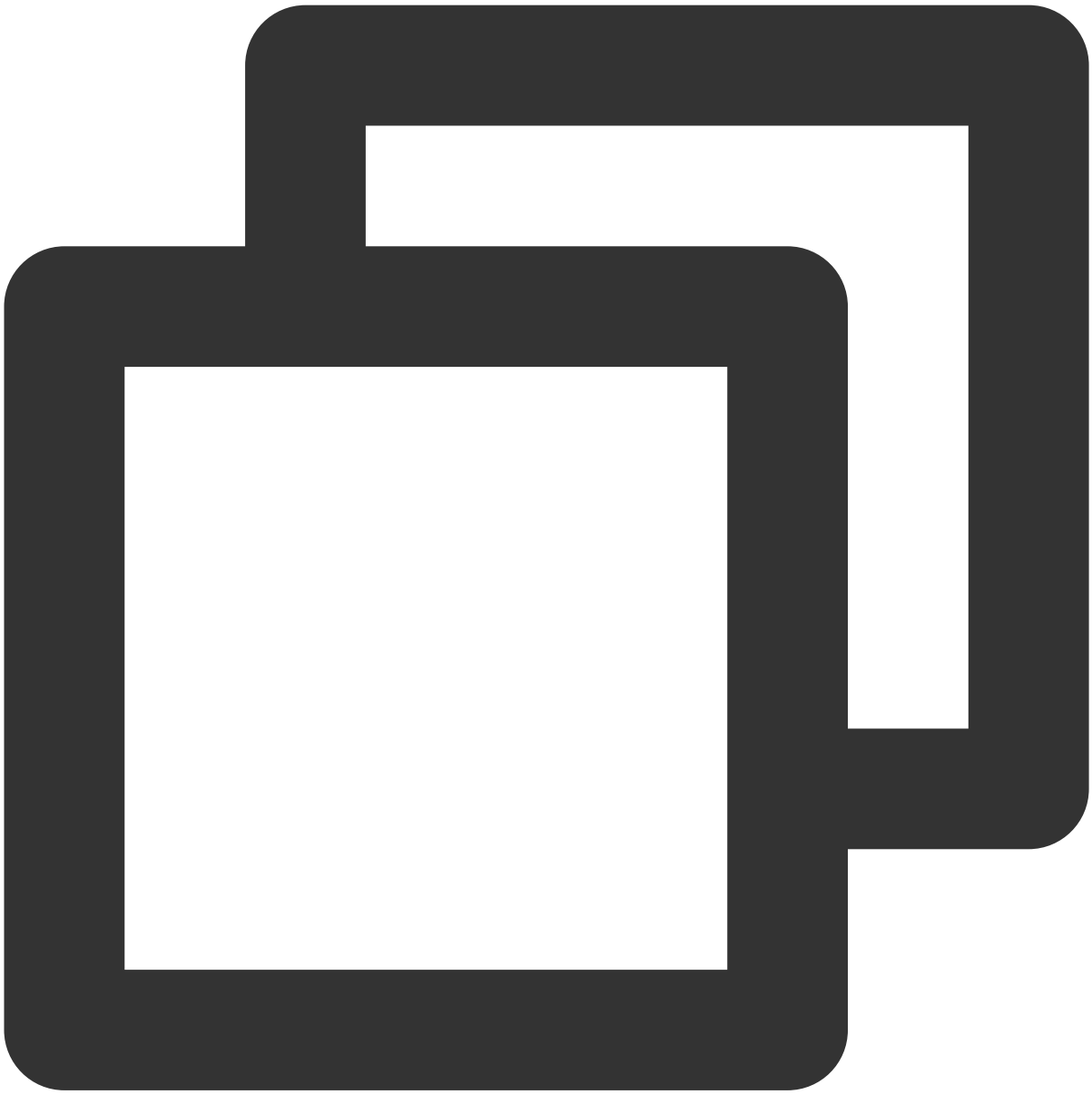
示例代码：



```
for (var i = 0; i < 3; ++i) {
  console.log(i);
  if( i >= 1) break;
```

```
}
```

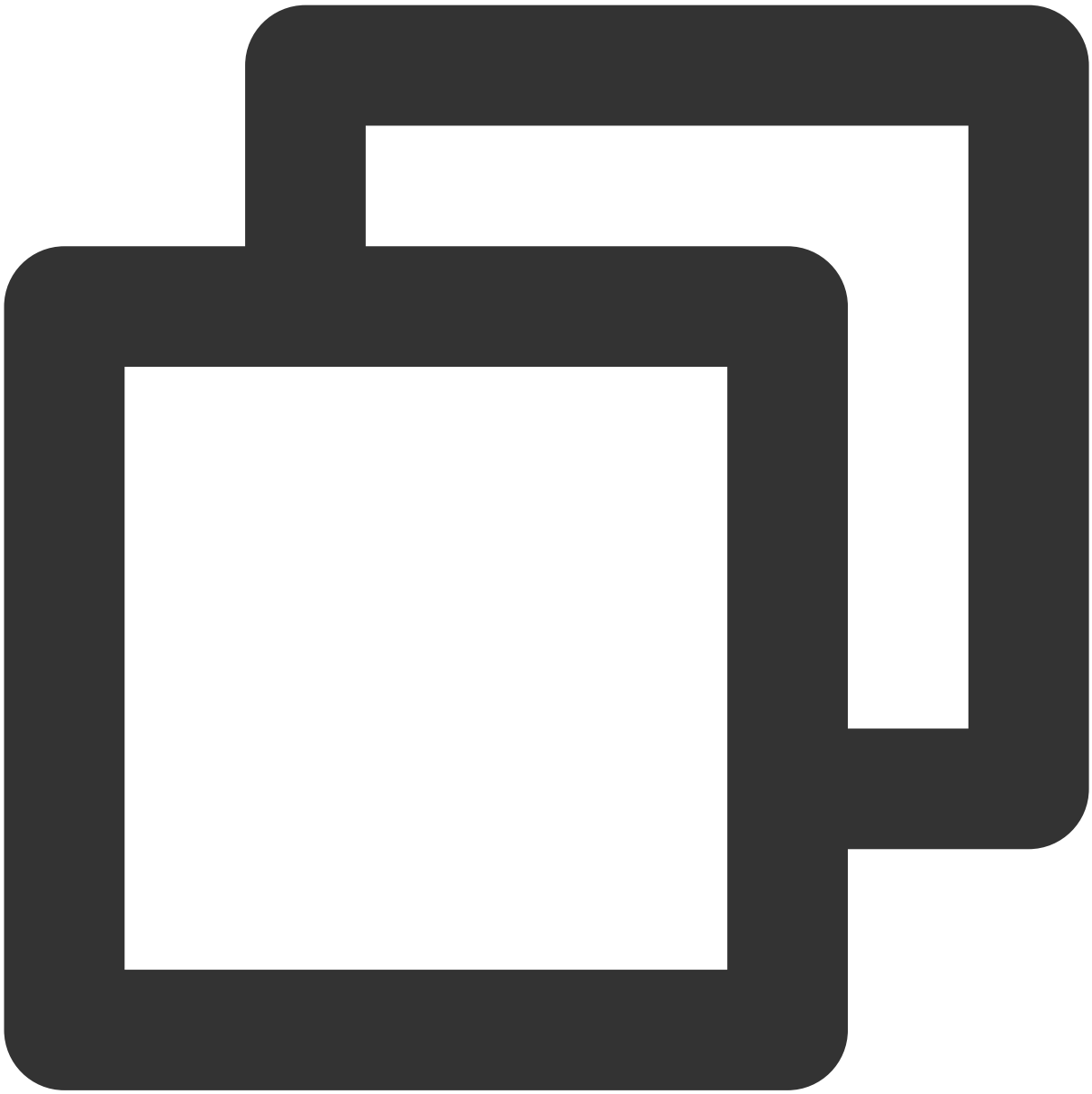
输出：



```
0
1
```

while 语句

示例语法：



```
while (表达式)
  语句;

while (表达式){
  代码块;
}

do {
  代码块;
```

```
} while (表达式)
```

当表达式为 true 时，循环执行 `语句` 或 `代码块` ；

支持使用 `break` ， `continue` 关键词。

版权所有：腾讯云计算（北京）有限责任公司

数据类型

最近更新时间：2024-07-12 16:55:39

WXS 语言目前共有以下几种数据类型：

number ： 数值

string ：字符串

boolean：布尔值

object：对象

function：函数

array : 数组

date：日期

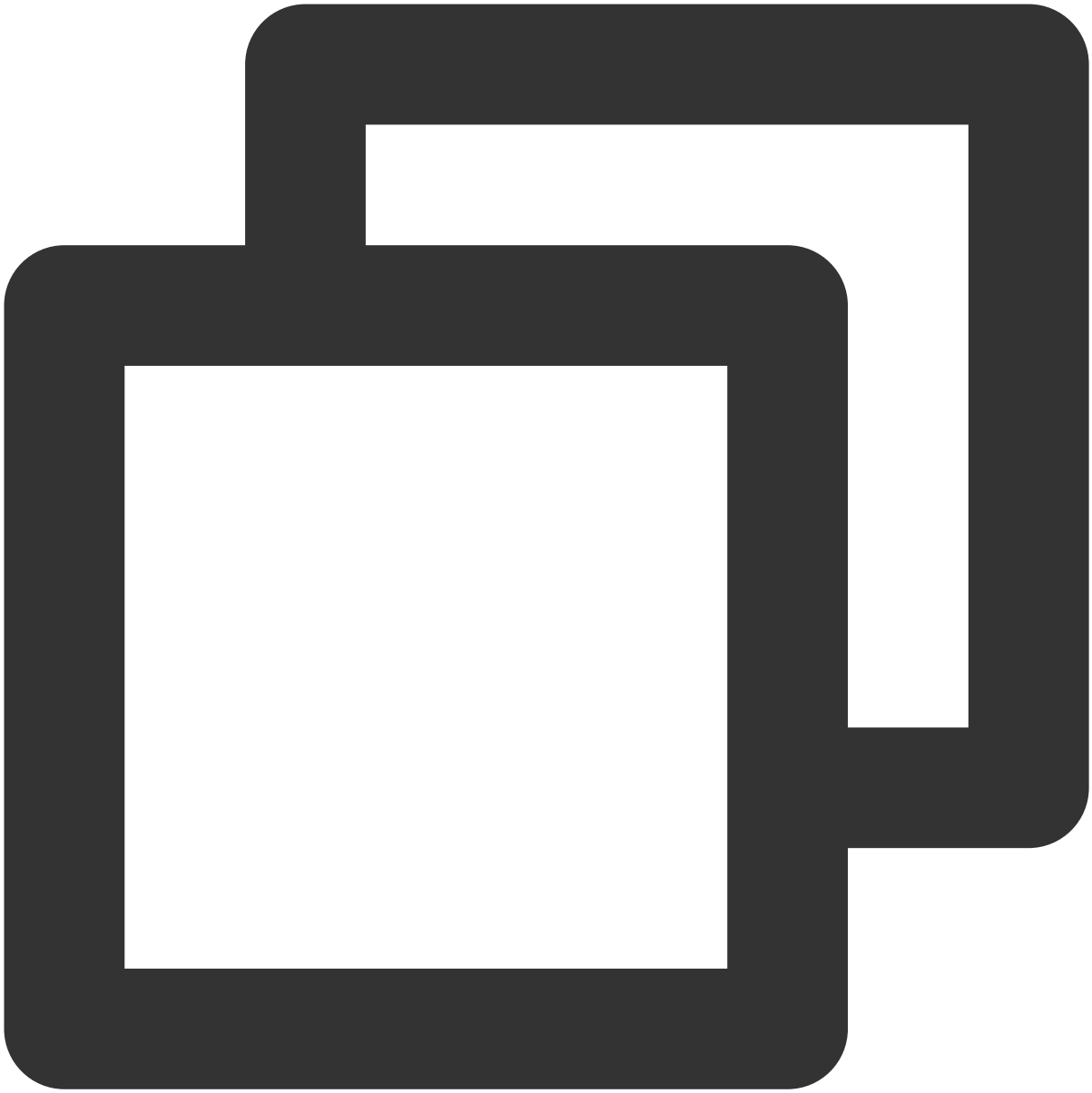
regexp：正则

number

语法

number 包括两种数值：整数，小数。

版权所有：腾讯云计算（北京）有限责任公司



```
var a = 10;
var PI = 3.141592653589793;
```

属性

`constructor` ：返回字符串 `"Number"` 。

方法

`toString` ；

版权所有：腾讯云计算（北京）有限责任公司

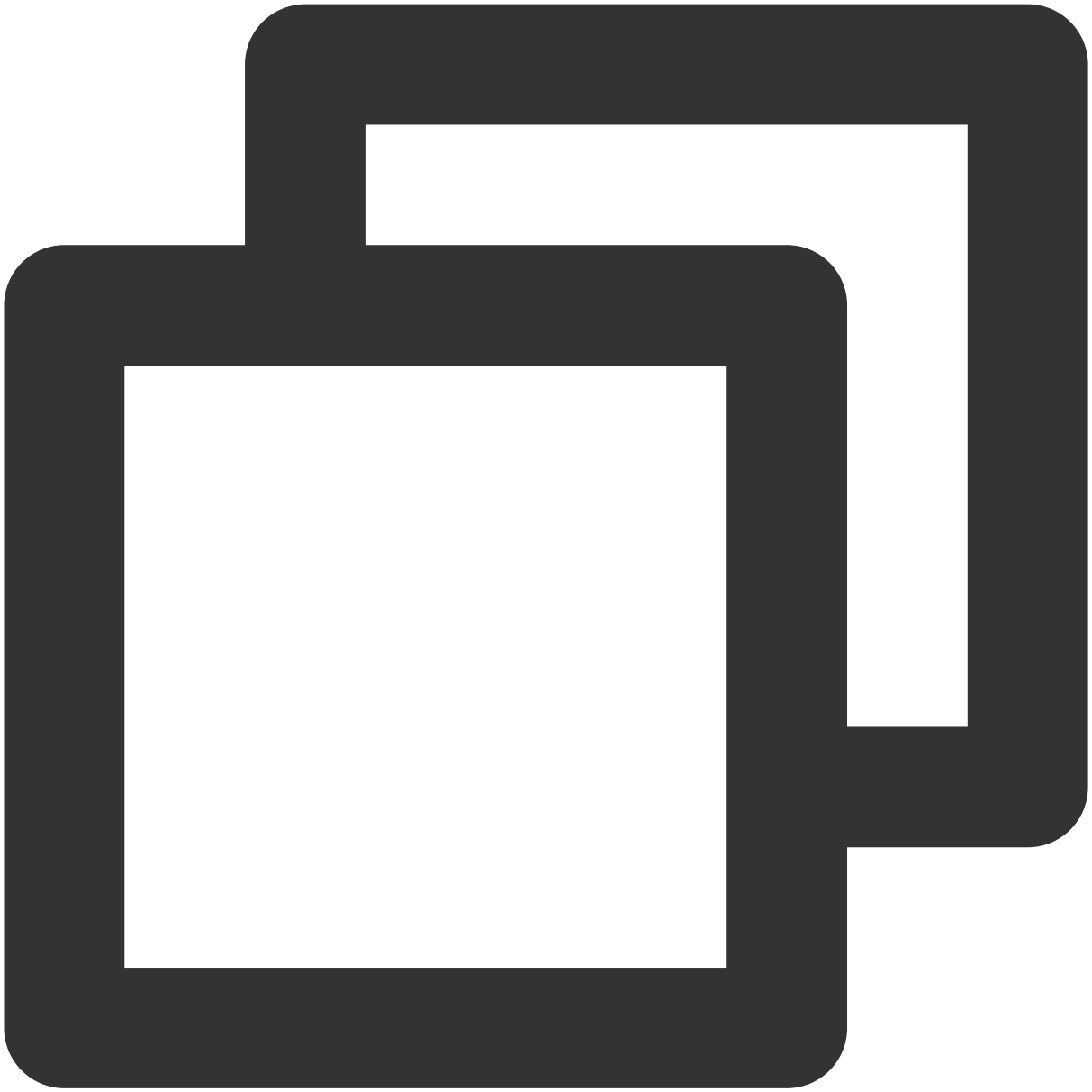
`toLocaleString` ；
`valueOf` ；
`toFixed` ；
`toExponential` ；
`toPrecision` 。

> 以上方法的具体使用请参考 `ES5` 标准。

string

语法

string 有两种写法：



```
'hello world';
"hello world";
```

属性

`constructor` ：返回字符串 `"String"` ；
`length` 。

版权所有：腾讯云计算（北京）有限责任公司

除constructor外属性的具体含义请参考 `ES5` 标准。

方法

`toString` ；
`valueOf` ；
`charAt` ；
`charCodeAt` ；
`concat` ；
`indexOf` ；
`lastIndexOf` ；
`localeCompare` ；
`match` ；
`replace` ；
`search` ；
`slice` ；
`split` ；
`substring` ；
`toLowerCase` ；
`toLocaleLowerCase` ；
`toUpperCase` ；
`toLocaleUpperCase` ；
`trim` 。

以上方法的具体使用请参考 `ES5` 标准。

boolean

语法

布尔值只有两个特定的值： `true` 和 `false` 。

属性

`constructor` ：返回字符串 `"Boolean"` 。

版权所有：腾讯云计算（北京）有限责任公司

方法

`toString` ；

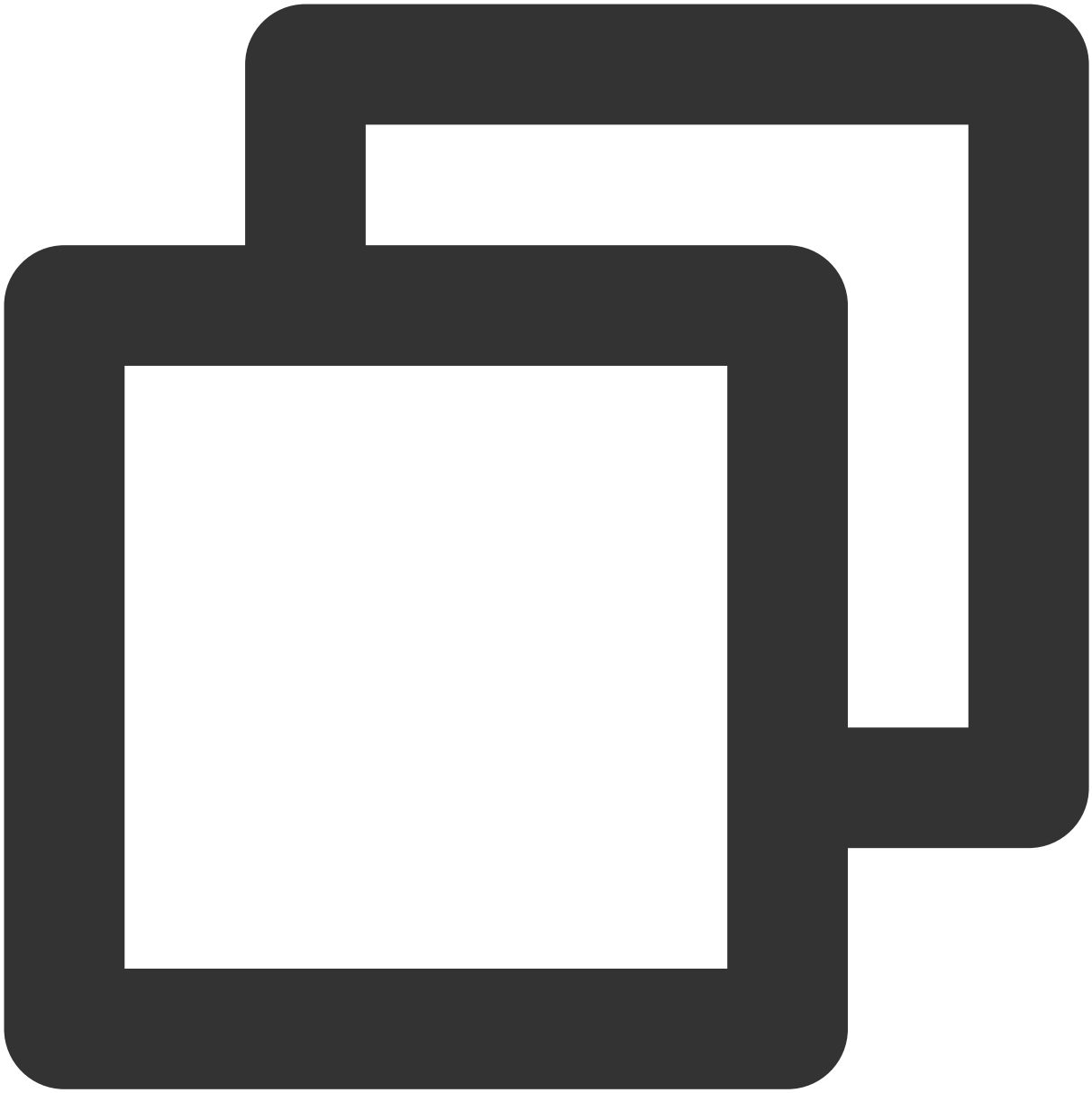
`valueOf` 。

> 以上方法的具体使用请参考 `ES5` 标准。

object

语法

object 是一种无序的键值对。使用方法如下所示：



```
var o = {} //生成一个新的空对象

//生成一个新的非空对象
o = {
  'string'   : 1,   //object 的 key 可以是字符串
  const_var : 2,   //object 的 key 也可以是符合变量定义规则的标识符
  func       : {}, //object 的 value 可以是任何类型
};

//对象属性的读操作
console.log(1 === o['string']);
```

版权所有：腾讯云计算（北京）有限责任公司

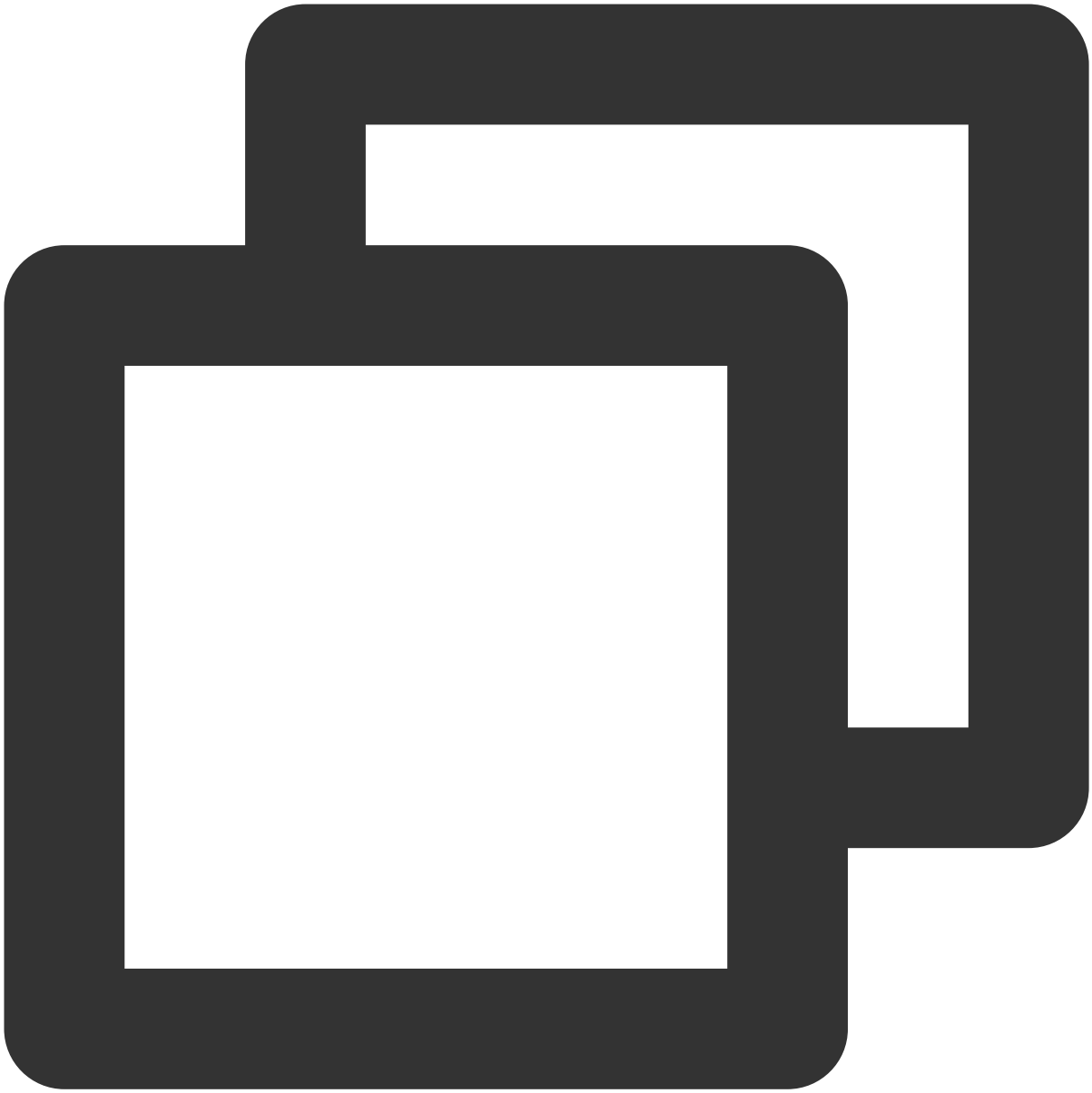
```
console.log(2 === o.const_var);

//对象属性的写操作
o['string']++;
o['string'] += 10;
o.const_var++;
o.const_var += 10;

//对象属性的读操作
console.log(12 === o['string']);
console.log(13 === o.const_var);
```

属性

`constructor` ：返回字符串 `"Object"` 。



```
console.log("Object" === {k:"1",v:"2"}.constructor)
```

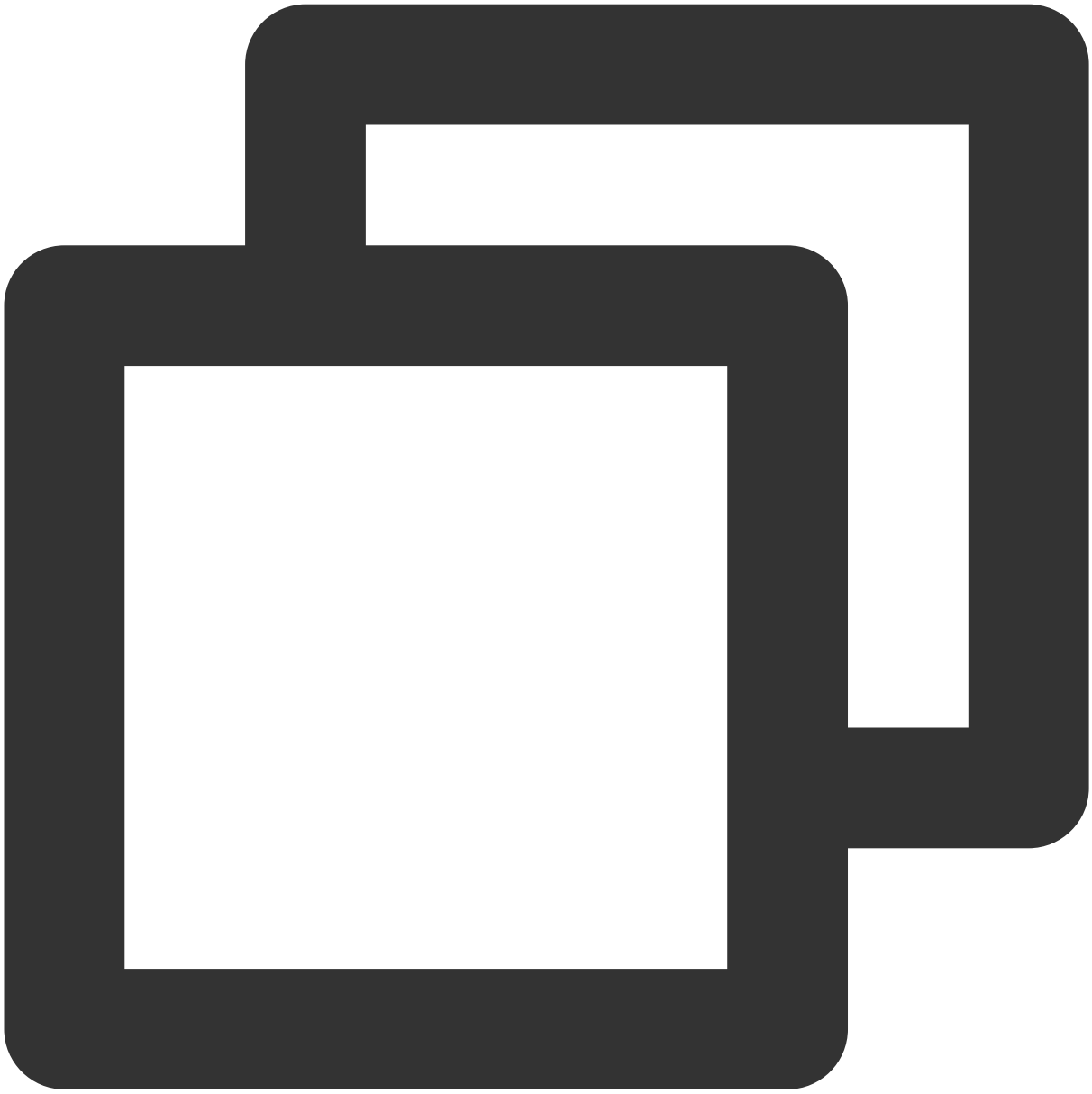
方法

`toString` ：返回字符串 `"[object Object]"` 。

function

语法

function 支持以下的定义方式：

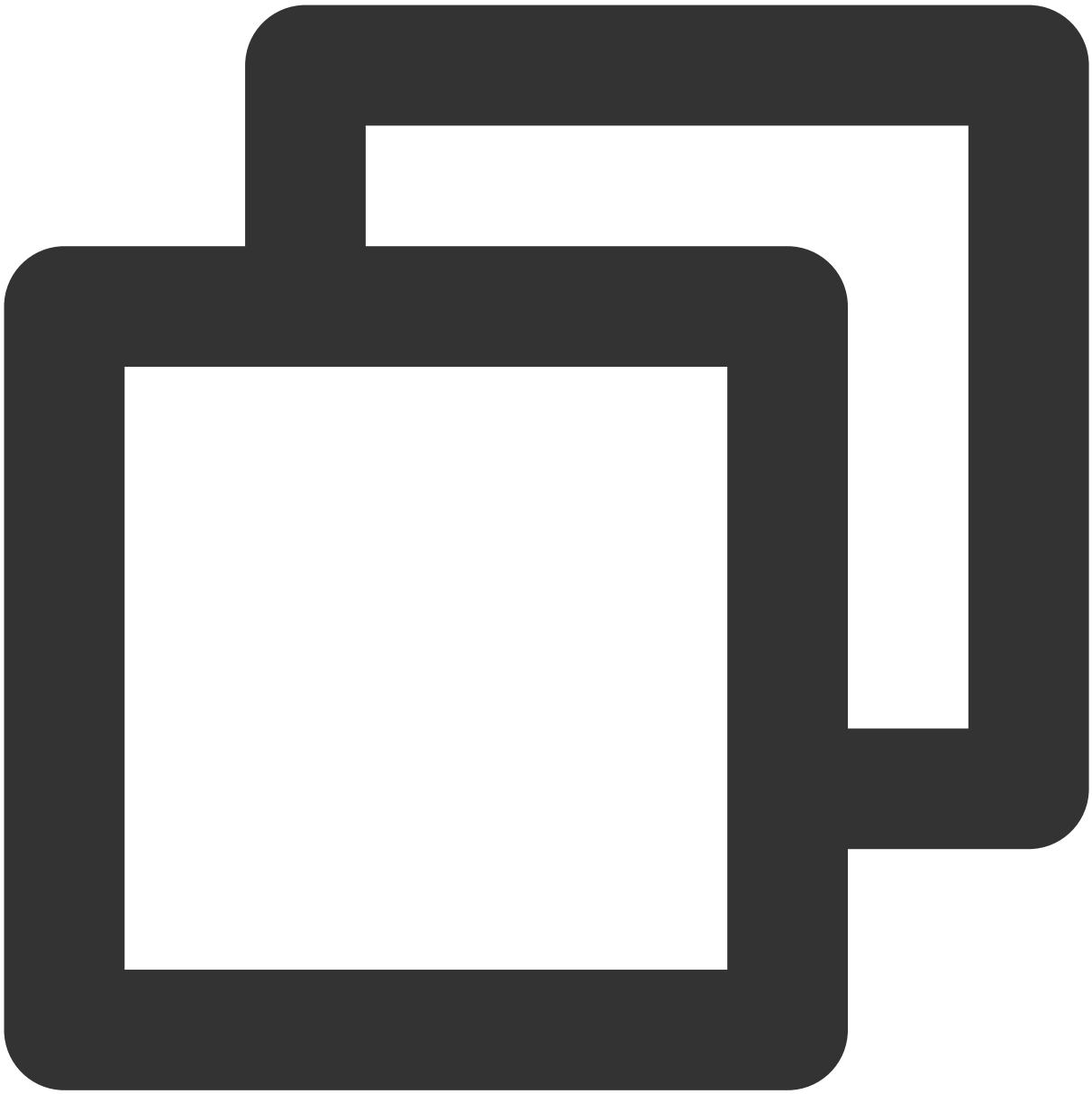


```
//方法 1
function a (x) {
  return x;
}

//方法 2
var b = function (x) {
  return x;
```

```
}
```

function 同时也支持以下的语法（匿名函数，闭包等）：



```
var a = function (x) {
  return function () { return x;}
}

var b = a(100);
console.log( 100 === b() );
```

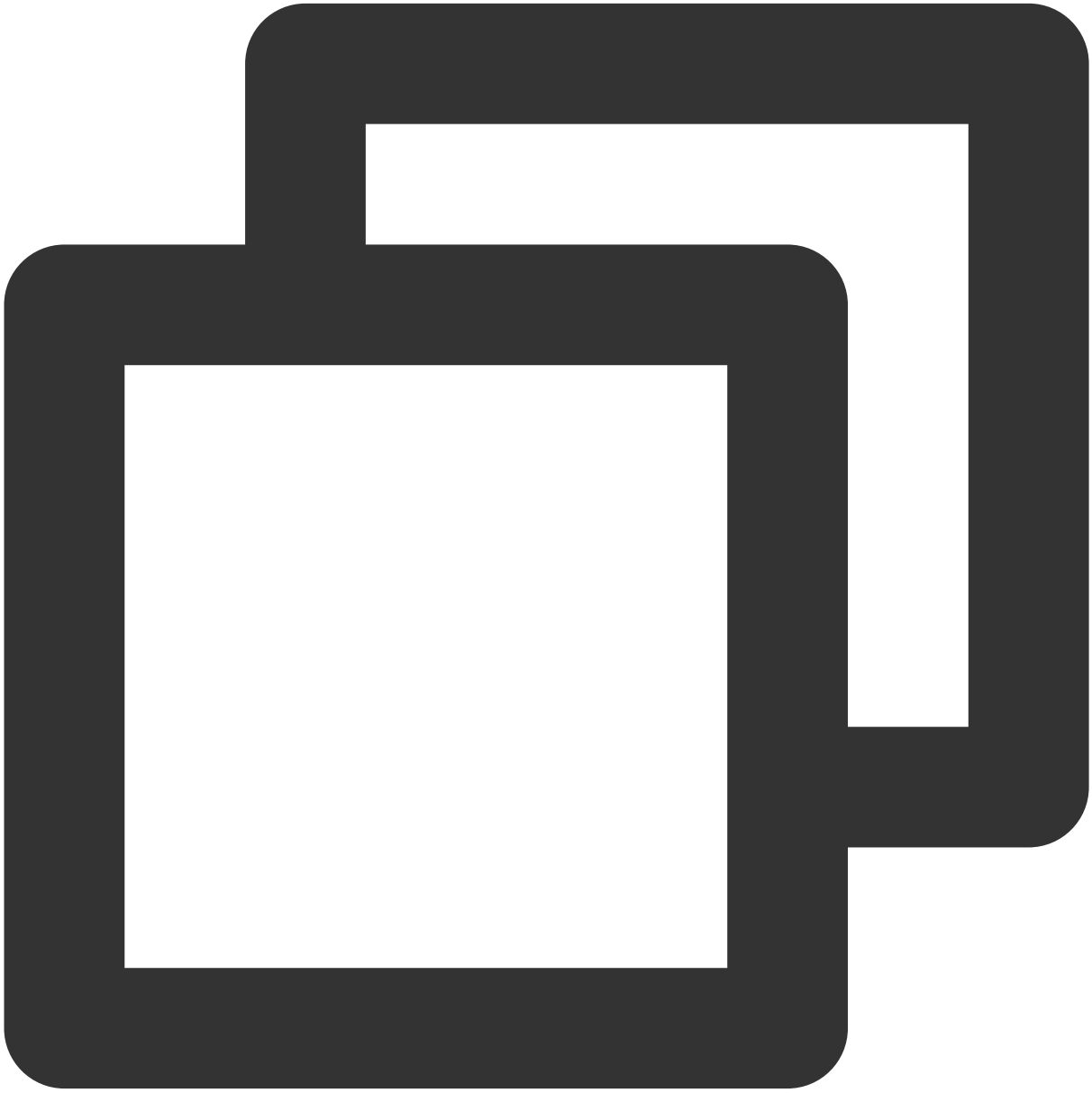
arguments

function 里面可以使用 `arguments` 关键词。该关键词目前只支持以下的属性：

`length`：传递给函数的参数个数；

`[index]`：通过 `index` 下标可以遍历传递给函数的每个参数。

示例代码：



```
var a = function(){
  console.log(3 === arguments.length);
  console.log(100 === arguments[0]);
  console.log(200 === arguments[1]);
```

```
  console.log(300 === arguments[2]);
};
a(100,200,300);
```

属性

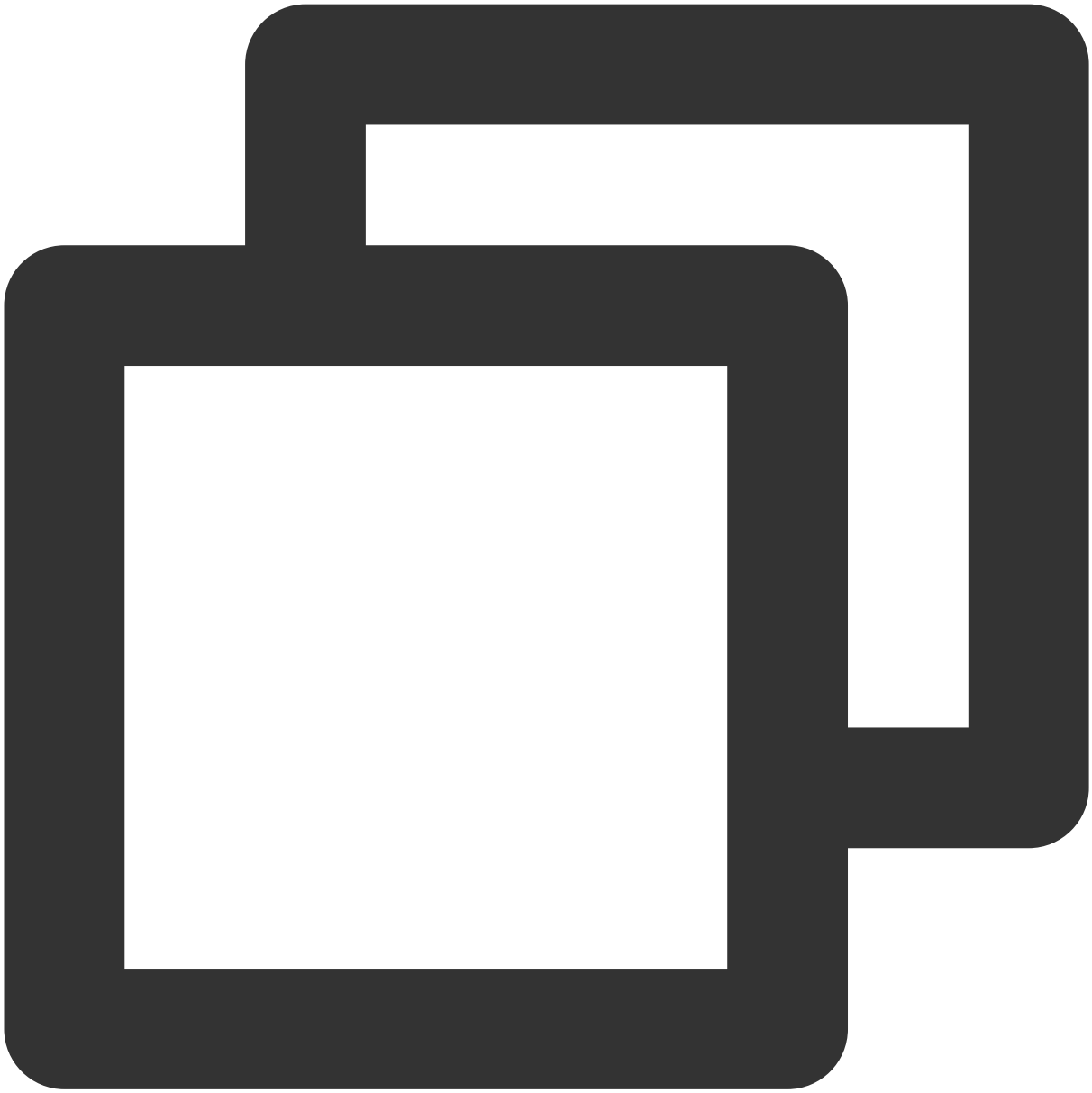
`constructor` ：返回字符串 `"Function"` ；

`length` ：返回函数的形参个数。

方法

`toString` ：返回字符串 `"[function Function]"` 。

示例代码:



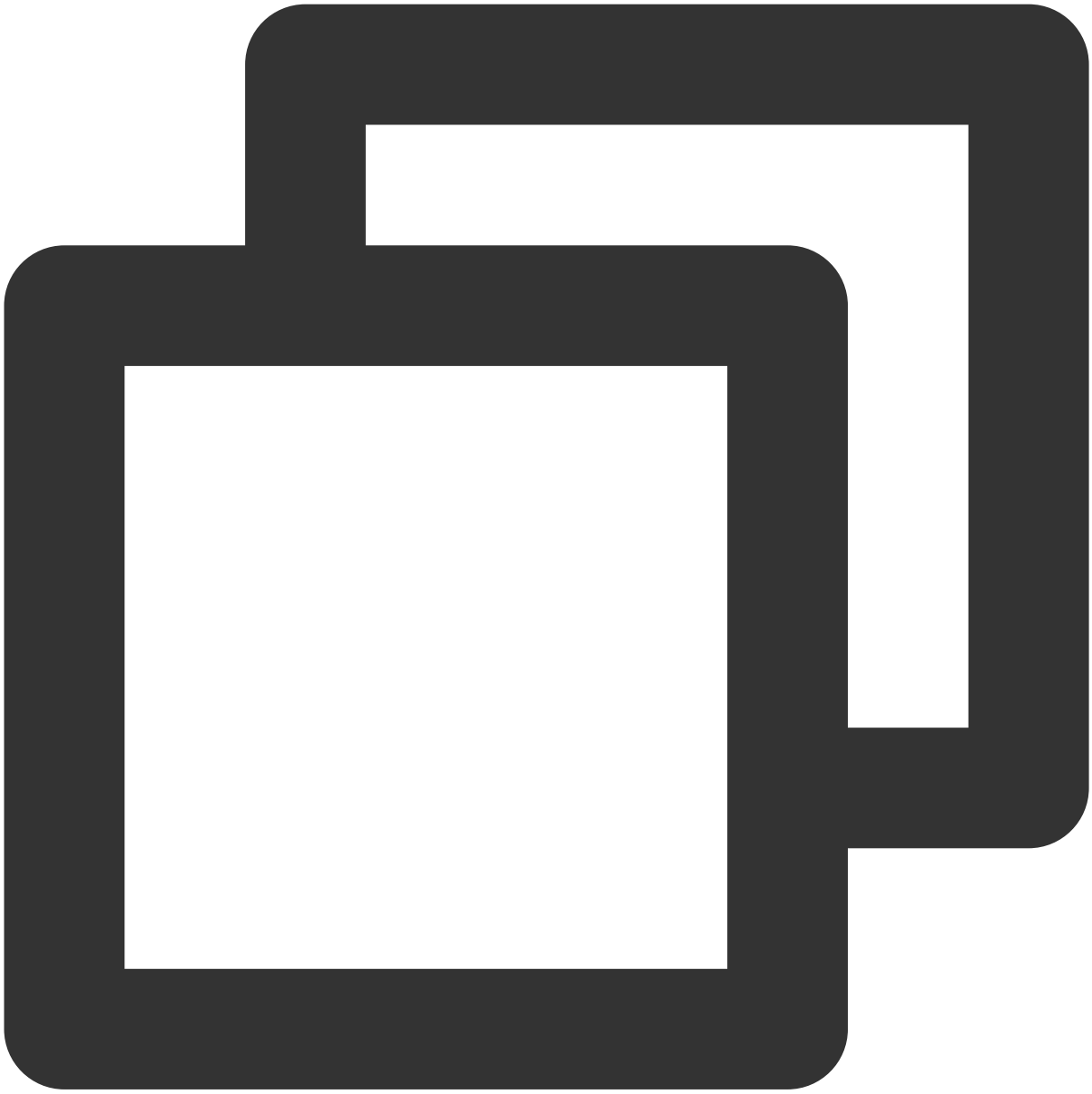
```
var func = function (a,b,c) { }

console.log("Function" === func.constructor);
console.log(3 === func.length);
console.log("[function Function]" === func.toString());
```

array

语法

array 支持以下的定义方式：



```
var a = [];       //生成一个新的空数组

a = [1,"2",{},function(){}];   //生成一个新的非空数组，数组元素可以是任何类型
```

属性

`constructor` ：返回字符串 `"Array"` ；

`length` 。

除constructor外属性的具体含义请参考 `ES5` 标准。

方法

`toString` ；
`concat` ；
`join` ；
`pop` ；
`push` ；
`reverse` ；
`shift` ；
`slice` ；
`sort` ；
`splice` ；
`unshift` ；
`indexOf` ；
`lastIndexOf` ；
`every` ；
`some` ；
`forEach` ；
`map` ；
`filter` ；
`reduce` ；
`reduceRight` 。

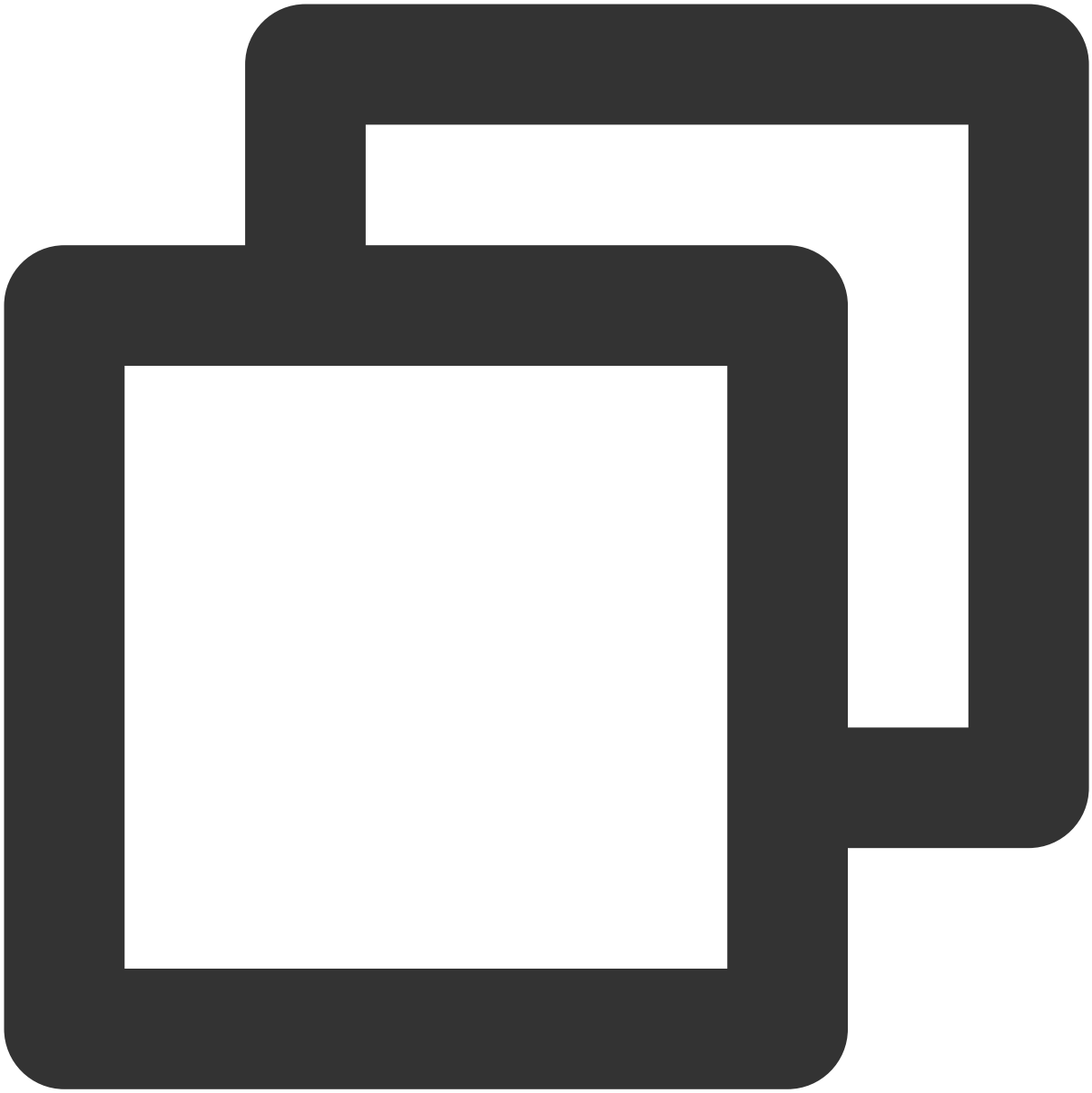
以上方法的具体使用请参考 `ES5` 标准。

date

语法

生成 date 对象需要使用 `getDate` 函数, 返回一个当前时间的对象。

版权所有：腾讯云计算（北京）有限责任公司



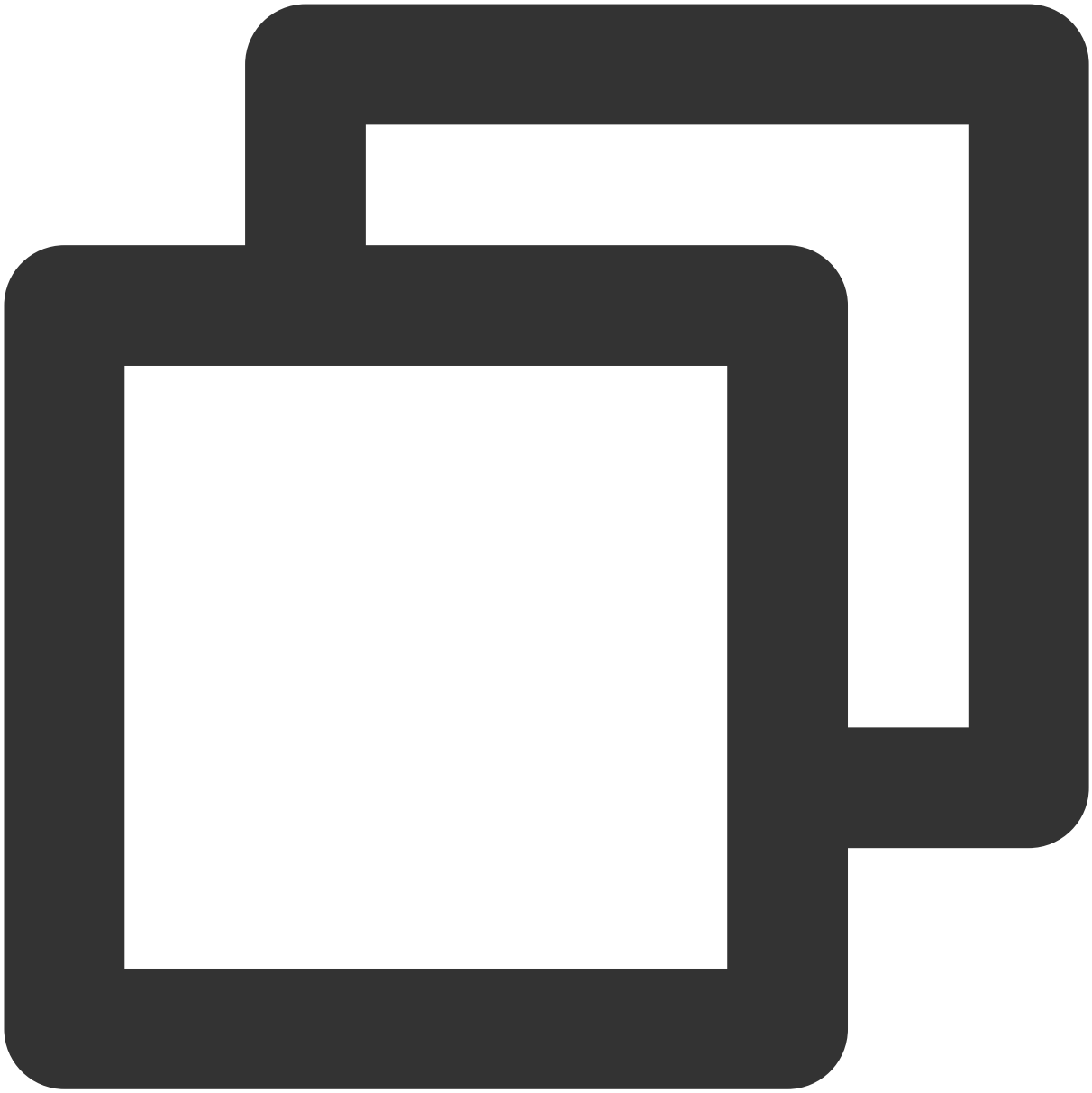
```
getDate()
getDate(milliseconds)
getDate(datestring)
getDate(year, month[, date[, hours[, minutes[, seconds[, milliseconds]]]]])
```

参数

`milliseconds`：从1970年1月1日00:00:00 UTC开始计算的毫秒数；

`datestring`：日期字符串，其格式为："month day, year hours:minutes:seconds"。

示例代码：



```
var date = getDate(); //返回当前时间对象

date = getDate(1500000000000);
// Fri Jul 14 2017 10:40:00 GMT+0800 (中国标准时间)
date = getDate('2017-7-14');
// Fri Jul 14 2017 00:00:00 GMT+0800 (中国标准时间)
date = getDate(2017, 6, 14, 10, 40, 0, 0);
// Fri Jul 14 2017 10:40:00 GMT+0800 (中国标准时间)
```

属性

版权所有：腾讯云计算（北京）有限责任公司

`constructor` ：返回字符串 `"Date"` 。

方法

`toString` ；
`toDateString` ；
`toTimeString` ；
`toLocaleString` ；
`toLocaleDateString` ；
`toLocaleTimeString` ；
`valueOf` ；
`getTime` ；
`getFullYear` ；
`getUTCFullYear` ；
`getMonth` ；
`getUTCMonth` ；
`getDate` ；
`getUTCDate` ；
`getDay` ；
`getUTCDay` ；
`getHours` ；
`getUTCHours` ；
`getMinutes` ；
`getUTCMinutes` ；
`getSeconds` ；
`getUTCSeconds` ；
`getMilliseconds` ；
`getUTCMilliseconds` ；
`getTimezoneOffset` ；
`setTime` ；
`setMilliseconds` ；
`setUTCMilliseconds` ；
`setSeconds` ；
`setUTCSeconds` ；
`setMinutes` ；
`setUTCMinutes` ；
`setHours` ；

版权所有：腾讯云计算（北京）有限责任公司

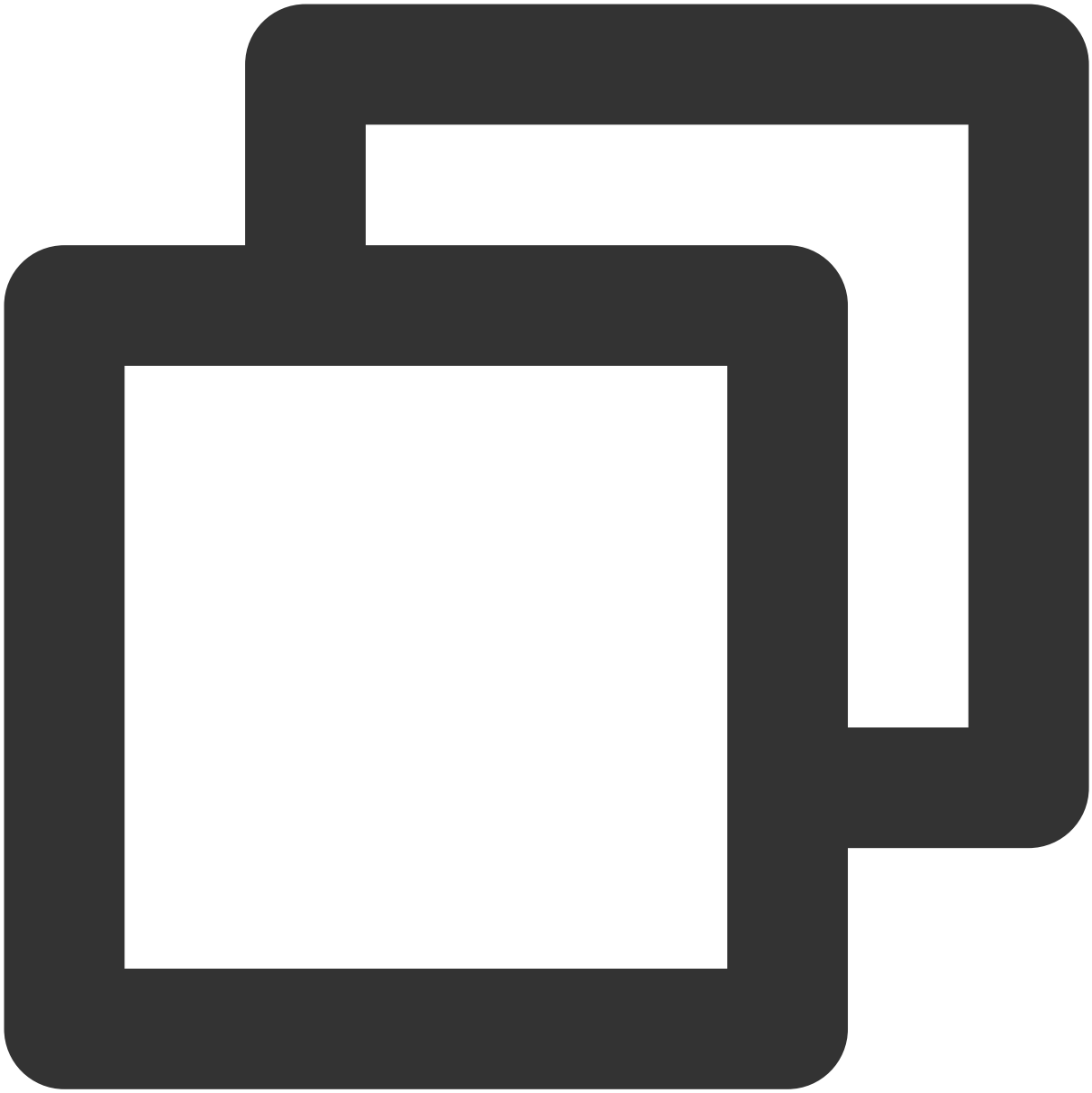
`setUTCHours` ；
`setDate` ；
`setUTCDate` ；
`setMonth` ；
`setUTCMonth` ；
`setFullYear` ；
`setUTCFullYear` ；
`toUTCString` ；
`toISOString` ；
`toJSON` 。

> 以上方法的具体使用请参考 `ES5` 标准。

regexp

语法

生成 regexp 对象需要使用 `getRegExp` 函数。



```
getRegExp(pattern[, flags])
```

参数：

`pattern`：正则表达式的内容；

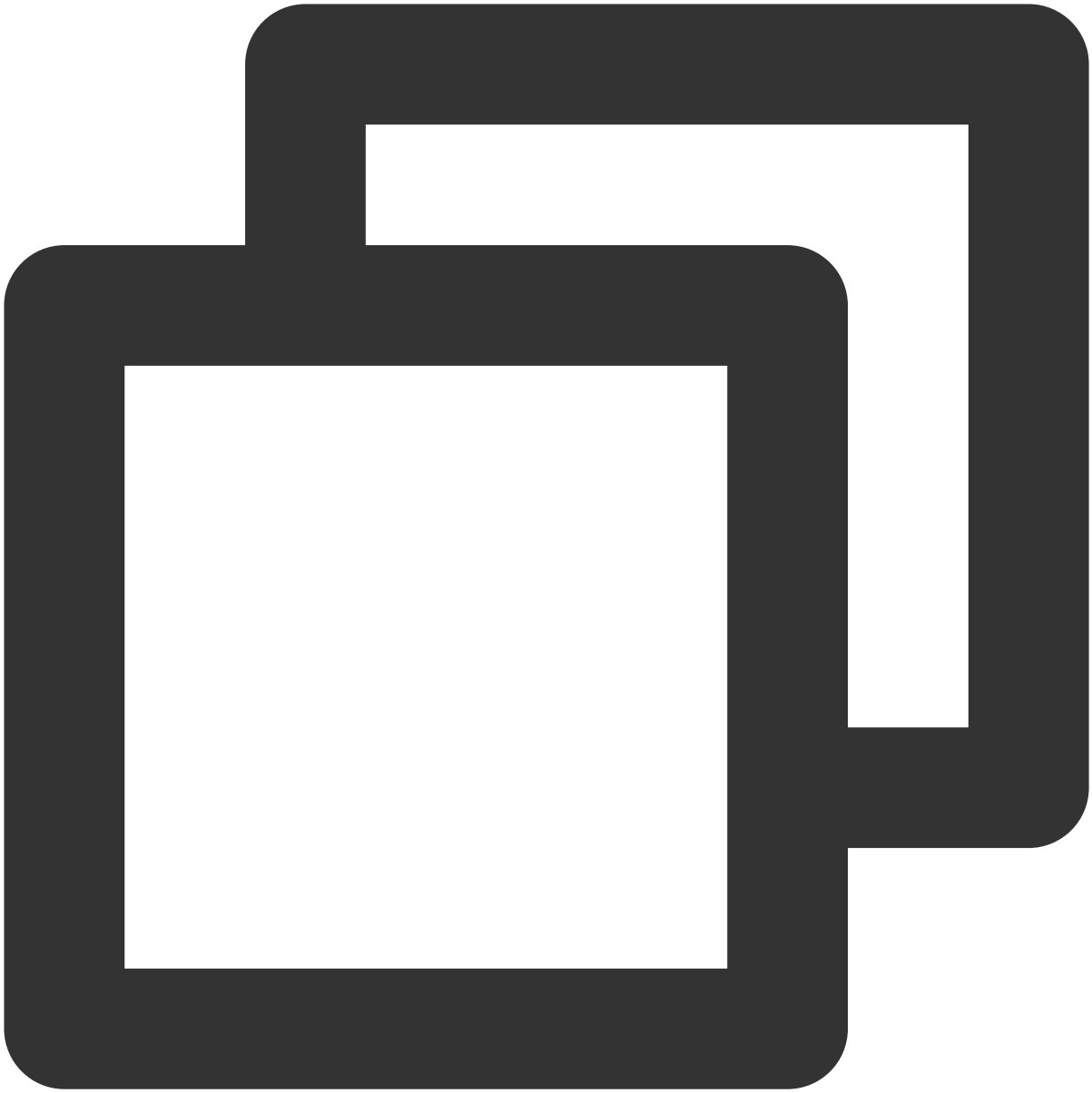
`flags`：修饰符。该字段只能包含以下字符:

`g`：global；

`i`：ignoreCase；

`m`：multiline。

示例代码：



```
var a = getRegExp("x", "img");
console.log("x" === a.source);
console.log(true === a.global);
console.log(true === a.ignoreCase);
console.log(true === a.multiline);
```

属性

`constructor` ：返回字符串 `"RegExp"` ；

`source` ；

`global` ；
`ignoreCase` ；
`multiline` ；
`lastIndex` 。

除constructor外属性的具体含义请参考 `ES5` 标准。

方法

`exec` ；
`test` ；
`toString` 。

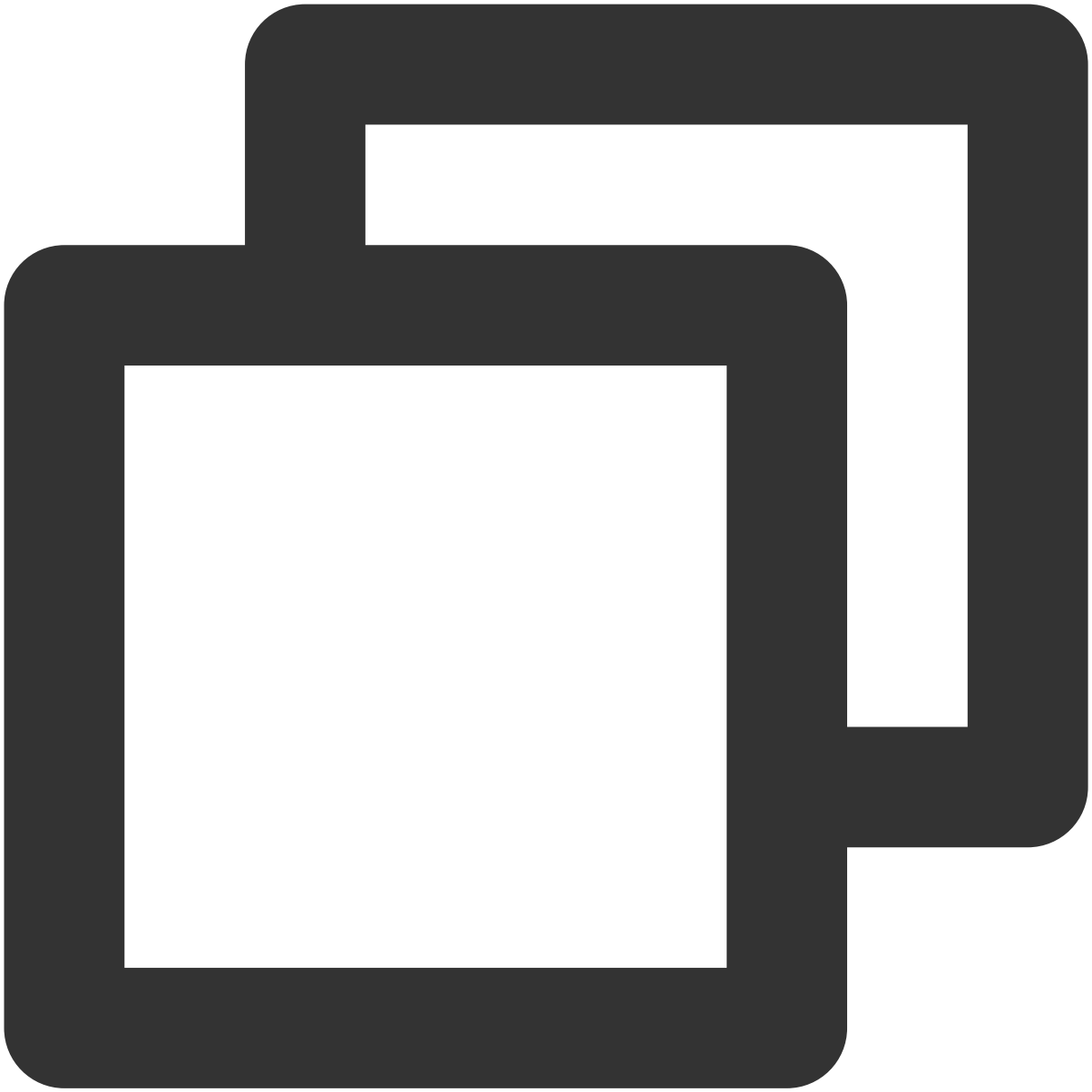
以上方法的具体使用请参考 `ES5` 标准。

数据类型判断

constructor 属性

数据类型的判断可以使用 `constructor` 属性。

示例代码：



```
var number = 10;
console.log( "Number" === number.constructor );

var string = "str";
console.log( "String" === string.constructor );

var boolean = true;
console.log( "Boolean" === boolean.constructor );

var object = {};
console.log( "Object" === object.constructor );
```

版权所有：腾讯云计算（北京）有限责任公司

```
var func = function(){};
console.log( "Function" === func.constructor );

var array = [];
console.log( "Array" === array.constructor );

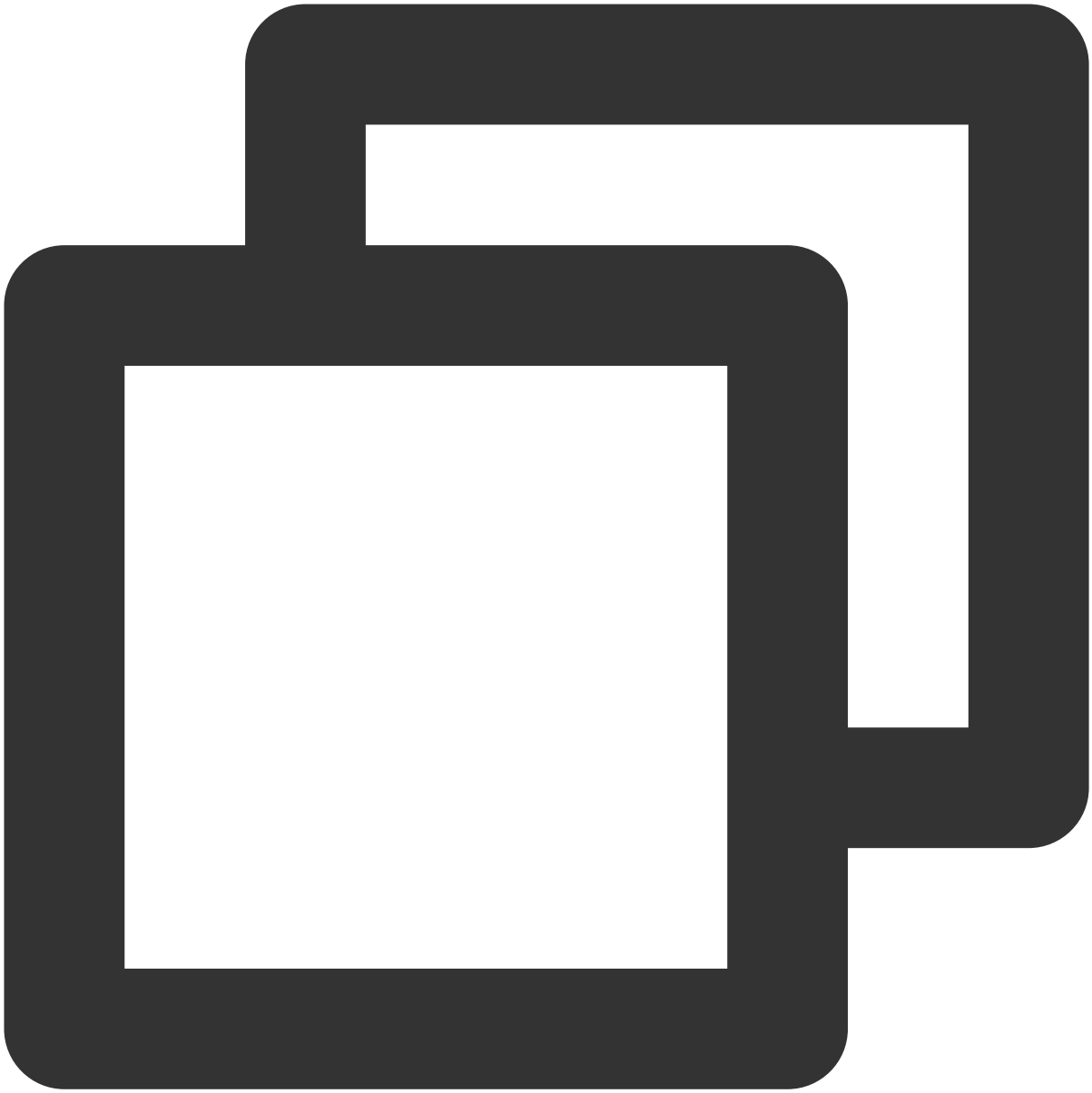
var date = getDate();
console.log( "Date" === date.constructor );

var regexp = getRegExp();
console.log( "RegExp" === regexp.constructor );
```

typeof

使用 `typeof` 也可以区分部分数据类型。

示例代码：



```
var number = 10;
var boolean = true;
var object = {};
var func = function(){};
var array = [];
var date = getDate();
var regexp = getRegExp();

console.log( 'number' === typeof number );
console.log( 'boolean' === typeof boolean );
console.log( 'object' === typeof object );
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log( 'function' === typeof func );
console.log( 'object' === typeof array );
console.log( 'object' === typeof date );
console.log( 'object' === typeof regexp );

console.log( 'undefined' === typeof undefined );
console.log( 'object' === typeof null );
```

版权所有：腾讯云计算（北京）有限责任公司

基础类库

最近更新时间：2024-07-12 16:53:05

1. console

`console.log` 方法用于在 console 窗口输出信息。它可以接受多个参数，将它们的结果连接起来输出。

2. Math

属性

`E`
`LN10`
`LN2`
`LOG2E`
`LOG10E`
`PI`
`SQRT1_2`
`SQRT2`

> 以上属性的具体使用请参考 `ES5` 标准。

方法

`abs`
`acos`
`asin`
`atan`
`atan2`
`ceil`
`cos`
`exp`
`floor`
`log`
`max`

版权所有：腾讯云计算（北京）有限责任公司

`min`
`pow`
`random`
`round`
`sin`
`sqrt`
`tan`

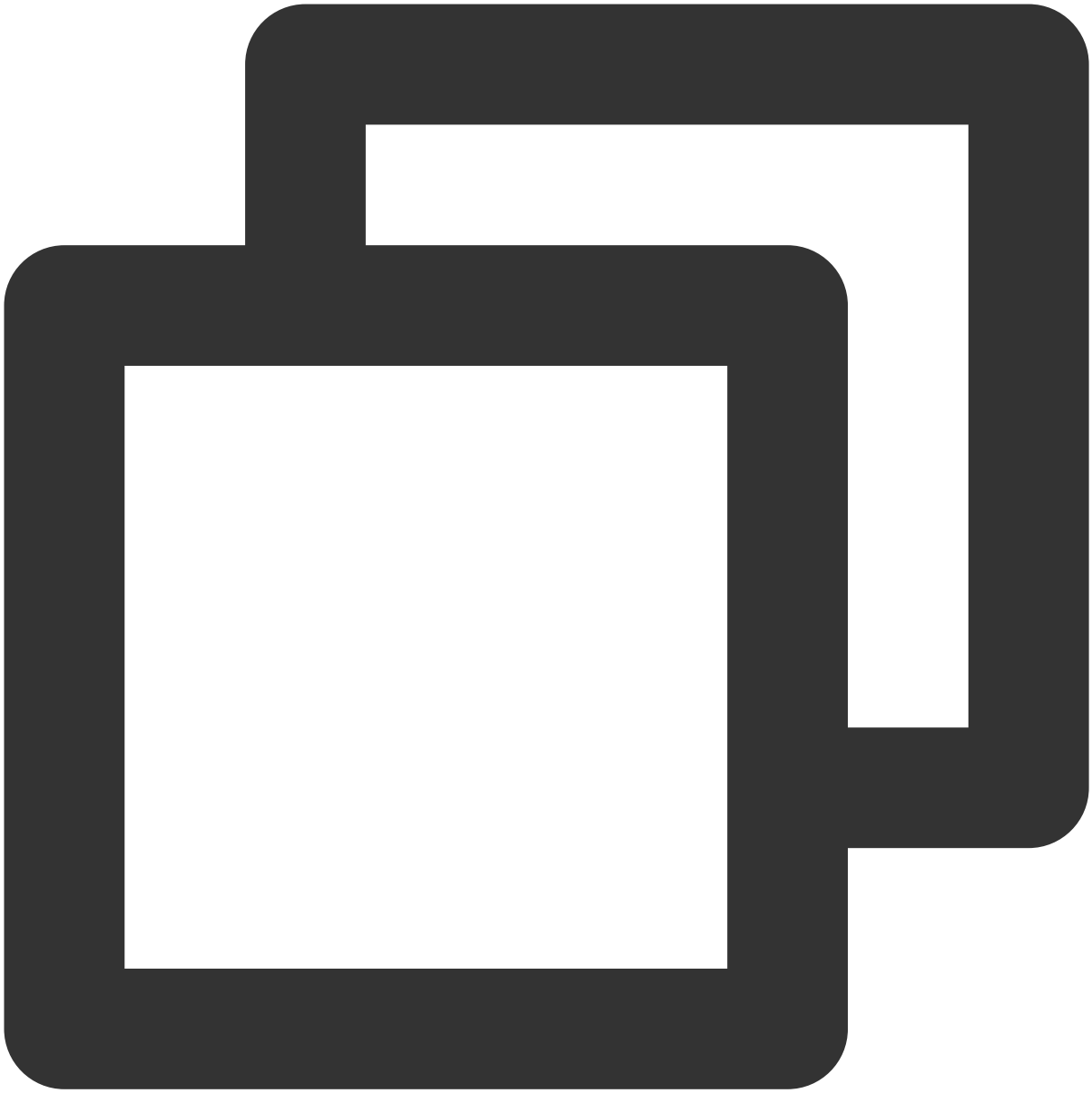
> 以上属性的具体使用请参考 `ES5` 标准。

3. JSON

方法

`stringify(object)`：将 `object` 对象转换为 `JSON` 字符串，并返回该字符串。
`parse(string)`：将 `JSON` 字符串转化成对象，并返回该对象。

示例代码：



```
console.log(undefined === JSON.stringify());
console.log(undefined === JSON.stringify(undefined));
console.log("null"===JSON.stringify(null));

console.log("111"===JSON.stringify(111));
console.log('"111"'===JSON.stringify("111"));
console.log("true"===JSON.stringify(true));
console.log(undefined===JSON.stringify(function(){}));


console.log(undefined===JSON.parse(JSON.stringify()));
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(undefined===JSON.parse(JSON.stringify(undefined)));
console.log(null===JSON.parse(JSON.stringify(null)));

console.log(111===JSON.parse(JSON.stringify(111)));
console.log("111"===JSON.parse(JSON.stringify("111")));
console.log(true===JSON.parse(JSON.stringify(true)));

console.log(undefined===JSON.parse(JSON.stringify(function(){})));
```

4. Number

属性

`MAX_VALUE`

`MIN_VALUE`

`NEGATIVE_INFINITY`

`POSITIVE_INFINITY`

> 以上属性的具体使用请参考 `ES5` 标准。

5. Date

属性

`parse`

`UTC`

now

> 以上属性的具体使用请参考 `ES5` 标准。

6. Global

属性

`NaN`

版权所有：腾讯云计算（北京）有限责任公司

`Infinity`

`undefined`

以上属性的具体使用请参考 `ES5` 标准。

方法

`parseInt`

`parseFloat`

`isNaN`

`isFinite`

`decodeURI`

`decodeURIComponent`

`encodeURI`

`encodeURIComponent`

以上方法的具体使用请参考 `ES5` 标准。

版权所有：腾讯云计算（北京）有限责任公司

基础能力

最近更新时间：2024-03-04 23:07:33

网络

说明：

在小程序中使用网络相关的 API 时，需要注意下列问题，请开发者提前了解。

服务器域名配置

每个小程序需要事先设置一个通讯域名，小程序只可以跟指定的域名与进行网络通信。包括普通 HTTPS 请求（wx.request）、上传文件（wx.uploadFile）。

配置流程

服务器域名请在小程序后台 > 设置 > 开发设置 > 服务器域名中进行配置，配置时需要注意：
域名只支持 https （wx.request）、（wx.uploadFile）协议。
域名不能使用 IP 地址或 localhost。
可以配置端口，如 https://miniApp.com:8080，但是配置后只能向 https://miniApp.com:8080 发起请求。如果向 https://miniApp.com、https://miniApp.com:9091 等 URL 请求则会失败。
如果不配置端口。如 https://miniApp.com，那么请求的 URL 中也不能包含端口，甚至是默认的 443 端口也不可以。如果向 https://miniApp.com:443 请求则会失败。

网络请求

超时时间

默认超时时间和最大超时时间都是60s。
超时时间可以在 app.json 中配置。

使用限制

网络请求的 referer header 不可设置。其格式固定为 https://appservice.wx.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为0表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本。
wx.request、wx.uploadFile 的最大并发限制是 10 个。
小程序进入后台运行后，如果5s内网络请求没有结束，会回调错误信息 fail interrupted；在回到前台之前，网络请求接口调用都会无法调用。

返回值编码

建议服务器返回值使用 **UTF-8** 编码。对于非 UTF-8 编码，小程序会尝试进行转换，但是会有转换失败的可能。

版权所有：腾讯云计算（北京）有限责任公司

小程序会自动对 BOM 头进行过滤（只过滤一个 BOM 头）。

回调函数

只要成功接收到服务器返回，无论 `statusCode` 是多少，都会进入 `success` 回调。请开发者根据业务逻辑对返回值进行判断。

常见问题

HTTPS 证书

小程序必须使用 HTTPS/WSS 发起网络请求。请求时系统会对服务器域名使用的 HTTPS 证书进行校验，如果校验失败，则请求不能成功发起。由于系统限制，不同平台对于证书要求的严格程度不同。为了保证小程序的兼容性，建议开发者按照最高标准进行证书配置，并使用相关工具检查现有证书是否符合要求。
对证书要求如下：
HTTPS 证书必须有效；
证书必须被系统信任，即根证书被已系统内置。
部署 SSL 证书的网站域名必须与证书颁发的域名一致。
证书必须在有效期内。
证书的信任链必需完整（需要服务器配置）。
`iOS` 不支持自签名证书。
`iOS` 下证书必须满足苹果 App Transport Security (ATS) 的要求。
TLS 必须支持 1.2 及以上版本。部分旧 `Android` 机型还未支持 TLS 1.2，请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本。
部分 CA 可能不被操作系统信任，请开发者在选择证书时注意小程序和各系统的相关通告。
说明：
证书有效性可以使用 `openssl s_client -connect example.com:443` 命令验证，也可以使用其他 在线工具。
除了网络请求 API 外，小程序中其他 HTTPS 请求如果出现异常，也请按上述流程进行检查。如 https 的图片无法加载、音视频无法播放等。

跳过域名校验

在开发者工具中，可以临时开启 `开发环境不校验请求域名` 、 `TLS版本及 HTTPS 证书` 选项，跳过服务器域名的校验。此时，在开发者工具中及手机开启调试模式时，不会进行服务器域名的校验。
在服务器域名配置成功后，建议开发者关闭此选项进行开发，并在各平台下进行测试，以确认服务器域名配置正确。
说明：
如果手机上出现 “打开调试模式可以发出请求，关闭调试模式无法发出请求” 的现象，请确认是否跳过了域名校验，并确认服务器域名和证书配置是否正确。

版权所有：腾讯云计算（北京）有限责任公司

如遇到“服务器内部错误”，请尝试以下办法自查 ①更换浏览器 ②刷新网页 ③排查文件是否放在正确的服务器根目录下。 排查办法：按照以下规则拼接网址，并且自行尝试访问是否成功：https://m.da9c.cn/文件名。

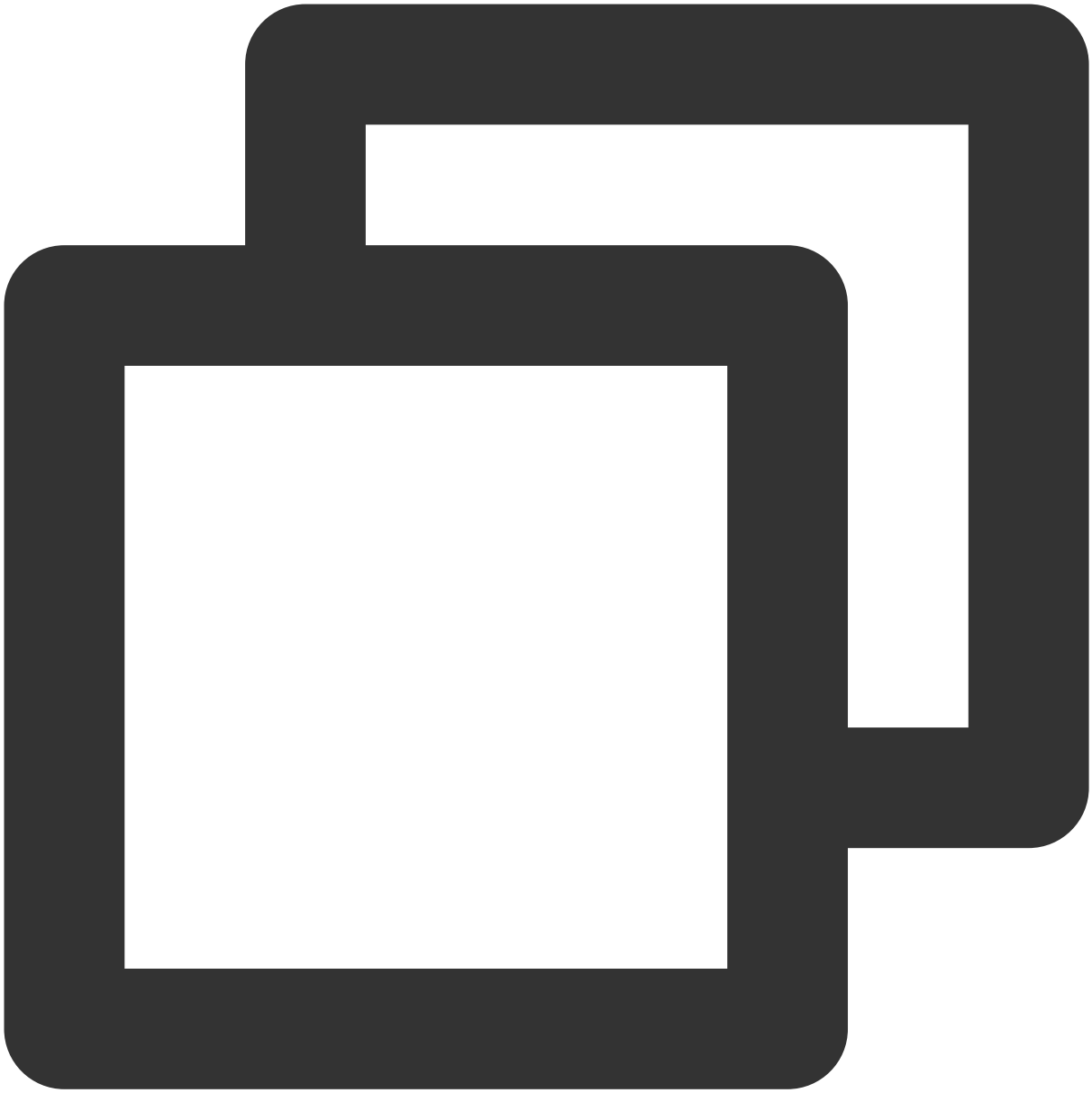
存储

每个小程序都可以有自己的本地缓存，可以通过 wx.setStorage/qq.setStorageSync、wx.getStorage/wx.getStorageSync、qq.clearStorage/wx.clearStorageSync，wx.removeStorage/wx.removeStorageSync 对本地缓存进行读写和清理。
同一个用户，同一个小程序 storage 上限为 10MB。storage 以用户维度隔离，同一台设备上，A 用户无法读取到 B 用户的数据。
说明：
如果用户储存空间不足，应用会清空最近最久未使用的小程序的本地缓存，因此并不建议将关键信息全部存在 storage，以防储存空间不足或用户换设备的情况。

Canvas画布

所有在 <canvas> 中的画图必须用 JavaScript 完成：
WXML：在接下来的示例中如无特殊声明都会用这个 WXML 为模板，不再重复。

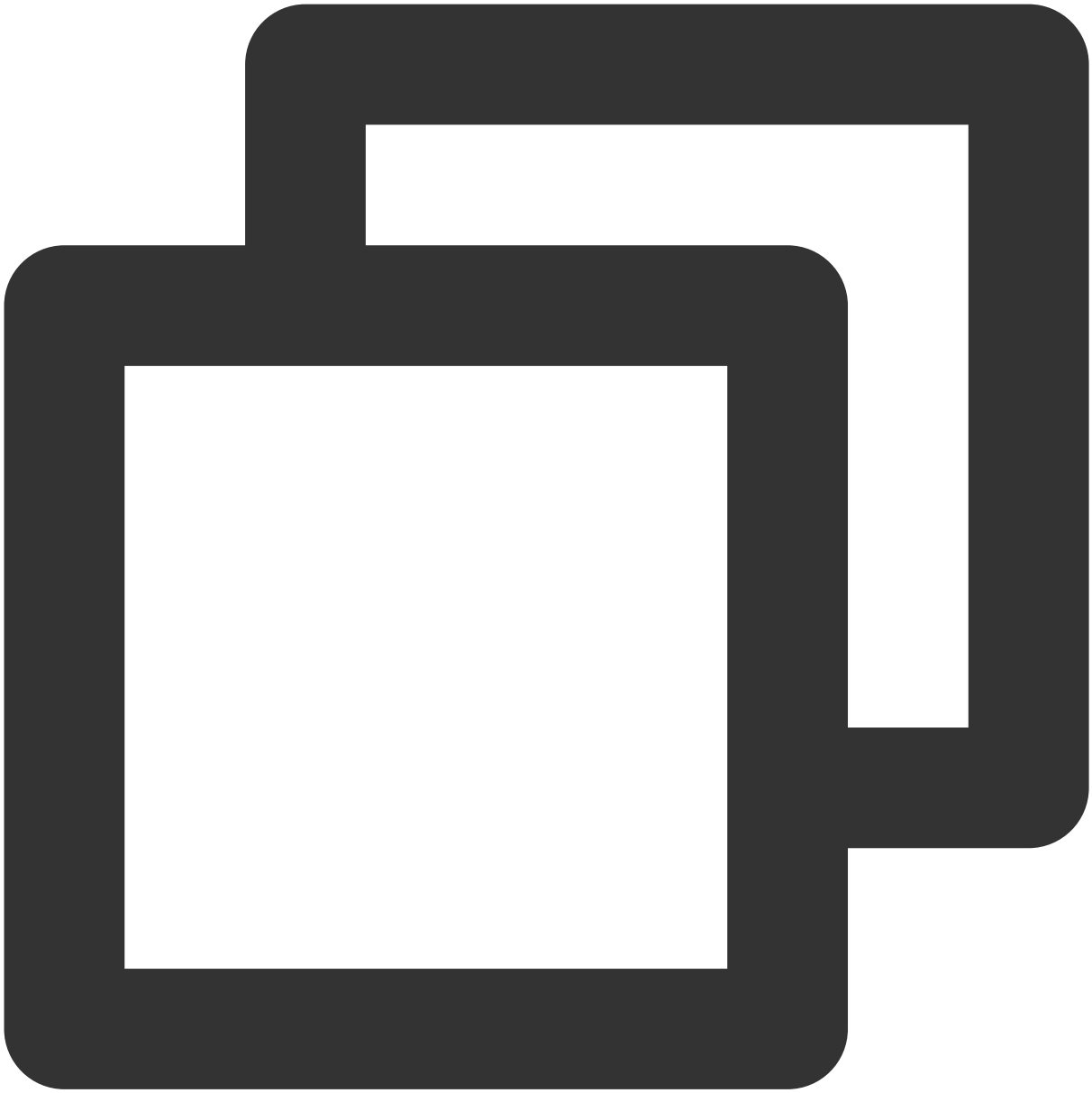
版权所有：腾讯云计算（北京）有限责任公司



```
<canvas canvas-id="myCanvas" style="border: 1px solid;" />
```

JS：我们在接下来的例子中会将 JS 放在 onLoad 中

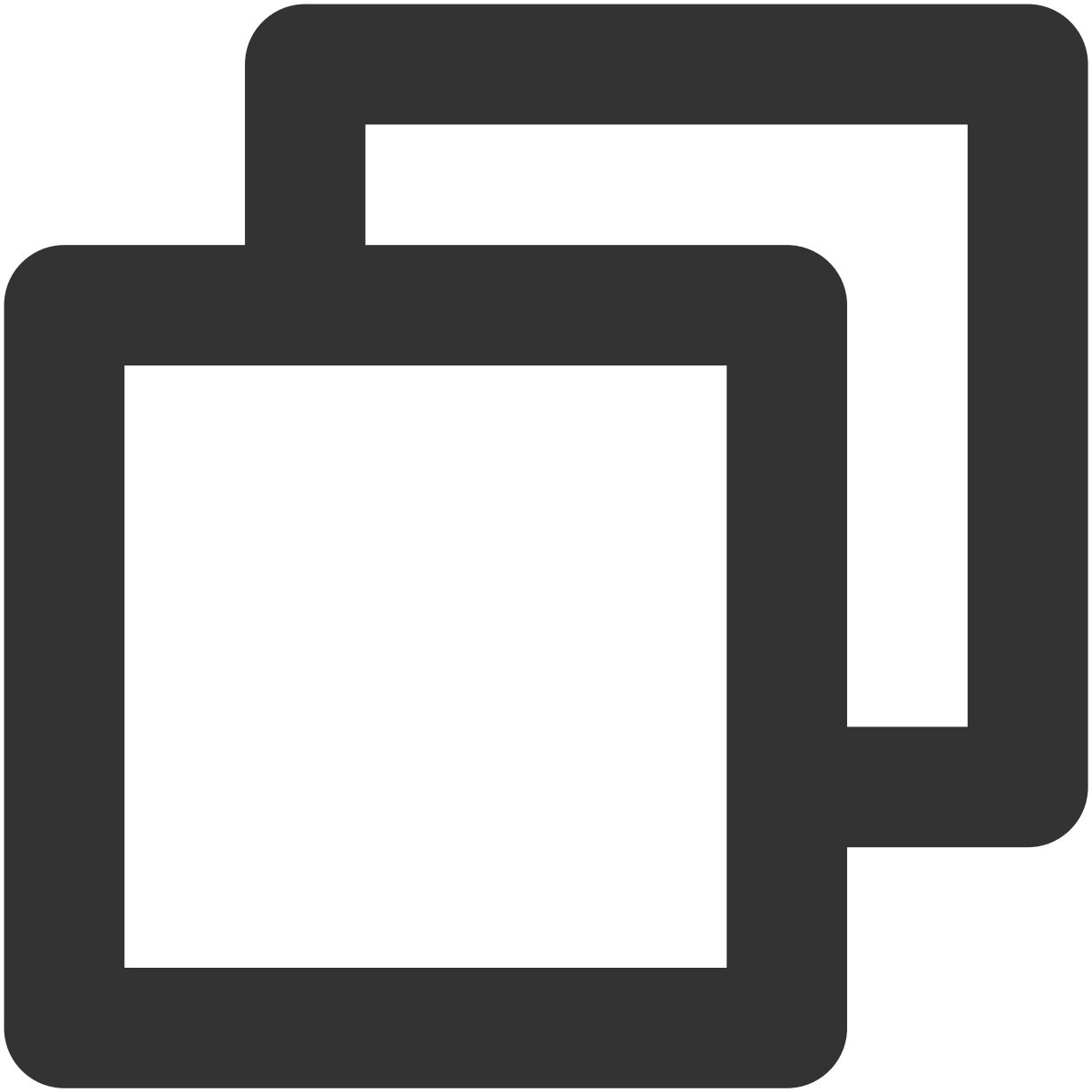
版权所有：腾讯云计算（北京）有限责任公司



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

第一步：创建一个 Canvas 绘图上下文

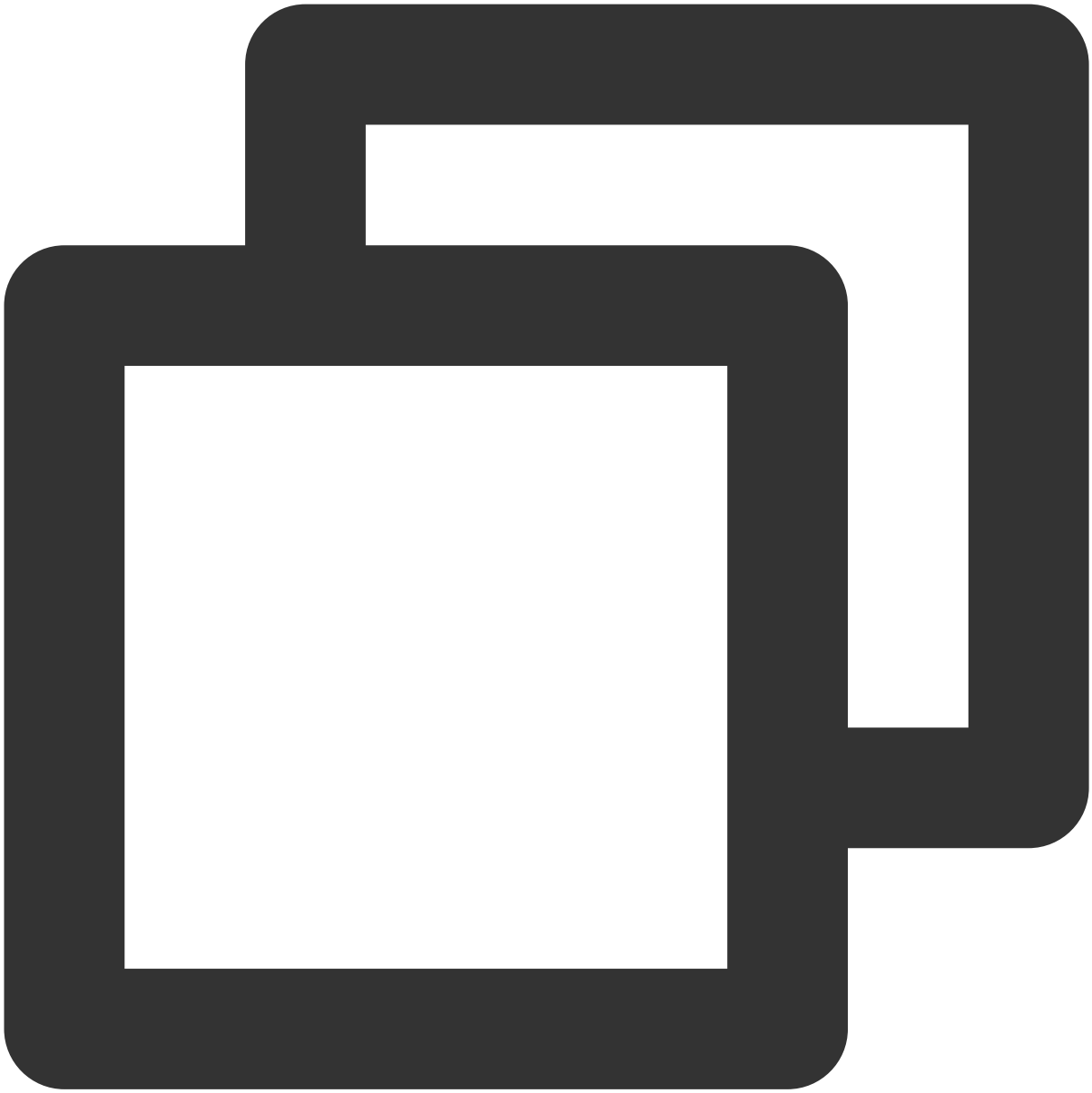
首先，我们需要创建一个 Canvas 绘图上下文 CanvasContext。CanvasContext 是小程序内建的一个对象，有一些绘图的方法：



```
const ctx = wx.createCanvasContext('myCanvas')
```

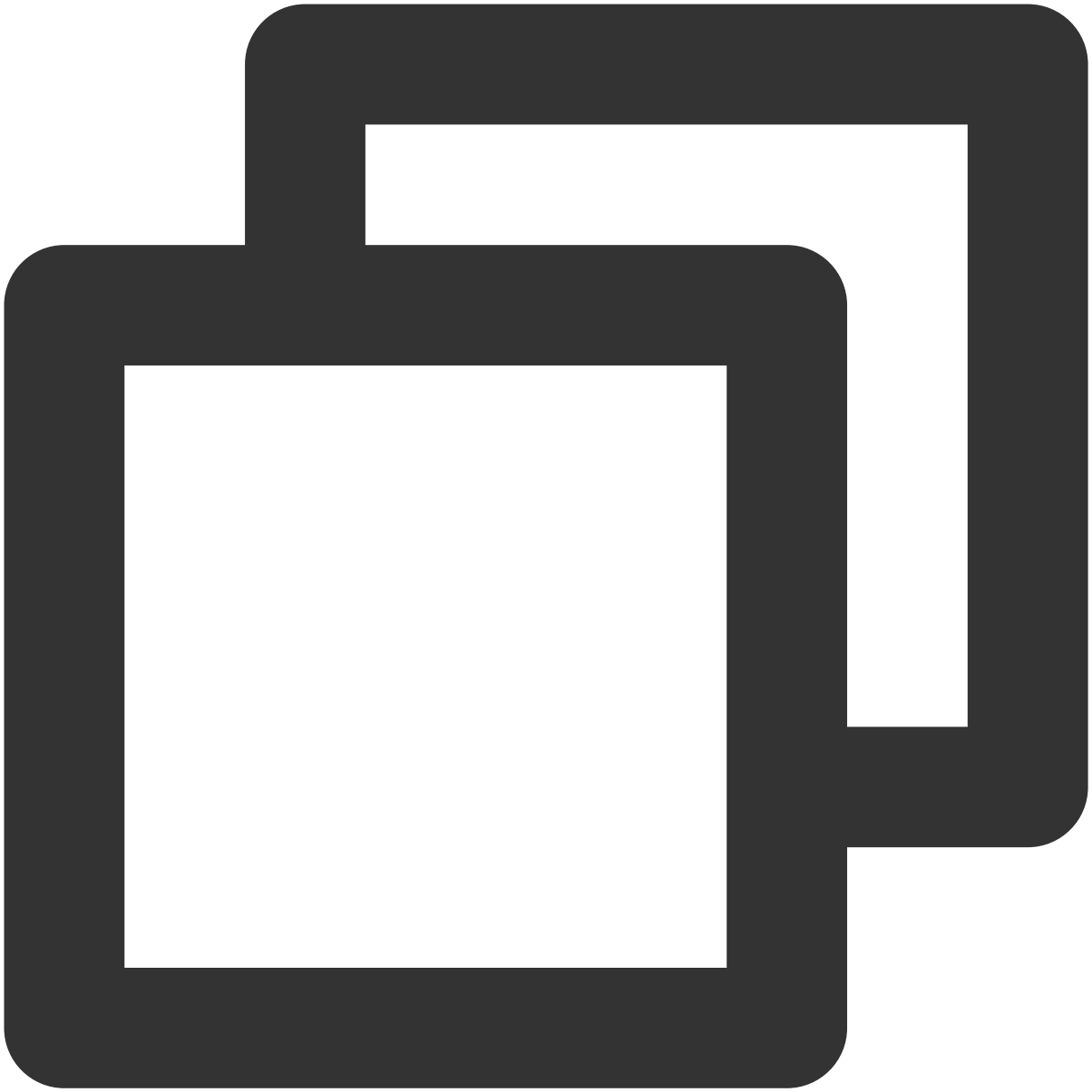
第二步：使用 Canvas 绘图上下文进行绘图描述

接着，我们来描述要在 Canvas 中绘制什么内容。设置绘图上下文的填充色为红色：



```
ctx.setFillStyle('red')
```

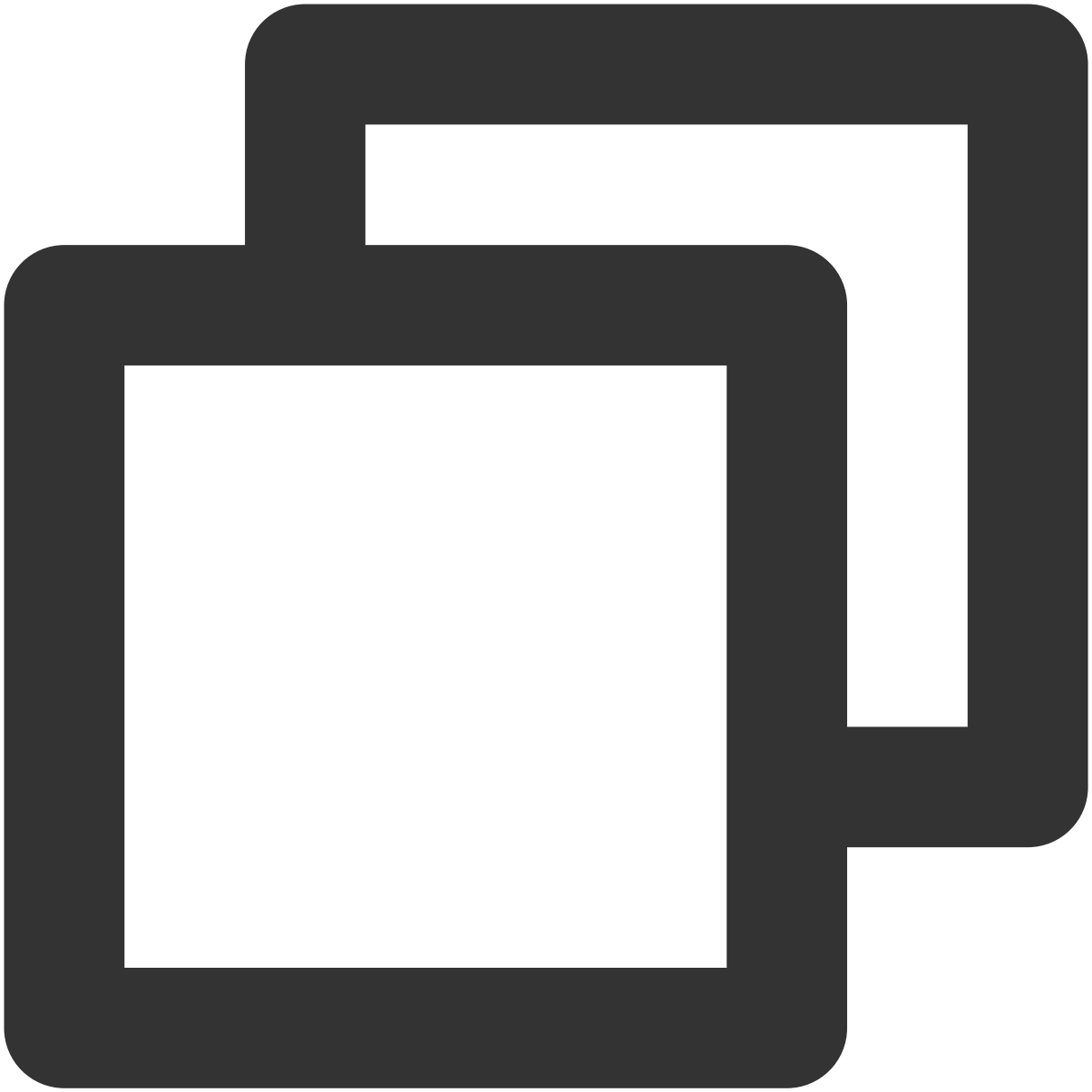
用 `fillRect(x, y, width, height)` 方法画一个矩形，填充为刚刚设置的红色：



```
ctx.fillRect(10, 10, 150, 75)
```

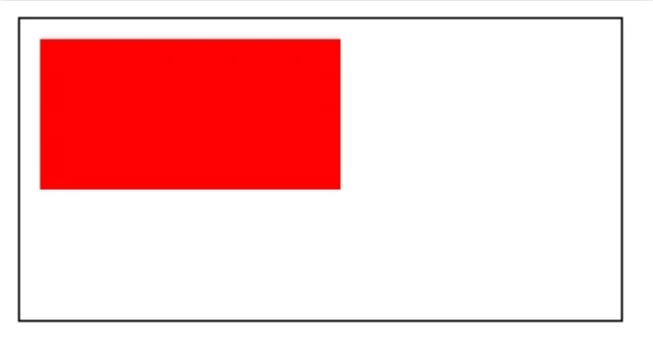
第三步：画图

告诉 <canvas> 组件，您要将刚刚的描述绘制上去：



```
ctx.draw()
```

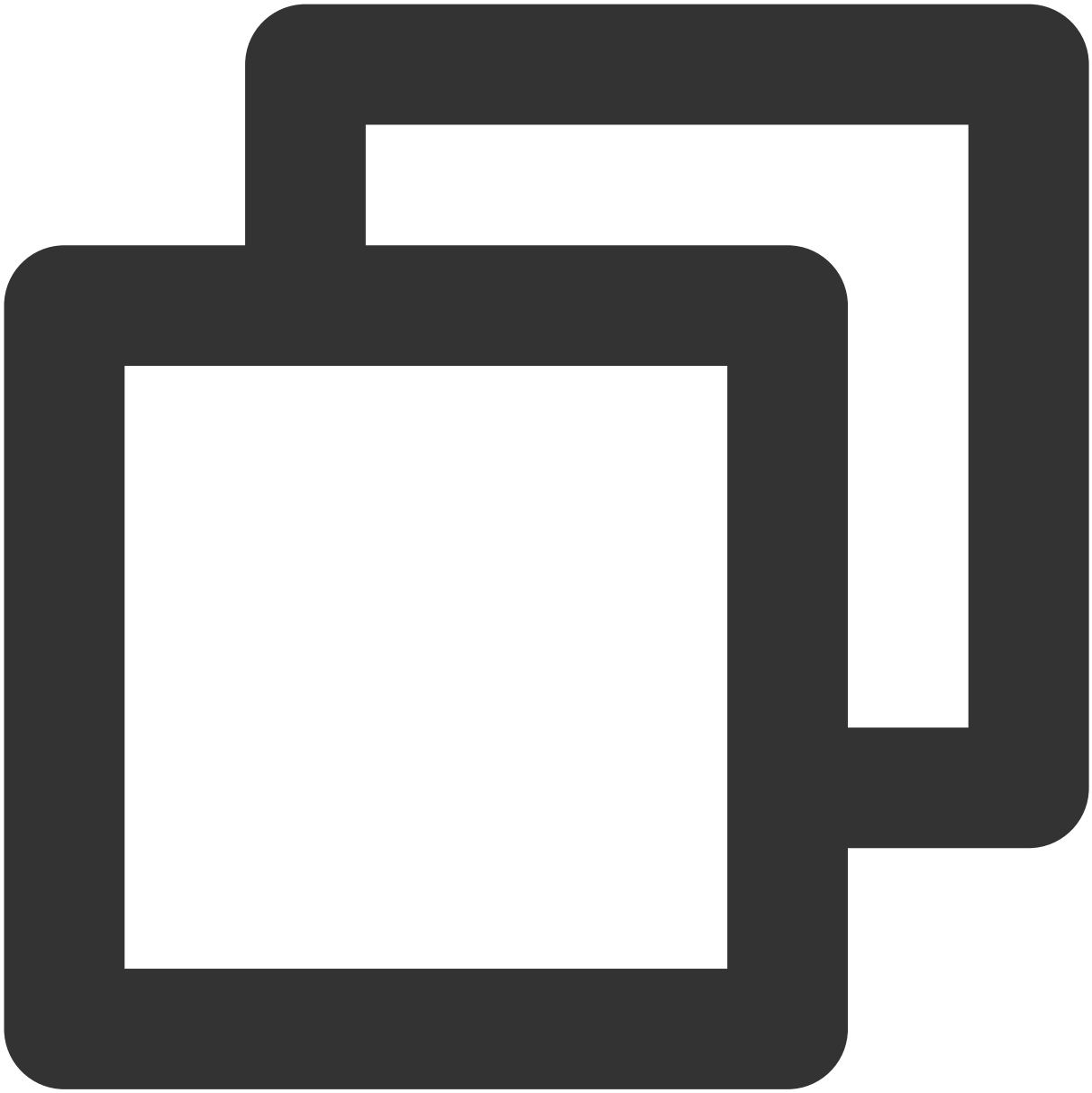
版权所有：腾讯云计算（北京）有限责任公司



坐标系

可以在 <canvas> 中加上一些事件，来观测它的坐标系：

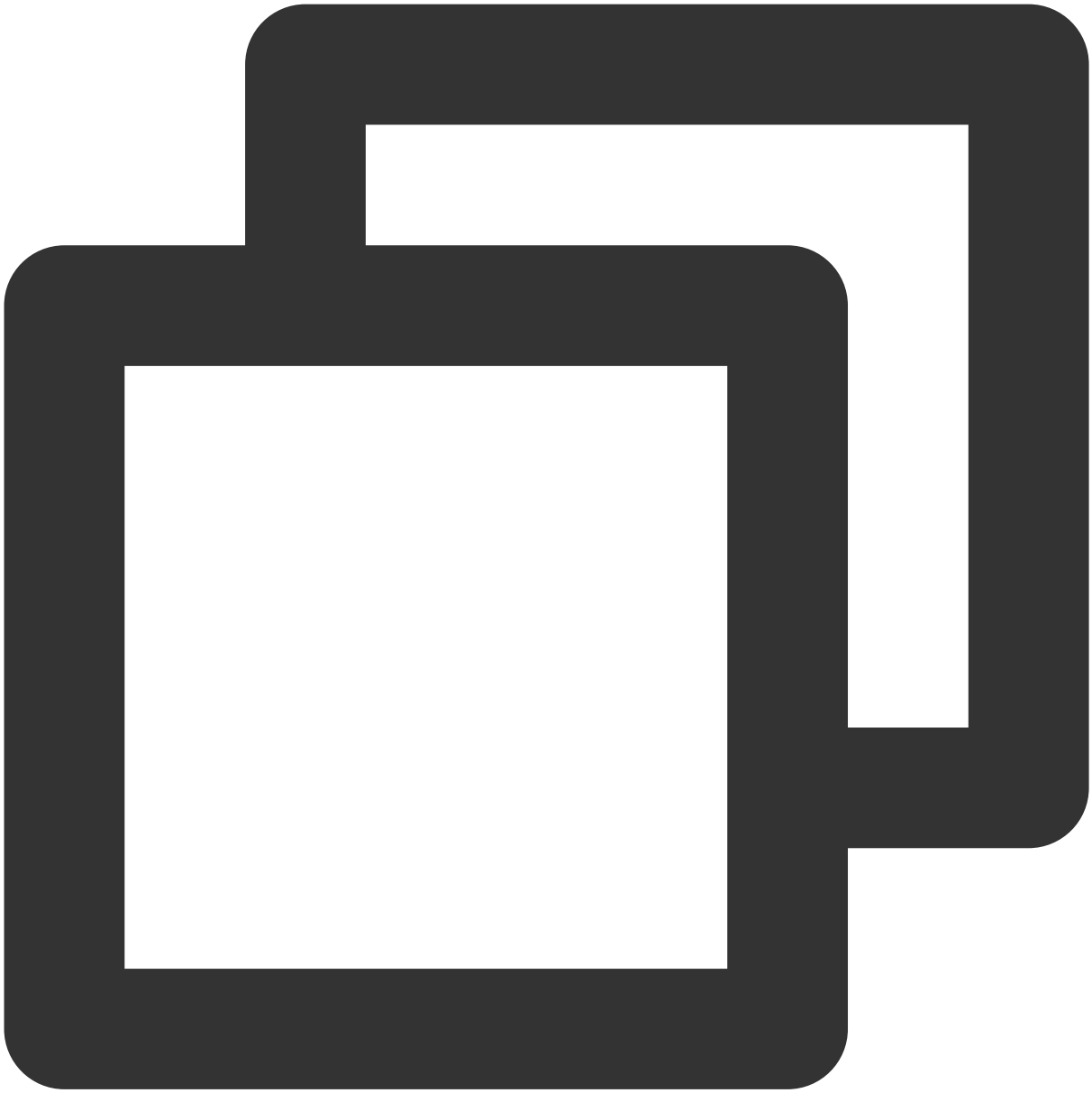
版权所有：腾讯云计算（北京）有限责任公司



```
<canvas
  canvas-id="myCanvas"
  style="margin: 5px; border:1px solid #d3d3d3;"
  bindtouchstart="start"
  bindtouchmove="move"
  bindtouchend="end"
/>

<view hidden="{{hidden}}">
  Coordinates: ({{x}}, {{y}})
</view>
```

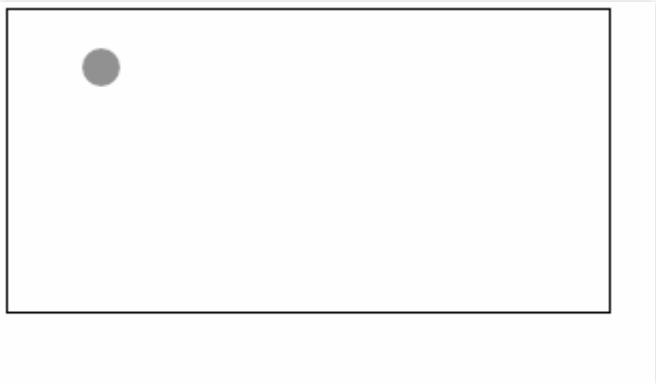
版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    x: 0,
    y: 0,
    hidden: true
  },
  start(e) {
    this.setData({
      hidden: false,
      x: e.touches[0].x,
```

```
        y: e.touches[0].y
      })
    },
    move(e) {
      this.setData({
        x: e.touches[0].x,
        y: e.touches[0].y
      })
    },
    end(e) {
      this.setData({
        hidden: true
      })
    }
  })
```

当您把手指放到 canvas 中，就会在下边显示出触碰点的坐标：



渐变

渐变能用于填充一个矩形，圆，线，文字等。填充色可以不固定位固定的一种颜色。

我们提供了两种颜色渐变的方式：

`createLinearGradient(x, y, x1, y1)` 创建一个线性的渐变。

`createCircularGradient(x, y, r)` 创建一个从圆心开始的渐变。

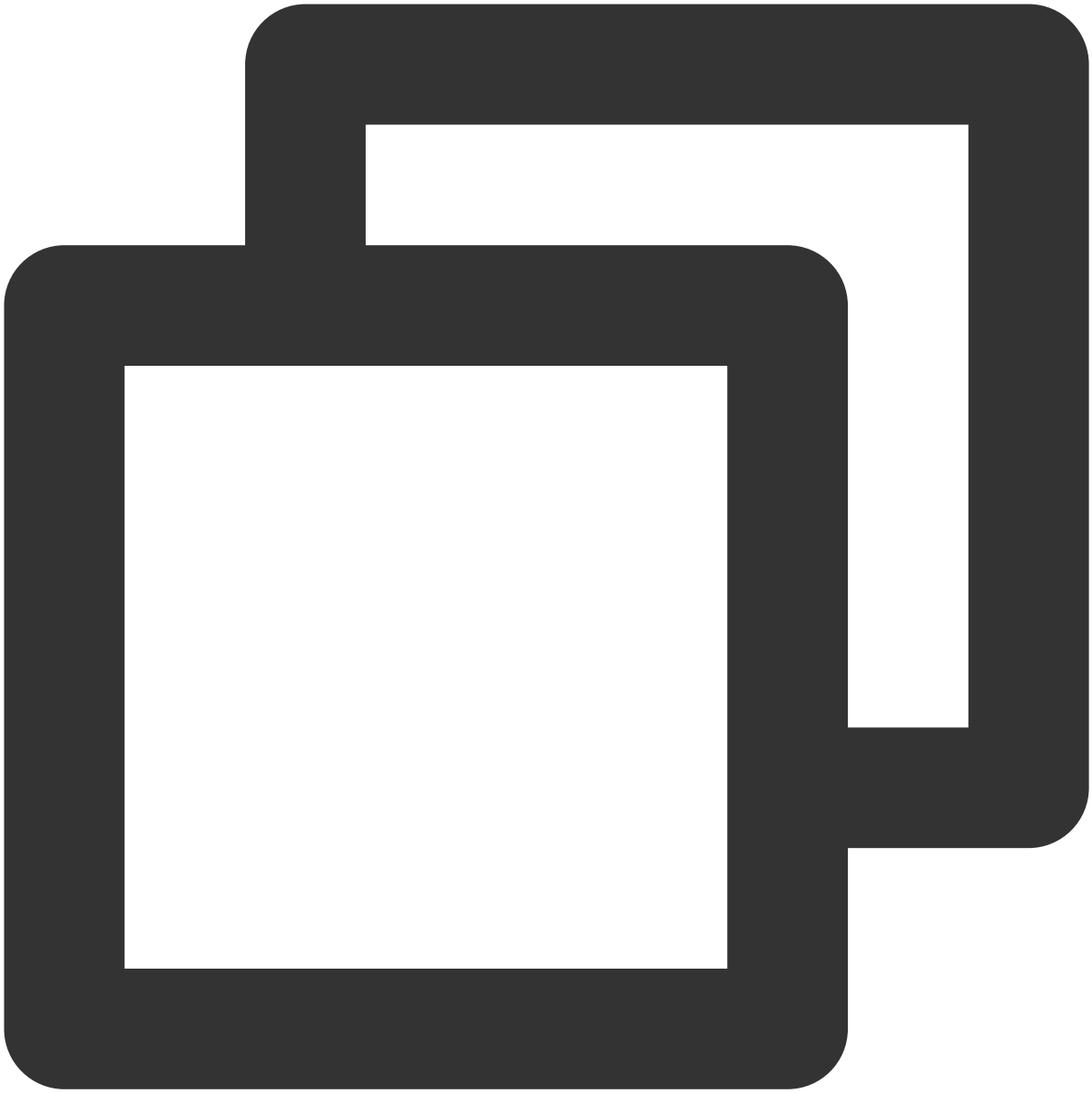
创建了一个渐变对象，就必须添加两个颜色渐变点。

`addColorStop(position, color)` 方法用于指定颜色渐变点的位置和颜色，位置必须位于0到1之间。

可以用 setFillStyle 和 setStrokeStyle 方法设置渐变，然后进行画图描述。

使用 `createLinearGradient()`

版权所有：腾讯云计算（北京）有限责任公司

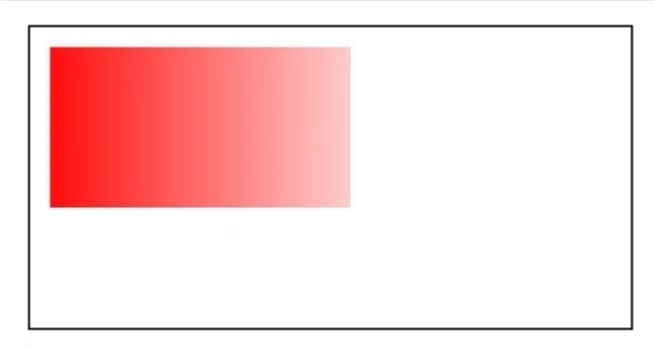


```
const ctx = wx.createCanvasContext('myCanvas')

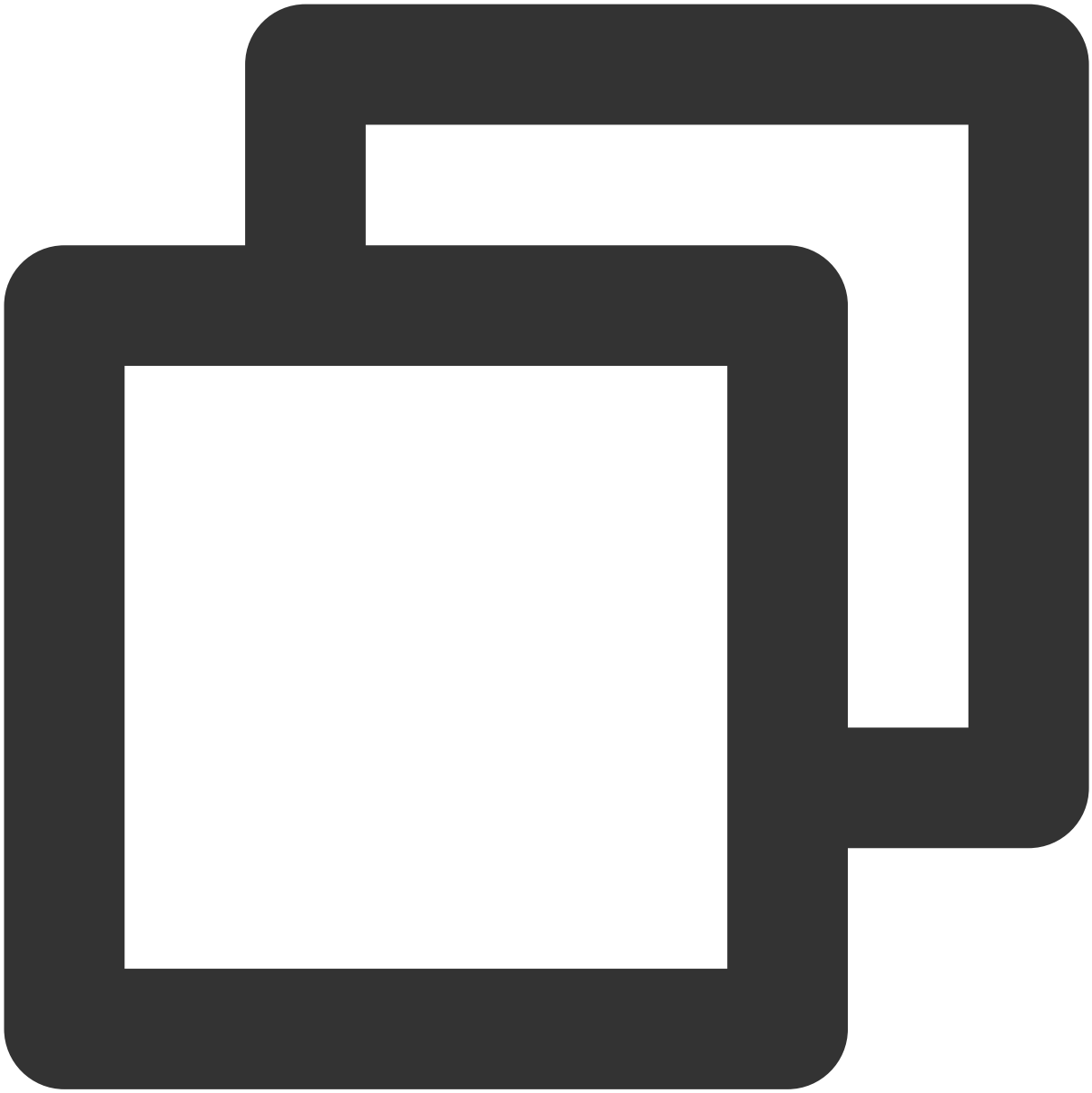
// Create linear gradient
const grd = ctx.createLinearGradient(0, 0, 200, 0)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```

版权所有：腾讯云计算（北京）有限责任公司



使用 `createCircularGradient()`

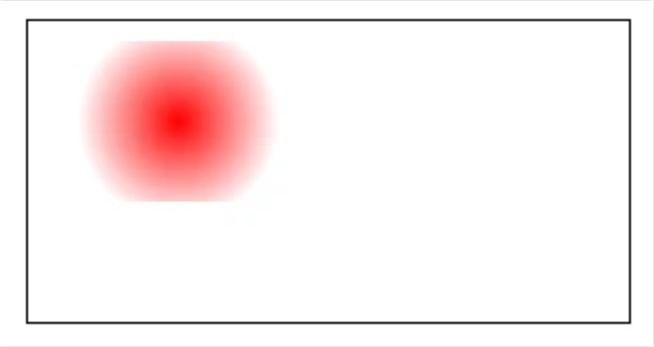


```
const ctx = wx.createCanvasContext('myCanvas')

// Create circular gradient
const grd = ctx.createCircularGradient(75, 50, 50)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```

版权所有：腾讯云计算（北京）有限责任公司

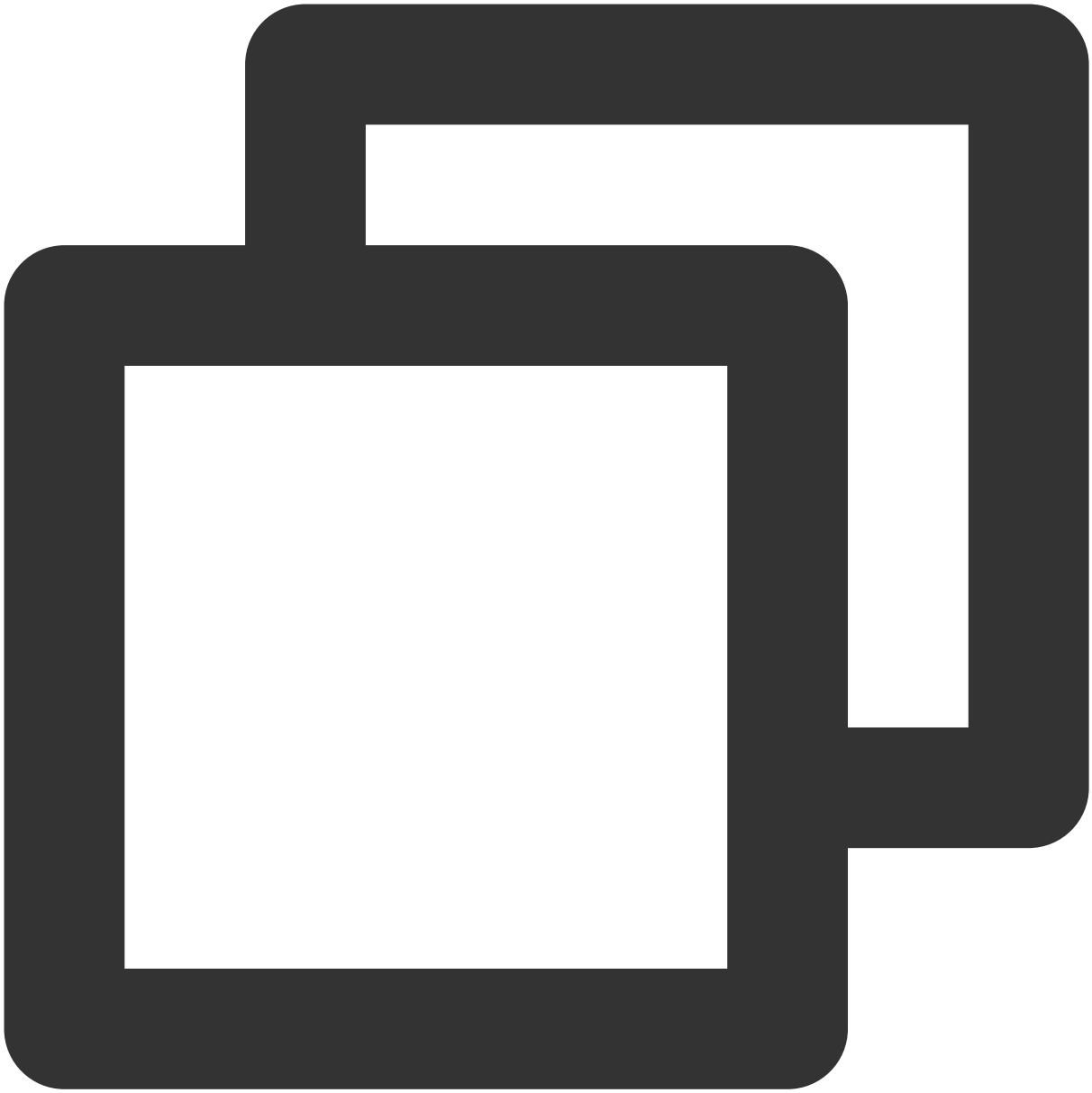


自定义 tabBar

自定义 tabBar 可以让开发者更加灵活地设置 tabBar 样式，以满足更多个性化的场景。
在自定义 tabBar 模式下
为了保证低版本兼容以及区分哪些页面是 tab 页，tabBar 的相关配置项需完整声明，但这些字段不会作用于自定义 tabBar 的渲染。
此时需要开发者提供一个自定义组件来渲染 tabBar，所有 tabBar 的样式都由该自定义组件渲染。推荐用 fixed 在底部的 <cover-view> + <cover-image> 组件渲染样式，以保证 tabBar 层级相对较高。
与 tabBar 样式相关的接口，如 wx.setTabBarItem 等将失效。
每个 tab 页下的自定义 tabBar 组件实例是不同的，可通过自定义组件下的 getTabBar 接口，获取当前页面的自定义 tabBar 组件实例。
说明：
如需实现 tab 选中态，要在当前页面下，通过 getTabBar 接口获取组件实例，并调用 setData 更新选中态。可参考下文 配置信息 中的示例代码。

配置信息

在 `app.json` 中的 `tabBar` 项指定 `custom` 字段，同时其余 `tabBar` 相关配置也补充完整。
所有 tab 页的 json 里需声明 `usingComponents` 项，也可以在 `app.json` 全局开启。
示例代码：

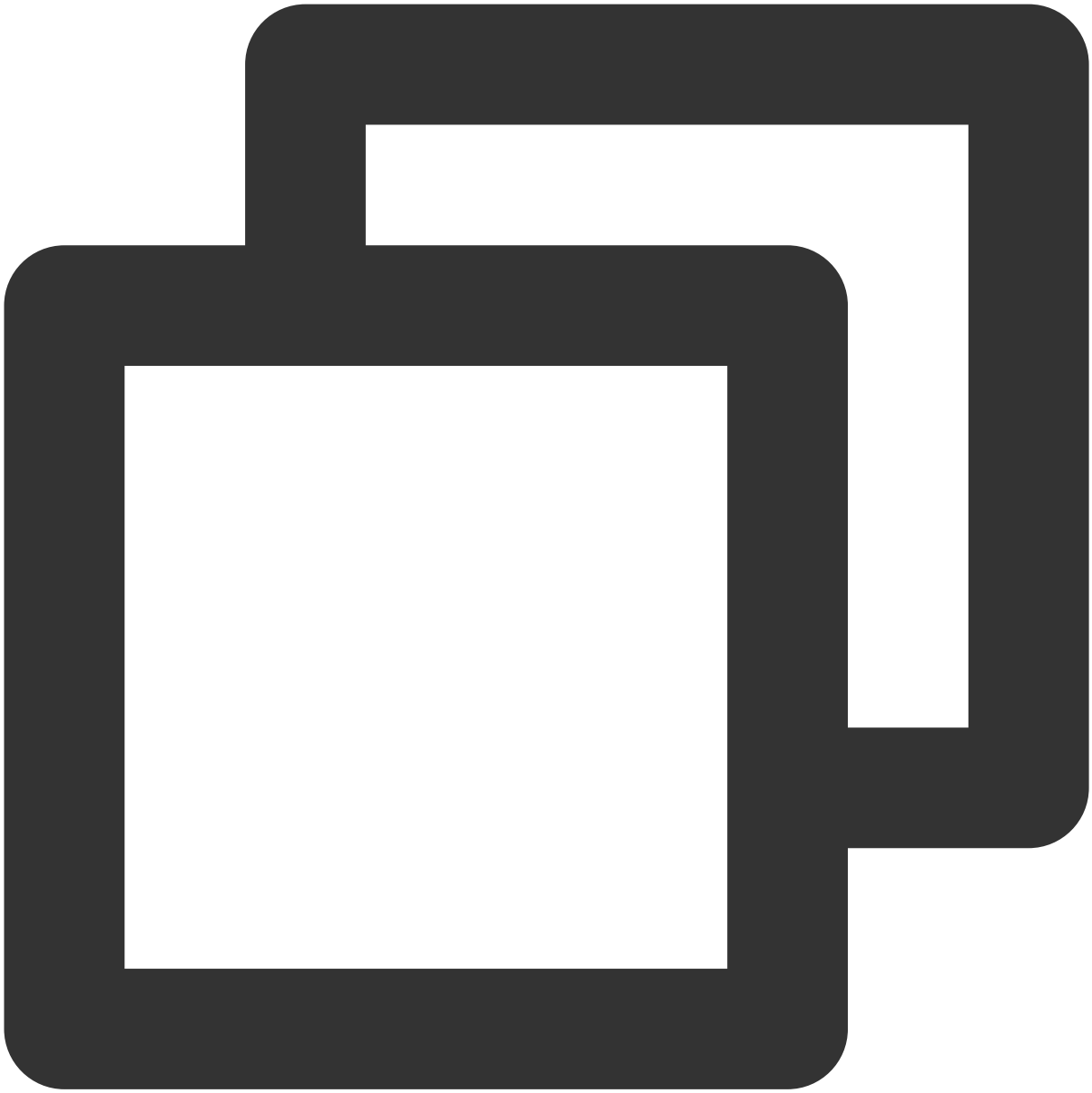


```
{
  "tabBar": {
    "custom": true,
    "color": "#000000",
    "selectedColor": "#000000",
    "backgroundColor": "#000000",
    "list": [
      {
        "pagePath": "page/develop/miniprogram/develop/miniprogram/component/index",
        "text": "组件"
      },
```

```
        {
          "pagePath": "page/develop/miniprogram/API/index",
          "text": "接口"
        }
      ]
    },
    "usingComponents": {}
}
```

添加 tabBar 代码

在代码根目录下添加入口文件：

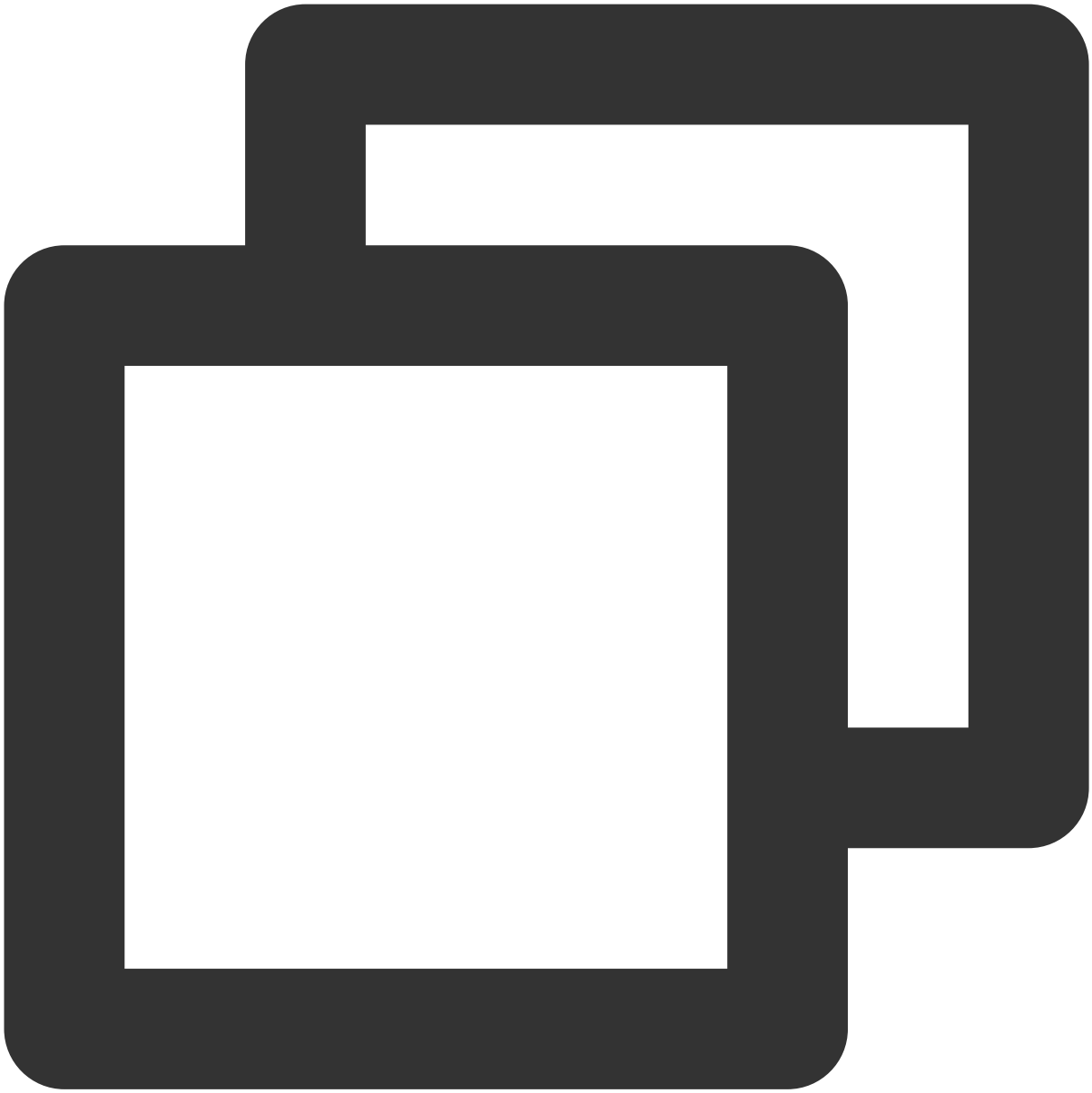


```
custom-tab-bar/index.js
custom-tab-bar/index.json
custom-tab-bar/index.wxml
custom-tab-bar/index.wxss
```

编写 tabBar 代码

用自定义组件的方式编写即可，该自定义组件完全接管 tabBar 的渲染。另外，自定义组件新增 `getTabBar` 接口，可获取当前页面下的自定义 tabBar 组件实例。

示例代码：



```
// page/index/index.js

Page({
  onShow:function(){

    // 取当前页面的TabBar实例，设置选中态
    if (typeof this.getTabBar === 'function' && this.getTabBar()) {
      this.getTabBar().setData({
        selected: 0
      })
    }
```

```
    }
})
```

版权所有：腾讯云计算（北京）有限责任公司

分包加载

最近更新时间：2024-03-04 23:07:33

某些情况下，开发者需要将小程序划分成不同的子包，在构建时打包成不同的分包，用户在使用时按需进行加载。在构建小程序分包项目时，构建会输出一个或多个分包。每个使用分包小程序必定含有一个主包。所谓的主包，即放置默认启动页面/TabBar 页面，以及一些所有分包都需用到公共资源/JS 脚本；而分包则是根据开发者的配置进行划分。

在小程序启动时，默认会下载主包并启动主包内页面，当用户用户进入分包内某个页面时，客户端会把对应分包下载下来，下载完成后再进行展示。

目前小程序分包大小有以下限制：

整个小程序所有分包大小不超过 24M；

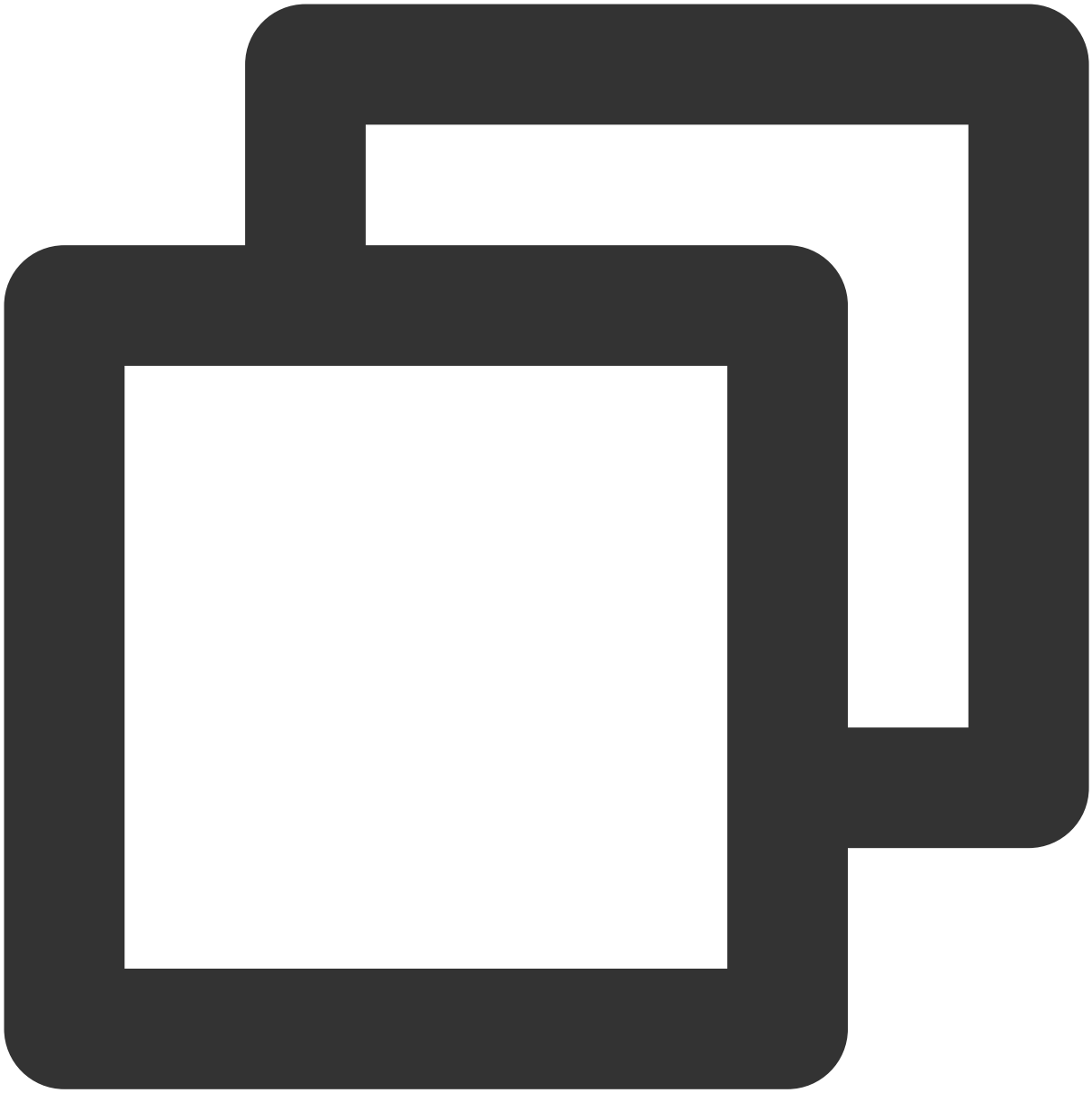
单个分包/主包大小不能超过 2M。

对小程序进行分包，可以优化小程序首次启动的下载时间，以及在多团队共同开发时可以更好的解耦协作。可参见如下三种分包方式：

方法一：使用分包

配置方法

假设支持分包的小程序目录结构如下：

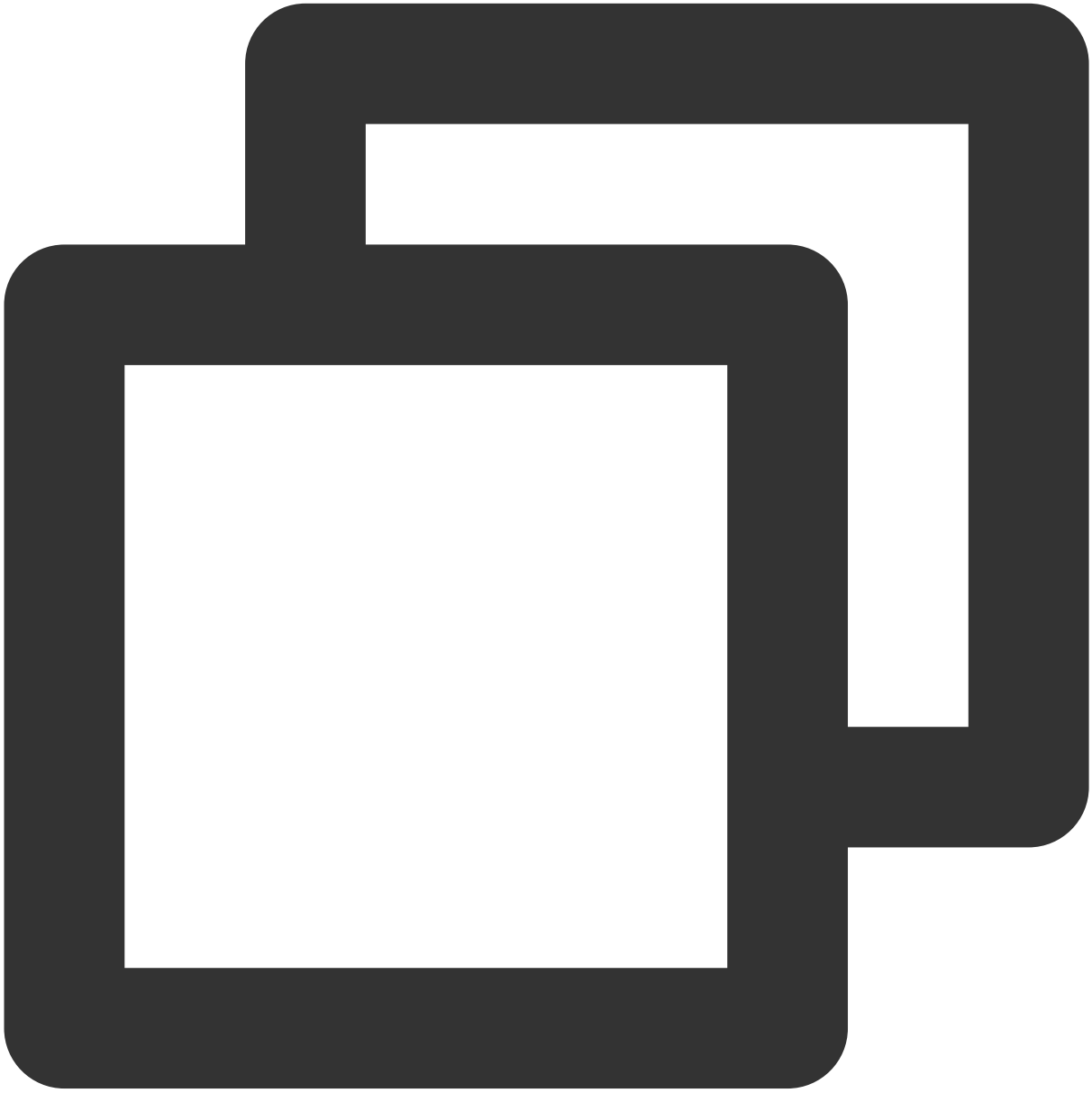


```
├── app.js
├── app.json
├── app.wxss
├── packageA
│   └── pages
│       ├── cat
│       └── dog
├── packageB
│   └── pages
│       ├── apple
│       └── banana
```

版权所有：腾讯云计算（北京）有限责任公司

```
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 app.json subpackages 字段声明项目分包结构（写成 subPackages 也同样支持）：



```
{
  "pages": ["pages/index", "pages/logs"],
  "subpackages": [
    {
      "root": "packageA",
```

```
      "pages": ["pages/cat", "pages/dog"]
    },
    {
      "root": "packageB",
      "name": "pack2",
      "pages": ["pages/apple", "pages/banana"]
    }
  ]
}
```

`subpackages` 中，每个分包的配置有以下几项：

字段	类型	说明
root	String	分包根目录
name	StringArray	分包别名，分包预下载 时使用，如将分包配置到 preloadRule 中，则必须指定
pages	StringArray	分包页面路径，相对于分包根目录
independent	Boolean	分包是否是 独立分包

打包原则

声明 `subpackages` 后，将按 `subpackages` 配置路径进行打包， `subpackages` 配置路径外的目录将被打包到 App（主包） 中；
App（主包）也可以有自己的 pages（即最外层的 pages 字段）；
`subpackage` 的根目录不能是另外一个 `subpackage` 内的子目录；
`tabBar` 页面必须在 app（主包）内。

引用原则

`packageA` 无法 require `packageB` JS 文件，但可以 require app、自己 package 内的 JS 文件；
`packageA` 无法 import `packageB` 的 template，但可以 require app、自己 package 内的 template；
`packageA` 无法使用 `packageB` 的资源，但可以使用 App、自己 package 内的资源。

低版本兼容

由 QQ 后台编译来处理旧版本客户端的兼容，后台会编译两份代码包，一份是分包后代码，另外一份是整包的兼容代码。 新客户端用分包，老客户端还是用的整包，完整包会把各个 `subpackage` 里面的路径放到 pages 中。

方法二：独立分包

版权所有：腾讯云计算（北京）有限责任公司

独立分包是小程序中一种特殊类型的分包，可以独立于主包和其他分包运行。从独立分包中页面进入小程序时，不需要下载主包。当用户进入普通分包或主包内页面时，主包才会被下载。
开发者可以按需将某些具有一定功能独立性的页面配置到独立分包中。当小程序从普通的分包页面启动时，需要首先下载主包；而独立分包不依赖主包即可运行，可以很大程度上提升分包页面的启动速度。
一个小程序中可以有多个独立分包。

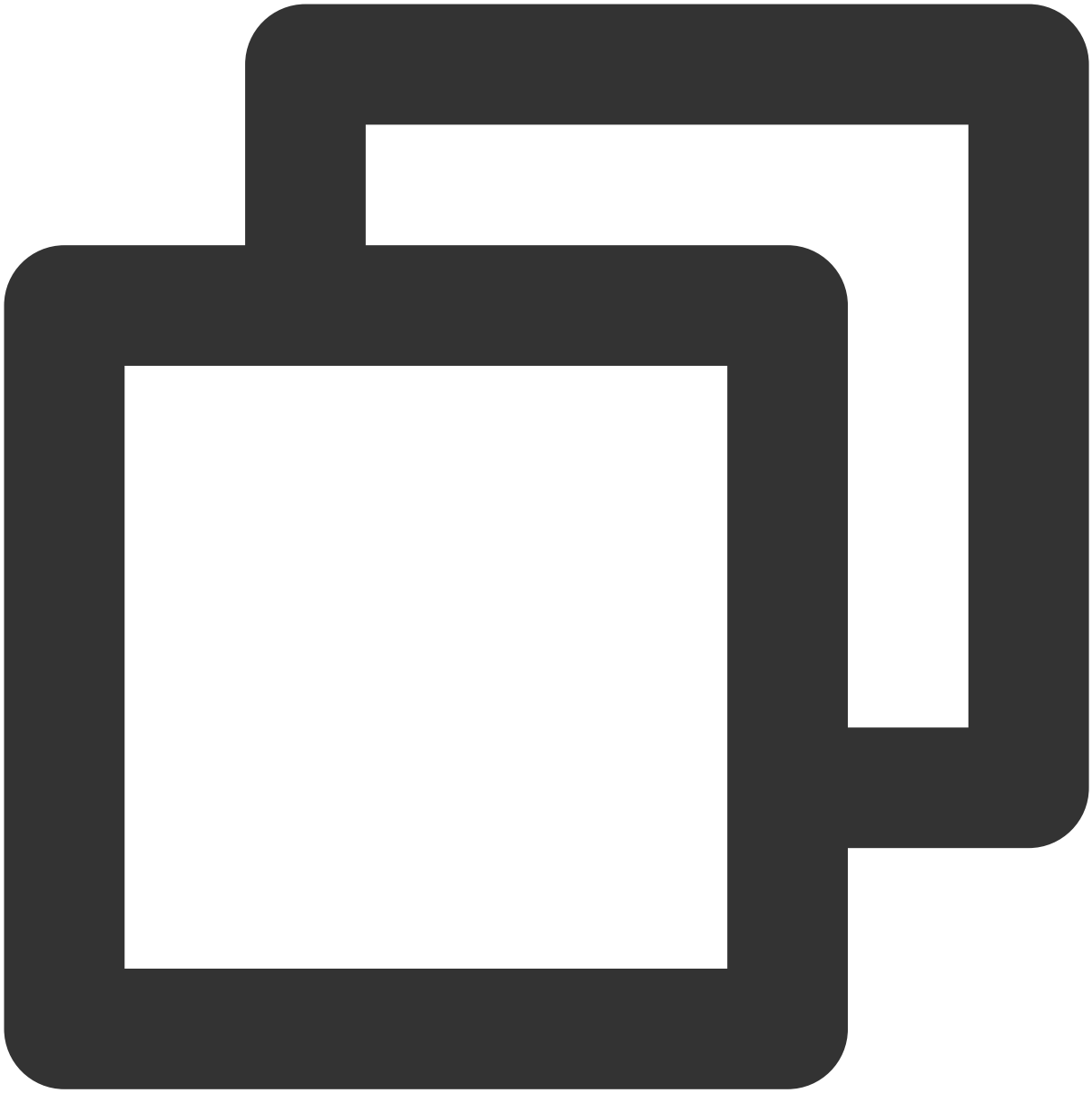
限制

独立分包属于分包的一种。普通分包的所有限制都对独立分包有效。独立分包中插件、自定义组件的处理方式同普通分包。
此外，使用独立分包时要注意：
独立分包中不能依赖主包和其他分包中的内容，包括 js 文件、template、wxss、自定义组件、插件等。主包中的 app.wxss 对独立分包无效，应避免在独立分包页面中使用 app.wxss 中的样式；
App 只能在主包内定义，独立分包中不能定义 App，会造成无法预期的行为；
独立分包中暂时不支持使用插件。

配置方法

假设小程序目录结构如下：

版权所有：腾讯云计算（北京）有限责任公司

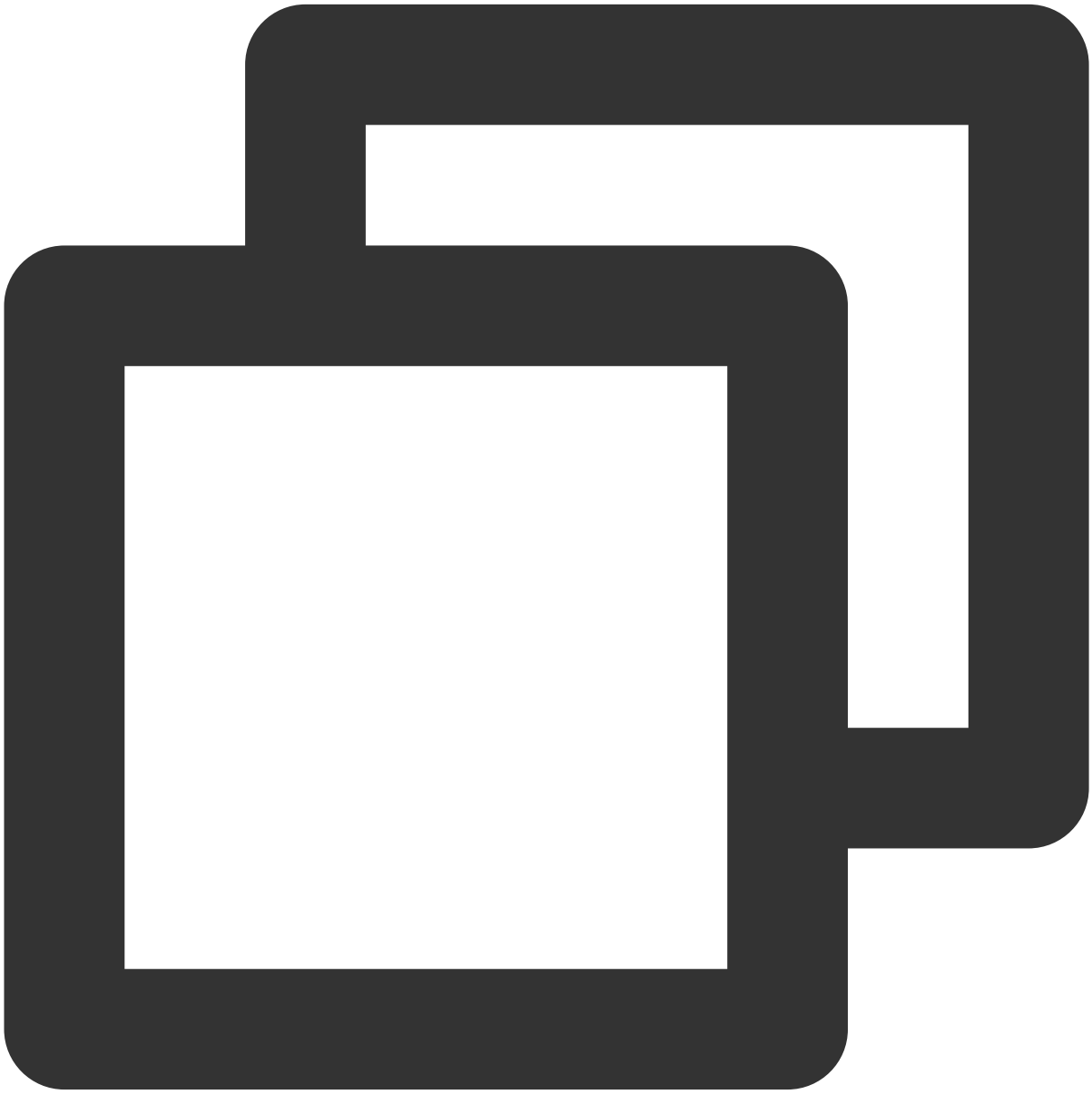


```
├── app.js
├── app.json
├── app.wxss
├── moduleA
│   └── pages
│       ├── rabbit
│       └── squirrel
├── moduleB
│   └── pages
│       ├── pear
│       └── pineapple
```

版权所有：腾讯云计算（北京）有限责任公司

```
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 `app.json` 的 s `ubpackages` 字段中对应的分包配置项中定义 `independent` 字段声明对应分包为独立分包。



```
{
  "pages": ["pages/index", "pages/logs"],
  "subpackages": [
```

```
    {
      "root": "moduleA",
      "pages": ["pages/rabbit", "pages/squirrel"]
    },
    {
      "root": "moduleB",
      "pages": ["pages/pear", "pages/pineapple"],
      "independent": true
    }
  ]
}
```

注意事项

关于 getApp()

与普通分包不同，独立分包运行时，`App` 并不一定被注册，因此 getApp() 也不一定可以获得 App 对象：

当用户从独立分包页面启动小程序时，主包不存在，App 也不存在，此时调用 getApp() 获取到的是 `undefined`。当用户进入普通分包或主包内页面时，主包才会被下载，App 才会被注册。

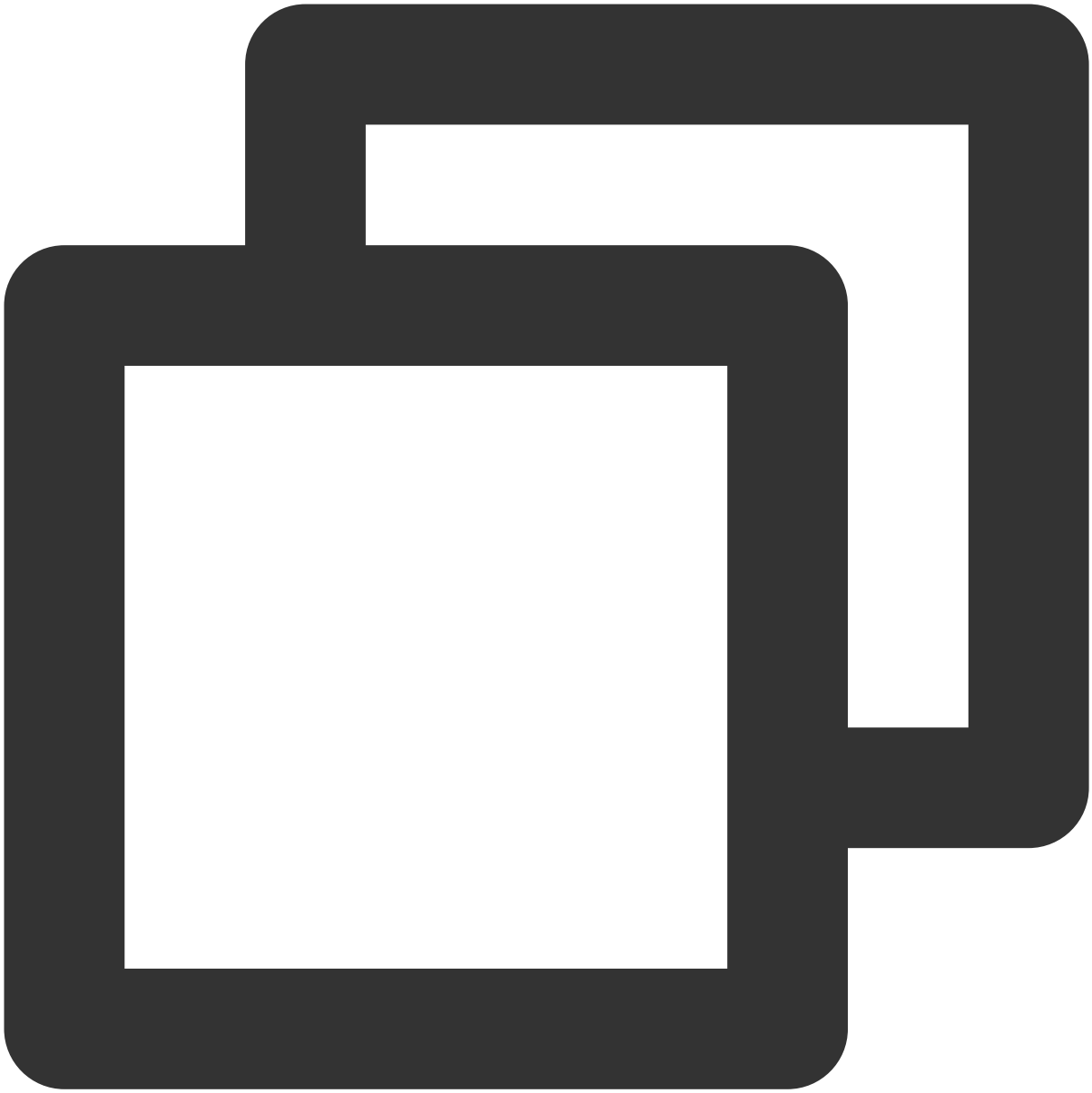
当用户是从普通分包或主包内页面跳转到独立分包页面时，主包已经存在，此时调用 getApp() 可以获取到真正的 App。

由于这一限制，开发者无法通过 App 对象实现独立分包和小程序其他部分的全局变量共享。

getApp 支持 allowDefault 参数，在 App 未定义时返回一个默认实现。当主包加载，App 被注册时，默认实现中定义的属性会被覆盖合并到真正的 App 中。

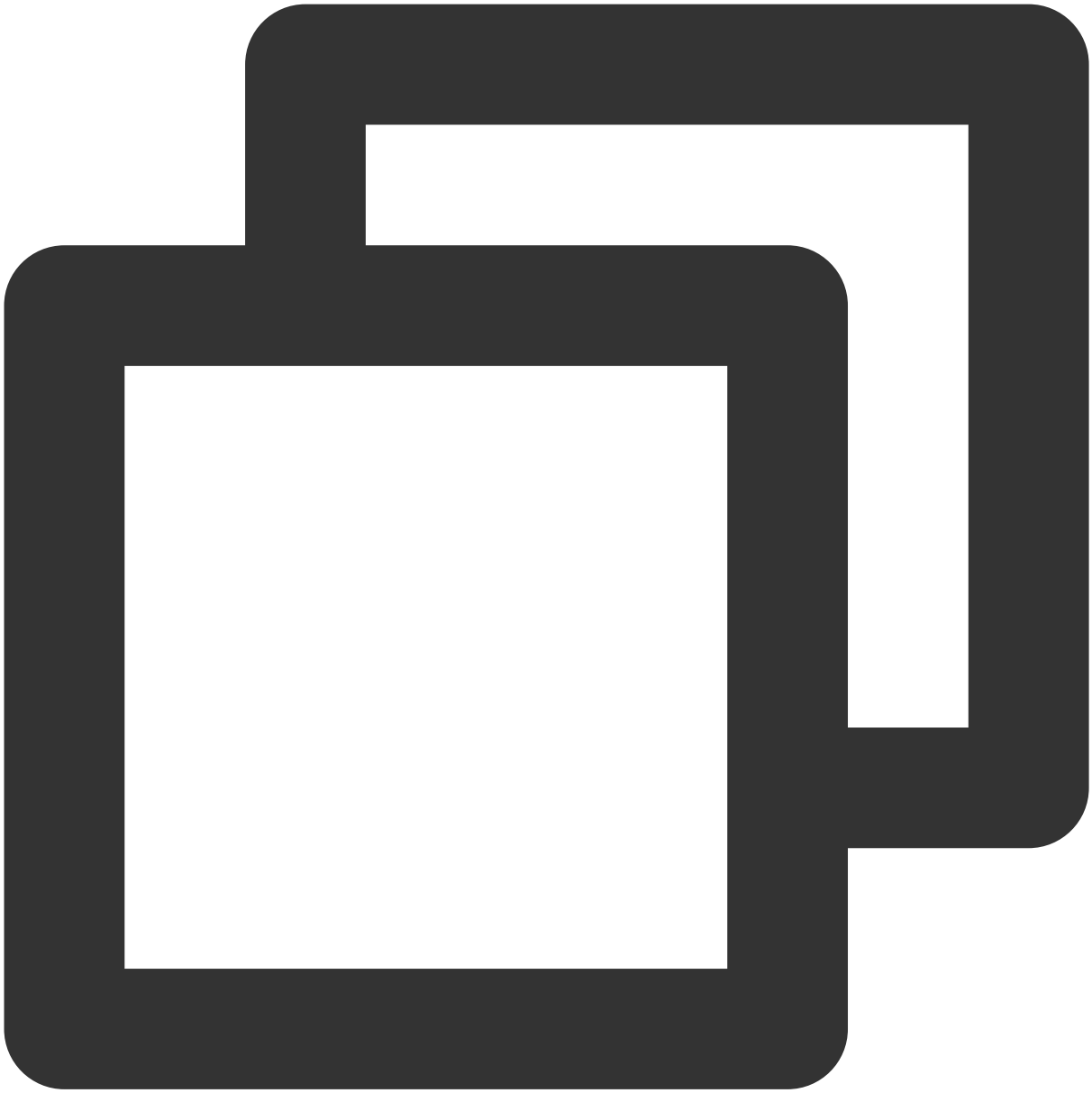
示例代码

独立分包：



```
const app = getApp({allowDefault: true}) // {}
app.data = 456
app.global = {}
```

app.js 中：



```
App({
  data: 123,
  other: 'hello'
})

console.log(getApp()) // {global: {}, data: 456, other: 'hello'}
```

关于 App 生命周期

当从独立分包启动小程序时，主包中 App 的 `onLaunch` 和首次 `onShow` 会在从独立分包页面首次进入主包或其他普通分包页面时调用。

版权所有：腾讯云计算（北京）有限责任公司

由于独立分包中无法定义 App，小程序生命周期的监听可以使用 `wx.onAppShow`，`wx.onAppHide` 完成。App 上的其他事件可以使用 wx.onError，wx.onPageNotFound 监听。

低版本兼容

注意：

在兼容模式下，主包中的 app.wxss 可能会对独立分包中的页面产生影响，因此应避免在独立分包页面中使用 app.wxss 中的样式。

方法三：分包预下载

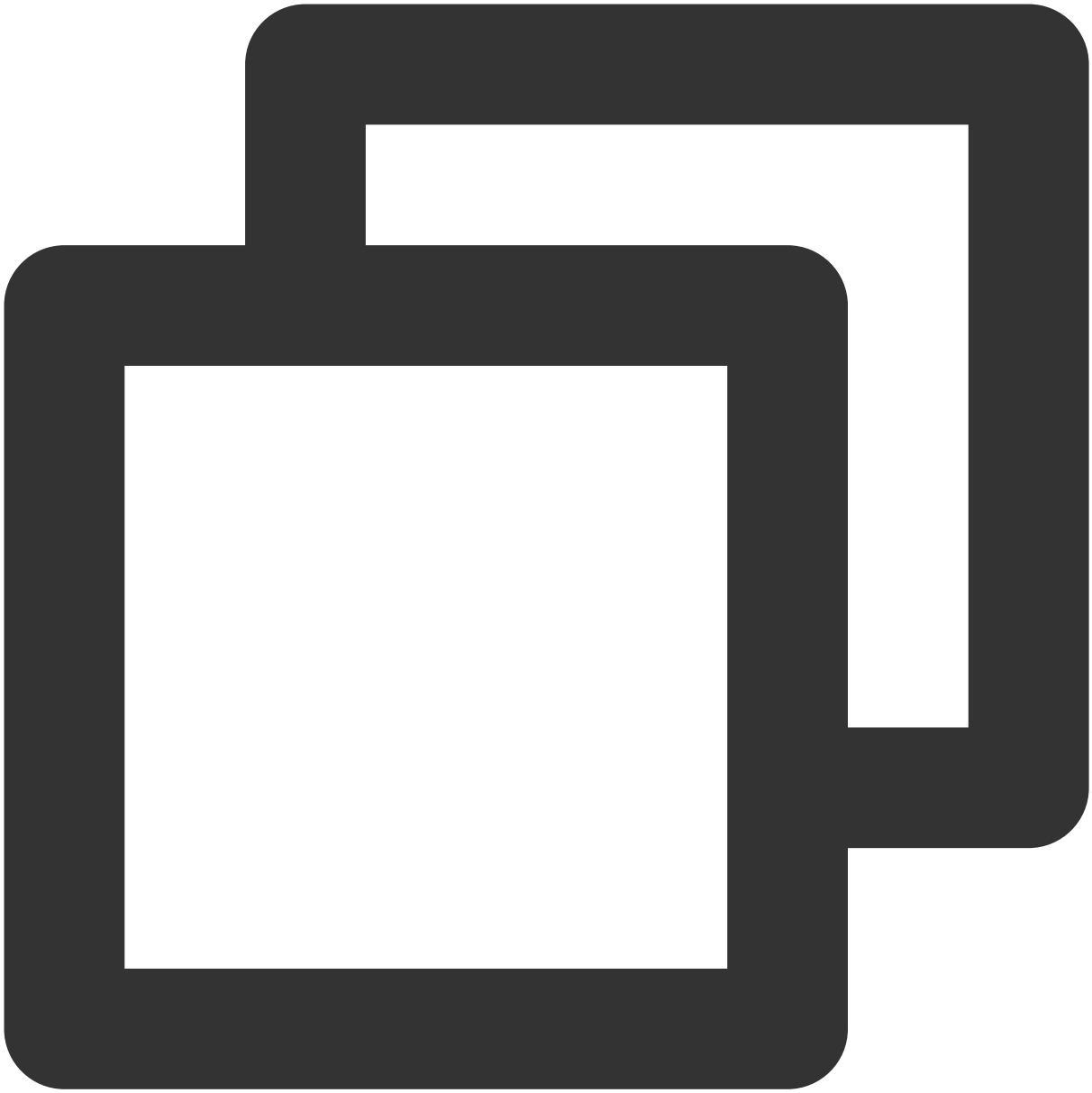
限制

同一个分包中的页面享有共同的预下载大小限额2M，限额会在工具中打包时校验。
如，页面 A 和 B 都在同一个分包中，A 中预下载总大小0.5M的分包，B 中最多只能预下载总大小1.5M的分包。

配置方法

预下载分包行为在进入某个页面时触发，通过在 `app.json` 增加 `preloadRule` 配置来控制。

版权所有：腾讯云计算（北京）有限责任公司



```
{
  "pages": ["pages/index"],
  "subpackages": [
    {
      "root": "important",
      "pages": ["index"]
    },
    {
      "root": "sub1",
      "pages": ["index"]
    },
```

```
    {
      "name": "hello",
      "root": "path/to",
      "pages": ["index"]
    },
    {
      "root": "sub3",
      "pages": ["index"]
    },
    {
      "root": "indep",
      "pages": ["index"],
      "independent": true
    }
  ],
  "preloadRule": {
    "pages/index": {
      "network": "all",
      "packages": ["important"]
    },
    "sub1/index": {
      "packages": ["hello", "sub3"]
    },
    "sub3/index": {
      "packages": ["path/to"]
    },
    "indep/index": {
      "packages": ["__APP__"]
    }
  }
}
```

preloadRule 中，key 是页面路径，value 是进入此页面的预下载配置，每个配置有以下几项：

字段	类型	必填	默认值	说明
packages	StringArray	是	无	进入页面后预下载分包的 root 或 name。__APP__ 表示主包。
network	String	否	Wi-Fi	在指定网络下预下载，可选值为： all：不限网络 Wi-Fi：仅 Wi-Fi 下预下载

可用性

最近更新时间：2024-03-04 23:13:21

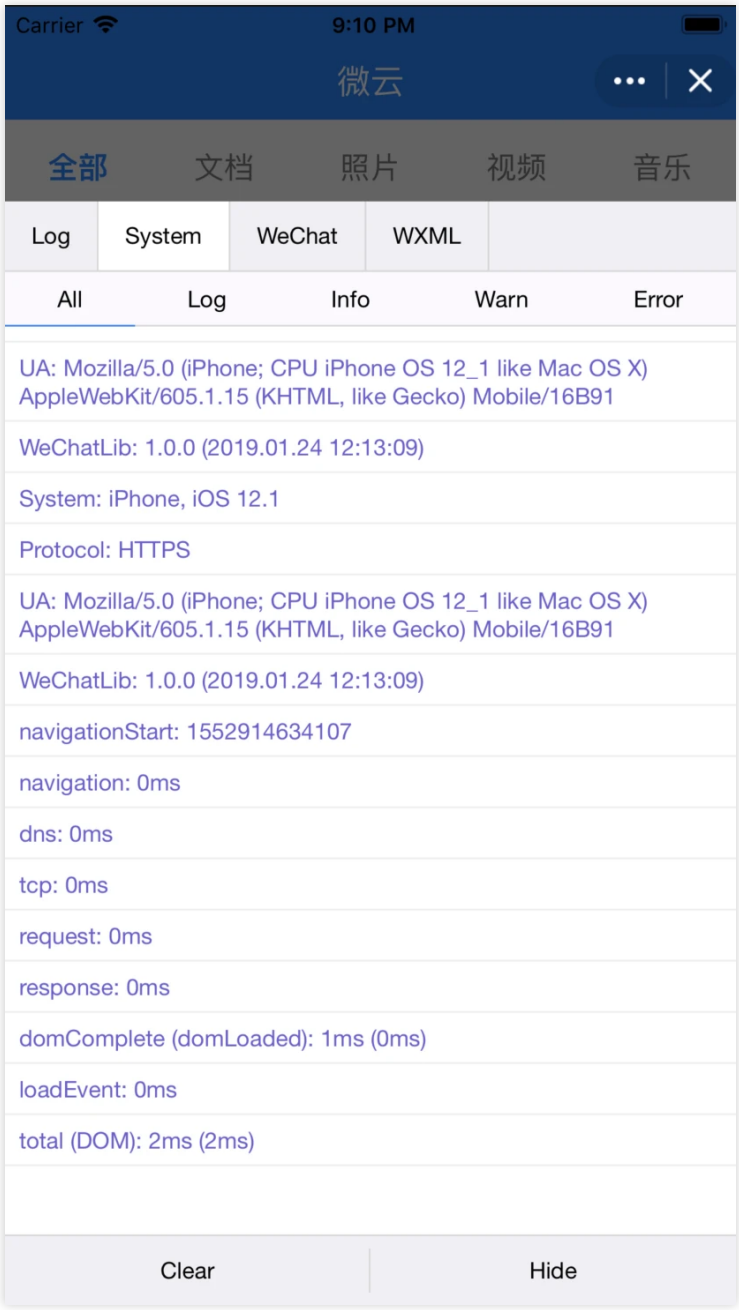
调试

vConsole

在真机上，如果想要查看 console API 输出的日志内容和额外的调试信息，单击屏幕右上角的按钮打开的菜单里选择**打开调试**。此时小程序会退出，重新打开后会右下角会出现一个 `vConsole` 按钮。单击 `vConsole` 按钮可以打开日志面板。

小程序的 vConsole 展示内容如下：

版权所有：腾讯云计算（北京）有限责任公司



vConsole 使用说明

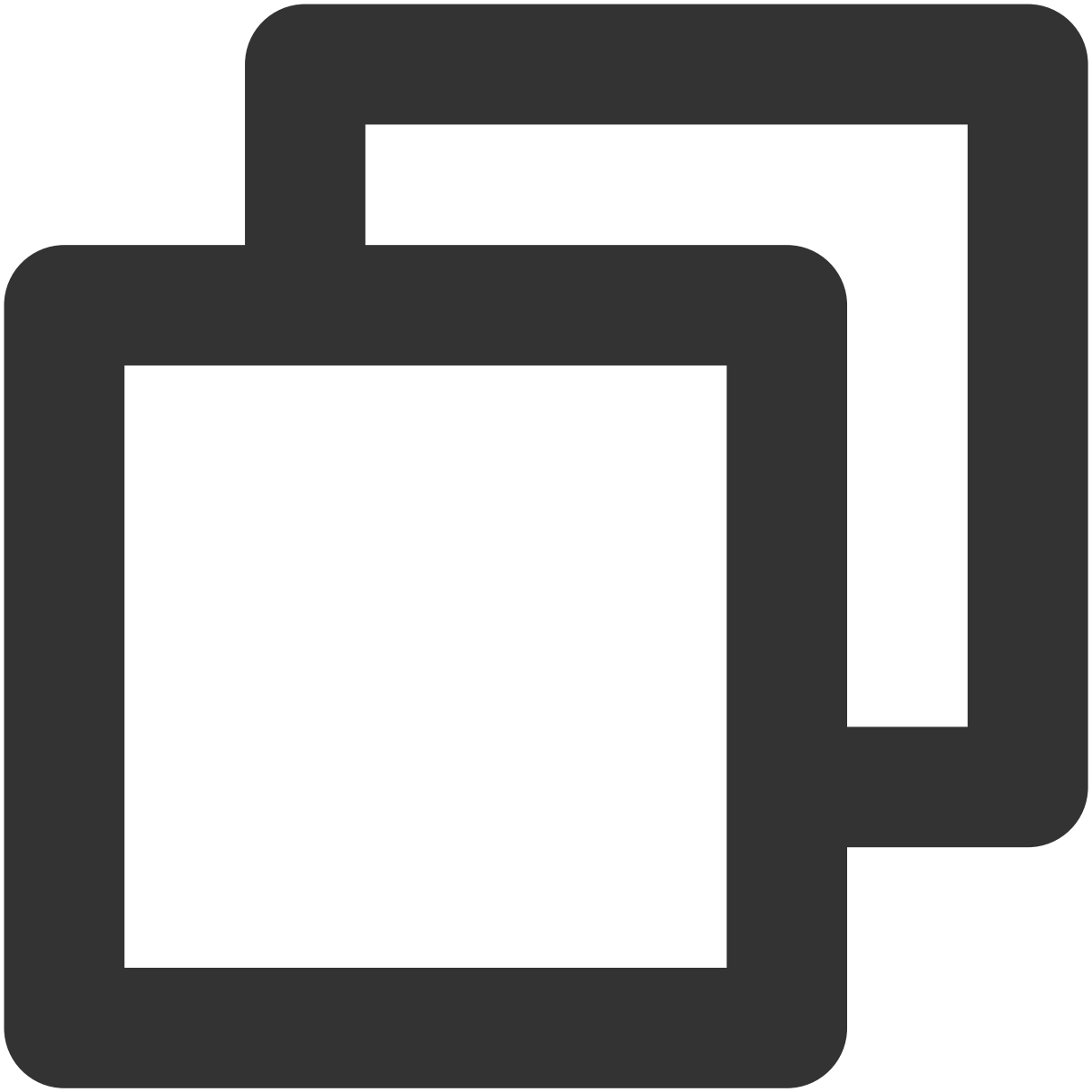
由于实现机制的限制，开发者调用 console API 打印的日志内容，是转换成 `JSON` 字符串后传输给 `vConsole` 的，导致 `vConsole` 中展示的内容会有一些限制：

除了 `Number` 、 `String` 、 `Boolean` 、 `null` 外，其他类型都会被作为 `Object` 处理展示，打印对象及原型链中的 Enumerable 属性。

`Infinity` 和 `NaN` 会显示为 `null` 。

`undefined` 、 `ArrayBuffer` 、 `Function` 类型无法显示。

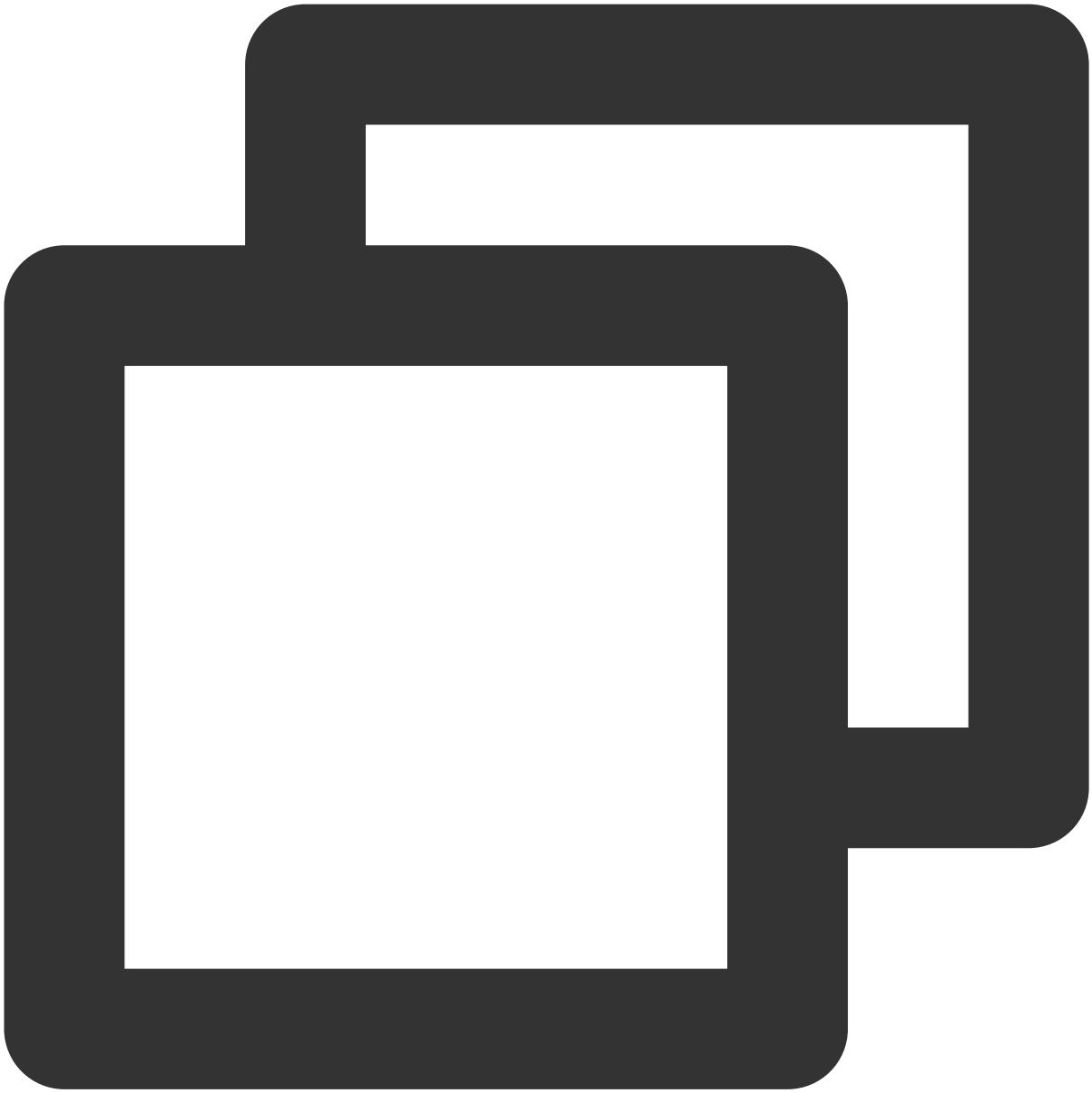
无法打印存在循环引用的对象。



```
const a = {}
a.b = a
console.log(a)
```

针对上述问题，小程序在使用 vConsole 时做了一些处理

版权所有：腾讯云计算（北京）有限责任公司



```
const circular = {x: {}, c: {}}
circular.x = [{promise: Promise.resolve()}]
circular.a = circular
circular.c.x0 = circular.x[0]

console.log(circular)
// "{a: '<Circular: @>', c: {x0: '<Circular: @.x[0]>'}, x: [{promise: '<Promise>'}]
```

说明：

尽量避免在非调试情景下打印结构过于复杂或内容过长的日志内容，可能会带来额外耗时。

版权所有：腾讯云计算（北京）有限责任公司

运行环境

小程序运行环境

小程序运行在三端：iOS（iPhone/iPad）、Android 和用于调试的开发者工具。
三端的脚本执行环境以及用于渲染非原生组件的环境是各不相同：
在 iOS 上，小程序逻辑层的 javascript 代码运行在 JavaScriptCore 中，视图层是由 WKWebView 来渲染的，环境有 iOS8、iOS9、iOS10。
在 Android 上，小程序逻辑层的 javascript 代码运行中 X5 JSCore 中，视图层基于 Mobile Chrome 97 内核实现了 同层渲染 的功能。
在 开发工具上，小程序逻辑层的 javascript 代码是运行在 NW.js 中，视图层是由 Chromium 60 Webview 来渲染的。

运行限制

基于安全考虑，小程序中不支持动态执行 JS 代码，即：
不支持使用 `eval` 执行 `JS` 代码。
不支持使用 `new Function` 创建函数。

平台差异

尽管三端的环境是十分相似的，但是还是有些许区别：
`JavaScript` 语法和 API 支持不一致：语法上开发者可以通过开启 `ES6` 转 `ES5` 的功能来规避；此外，小程序基础库内置了必要的 Polyfill，来弥补 API 的差异。
`WXSS` 渲染表现不一致：尽管可以通过开启样式补全来规避大部分的问题，还是建议开发者需要在 iOS 和 Android 上分别检查小程序的真实表现。
说明：
开发者工具仅供调试使用，最终的表现以客户端为准。

客户端 ES6 API 支持情况

说明：
以下表格中 √ 为支持，× 为该项不支持

String	iOS 8	iOS9	iOS10	Android
codePointAt	√	√	√	√
normalize	×	×	√	√
includes	√	√	√	√
startsWith	√	√	√	√
endsWith	√	√	√	√

版权所有：腾讯云计算（北京）有限责任公司

repeat	√	√	√	√
String.fromCodePoint	√	√	√	√

Array	iOS 8	iOS9	iOS10	Android
copyWithin	√	√	√	√
find	√	√	√	√
findIndex	√	√	√	√
fill	√	√	√	√
entries	√	√	√	√
keys	√	√	√	√
values	×	√	√	×
includes	×	√	√	√
Array.from	√	√	√	√
Array.of	√	√	√	√

Number	iOS 8	iOS9	iOS10	Android
isFinite	√	√	√	√
isNaN	√	√	√	√
parseInt	√	√	√	√
parseFloat	√	√	√	√
isInteger	√	√	√	√
EPSILON	√	√	√	√
isSafeInteger	√	√	√	√

Math	iOS 8	iOS9	iOS10	Android			
trunc	√	√	√	√			

sign	√	√	√	√
cbrt	√	√	√	√
clz32	√	√	√	√
imul	√	√	√	√
fround	√	√	√	√
hypot	√	√	√	√
expm1	√	√	√	√
log1p	√	√	√	√
log10	√	√	√	√
log2	√	√	√	√
sinh	√	√	√	√
cosh	√	√	√	√
tanh	√	√	√	√
asinh	√	√	√	√
acosh	√	√	√	√
atanh	√	√	√	√

Object	iOS 8	iOS9	iOS10	Android
is	√	√	√	√
assign	√	√	√	√
getOwnPropertyDescriptor	√	√	√	√
keys	√	√	√	√
getOwnPropertyNames	√	√	√	√

版权所有：腾讯云计算（北京）有限责任公司

getOwnPropertySymbols	√	√	√	√

Other	iOS 8	iOS9	iOS10	Android
Symbol	√	√	√	√
Set	√	√	√	√
Map	√	√	√	√
Proxy	×	×	√	×
Reflect	√	√	√	√
Promise	√	√	√	√

运行机制

小程序启动会有两种情况，一种是**冷启动**，一种是**热启动**。 假如用户已经打开过某小程序，然后在一定时间内再次打开该小程序，此时无需重新启动，只需将后台态的小程序切换到前台，这个过程就是热启动；冷启动指的是用户首次打开或小程序被宿主 App 主动销毁后再次打开的情况，此时小程序需要重新加载启动。

更新机制

小程序冷启动时如果发现有新版本，将会异步下载新版本的代码包，并同时用客户端本地的包进行启动，即新版本的小程序需要等下一次冷启动才会应用上。 如果需要马上应用最新版本，可以使用 wx.getUpdateManager API 进行处理。

运行机制

小程序没有重启的概念。
当小程序进入后台，客户端会维持一段时间的运行状态，超过一定时间后（目前是5分钟）会被容器主动销毁。
在 iOS 上，当客户端在一定时间间隔内（目前是 5 秒）连续收到两次及以上系统内存告警时，会主动进行小程序的销毁，并提示用户**该小程序可能导致宿主响应变慢，被终止**。建议做必要的内存清理。

版权所有：腾讯云计算（北京）有限责任公司

组件

组件概览

最近更新时间：2024-07-12 18:23:20

视图容器

名称	功能说明	最低版本
scroll-view	可滚动视图区域	1.5.0
swiper	滑块视图容器	1.5.0
swiper-item	仅可放置在 swiper 组件中，宽高自动设置为100%	1.5.0
view	视图容器	1.5.0
movable-area	movable-view 的可移动区域	1.5.0
movable-view	可移动的视图容器，在页面中可以拖拽滑动	1.5.0
cover-image	覆盖在原生组件之上的图片视图	1.5.0
cover-view	覆盖在原生组件之上的文本视图	1.5.0

基础内容

名称	功能说明	最低版本
icon	图标组件	1.5.0
progress	进度条	1.5.0
text	文本	1.5.0
rich-text	富文本	1.5.0

表单组件

版权所有：腾讯云计算（北京）有限责任公司

名称	功能说明	最低版本
button	按钮	1.5.0
checkbox	多选项目	1.5.0
checkbox-group	多项选择器，内部由多个 checkbox 组成	1.5.0
editor	富文本编辑器	1.5.18
form	表单	1.5.0
input	输入框	1.5.0
label	用来改进表单组件的可用性	1.5.0
picker	从底部弹起的滚动选择器	1.5.0
picker-view	嵌入页面的滚动选择器	1.5.0
picker-view-column	滚动选择器子项	1.5.0
radio	单选项目	1.5.0
radio-group	单项选择器，内部由多个 radio 组成	1.5.0
slider	滑动选择器	1.5.0
switch	开关选择器	1.5.0
textarea	多行输入框	1.5.0

导航

名称	功能说明	最低版本
navigator	页面链接	1.5.0

媒体组件

名称	功能说明	

版权所有：腾讯云计算（北京）有限责任公司

audio	音频	1.5.0
image	图片	1.5.0
video	视频	1.5.0
animation-video	透明视频	1.5.0
camera	系统相机	1.5.0
live-player	实时音视频播放	1.5.0
live-pusher	实时音视频录制	1.5.0

地图

名称	功能说明	最低版本
map	地图	1.5.0

画布

名称	功能说明	最低版本
Canvas	画布	1.5.0

开放能力

名称	功能说明	最低版本
web-view	承载网页的容器	1.5.0

版权所有：腾讯云计算（北京）有限责任公司

原生组件

最近更新时间：2024-07-12 18:22:46

原生组件

在腾讯云小程序平台开发中，部分组件是由客户端创建的原生组件。这些原生组件是由小程序客户端实现的，具有更高的性能和更丰富的功能。
原生组件包括：
camera：相机组件，用于调用摄像头拍照或录像；
Canvas：画布组件，用于绘制图形、动画等；
input：仅在 focus 时表现为原生组件；
textarea：多行输入框组件，用于输入多行文本；
video：视频组件，用于播放视频；
liver-player：直播播放器组件，用于播放直播流；
map：地图组件，用于显示地图和地图上的标记、路线等。

原生组件的使用限制

由于原生组件脱离在 WebView 渲染流程外，因此在使用时有以下限制：
原生组件的层级是 `最高` 的，所以页面中的其他组件无论设置 `z-index` 为多少，都无法盖在原生组件上。后插入的原生组件可以覆盖之前的原生组件。
原生组件无法在 `<picker-view>` 中使用。
部分 CSS 样式无法应用于原生组件，例如：
无法对原生组件设置 CSS 动画。
无法定义原生组件为 `position: fixed` 。
不能在父级节点使用 `overflow: hidden` 来裁剪原生组件的显示区域。
原生组件的事件监听不能使用 `bind:eventname` 的写法，只支持 `bindeventname` 。原生组件也不支持 `catch` 和 `capture` 的事件绑定方式。
原生组件会遮挡 vConsole 弹出的调试面板。
说明：
在工具上，原生组件是用 web 组件模拟的，因此很多情况并不能很好的还原真机的表现，建议开发者在使用到原生组件时尽量在真机上进行调试。

版权所有：腾讯云计算（北京）有限责任公司

视图容器

最近更新时间：2024-07-12 18:20:43

scroll-view

功能说明：可滚动视图区域。使用竖向滚动时，需要给 `<scroll-view>` 一个固定高度，通过 WXSS 设置 height。

参数及说明：

属性	类型	默认值	说明
scroll-x	boolean	false	允许横向滚动
scroll-y	boolean	false	允许纵向滚动
upper-threshold	number/string	50	距顶部/左边多远时，触发 scrolltoupper
lower-threshold	number/tring	50	距底部/右边多远时，触发 scrolltolower 事件
scroll-top	number/string	-	设置竖向滚动条位置
scroll-left	number/string	-	设置横向滚动条位置
scroll-into-view	string	-	值应为某子元素 id（id 不能以数字开头）。设置哪个方向可滚动，则在哪个方向滚动到该元素
scroll-with-animation	boolean	false	在设置滚动条位置时使用动画过渡
enable-back-to-top	boolean	false	iOS 单击顶部状态栏、Android 双击标题栏时，滚动条返回顶部，只支持竖向
refresher-enabled	boolean	false	开启自定义下拉刷新
refresher-threshold	number	45	设置自定义下拉刷新阈值
refresher-default-style	string	"black"	设置自定义下拉刷新默认样式，支持设置 `black | white | none`，`none` 表示不使用默认样式
refresher-background	string	-	设置自定义下拉刷新区域背景颜色，默认为透明
refresher-triggered	boolean	false	设置当前下拉刷新状态，true 表示下拉刷新已经被触发，false 表示下拉刷新未被触发

版权所有：腾讯云计算（北京）有限责任公司

bounces	boolean	true	iOS 下 scroll-view 边界弹性控制 (同时开启 enhanced 属性后生效)
show-scrollbar	boolean	true	滚动条显隐控制 (同时开启 enhanced 属性后生效)
fast-deceleration	boolean	false	滑动减速速率控制, 仅在 iOS 下生效 (同时开启 enhanced 属性后生效)
binddragstart	eventhandle	-	滑动开始事件 (同时开启 enhanced 属性后生效) detail { scrollTop, scrollLeft }
binddragging	eventhandle	-	滑动事件 (同时开启 enhanced 属性后生效) detail { scrollTop, scrollLeft }
binddragend	eventhandle	-	滑动结束事件 (同时开启 enhanced 属性后生效) detail { scrollTop, scrollLeft, velocity }
bindscrolltoupper	eventhandle	-	滚动到顶部/左边时触发
bindscrolltolower	eventhandle	-	滚动到底部/右边时触发
bindscroll	eventhandle	-	滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}
scroll-anchoring	boolean	false	开启 scroll anchoring 特性，即控制滚动位置不随内容变化而抖动，仅在 iOS 下生效，安卓下可参考CSS `overflow-anchor` 属性
enhanced	boolean	false	启用 scroll-view 增强特性，启用后可通过 ScrollViewContext操作 scroll-view
paging-enabled	boolean	false	分页滑动效果 (同时开启 enhanced 属性后生效)

注意：

滚动条的长度是预估的，若直接子节点的高度差别较大，则滚动条长度可能会不准确。

请勿在 scroll-view 中使用 textarea、map、canvas、video 组件。

scroll-into-view 的优先级高于 scroll-top。

在滚动 scroll-view 时会阻止页面回弹，所以在 scroll-view 中滚动，是无法触发 onPullDownRefresh。

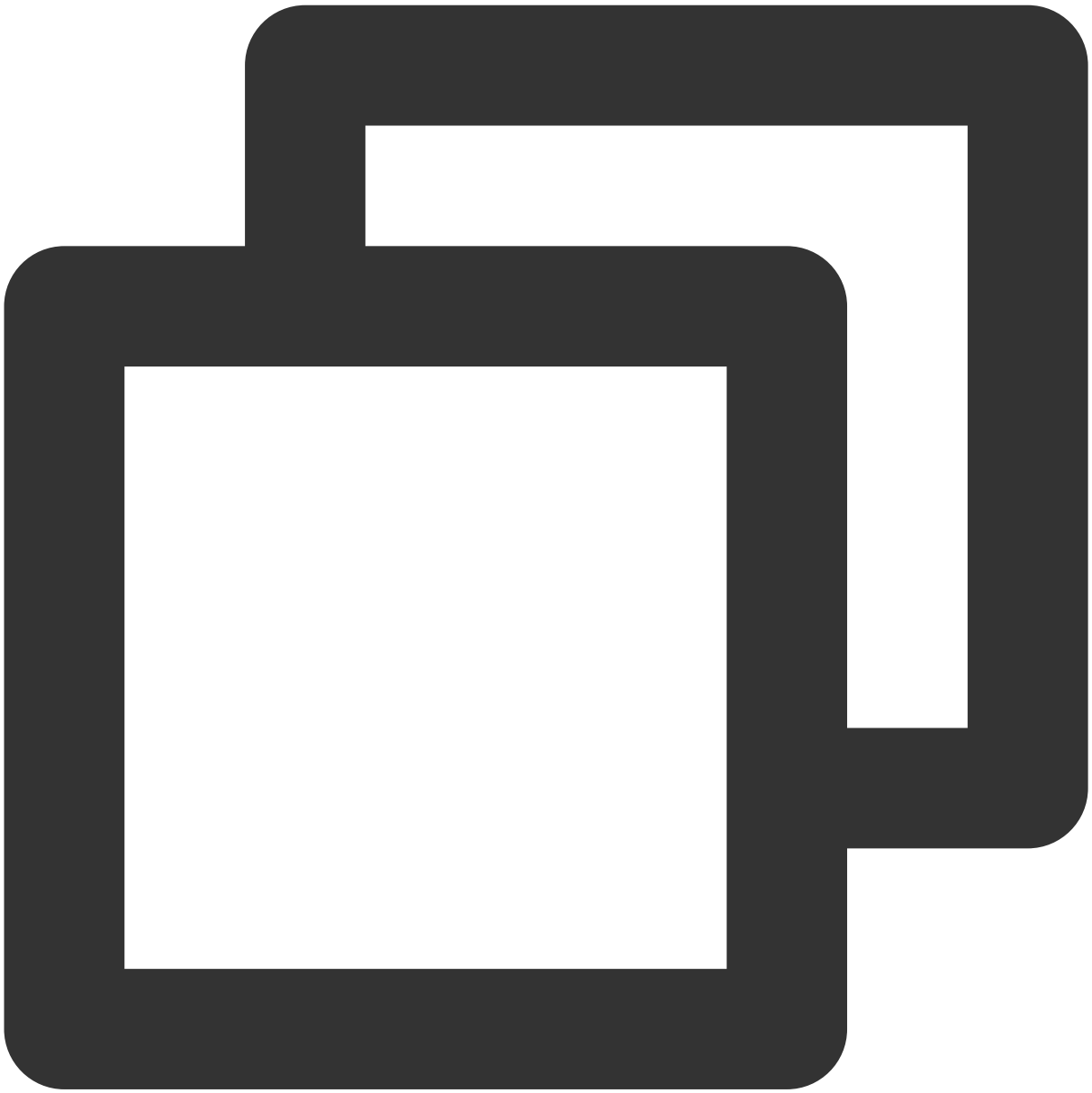
若要使用下拉刷新，请使用页面的滚动，而不是 scroll-view ，这样也能通过点击顶部状态栏回到页面顶部。

示例代码：

WXML

JAVASCRIPT

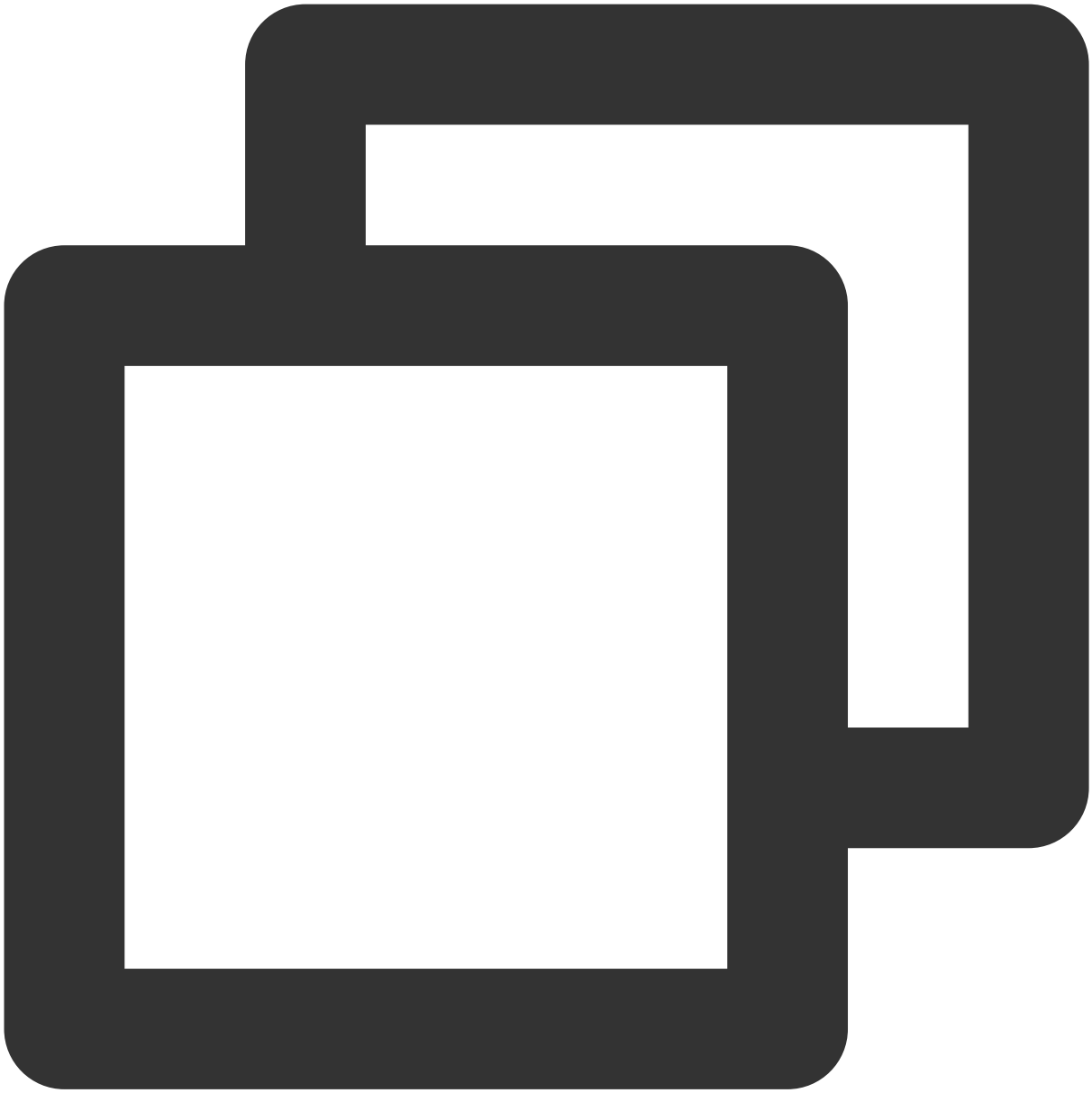
WXSS



```
<view class="container">
  <view class="page-body">
    <view class="page-section">
      <view class="page-section-title">
        <text>Vertical Scroll</text>
      </view>
      <view class="page-section-spacing">
        <scroll-view scroll-y="true" style="height: 300rpx;" bindscrolltoupper="upp
          <view id="demo1" class="scroll-view-item demo-text-1"></view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
          <view id="demo2"  class="scroll-view-item demo-text-2"></view>
          <view id="demo3" class="scroll-view-item demo-text-3"></view>
        </scroll-view>
      </view>
    </view>
    <view class="page-section">
      <view class="page-section-title">
        <text>Horizontal Scroll</text>
      </view>
      <view class="page-section-spacing">
        <scroll-view class="scroll-view_H" scroll-x="true" bindscroll="scroll" styl
          <view id="demo1" class="scroll-view-item_H demo-text-1"></view>
          <view id="demo2"  class="scroll-view-item_H demo-text-2"></view>
          <view id="demo3" class="scroll-view-item_H demo-text-3"></view>
        </scroll-view>
      </view>
    </view>
  </view>
</view>
```



```
const order = ['demo1', 'demo2', 'demo3']

Page({
  onShareAppMessage() {
    return {
      title: 'scroll-view',
      path: 'page/component/pages/scroll-view/scroll-view'
    }
  },

  data: {
```

版权所有：腾讯云计算（北京）有限责任公司

```
    toView: 'green'
  },

  upper(e) {
    console.log(e)
  },

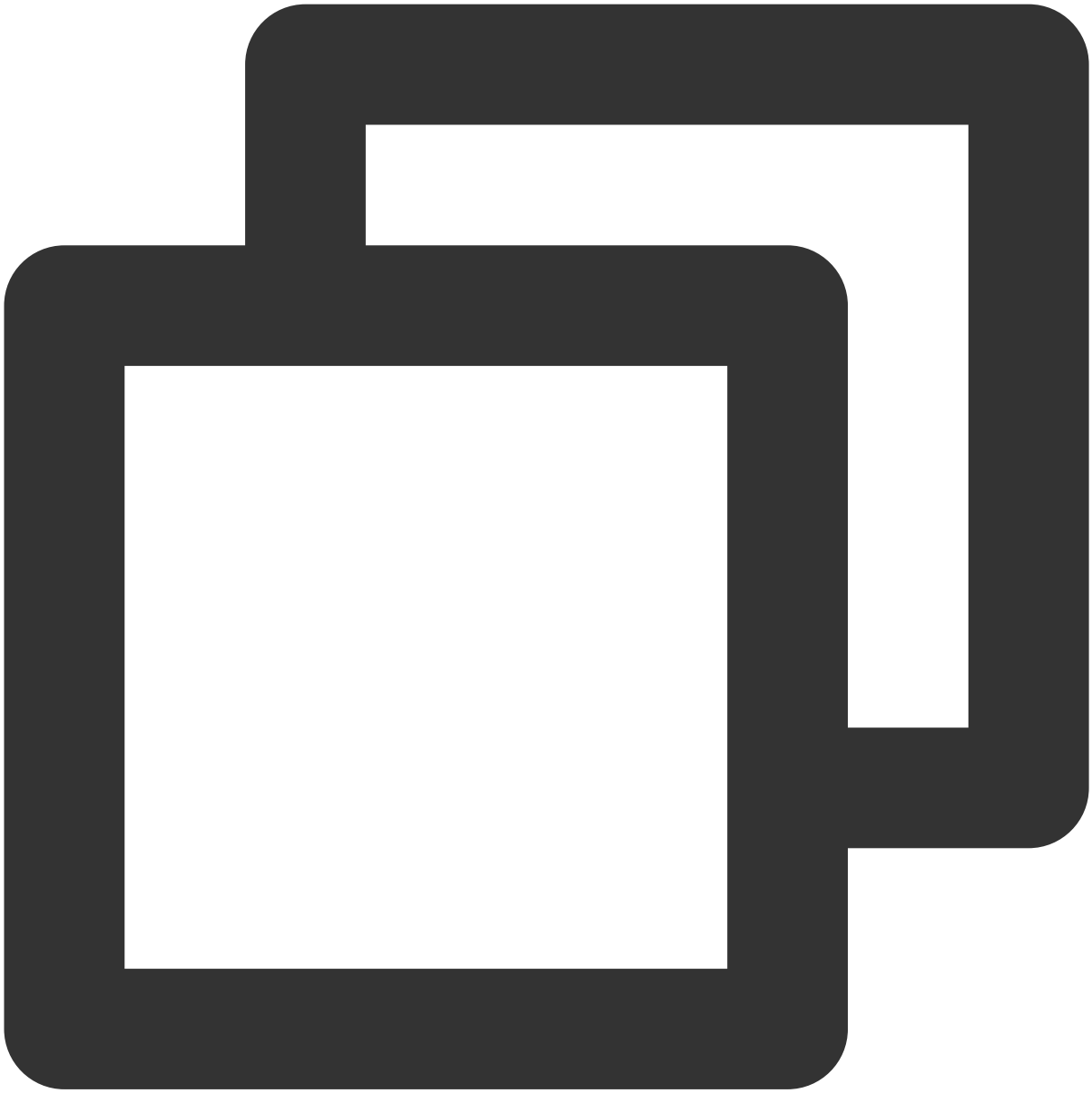
  lower(e) {
    console.log(e)
  },

  scroll(e) {
    console.log(e)
  },

  scrollToTop() {
    this.setAction({
      scrollTop: 0
    })
  },

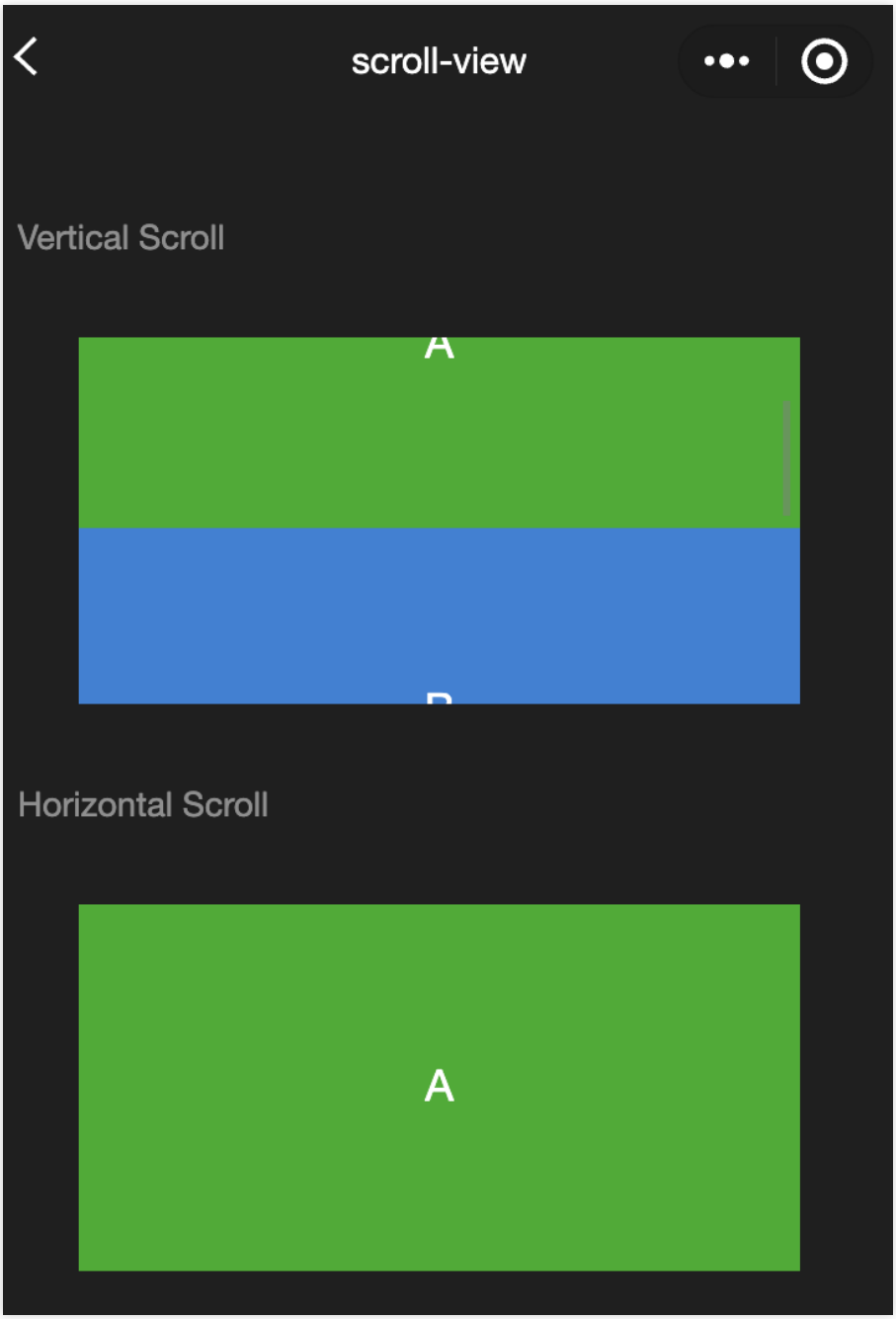
  tap() {
    for (let i = 0; i < order.length; ++i) {
      if (order[i] === this.data.toView) {
        this.setData({
          toView: order[i + 1],
          scrollTop: (i + 1) * 200
        })
        break
      }
    }
  },

  tapMove() {
    this.setData({
      scrollTop: this.data.scrollTop + 10
    })
  }
})
```



```
.page-section-spacing{
  margin-top: 60rpx;
}
.scroll-view_H{
  white-space: nowrap;
}
.scroll-view-item{
  height: 300rpx;
}
.scroll-view-item_H{
  display: inline-block;
```

```
  width: 100%;
  height: 300rpx;
}
```



swiper

功能说明：滑块视图容器。 `change` 事件返回 `detail` 中包含一个 `source` 字段，表示导致变更的原因，可能值如下：

`autoplay` 自动播放导致 swiper 变化；

`touch` 用户滑动引起 swiper 变化；

其他原因将用空字符串表示。

参数及说明：

属性	类型	默认值	说明
indicator-dots	boolean	false	是否显示面板指示点
indicator-color	color	rgba(0, 0, 0, .3)	指示点颜色
indicator-active-color	color	#000000	当前选中的指示点颜色
autoplay	boolean	false	是否自动切换
current	number	0	当前所在滑块的 index
current-item-id	String	""	当前所在滑块的 item-id ，不能与 current 被同时指定
interval	number	5000	自动切换时间间隔
duration	number	500	滑动动画时长
circular	boolean	false	是否采用衔接滑动
vertical	boolean	false	滑动方向是否为纵向
display-multiple-items	number	1	同时显示的滑块数量
previous-margin	string	"0px"	前边距，可用于露出前一项的一小部分，接受 px 和 rpx 值
easing-function	string	"default"	指定 swiper 切换缓动动画类型，合法值如下： `default` ：默认缓动函数 `linear` ：线性动画 `easeInCubic` ：缓入动画 `easeOutCubic` ：缓出动画 `easeInOutCubic` ：缓入缓出动画
next-margin	string	"0px"	后边距，可用于露出后一项的一小部分，接受 px 和 rpx 值
bindchange	eventhandle	-	current 改变时会触发 change 事件，event.detail = {current: current, source: source}
bindtransition	eventhandle	-	swiper-item 的位置发生改变时会触发 transition 事件，event.detail = {dx: dx, dy: dy}

版权所有：腾讯云计算（北京）有限责任公司

bindanimationfinish	eventhandle	-	动画结束时会触发 animationfinish 事件，event.detail 同上
skip-hidden-item-layout	boolean	false	是否跳过未显示的滑块布局，设为 true 可优化复杂情况下的滑动性能，但会丢失隐藏状态滑块的布局信息

注意：

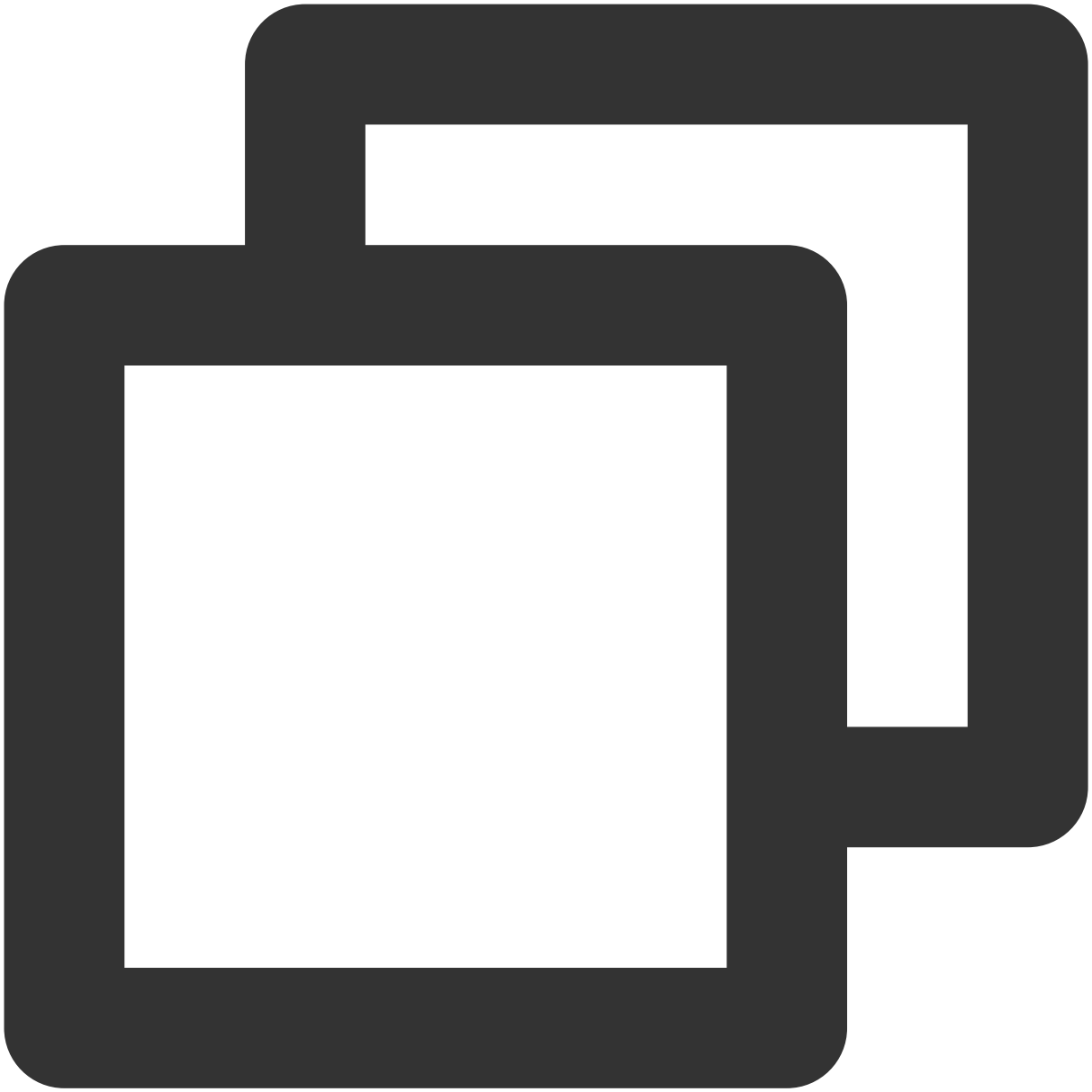
其中只可放置 swiper-item 组件，否则会导致未定义的行为。

如果在 bindchange 的事件回调函数中使用 setData 改变 current 值，则有可能导致 setData 被不停地调用，因而通常情况下请在改变 。current 值前检测 source 字段来判断是否是由于用户触摸引起。

示例代码：

WXML

JAVASCRIPT

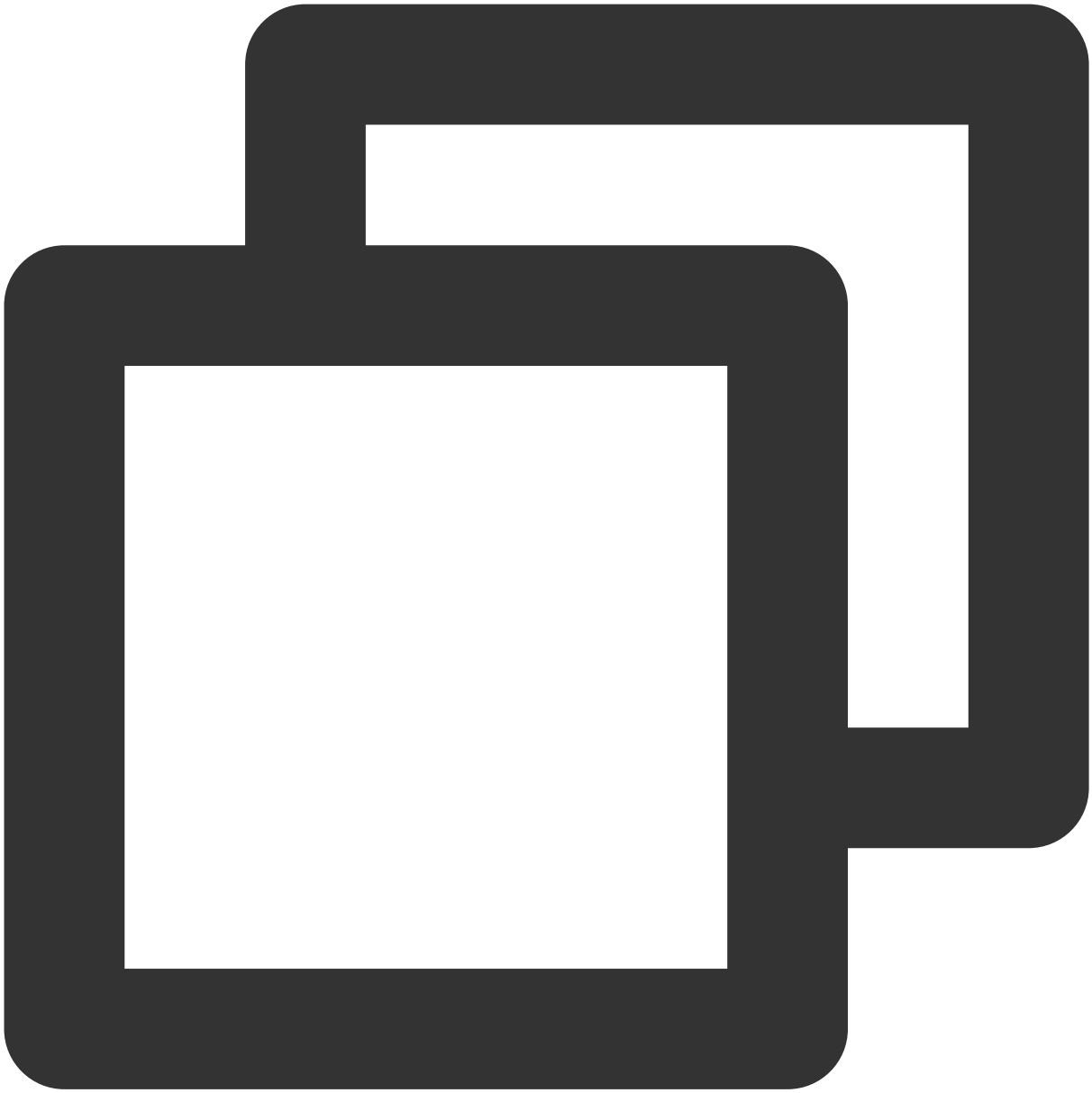


```
<view class="container">
  <view class="page-body">
    <view class="page-section page-section-spacing swiper">
      <swiper indicator-dots="{{indicatorDots}}"
        autoplay="{{autoplay}}" interval="{{interval}}" duration="{{duration}}">
        <block wx:for="{{background}}" wx:key="*this">
          <swiper-item>
            <view class="swiper-item {{item}}"></view>
          </swiper-item>
        </block>
      </swiper>
```

版权所有：腾讯云计算（北京）有限责任公司

```
    </view>
    <view class="page-section" style="margin-top: 40rpx;margin-bottom: 0;">
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_switch">
          <view class="weui-cell__bd">指示点</view>
          <view class="weui-cell__ft">
            <switch checked="{{indicatorDots}}" bindchange="changeIndicatorDots" />
          </view>
        </view>
        <view class="weui-cell weui-cell_switch">
          <view class="weui-cell__bd">自动播放</view>
          <view class="weui-cell__ft">
            <switch checked="{{autoplay}}" bindchange="changeAutoplay" />
          </view>
        </view>
      </view>
    </view>

    <view class="page-section page-section-spacing">
      <view class="page-section-title">
        <text>幻灯片切换时长(ms)</text>
        <text class="info">{{duration}}</text>
      </view>
      <slider bindchange="durationChange" value="{{duration}}" min="500" max="2000"
      <view class="page-section-title">
        <text>自动播放间隔时长(ms)</text>
        <text class="info">{{interval}}</text>
      </view>
      <slider bindchange="intervalChange" value="{{interval}}" min="2000" max="1000
    </view>
  </view>
</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'swiper',
      path: 'page/component/pages/swiper/swiper'
    }
  },

  data: {
    background: ['demo-text-1', 'demo-text-2', 'demo-text-3'],
    indicatorDots: true,
```

版权所有：腾讯云计算（北京）有限责任公司

```
    vertical: false,
    autoplay: false,
    interval: 2000,
    duration: 500
  },

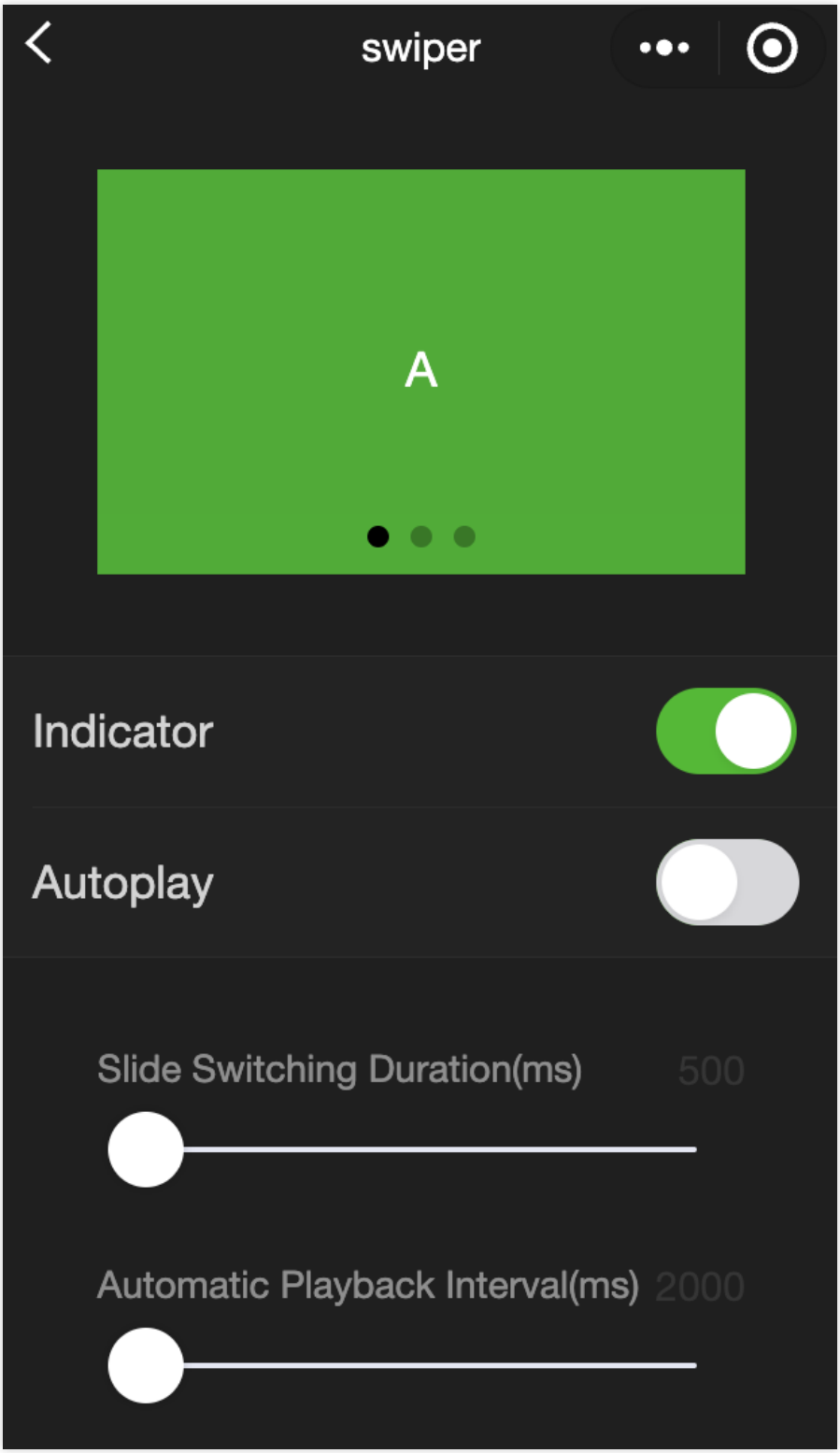
  changeIndicatorDots() {
    this.setData({
      indicatorDots: !this.data.indicatorDots
    })
  },

  changeAutoplay() {
    this.setData({
      autoplay: !this.data.autoplay
    })
  },

  intervalChange(e) {
    this.setData({
      interval: e.detail.value
    })
  },

  durationChange(e) {
    this.setData({
      duration: e.detail.value
    })
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司



swiper-item

功能说明：仅可放置在 swiper 组件中，宽高自动设置为100%。

参数及说明：

属性名	类型	默认值	说明

item-id	string	""	该 swiper-item 的标识符

view

功能描述：视图容器。

参数及说明：

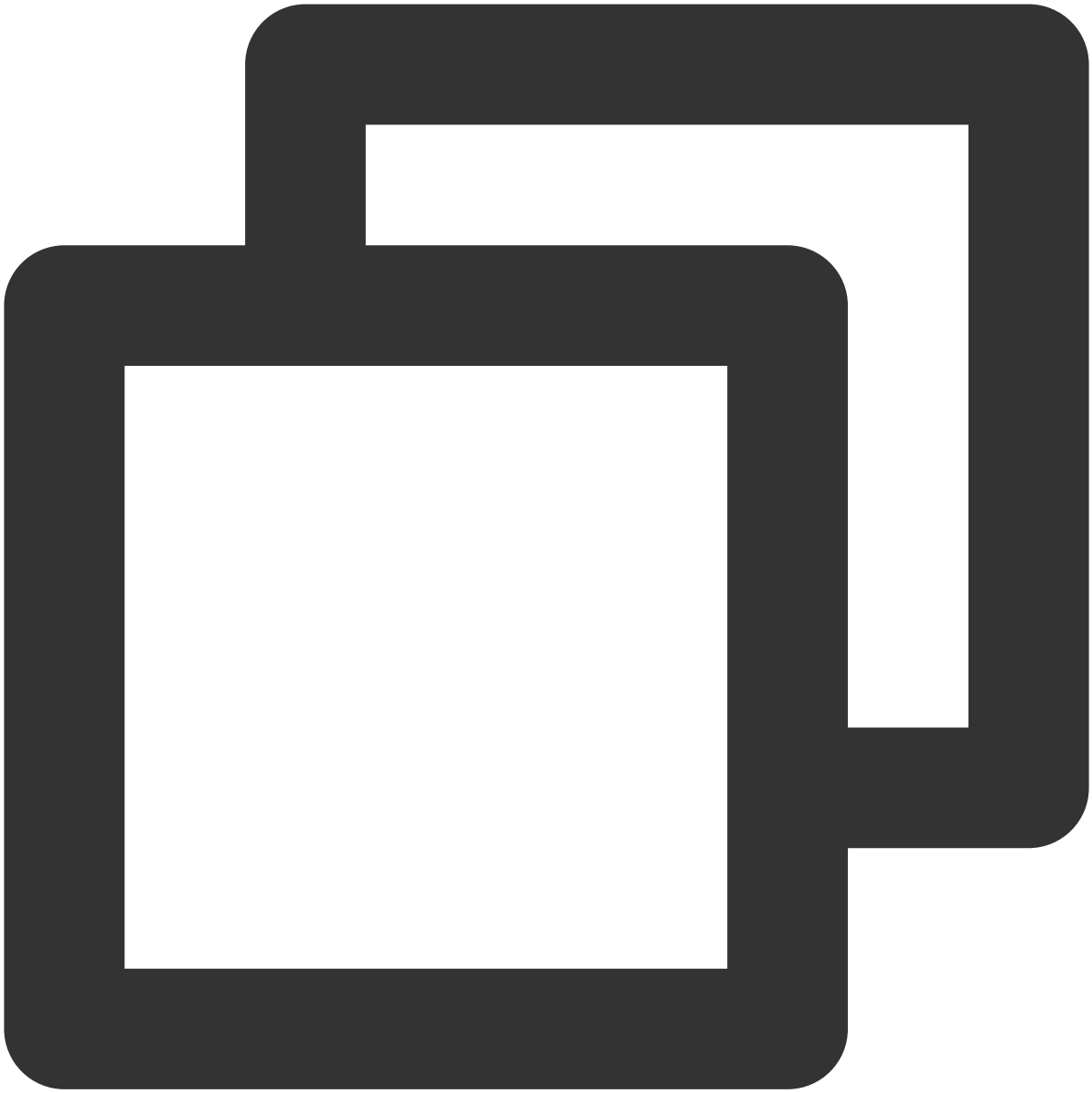
属性名	类型	默认值	说明
hover-class	string	-	指定按下去的样式类。当 `hover-class="none"` 时，没有点击态效果
hover-stop-propagation	boolean	false	指定是否阻止本节点的祖先节点出现点击态
hover-start-time	number	50	按住后多久出现点击态，单位毫秒
hover-stay-time	number	400	手指松开后点击态保留时间，单位毫秒

注意：

如果需要使用滚动视图，请使用scroll-view。

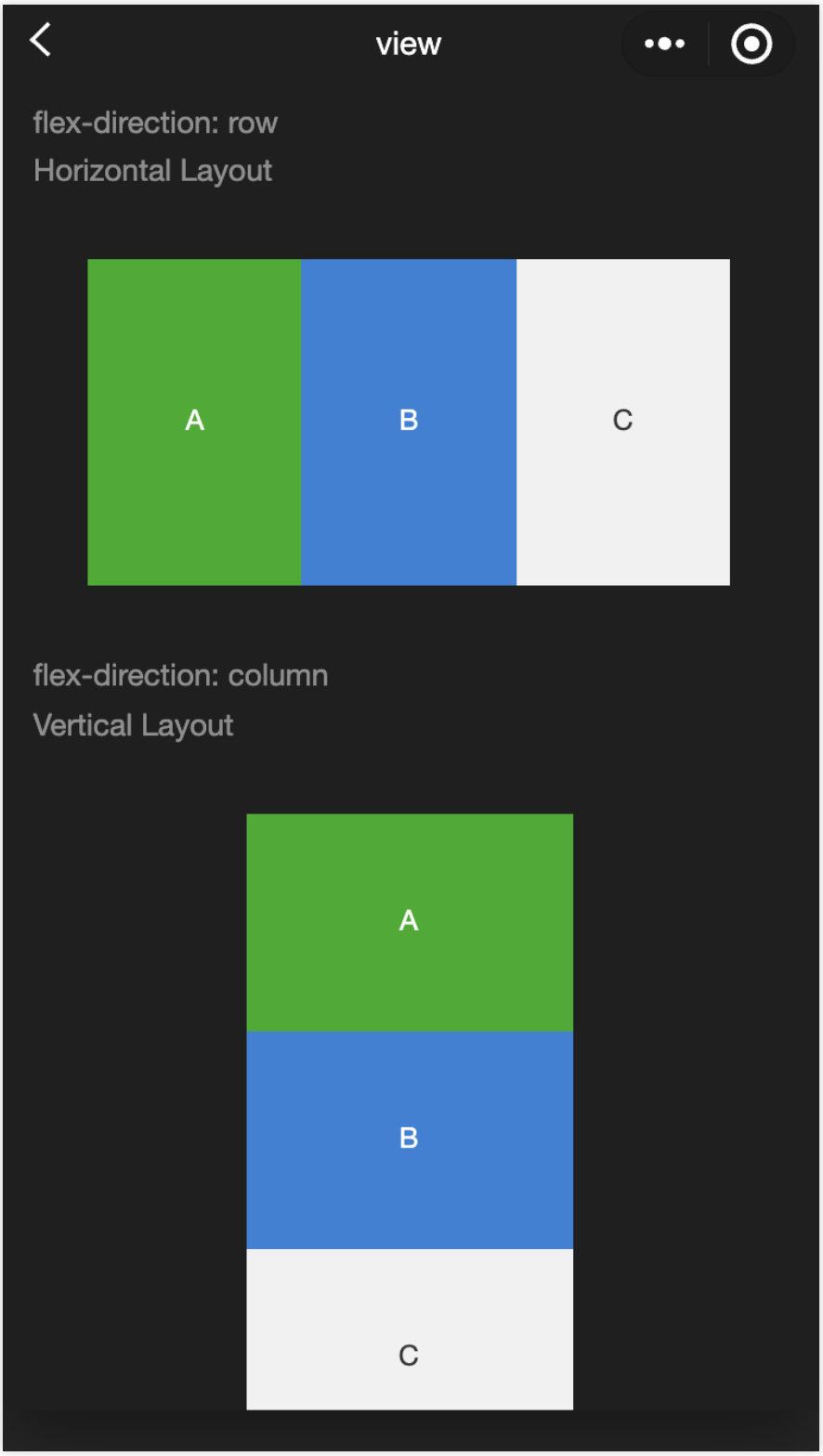
示例代码：

WXML



```
<view class="container">
  <view class="page-body">
    <view class="page-section">
      <view class="page-section-title">
        <text>flex-direction: row\\n横向布局</text>
      </view>
      <view class="page-section-spacing">
        <view class="flex-wrp" style="flex-direction:row;">
          <view class="flex-item demo-text-1"></view>
          <view class="flex-item demo-text-2"></view>
          <view class="flex-item demo-text-3"></view>
```

```
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="page-section-title">
        <text>flex-direction: column\\n纵向布局</text>
      </view>
      <view class="flex-wrp" style="flex-direction:column;">
        <view class="flex-item flex-item-V demo-text-1"></view>
        <view class="flex-item flex-item-V demo-text-2"></view>
        <view class="flex-item flex-item-V demo-text-3"></view>
      </view>
    </view>
  </view>
</view>
```



movable-area

功能说明：movable-view 的可移动区域。

参数及说明：

属性名	类型	默认值	说明

版权所有：腾讯云计算（北京）有限责任公司

scale-area	boolean	false	当里面的 movable-view 设置为支持双指缩放时，设置此值可将缩放手势生效区域修改为整个movable-area。

注意：

movable-area 必须设置 width 和 height 属性，不设置默认为10px。

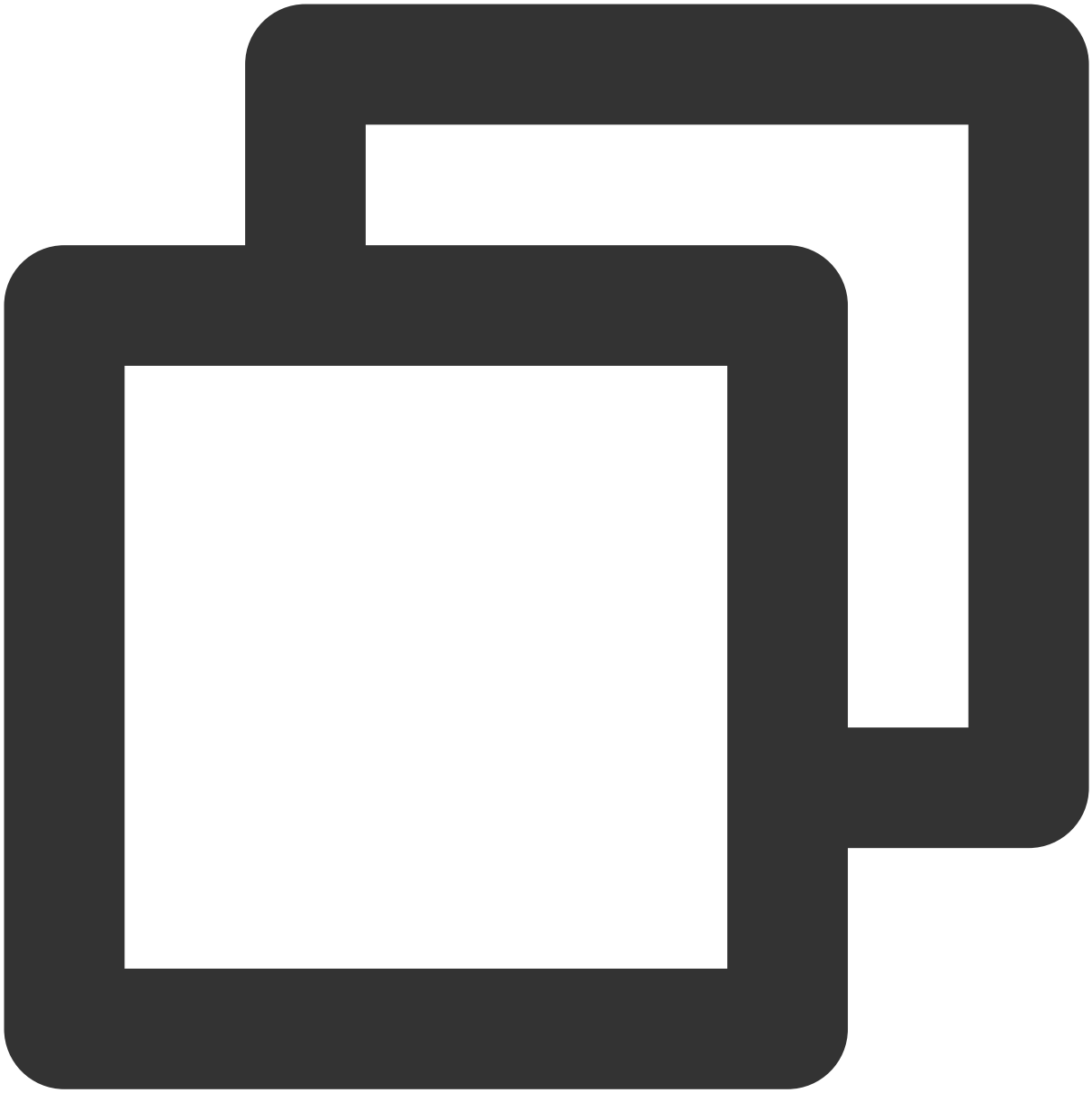
当 movable-view 小于 movable-area 时，movable-view 的移动范围是在 movable-area 内。

当 movable-view 大于 movable-area 时，movable-view 的移动范围必须包含 movable-area（x 轴方向和 y 轴方向分开考虑）。

示例代码：

WXML

JAVASCRIPT



```
<view class="container">

  <view class="page-body">
    <view class="page-section">
      <view class="page-section-title first">movable-view区域小于movable-area</view>
      <movable-area>
        <movable-view x="{{x}}" y="{{y}}" direction="all">text</movable-view>
      </movable-area>
    </view>
    <view class="btn-area">
      <button bindtap="tap" class="page-body-button" type="primary">点击移动到 (30px,
```

```
  </view>

  <view class="page-section">
    <view class="page-section-title">movable-view区域大于movable-area</view>
    <movable-area>
      <movable-view class="max" direction="all">text</movable-view>
    </movable-area>
  </view>

  <view class="page-section">
    <view class="page-section-title">只可以横向移动</view>
    <movable-area>
      <movable-view direction="horizontal">text</movable-view>
    </movable-area>
  </view>

  <view class="page-section">
    <view class="page-section-title">只可以纵向移动</view>
    <movable-area>
      <movable-view direction="vertical">text</movable-view>
    </movable-area>
  </view>

  <view class="page-section">
    <view class="page-section-title">可超出边界</view>
    <movable-area>
      <movable-view direction="all" out-of-bounds>text</movable-view>
    </movable-area>
  </view>

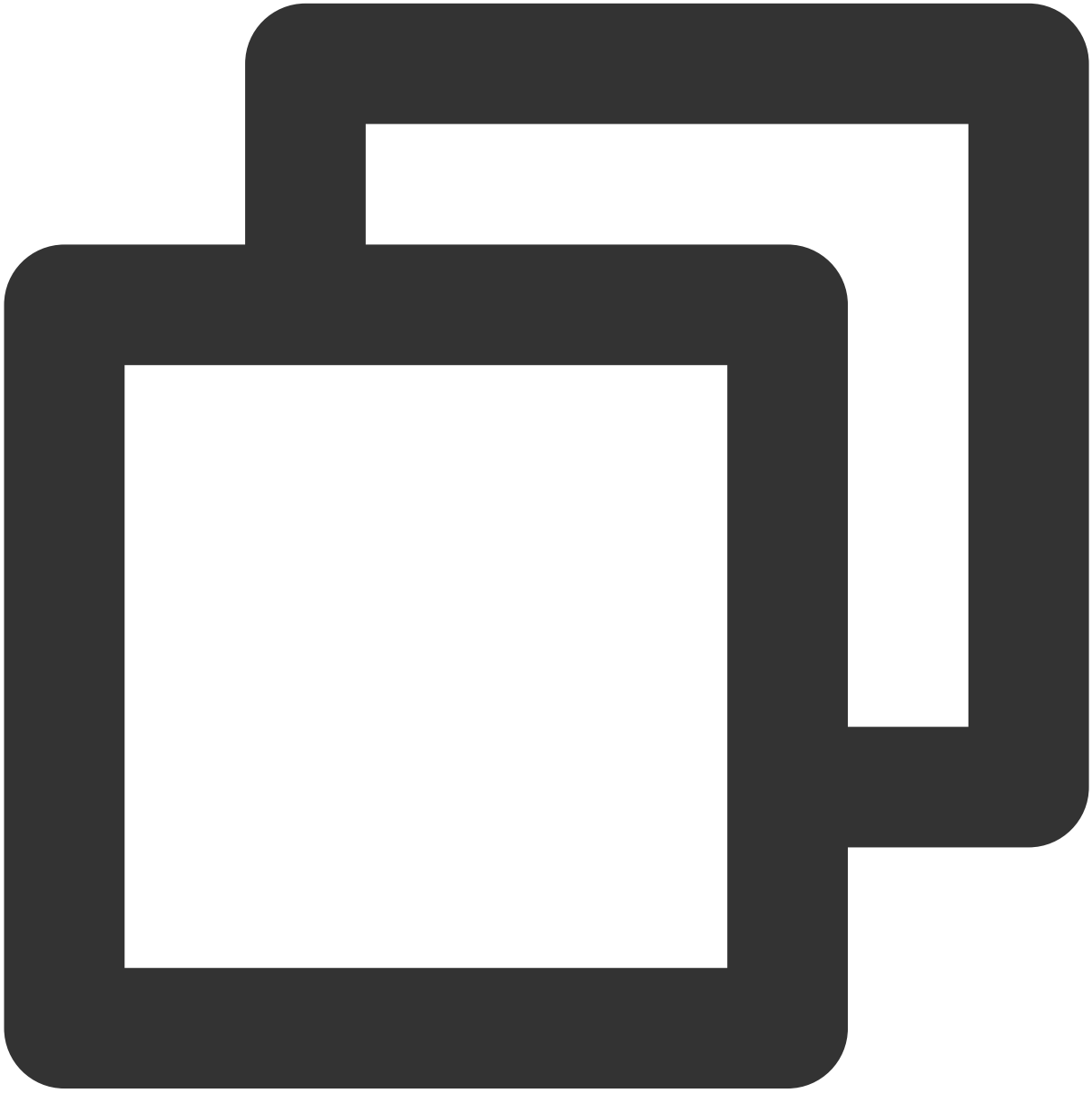
  <view class="page-section">
    <view class="page-section-title">带有惯性</view>
    <movable-area>
      <movable-view direction="all" inertia>text</movable-view>
    </movable-area>
  </view>

  <view class="page-section">
    <view class="page-section-title">可放缩</view>
    <movable-area scale-area>
      <movable-view direction="all" bindchange="onChange" bindscale="onScale" sca
    </movable-area>
  </view>

  <view class="btn-area">
    <button bindtap="tap2" class="page-body-button" type="primary">点击放大3倍</but
  </view>
```

```
  </view>

</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'movable-view',
      path: 'page/component/pages/movable-view/movable-view'
    }
  },
```

版权所有：腾讯云计算（北京）有限责任公司

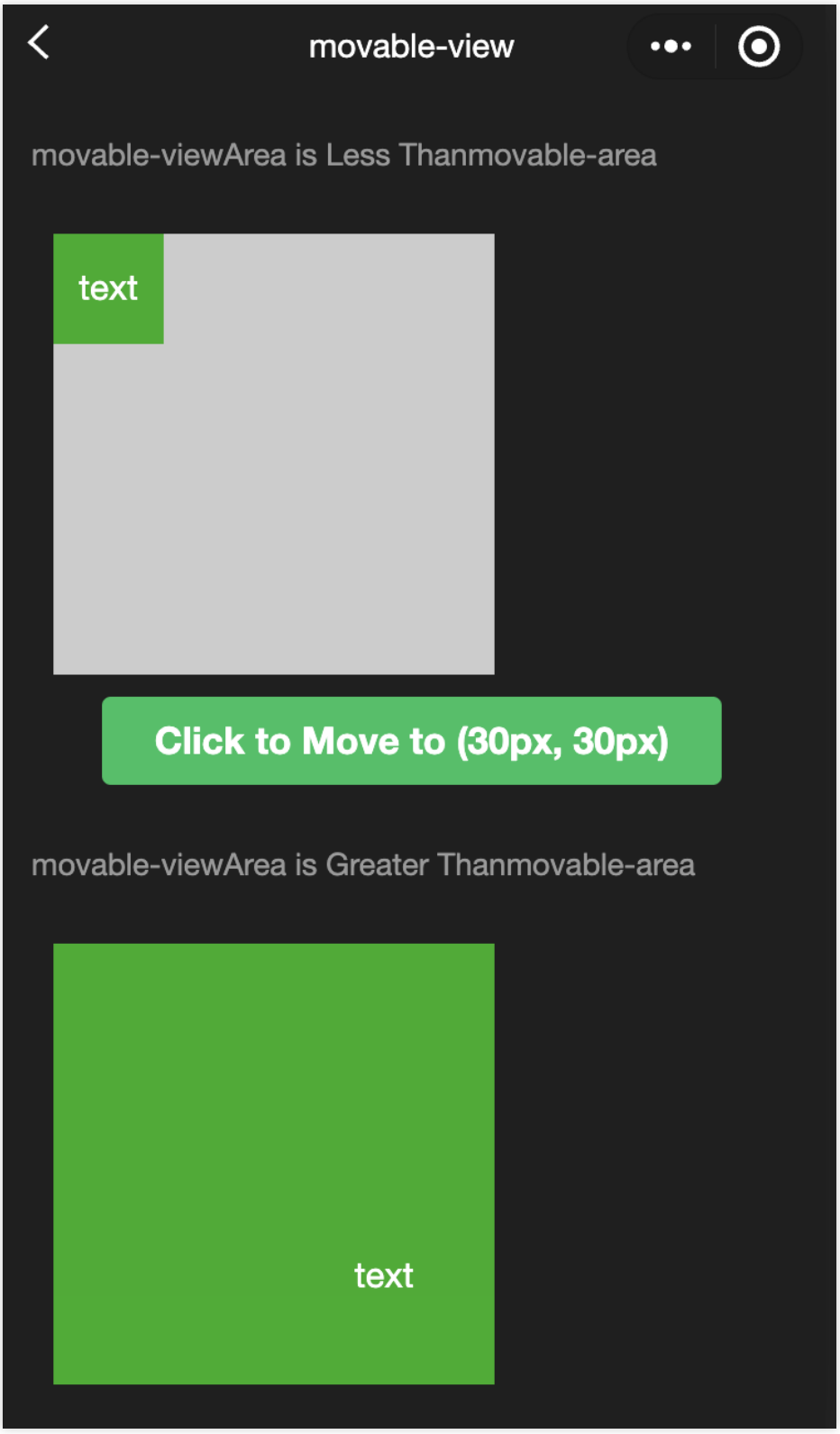
```
  data: {
    x: 0,
    y: 0,
    scale: 2,
  },

  tap() {
    this.setData({
      x: 30,
      y: 30
    })
  },

  tap2() {
    this.setData({
      scale: 3
    })
  },

  onChange(e) {
    console.log(e.detail)
  },

  onScale(e) {
    console.log(e.detail)
  }
})
```



movable-view

功能说明：可移动的视图容器，在页面中可以拖拽滑动。movable-view 必须在 movable-area 组件中，并且必须是直接子节点，否则不能移动。

参数及说明：

属性名	类型	默认值	说明
direction	string	none	movable-view 的移动方向，属性值有 all、vertical、horizontal、none
inertia	boolean	false	movable-view 是否带有惯性
out-of-bounds	boolean	false	超过可移动区域后，movable-view 是否还可以移动
x	number / string	0	定义 x 轴方向的偏移，如果 x 的值不在可移动范围内，会自动移动到可移动范围；改变 x 的值会触发动画
y	number / string	0	定义 y 轴方向的偏移，如果 y 的值不在可移动范围内，会自动移动到可移动范围；改变 y 的值会触发动画
damping	number	20	阻尼系数，用于控制 x 或 y 改变时的动画和过界回弹的动画，值越大移动越快
friction	number	2	摩擦系数，用于控制惯性滑动的动画，值越大摩擦力越大，滑动越快停止；必须大于0，否则会被设置成默认值
disabled	boolean	false	是否禁用
scale	boolean	false	是否支持双指缩放，默认缩放手势生效区域是在 movable-view 内
scale-min	number	0.5	定义缩放倍数最小值
scale-max	number	10	定义缩放倍数最大值
scale-value	number	1	定义缩放倍数，取值范围为0.5 - 10
animation	boolean	true	定义缩放倍数，取值范围为0.5 - 10
bindchange	eventhandle	-	拖动过程中触发的事件，event.detail = {x: x, y: y, source: source}，其中 source 表示产生移动的原因，值可为： `touch` ：拖动； `touch-out-of-bounds` ：超出移动范围； `out-of-bounds` ：超出移动范围后的回弹； `friction` ：惯性； 空字符串 ：setData；
bindscale	eventhandle	-	缩放过程中触发的事件，event.detail = {x: x, y: y,

			scale: scale}

除了基本事件外，movable-view 提供了两个特殊事件。

事件名	类型	触发条件
htouchmove	eventhandle	初次手指触摸后移动为横向的移动，如果 catch 此事件，则意味着 touchmove 事件也被 catch
vtouchmove	eventhandle	初次手指触摸后移动为纵向的移动，如果 catch 此事件，则意味着 touchmove 事件也被 catch

注意：

movable-view 必须设置 width 和 height 属性，不设置默认为10px。

movable-view 默认为绝对定位，top 和 left 属性为0px。

cover-image

功能说明：覆盖在原生组件之上的图片视图，可覆盖的原生组件同 cover-view，支持嵌套在 cover-view 里。

参数及说明：

属性名	类型	默认值	说明
src	string	-	图标路径，支持临时路径、网络地址； 暂不支持 `SVG` 和 `BASE64` 格式；
bindload	eventhandle	-	图片加载失败时触发
binderror	eventhandle	-	图片加载失败时触发

cover-view

功能说明：覆盖在原生组件之上的文本视图，可覆盖的原生组件包括map、video、canvas、camera、live-player、live-pusher，只支持嵌套cover-view、cover-image，可在cover-view中使用button。

参数及说明：

属性名	类型	默认值	说明
scroll-top	number / string	-	设置顶部滚动偏移量，仅在设置了 overflow-y: scroll 成为滚动元素后生效

版权所有：腾讯云计算（北京）有限责任公司

注意：

事件模型遵循冒泡模型，但不会冒泡到原生组件。

文本建议都套上 cover-view 标签，避免排版错误。

只支持基本的定位、布局、文本样式。不支持设置单边的 border、background-image、shadow、overflow: visible 等。

建议子节点不要溢出父节点。

支持使用 z-index 控制层级。

默认设置的样式有：white-space: nowrap; line-height: 1.2; display: block。

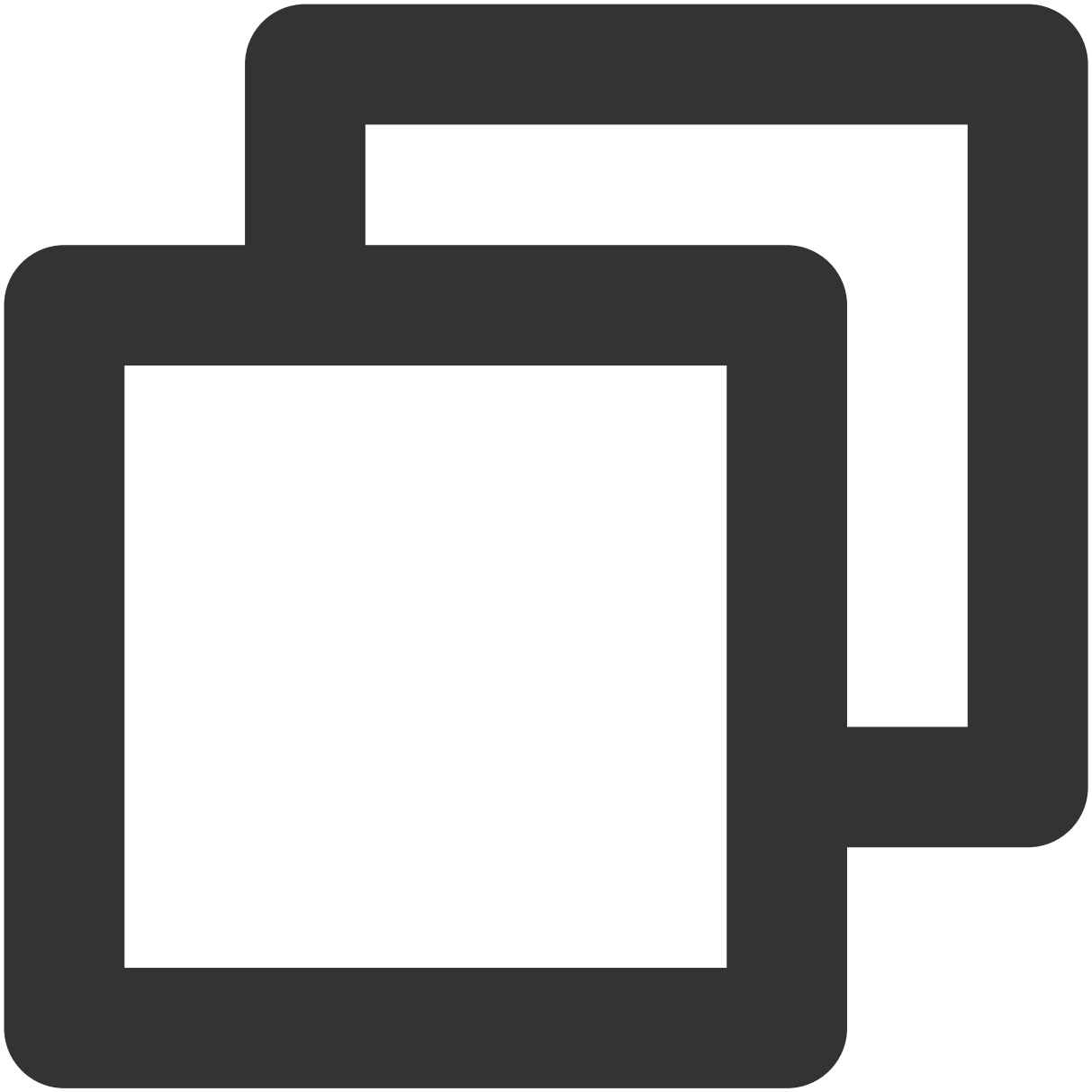
自定义组件嵌套 cover-view 时，自定义组件的 slot 及其父节点暂不支持通过 wx:if 控制显隐，否则会导致 cover-view 不显示。

示例代码：

WXML

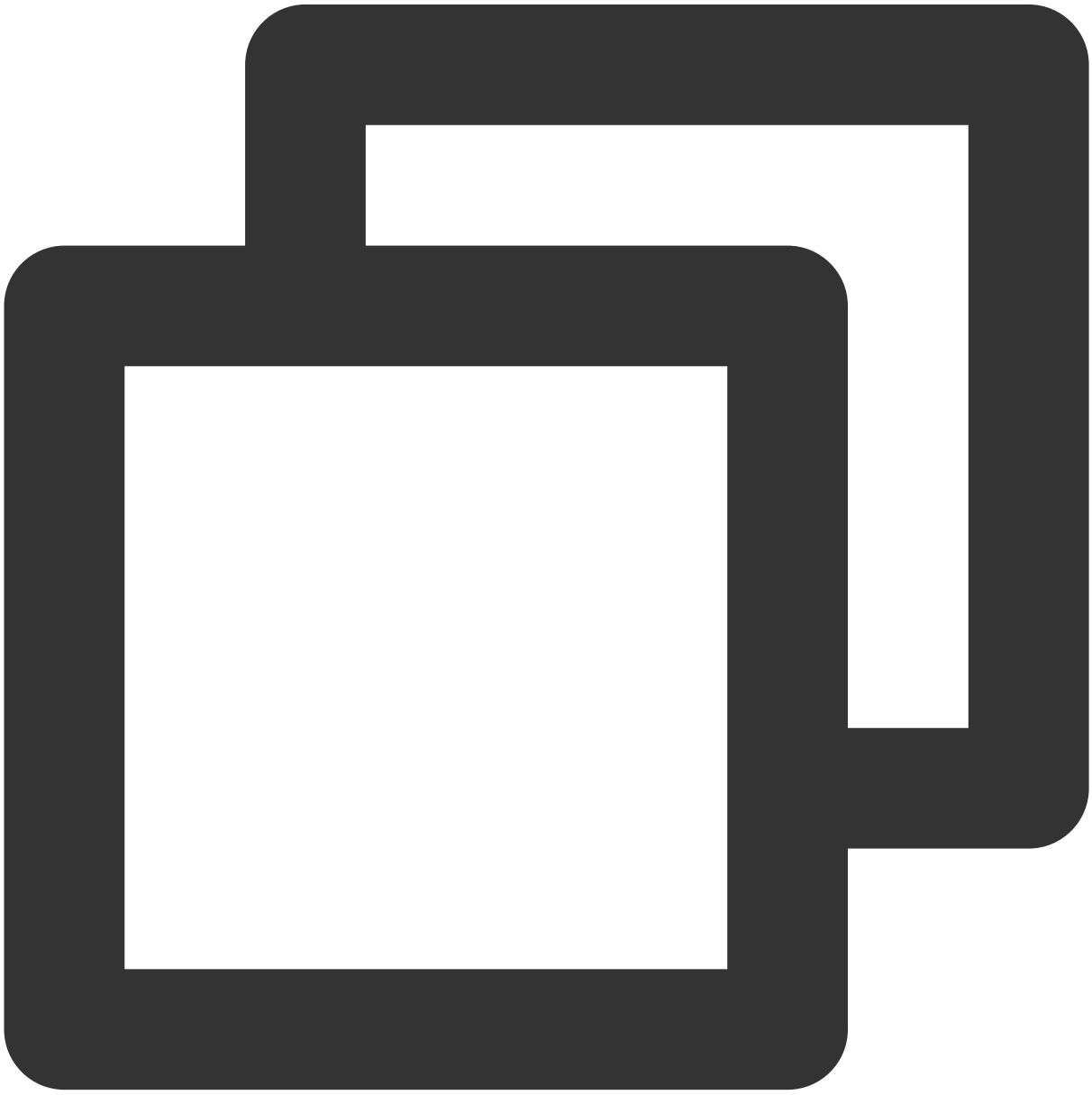
JAVASCRIPT

WXSS



```
<view class="container">
  <view class="page-body">
    <view class="page-section page-section-gap">
      <map
        style="width: 100%; height: 300px;"
        latitude="{{latitude}}"
        longitude="{{longitude}}"
      >
        <cover-view class="cover-view">
          <cover-view class="container">
            <cover-view class="flex-wrp" style="flex-direction:row;">
```

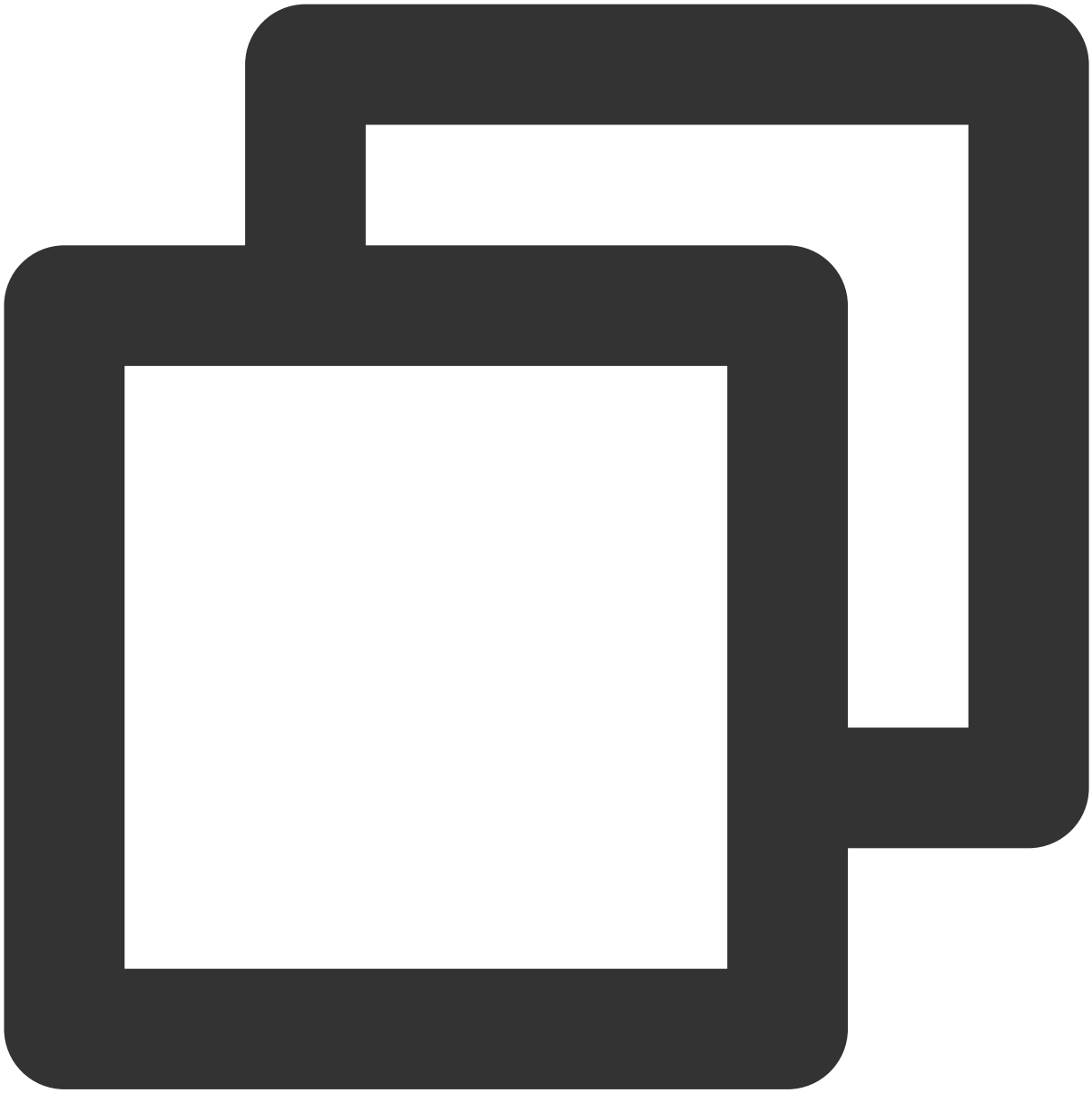
```
            <cover-view class="flex-item demo-text-1"></cover-view>
            <cover-view class="flex-item demo-text-2"></cover-view>
            <cover-view class="flex-item demo-text-3"></cover-view>
          </cover-view>
        </cover-view>
      </cover-view>
    </map>
  </view>
 </view>
</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'cover-view',
      path: 'page/component/pages/cover-view/cover-view'
    }
  },

  data: {
    latitude: 23.099994,
    longitude: 113.324520,
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司



```
.cover-view {
  position: absolute;
  top: calc(50% - 150rpx);
  left: calc(50% - 300rpx);
  /* opacity: .7; */
}

.flex-wrp{
  display:flex;
}
```

版权所有：腾讯云计算（北京）有限责任公司

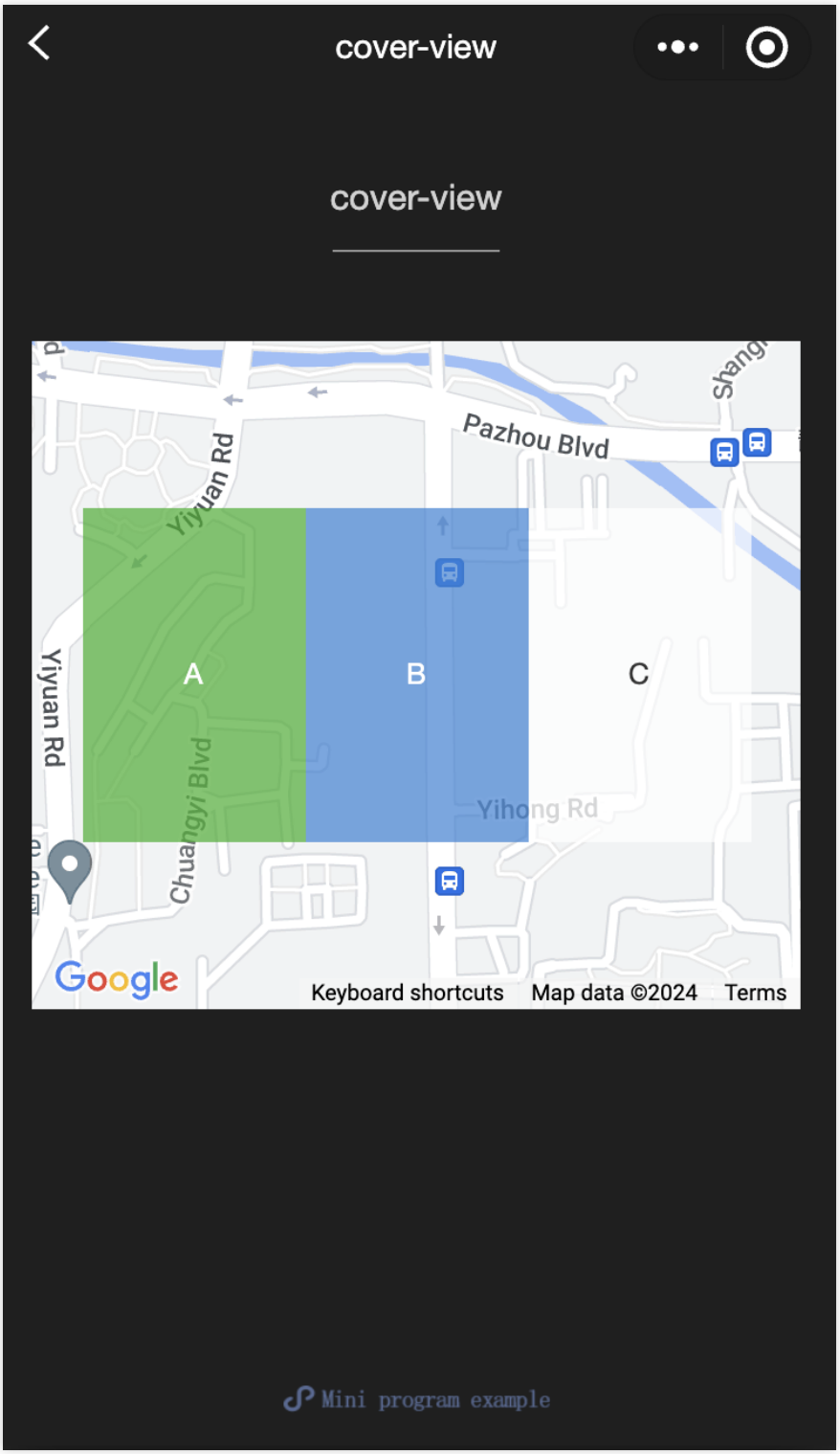
```
.flex-item{
  width: 200rpx;
  height: 300rpx;
  font-size: 26rpx;
}

.demo-text-1 {
  background: rgba(26, 173, 25, 0.7);
}

.demo-text-2 {
  background: rgba(39, 130, 215, 0.7);
}

.demo-text-3 {
  background: rgba(255, 255, 255, 0.7);
}
```

版权所有：腾讯云计算（北京）有限责任公司


cover-view
cover-view
Pazhou Blvd
Yiyuan Rd
Yiyuan Rd
Chuangyi Blvd
Yihong Rd
A
B
C
Google
Keyboard shortcuts
Map data ©2024
Terms
Mini program example

基础内容

最近更新时间：2024-07-12 18:17:56

icon

功能说明：图标组件。

参数及说明：

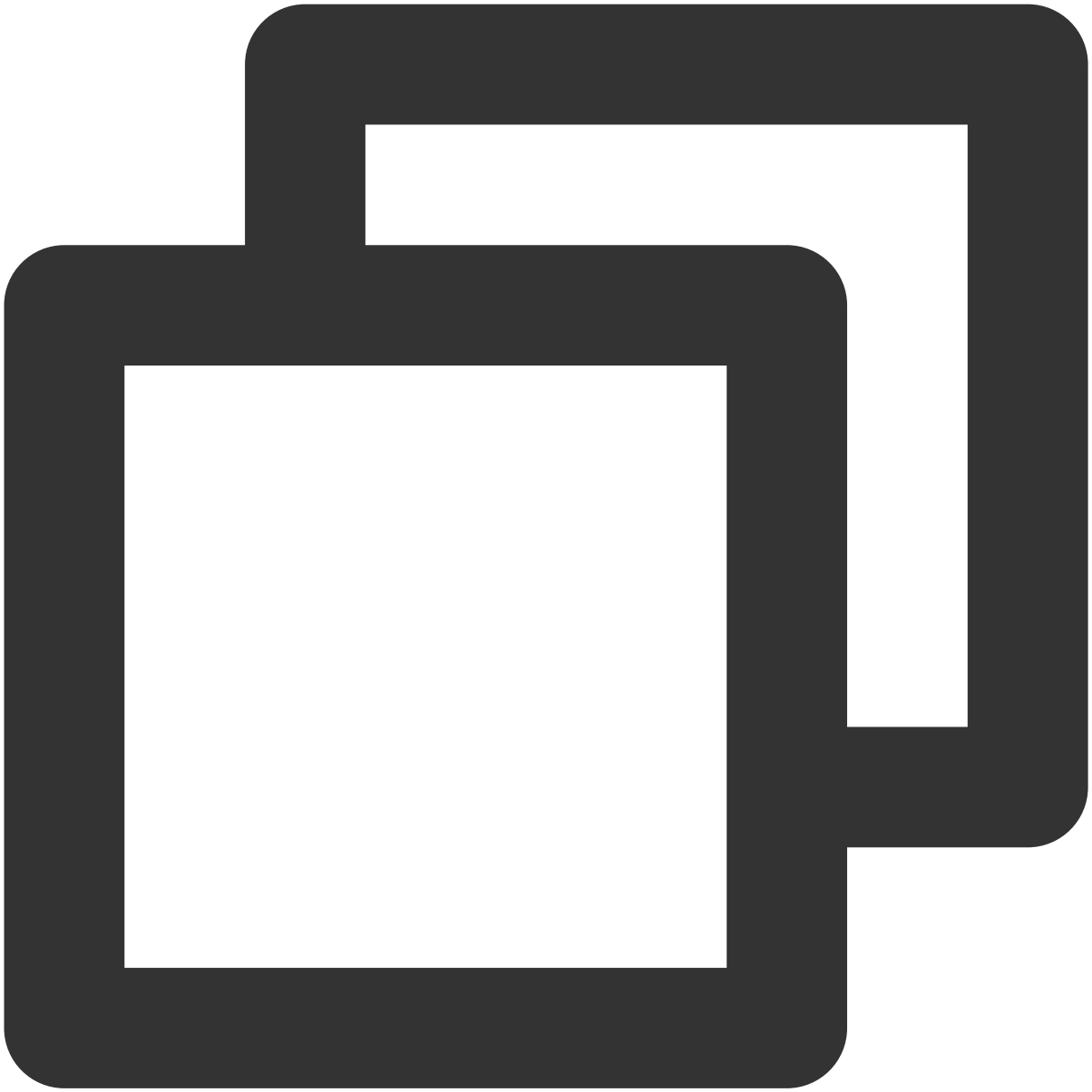
属性	类型	默认值	说明
type	string	-	icon 的类型，有效值：success，success_no_circle，info，warn，waiting，cancel，download，search，clear
size	umber / string	23	icon 的大小，单位默认为 px
color	string	-	icon 的颜色，同 css 的 color

示例代码：

WXML

JAVASCRIPT

版权所有：腾讯云计算（北京）有限责任公司



```
<view class="container">
  <view class="icon-box">
    <icon class="icon-box-img" type="success" size="93"></icon>
    <view class="icon-box-ctn">
      <view class="icon-box-title">成功</view>
      <view class="icon-box-desc">用于表示操作顺利完成</view>
    </view>
  </view>
  <view class="icon-box">
    <icon class="icon-box-img" type="info" size="93"></icon>
    <view class="icon-box-ctn">
```

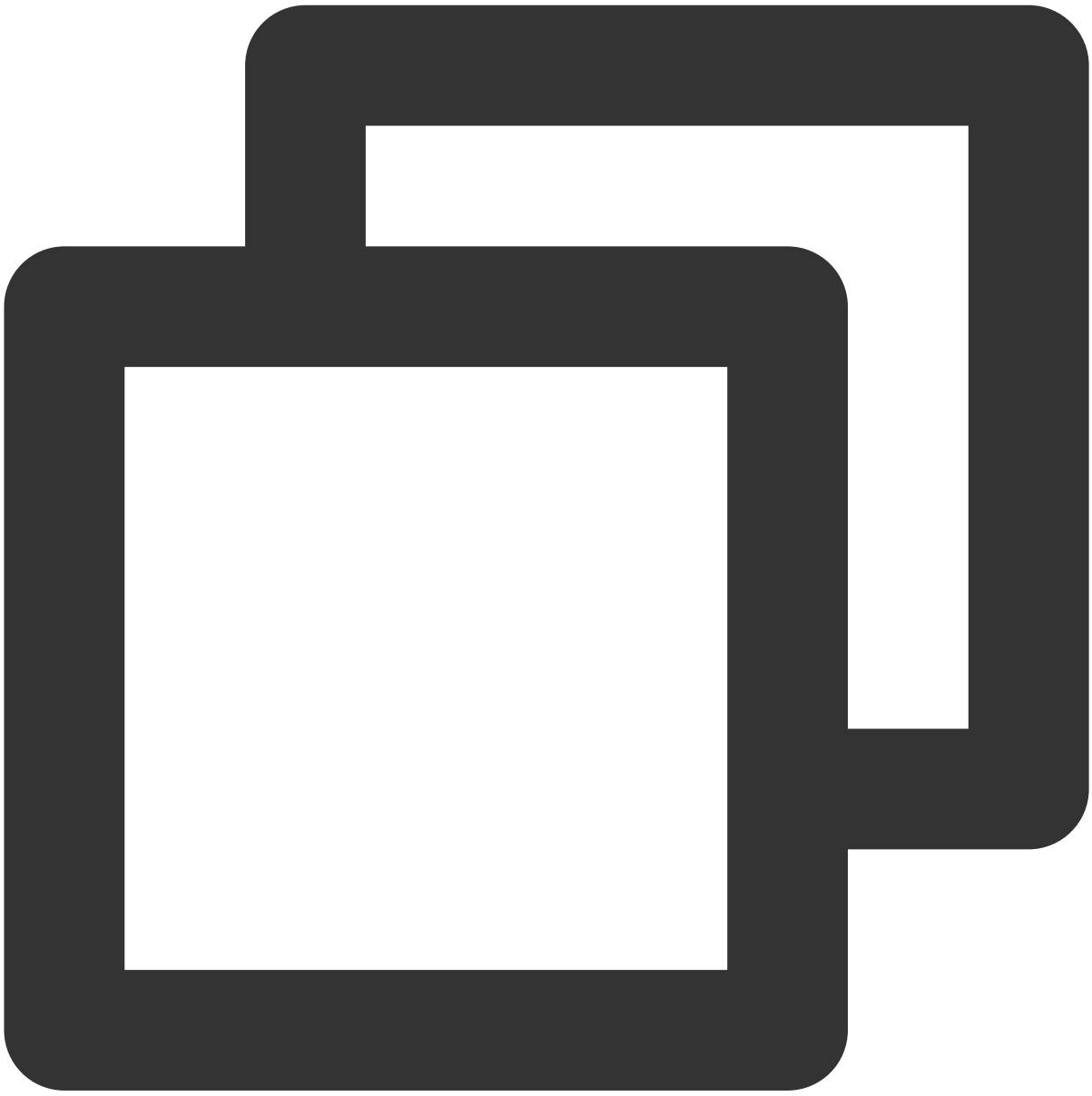
版权所有：腾讯云计算（北京）有限责任公司

```
      <view class="icon-box-title">提示</view>
      <view class="icon-box-desc">用于表示信息提示；也常用于缺乏条件的操作拦截，提示用户所需
    </view>
  </view>
  <view class="icon-box">
    <icon class="icon-box-img" type="warn" size="93" color="#C9C9C9"></icon>
    <view class="icon-box-ctn">
      <view class="icon-box-title">普通警告</view>
      <view class="icon-box-desc">用于表示操作后将引起一定后果的情况；也用于表示由于系统原因
    </view>
  </view>
  <view class="icon-box">
    <icon class="icon-box-img" type="warn" size="93"></icon>
    <view class="icon-box-ctn">
      <view class="icon-box-title">强烈警告</view>
      <view class="icon-box-desc">用于表示由于用户原因造成的负向结果；也用于表示操作后将引起
    </view>
  </view>
  <view class="icon-box">
    <icon class="icon-box-img" type="waiting" size="93"></icon>
    <view class="icon-box-ctn">
      <view class="icon-box-title">等待</view>
      <view class="icon-box-desc">用于表示等待，告知用户结果需等待</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="success_no_circle" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">多选控件图标_已选择</view>
      <view class="icon-box-desc">用于多选控件中，表示已选择该项目</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="circle" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">多选控件图标_未选择</view>
      <view class="icon-box-desc">用于多选控件中，表示该项目可被选择，但还未选择</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="warn" size="23"></icon>
    </view>
```

```
    <view class="icon-box-ctn">
      <view class="icon-box-title">错误提示</view>
      <view class="icon-box-desc">用于在表单中表示出现错误</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="success" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">单选控件图标_已选择</view>
      <view class="icon-box-desc">用于单选控件中，表示已选择该项目</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="download" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">下载</view>
      <view class="icon-box-desc">用于表示可下载</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="info_circle" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">提示</view>
      <view class="icon-box-desc">用于在表单中表示有信息提示</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="cancel" size="23"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">停止或关闭</view>
      <view class="icon-box-desc">用于在表单中，表示关闭或停止</view>
    </view>
  </view>
  <view class="icon-box">
    <view class="icon-small-wrp">
      <icon class="icon-small" type="search" size="14"></icon>
    </view>
    <view class="icon-box-ctn">
      <view class="icon-box-title">搜索</view>
```

版权所有：腾讯云计算（北京）有限责任公司

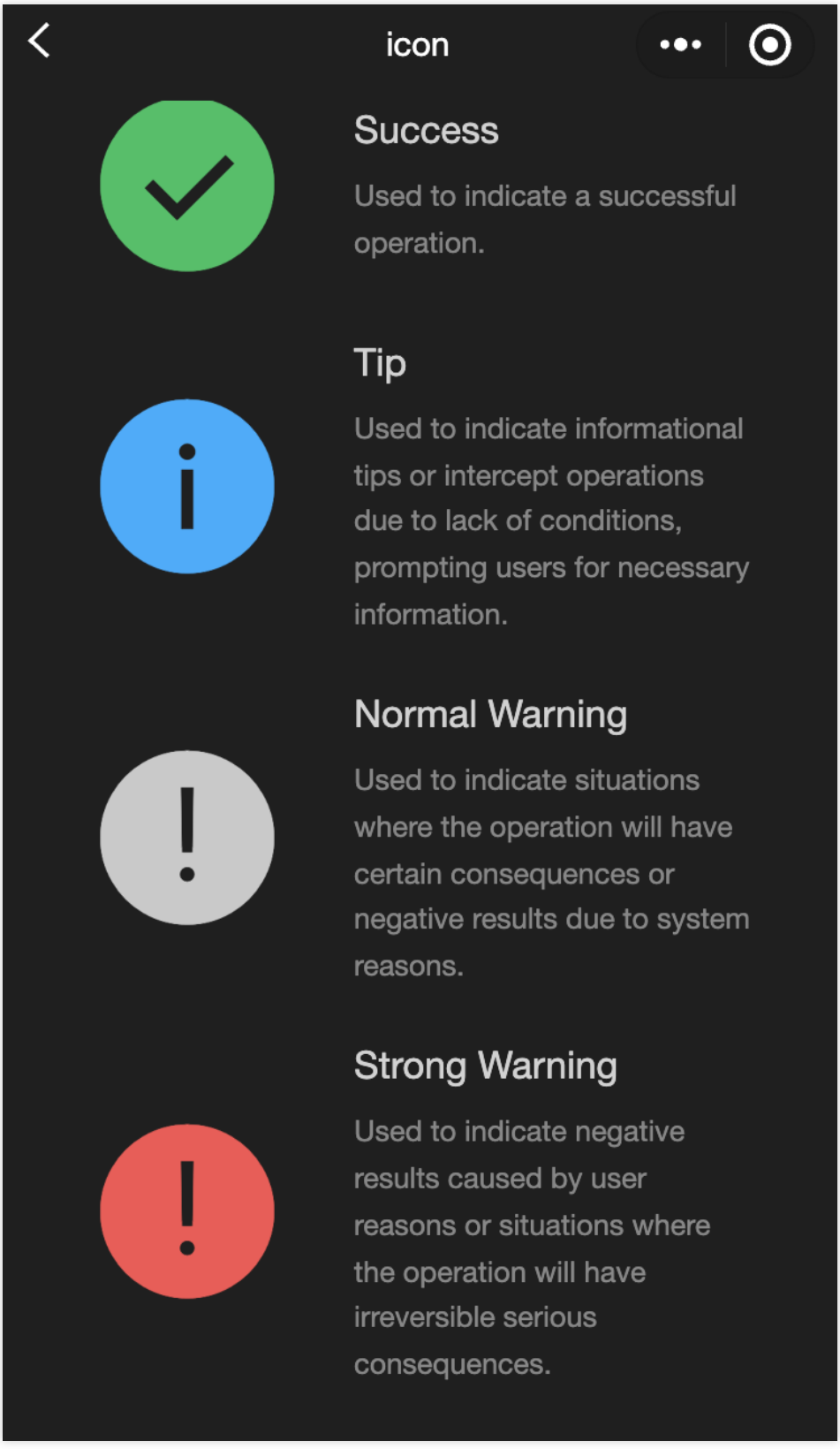
```
      <view class="icon-box-desc">用于搜索控件中，表示可搜索</view>
    </view>
  </view>
</view>
```



```
Page({
  data: {
    iconSize: [20, 30, 40, 50, 60, 70],
    iconColor: [
      'red', 'orange', 'yellow', 'green', 'rgb(0,255,255)', 'blue', 'purple'
    ],
```

```
    iconType: [
      'success', 'success_no_circle', 'info', 'warn', 'waiting', 'cancel', 'downloa
    ]
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司



progress

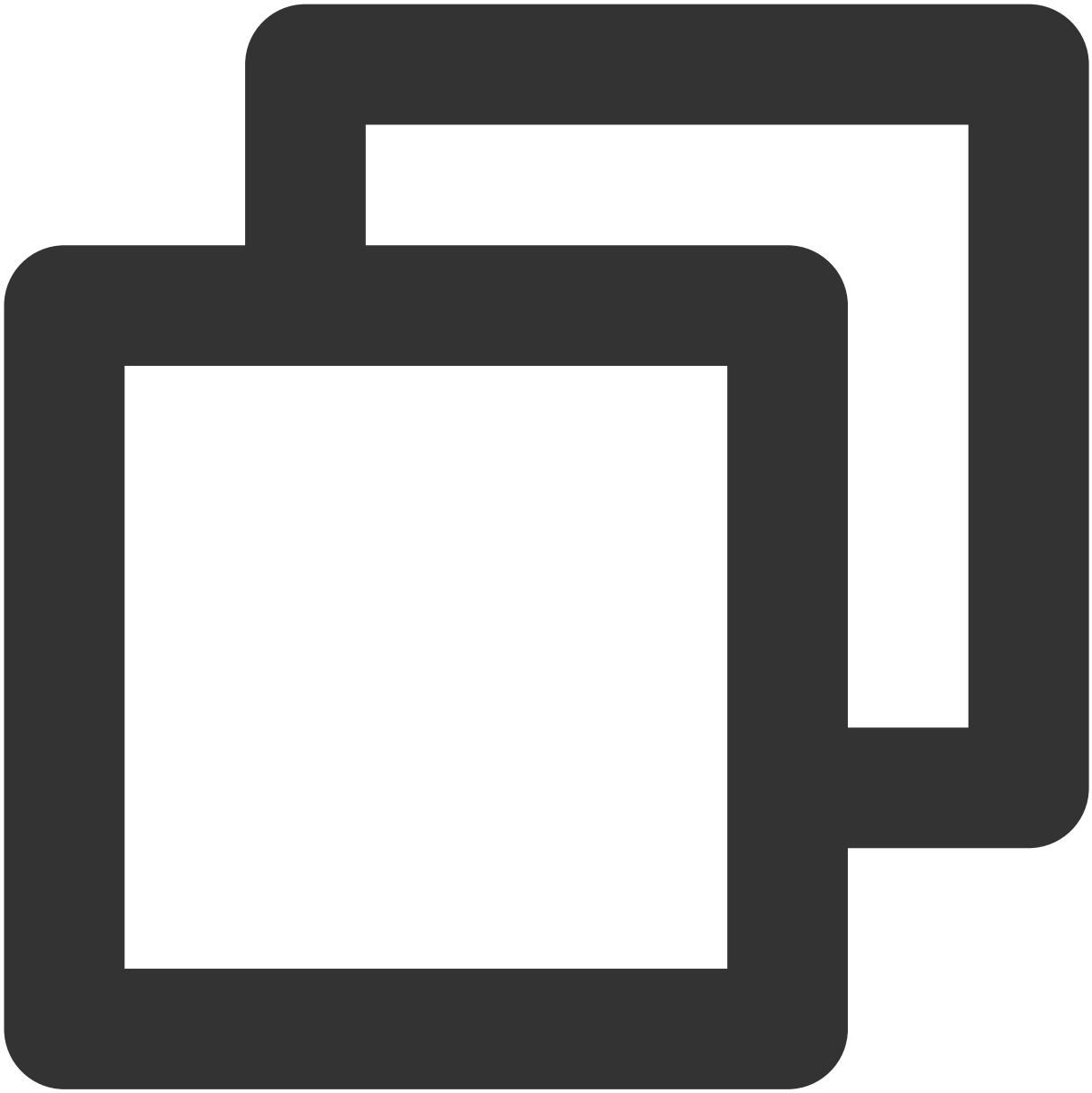
功能说明：进度条。

参数及说明：

属性	类型	默认值	说明
percent	float	-	百分比0~100
show-info	boolean	false	在进度条右侧显示百分比
border-radius	number / string	0	圆角大小，单位：px
font-size	number / string	16	右侧百分比字体大小，单位：px
stroke-width	number / string	6	进度条线的宽度
color	color	#09BB07	进度条颜色（请使用 activeColor）
activeColor	color	#09BB07	已选择的进度条的颜色
backgroundColor	color	#EBEBEB	未选择的进度条的颜色
active	boolean	false	进度条从左往右的动画
active-mode	string	backwards	backwards: 动画从头播； forwards：动画从上次结束点接着播；
bindactiveend	eventhandle	-	动画完成事件

示例代码：

WXML



```
<view class="progress-box">
  <progress percent="20" show-info stroke-width="3"/>
</view>

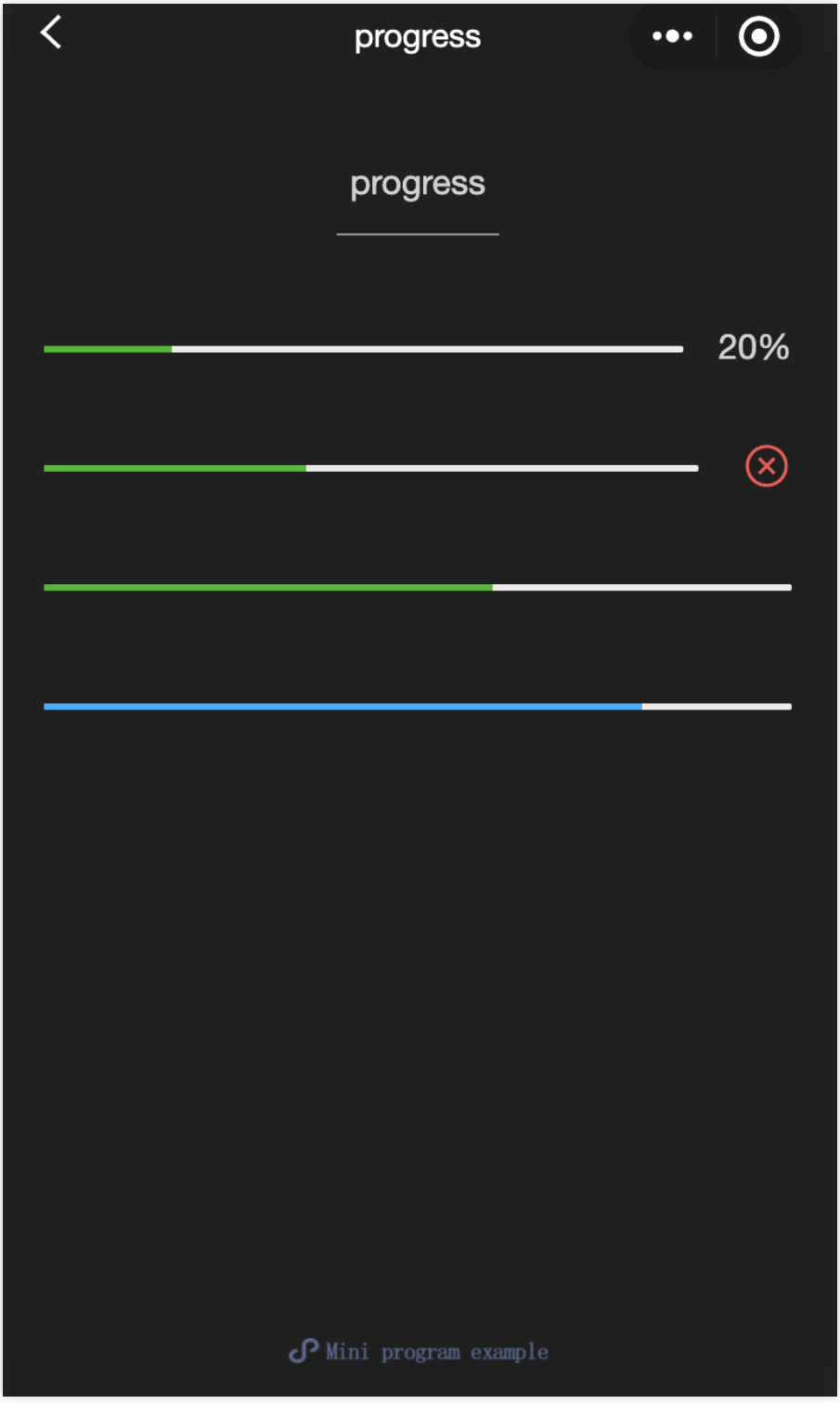
<view class="progress-box">
  <progress percent="40" active stroke-width="3" />
  <icon class="progress-cancel" type="cancel"></icon>
</view>

<view class="progress-box">
  <progress percent="60" active stroke-width="3" />
```

版权所有：腾讯云计算（北京）有限责任公司

```
</view>

<view class="progress-box">
  <progress percent="80" color="#10AEFF" active stroke-width="3" />
</view>
```



text

功能说明：文本。

参数及说明：

属性	类型	默认值	说明
selectable	boolean	false	文本是否可选
space	string	-	显示连续空格： ensp：中文字符空格一半大小 emsp：中文字符空格大小 nbsp：根据字体设置的空格大小
decode	boolean	false	是否解码

注意：

decode可以解析的有 `< > & '    ` 。

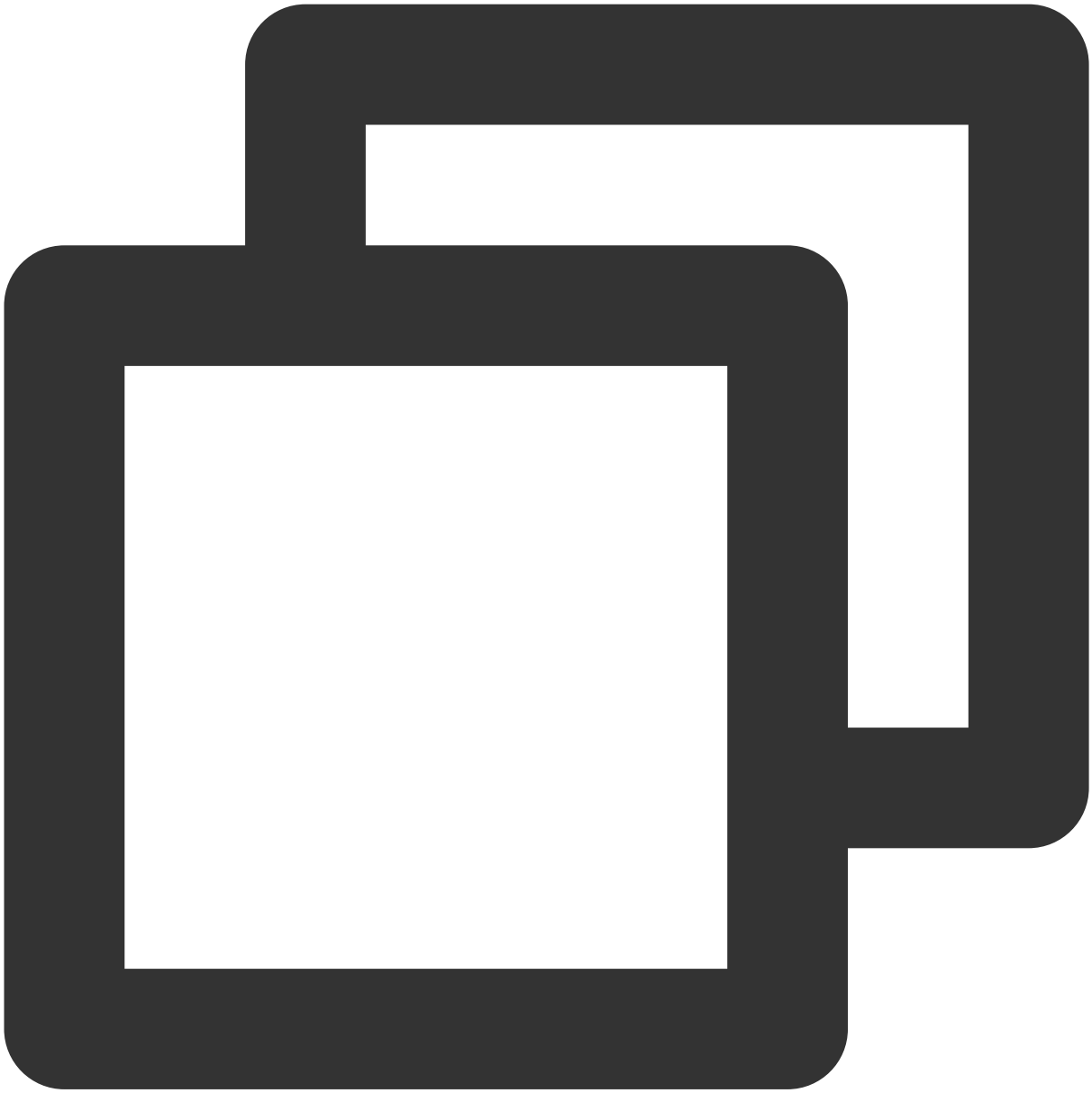
各个操作系统的空格标准并不一致。

text 组件内只支持 text 嵌套。

除了文本节点以外的其他节点都无法长按选中。

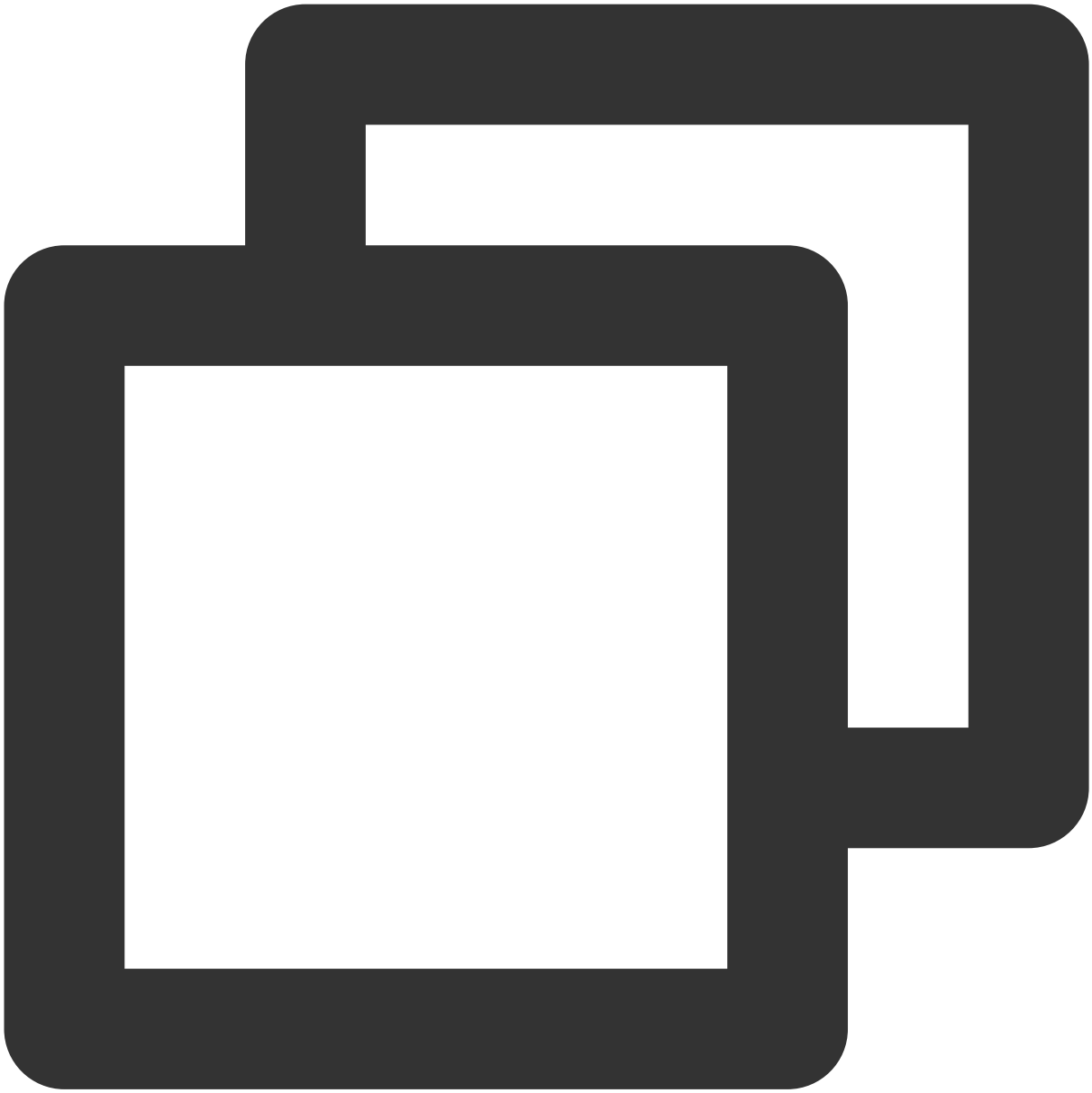
WXML

JAVASCRIPT



```
<view class="container">
  <view class="page-body">
    <view class="page-section page-section-spacing">
      <view class="text-box" scroll-y="true" scroll-top="{{scrollTop}}">
        <text>{{text}}</text>
      </view>
      <button disabled="{{!canAdd}}" bindtap="add">add line</button>
      <button disabled="{{!canRemove}}" bindtap="remove">remove line</button>
    </view>
  </view>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
const texts = [
    'lin1: text1',
    'lin2: text2',
    'lin3: text3',
    'lin4: text4',
    'lin5: text5',
    'lin6: text6',
    'lin7: text7',
    'lin8: text8',
    'lin9: text9',
```

版权所有：腾讯云计算（北京）有限责任公司

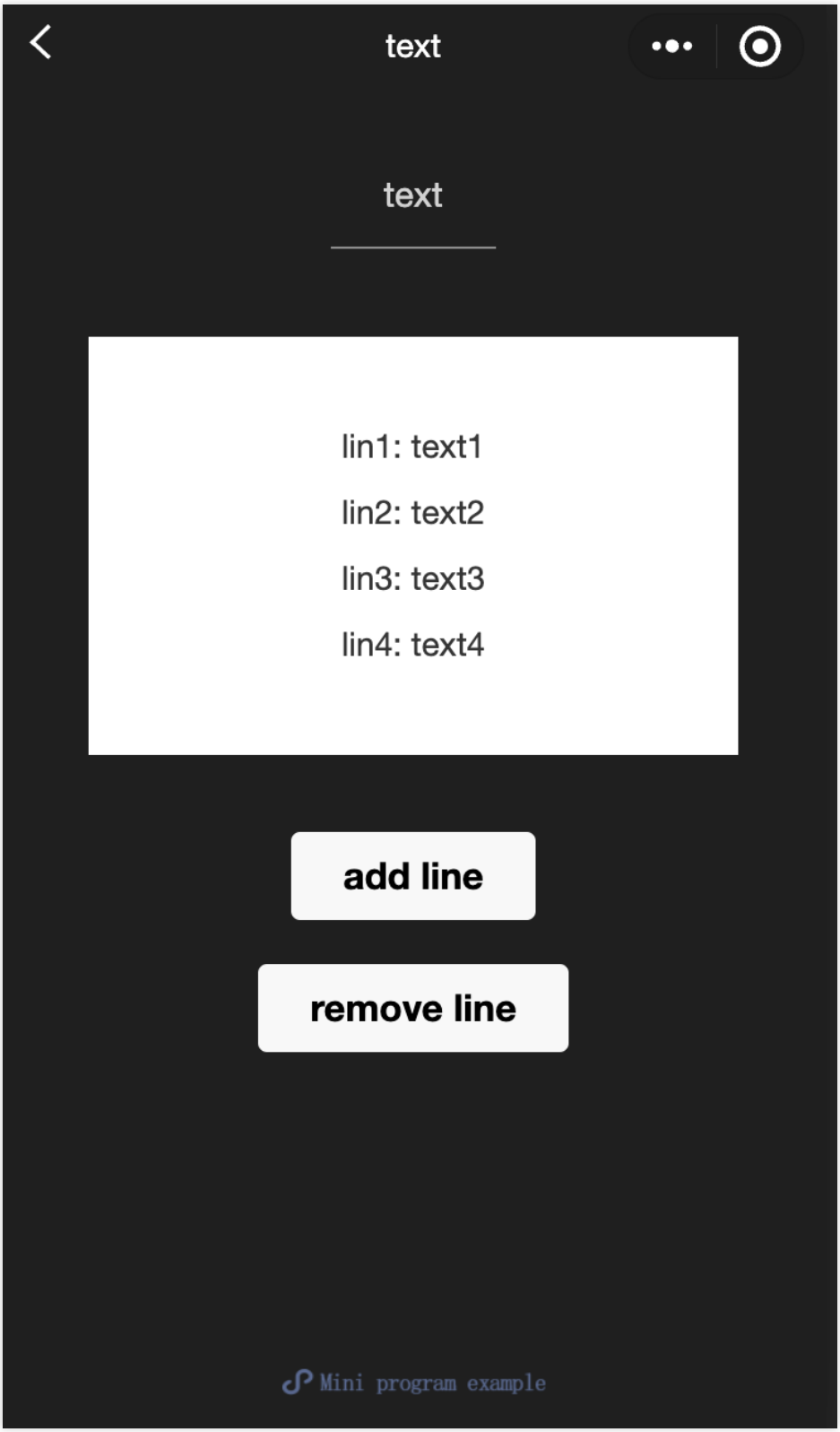
```
    'lin10: text10',
    'lin11: text11',
    'lin12: text12',
    '......'
  ]

Page({
  onShareAppMessage() {
    return {
      title: 'text',
      path: 'page/component/pages/text/text'
    }
  },

  data: {
    text: '',
    canAdd: true,
    canRemove: false
  },
  extraLine: [],

  add() {
    this.extraLine.push(texts[this.extraLine.length % 12])
    this.setData({
      text: this.extraLine.join('\\n'),
      canAdd: this.extraLine.length < 12,
      canRemove: this.extraLine.length > 0
    })
    setTimeout(() => {
      this.setData({
        scrollTop: 99999
      })
    }, 0)
  },
  remove() {
    if (this.extraLine.length > 0) {
      this.extraLine.pop()
      this.setData({
        text: this.extraLine.join('\\n'),
        canAdd: this.extraLine.length < 12,
        canRemove: this.extraLine.length > 0,
      })
    }
    setTimeout(() => {
      this.setData({
        scrollTop: 99999
      })
```

```
      }, 0)
    }
})
```


text
text
lin1: text1
lin2: text2
lin3: text3
lin4: text4
add line
remove line
Mini program example

rich-text

功能说明：富文本。

参数及说明：

属性	类型	默认值	说明
nodes	array / string	[]	节点列表 / HTML String
space	string	-	显示连续空格： ensp：中文字符空格一半大小 emsp：中文字符空格大小 nbsp：根据字体设置的空格大小

支持默认事件，包括：`tap`、`touchstart`、`touchmove`、`touchcancel`、`touchend` 和 `longtap`。

nodes

nodes 属性推荐使用 Array 类型，由于组件会将 String 类型转换为 Array 类型，因而性能会有所下降。现支持两种节点，通过 type 来区分，分别是元素节点和文本节点，默认是元素节点，在富文本区域里显示的 HTML 节点。

元素节点：type = node

属性	说明	类型	必填	备注
name	标签名	string	是	支持部分受信任的 HTML 节点
attrs	属性	object	否	支持部分收信人的属性，遵循 Pascal 命名法
children	子节点列表	array	否	结构和 nodes 一致

文本节点：type=text

属性	说明	类型	必填	备注
text	文本	string	是	支持 entities

受信任的 HTML 节点及属性

全局支持 class 和 style 属性，**不支持 id 属性。**

节点	属性	节点	属性
a	-	img	alt,src,height,width
abbr	-	ins	-
b	-	label	-
br	-	legend	-

版权所有：腾讯云计算（北京）有限责任公司

code	-	li	-
col	span,width	ol	start,type
colgroup	span,width	p	-
dd	-	q	-
del	-	span	-
div	-	strong	-
dl	-	sub	-
dt	-	sup	-
em	-	table	width
fieldset	-	tbody	-
h1	-	td	colspan,height,rowspan,width
h2	-	tfoot	-
h3	-	th	colspan,height,rowspan,width
h4	-	thead	-
h5	-	tr	-
h6	-	ul	-
hr	-	-	-

注意：

nodes 不推荐使用 String 类型，性能会有所下降。

rich-text 组件内屏蔽所有节点的事件。

attrs 属性不支持 id，支持 class。

name 属性大小写不敏感。

如果使用了不受信任的 HTML 节点，该节点及其所有子节点将会被移除。

img 标签仅支持网络图片。

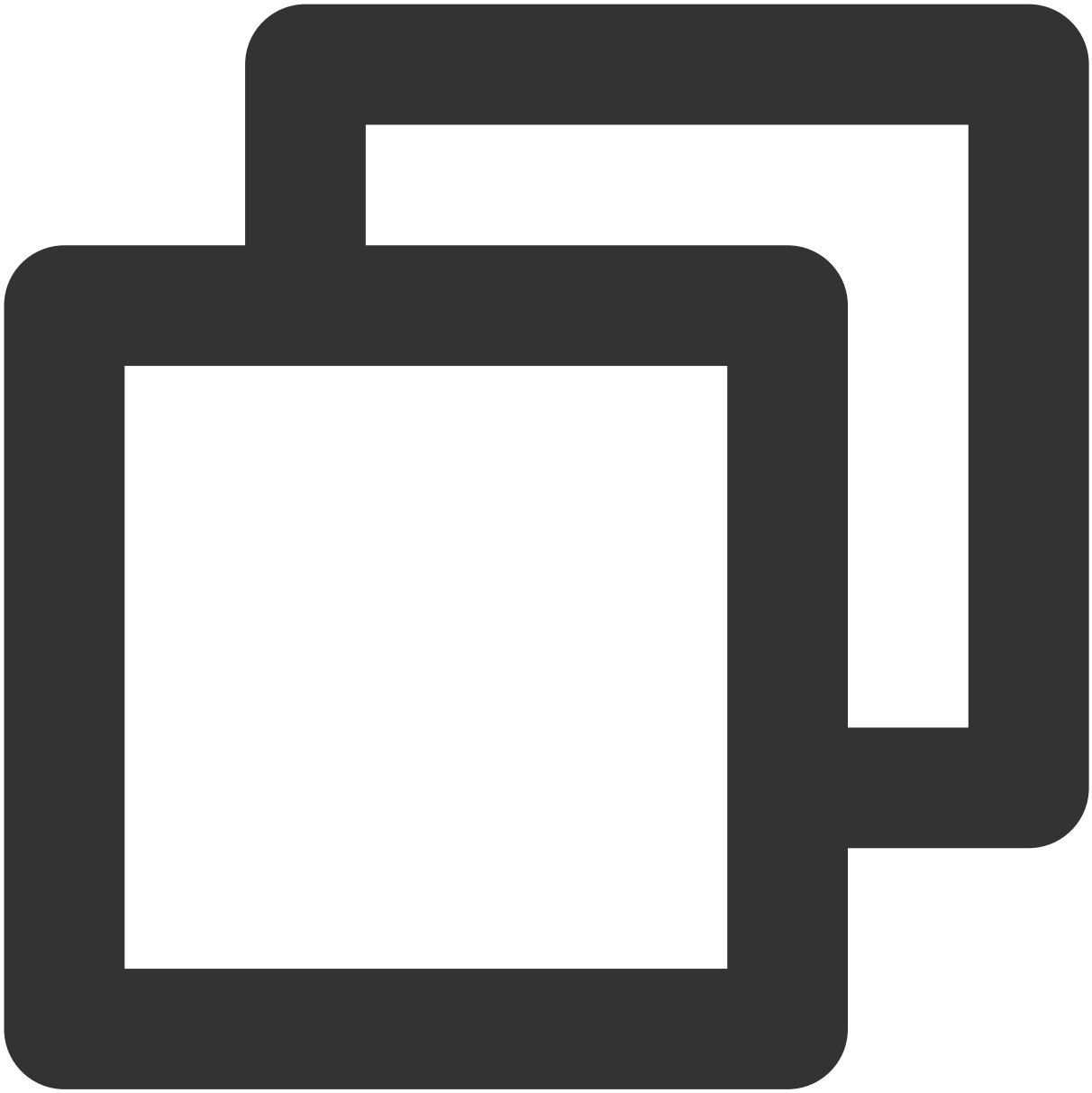
如果在自定义组件中使用 rich-text 组件，那么仅自定义组件的 wxss 样式对 rich-text 中的 class 生效。

示例代码

WXML

JAVASCRIPT

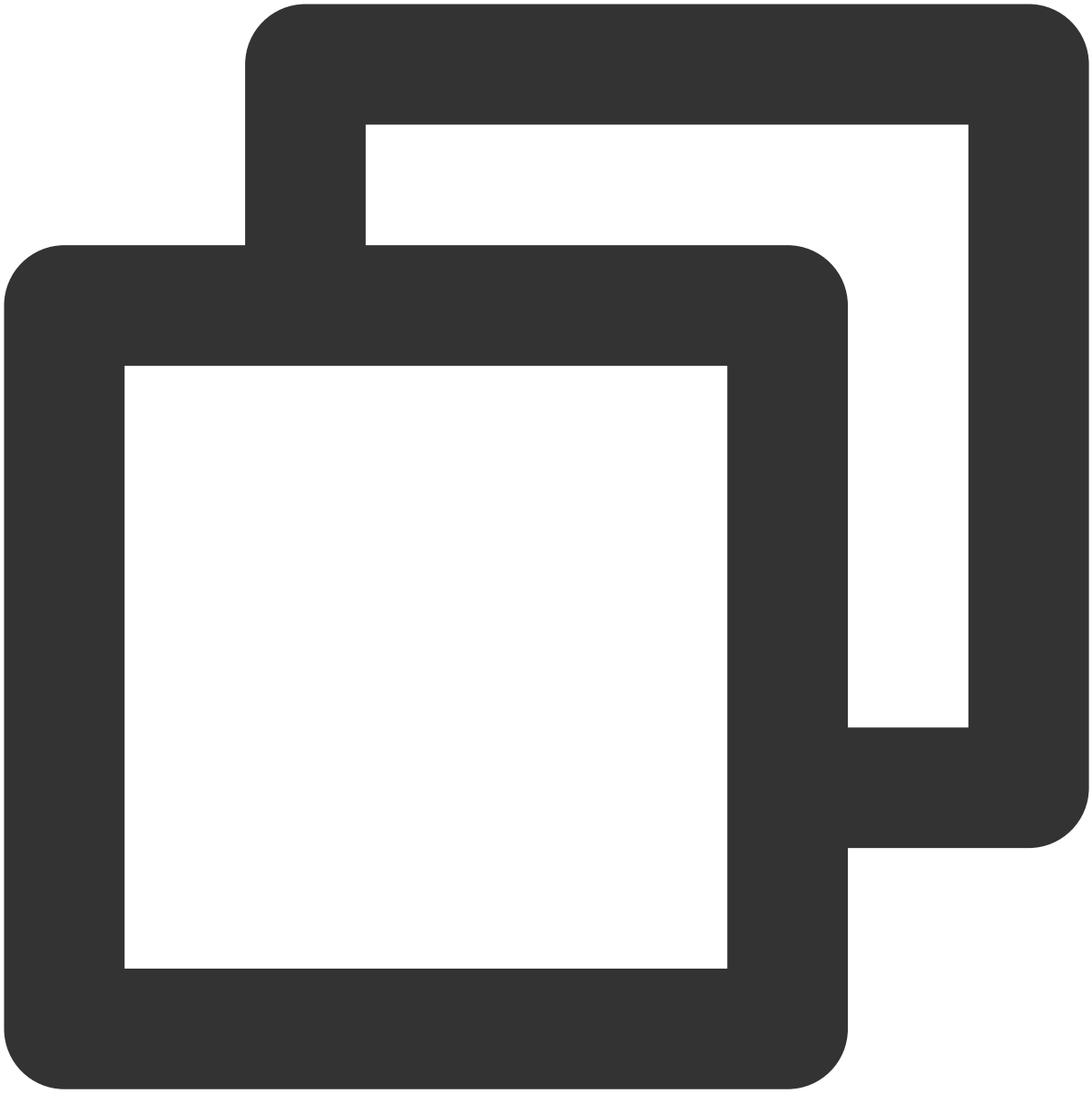
版权所有：腾讯云计算（北京）有限责任公司



```
<view class="container">
  <view class="page-body">
    <view class="page-section">
      <view class="page-section-title">通过HTML String渲染</view>
      <view class="page-content">
        <scroll-view scroll-y>{{htmlSnip}}</scroll-view>
        <button style="margin: 30rpx 0" type="primary" bindtap="renderHtml">渲染HTMI
        <block wx:if="{{renderedByHtml}}">
          <rich-text nodes="{{htmlSnip}}"></rich-text>
        </block>
      </view>
```

```
    </view>

    <view class="page-section">
      <view class="page-section-title">通过节点渲染</view>
      <view class="page-content">
        <scroll-view scroll-y>{{nodeSnip}}</scroll-view>
        <button style="margin: 30rpx 0" type="primary" bindtap="renderNode">渲染Node
        <block wx:if="{{renderedByNode}}">
          <rich-text nodes="{{nodes}}"></rich-text>
        </block>
      </view>
    </view>
  </view>
</view>
```



```
const htmlSnip =
`<div class="div_class">
  <h1>Title</h1>
  <p class="p">
    Life is <i>like</i> a box of
    <b> chocolates</b>.
  </p>
</div>
`

const nodeSnip =
```

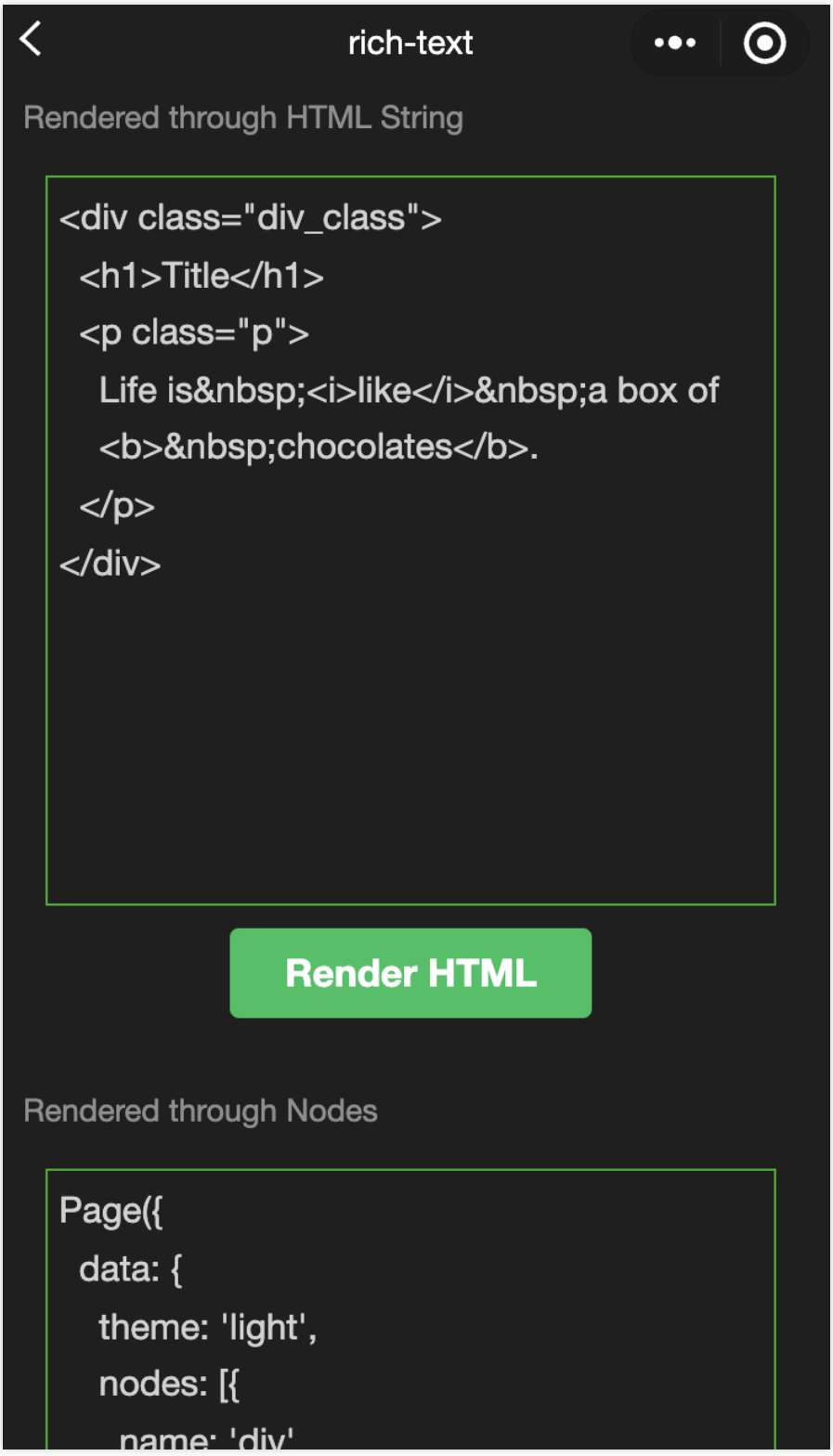
```
`Page({
  data: {
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: red;'
      },
      children: [{
        type: 'text',
        text: 'You never know what you're gonna get.'
      }]
    }]
  }
})
`

Page({
  onShareAppMessage() {
    return {
      title: 'rich-text',
      path: 'page/component/pages/rich-text/rich-text'
    }
  },

  data: {
    htmlSnip,
    nodeSnip,
    renderedByHtml: false,
    renderedByNode: false,
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: #1AAD19;'
      },
      children: [{
        type: 'text',
        text: 'You never know what you\\'re gonna get.'
      }]
    }]
  },
  renderHtml() {
    this.setData({
      renderedByHtml: true
    })
  },
```

```
  renderNode() {
    this.setData({
      renderedByNode: true
    })
  },
  enterCode(e) {
    console.log(e.detail.value)
    this.setData({
      htmlSnip: e.detail.value
    })
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司


rich-text
Rendered through HTML String

<h1>Title</h1>

Life is like a box of
 chocolates.

Render HTML
Rendered through Nodes
Page({
data: {
theme: 'light',
nodes: [{
name: 'div'

地图组件

最近更新时间：2024-07-12 18:17:08

map

功能说明：地图。该组件是 原生组件，使用时请注意相关限制，相关 API 可参见 wx.createMapContext。

支持度说明：

系统地图：仅 iOS 支持。

谷歌地图：Android 支持、IDE 支持。

华为地图：仅 Android 支持。

腾讯地图：IOS支持、Android 支持、IDE 支持。

参数及说明：

属性	类型	默认值	必填	说明	iOS 支持度	Android 支持度	IDE 支持度	
type	string	tencent	否	地图类型，tencent 或 google	否	否	是	
longitude	number	-	是	中心经度	是	是	是	
latitude	number	-	是	中心纬度	是	是	是	
scale	number	16	否	缩放级别，取值范围为3～20	是	是	是	
min-scale	number	3	否	最小缩放级别		是	是	
max-scale	number	20	否	最大缩放级别		是	是	
markers	array.<marker>	-	否	标记点	是	是	是	

版权所有：腾讯云计算（北京）有限责任公司

polyline	array.<polyline>	-	否	路线	是	是	是	
circles	array.<circle>	-	否	圆	是	是	是	
controls	Array.<control>	-	否	控件（即将废弃，建议使用 cover-view 代替）	否	否	否	
include-points	array.<point>	-	否	缩放视野以包含所有给定的坐标点	是	是	是	
show-location	boolean	false	否	显示带有方向的当前定位点	是	是	是	
polygons	array.<polygon>	-	否	多边形	是	是	是	
subkey	string	-	否	个性化地图使用的 key	否	否	是 type=tencent	
layer-style	number	1	否	地图能力【个性化地图】配置的 style	否	否	否	
rotate	number	0	否	旋转角度，范围0～360，地图正北与设备 y 轴的夹角角度	否	是	否	
skew	number	0	否	倾斜角度，范围	系统	是	否	

版权所有：腾讯云计算（北京）有限责任公司

				0 ~ 40 , 关于 z 轴的倾角	地图: 否 腾讯地图: 是 百度地图: 是 高德地图: 是			
enable-3D	boolean	false	否	展示3D楼块	系统地图: 否 腾讯地图: 是 百度地图: 否 高德地图: 否	谷歌地图: 是 华为地图: 是 腾讯地图: 是 百度地图: 是 高德地图: 否	否	
show-compass	boolean	false	否	显示指南针	是	是	否	
show-scale	boolean	false	否	显示比例尺	是	谷歌地图: 否	是	

版权所有：腾讯云计算（北京）有限责任公司

						华为地图: 否 腾讯地图: 是 百度地图: 是 高德地图: 是		
enable-overlooking	boolean	false	否	开启俯视	系统地图: 否 腾讯地图: 是 百度地图: 是 高德地图: 是	是	否	
enable-zoom	boolean	true	否	是否支持缩放	是	是	是	
enable-scroll	boolean	true	否	是否支持拖动	是	是	是	
enable-rotate	boolean	-	否	是否支持旋转	是	是	否	
enable-satellite	boolean	false	否	是否开启卫星图	是	谷歌地图: 是 华为地图: 否	是 type=google	

版权所有：腾讯云计算（北京）有限责任公司

						腾讯地图: 是 百度地图: 是 高德地图: 是		
enable-traffic	boolean	false	否	是否开启实时路况	是	是	是	
enable-poi	boolean	true	否	是否展示 POI 点	系统地图: 否 腾讯地图: 是 百度地图: 否 高德地图: 否	谷歌地图: 是 华为地图: 否 腾讯地图: 是 百度地图: 是 高德地图: 是	是 type=google	
enable-building	boolean	true	否	是否展示建筑物	系统地图: 否 腾讯地图: 是 百度地图: 是	谷歌地图: 是 华为地图: 否 腾讯地图: 是 百度地图: 否 高德地图: 是	是 type=google	

版权所有：腾讯云计算（北京）有限责任公司

					高德地图:是			
setting	object	-	否	配置项	否	否	是	
bindtap	eventhandle	-	否	单击地图时触发,返回经纬度信息	是	是	是	
bindmarkertap	eventhandle	-	否	点击标记点时触发 e.detail = {markerId}	是	是	是	
bindlabeltap	eventhandle	-	否	点击 label 时触发,e.detail = {markerId}	是	否	是 type=tencent	
bindcontroltap	eventhandle	-	否	点击控件时触发,e.detail = {controlId}	否	否	否	
bindcallouttap	eventhandle	-	否	点击标记点对应的气泡时触发 e.detail = {markerId}	否	否	是	
bindupdated	eventhandle	-	否	在地图渲染更新完成时触发	否	是	是	
bindregionchange	eventhandle	-	否	视野发生变化时触发	是	是	是	
bindpoitap	eventhandle	-	否	单击地图 poi 点时触发 e.detail = {name,	否	是	是	

版权所有：腾讯云计算（北京）有限责任公司

				longitude, latitude}				
bindanchorpointtap	eventhandle	-	否	点击定位标时触发, e.detail = {longitude, latitude}	否	否	否	

注意：

地图中的颜色值color/borderColor/bgColor等需使用6位（8位）十六进制表示，8位时后两位表示 alpha 值，如：#000000AA。

地图组件的经纬度必填, 如果不填经纬度则默认值是北京的经纬度。

map 组件使用的经纬度是火星坐标系，调用 wx.getLocation 接口需要指定 type 为 gcj02。

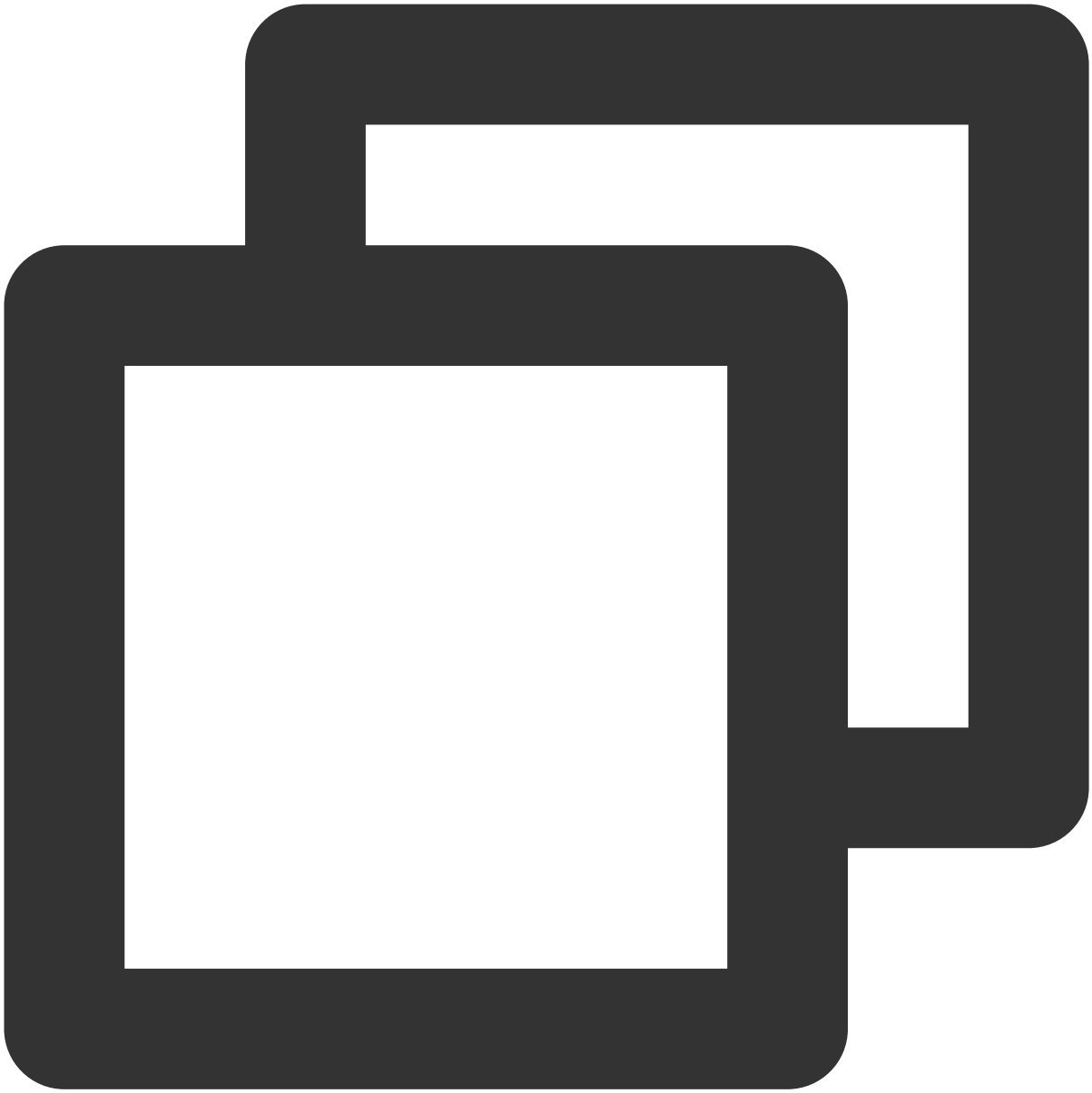
regionchange返回值

视野改变时，regionchange 会触发两次，返回的 type 值分别为 begin 和 end。

begin 阶段返回 causedBy：有效值为 gesture（手势触发）、update（接口触发）。

end 阶段返回 causedBy：有效值为 drag（拖动导致）、scale（缩放导致）、update（调用更新接口导致）。

版权所有：腾讯云计算（北京）有限责任公司

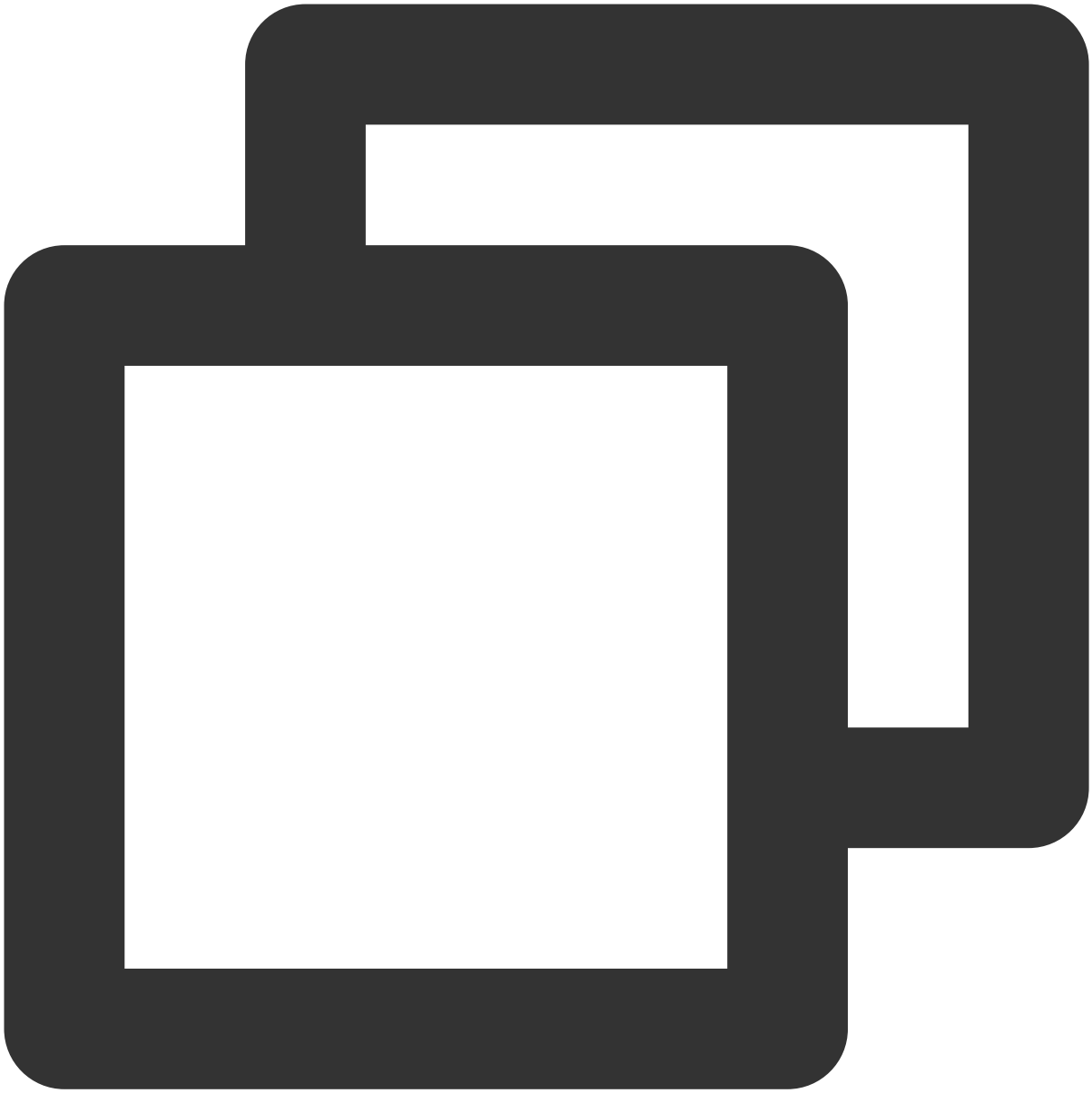


```
e = {causedBy, type, detail: {rotate, skew, scale, centerLocation, region}}
```

setting

提供 setting 对象统一设置地图配置。同时对于一些动画属性如 rotate 和 skew，通过 setData 分开设置时无法同时生效，需通过 settting 统一修改。

版权所有：腾讯云计算（北京）有限责任公司



```
// 默认值
const setting = {
  rotate: 0,
  showLocation: false,
  subKey: '',
  layerStyle: 1,
  enableZoom: true,
  enableScroll: true,
  enableRotate: false,
  showCompass: false,
  enableSatellite: false,
```

版权所有：腾讯云计算（北京）有限责任公司

```
}

this.setData({
  // 仅设置的属性会生效，其它的不受影响
  setting: {
    enableZoom: false,
    enableScroll: false,
  }
})
```

maker

标记点，用于在地图上显示标记的位置。

属性	说明	类型	必填	备注	iOS 支持	Android 支持	IDE 支持
id	标记点 id	number	否	marker 点击事件回调会返回此 id。建议为每个 marker设置 ID，保证更新marker时有更好的性能。	是	是	是
clusterId	聚合簇的 id	number	否	自定义点聚合簇效果时使用	否	否	否
joinCluster	是否参与点聚合	boolean	否	默认不参与点聚合	否	谷歌地图:是 华为地图:是 腾讯地图:是 百度地图:否 高德地图:否	否
latitude	纬度	number	是	浮点数，范围 -90 ~ 90	是	是	是
longitude	经度	number	是	浮点数，范围 -90 ~ 90	是	是	是

版权所有：腾讯云计算（北京）有限责任公司

title	标注点名	string	否	点击时显示，callout存在时将被忽略	是	是	是 type=tencent
zIndex	显示层级	number	否	-	否	是	是
iconPath	显示的图标	string	是	项目目录下的图片路径，支持网络路径、本地路径、代码包路径	是	是	是
rotate	旋转角度	number	否	顺时针旋转的角度，范围0~360，默认为0	否	是	是 type=tencent
alpha	标注的透明度	number	否	默认1，无透明，范围0~1	否	是	是 type=tencent
width	标注图标宽度	number/string	否	默认为图片实际宽度	否	是	否
height	标注图标高度	number/string	否	默认为图片实际高度	否	是	否
callout	标记点上方的气泡窗口	object	否	支持的属性见下表，可识别换行符。	是	是	是
customCallout	自定义气泡窗口	object	否	支持的属性见下表	否	否	否
label	为标记点旁边	object	否	支持的属性见下表，可	否	否	是

版权所有：腾讯云计算（北京）有限责任公司

	增加标签			识别换行符。			
anchor	经纬度在标注图标的锚点，默认底边中点	object	否	{x, y}，x 表示横向(0-1)，y 表示竖向(0-1)。{x: .5, y: 1} 表示底边中点	否	是	否

marker 上的气泡 callout

属性	说明	类型	iOS 支持度	Android 支持度	IDE 支持度
content	文本	string	是	是	是
color	文本颜色	string	否	是	是
fontSize	文字大小	number	否	是	是
borderRadius	边框圆角	number	否	是	是
borderWidth	边框宽度	number	否	是	是
borderColor	边框颜色	string	否	是	是
bgColor	背景色	string	否	是	是
padding	文本编译留白	number	否	是	是
display	'BYCLICK':点击显示; 'ALWAYS':常显	string	否	否	是
textAlign	文本对齐方式。有效值: left, right, center	string	否	是	否
anchorX	横向偏移量，向右为正数	number	否	谷歌地图:是	否

版权所有：腾讯云计算（北京）有限责任公司

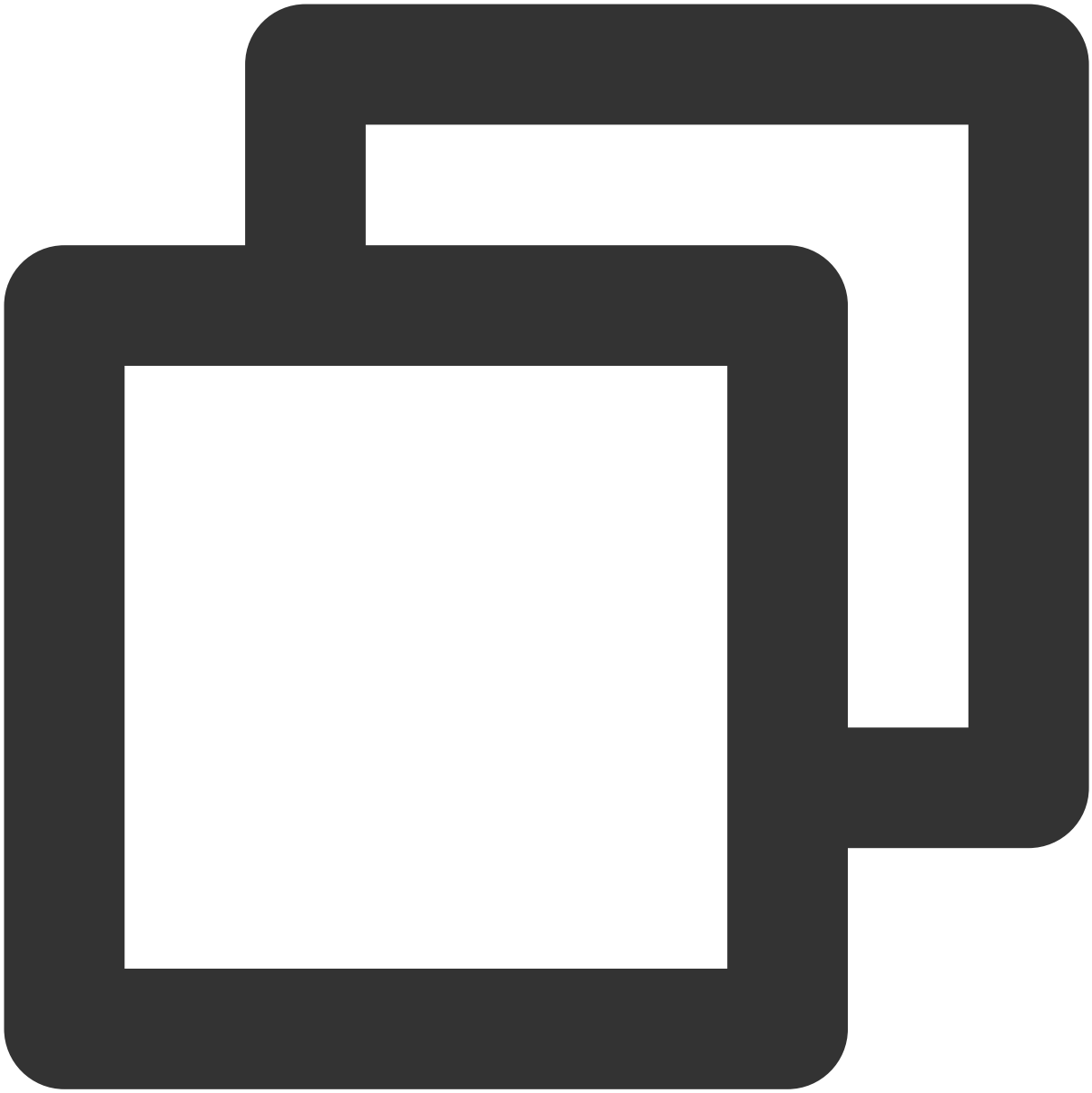
				华为地图:是 腾讯地图:是 百度地图:否 高德地图:是	
anchorY	纵向偏移量，向下为正数	number	否	谷歌地图:是 华为地图:是 腾讯地图:是 百度地图:否 高德地图:是	否

marker 上的自定义气泡 customCallout

`customCallout` 存在时将忽略 `callout` 与 `title` 属性。自定义气泡采用 `cover-view` 定制，灵活度更高。

属性	说明	类型	iOS 支持度	Android 支持度	IDE 支持度
display	'BYCLICK':点击显示; 'ALWAYS':常显	string	否	否	否
anchorX	横向偏移量，向右为正数	number	否	否	否
anchorY	纵向偏移量，向下为正数	number	否	否	否

使用方式如下， `map` 组件下添加名为 callout 的 slot 节点，其内部的 `cover-view` 通过 `marker-id` 属性与 `marker` 绑定。当 `marker` 创建时，该 `cover-view` 显示的内容将作为 `callout` 显示在标记点上方。



```
<map>
  <cover-view slot="callout">
    <cover-view marker-id="1"></cover-view>
    <cover-view marker-id="2"></cover-view>
  </cover-view>
</map>
```

marker 上的气泡 label

属性	说明	类型	iOS 支	Android 支	IDE 支持度

版权所有：腾讯云计算（北京）有限责任公司

			持度	持度	
content	文本	string	否	否	是
color	文本颜色	string	否	否	是
fontSize	文字大小	number	否	否	是
anchorX	label 的坐标，原点是 marker 对应的经纬度	number	否	否	是
anchorY	label 的坐标，原点是 marker 对应的经纬度	number	否	否	是
borderWidth	边框宽度	number	否	否	是
borderColor	边框颜色	string	否	否	是
borderRadius	边框圆角	number	否	否	是
bgColor	背景色	string	否	否	是
padding	文本边缘留白	number	否	否	是
textAlign	文本对齐方式。有效值：left, right，center	string	否	否	是

点聚合

功能说明：当地图上需要展示的标记点 marker 过多时，可能会导致界面上 marker 出现压盖，展示不全，并导致整体性能变差。针对此类问题，推出点聚合能力。

系统地图（仅 iOS 支持）

使用流程如下：

MapContext.addMarkers 指定参与聚合的 marker。

（可选） `MapContext.on('markerClusterCreate', callback)` 触发时，通过 MapContext.addMarkers 更新聚合簇的样式。

MapContext.removeMarkers 移除参与聚合的 marker。

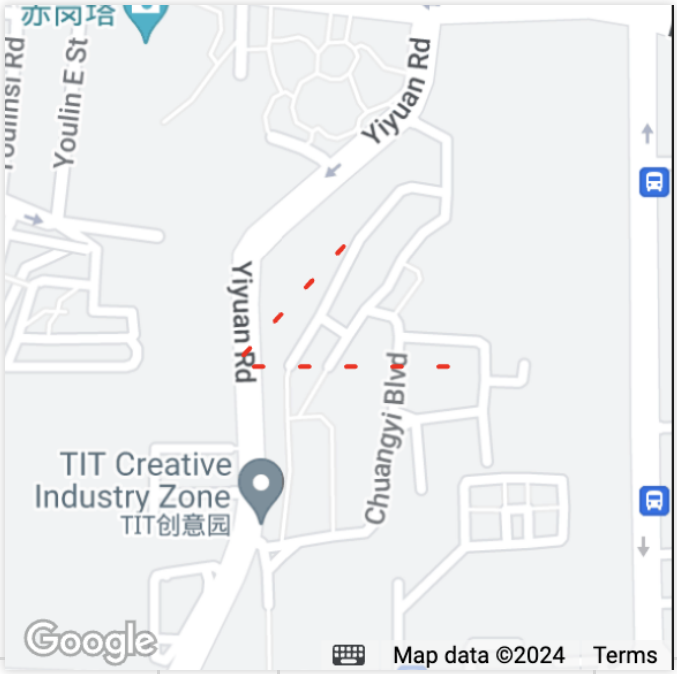
谷歌地图（仅 Android 支持）

使用方法为添加 marker 时指定 joinCluster 即可，符合条件的 marker 将会自动进行聚合。

polyline

功能说明：指定一系列坐标点，从数组第一项连线至最后一项。绘制彩虹线时，需指定不同分段的颜色，如 points 包含 5 个点，则 colorList 应传入 4 个颜色值；若 colorList 长度小于 points.length - 1，则剩下的分段颜色与最后一项保持一致。

版权所有：腾讯云计算（北京）有限责任公司



属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
points	经纬度数组	Array	是	[{latitude: 0, longitude: 0}]	是	是	是
color	线的颜色	string	否	十六进制	是	是	是
colorList	彩虹线	Array	否	存在时忽略 color 值	否	谷歌地图: 否 华为地图: 否 腾讯地图: 是 百度地图: 是 高德地图: 是	否
width	线的宽度	number	否		是	是	是
dottedLine	是否虚线	boolean	否	默认 false	是	是	是
arrowLine	带箭头的	boolean	否	-	否	是	否

	线						
arrowIconPath	更换箭头图标	string	否	-	否	是	否
borderColor	线的边框颜色	string	否	-	否	谷歌地图:否 华为地图:否 腾讯地图:是 百度地图:否 高德地图:否	是
borderWidth	线的厚度	number	否	-	否	谷歌地图:否 华为地图:否 腾讯地图:是 百度地图:否 高德地图:否	是
level	压盖关系	string	否	默认为abovelabels	否	谷歌地图:否 华为地图:否 腾讯地图:是 百度地图:否 高德地图:否	否
textStyle	文字样式	TextStyle	否	折线上文本样式	否	谷歌地图:否 华为地图:否 腾讯地图:是	否

版权所有：腾讯云计算（北京）有限责任公司

						百度地图: 否 高德地图: 否	
segmentTexts	分段文本	Array<SegmentText>	否	折线上文本内容和位置	否	谷歌地图: 否 华为地图: 否 腾讯地图: 是 百度地图: 否 高德地图: 否	否

SegmentText

属性	说明	类型
name	名称	string
startIndex	起点	number
endIndex	终点	number

TextStyle

属性	说明	类型	默认值
textColor	文本颜色	string	#000000
strokeColor	描边颜色	number	#ffffff
fontSize	文字大小	number	14

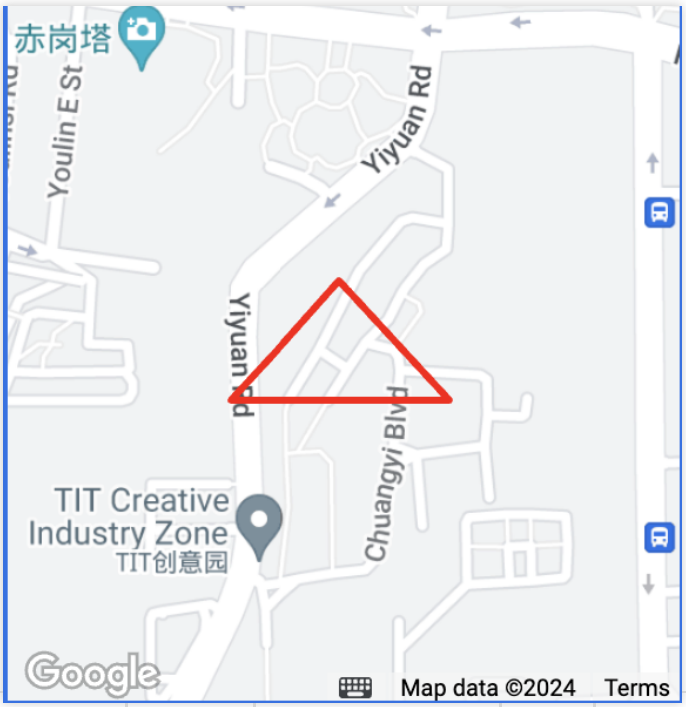
`level` 字段表示与其它地图元素的压盖关系，可选值如下：

值	说明
abovelabels	显示在所有 POI 之上
abovebuildings	显示在楼块之上 POI 之下
aboveroads	显示在道路之上楼块之下

版权所有：腾讯云计算（北京）有限责任公司

polygon

功能说明：指定一些列坐标点，根据points坐标数据生成闭合多边形。



属性	说明	类型	必填	备注	iOS支持	Android支持	IDE支持
dashArray	边线虚线	Array<number>	否	默认值 [0, 0] 为实线，[10, 10]表示十个像素的实线和十个像素的空白（如此反复）组成的虚线	否	谷歌地图:否 华为地图:否 腾讯地图:是 百度地图:否 高德地图:否	否
points	经纬度数组	array	是	[{latitude: 0, longitude: 0}]	是	是	是
strokeWidth	描边的宽度	number	否	-	是	是	否
strokeColor	描边的颜色	string	否	十六进制	是	是	否

fillColor	填充颜色	string	否	十六进制	是	是	是
zIndex	设置多边形 Z 轴数值	number	否	-	否	是	是
level	压盖关系	string	否	默认为 abovelabels	否	谷歌地图:否 华为地图:否 腾讯地图:是 百度地图:否 高德地图:否	否

circle

功能说明：在地图上显示圆。



属性	说明	类型	必填	备注	iOS支持度	Android支持度	IDE支持度
latitude	纬度	number	是	浮点数，范围 -90 ~ 90	是	是	是

longitude	经度	number	是	浮点数，范围 -180 ~ 180	是	是	是
color	描边的颜色	string	否	浮点数，范围 -180 ~ 180	是	是	是
fillColor	填充颜色	string	否	十六进制	是	是	是
radius	半径	number	是	-	是	是	是
strokeWidth	描边的宽度	number	否	-	是	是	否
level	压盖关系	string	否	默认为 abovelabels	否	谷歌地图: 否 华为地图: 否 腾讯地图: 是 百度地图: 否 高德地图: 否	否

control

功能说明：在地图上显示控件，控件不随着地图移动。即将废弃，推荐使用 cover-view。

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
id	控件 id	number	否	在控件点击事件会回调会返回此 id	否	否	否
position	空间在地图的位置	object	是	空间相对地图位置	否	否	否
iconPath	显示的图标	string	是	项目目录下的图片路径，支持本地路径、代码包路径	否	否	否
clickable	是否可点击	boolean	否	默认不可点击	否	否	否

position

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
left	距离地图的左边界多远	number	否	默认为0	否	否	否
top	距离地图的上边界多远	number	否	默认为0	否	否	否
width	空间宽度	number	否	默认为图片宽度	否	否	否
height	空间高度	number	否	默认为图片高度	否	否	否

bindregionchange 返回值

属性	说明	类型	备注
type	视野变化开始、结束时触发	string	视野变化开始为 begin，结束为 end
causedBy	导致视野变化的原因	string	拖动地图导致（drag）、缩放导致（scale）、调用接口导致（update）

比例尺

scale	3	4	5	6	7	8	9	10	11
比例	1000km	500km	200km	100km	50km	50km	20km	10km	5km
scale	12	13	14	15	16	17	18	19	20
比例	2km	1km	500m	200m	100m	50m	50m	20m	10m

版权所有：腾讯云计算（北京）有限责任公司

媒体组件

最近更新时间：2024-07-12 18:14:26

audio

功能说明：音频。本组件已停止维护，建议使用 wx.createInnerAudioContext 代替。

参数及说明：

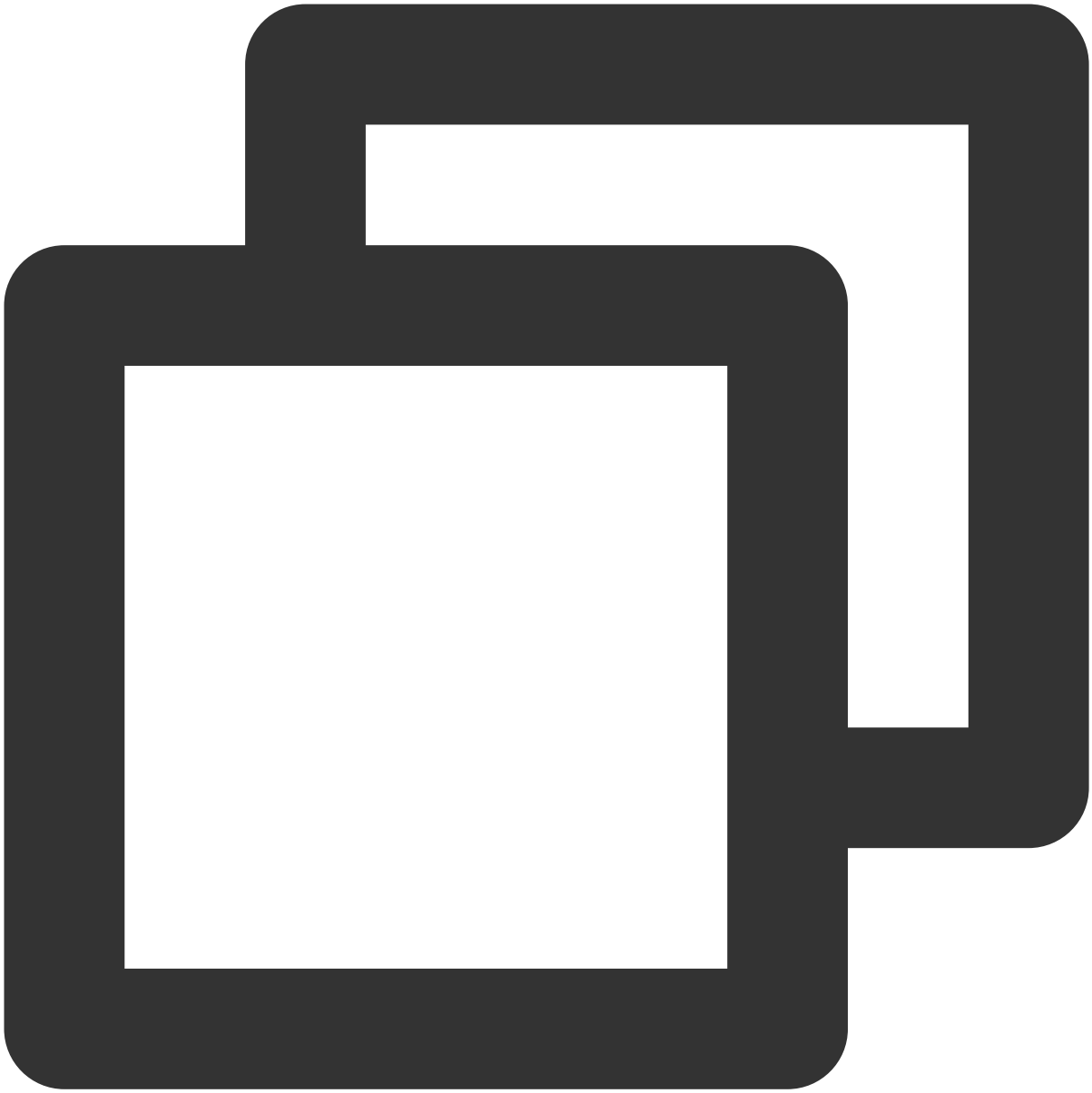
属性	类型	默认值	必填	说明
id	string	-	否	audio 组件的唯一标识符
src	string	-	否	要播放音频的资源地址
loop	boolean	false	否	是否循环播放
controls	boolean	false	否	是否显示默认控件
poster	string	-	否	默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false 则设置 poster 无效
name	string	未知歌曲	否	默认控件上的音频名字，如果 controls 属性值为 false 则设置 name 无效
author	string	未知作者	否	默认控件上的作者名字，如果 controls 属性值为 false 则设置 author 无效
binderror	eventhandle	-	否	当发生错误时触发 error 事件，detail = {errMsg:MediaError.code}
bindplay	eventhandle	-	否	当开始/继续播放时触发 play 事件
bindpause	eventhandle	-	否	当暂停播放时触发 pause 事件
bindtimeupdate	eventhandle	-	否	当播放进度改变时触发 timeupdate 事件，detail = {currentTime, duration}
bindended	eventhandle	-	否	当播放到末尾时触发 ended 事件

示例代码：

WXML

JAVASCRIPT

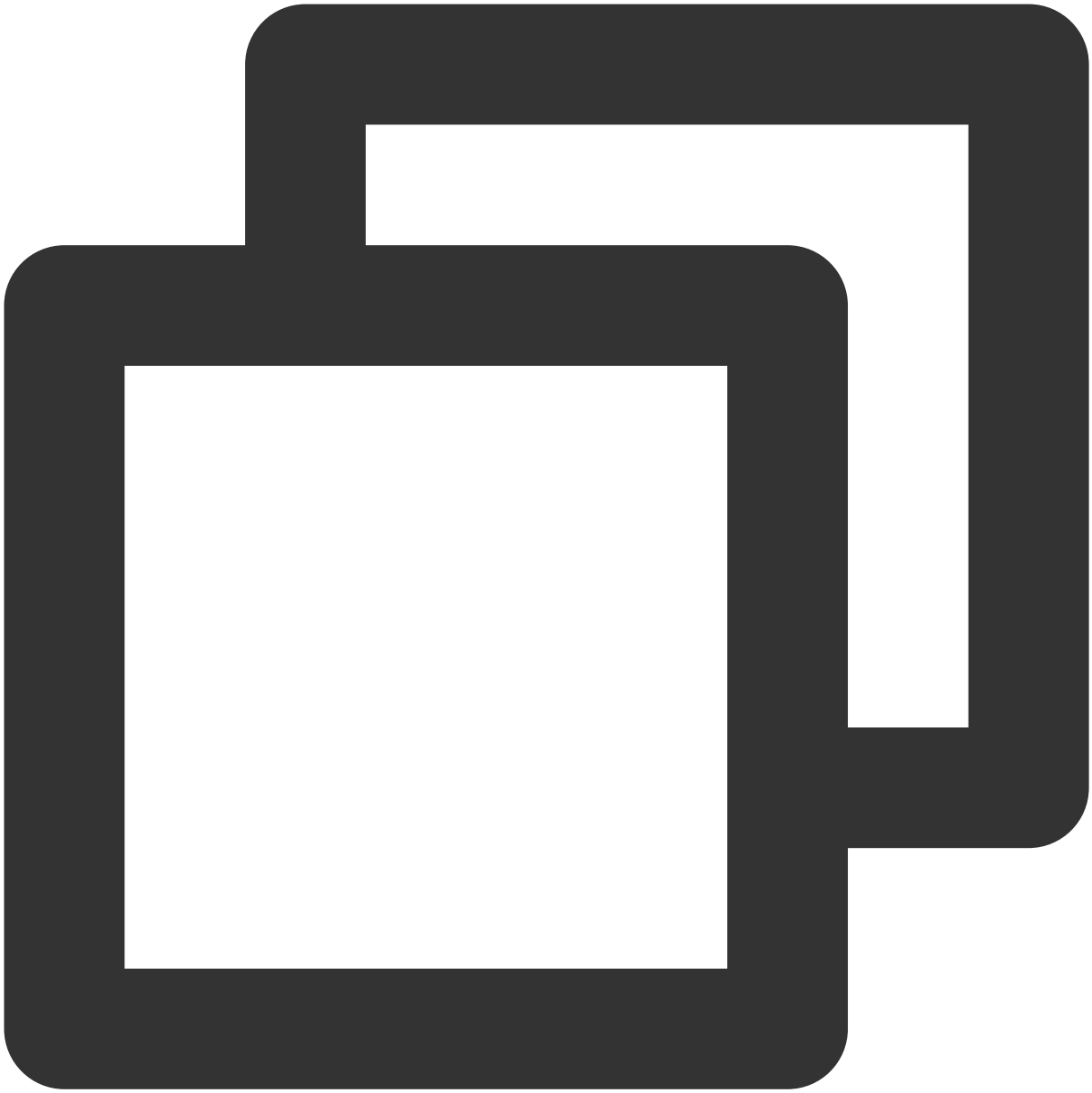
版权所有：腾讯云计算（北京）有限责任公司



```
<!-- audio.wxml -->
<audio poster="{{poster}}" name="{{name}}" author="{{author}}" src="{{src}}" id="my

<button type="primary" bindtap="audioPlay">播放</button>
<button type="primary" bindtap="audioPause">暂停</button>
<button type="primary" bindtap="audio14">设置当前播放时间为14秒</button>
<button type="primary" bindtap="audioStart">回到开头</button>
```

版权所有：腾讯云计算（北京）有限责任公司



```
// audio.js
Page({
  onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioCtx = wx.createAudioContext('myAudio')
  },
  data: {
    poster: 'http://y.gtimg.cn/music/photo_new/T002R300x300M000003rsKF44GyaSk.jpg?m
    name: '此时此刻',
    author: '许巍',
    src: 'http://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?guid=ffffffff82def
```

版权所有：腾讯云计算（北京）有限责任公司

```
  },
  audioPlay: function () {
    this.audioCtx.play()
  },
  audioPause: function () {
    this.audioCtx.pause()
  },
  audio14: function () {
    this.audioCtx.seek(14)
  },
  audioStart: function () {
    this.audioCtx.seek(0)
  }
})
```

image

功能说明：图片。

参数及说明

属性	类型	默认值	说明
src	string	-	图片资源地址
mode	string	'scaleToFill'	图片裁剪、缩放的模式
show-menu-by-longpress	boolean	false	长按图片显示类似保存图片的菜单
lazy-load	boolean	false	图片懒加载。只针对 page 与 scroll-view 下的 image 有效
binderror	handleevent	-	当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	handleevent	-	当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {height:'图片高度px', width:'图片宽度px'}

注意：

image 组件默认宽度300px、高度225px。

image 组件进行缩放时，计算出来的宽高可能带有小数，在不同 webview 内核下渲染可能会被抹去小数部分。

mode 有效值：mode 有 13 种模式，其中 4 种是缩放模式，9 种是裁剪模式。

模式	值	说明

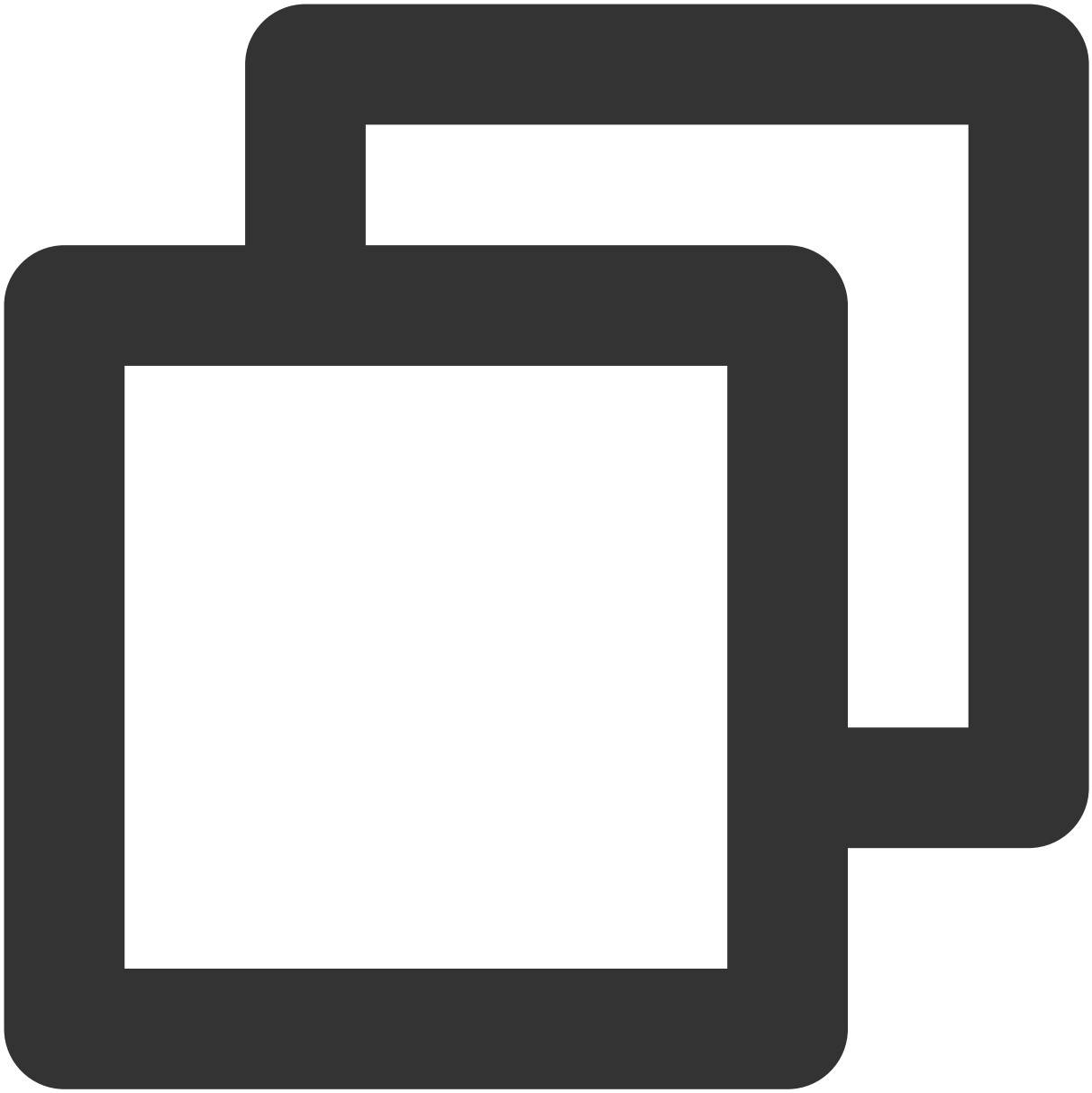
缩放	scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
缩放	aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。即可以完整地将图片显示出来
缩放	aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。即图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取
缩放	widthFix	宽度不变，高度自动变化，保持原图宽高比不变
裁剪	top	不缩放图片，只显示图片的顶部区域
裁剪	bottom	不缩放图片，只显示图片的底部区域
裁剪	center	不缩放图片，只显示图片的中间区域
裁剪	left	不缩放图片，只显示图片的左边区域
裁剪	right	不缩放图片，只显示图片的右边区域
裁剪	top left	不缩放图片，只显示图片的左上角区域
裁剪	top right	不缩放图片，只显示图片的右上角区域
裁剪	bottom left	不缩放图片，只显示图片的左下边区域
裁剪	bottom right	不缩放图片，只显示图片的右下边区域

示例代码：

WXML

JAVASCRIPT

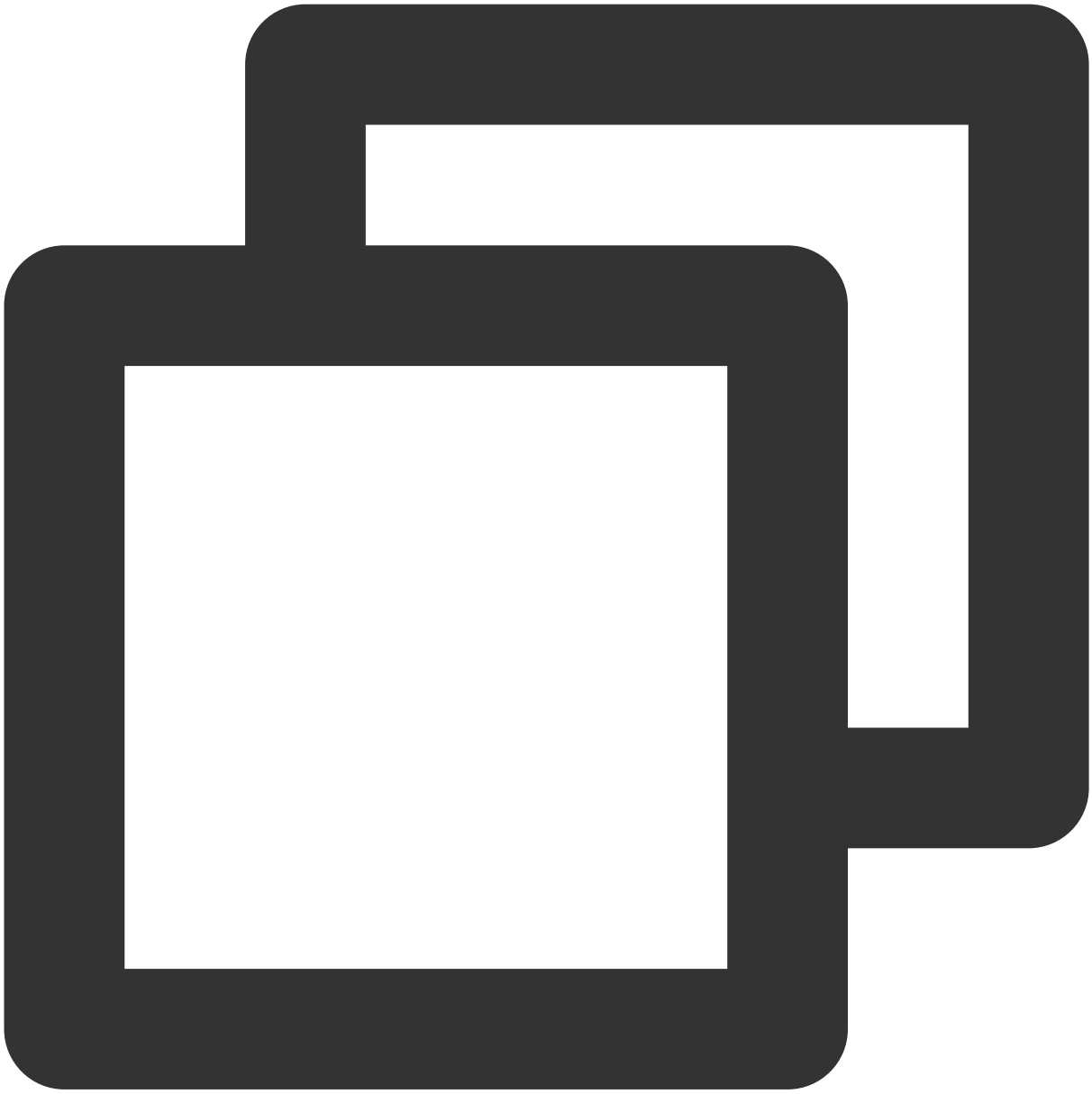
版权所有：腾讯云计算（北京）有限责任公司



```
<view class="page">
  <view class="page__hd">
    <text class="page__title">image</text>
    <text class="page__desc">图片</text>
  </view>
  <view class="page__bd">
    <view class="section section_gap" wx:for="{{array}}" wx:for-item="item">
      <view class="section__title">{{item.text}}</view>
      <view class="section__ctn">
        <image style="width: 200px; height: 200px; background-color: #eeeeee;" mode
      </view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
    </view>
  </view>
</view>
```



```
Page({
  data: {
    array: [{
      mode: 'scaleToFill',
      text: 'scaleToFill：不保持纵横比缩放图片，使图片完全适应'
    }, {
      mode: 'aspectFit',
```

```
      text: 'aspectFit：保持纵横比缩放图片，使图片的长边能完全显示出来'
    }, {
      mode: 'aspectFill',
      text: 'aspectFill：保持纵横比缩放图片，只保证图片的短边能完全显示出来'
    }, {
      mode: 'top',
      text: 'top：不缩放图片，只显示图片的顶部区域'
    }, {
      mode: 'bottom',
      text: 'bottom：不缩放图片，只显示图片的底部区域'
    }, {
      mode: 'center',
      text: 'center：不缩放图片，只显示图片的中间区域'
    }, {
      mode: 'left',
      text: 'left：不缩放图片，只显示图片的左边区域'
    }, {
      mode: 'right',
      text: 'right：不缩放图片，只显示图片的右边边区域'
    }, {
      mode: 'top left',
      text: 'top left：不缩放图片，只显示图片的左上边区域'
    }, {
      mode: 'top right',
      text: 'top right：不缩放图片，只显示图片的右上边区域'
    }, {
      mode: 'bottom left',
      text: 'bottom left：不缩放图片，只显示图片的左下边区域'
    }, {
      mode: 'bottom right',
      text: 'bottom right：不缩放图片，只显示图片的右下边区域'
    }],
    src: 'https://res.wx.qq.com/wxdoc/dist/assets/img/0.4cb08bb4.jpg'
  },
  imageError: function(e) {
    console.log('image3发生error事件，携带值为', e.detail.errMsg)
  }
})
```

video

功能说明：视频。低版本为原生组件，使用时请注意 原生组件 使用限制。

属性	类型	默认值	说明

src	string	-	要播放视频的资源地址，支持云文件 ID
duration	number	-	指定视频时长
controls	boolean	true	是否显示默认播放控件（播放/暂停按钮、播放进度、时间）
danmu-list	Object Array	-	弹幕列表
danmu-btn	boolean	false	是否显示弹幕按钮，只在初始化时有效，不能动态变更
enable-danmu	boolean	false	是否展示弹幕，只在初始化时有效，不能动态变更
autoplay	boolean	false	是否自动播放
loop	boolean	false	是否循环播放
muted	boolean	false	是否静音播放
initial-time	number	-	指定视频初始播放位置
page-gesture	boolean	false	在非全屏模式下，是否开启亮度与音量调节手势
direction	number	-	设置全屏时视频的方向，不指定则根据宽高比自动判断。 有效值为0（正常竖向） 有效值为90（屏幕逆时针90度） 有效值为-90（屏幕顺时针90度）
show-progress	boolean	true	若不设置，宽度大于240时才会显示
show-fullscreen-btn	boolean	true	是否显示全屏按钮
show-play-btn	boolean	true	是否显示视频底部控制栏的播放按钮
show-center-play-btn	boolean	true	是否显示视频中间的播放按钮
enable-progress-gesture	boolean	true	是否开启控制进度的手势
object-fit	string	contain	当视频大小与 video 容器大小不一致时，视频的表现形式。 contain：包含； fill：填充； cover：覆盖；
poster	string	-	视频封面的图片网络资源地址或云文件 ID。若 controls 属性值为 false 则设置 poster 无效

版权所有：腾讯云计算（北京）有限责任公司

show-mute-btn	boolean	false	是否显示静音按钮
title	string	-	视频的标题，全屏时在顶部展示
play-btn-position	string	bottom	播放按钮的位置， 有效值为 bottom（controls bar 上）； 有效值为 center（视频中间）；
enable-play-gesture	boolean	false	是否开启播放手势，即双击切换播放/暂停
auto-pause-if-navigate	boolean	true	当跳转到其它小程序页面时，是否自动暂停本页面的视频
auto-pause-if-open-native	boolean	true	当跳转到其它原生页面时，是否自动暂停本页面的视频
vslide-gesture	boolean	false	在非全屏模式下，是否开启亮度与音量调节手势（同 page-gesture）
vslide-gesture-in-fullscreen	boolean	true	在全屏模式下，是否开启亮度与音量调节手势
picture-in-picture-mode	string/Array	-	设置小窗模式：push, pop，空字符串或通过数组形式设置多种模式（如：["push", "pop"]）
enable-auto-rotation	boolean	false	是否开启手机横屏时自动全屏，当系统设置开启自动旋转时生效
bindplay	eventhandle	-	当开始/继续播放时触发 play 事件
bindpause	eventhandle	-	当暂停播放时触发 pause 事件
bindended	eventhandle	-	当播放到末尾时触发 ended 事件
bindtimeupdate	eventhandle	-	播放进度变化时触发，event.detail = {currentTime, duration} 。触发频率250ms一次
bindfullscreenchange	eventhandle	-	视频进入和退出全屏时触发，event.detail = {fullScreen, direction}，direction 有效值为 vertical 或 horizontal
bindwaiting	eventhandle	-	视频出现缓冲时触发
binderror	eventhandle	-	视频播放出错时触发
bindprogress	eventhandle	-	加载进度变化时触发，只支持一段加载。event.detail = {buffered}，百分比

版权所有：腾讯云计算（北京）有限责任公司

bindloadedmetadata	eventhandle	-	视频元数据加载完成时触发。event.detail = {width, height, duration}

注意：

<video>默认宽度300px、高度225px，可通过 wxss 设置宽高。

小窗特性说明：

video 小窗支持以下三种触发模式（在组件上设置 picture-in-picture-mode 属性）：

push 模式，即从当前页跳转至下一页时出现小窗（页面栈 push）。

pop 模式，即离开当前页面时触发（页面栈 pop）。

以上两种路由行为均触发小窗。

此外，小窗还支持以下特性：

小窗容器尺寸会根据原组件尺寸自动判断。

点击小窗，用户会被导航回小窗对应的播放器页面。

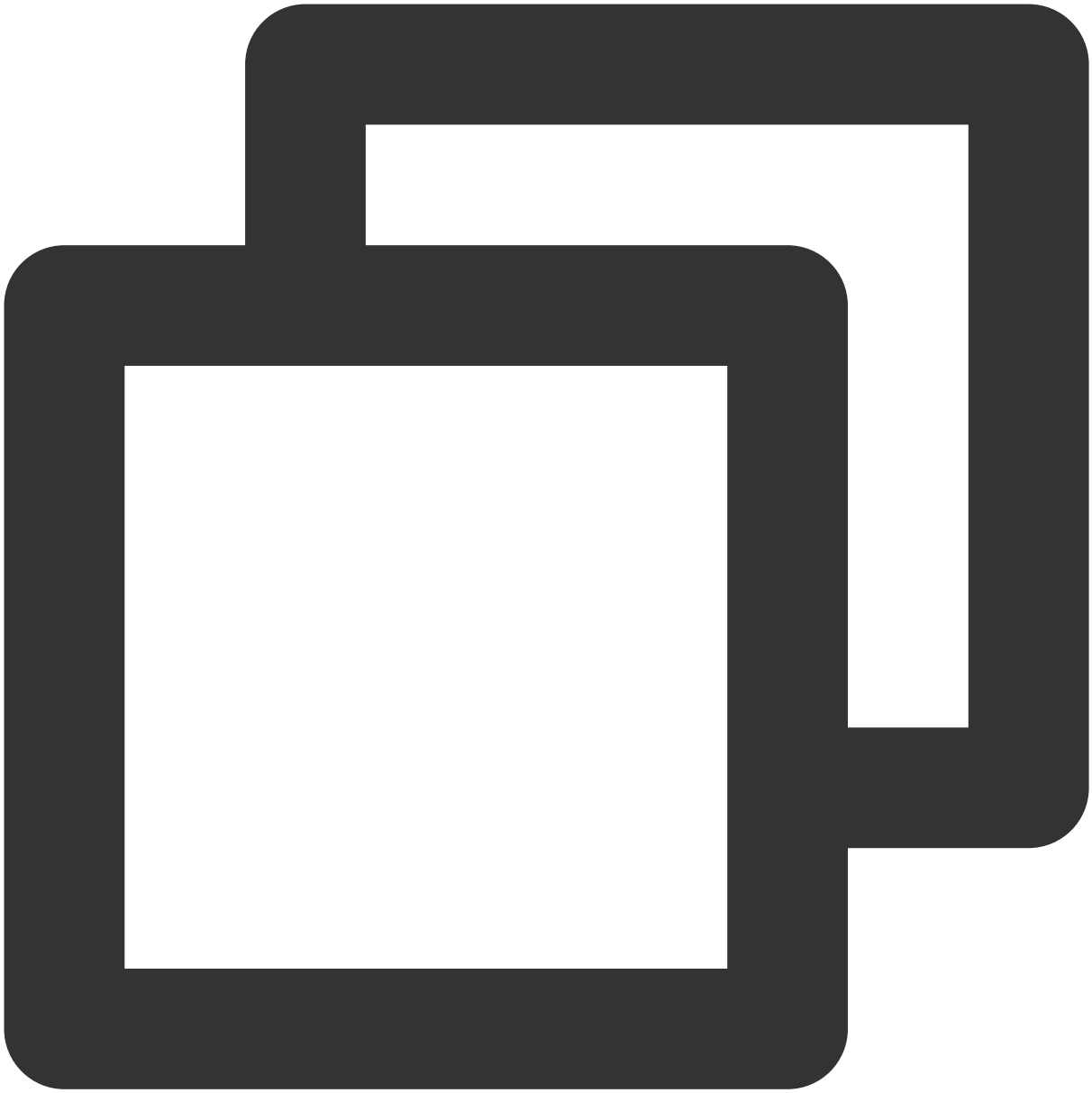
小窗出现后，用户可点击小窗右上角的关闭按钮或调用 context.exitPictureInPicture() 接口关闭小窗。

当播放器进入小窗模式后，播放器所在页面处于 hide 状态(触发 onHide 生命周期)，该页面不会被销毁。当小窗被关闭时，播放器所在页面会被 unload (触发 onUnload 生命周期)。

示例代码：

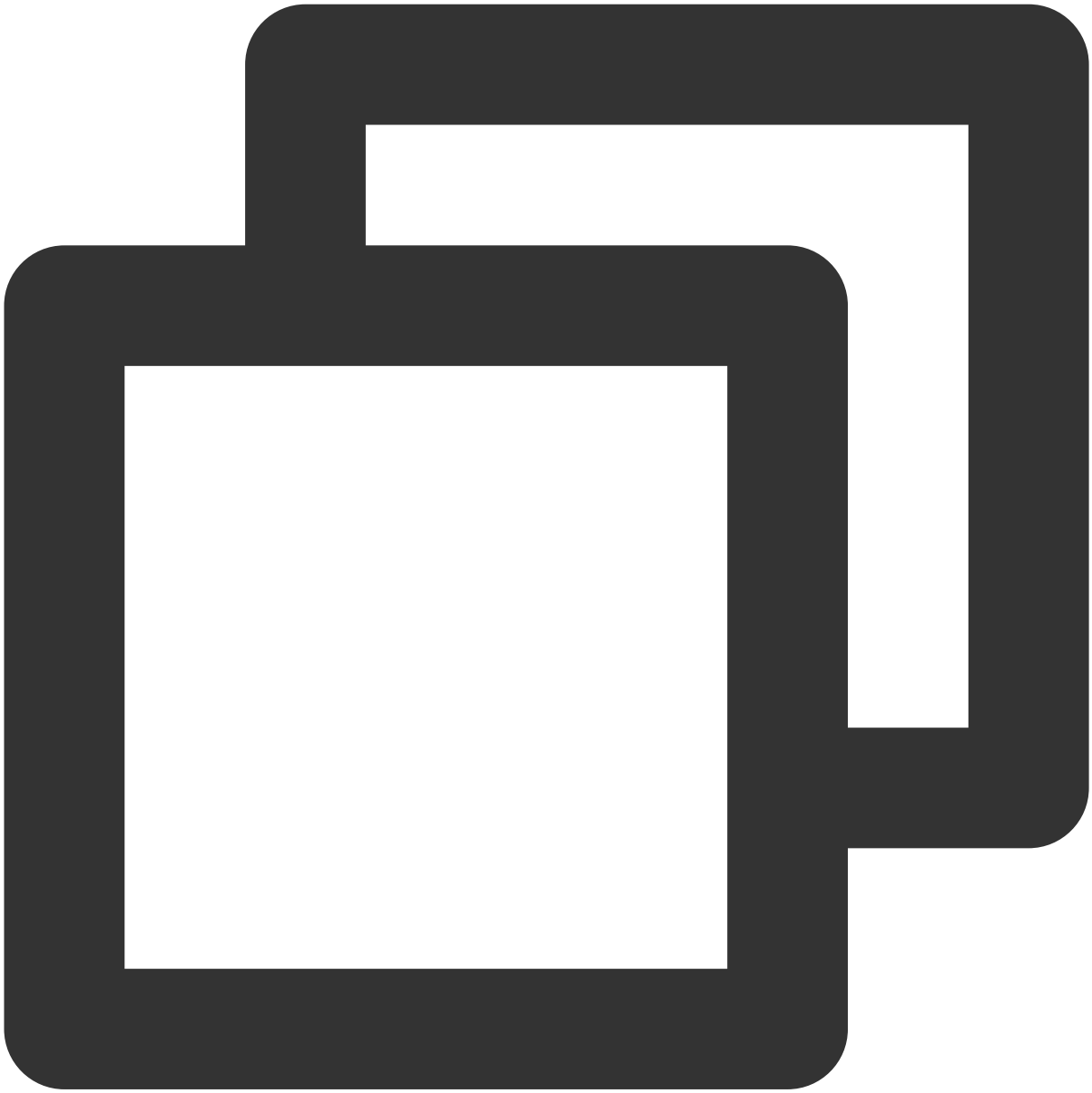
WXML

JAVASCRIPT



```
<view class="page-body">
  <view class="page-section tc">
    <video
      id="myVideo"
      src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?filekey=3028020101
      binderror="videoErrorCallback"
      danmu-list="{{danmuList}}"
      enable-danmu
      danmu-btn
      show-center-play-btn='{{false}}'
      show-play-btn="{{true}}"
```

```
      controls
      picture-in-picture-mode="{{['push', 'pop']}}"
      bindenterpictureinpicture='bindVideoEnterPictureInPicture'
      bindleavepictureinpicture='bindVideoLeavePictureInPicture'
    ></video>
    <view style="margin: 30rpx auto" class="weui-label">弹幕内容</view>
    <input bindblur="bindInputBlur" class="weui-input" type="text" placeholder="在此
    <button style="margin: 30rpx auto"  bindtap="bindSendDanmu" class="page-body-bu
    <navigator style="margin: 30rpx auto"  url="picture-in-picture" hover-class="ot
      <button type="primary" class="page-body-button" bindtap="bindPlayVideo">小窗模
    </navigator>
  </view>
</view>
```



```
function getRandomColor() {
  const rgb = []
  for (let i = 0; i < 3; ++i) {
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length === 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}

Page({
```

```
  onShareAppMessage() {
    return {
      title: 'video',
      path: 'page/component/pages/video/video'
    }
  },

  onReady() {
    this.videoContext = wx.createVideoContext('myVideo')
  },

  onHide() {

  },

  inputValue: '',
  data: {
    src: '',
    danmuList:
    [{
      text: '第 1s 出现的弹幕',
      color: '#ff0000',
      time: 1
    }, {
      text: '第 3s 出现的弹幕',
      color: '#ff00ff',
      time: 3
    }],
  },

  bindInputBlur(e) {
    this.inputValue = e.detail.value
  },

  bindButtonTap() {
    const that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: ['front', 'back'],
      success(res) {
        that.setData({
          src: res.tempFilePath
        })
      }
    })
  },
```

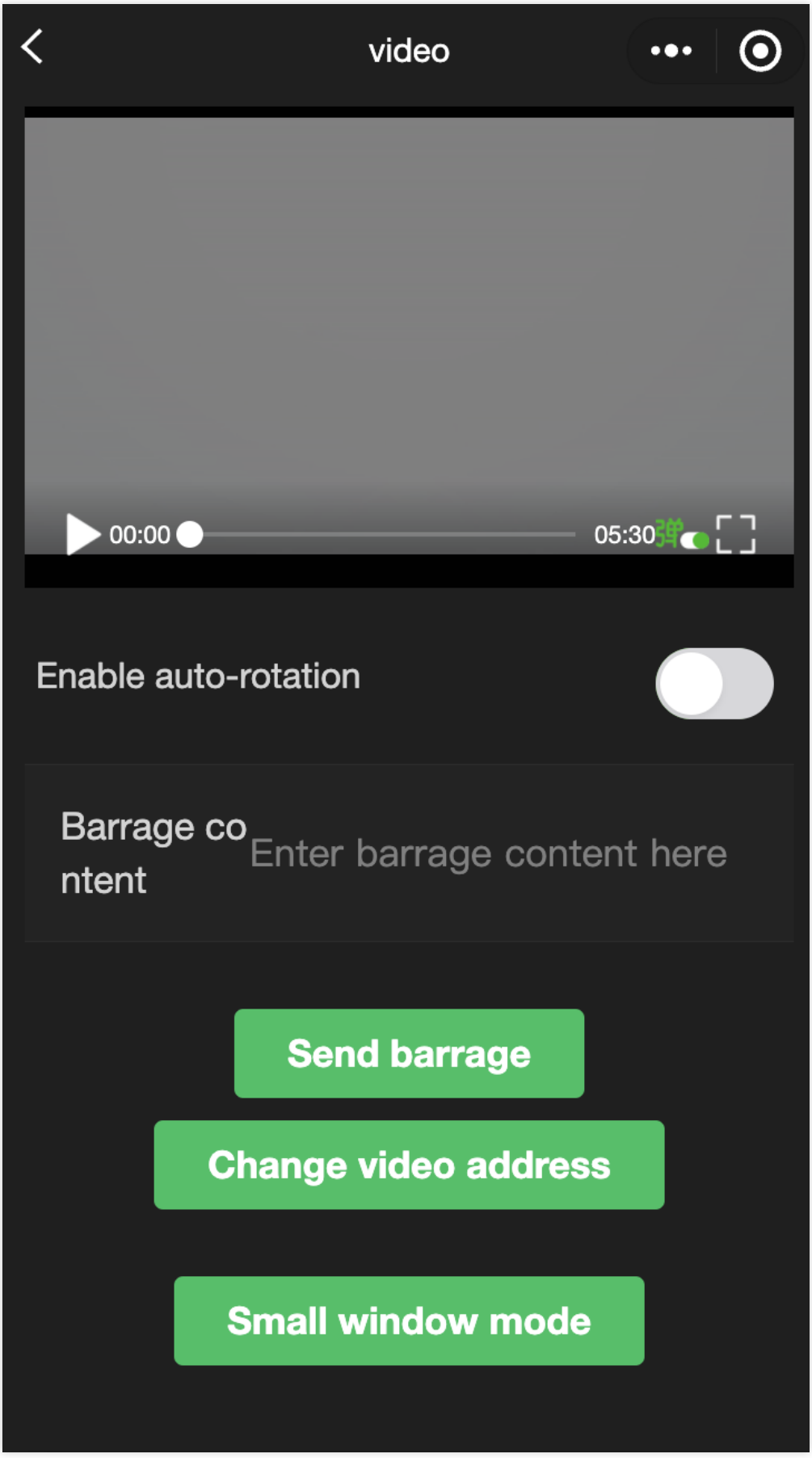
版权所有：腾讯云计算（北京）有限责任公司

```
  bindVideoEnterPictureInPicture() {
    console.log('进入小窗模式')
  },

  bindVideoLeavePictureInPicture() {
    console.log('退出小窗模式')
  },

  bindPlayVideo() {
    console.log('1')
    this.videoContext.play()
  },
  bindSendDanmu() {
    this.videoContext.sendDanmu({
      text: this.inputValue,
      color: getRandomColor()
    })
  },

  videoErrorCallback(e) {
    console.log('视频错误信息:')
    console.log(e.detail.errMsg)
  }
})
```



版权所有：腾讯云计算（北京）有限责任公司

animation-video

功能说明：animation-video 属于前端组件，为小程序提供了将特定视频资源渲染为透明背景动效的能力，可以帮助开发者低成本实现更为沉浸，丰富的动画效果。animation-video 组件还提供丰富的 API 来控制动画的播放，暂停，跳到指定位置等。相关 API 可参见 wx.createAnimationVideo。

参数及说明：

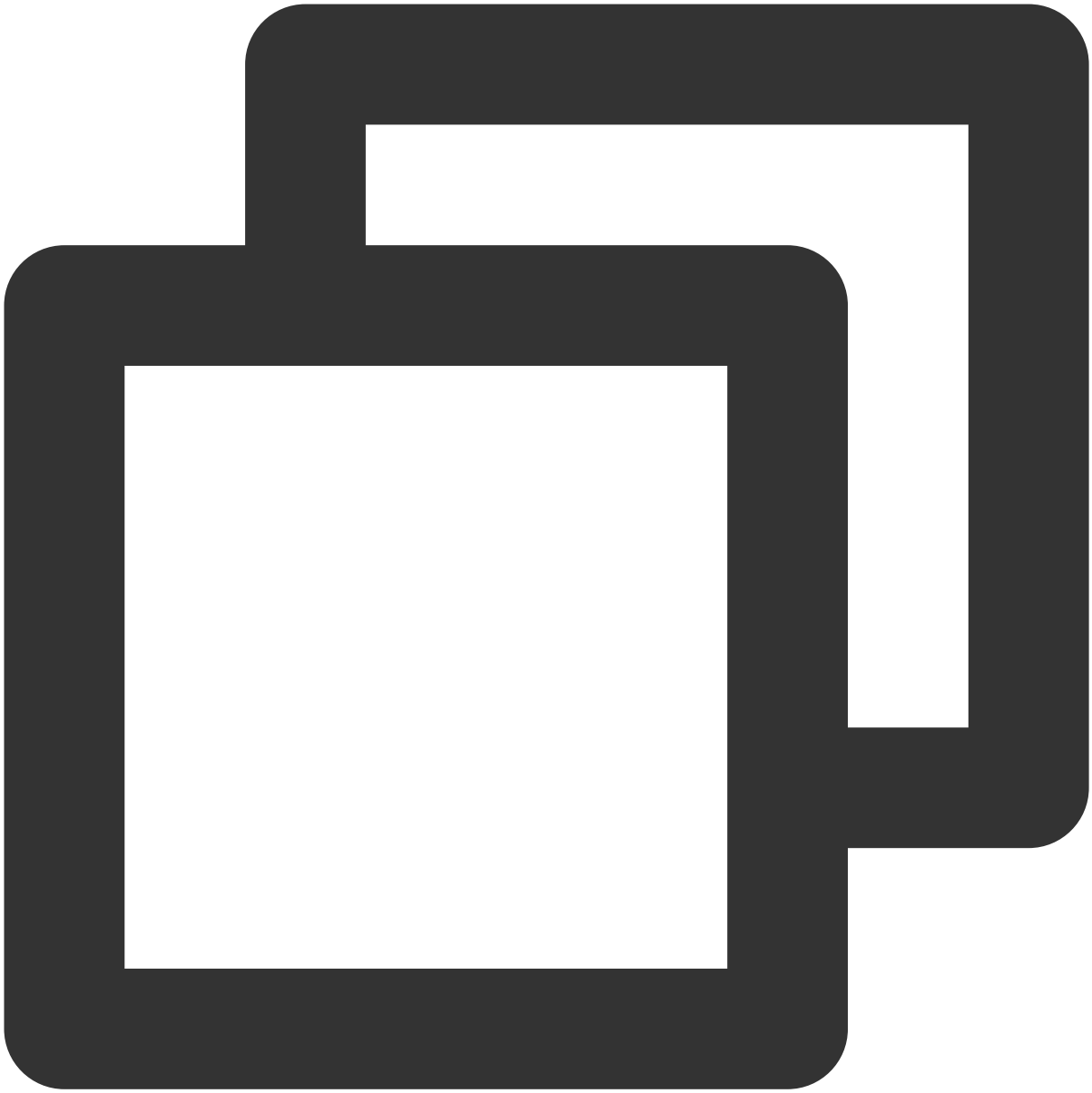
属性	类型	默认值	必填	说明
resource-width	number	800	否	组件使用的 video 视频资源的宽度（单位：px）
resource-height	number	400	否	组件使用的 video 视频资源的高度（单位：px）
canvas-style	string	width:400px; height:400px;	否	用于设置动画画布的 CSS 样式
path	string		是	动画资源地址，支持相对路径以及远程地址。如果是远程路径，**response header 里需要设置 Access-Control-Allow-Origin 来防止跨域问题**
loop	boolean	false	否	动画是否循环播放
autoplay	boolean	false	否	动画是否自动
alpha-direction	string	left	否	视频资源中 alpha 通道的方向，left 表示 alpha 通道在资源的左边，right 表示 alpha 通道在资源的右边。
bindstarted	eventhandle	-	否	动画开始播放的回调
bindended	eventhandle	-	否	当播放到末尾时触发 ended 事件（自然播放结束会触发回调，循环播放结束及暂停动画不会触发）

示例代码：

WXML

JAVASCRIPT

版权所有：腾讯云计算（北京）有限责任公司

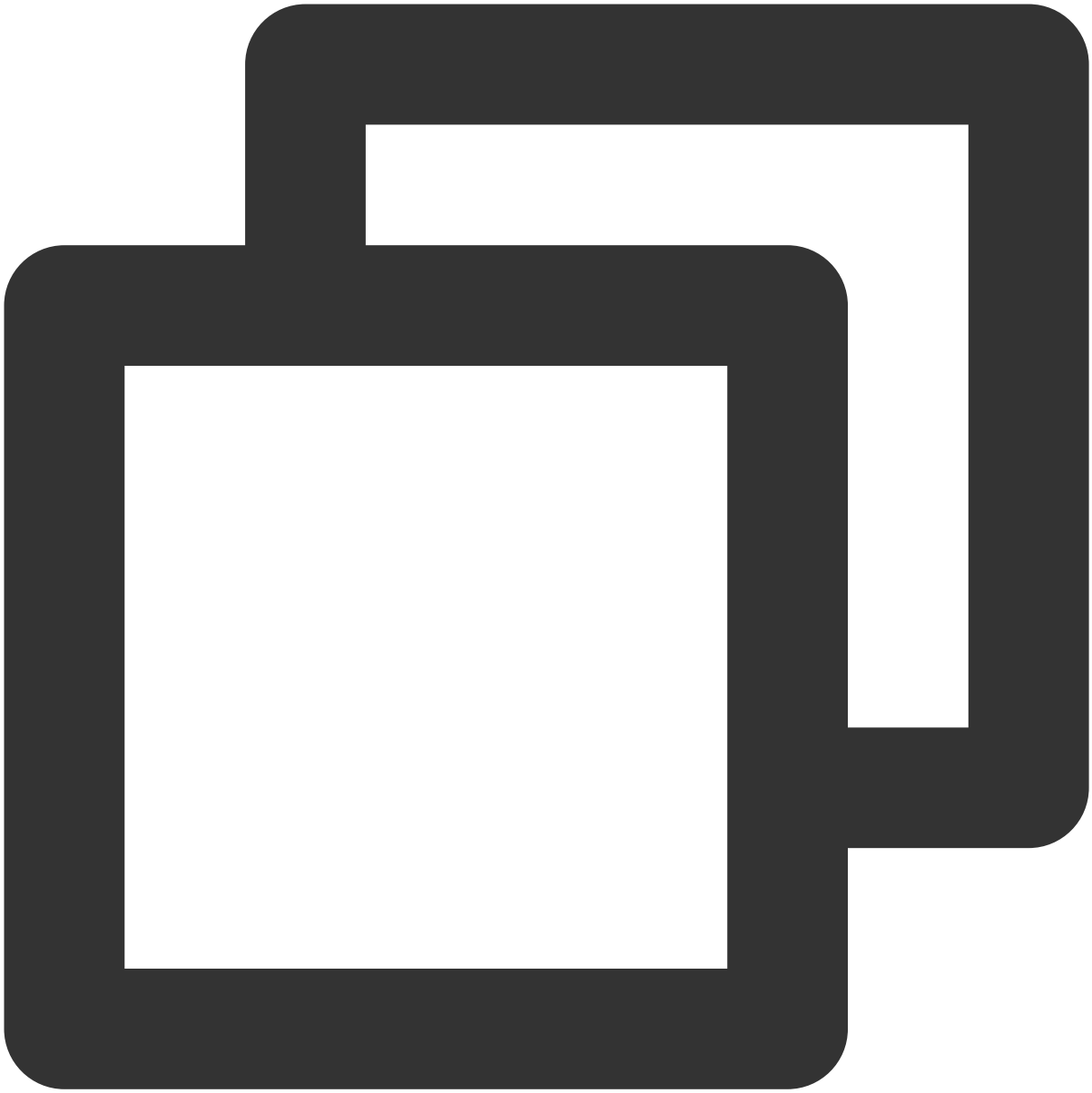


```
<view class="wrap">
    <view class="card-area">
        <view class="top-description border-bottom">
            <view>视频右侧为动画 alpha 通道信息</view>
            <view>alpha-direction='right'</view>
        </view>
        <view class="video-area">
            <animation-video
                path="{{rightAlphaSrcPath}}"
                loop="{{loop}}"
                resource-width="800"
```

版权所有：腾讯云计算（北京）有限责任公司

```
                resource-height="400"
                canvas-style="width:200px;height:200px"
                autoplay="{{autoplay}}"
                bindstarted="onStarted"
                bindended="onEnded"
                alpha-direction='right'
            ></animation-video>
        </view>
    </view>

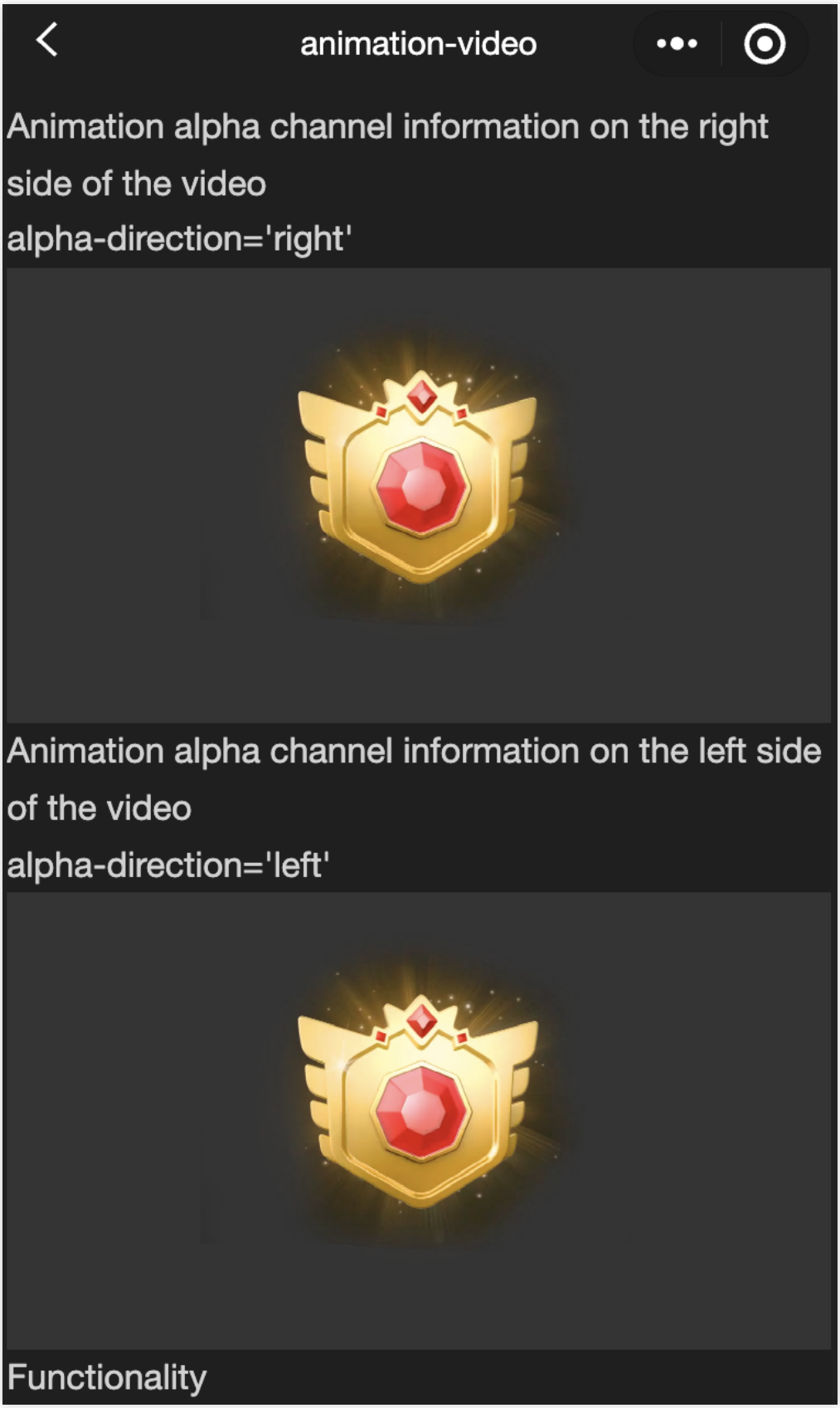

    <view class="card-area">
        <view class="top-description border-bottom">
            <view>视频左侧为动画 alpha 通道信息</view>
            <view>alpha-direction='left'</view>
        </view>
        <view class="video-area">
            <animation-video
                path="{{leftAlphaSrcPath}}"
                loop="{{loop}}"
                resource-width="800"
                resource-height="400"
                canvas-style="width:200px;height:200px"
                autoplay="{{autoplay}}"
                bindstarted="onStarted"
                bindended="onEnded"
                alpha-direction='left'
            ></animation-video>
        </view>
    </view>
</view>
```



```
Page({
 data: {
   loop: true,
   leftAlphaSrcPath: 'https://efe-h2.cdn.bcebos.com/ceug/resource/res/2020-1/15779
   rightAlphaSrcPath: 'https://b.bdstatic.com/miniapp/assets/docs/alpha-right-exam
   autoplay: true
 },
 onStarted() {
   console.log('===onStarted');
 },
 onEnded() {
```

```
    console.log('===onEnded');
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司


animation-video
Animation alpha channel information on the right side of the video
alpha-direction='right'
Animation alpha channel information on the left side of the video
alpha-direction='left'
Functionality

camera

功能说明：系统相机。该组件是 原生组件，使用时请注意相关限制。相关 API 请参见wx.createCameraContext。

参数及说明：

属性	类型	合法值及说明	默认值	说明
mode	string	normal：相机模式 scanCode：扫码模式	normal	应用模式，只在初始化时有效，不能动态变更
device-position	string	front：前置 back：后置	back	摄像头朝向
flash	string	auto：自动 on：打开 off：关闭 torch：常亮	auto	闪光灯，值为 auto, on, off
bindstop	eventhandle	-	-	摄像头在非正常终止时触发，如退出后台等情况
binderror	eventhandle	-	-	用户不允许使用摄像头时触发
bindscancode	eventhandle	-	-	在扫码识别成功时触发，仅在 mode="scanCode" 时生效

注意：

需要“用户授权”：scope.camera。

同一页面只能插入一个 camera 组件。

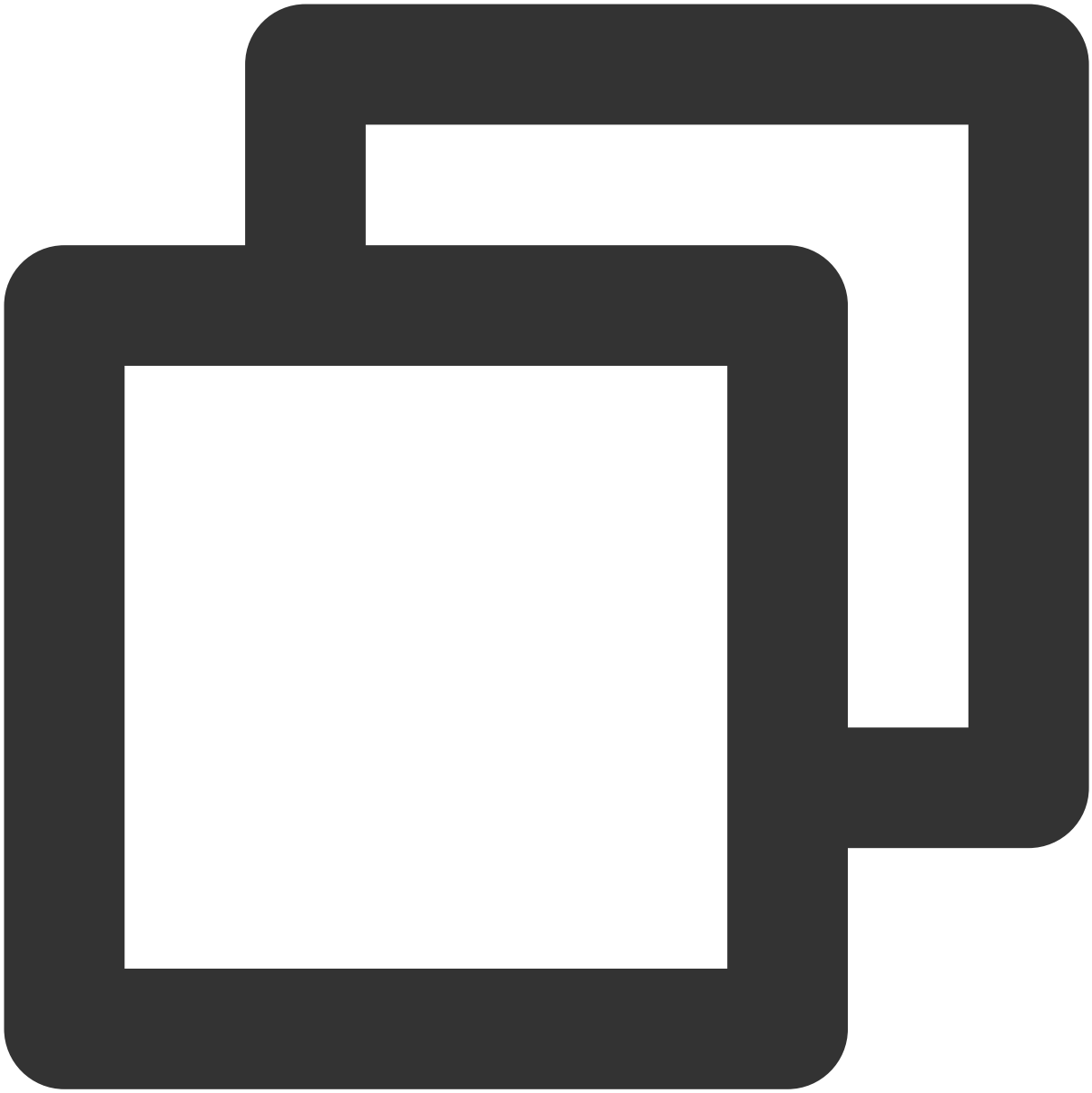
onCameraFrame 接口根据 frame-size 返回不同尺寸的原始帧数据，与 Camera 组件展示的图像不同，其实际像素值由系统决定。

示例代码：

WXML

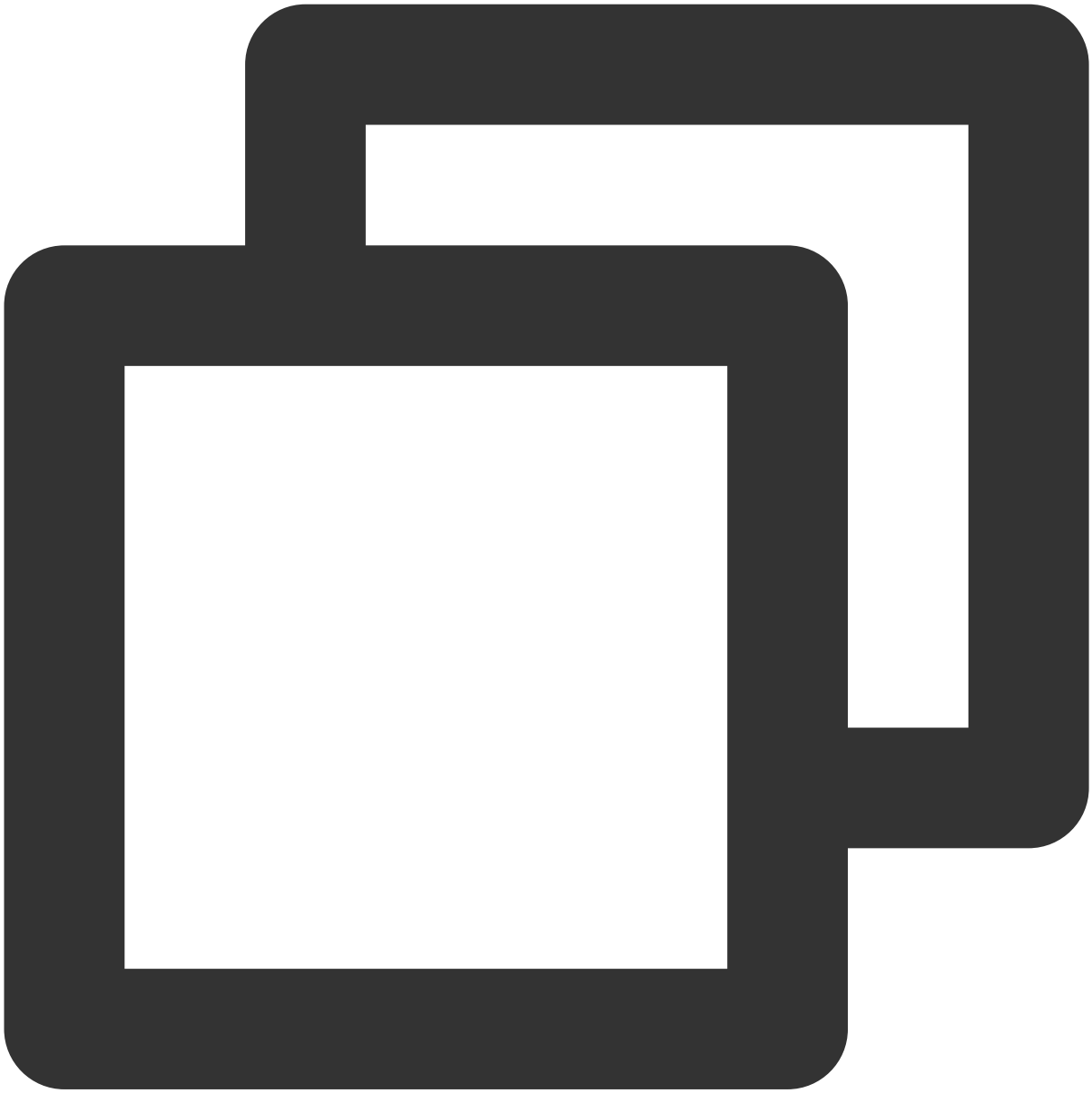
JAVASCRIPT

版权所有：腾讯云计算（北京）有限责任公司



```
<camera
  device-position="back"
  flash="off"
  binderror="error"
  style="width: 100%; height: 300px;"
></camera>
<button type="primary" bindtap="takePhoto">拍照</button>
<view>预览</view>
<image mode="widthFix" src="{{src}}"></image>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  takePhoto() {
    const ctx = wx.createCameraContext()
    ctx.takePhoto({
      quality: 'high',
      success: (res) => {
        this.setData({
          src: res.tempImagePath
        })
      }
    })
```

```
  },
  error(e) {
    console.log(e.detail)
  }
})
```

live-player

功能说明：实时音视频播放。该组件是原生组件，使用时请注意 原生组件。

参数及说明：

属性	类型	合法值及说明	默认值	说明
src	string	-	-	音视频地址。目前仅支持 flv, rtmp 格式
mode	string	live：直播 RTC：实时通话	live	模式
autoplay	boolean	-	false	自动播放
muted	boolean	-	false	是否静音
orientation	string	vertical：竖直 horizontal：水平	vertical	画面方向
object-fit	string	contain：图像长边填满屏幕，短边区域会被填充黑色 fillCrop：图像铺满屏幕，超出显示区域的部分将被截掉	contain	填充模式，可选值有 contain, fillCrop
min-cache	number	-	1	最小缓冲区，单位 s（RTC 模式推荐 0.2s）
max-cache	number	-	3	最大缓冲区，单位 s（RTC 模式推荐 0.8s）。缓冲区用来抵抗网络波动，缓冲数据越多，网络抗性越好，但时延越大。

sound-mode	string	speaker：扬声器 ear：听筒	speaker	声音输出方式
auto-pause-if-navigate	boolean	-	true	当跳转到其它小程序页面时，是否自动暂停本页面的实时音视频播放
auto-pause-if-open-native	boolean	-	true	当跳转到其它宿主APP原生页面时，是否自动暂停本页面的实时音视频播放
picture-in-picture-mode	string/Array	[]：取消小窗 push：路由 push 时触发小窗 pop：路由 pop 时触发小窗	-	设置小窗模式： push, pop, 空字符串或通过数组形式设置多种模式（如： ["push", "pop"]）
enable-auto-rotation	boolean	-	false	是否开启手机横屏时自动全屏，当系统设置开启自动旋转时生效
enable-metadata	boolean	-	false	是否回调 metadata
bindstatechange	eventhandle	-	-	播放状态变化事件，detail = {code}
bindfullscreenchange	eventhandle	-	-	全屏变化事件，detail = {direction, fullScreen}
bindnetstatus	eventhandle	-	-	网络状态通知，detail = {info}
bindaudiovolumenotify	eventhandle	-	-	播放音量大小通知，detail = {}
bindenterpictureinpicture	eventhandle	-	-	播放器进入小窗
bindleavepictureinpicture	eventhandle	-	-	播放器退出小窗
bindmetadatachange	eventhandle	-	-	metadata通知，detail = {info}

状态码

代码	说明
2001	已经连接服务器

版权所有：腾讯云计算（北京）有限责任公司

2002	已经连接服务器，开始拉流
2003	网络接收到首个视频数据包（IDR）
2004	视频开始播放
2006	视频播放结束
2007	视频播放 Loading
2008	解码器启动
2009	视频分辨率改变
-2301	网络断连，且多次重连无效，更多重试请自行重启播放
-2302	获取加速拉流地址失败
2101	当前视频帧解码失败
2102	当前音频帧解码失败
2103	网络断连，已启动自动重连
2104	网络来包不稳。可能是下行带宽不足，或由于主播端出流不均匀
2105	当前视频播放出现卡顿
2106	硬解启动失败，采用软解
2107	当前视频帧不连续，可能丢帧
2108	当前流硬解第一个I帧失败，SDK 自动切软解
3001	RTMP -DNS 解析失败
3002	RTMP 服务器连接失败
3003	RTMP 服务器握手失败
3005	RTMP 读/写失败

网络状态数据

键名	说明
videoBitrate	当前视频数据接收比特率，单位：kbps

版权所有：腾讯云计算（北京）有限责任公司

audioBitrate	当前音频数据接收比特率，单位：kbps
videoFPS	当前视频帧率
videoGOP	当前视频 GOP，即每两个关键帧（I帧）间隔时长，单位：s
netSpeed	当前的发送/接收速度
netJitter	网络抖动情况，抖动越大，网络越不稳定
videoWidth	视频画面的宽度
videoHeight	视频画面的高度

注意：

live-player 默认宽度300px、高度225px，可通过 wxss 设置宽高。

开发者工具上暂不支持。

小窗特性说明：

video 小窗支持以下三种触发模式（在组件上设置 picture-in-picture-mode 属性）：

push 模式，即从当前页跳转至下一页时出现小窗（页面栈 push）。

pop 模式，即离开当前页面时触发（页面栈 pop）。

以上两种路由行为均触发小窗。

此外，小窗还支持以下特性：

小窗容器尺寸会根据原组件尺寸自动判断。

点击小窗，用户会被导航回小窗对应的播放器页面。

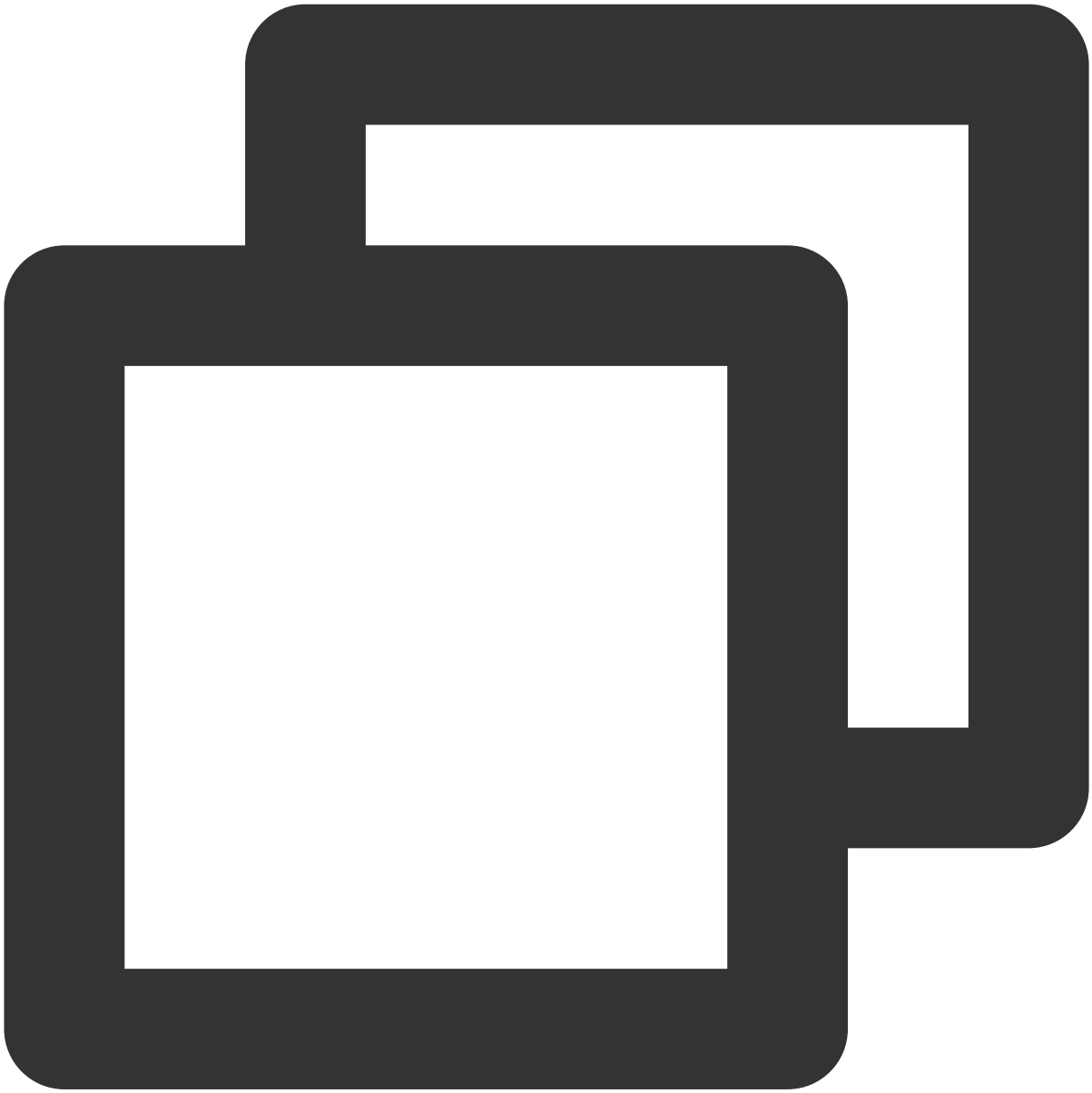
小窗出现后，用户可点击小窗右上角的关闭按钮或调用 context.exitPictureInPicture() 接口关闭小窗。

当播放器进入小窗模式后，播放器所在页面处于 hide 状态(触发 onHide 生命周期)，该页面不会被销毁。当小窗被关闭时，播放器所在页面会被 unload (触发 onUnload 生命周期)。

示例代码：

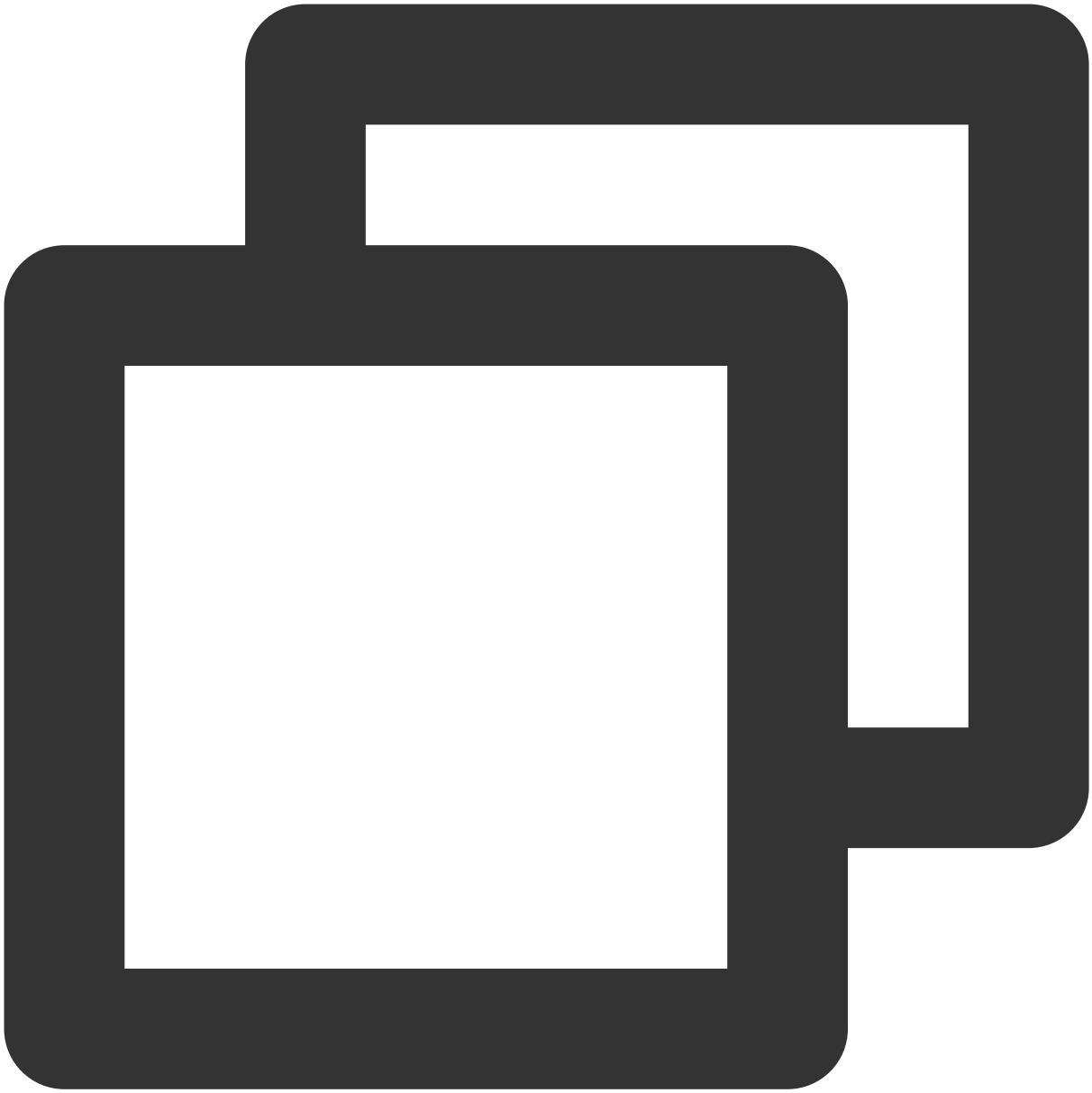
WXML

JAVASCRIPT



```
<live-player
    src="https://domain/pull_stream"
    mode="RTC"
    autoplay
    bindstatechange="statechange"
    binderror="error"
    style="width: 300px; height: 225px;"
/>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  statechange(e) {
    console.log('live-player code:', e.detail.code)
  },
  error(e) {
    console.error('live-player error:', e.detail.errMsg)
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

live-pusher

功能说明：实时音视频录制（暂不支持同层渲染）。

参数及说明：

属性	类型	合法值及说明	默认值	说明
url	string	-	-	推流地址。目前仅支持 rtmp
mode	string	-	RTC	SD（标清）, HD（高清）, FHD（超清）, RTC（实时通话）
autopush	boolean	-	false	自动推流
muted	boolean	-	false	是否静音
enable-camera	boolean	-	true	开启摄像头
auto-focus	boolean	-	true	自动聚集
orientation	string	vertical：竖直 horizontal：水平	vertical	画面方向
beauty	number	-	0	美颜，取值范围 0-9 ，0 表示关闭
whiteness	number	-	0	美白，取值范围 0-9 ，0 表示关闭
aspect	string	-	9：16	宽高比，可选值有 3:4, 9:16
min-bitrate	number	-	200	最小码率
max-bitrate	number	-	1000	最大码率
audio-quality	string	-	height	高音质(48KHz)或低音质(16KHz)，值为high, low
waiting-image	string	-	-	进入后台时推流的等待画面
waiting-image-hash	string	-	-	等待画面资源的 MD5 值
zoom	boolean	-	false	调整焦距
device-position	string	-	-	前置或后置，值为 front，back

版权所有：腾讯云计算（北京）有限责任公司

background-mute	boolean	-	false	进入后台时是否静音
mirror	boolean	-	false	设置推流画面是否镜像，产生的效果在 live-player 反应到
local-mirror	string	auto：前置摄像头镜像，后置摄像头不镜像 enable：前后置摄像头均镜像 disable：前后置摄像头均不镜像	auto	控制本地预览画面是否镜像
audio-reverb-type	number	0：关闭 1：KTV 2：小房间 3：大会堂 4：低沉 5：洪亮 6：金属声 7：磁性	0	音频混响类型
enable-mic	boolean	-	-	开启或关闭麦克风
enable-agc	boolean	-	false	是否开启音频自动增益
enable-ans	boolean	-	false	是否开启音频噪声抑制
audio-volume-type	string	auto：自动 media：媒体音量 voicecall：通话音量	auto	音量类型
video-width	number	-	360	上推的视频流的分辨率宽度
video-height	number	-	640	上推的视频流的分辨率高度
bindstatechange	eventhandle	-	-	状态变化事件，detail = {code}
bindnetstatus	eventhandle	-	-	网络状态通知，detail = {info}
binderror	eventhandle	-	-	渲染错误事件，detail = {errMsg, errCode}

版权所有：腾讯云计算（北京）有限责任公司

bindbgmstart	eventhandle	-	-	背景音开始播放时触发
bindbgmprogress	eventhandle	-	-	背景音进度变化时触发，detail = {progress, duration}
bindbgmcomplete	eventhandle	-	-	背景音播放完成时触发

注意：

需要“用户授权”scope.camera、scope.record。

开发者工具上暂不支持。

live-pusher 默认宽度为100%、无默认高度，请通过wxss设置宽高。

错误码（errCode）

代码	说明
10001	用户禁止使用摄像头
10002	用户禁止使用录音
10003	背景音资源（BGM）加载失败
10004	等待页面资源（waiting-image）加载失败

状态码（code）

代码	说明
1001	已经连接推流服务器
1002	已经与服务器握手完毕,开始推流
1003	打开摄像头成功
1004	录屏启动成功
1005	推流动态调整分辨率
1006	推流动态调整码率
1007	首帧画面采集完成
1008	编码器启动
-1301	打开摄像头失败
-1302	打开麦克风失败

版权所有：腾讯云计算（北京）有限责任公司

-1303	视频编码失败
-1304	音频编码失败
-1305	不支持的视频分辨率
-1306	不支持的音频采样率
-1307	网络断连，且经多次重连抢救无效，更多重试请自行重启推流
-1308	开始录屏失败，可能是被用户拒绝
-1309	录屏失败，不支持的 Android 系统版本，需要5.0以上的系统
-1310	录屏被其他应用打断了
-1311	Android Mic 打开成功，但是录不到音频数据
-1312	录屏动态切横竖屏失败
1101	网络状况不佳：上行带宽太小，上传数据受阻
1102	网络断连, 已启动自动重连
1103	硬编码启动失败,采用软编码
1104	视频编码失败
1105	新美颜软编码启动失败，采用老的软编码
1106	新美颜软编码启动失败，采用老的软编码
3001	RTMP -DNS 解析失败
3002	RTMP 服务器连接失败
3003	RTMP 服务器握手失败
3004	RTMP 服务器主动断开，请检查推流地址的合法性或防盗链有效期
3005	RTMP 读/写失败

网络状态数据（info）

键名	说明
videoBitrate	当前视频编/码器输出的比特率，单位：kbps
audioBitrate	当前音频编/码器输出的比特率，单位：kbps

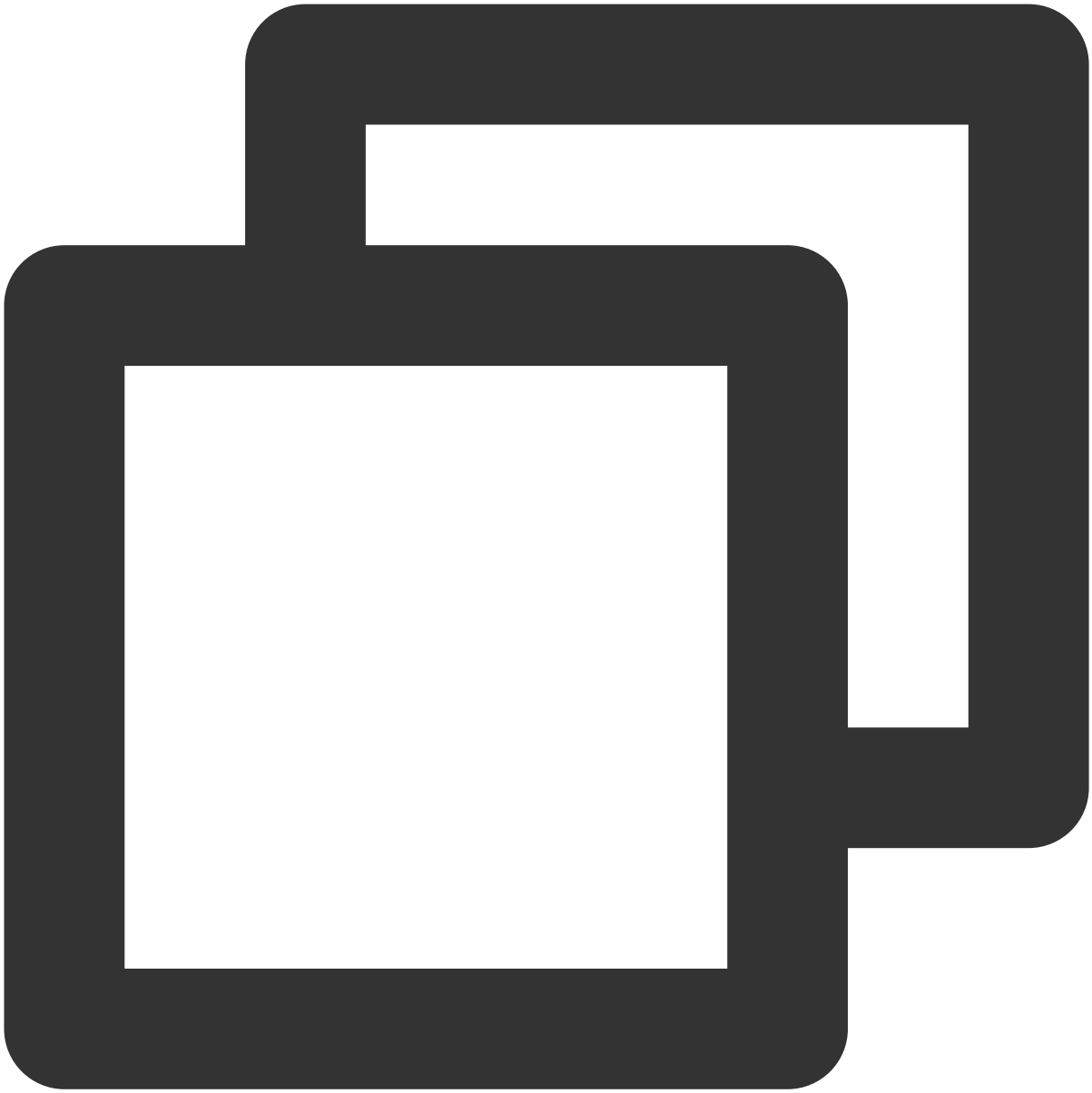
版权所有：腾讯云计算（北京）有限责任公司

videoFPS	当前视频帧率
videoGOP	当前视频 GOP，也就是每两个关键帧（I帧）间隔时长，单位：s
netSpeed	当前的发送/接收速度
netJitter	网络抖动情况，抖动越大，网络越不稳定
videoWidth	视频画面的宽度
videoHeight	视频画面的高度

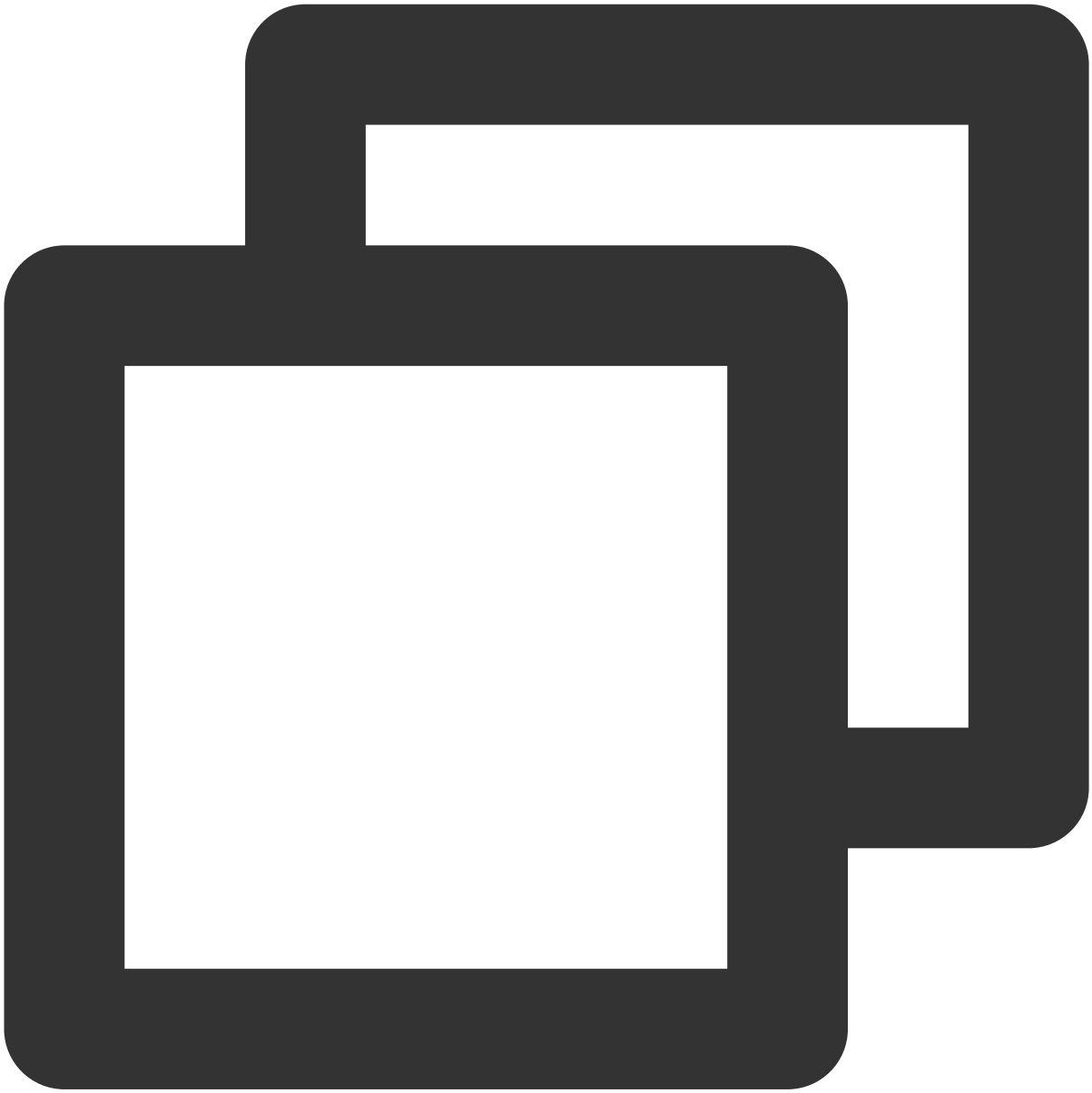
示例代码：

WXML

JAVASCRIPT



```
<live-pusher
    url="https://domain/push_stream"
    mode="RTC"
    autopush
    bindstatechange="statechange"
    style="width: 300px; height: 225px;"
/>
```



```
Page({
  statechange(e) {
    console.log('live-pusher code:', e.detail.code)
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

开放能力

最近更新时间：2024-07-12 18:14:03

web-view

功能说明：承载网页的容器。会自动铺满整个小程序页面。

参数及说明：

属性	类型	默认值	说明
src	string	-	webview 指向网页的链接。网页需登录小程序管理后台配置业务域名。
bindmessage	eventhandler	-	网页使用 wx.miniProgram.postMessage 方法向小程序发送消息时，小程序会在特定时机（小程序后退、组件销毁、分享）触发并收到消息。e.detail = { data }，data 是多次 postMessage 的参数组成的数组
bindload	eventhandler	-	网页加载成功时候触发此事件。e.detail = { src }
binderror	eventhandler	-	网页加载失败的时候触发此事件。e.detail = { url, fullUrl }，其中 fullUrl 为加载失败时的完整 url

注意：

网页内 iframe 的域名也需要配置到域名白名单。

开发者工具上，可以在 <web-view> 组件上击调试按钮，打开 <web-view> 组件的调试。

每个页面只能有一个 <web-view>，<web-view> 会自动铺满整个页面，并覆盖其他组件。

<web-view> 网页与小程序之间不支持除 JSSDK 提供的接口之外的通信。

避免在链接中带有中文字符，在 iOS 中会有打开白屏的问题，建议加一下 encodeURIComponent。

相关接口1

`web-view` 网页中可使用 JSSDK 提供的接口返回小程序页面。 支持的接口有：

接口名	说明
wx.miniProgram.navigateTo	参数与小程序接口一致
wx.miniProgram.navigateBack	参数与小程序接口一致
wx.miniProgram.switchTab	参数与小程序接口一致

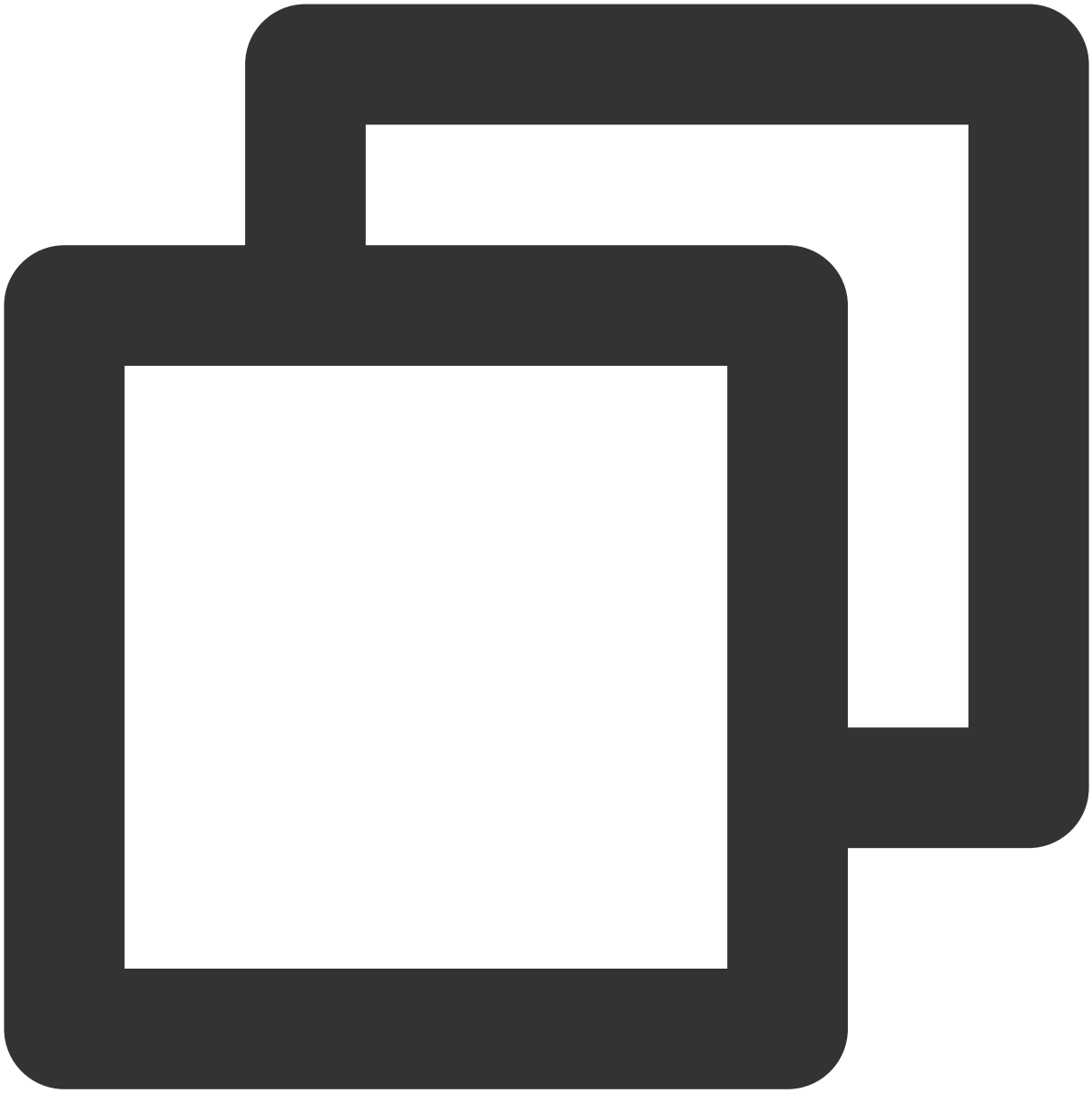
版权所有：腾讯云计算（北京）有限责任公司

wx.miniProgram.reLaunch	参数与小程序接口一致
wx.miniProgram.redirectTo	参数与小程序接口一致
wx.miniProgram.postMessage	向小程序发送消息，会在以下特定时机触发组件的 message 事件：小程序后退、组件销毁、分享、复制链接
wx.miniProgram.getEnv	获取当前环境

示例代码：

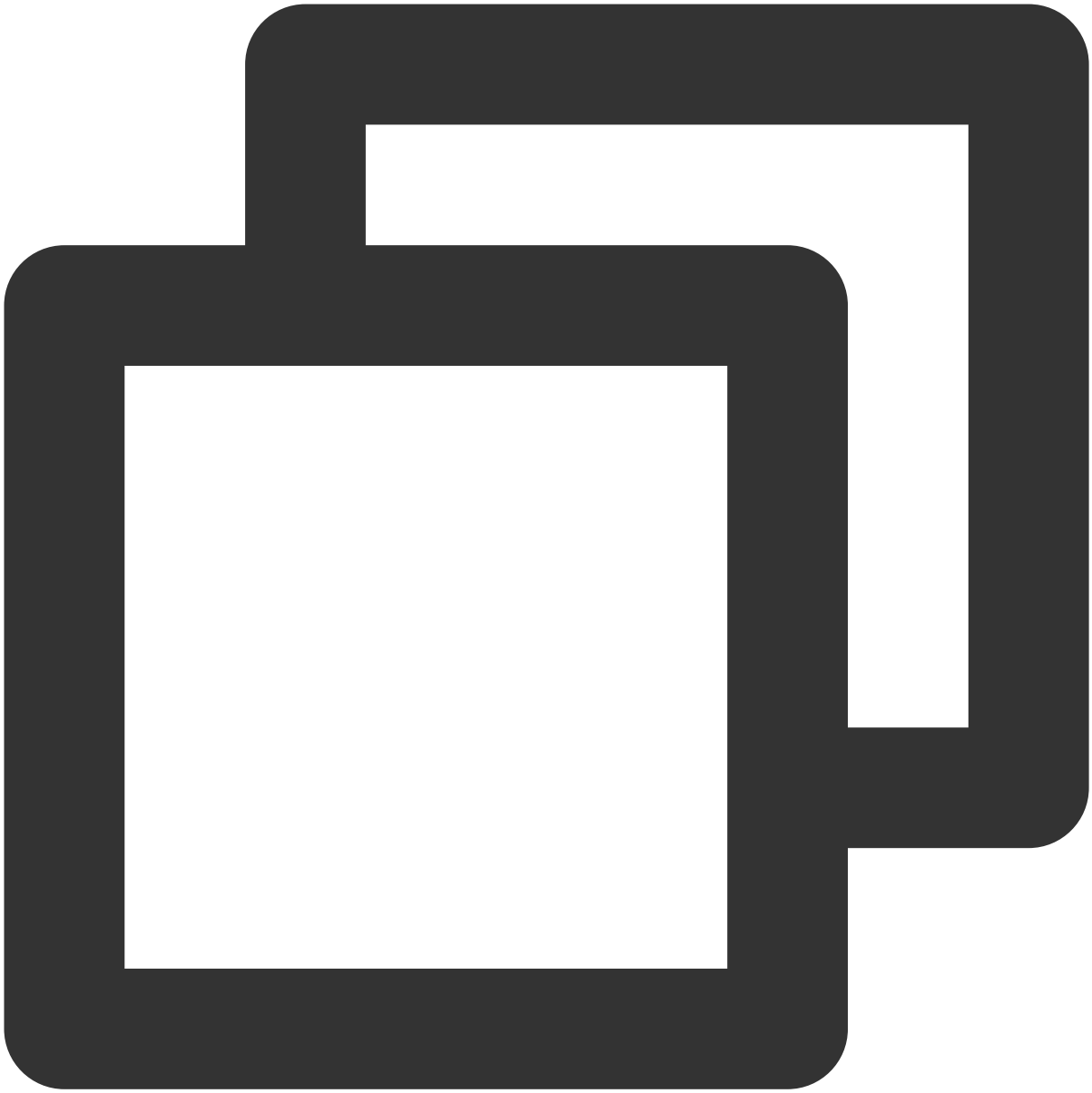
HTML

JAVASCRIPT



```
<script type="text/javascript" src="https://github.com/TencentCloud/tcmpp-demo-mini
```

版权所有：腾讯云计算（北京）有限责任公司

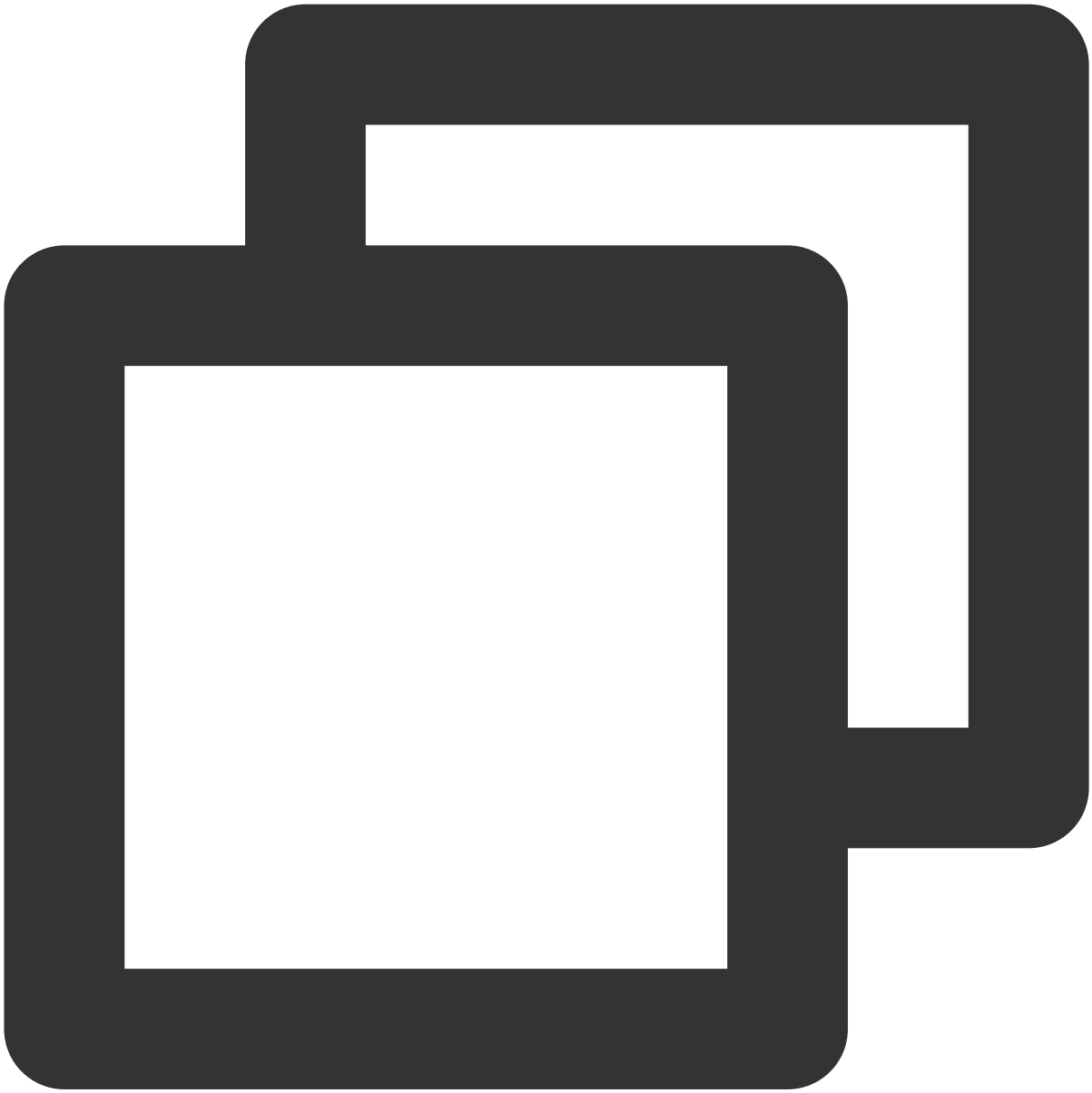


```
wx.miniProgram.navigateTo({url: '/path/to/page'})
wx.miniProgram.postMessage({ data: 'foo' })
wx.miniProgram.postMessage({ data: {foo: 'bar'} })
wx.miniProgram.getEnv(function(res) { console.log(res.miniprogram) })
```

相关接口2

用户分享时可获取当前 `web-view` 的 URL，即在 `onShareAppMessage` 回调中返回 `webViewUrl` 参数。

示例代码：

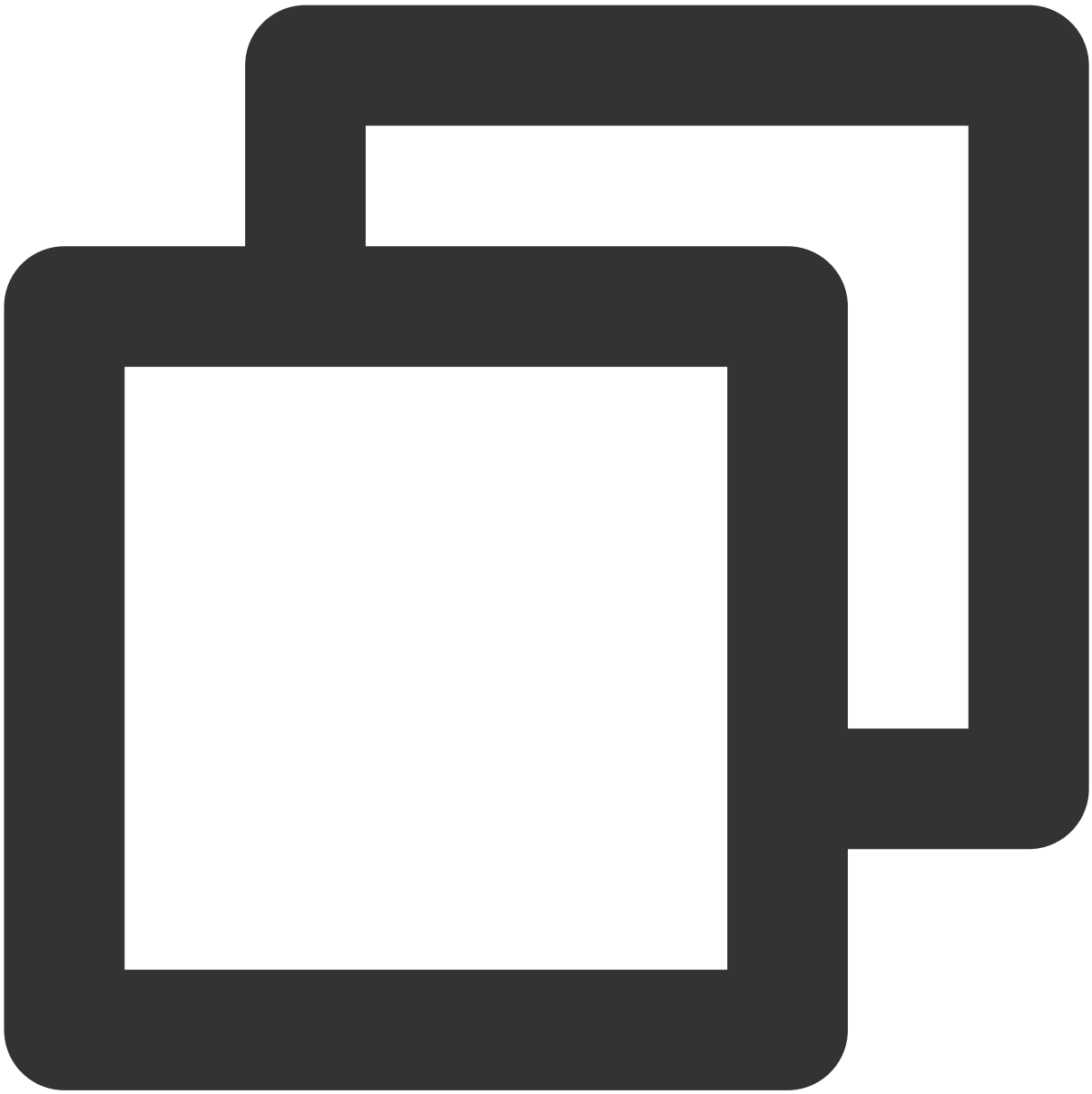


```
Page({
  onShareAppMessage(options) {
    console.log(options.webViewUrl)
  }
})
```

相关接口3

在网页内可通过 `window.__wxjs_environment` 变量判断是否在小程序环境，建议在 `WeixinJSBridgeReady` 回调中使用，也可以使用 JSSDK 提供的 `getEnv` 接口。

示例代码：



```
// web-view下的页面内
function ready() {
  console.log(window.__wxjs_environment === 'miniprogram') // true
}
if (!window.QQJSBridge || !QQJSBridge.invoke) {
  document.addEventListener('WeixinJSBridgeReady', ready, false)
} else {
  ready()
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 或者
wx.miniProgram.getEnv(function(res) {
  console.log(res.miniprogram) // true
})
```

相关接口4

可以通过判断 `userAgent` 中包含 `miniProgram` 字样来判断小程序 `web-view` 环境。

Canvas 画布

最近更新时间：2024-07-12 18:13:24

功能说明：画布。该组件是 原生组件，使用时请注意相关限制。相关 API wx.createCanvasContext。现支持新 Canvas 2D 接口，旧版接口不再维护。

参数及说明：

属性名	类型	默认值	说明
type	string	2d	支持 canvas 类型，目前仅支持2d
canvas-id	string	-	canvas 组件的唯一标识符
disable-scroll	boolean	false	当在 canvas 中移动时且有绑定手势事件时，禁止屏幕滚动以及下拉刷新
bindtouchstart	eventhandle	-	手指触摸动作开始
bindtouchmove	eventhandle	-	手指触摸后移动
bindtouchend	eventhandle	-	手指触摸动作结束
bindtouchcancel	eventhandle	-	手指触摸动作被打断，如来电提醒，弹窗
bindlongtap	eventhandle	-	手指长按500ms之后触发，触发了长按事件后进行移动不会触发屏幕的滚动
binderror	eventhandle	-	当发生错误时触发 error 事件，detail = {errMsg: 'something wrong'}

注意：

canvas 标签默认宽度300px、高度150px。

同一页面中的 canvas-id 不可重复，如果使用一个已经出现过的 canvas-id，该 canvas 标签对应的画布将被隐藏并不再正常工作。

避免设置过大的宽高，在安卓下会有crash的问题。

旧版 canvas 接口使用 canvas-id 属性唯一标识 canvas；新版 Canvas 2D 可直接使用 id 标识。

旧版 canvas 接口使用 wx.createCanvasContext 同步获取 CanvasContext。新版 Canvas 2D 接口需要先通过 SelectorQuery 异步获取 Canvas 对象，再通过 Canvas.getContext 获取渲染上下文 RenderingContext。

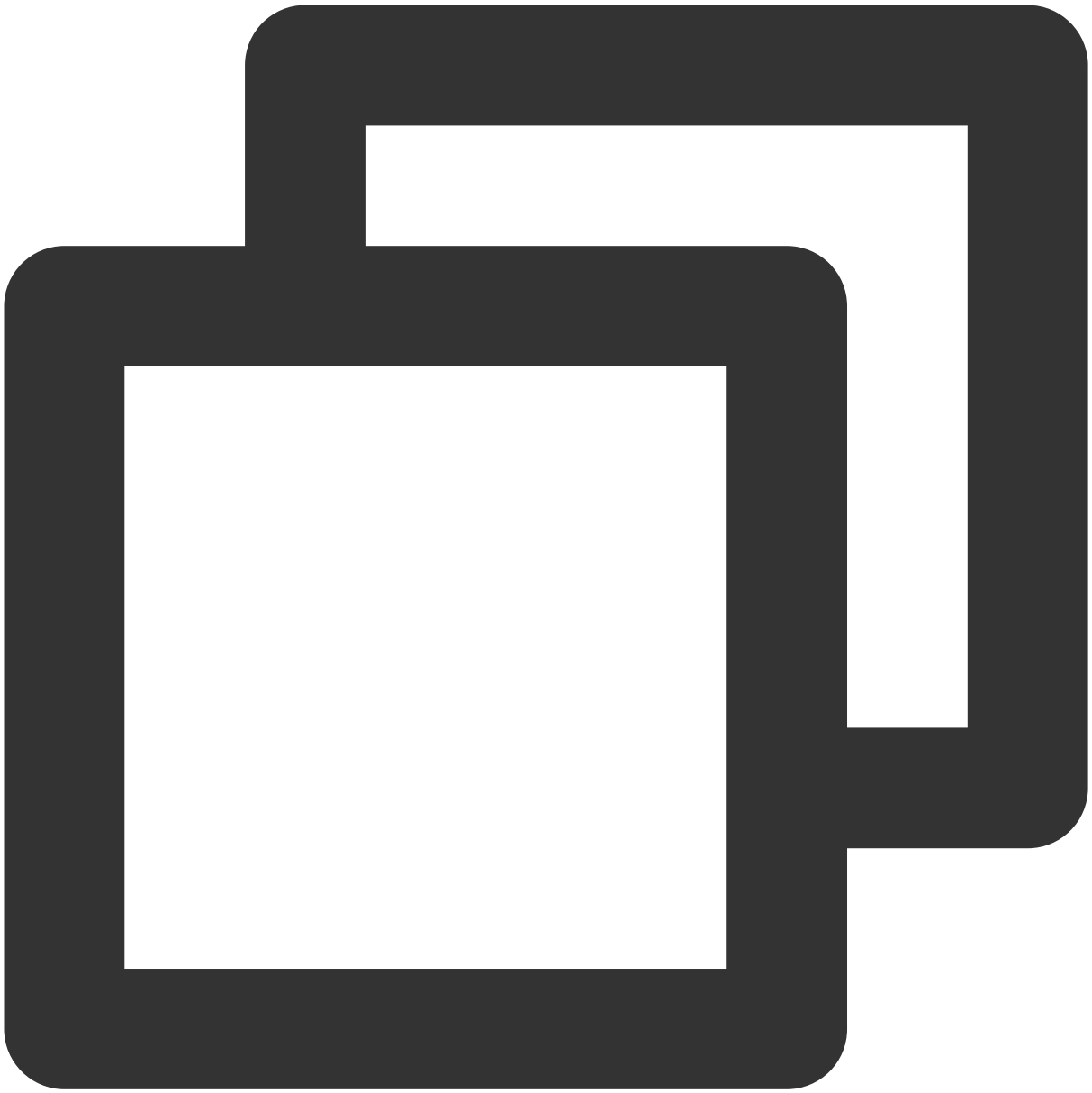
旧版 canvas 接口的画布大小是根据实际渲染宽度决定的，开发者无法修改。新版 Canvas 2D 接口允许开发者自由修改画布的逻辑大小，默认宽高为 300*150。不同的设备上，存在物理像素和逻辑像素不相等的情况，所以一般我们需要用 wx.getWindowInfo 获取设备的像素比，乘上 canvas 的实际大小。

示例代码（新版）：

版权所有：腾讯云计算（北京）有限责任公司

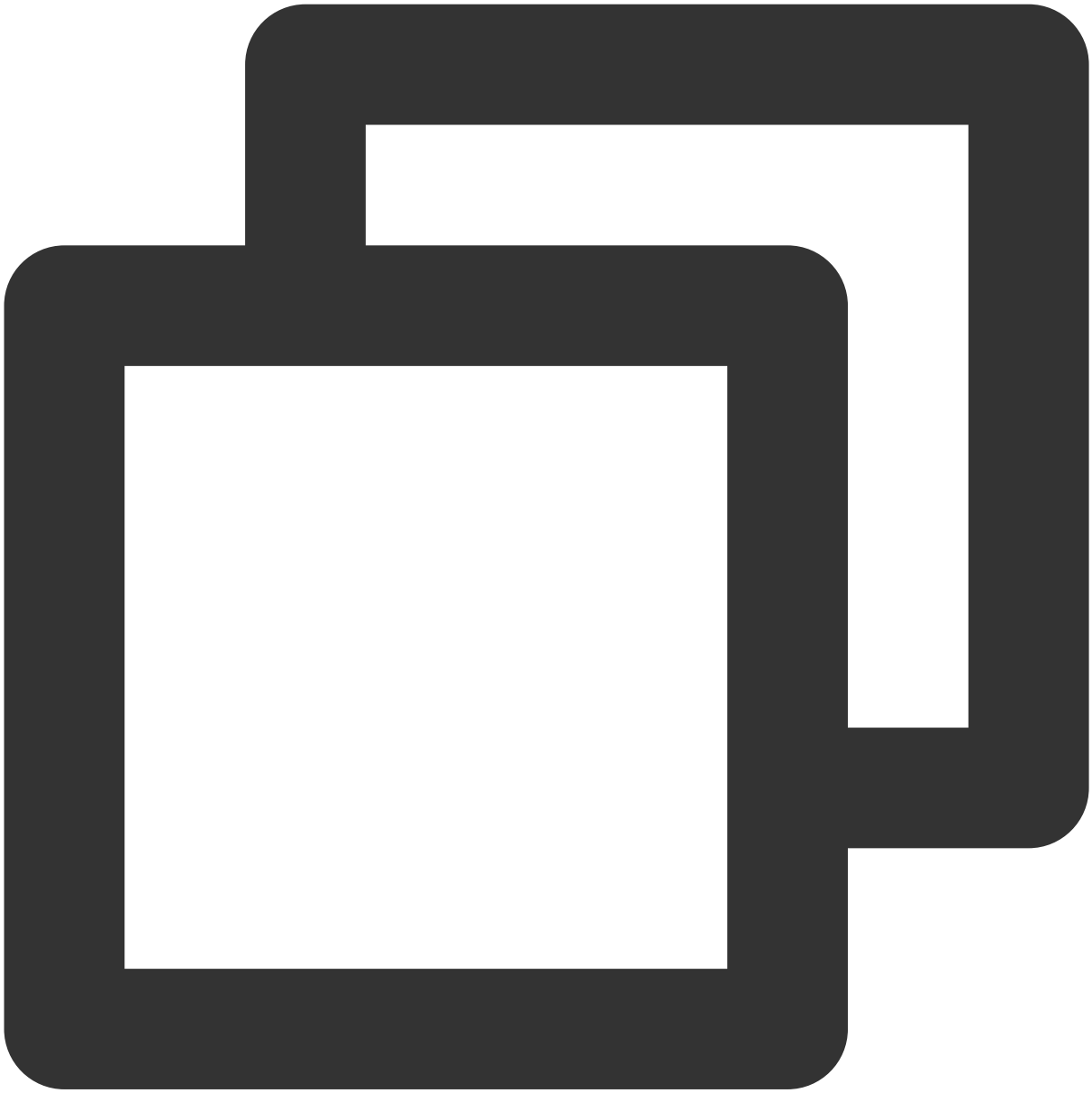
WXML

JAVASCRIPT



```
<canvas type="2d" id="myCanvas"></canvas>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  onReady() {
    const query = wx.createSelectorQuery()
    query.select('#myCanvas')
      .fields({ node: true, size: true })
      .exec((res) => {
        const canvas = res[0].node
        const ctx = canvas.getContext('2d')

        const dpr = wx.getSystemInfoSync().pixelRatio
        canvas.width = res[0].width * dpr
```

版权所有：腾讯云计算（北京）有限责任公司

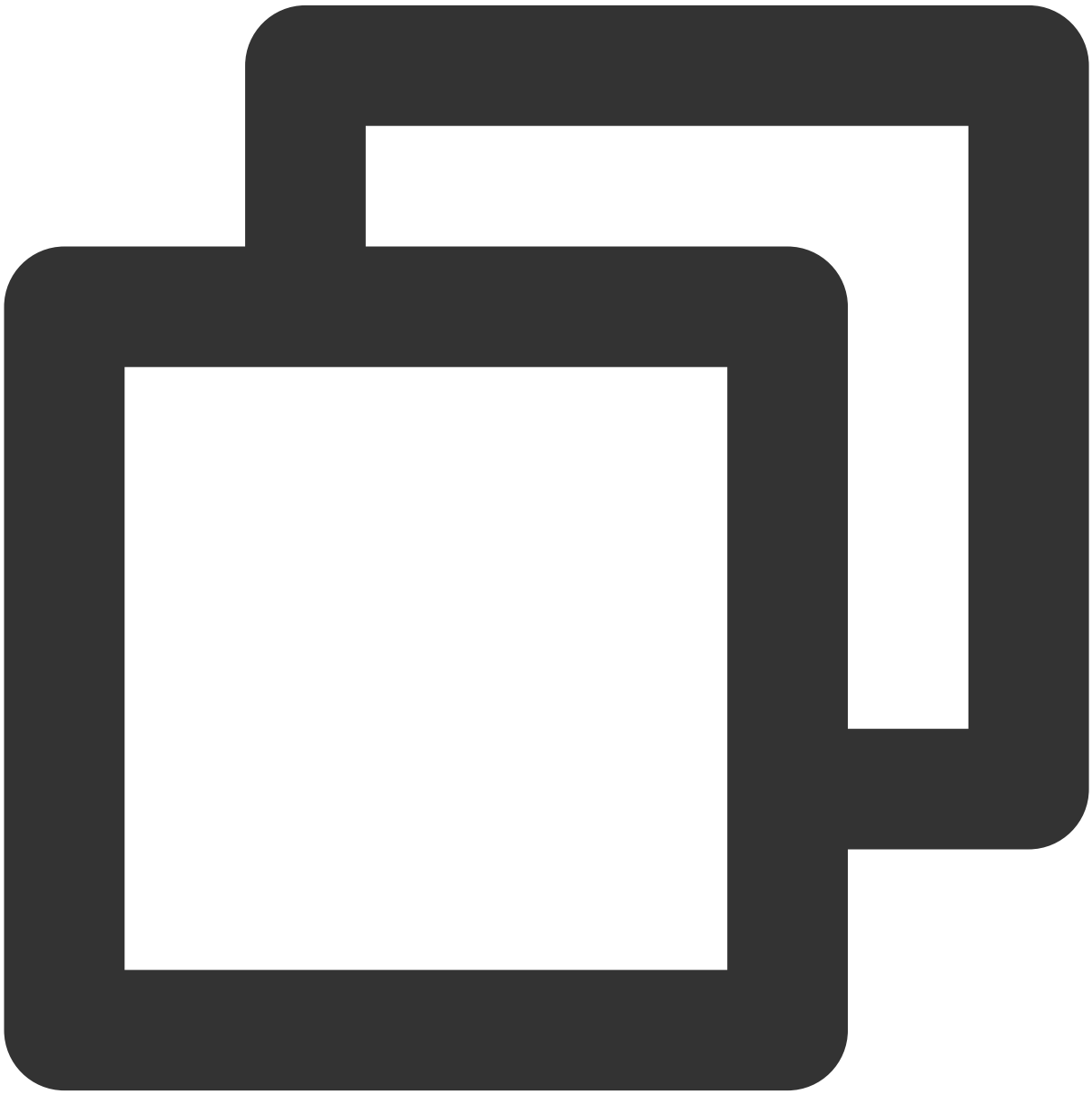
```
        canvas.height = res[0].height * dpr
        ctx.scale(dpr, dpr)

        ctx.fillRect(0, 0, 100, 100)
      })
  }
})
```

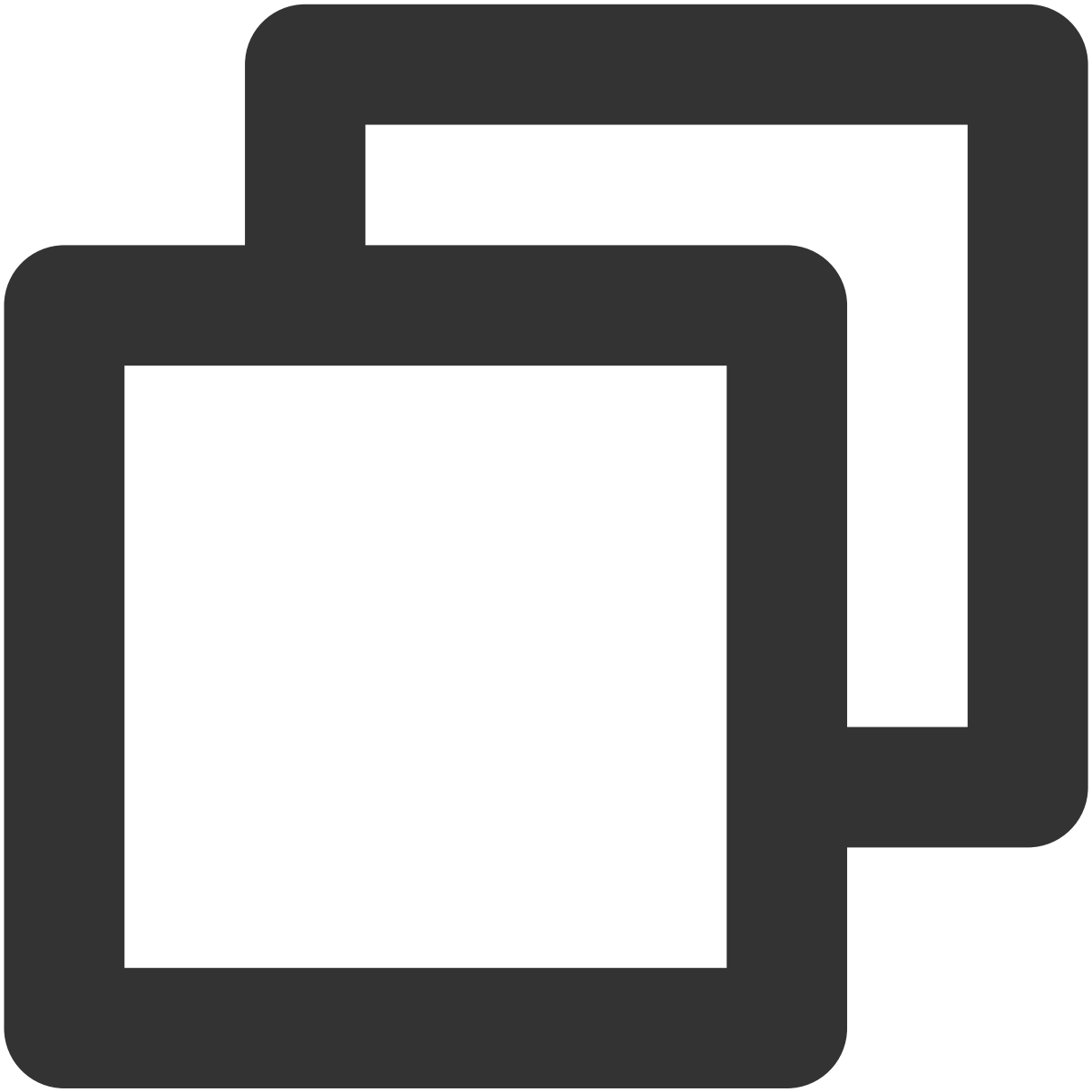
示例代码（旧版）：

WXML

JAVASCRIPT



```
<!-- canvas.wxml -->
<canvas style="width: 300px; height: 200px;" canvas-id="firstCanvas"></canvas>
<!-- 当使用绝对定位时，文档流后边的 canvas 的显示层级高于前边的 canvas -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas"></canvas>
<!-- 因为 canvas-id 与前一个 canvas 重复，该 canvas 不会显示，并会发送一个错误事件到 AppServ
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas" binderror="ca
```



```
Page({
  canvasIdErrorCallback: function (e) {
    console.error(e.detail.errMsg)
```

版权所有：腾讯云计算（北京）有限责任公司

```
  },
  onReady: function (e) {
    // 使用 wx.createContext 获取绘图上下文 context
    var context = wx.createCanvasContext('firstCanvas')

    context.setStrokeStyle("#00ff00")
    context.setLineWidth(5)
    context.rect(0, 0, 200, 200)
    context.stroke()
    context.setStrokeStyle("#ff0000")
    context.setLineWidth(2)
    context.moveTo(160, 100)
    context.arc(100, 100, 60, 0, 2 * Math.PI, true)
    context.moveTo(140, 100)
    context.arc(100, 100, 40, 0, Math.PI, false)
    context.moveTo(85, 80)
    context.arc(80, 80, 5, 0, 2 * Math.PI, true)
    context.moveTo(125, 80)
    context.arc(120, 80, 5, 0, 2 * Math.PI, true)
    context.stroke()
    context.draw()
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

导航组件

最近更新时间：2024-07-12 18:13:13

navigator

功能说明：页面链接。

参数及说明：

属性	类型	合法值及说明	默认值	必填	说明
target	string	self：当前小程序 miniProgram：其他小程序	self	否	在哪个目标上发生跳转，默认当
url	string	-	-	否	当前小程序内的跳转
open-type	string	navigate：对应 wx.navigateTo 或 wx.navigateToMiniProgram 的功能 redirect：对应 wx.redirectTo 的功能 switchTab：对应 wx.switchTab 的功能 reLaunch：对应 wx.reLaunch 的功能 navigateBack：对应 wx.navigateBack 的功能 exit：退出小程序，target="miniProgram"时生效	navigate	否	跳转方式
delta	number	-	1	否	当 open-type 为 'navigateBack' 退的层数
app-id	string	-	-	否	当 target="miniProgram"且 open type="navigate"时有效，要打开
path	string	-	-	否	当 target="miniProgram"时有效 径，如果为空则打开首页
extra-data	object	-	-	否	当 target="miniProgram"时有效 标小程序的数据，目标小程序可

版权所有：腾讯云计算（北京）有限责任公司

					在 App.onLaunch()， Ap 获取到这份数据。
version	string	develop：开发版 trial：体验版 release：正式版，仅在当前小程序为开发版或体验版时此参数有效；如果当前小程序是正式版，则打开的小程序必定是正式版	release	否	当 target="miniProgram"且 oper type="navigate"时有效，要打开
hover-class	string	-	navigator-hover	否	指定点击时的样式类，当 hove class="none" 时，没有点击
hover-stop-propagation	boolean	-	false	否	指定是否阻止本节点的祖先节点
hover-start-time	number	-	50	否	按住后多久出现点击态，单位毫
hover-stay-time	number	-	600	否	手指松开后点击态保留时间，单
bindsuccess	string	-	-	否	当 target="miniProgram"时有效 功
bindfail	string	-	-	否	当 target="miniProgram"时有效 败
bindcomplete	string	-	-	否	当 target="miniProgram"时有效 成

使用限制：需要用户确认跳转

在跳转至其他小程序前，将统一增加弹窗，询问是否跳转，用户确认后才可以跳转至其他小程序。如果用户单击取消，则回调 `fail cancel` 。

注意：

`navigator-hover` 默认为 `{background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}`，`<navigator>` 的子节点背景色应为透明色。

示例代码：

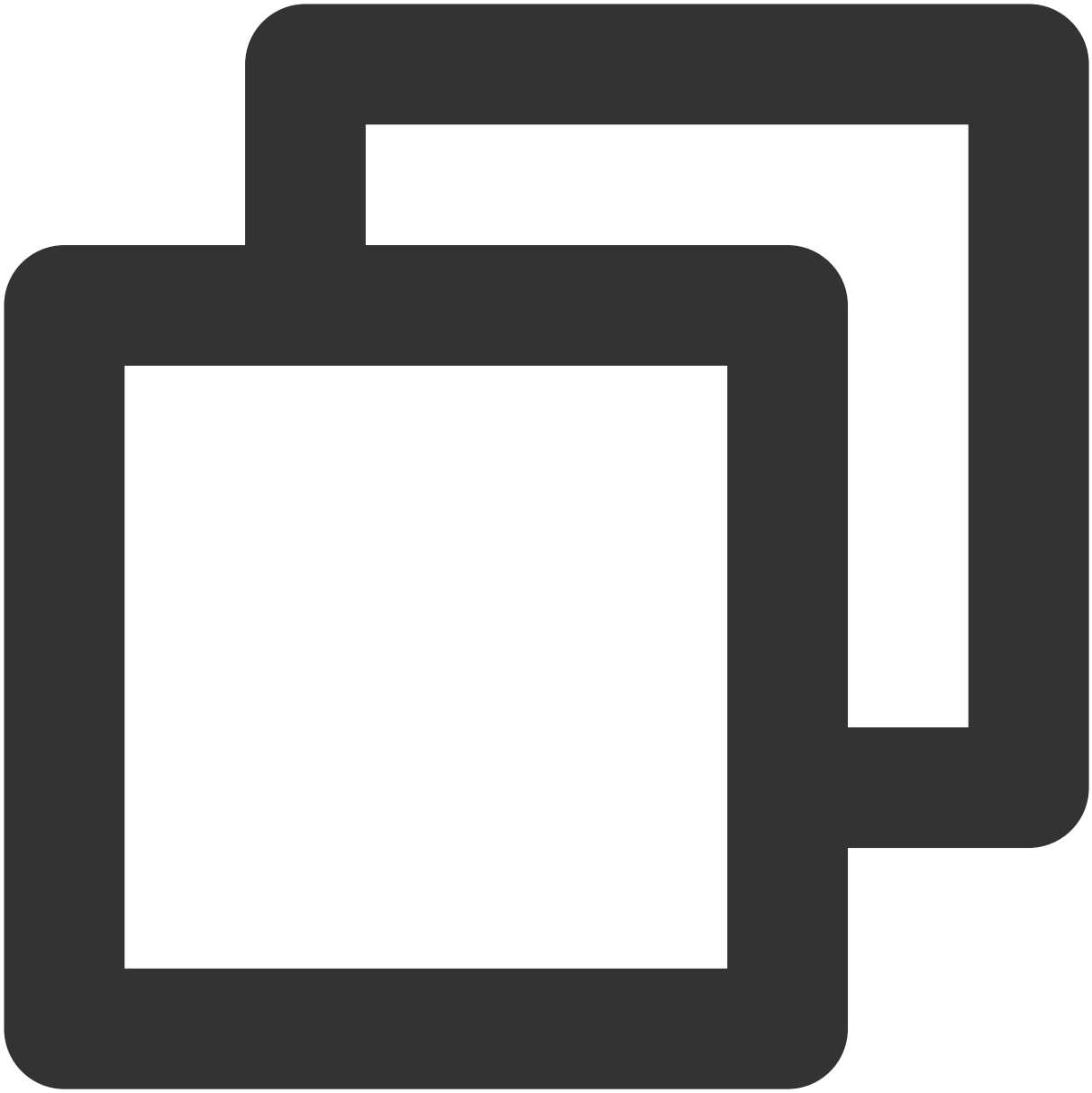
sample.wxml

navigator.wxml

redirect.wxml

JAVASCRIPT

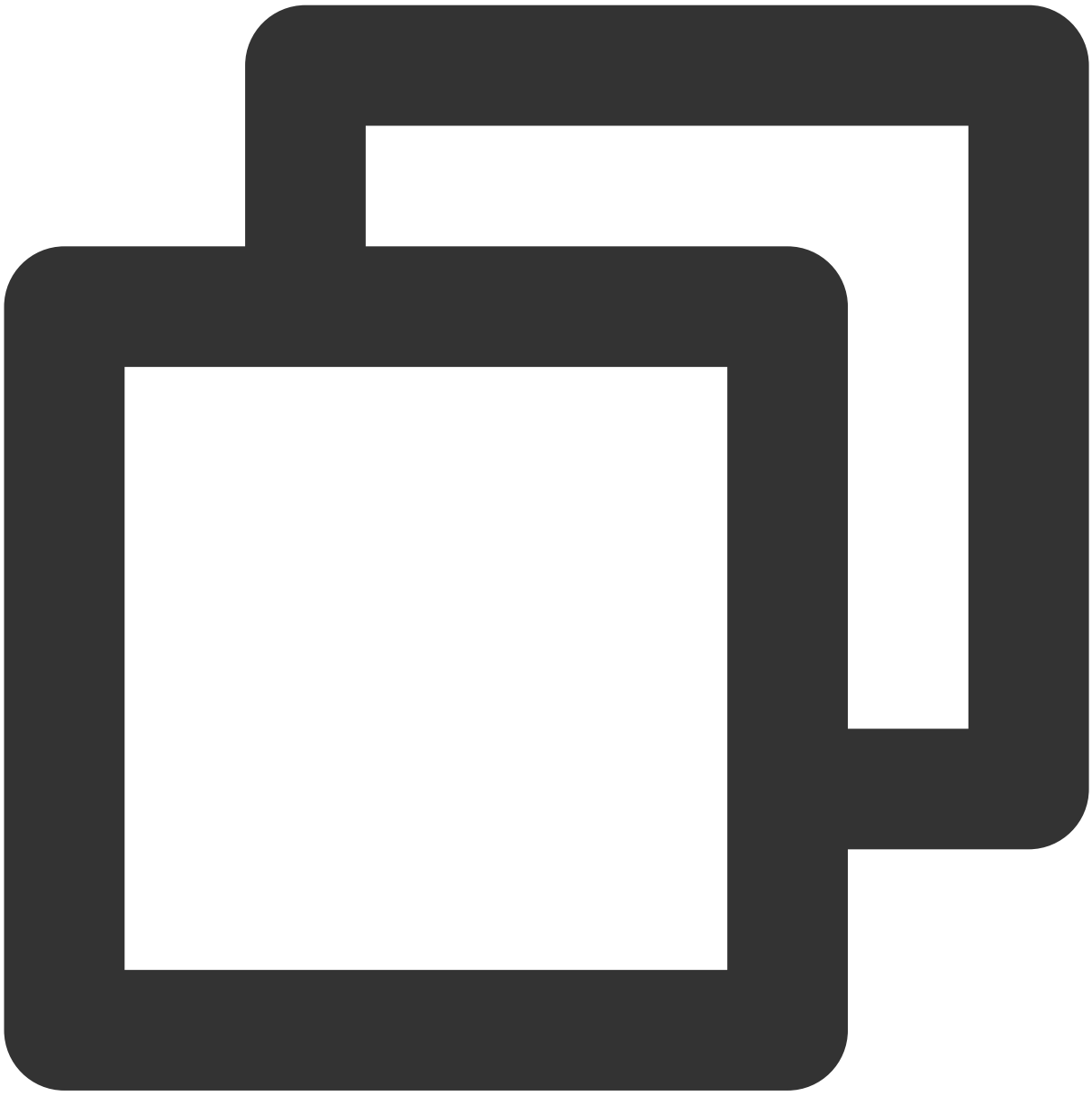
WXSS



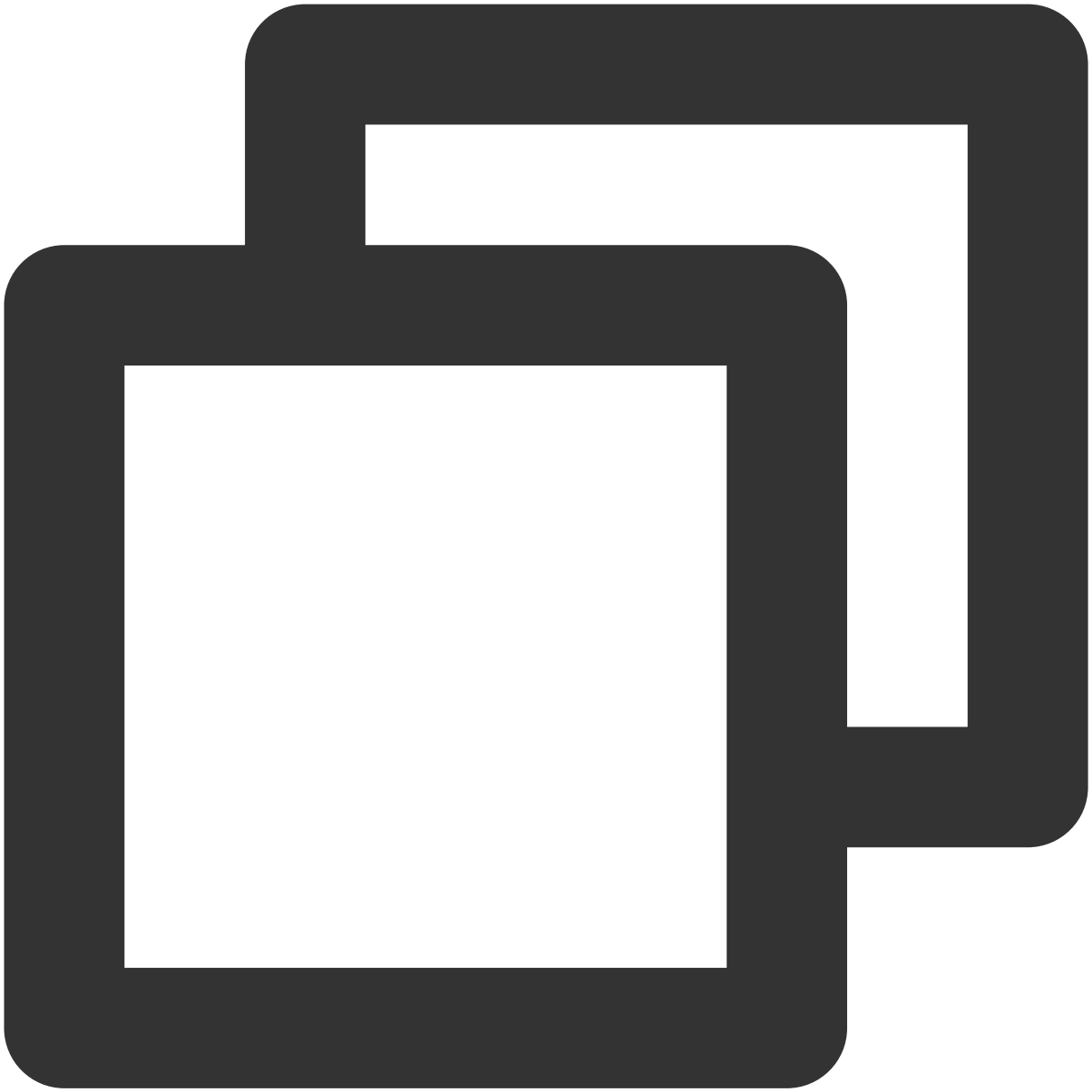
```
<view class="btn-area">
  <navigator
    url="/page/navigate/navigate?title=navigate"
    hover-class="navigator-hover"
  >
    跳转到新页面
  </navigator>
  <navigator
    url="../../redirect/redirect/redirect?title=redirect"
    open-type="redirect"
    hover-class="other-navigator-hover"
```

版权所有：腾讯云计算（北京）有限责任公司

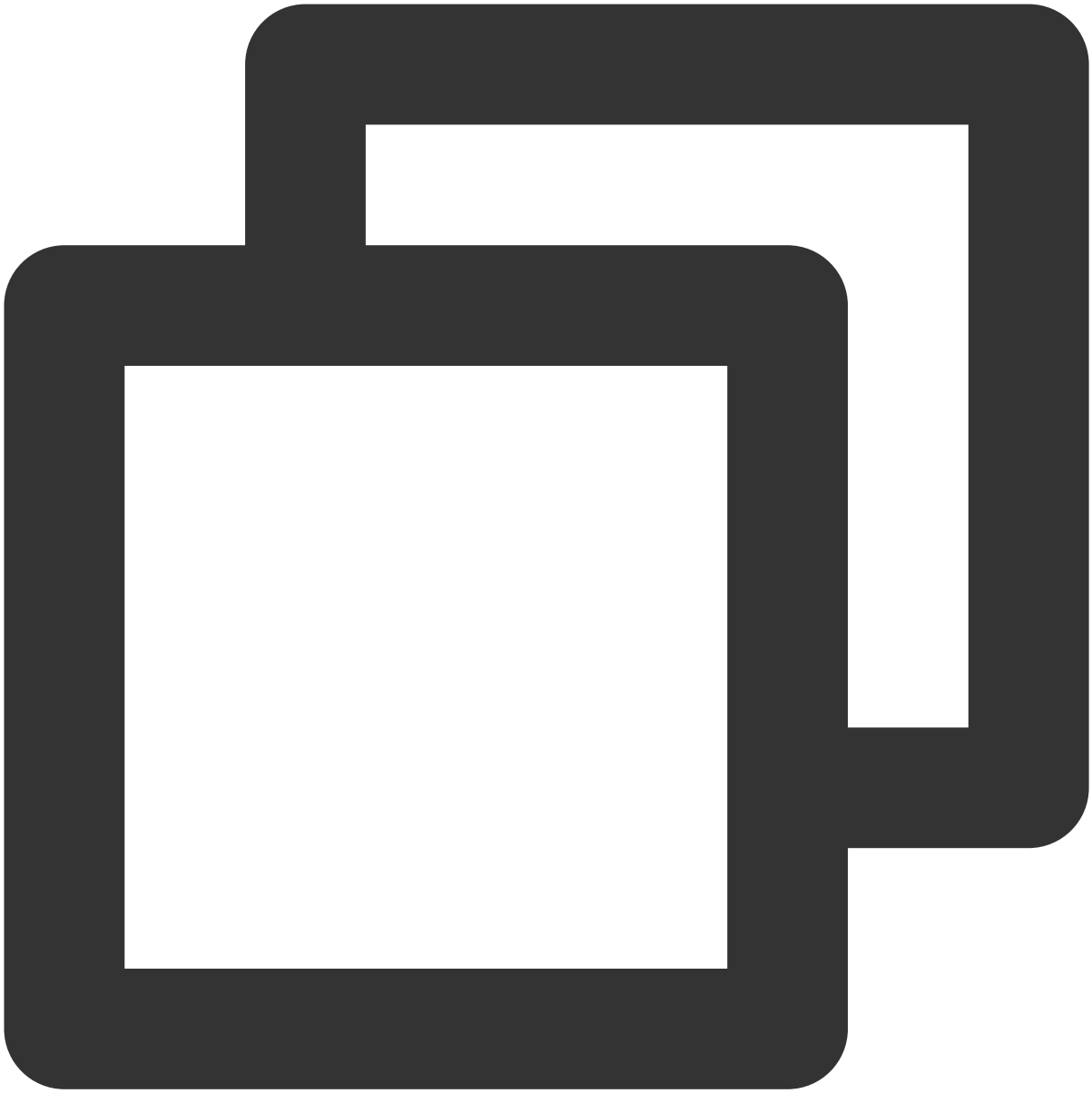
```
  >
    在当前页打开
  </navigator>
</view>
```



```
<view style="text-align:center"> {{title}} </view>
<view> 点击左上角返回回到之前页面 </view>
```

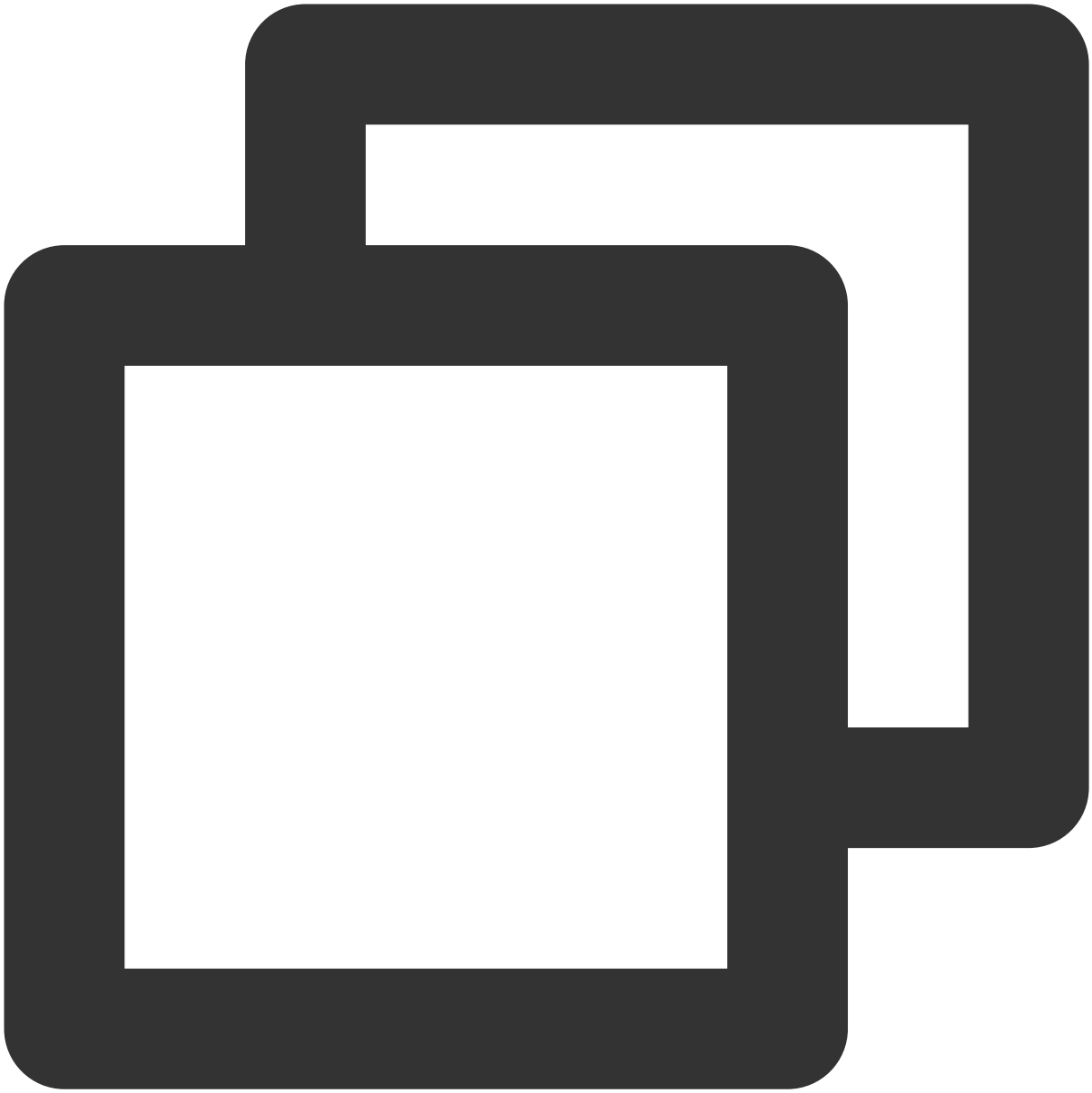


```
<view style="text-align:center"> {{title}} </view>
<view> 点击左上角返回回到上级页面 </view>
```



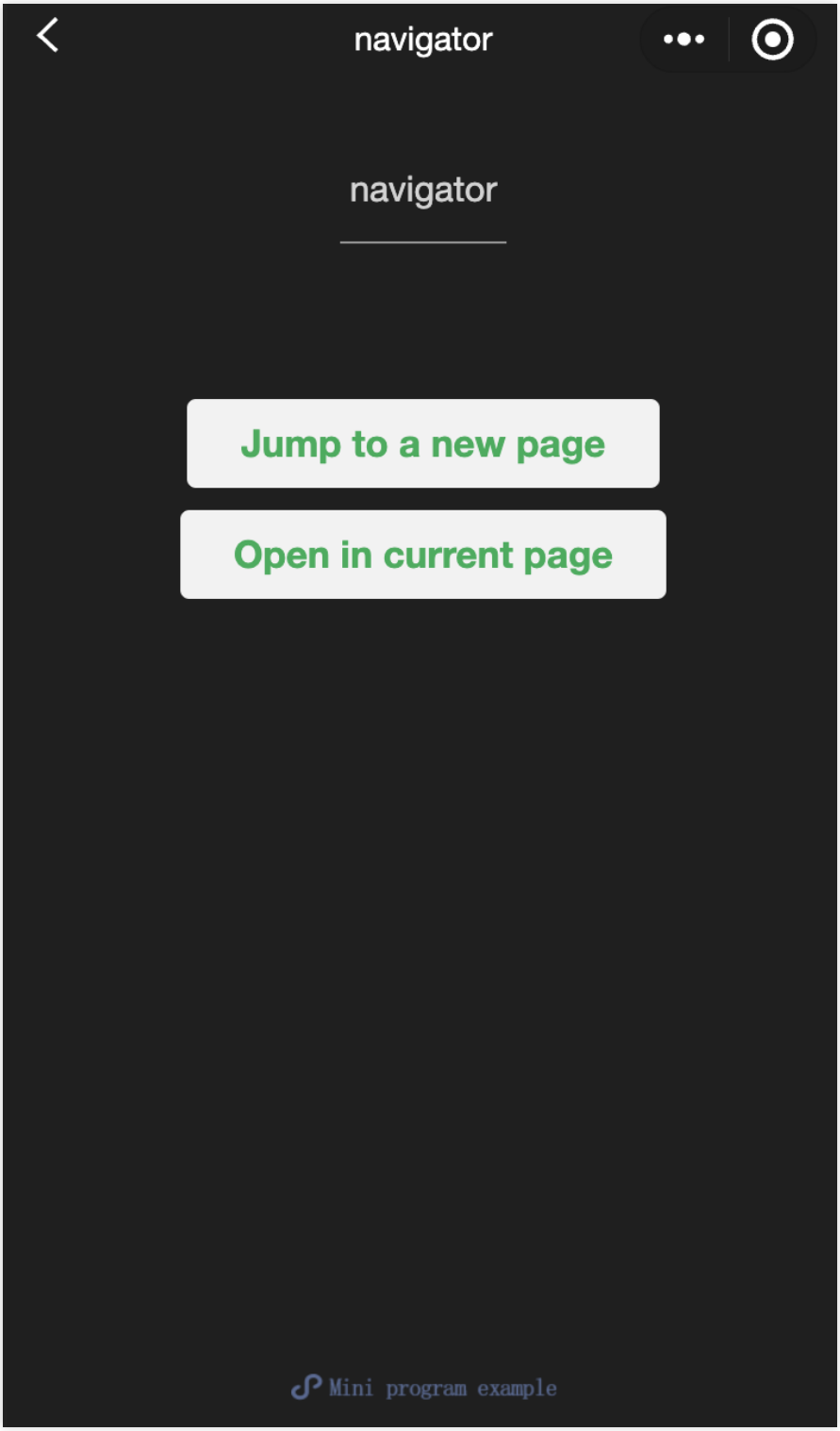
```
Page({
  onLoad: function(options) {
    this.setData({
      title: options.title
    })
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司



```
/** wxss **/
/** 修改默认的navigator点击态 **/
.navigator-hover {
  color: blue;
}
/** 自定义其他点击态样式类 **/
.other-navigator-hover {
  color: red;
}
```

版权所有：腾讯云计算（北京）有限责任公司



版权所有：腾讯云计算（北京）有限责任公司

表单组件

最近更新时间：2024-07-12 18:12:37

button

功能说明：按钮。

参数及说明：

属性	类型	合法值及说明	默认值	必填	说明
size	string	default：默认大小 mini：小尺寸	default	否	按钮的大小
type	string	primary：绿色 default：白色 warn：红色	default	否	按钮的样式类型
plain	boolean	-	false	否	按钮是否镂空，背景色
disabled	boolean	-	false	否	是否禁用
loading	boolean	-	false	否	名称前是否带 loading 图标
form-type	string	submit：提交表单 reset：重置表单	-	否	用于 form 组件，单击分别会触发 form 组件的 submit/reset 事件
open-type	string	share：触发用户转发 getPhoneNumber：手机号快速验证，向用户申请，并在用户同意后，快速填写和验证手机 getUserInfo：获取用户信息，可以从 bindgetuserinfo 回调中获取到用户信息 openSetting：打开授权设置页 feedback：打开“意见反馈”页面，用户	-	否	宿主app开放能力

		可提交反馈内容并上传日志			
hover-class	string	-	button-hover	否	指定按钮按下去的样式类。当 `hover-class="none";` 时，没有单击态效果
hover-start-time	number	-	20	否	按住后多久出现单击态，单位毫秒
hover-stay-time	number	-	70	否	手指松开后单击态保留时间，单位毫秒
lang	string	en：英文 zh_CN：简体中文 zh_TW：繁体中文	en	否	指定返回用户信息的语言
bindgetuserinfo	eventhandle	-	-	否	用户点击该按钮时，会返回获取到的用户信息，回调的 detail 数据与 wx.getUserInfo 返回的一致，open-type="getUserInfo"时有效
bindgetphonenumber	eventhandle	-	-	否	手机号快速验证回调，open-type=getPhoneNumber 时有效。**注意：在触发 bindgetphonenumber 回调后应立即隐藏手机号按钮组件，或置为 disabled 状态，避免用户重复授权手机号产生额外费用。**
bindopensetting	eventhandle	-	-	否	在打开授权设置页后回调，open-type=openSetting时有效

注意：

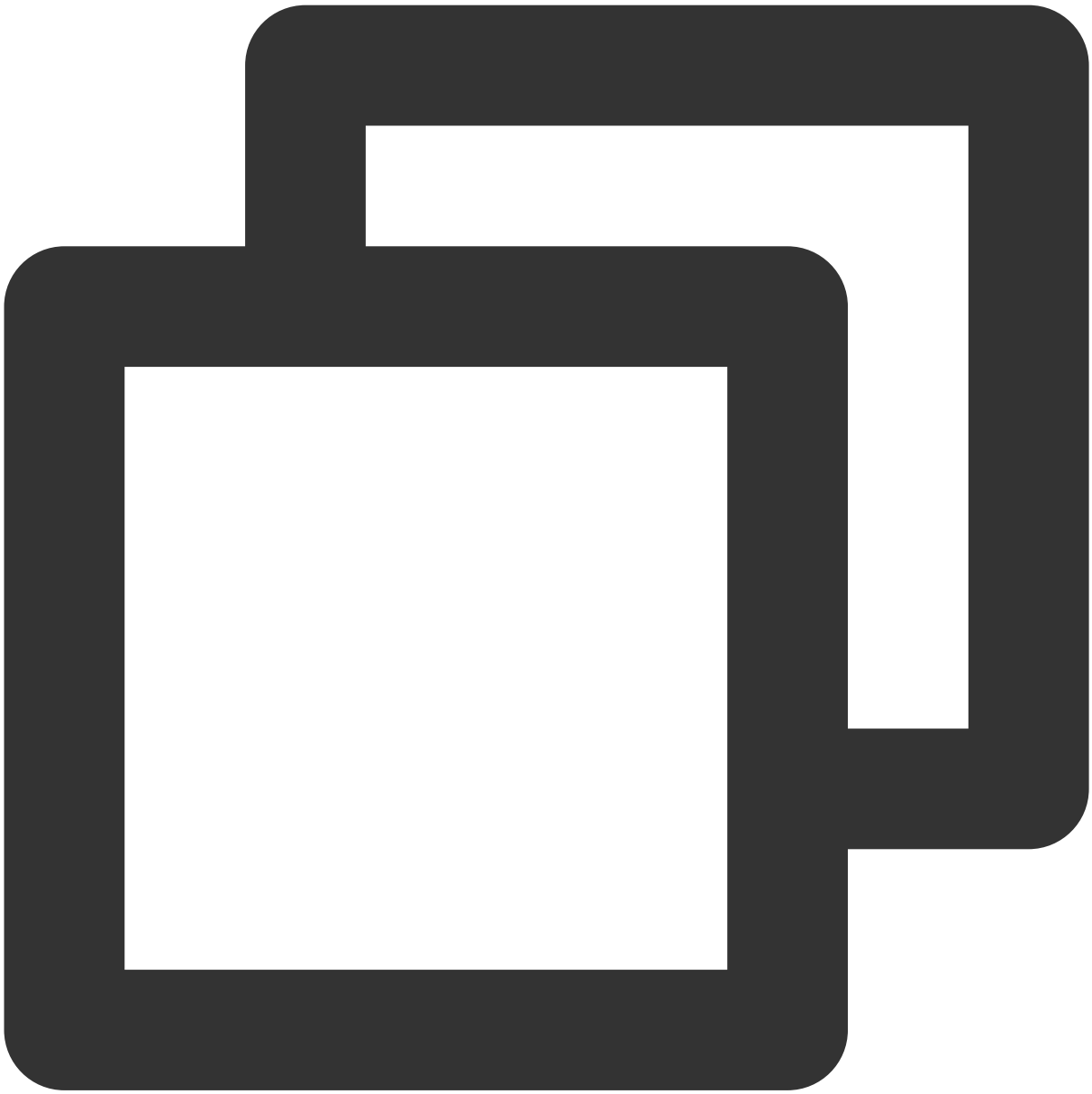
目前设置了 form-type 的 button 只会对当前组件中的 form 有效。因而，将 button 封装在自定义组件中，而 form 在自定义组件外，将会使这个 button 的 form-type 失效。

示例代码：

WXML

JAVASCRIPT

WXSS



```
<view class="page-body">
  <view class="btn-area" id="buttonContainer">
    <button type="primary">页面主操作 Normal</button>
    <button type="primary" loading="true">页面主操作 Loading</button>
    <button type="primary" disabled="true">页面主操作 Disabled</button>

    <button type="default">页面次要操作 Normal</button>
    <button type="default" disabled="true">页面次要操作 Disabled</button>
```

版权所有：腾讯云计算（北京）有限责任公司

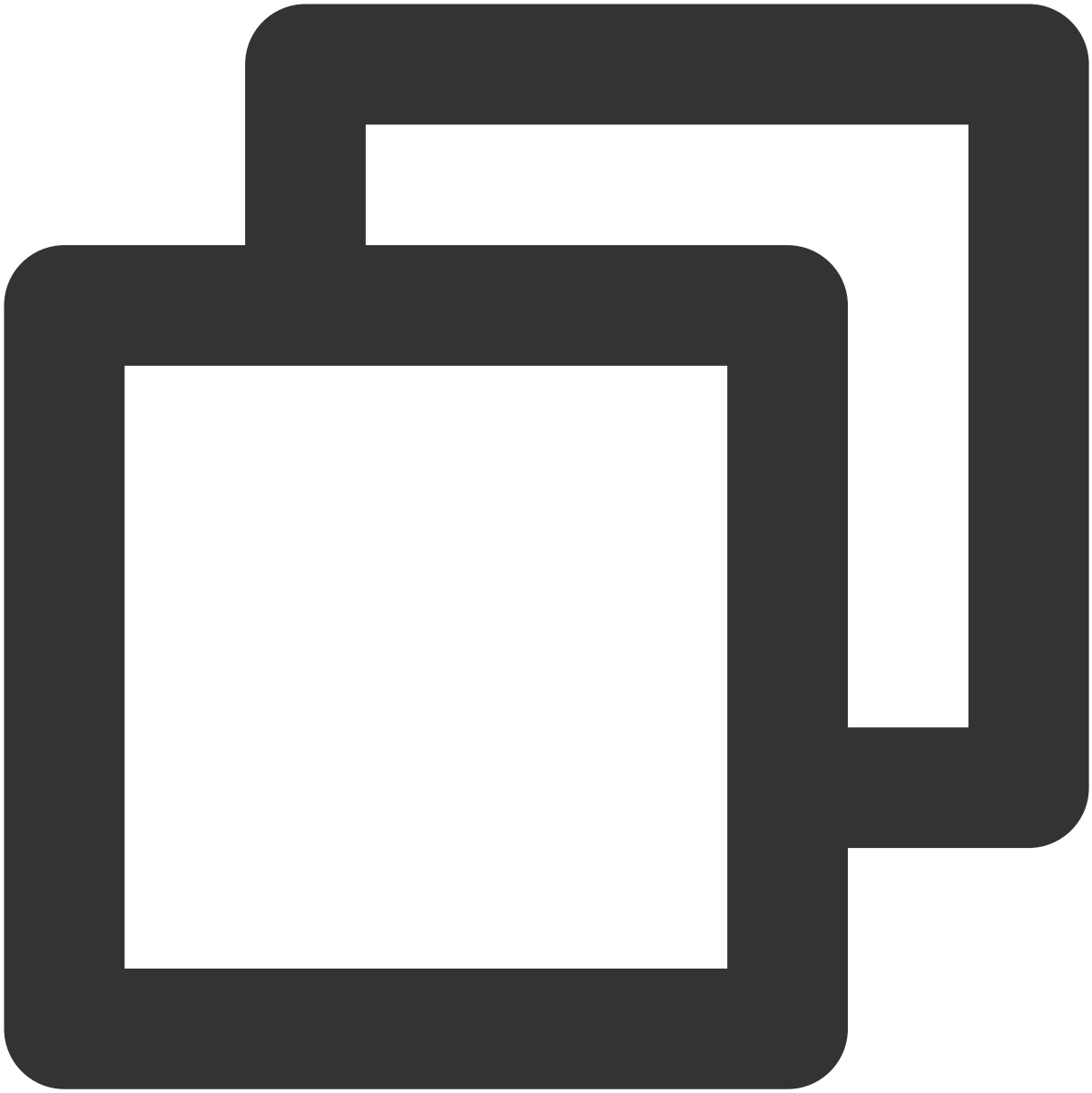
```
    <button type="warn">警告类操作 Normal</button>
    <button type="warn" disabled="true">警告类操作 Disabled</button>

    <view class="button-sp-area">
      <button type="primary" plain="true">按钮</button>
      <button type="primary" disabled="true" plain="true">不可点击的按钮</button>

      <button type="default" plain="true">按钮</button>
      <button type="default" disabled="true" plain="true">按钮</button>

      <button class="mini-btn" type="primary" size="mini">按钮</button>
      <button class="mini-btn" type="default" size="mini">按钮</button>
      <button class="mini-btn" type="warn" size="mini">按钮</button>
    </view>
  </view>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
const types = ['default', 'primary', 'warn']
const pageObject = {
  data: {
    defaultSize: 'default',
    primarySize: 'default',
    warnSize: 'default',
    disabled: false,
    plain: false,
    loading: false
  },
```

```
  onShareAppMessage() {
    return {
      title: 'button',
      path: 'page/component/pages/button/button'
    }
  },

  setDisabled() {
    this.setData({
      disabled: !this.data.disabled
    })
  },

  setPlain() {
    this.setData({
      plain: !this.data.plain
    })
  },

  setLoading() {
    this.setData({
      loading: !this.data.loading
    })
  },

  handleContact(e) {
    console.log(e.detail)
  },

  handleGetPhoneNumber(e) {
    console.log(e.detail)
  },

  handleGetUserInfo(e) {
    console.log(e.detail)
  },

  handleOpenSetting(e) {
    console.log(e.detail.authSetting)
  },

  handleGetUserInfo(e) {
    console.log(e.detail.userInfo)
  }
}

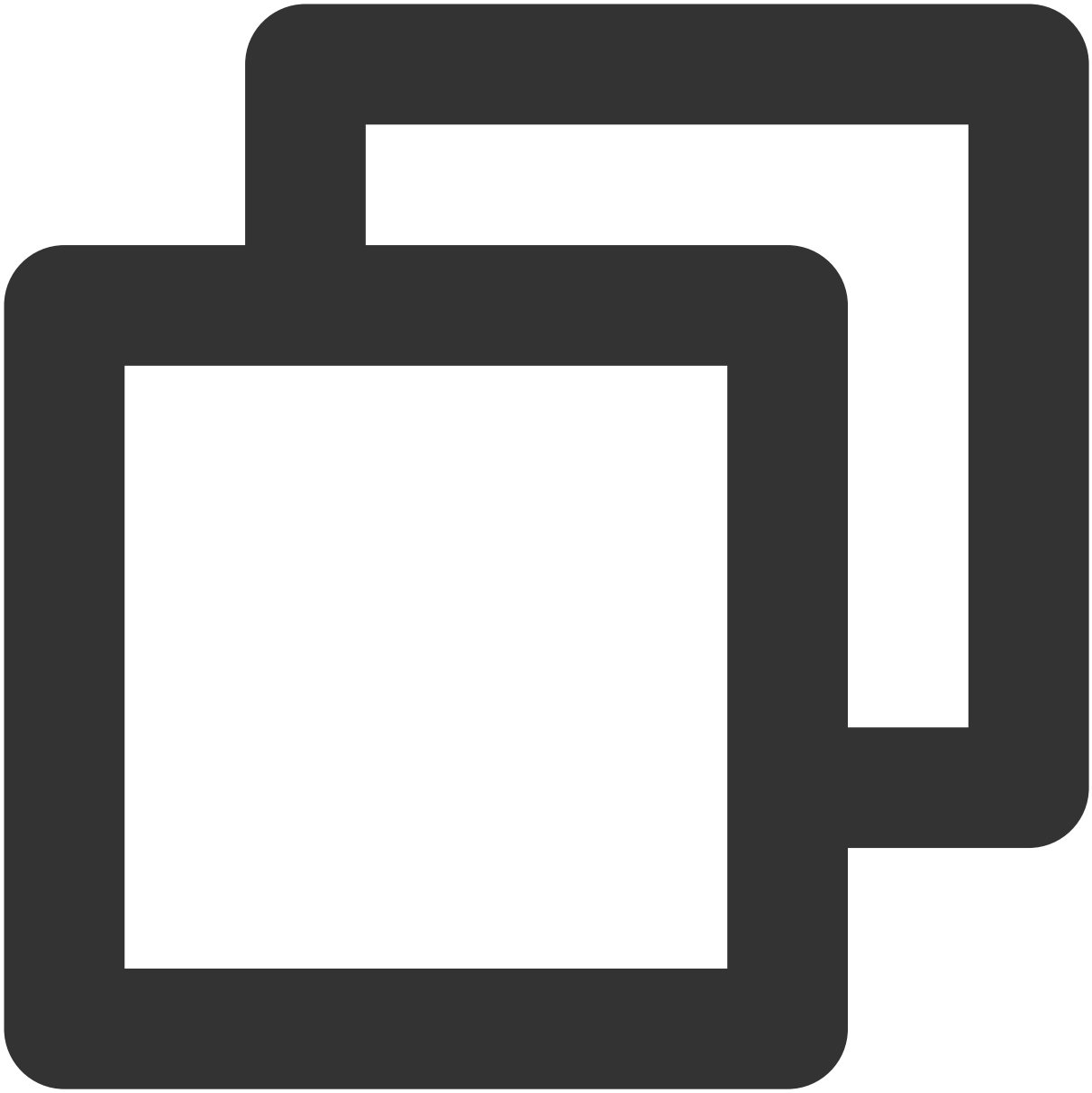
for (let i = 0; i < types.length; ++i) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
  (function (type) {
    pageObject[type] = function () {
      const key = type + 'Size'
      const changedData = {}
      changedData[key] =
        this.data[key] === 'default' ? 'mini' : 'default'
      this.setData(changedData)
    }
  }(types[i]))
}

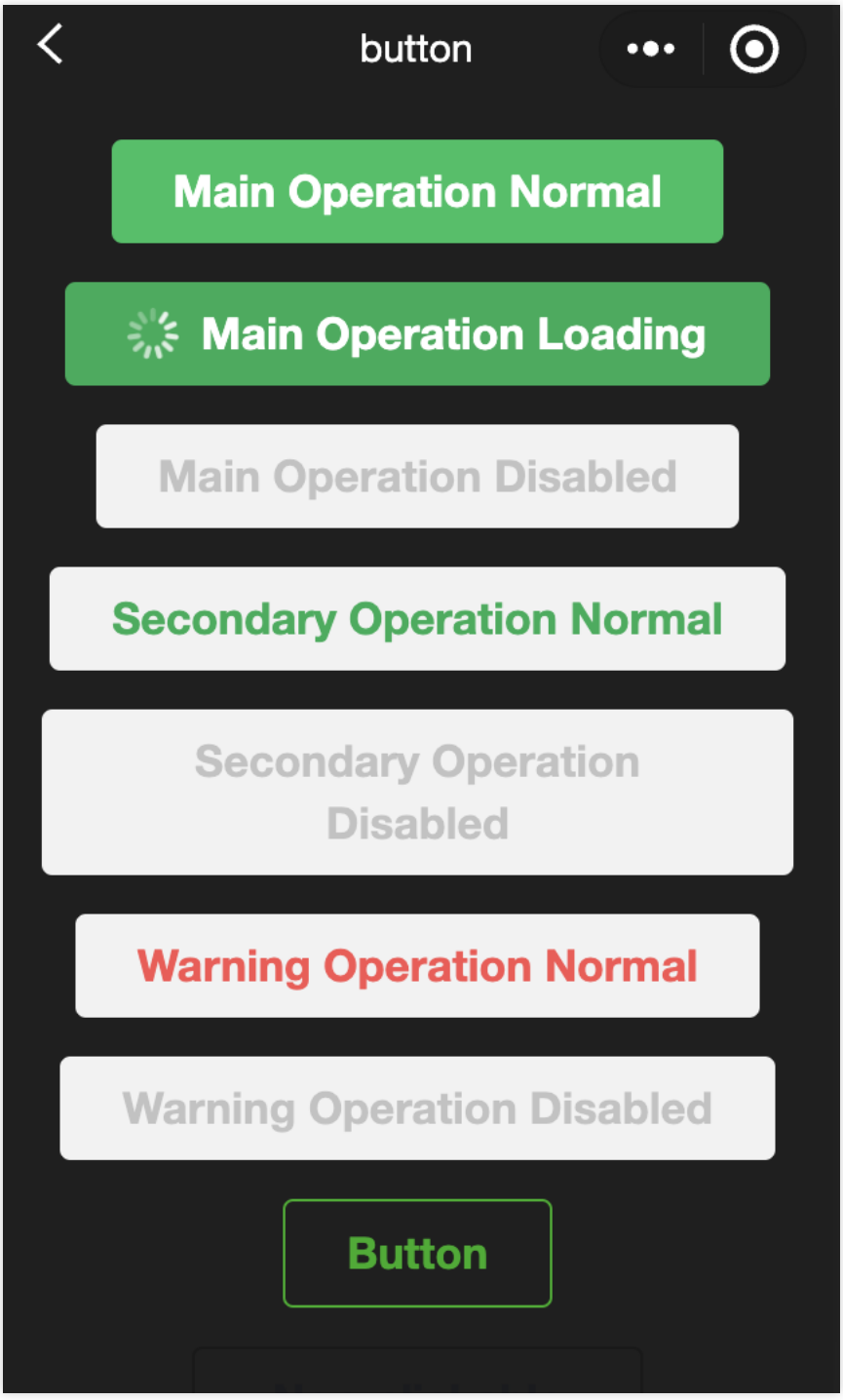
Page(pageObject)
```

版权所有：腾讯云计算（北京）有限责任公司



```
button{
  margin-top: 30rpx;
  margin-bottom: 30rpx;
}
.button-sp-area{
  margin: 0 auto;
  width: 60%;
}
.mini-btn{
  margin-right: 10rpx;
}
```

版权所有：腾讯云计算（北京）有限责任公司



checkbox

功能说明：多选项目。

参数及说明：

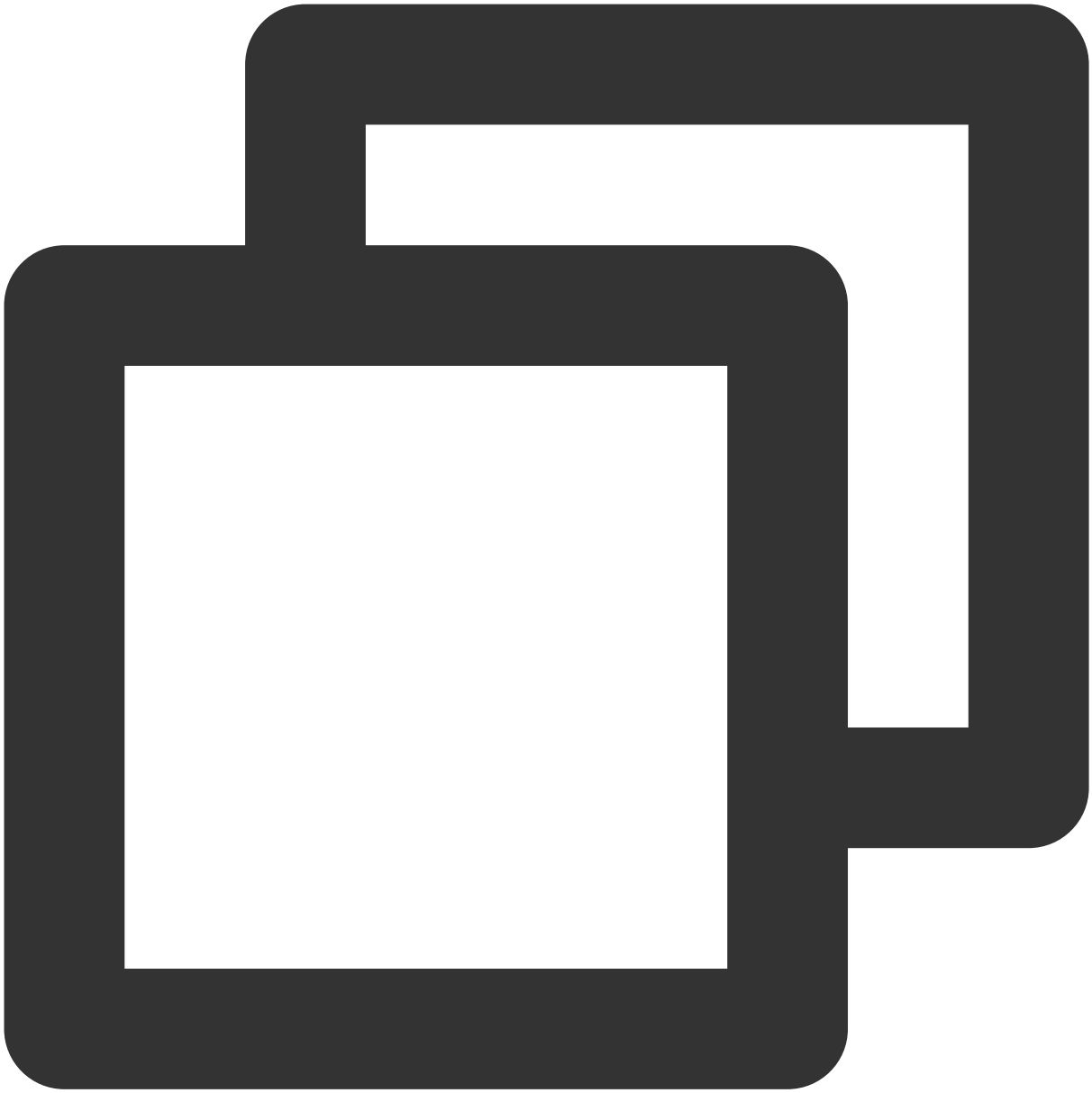
属性	类型	默认值	必填	说明
value	string	-	否	checkbox 标识，选中时触发 checkbox-group的

				change 事件，并携带 checkbox 的 value
disabled	boolean	false	否	是否禁用
checked	boolean	false	否	当前是否选中，可用来设置默认选中
color	string	#09BB07	否	checkbox 的颜色，同 css 的 color

示例代码：

WXML

JAVASCRIPT

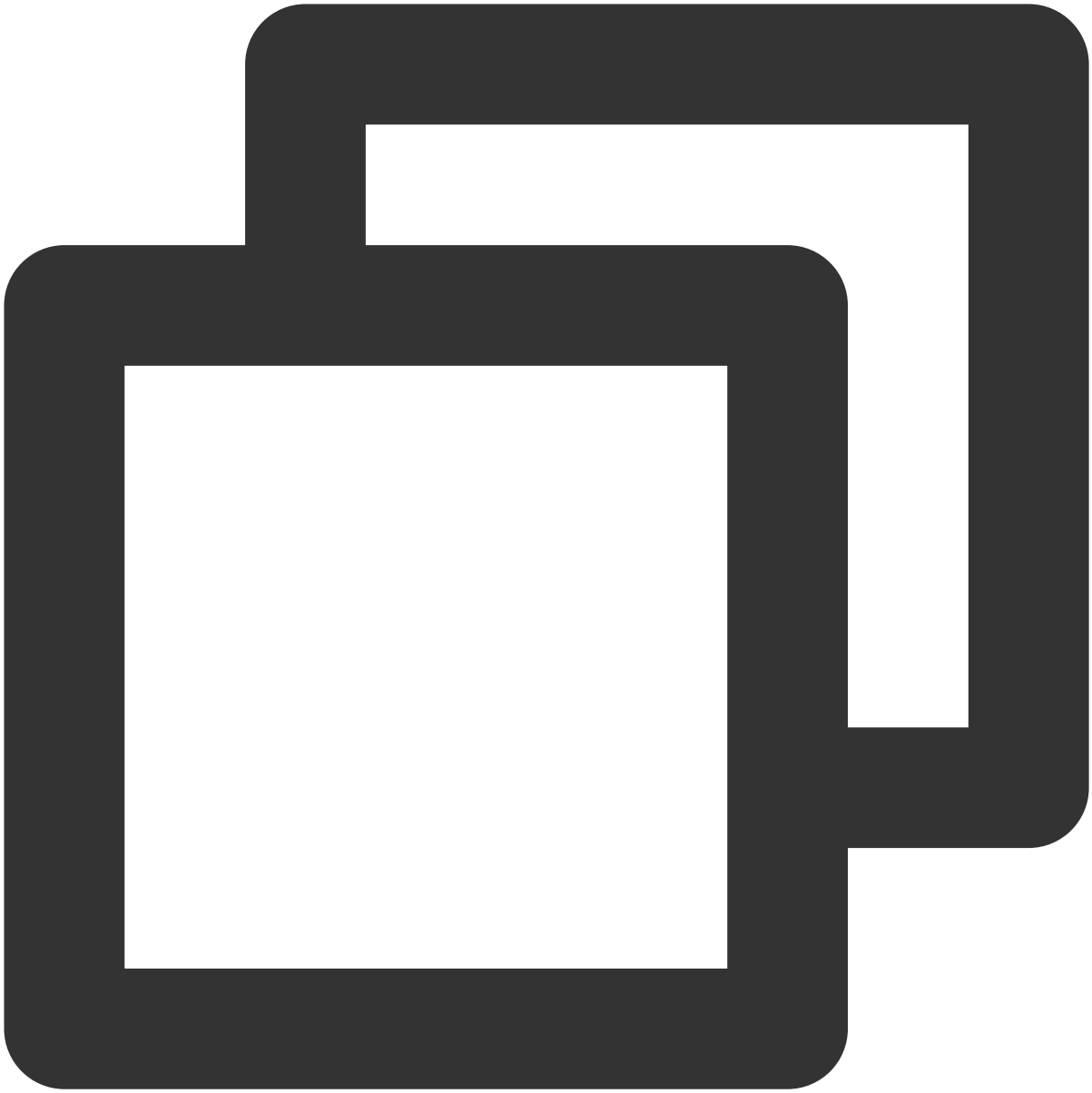


```
<view class="container">
  <view class="page-body">
    <view class="page-section page-section-gap">
      <view class="page-section-title">默认样式</view>
      <label class="checkbox">
        <checkbox value="cb" checked="true"/>选中
      </label>
      <label class="checkbox">
        <checkbox value="cb" />未选中
      </label>
    </view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
    <view class="page-section">
      <view class="page-section-title">推荐展示样式</view>
      <view class="weui-cells weui-cells_after-title">
        <checkbox-group bindchange="checkboxChange">
          <label class="weui-cell weui-check__label" wx:for="{{items}}" wx:key="{{i
            <view class="weui-cell__hd">
              <checkbox value="{{item.value}}" checked="{{item.checked}}"/>
            </view>
            <view class="weui-cell__bd">{{item.name}}</view>
          </label>
        </checkbox-group>
      </view>
    </view>
  </view>

</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'checkbox',
      path: 'page/component/pages/checkbox/checkbox'
    }
  },

  data: {
    items: [
      {value: 'USA', name: '美国'},
```

版权所有：腾讯云计算（北京）有限责任公司

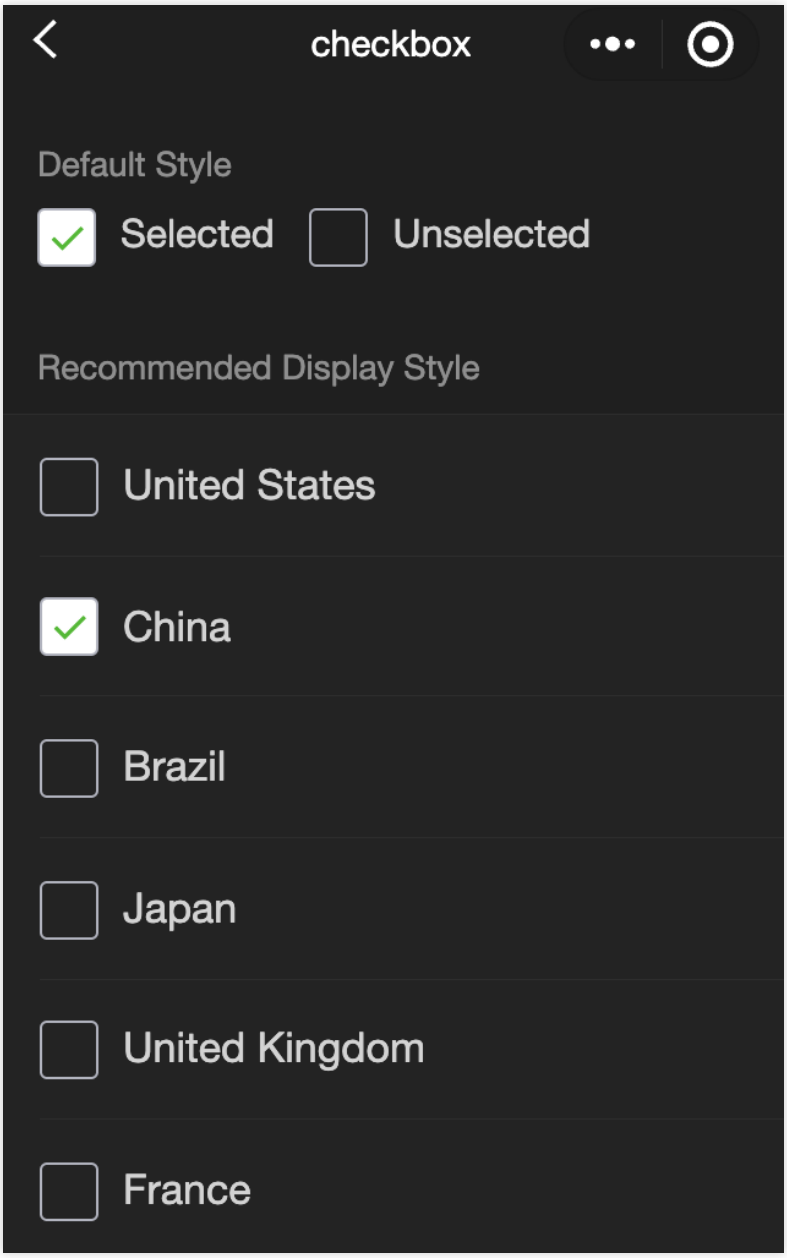
```
      {value: 'CHN', name: '中国', checked: 'true'},
      {value: 'BRA', name: '巴西'},
      {value: 'JPN', name: '日本'},
      {value: 'ENG', name: '英国'},
      {value: 'FRA', name: '法国'}
    ]
  },

  checkboxChange(e) {
    console.log('checkbox发生change事件，携带value值为：', e.detail.value)

    const items = this.data.items
    const values = e.detail.value
    for (let i = 0, lenI = items.length; i < lenI; ++i) {
      items[i].checked = false

      for (let j = 0, lenJ = values.length; j < lenJ; ++j) {
        if (items[i].value === values[j]) {
          items[i].checked = true
          break
        }
      }
    }

    this.setData({
      items
    })
  }
})
```



checkbox-group

功能描述：多项选择器，内部由多个 checkbox组成。

参数及说明：

属性	类型	默认值	必填	说明
bindchange	eventhandle	-	否	checkbox-group 中选中项发生改变时触发 change 事件，detail = {value:[选中的 checkbox 的 value 的数组]}

版权所有：腾讯云计算（北京）有限责任公司

editor

功能说明：富文本编辑器，可以对图片、文字进行编辑。
编辑器导出内容支持带标签的 `html` 和纯文本的 `text`，编辑器内部采用 `delta` 格式进行存储。
通过 `setContents` 接口设置内容时，解析插入的 `html` 可能会由于一些非法标签导致解析错误，建议开发者在小程序内使用时通过 `delta` 进行插入。
富文本组件内部引入了一些基本的样式使得内容可以正确的展示，开发时可以进行覆盖。
图片控件仅初始化时设置有效。
相关 api：EditorContext
参数及说明：

属性	类型	默认值	必填	说明
read-only	boolean	false	否	设置编辑器为只读
placeholder	string	-	否	提示信息
show-img-size	boolean	false	否	点击图片时显示图片大小控件
show-img-toolbar	boolean	false	否	点击图片时显示工具栏控件
show-img-resize	boolean	false	否	点击图片时显示修改尺寸控件
bindready	eventhandle	-	否	编辑器初始化完成时触发
bindfocus	eventhandle	-	否	编辑器聚焦时触发，event.detail = {html, text, delta}
bindblur	eventhandle	-	否	编辑器失去焦点时触发，detail = {html, text, delta}
bindinput	eventhandle	-	否	编辑器内容改变时触发，detail = {html, text, delta}
bindstatuschange	eventhandle	-	否	通过 Context 方法改变编辑器内样式时触发，返回选区已设置的样式

编辑器内支持部分 `HTML` 标签和内联样式，不支持 `class` 和 `id`
支持的标签
不满足的标签会被忽略，例如 `<div>` 会被转换为 `<p>` 储存。

类型	节点
行内元素	`<span>` `<strong>` `<b>` `<ins>` `<em>` `<i>` `<u>` `<a>` `<del>` `<s>` `<sub>` `<sup>` `<img>`

版权所有：腾讯云计算（北京）有限责任公司

块级元素	`<p>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<ol>` `<ul>` `<li>`

支持的内联样式

内联样式仅能设置在行内元素或块级元素上，不能同时设置。例如 `font-size` 归类为行内元素属性，在 `p` 标签上设置是无效的。

类型	样式
块级样式	`text-align` 、 `direction` 、 `margin` 、 `margin-top` 、 `margin-left` 、 `margin-right` 、 `margin-bottom` `padding` 、 `padding-top` 、 `padding-left` 、 `padding-right` 、 `padding-bottom` 、 `line-height` 、 `text-indent`
行内样式	`font` 、 `font-size` 、 `font-style` 、 `font-variant` 、 `font-weight` 、 `font-family` 、 `letter-spacing` 、 `text-decoration` 、 `color` 、 `background-color`

注意：

插入的 html 中事件绑定会被移除。

粘贴时仅纯文本内容会被拷贝进编辑器。

插入 html 到编辑器内时，编辑器会删除一些不必要的标签，以保证内容的统一。例如 xxx 会改写为 xxx 。

编辑器聚焦时页面会被上推，系统行为以保证编辑区可见。

form

功能说明：表单，将组件内的用户输入的 slider、picker，checkbox，radio，switch，input 提交。

当单击 form 表单中 form-type 为 submit 的 button 组件时，会将表单组件中的 value 值进行提交，需要在表单组件中加上 name 来作为 key。

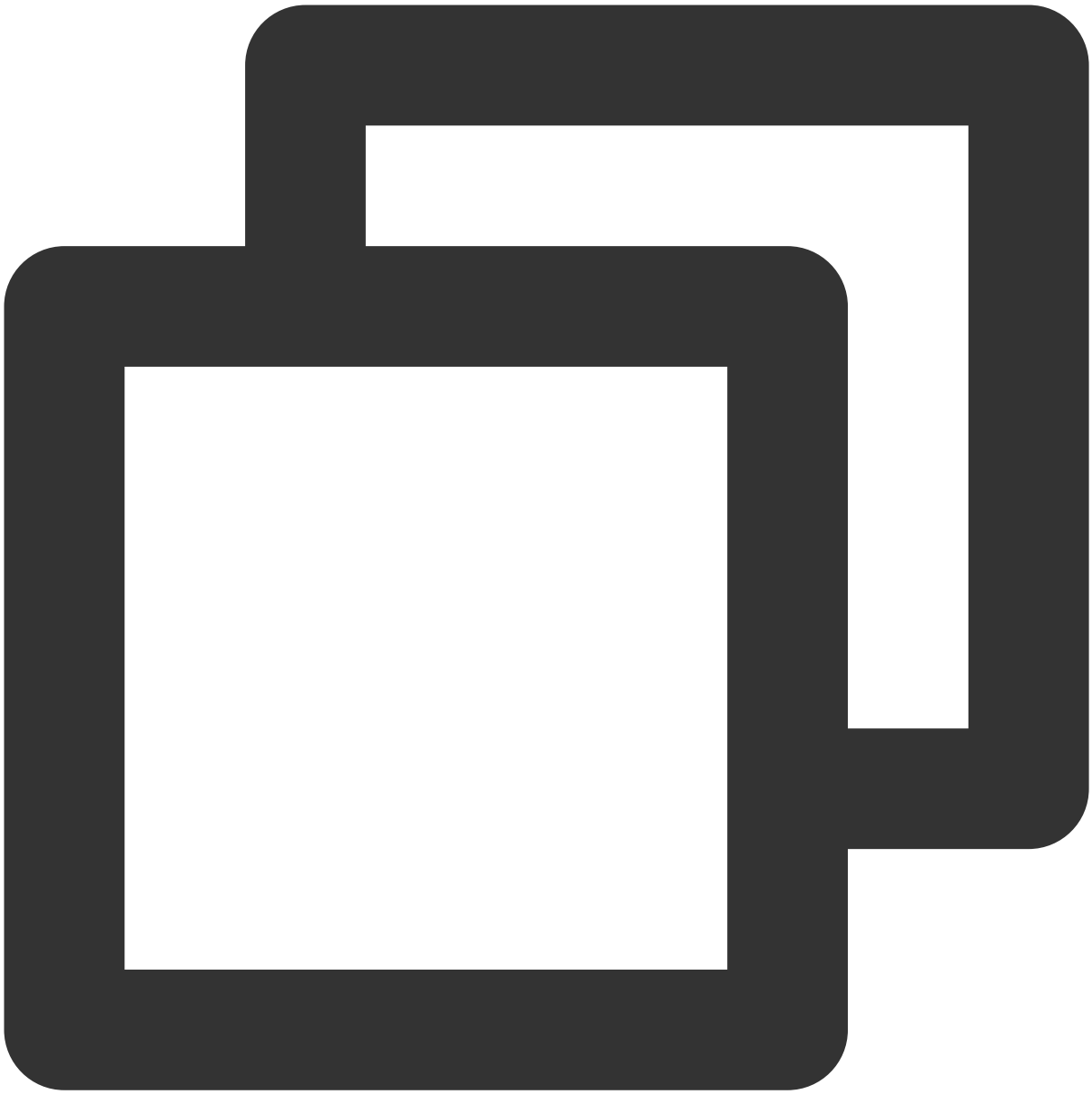
参数及说明：

属性	类型	默认值	必填	说明
bindsubmit	eventhandle	-	否	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'} , formId: ''}
bindreset	eventhandle	-	否	表单重置时会触发 reset 事件

示例代码：

WXML

JAVASCRIPT



```
<view class="container">
  <view class="page-body">
    <form catchsubmit="formSubmit" catchreset="formReset">
      <view class="page-section page-section-gap">
        <view class="page-section-title">switch</view>
        <switch name="switch"/>
      </view>

      <view class="page-section page-section-gap">
```

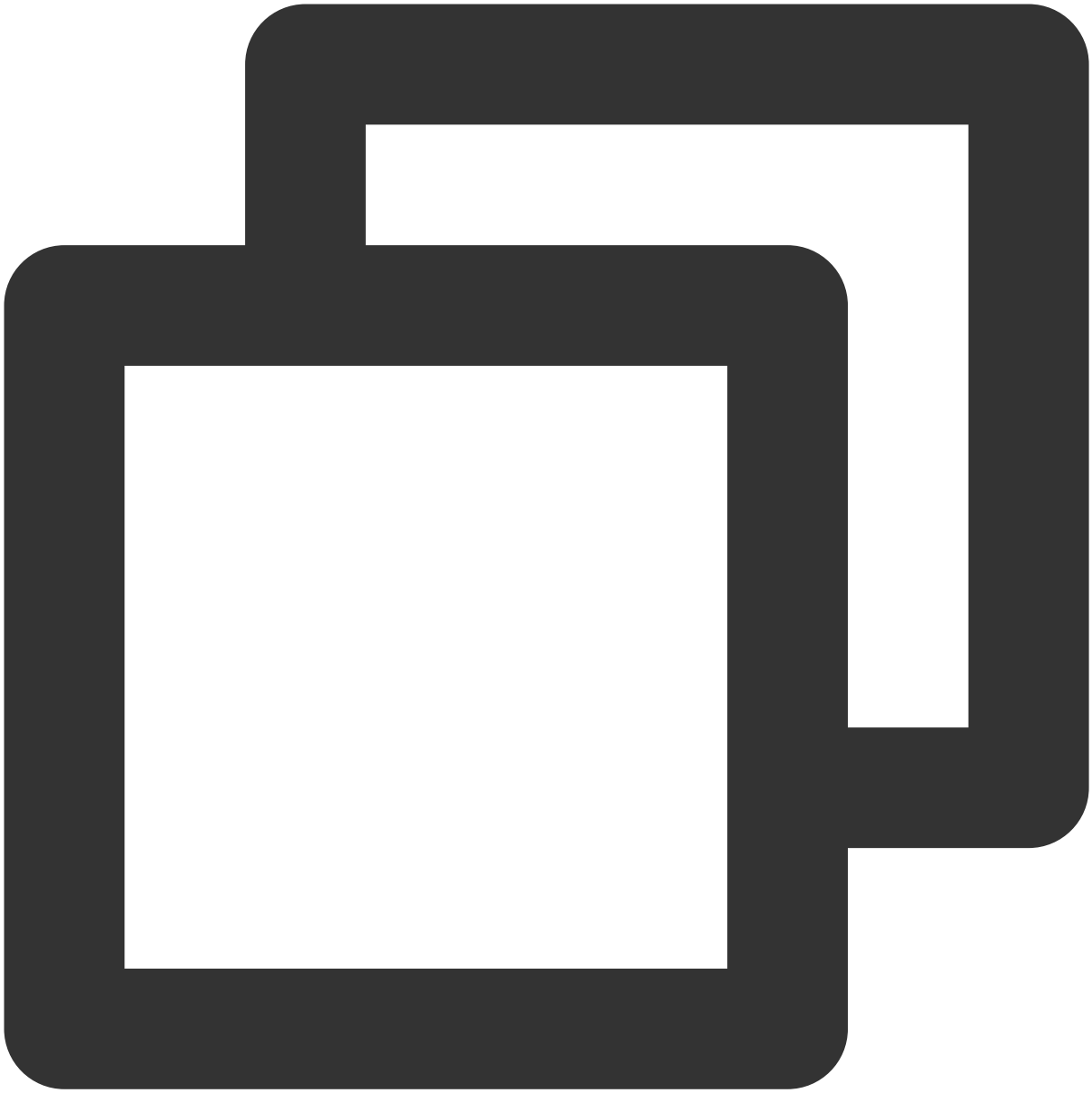
```
        <view class="page-section-title">radio</view>
        <radio-group name="radio">
          <label><radio value="radio1"/>选项一</label>
          <label><radio value="radio2"/>选项二</label>
        </radio-group>
      </view>

      <view class="page-section page-section-gap">
        <view class="page-section-title">checkbox</view>
        <checkbox-group name="checkbox">
          <label><checkbox value="checkbox1"/>选项一</label>
          <label><checkbox value="checkbox2"/>选项二</label>
        </checkbox-group>
      </view>

      <view class="page-section page-section-gap">
        <view class="page-section-title">slider</view>
        <slider value="50" name="slider" show-value ></slider>
      </view>

      <view class="page-section">
        <view class="page-section-title">input</view>
        <view class="weui-cells weui-cells_after-title">
          <view class="weui-cell weui-cell_input">
            <view class="weui-cell__bd" style="margin: 30rpx 0" >
              <input class="weui-input" name="input" placeholder="这是一个输入框" />
            </view>
          </view>
        </view>
      </view>

      <view class="btn-area">
        <button style="margin: 30rpx 0" type="primary" formType="submit">Submit</bu
        <button style="margin: 30rpx 0" formType="reset">Reset</button>
      </view>
    </form>
  </view>
</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'form',
      path: 'page/component/pages/form/form'
    }
  },

  data: {
    pickerHidden: true,
    chosen: ''
```

```
  },

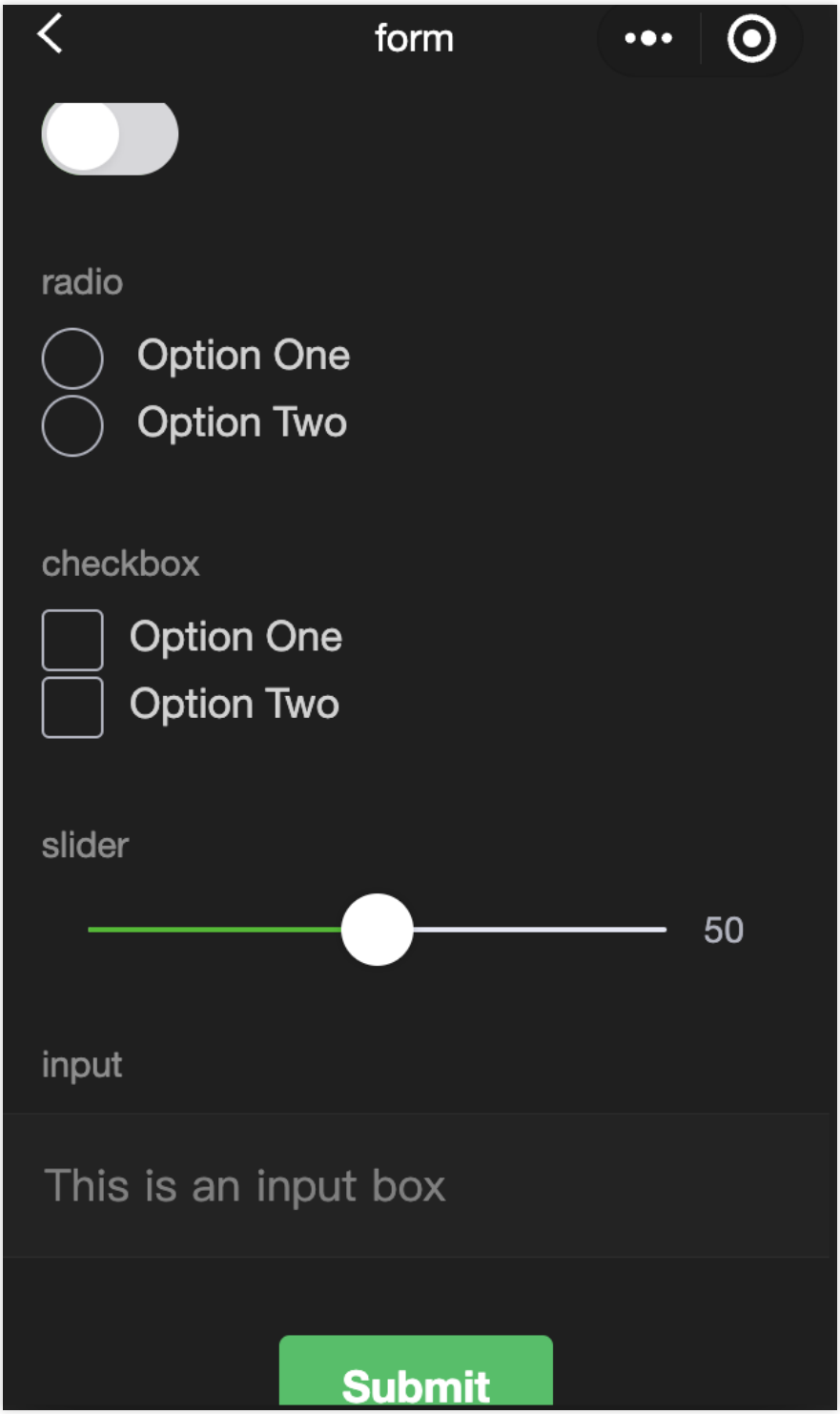
  pickerConfirm(e) {
    this.setData({
      pickerHidden: true
    })
    this.setData({
      chosen: e.detail.value
    })
  },

  pickerCancel() {
    this.setData({
      pickerHidden: true
    })
  },

  pickerShow() {
    this.setData({
      pickerHidden: false
    })
  },

  formSubmit(e) {
    console.log('form发生了submit事件，携带数据为：', e.detail.value)
  },

  formReset(e) {
    console.log('form发生了reset事件，携带数据为：', e.detail.value)
    this.setData({
      chosen: ''
    })
  }
})
```



input

功能说明：输入框，该组件是 原生组件，使用时请注意相关限制。

参数及说明：

属性	类型	合法值及说明	默认值	必填	说明

value	string	-	-	是	输入框的初始内容
type	string	text：文本输入键盘 number：数字输入键盘 idcard：身份证输入键盘 digit：带小数点的数字键盘	text	否	input 的类型
password	boolean	-	false	否	是否是密码类型
placeholder	string	-		是	输入框为空时占位符
placeholder-style	string	-	-	是	指定 placeholder 的样式
placeholder-class	string	-	input-placeholder	否	指定 placeholder 的样式类
disabled	boolean	-	false	否	是否禁用
maxlength	number	-	140	否	最大输入长度，设置为 -1 的时候不限制最大长度
cursor-spacing	number	-	0	否	指定光标与键盘的距离，取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离
auto-focus	boolean	-	false	否	(即将废弃，请直接使用 focus)自动聚焦，拉起键盘
focus	boolean	-	false	否	获取焦点
confirm-type	string	send：右下角按钮为“发送” search：右下角按钮	done	否	设置键盘右下角按钮的文字，仅在type='text'时生效

版权所有：腾讯云计算（北京）有限责任公司

		为“搜索” next：右下角按钮为“下一个” go：右下角按钮为“前往” done：右下角按钮为“完成”			
confirm-hold	boolean	-	false	否	点击键盘右下角按钮时是否保持键盘不收起
cursor	number	-	-	是	指定focus时的光标位置
selection-start	number	-	-1	否	光标起始位置，自动聚集时有效，需与selection-end搭配使用
selection-end	number	-	-1	否	光标结束位置，自动聚集时有效，需与selection-start搭配使用
adjust-position	boolean	-	true	否	键盘弹起时，是否自动上推页面
hold-keyboard	boolean	-	false	否	focus时，点击页面的时候不收起键盘
bindinput	eventhandle	-	-	是	键盘输入时触发，event.detail = {value, cursor, keyCode}，keyCode 为键值，处理函数可以直接 return 一个字符串，将替换输入框的内容。
bindfocus	eventhandle	-	-	是	输入框聚焦时触发，event.detail = { value, height }，height 为键盘高度
bindblur	eventhandle	-	-	是	输入框失去焦点时触发，event.detail = { value, encryptedValue, encryptError }

版权所有：腾讯云计算（北京）有限责任公司

bindconfirm	eventhandle	-	-	是	单击完成按钮时触发，event.detail = { value }
bindkeyboardheightchange	eventhandle	-	-	是	键盘高度发生变化的时候触发此事件，event.detail = {height: height, duration: duration}

注意：

confirm-type的最终表现与手机输入法本身的实现有关，部分安卓系统输入法和第三方输入法可能不支持或不完全支持。

input 组件是一个原生组件，字体是系统字体，所以无法设置 font-family。

在 input 聚焦期间，避免使用 css 动画。

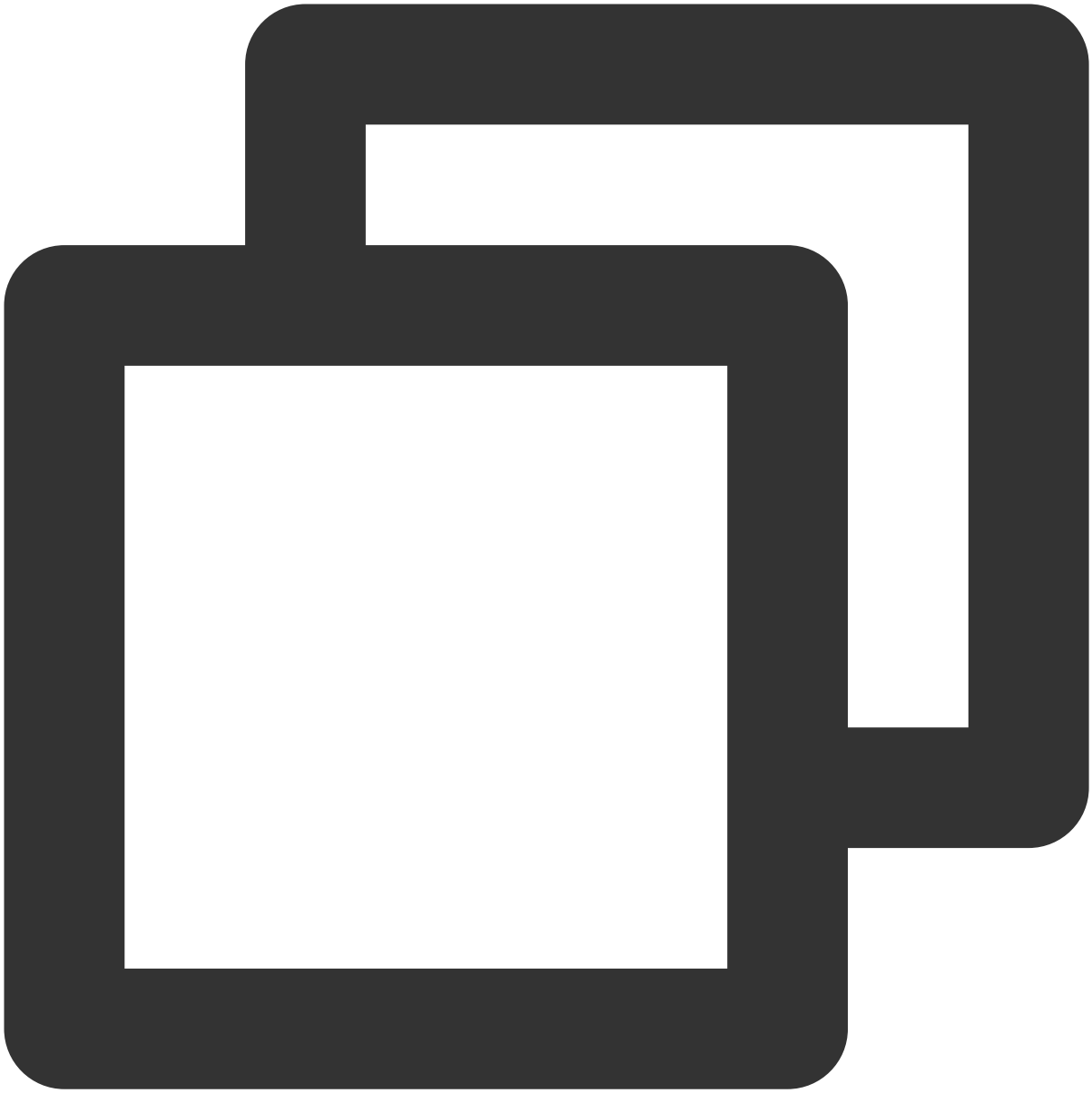
对于将 input 封装在自定义组件中、而 form 在自定义组件外的情况， form 将不能获得这个自定义组件中 input 的值。

键盘高度发生变化，keyboardheightchange事件可能会多次触发，开发者对于相同的height值应该忽略掉。

示例代码：

WXML

JAVASCRIPT



```
<view class="page-body">
  <view class="page-section">
    <view class="weui-cells__title">可以自动聚焦的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" auto-focus placeholder="将会获取焦点"/>
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制最大输入长度的input</view>
```

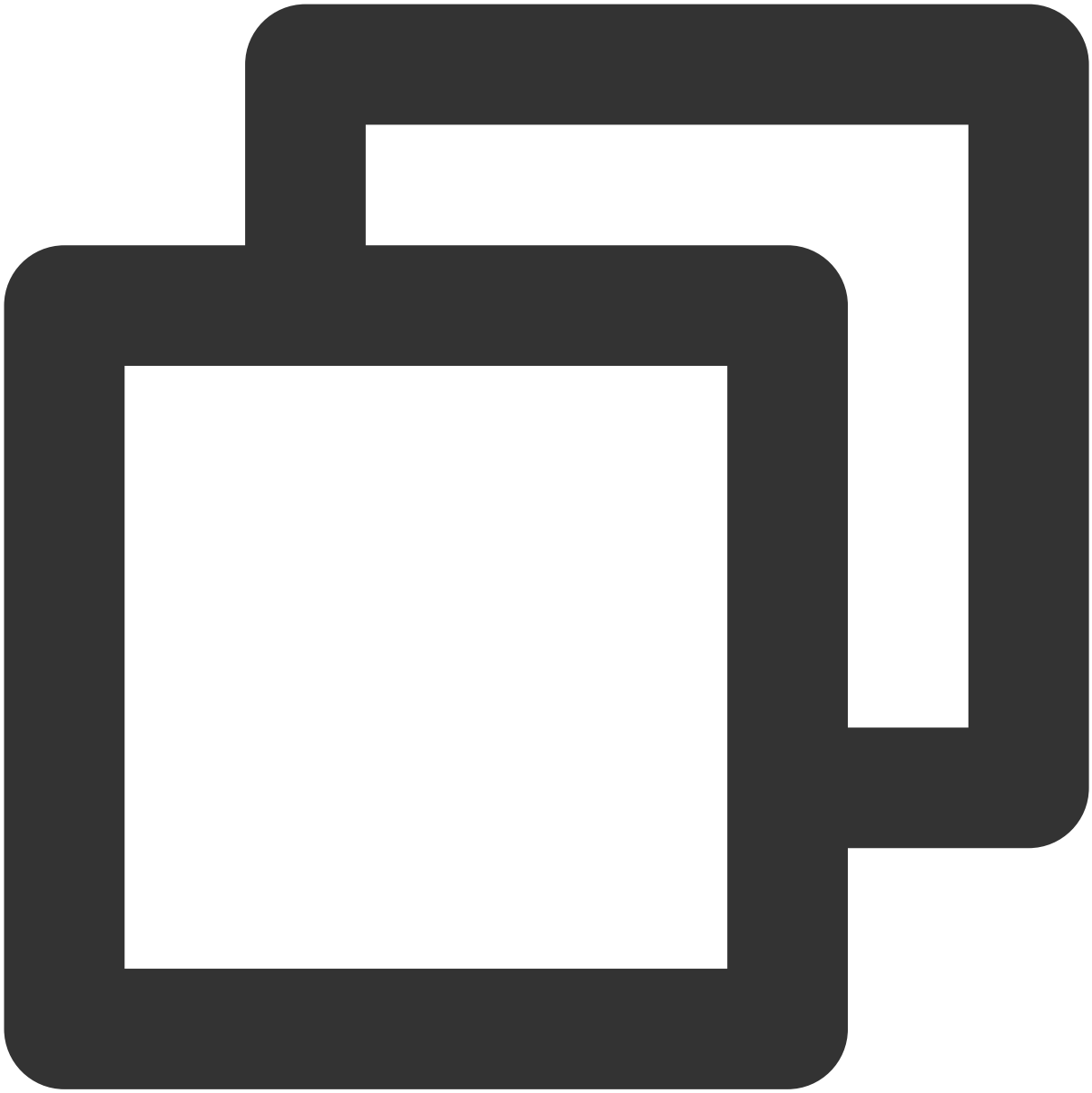
版权所有：腾讯云计算（北京）有限责任公司

```
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" maxlength="10" placeholder="最大输入长度为10" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">实时获取输入值：{{inputValue}}</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input"  maxlength="10" bindinput="bindKeyInput" placehol
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制输入的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input"  bindinput="bindReplaceInput" placeholder="连续的两
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制键盘的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input"  bindinput="bindHideKeyboard" placeholder="输入123
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">数字输入的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" type="number" placeholder="这是一个数字输入框" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">密码输入的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" password type="text" placeholder="这是一个密码输入框
      </view>
    </view>
  </view>
  <view class="page-section">
```

版权所有：腾讯云计算（北京）有限责任公司

```
    <view class="weui-cells__title">带小数点的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" type="digit" placeholder="带小数点的数字键盘"/>
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">身份证输入的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" type="idcard" placeholder="身份证输入键盘" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="weui-cells__title">控制占位符颜色的input</view>
    <view class="weui-cells weui-cells_after-title">
      <view class="weui-cell weui-cell_input">
        <input class="weui-input" placeholder-style="color:#F76260" placeholder="占
      </view>
    </view>
  </view>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    focus: false,
    inputValue: ''
  },
  bindKeyInput: function (e) {
    this.setData({
      inputValue: e.detail.value
    })
  },
  bindReplaceInput: function (e) {
```

```
    var value = e.detail.value
    var pos = e.detail.cursor
    var left
    if (pos !== -1) {
      // 光标在中间
      left = e.detail.value.slice(0, pos)
      // 计算光标的位置
      pos = left.replace(/11/g, '2').length
    }

    // 直接返回对象，可以对输入进行过滤处理，同时可以控制光标的位置
    return {
      value: value.replace(/11/g, '2'),
      cursor: pos
    }

    // 或者直接返回字符串,光标在最后边
    // return value.replace(/11/g,'2'),
  },
  bindHideKeyboard: function (e) {
    if (e.detail.value === '123') {
      // 收起键盘
      wx.hideKeyboard()
    }
  }
})
```

label

功能说明：用来改进表单组件的可用性。
使用 for 属性找到对应的 id，或者将控件放在该标签下，当单击时，就会触发对应的控件。 for 优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。目前可以绑定的控件有：button，checkbox，radio，switch，

input。

参数及说明：

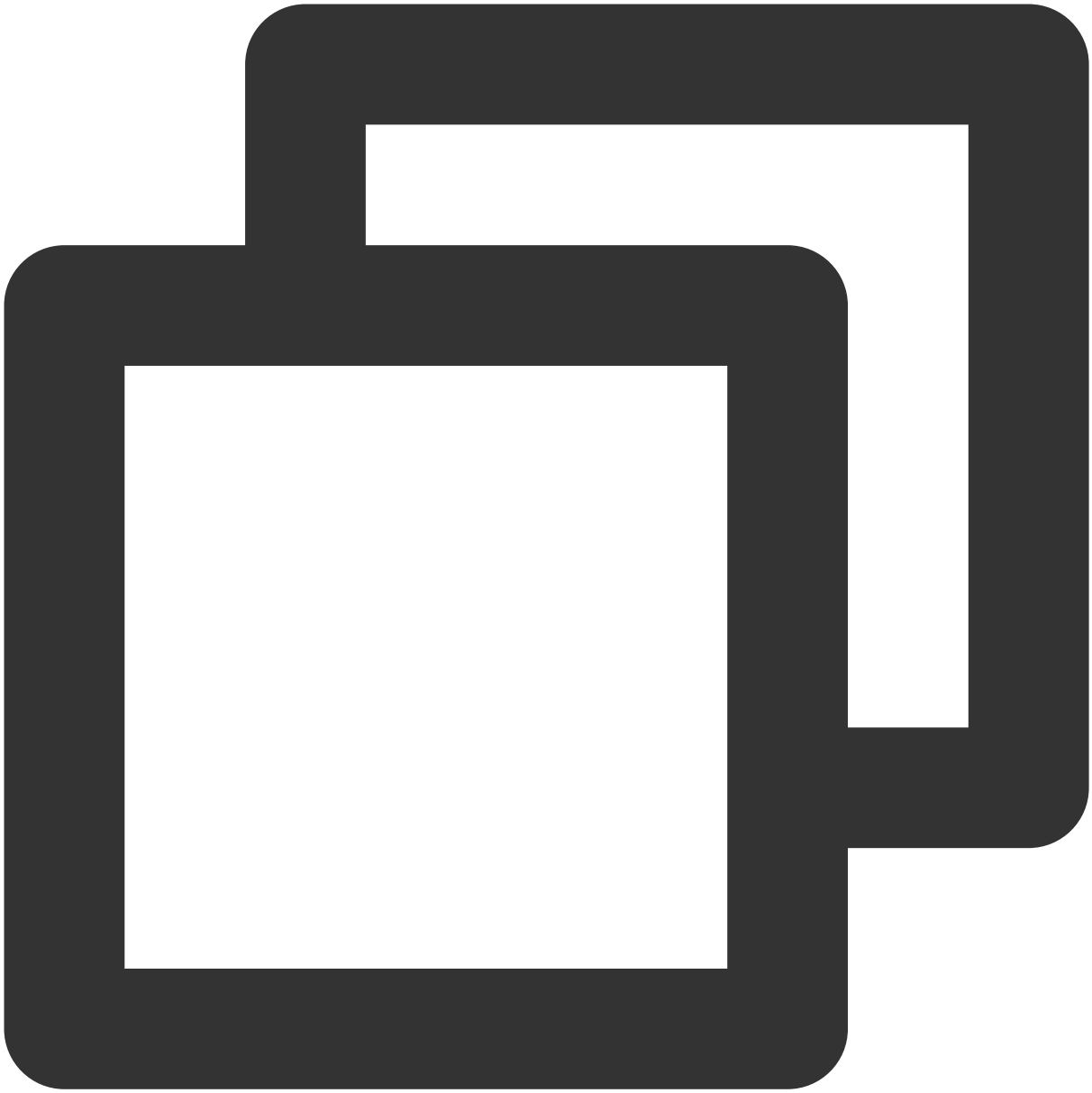
属性	类型	默认值	必填	说明
for	string	-	否	绑定控件的 id

示例代码：

WXML

JAVASCRIPT

WXSS

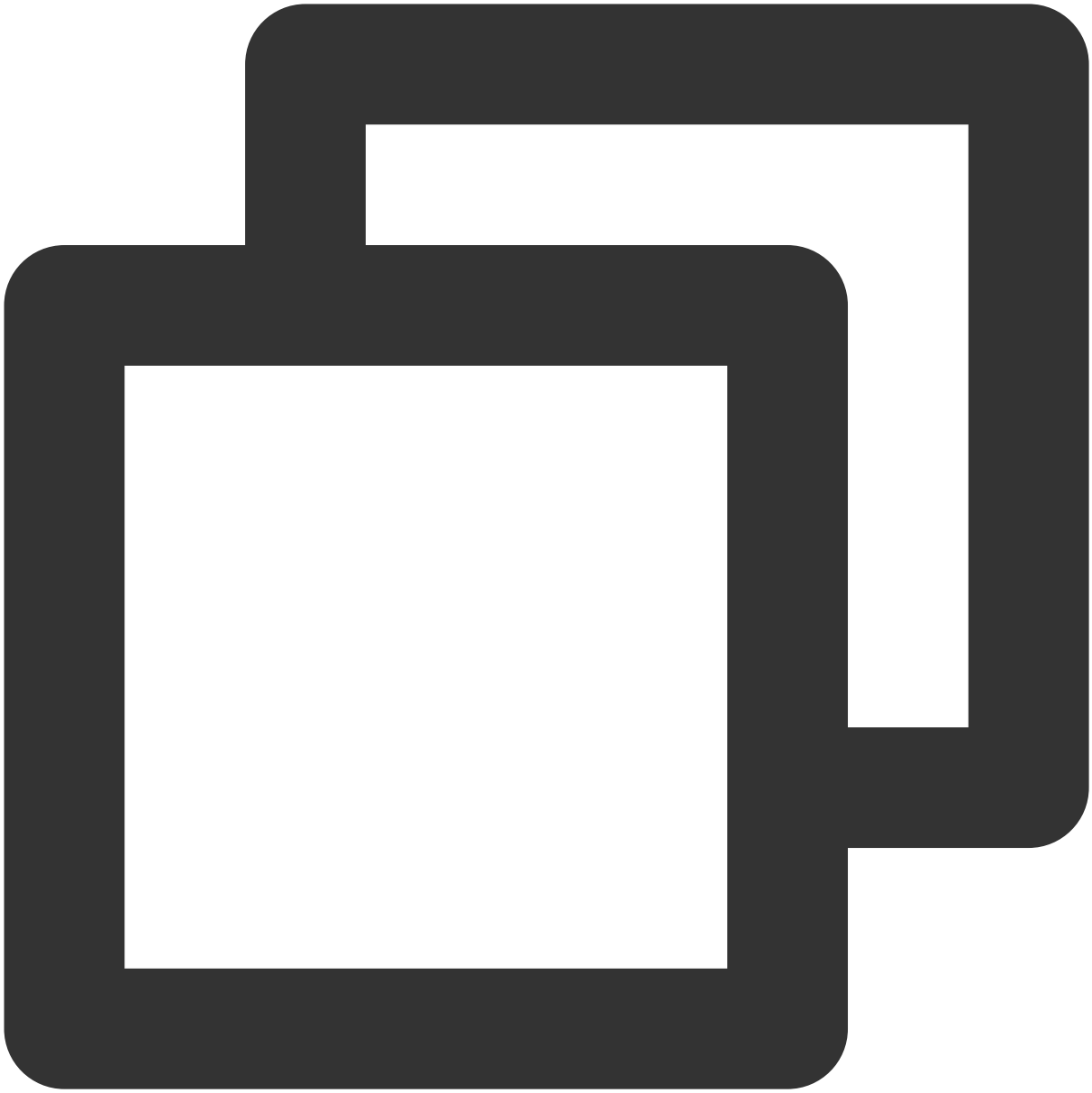


```
<view class="container">
  <view class="page-body">
    <view class="page-section page-section-gap">
      <view class="page-section-title">表单组件在label内</view>
      <checkbox-group class="group" bindchange="checkboxChange">
        <view class="label-1" wx:for="{{checkboxItems}}">
          <label>
            <checkbox value="{{item.name}}" checked="{{item.checked}}"></checkbox>
            <text class="label-1-text">{{item.value}}</text>
          </label>
        </view>
```

```
      </checkbox-group>
    </view>

    <view class="page-section page-section-gap">
      <view class="page-section-title">label用for标识表单组件</view>
      <radio-group class="group" bindchange="radioChange">
        <view class="label-2" wx:for="{{radioItems}}">
          <radio id="{{item.name}}" value="{{item.name}}" checked="{{item.checked}}
          <label class="label-2-text" for="{{item.name}}"><text>{{item.name}}</text
        </view>
      </radio-group>
    </view>

    <view class="page-section page-section-gap">
      <view class="page-section-title">label内有多个时选中第一个</view>
      <label class="label-3">
        <checkbox class="checkbox-3">选项一</checkbox>
        <checkbox class="checkbox-3">选项二</checkbox>
        <view class="label-3-text">点击该label下的文字默认选中第一个checkbox</view>
      </label>
    </view>
  </view>
</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'label',
      path: 'page/component/pages/label/label'
    }
  },

  data: {
    checkboxItems: [
      {name: 'USA', value: '美国'},
```

版权所有：腾讯云计算（北京）有限责任公司

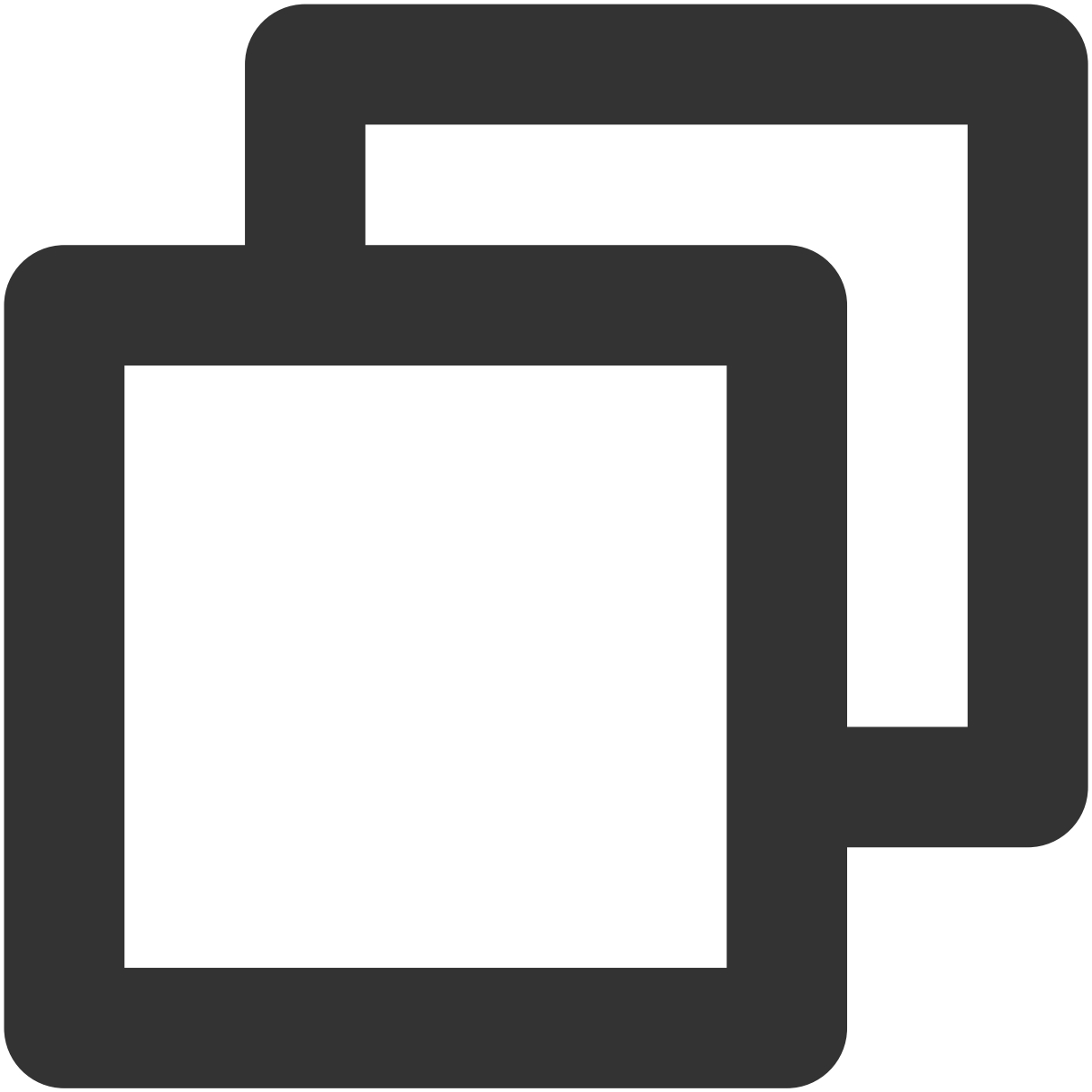
```
      {name: 'CHN', value: '中国', checked: 'true'}
    ],
    radioItems: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'}
    ],
    hidden: false
  },

  checkboxChange(e) {
    const checked = e.detail.value
    const changed = {}
    for (let i = 0; i < this.data.checkboxItems.length; i++) {
      if (checked.indexOf(this.data.checkboxItems[i].name) !== -1) {
        changed['checkboxItems[' + i + '].checked'] = true
      } else {
        changed['checkboxItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  },

  radioChange(e) {
    const checked = e.detail.value
    const changed = {}
    for (let i = 0; i < this.data.radioItems.length; i++) {
      if (checked.indexOf(this.data.radioItems[i].name) !== -1) {
        changed['radioItems[' + i + '].checked'] = true
      } else {
        changed['radioItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  },

  tapEvent() {
    console.log('按钮被点击')
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司



```
.label-1, .label-2{
  margin: 30rpx 0;
}
.label-3-text{
  color: #576B95;
  font-size: 28rpx;
}
.checkbox-3{
  display: block;
  margin: 30rpx 0;
}
```

版权所有：腾讯云计算（北京）有限责任公司

picker

功能描述：从底部弹起的滚动选择器。

参数及说明：

版权所有：腾讯云计算（北京）有限责任公司

属性	类型	合法值及说明	默认值	必填	说明
mode	string	selector：普通选择器 multiSelector：多列选择器 time：时间选择器 date：日期选择器 region：省市区选择器	selector	否	选择器类型
disabled	boolean	-	false	否	是否禁用
bindcancel	eventhandle	-	-	否	取消选择时触发

除了上述通用的属性，对于不同的 mode， `picker` 拥有不同的属性。

普通选择器：mode = selector

属性名	类型	默认值	说明
range	array/object array	[]	mode 为 selector 或 multiSelector 时，range 有效
range-key	string	-	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	number	0	表示选择了 range 中的第几个（下标从0开始）
bindchange	eventhandle	-	value 改变时触发 change 事件，event.detail = {value}

多列选择器：mode = multiSelector

属性名	类型	默认值	说明
range	array/object array	[]	mode 为 selector 或 multiSelector 时，range 有效
range-key	string	-	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	array	[]	表示选择了 range 中的第几个（下标从0开始）
bindchange	eventhandle	-	value 改变时触发 change 事件，event.detail = {value}
bindcolumnchange	eventhandle	-	列改变时触发

时间选择器：mode = time

版权所有：腾讯云计算（北京）有限责任公司

属性名	类型	默认值	说明
value	string	-	表示选中的时间，格式为"hh:mm"
start	string	-	表示有效时间范围的开始，字符串格式为"hh:mm"
end	string	-	表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	eventhandle	-	value 改变时触发 change 事件，event.detail = {value}

日期选择器：mode = date

属性名	类型	有效值及说明	默认值	说明
value	string	-	当天	表示选中的日期，格式为"YYYY-MM-DD"
start	string	-	-	表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"
end	string	-	-	表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"
fields	string	year：选择器粒度为年 month：选择器粒度为月份 day：选择器粒度为天	day	有效值 year,month,day，表示选择器的粒度
bindchange	eventhandle	-	-	value 改变时触发 change 事件，event.detail = {value}

省市区选择器：mode = region

属性名	类型	默认值	说明
value	Array	[]	表示选中的省市区，默认选中每一列的第一个值
custom-item	string	-	可为每一列的顶部添加一个自定义的项
bindchange	evenhandle	-	value 改变时触发 change 事件，event.detail = {value: value, code: code, postcode: postcode}，其中字段 code 是统计用区划代码，

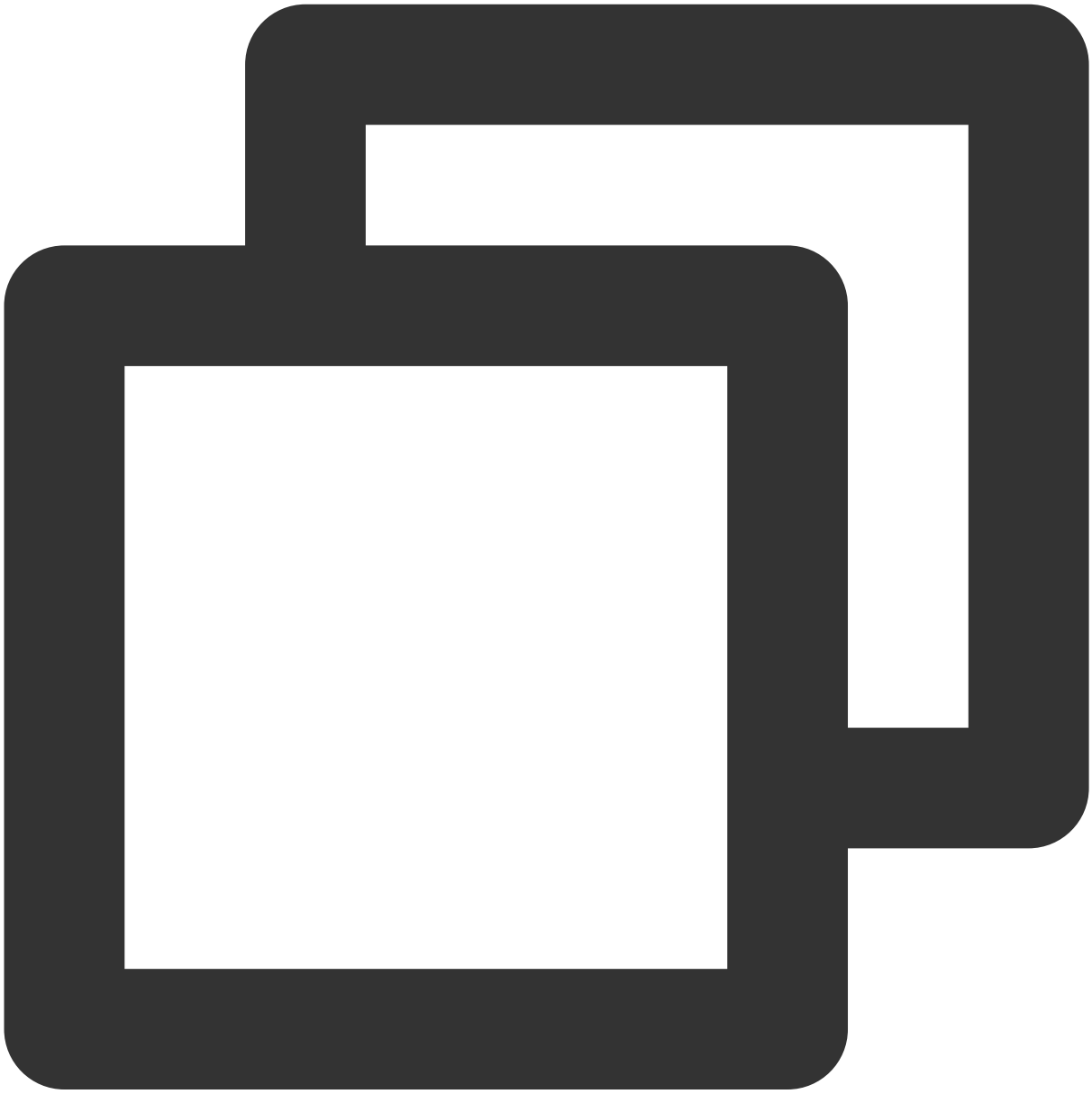
版权所有：腾讯云计算（北京）有限责任公司

			postcode 是邮政编码

示例代码：

WXML

JAVASCRIPT

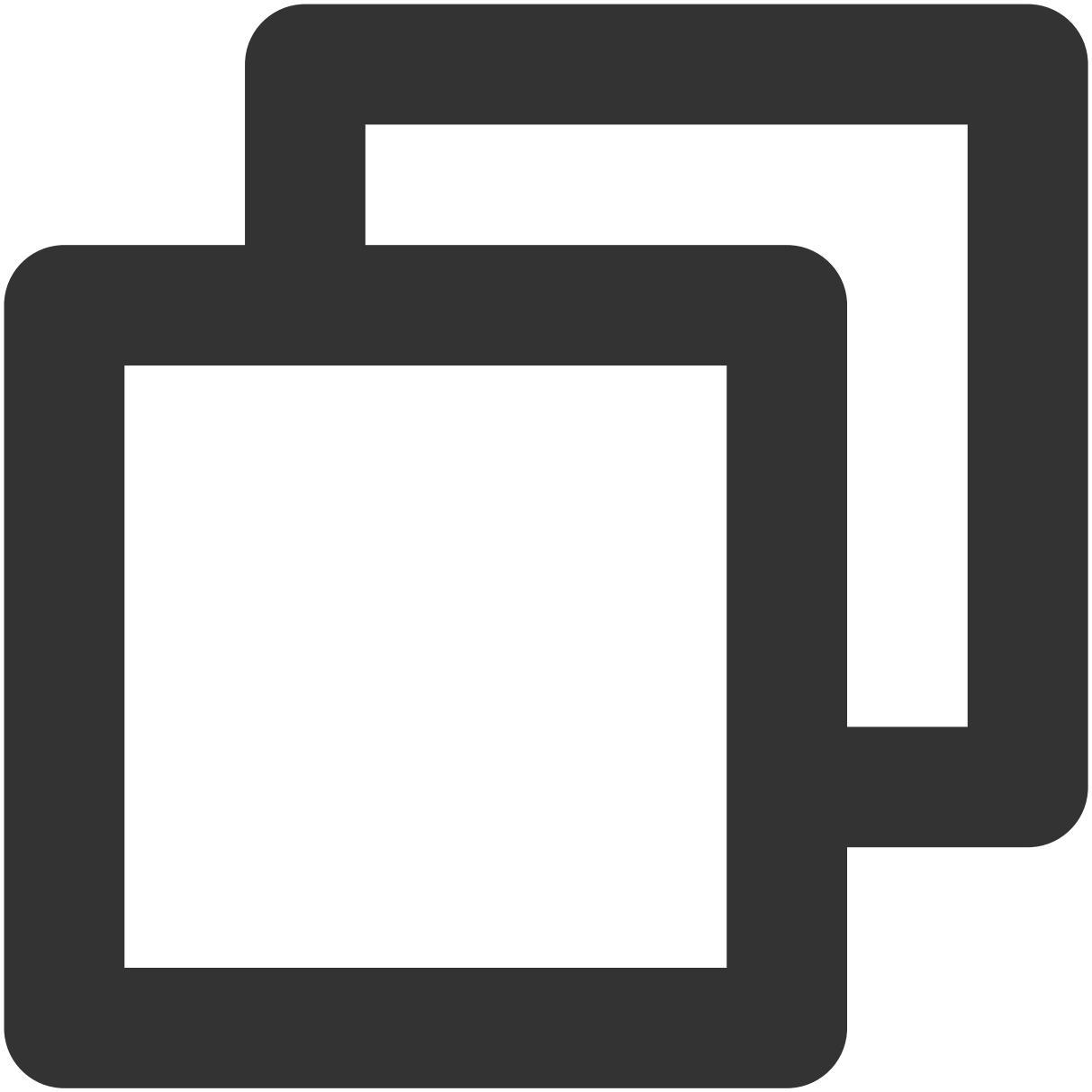


```
<view class="section">
  <view class="section__title">普通选择器</view>
  <picker bindchange="bindPickerChange" value="{{index}}" range="{{array}}">
    <view class="picker">
```

```
      当前选择：{{array[index]}}
    </view>
  </picker>
</view>
<view class="section">
  <view class="section__title">多列选择器</view>
  <picker mode="multiSelector" bindchange="bindMultiPickerChange" bindcolumnchange=
    <view class="picker">
      当前选择：{{multiArray[0][multiIndex[0]]}}, {{multiArray[1][multiIndex[1]]}}, {
    </view>
  </picker>
</view>
<view class="section">
  <view class="section__title">时间选择器</view>
  <picker mode="time" value="{{time}}" start="09:01" end="21:01" bindchange="bindTi
    <view class="picker">
      当前选择: {{time}}
    </view>
  </picker>
</view>

<view class="section">
  <view class="section__title">日期选择器</view>
  <picker mode="date" value="{{date}}" start="2015-09-01" end="2017-09-01" bindchan
    <view class="picker">
      当前选择: {{date}}
    </view>
  </picker>
</view>
<view class="section">
  <view class="section__title">省市区选择器</view>
  <picker mode="region" bindchange="bindRegionChange" value="{{region}}" custom-ite
    <view class="picker">
      当前选择：{{region[0]}}, {{region[1]}}, {{region[2]}}
    </view>
  </picker>
</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
Page({
  data: {
    array: ['美国', '中国', '巴西', '日本'],
    objectArray: [
      {
        id: 0,
        name: '美国'
      },
      {
        id: 1,
        name: '中国'
```

版权所有：腾讯云计算（北京）有限责任公司

```
      },
      {
        id: 2,
        name: '巴西'
      },
      {
        id: 3,
        name: '日本'
      }
    ],
    index: 0,
    multiArray: [['无脊柱动物', '脊柱动物'], ['扁性动物', '线形动物', '环节动物', '软体动物
    objectMultiArray: [
      [
        {
          id: 0,
          name: '无脊柱动物'
        },
        {
          id: 1,
          name: '脊柱动物'
        }
      ], [
        {
          id: 0,
          name: '扁性动物'
        },
        {
          id: 1,
          name: '线形动物'
        },
        {
          id: 2,
          name: '环节动物'
        },
        {
          id: 3,
          name: '软体动物'
        },
        {
          id: 4,
          name: '节肢动物'
        }
      ], [
        {
          id: 0,
          name: '猪肉绦虫'
```

```
        },
        {
          id: 1,
          name: '吸血虫'
        }
      ]
    ],
    multiIndex: [0, 0, 0],
    date: '2016-09-01',
    time: '12:01',
    region: ['广东省', '广州市', '海珠区'],
    customItem: '全部'
  },
  bindPickerChange: function(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      index: e.detail.value
    })
  },
  bindMultiPickerChange: function (e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      multiIndex: e.detail.value
    })
  },
  bindMultiPickerColumnChange: function (e) {
    console.log('修改的列为', e.detail.column, '，值为', e.detail.value);
    var data = {
      multiArray: this.data.multiArray,
      multiIndex: this.data.multiIndex
    };
    data.multiIndex[e.detail.column] = e.detail.value;
    switch (e.detail.column) {
      case 0:
        switch (data.multiIndex[0]) {
          case 0:
            data.multiArray[1] = ['扁性动物', '线形动物', '环节动物', '软体动物', '节肢动
            data.multiArray[2] = ['猪肉绦虫', '吸血虫'];
            break;
          case 1:
            data.multiArray[1] = ['鱼', '两栖动物', '爬行动物'];
            data.multiArray[2] = ['鲫鱼', '带鱼'];
            break;
        }
        data.multiIndex[1] = 0;
        data.multiIndex[2] = 0;
        break;
```

```
      case 1:
        switch (data.multiIndex[0]) {
          case 0:
            switch (data.multiIndex[1]) {
              case 0:
                data.multiArray[2] = ['猪肉绦虫', '吸血虫'];
                break;
              case 1:
                data.multiArray[2] = ['蛔虫'];
                break;
              case 2:
                data.multiArray[2] = ['蚂蚁', '蚂蟥'];
                break;
              case 3:
                data.multiArray[2] = ['河蚌', '蜗牛', '蛞蝓'];
                break;
              case 4:
                data.multiArray[2] = ['昆虫', '甲壳动物', '蛛形动物', '多足动物'];
                break;
            }
            break;
          case 1:
            switch (data.multiIndex[1]) {
              case 0:
                data.multiArray[2] = ['鲫鱼', '带鱼'];
                break;
              case 1:
                data.multiArray[2] = ['青蛙', '娃娃鱼'];
                break;
              case 2:
                data.multiArray[2] = ['蜥蜴', '龟', '壁虎'];
                break;
            }
            break;
        }
        data.multiIndex[2] = 0;
        break;
    }
    console.log(data.multiIndex);
    this.setData(data);
  },
  bindDateChange: function(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      date: e.detail.value
    })
  },
```

```
  bindTimeChange: function(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      time: e.detail.value
    })
  },
  bindRegionChange: function (e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      region: e.detail.value
    })
  }
})
```

picker-view

功能说明：嵌入页面的滚动选择器。其中只可放置picker-view-column组件，其它节点不会显示。

版权所有：腾讯云计算（北京）有限责任公司

参数及说明：

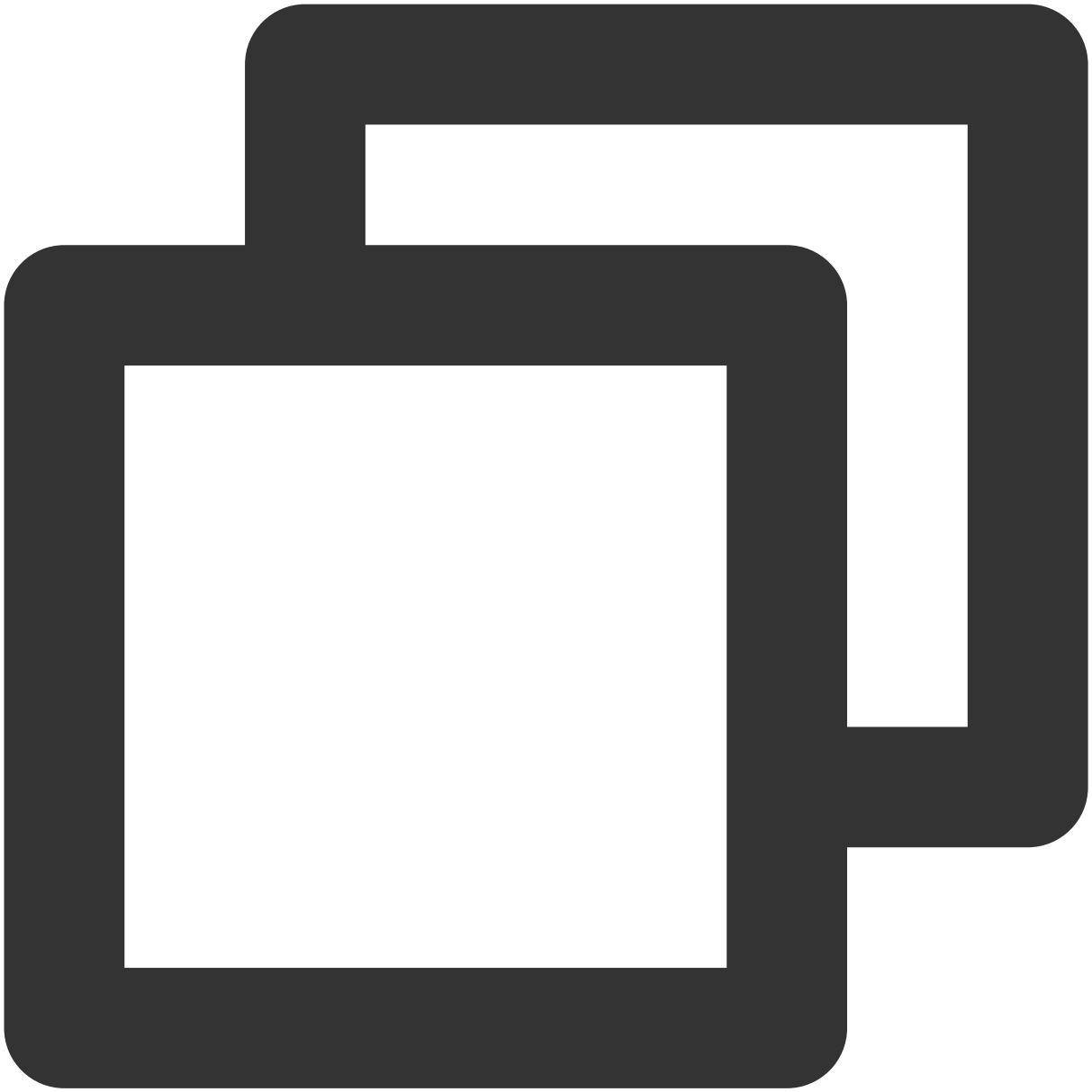
属性	类型	默认值	必填	说明
value	Array.<number>	-	否	数组中的数字依次表示 picker-view 内的 picker-view-column 选择的第几项（下标从 0 开始），数字大于 picker-view-column 可选项长度时，选择最后一项。
mask-class	string	-	否	设置蒙层的类名
indicator-style	string	-	否	设置选择器中间选中框的样式
bindchange	eventhandle	-	否	滚动选择时触发 change 事件，event.detail = {value}；value为数组，表示 picker-view 内的 picker-view-column 当前选择的是第几项（下标从 0 开始）
bindpickstart	eventhandle	-	否	当滚动选择开始时候触发事件
bindpickend	eventhandle	-	否	当滚动选择结束时候触发事件
indicator-class	string	-	否	设置选择器中间选中框的类名
mask-style	string	-	否	设置蒙层的样式
immediate-change	boolean	false	否	是否在手指松开时立即触发 change 事件。若不开启则会在滚动动画结束后触发 change 事件

示例代码：

WXML

JAVASCRIPT

版权所有：腾讯云计算（北京）有限责任公司

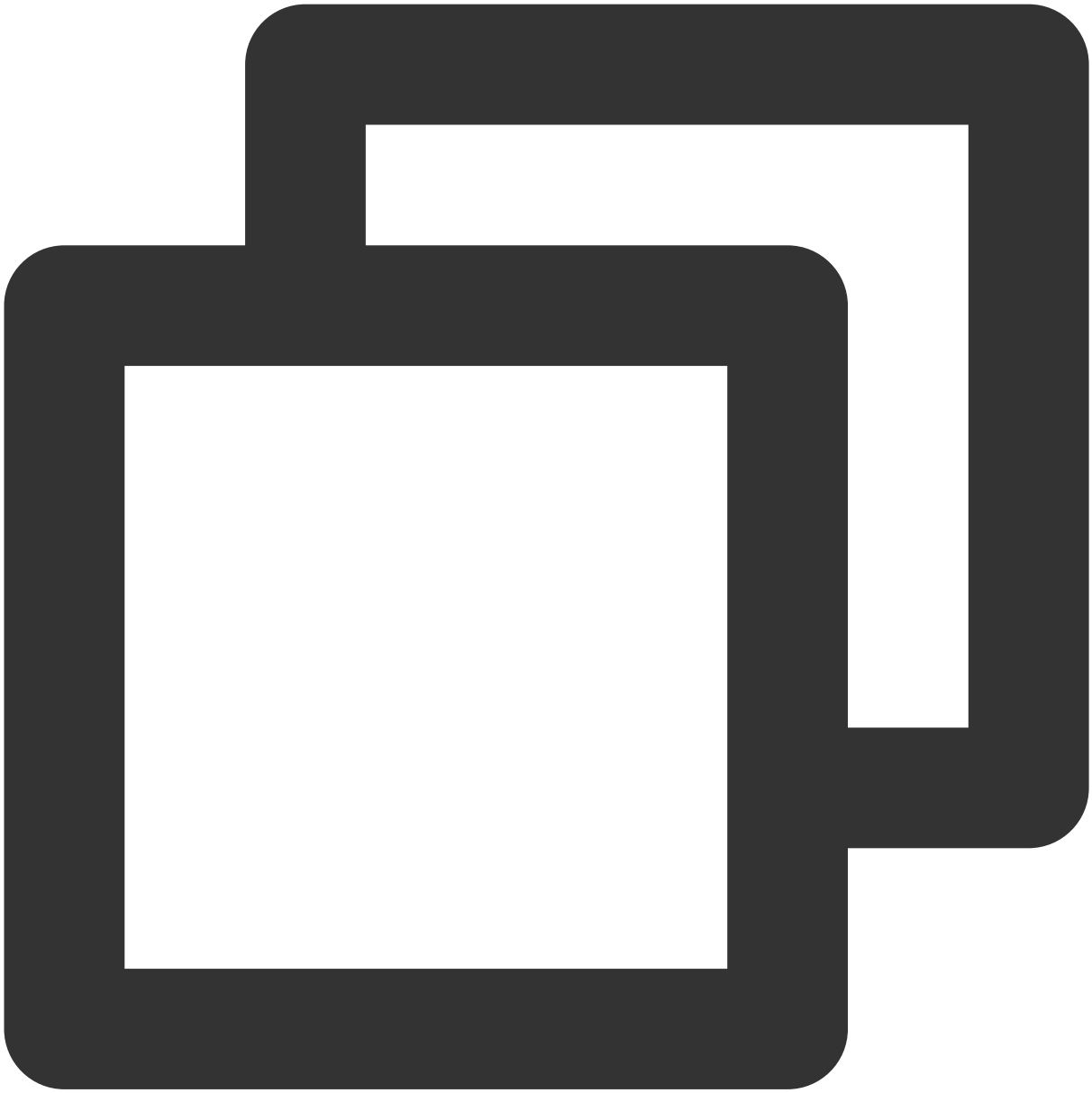


```
<view class="container">
  <view class="page-body">
    <view class="selected-date">{{year}}年{{month}}月{{day}}日{{isDaytime ? "白天" :
    <picker-view indicator-style="height: 50px;" style="width: 100%; height: 300px;
      <picker-view-column>
        <view wx:for="{{years}}" wx:key="{{years}}" style="line-height: 50px; text-
      </picker-view-column>
      <picker-view-column>
        <view wx:for="{{months}}" wx:key="{{months}}" style="line-height: 50px; tex
      </picker-view-column>
      <picker-view-column>
```

```
        <view wx:for="{{days}}" wx:key="{{days}}" style="line-height: 50px; text-al
      </picker-view-column>
      <picker-view-column>
        <view class="icon-container">
          <image class="picker-icon" src="../lib/daytime.png" />
        </view>
        <view class="icon-container">
          <image class="picker-icon" src="../lib/night.png" />
        </view>
      </picker-view-column>
    </picker-view>
  </view>

</view>
```

版权所有：腾讯云计算（北京）有限责任公司



```
const date = new Date()
const years = []
const months = []
const days = []

for (let i = 1990; i <= date.getFullYear(); i++) {
  years.push(i)
}

for (let i = 1; i <= 12; i++) {
  months.push(i)
```

版权所有：腾讯云计算（北京）有限责任公司

```
}

for (let i = 1; i <= 31; i++) {
  days.push(i)
}

Page({
  onShareAppMessage() {
    return {
      title: 'picker-view',
      path: 'page/component/pages/picker-view/picker-view'
    }
  },

  data: {
    years,
    year: date.getFullYear(),
    months,
    month: 2,
    days,
    day: 2,
    value: [9999, 1, 1],
    isDaytime: true,
  },

  bindChange(e) {
    const val = e.detail.value
    this.setData({
      year: this.data.years[val[0]],
      month: this.data.months[val[1]],
      day: this.data.days[val[2]],
      isDaytime: !val[3]
    })
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

picker-view-column

功能说明：滚动选择器子项。仅可放置于picker-view中，其子节点的高度会自动设置成与picker-view的选中框的高度一致。

版权所有：腾讯云计算（北京）有限责任公司

radio

功能说明：单选项目。

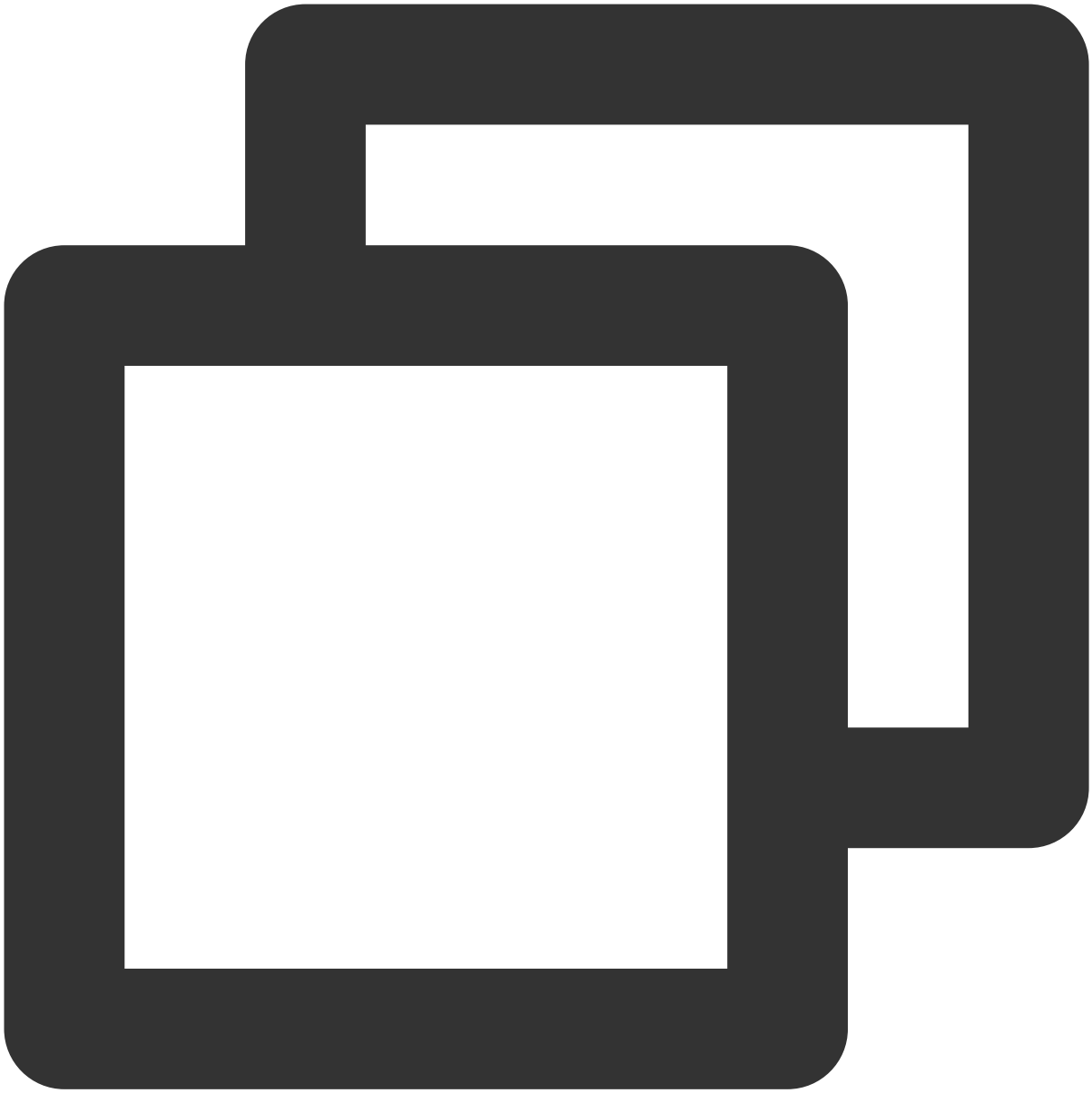
参数及说明

属性	类型	默认值	必填	说明
value	string	-	否	radio 标识。当该 radio 选中时，radio-group 的 change 事件会携带 radio 的 value
checked	boolean	false	否	当前是否选中
disabled	boolean	false	否	是否禁用
color	string	#09BB07	否	radio 的颜色，同 css 的 color

示例代码：

WXML

JAVASCRIPT

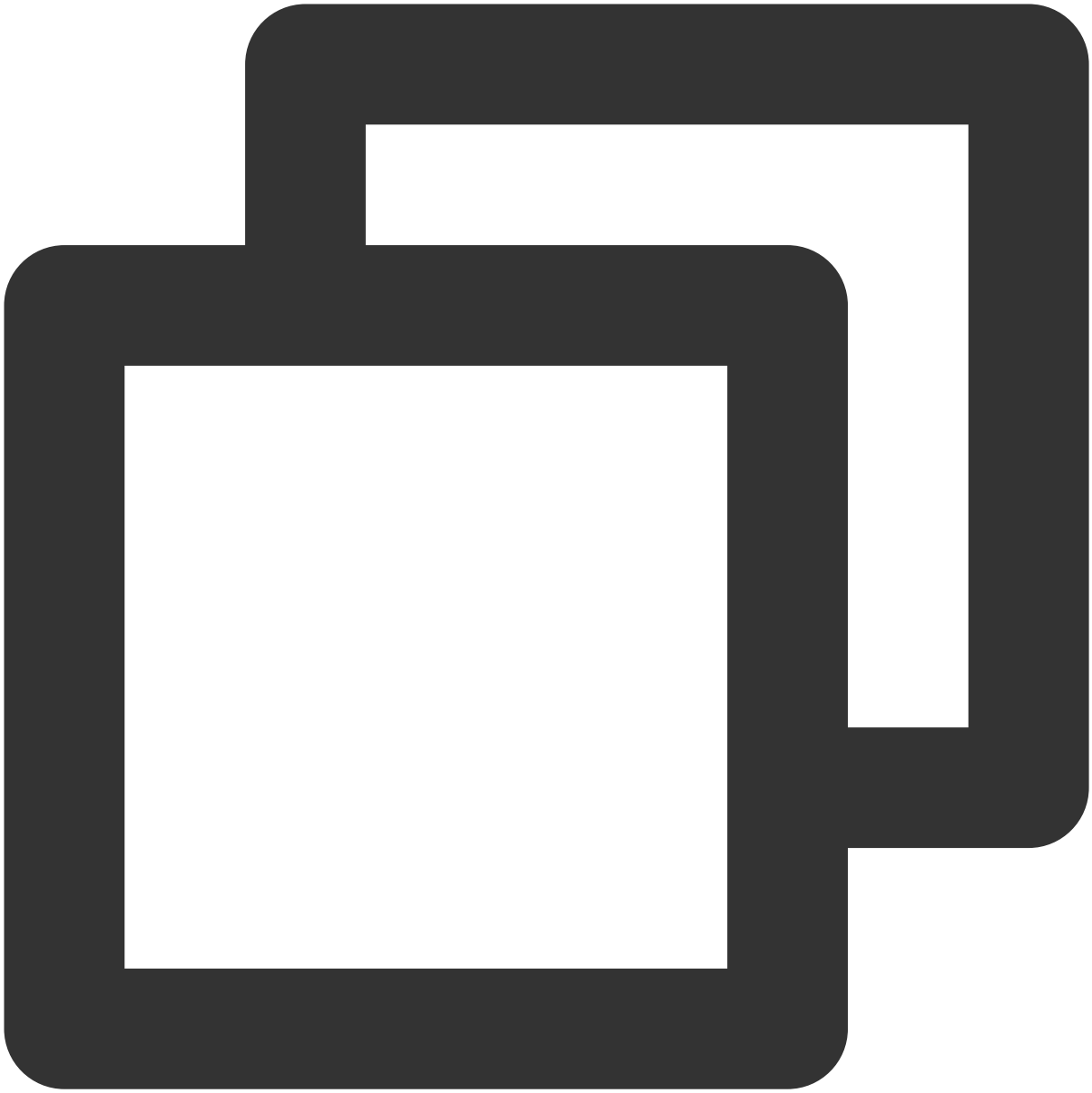


```
<view class="page-body">
  <view class="page-section">
    <view class="page-section-title">默认样式</view>
    <label class="radio">
      <radio value="r1" checked="true"/>选中
    </label>
    <label class="radio">
      <radio value="r2" />未选中
    </label>
  </view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
  <view class="page-section">
    <view class="page-section-title">推荐展示样式</view>
    <view class="weui-cells weui-cells_after-title">
      <radio-group bindchange="radioChange">
        <label class="weui-cell weui-check__label" wx:for="{{items}}" wx:key="{{ite

          <view class="weui-cell__hd">
            <radio value="{{item.value}}" checked="true"/>
          </view>
          <view class="weui-cell__bd">{{item.name}}</view>
        </label>
      </radio-group>
    </view>
  </view>
</view>
```



```
Page({
  onShareAppMessage() {
    return {
      title: 'radio',
      path: 'page/component/pages/radio/radio'
    }
  },

  data: {
    items: [
      {value: 'USA', name: '美国'},
```

版权所有：腾讯云计算（北京）有限责任公司

```
      {value: 'CHN', name: '中国', checked: 'true'},
      {value: 'BRA', name: '巴西'},
      {value: 'JPN', name: '日本'},
      {value: 'ENG', name: '英国'},
      {value: 'FRA', name: '法国'},
    ]
  },

  radioChange(e) {
    console.log('radio发生change事件，携带value值为：', e.detail.value)

    const items = this.data.items
    for (let i = 0, len = items.length; i < len; ++i) {
      items[i].checked = items[i].value === e.detail.value
    }

    this.setData({
      items
    })
  }
})
```

radio-group

功能说明：单项选择器，内部由多个 radio 组成。

参数及说明

属性	类型	默认值	必填	说明
bindchange	eventhandle	-	否	radio-group 中选中项发生改变时触发 change 事件，detail = {value:[选中的 radio 的 value 的数组]}

slider

功能说明：滑动选择器。

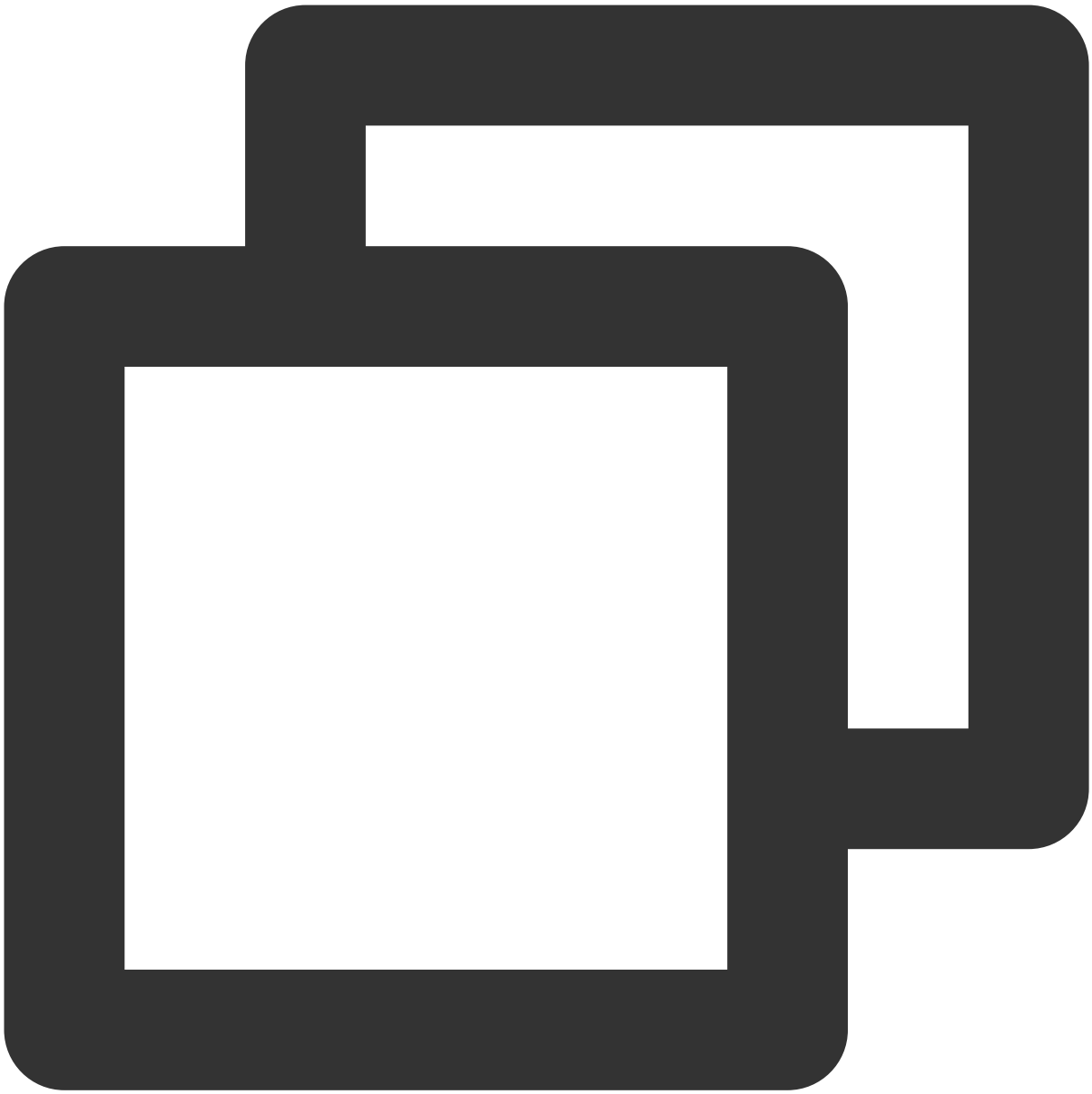
参数及说明

属性	类型	默认值	必填	说明
min	number	0	否	最小值
max	number	100	否	最大值
step	number	1	否	步长，取值必须大于 0，并且可被(max - min)整除
disabled	boolean	false	否	是否禁用
value	number	0	否	当前取值
color	color	#e9e9e9	否	背景条的颜色（请使用 backgroundColor）
selected-color	color	#1aad19	否	已选择的颜色（请使用 activeColor）
activeColor	color	#1aad19	否	已选择的颜色
backgroundColor	color	#e9e9e9	否	背景条的颜色
block-size	number	28	否	滑块的大小，取值范围为 12 - 28
block-color	color	#ffffff	否	滑块的颜色
show-value	boolean	false	否	是否显示当前 value
bindchange	eventhandle	-	否	完成一次拖动后触发的事件，event.detail = {value}
bindchanging	eventhandle	-	否	拖动过程中触发的事件，event.detail = {value}

示例代码：

WXML

JAVASCRIPT

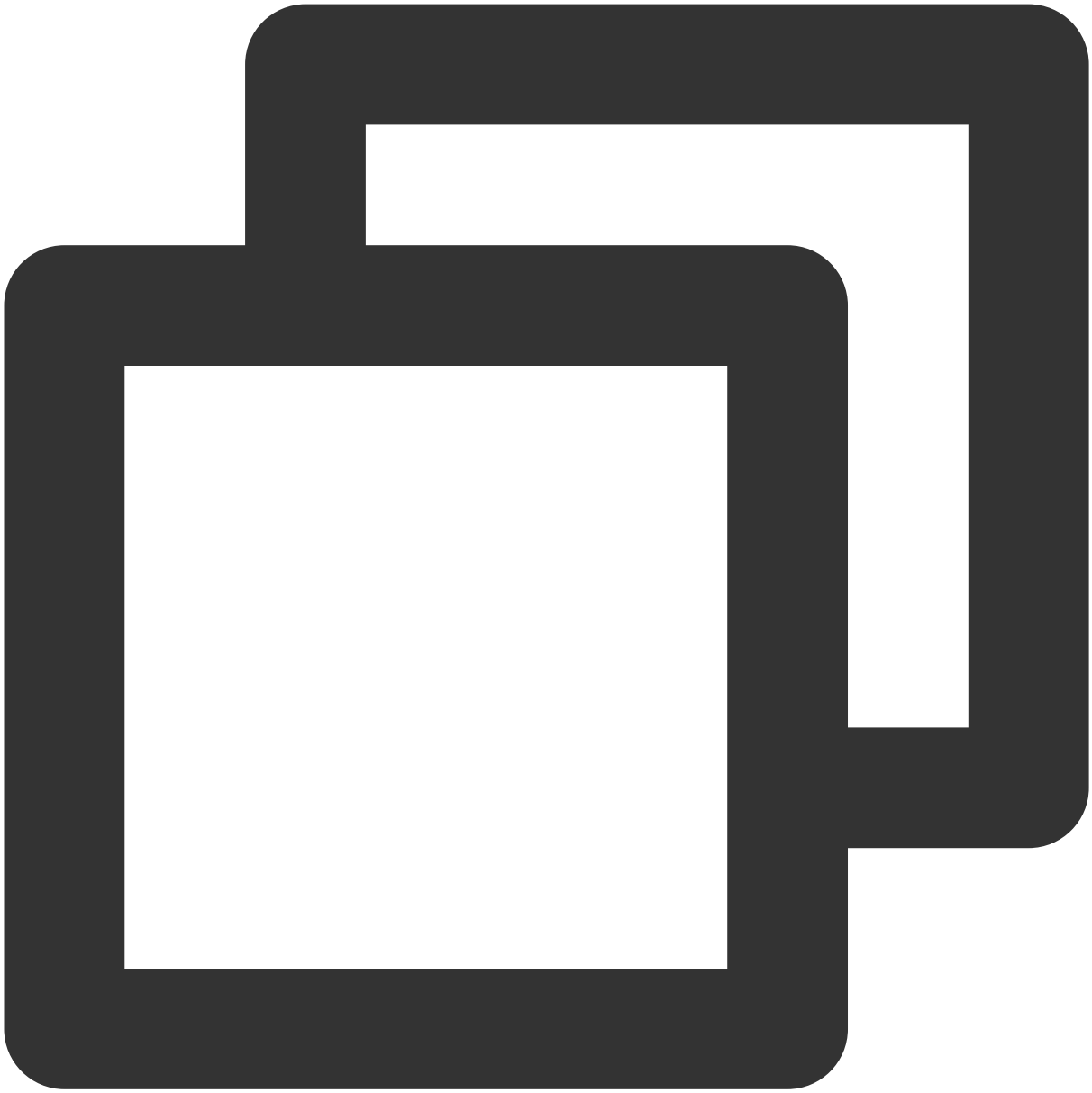


```
<view class="page">
    <view class="page__hd">
        <text class="page__title">slider</text>
        <text class="page__desc">滑块</text>
    </view>
    <view class="page__bd">
        <view class="page-section page-section-gap">
          <view class="page-section-title">设置step</view>
          <view class="body-view">
```

```
            <slider value="60" bindchange="slider2change" step="5"/>
          </view>
        </view>

        <view class="page-section page-section-gap">
          <view class="page-section-title">显示当前value</view>
          <view class="body-view">
            <slider value="50" bindchange="slider3change" show-value/>
          </view>
        </view>

        <view class="page-section page-section-gap">
          <view class="page-section-title">设置最小/最大值</view>
          <view class="body-view">
            <slider value="100" bindchange="slider4change" min="50" max="200" show-
          </view>
        </view>
    </view>
</view>
```



```
var pageData = {}
for (var i = 1; i < 5; ++i) {
  (function (index) {
    pageData[`slider${index}change`] = function (e) {
      console.log(`slider${index}发生change事件，携带值为`, e.detail.value)
    }
  })(i);
}
Page(pageData)
```

版权所有：腾讯云计算（北京）有限责任公司

switch

功能说明：开关选择器。

参数及说明

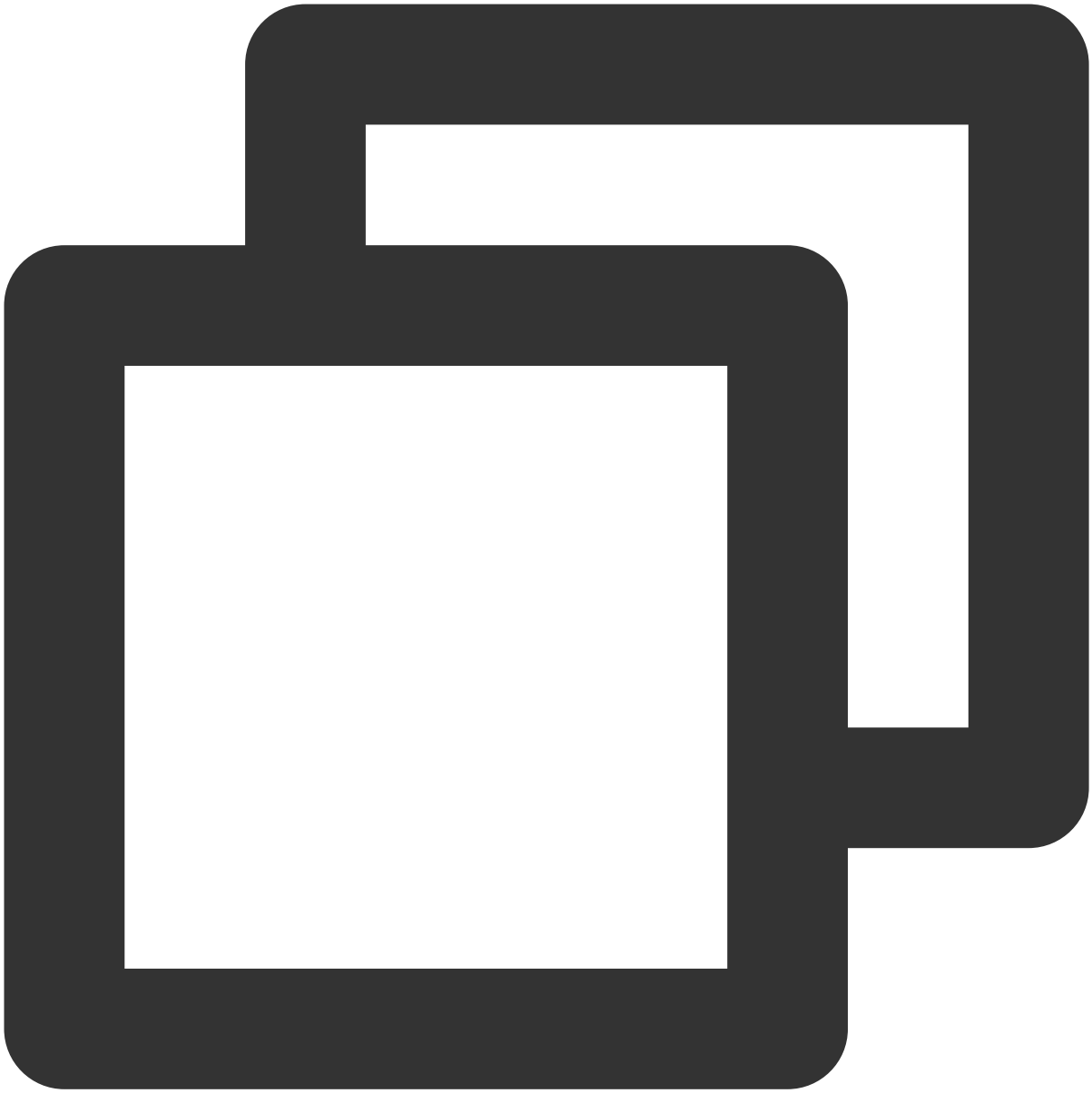
属性	类型	默认值	必填	说明

checked	boolean	false	否	是否选中
disabled	boolean	false	否	是否禁用
type	string	switch	否	样式，有效值：switch，checkbox
color	string	#04BE02	否	switch 的颜色，同 css 的 color
bindchange	eventhandle	-	否	点击导致 checked 改变时会触发 change 事件，event.detail={ value}

示例代码：

WXML

JAVASCRIPT

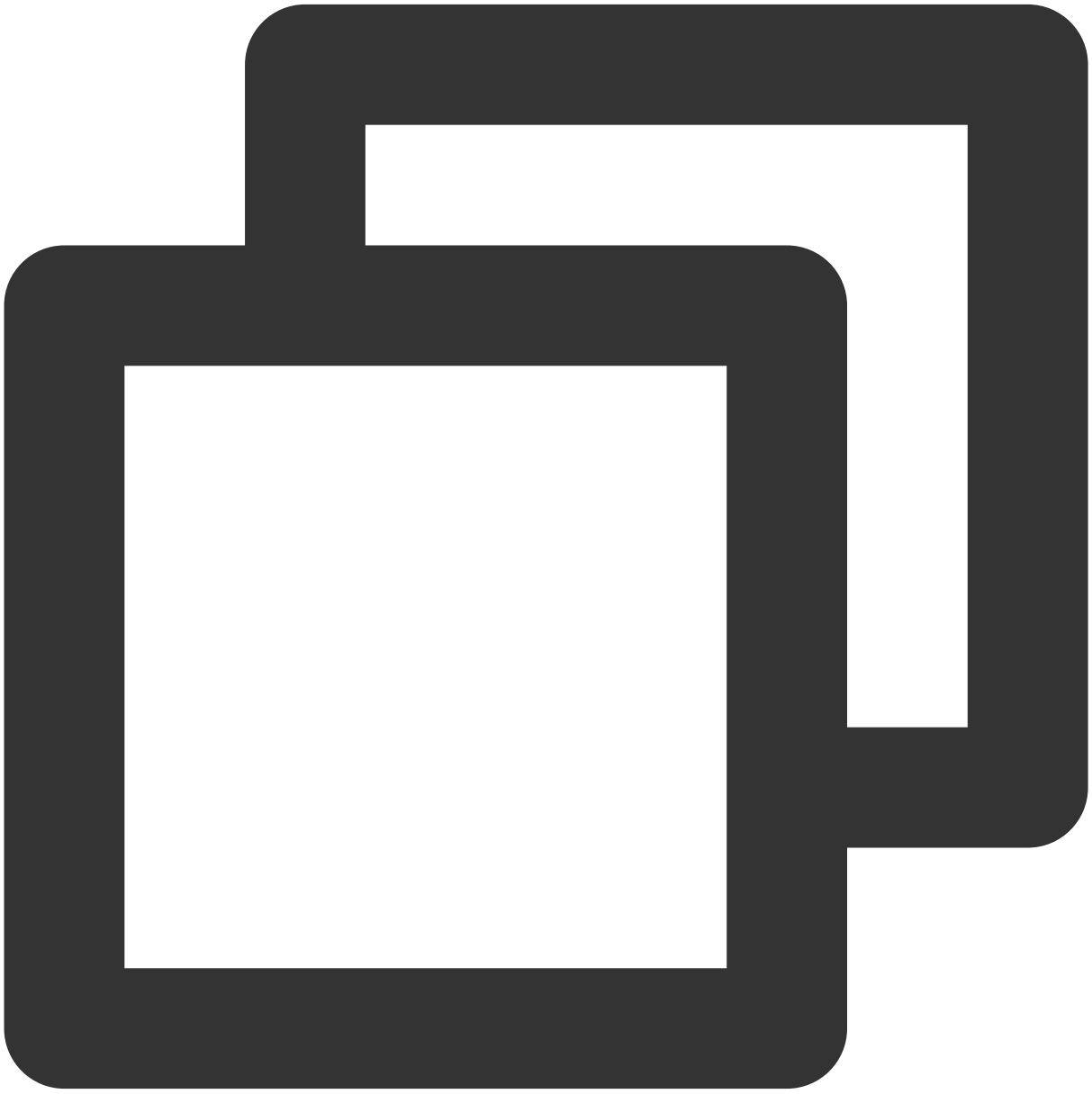


```
<view class="page">
    <view class="page__hd">
        <text class="page__title">switch</text>
        <text class="page__desc">开关</text>
    </view>
    <view class="page__bd">
        <view class="section section_gap">
            <view class="section__title">type="switch"</view>
            <view class="body-view">
                <switch checked="{{switch1Checked}}" bindchange="switch1Change"/>
            </view>
```

版权所有：腾讯云计算（北京）有限责任公司

```
        </view>

        <view class="section section_gap">
            <view class="section__title">type="checkbox"</view>
            <view class="body-view">
                <switch type="checkbox" checked="{{switch2Checked}}" bindchange="sw
            </view>
        </view>
    </view>
</view>
```



```
var pageData = {
  data: {
    switch1Checked: true,
    switch2Checked: false,
    switch1Style: '',
    switch2Style: 'text-decoration: line-through'
  }
}
for (var i = 1; i <= 2; ++i) {
  (function (index) {
    pageData[`switch${index}Change`] = function (e) {
      console.log(`switch${index}发生change事件，携带值为`, e.detail.value)
      var obj = {}
      obj[`switch${index}Checked`] = e.detail.value
      this.setData(obj)
      obj = {}
      obj[`switch${index}Style`] = e.detail.value ? '' : 'text-decoration: line-thr
      this.setData(obj)
    }
  })(i)
}
Page(pageData)
```

版权所有：腾讯云计算（北京）有限责任公司

textarea

版权所有：腾讯云计算（北京）有限责任公司

功能说明：多行输入框，该组件是 原生组件，使用时请注意相关限制。

参数及说明：

属性	类型	合法值及说明	默认值	必填	说明
value	string	-	-	否	输入框的内容
placeholder	string	-	-	否	输入框为空时占位符
placeholder-style	string	-	-	否	指定 placeholder 的样式，目前仅支持 color，font-size 和 font-weight
placeholder-class	string	-	textarea-placeholder	否	指定 placeholder 的样式类
fixed	boolean	-	false	否	如果 textarea 是在一个position:fixed的区域，需要显示指定属性 fixed 为 true
show-confirm-bar	boolean	-	true	否	是否显示键盘上方带有“完成”按钮那一栏
disabled	boolean	-	false	否	是否禁用
maxlength	number	-	140	否	最大输入长度，设置为 -1 的时候不限制最大长度
auto-focus	boolean	-	false	否	自动聚焦，拉起键盘。
focus	boolean	-	false	否	获取焦点
auto-height	boolean	-	false	否	是否自动增高，设置 auto-height 时，style.height 不生效
cursor-spacing	number / string	-	0	否	指定光标与键盘的距离。取textarea距离底部的距离和cursor-spacing指定的距离的最小值作为光标与键盘的距离

版权所有：腾讯云计算（北京）有限责任公司

cursor	number	-	-1	否	指定 focus 时的光标位置
selection-start	number	-	-1	否	光标起始位置，自动聚集时有效，需与 `selection-end` 搭配使用
selection-end	number	-	-1	否	光标结束位置，自动聚集时有效，需与 `selection-start` 搭配使用
adjust-position	boolean	-	true	否	键盘弹起时，是否自动上推页面
hold-keyboard	boolean	-	false	否	focus时，点击页面的时候不收起键盘
confirm-type	string	send：右下角按钮为“发送”； search：右下角按钮为“搜索”； next：右下角按钮为“下一个”； go：右下角按钮为“前往”； done：右下角按钮为“完成”； return：右下角按钮为“换行”；	return	否	设置键盘右下角按钮的文字
confirm-hold	boolean	-	false	否	点击键盘右下角按钮时是否保持键盘不收起
bindfocus	eventhandle	-	-	否	输入框聚焦时触发，event.detail = { value, height }，height 为键盘高度

版权所有：腾讯云计算（北京）有限责任公司

bindblur	eventhandle	-	-	否	输入框失去焦点时触发，event.detail = {value, cursor}
bindlinechange	eventhandle	-	-	否	输入框行数变化时调用，event.detail = {height: 0, heightRpx: 0, lineCount: 0}
bindinput	eventhandle	-	-	否	当键盘输入时，触发 input 事件，event.detail = {value, cursor, keyCode}，keyCode 为键值，目前工具还不支持返回keyCode参数。**bindinput 处理函数的返回值并不会反映到 textarea 上**
bindconfirm	eventhandle	-	-	否	点击完成时， 触发 confirm 事件，event.detail = {value: value}
bindkeyboardheightchange	eventhandle	-	-	否	键盘高度发生变化的时候触发此事件，event.detail = {height: height, duration: duration}

注意：

textarea 的 blur 事件会晚于页面上的 tap 事件，如果需要在 button 的点击事件获取 textarea，可以使用 form 的 bindsubmit。

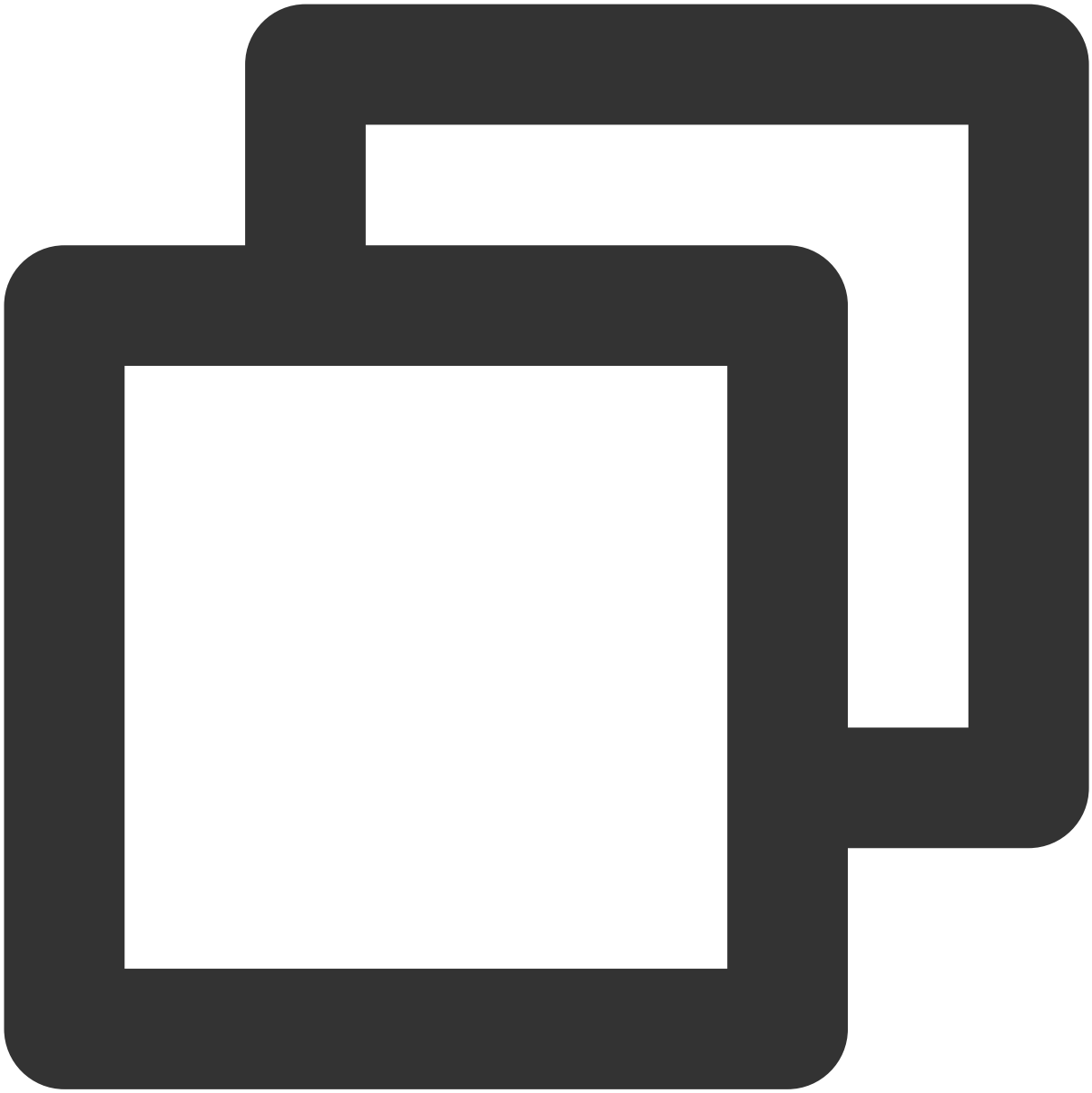
不建议在多行文本上对用户的输入进行修改，所以 textarea 的 bindinput 处理函数并不会将返回值反映到 textarea 上。

键盘高度发生变化，keyboardheightchange事件可能会多次触发，开发者对于相同的height值应该忽略掉。

示例代码：

WXML

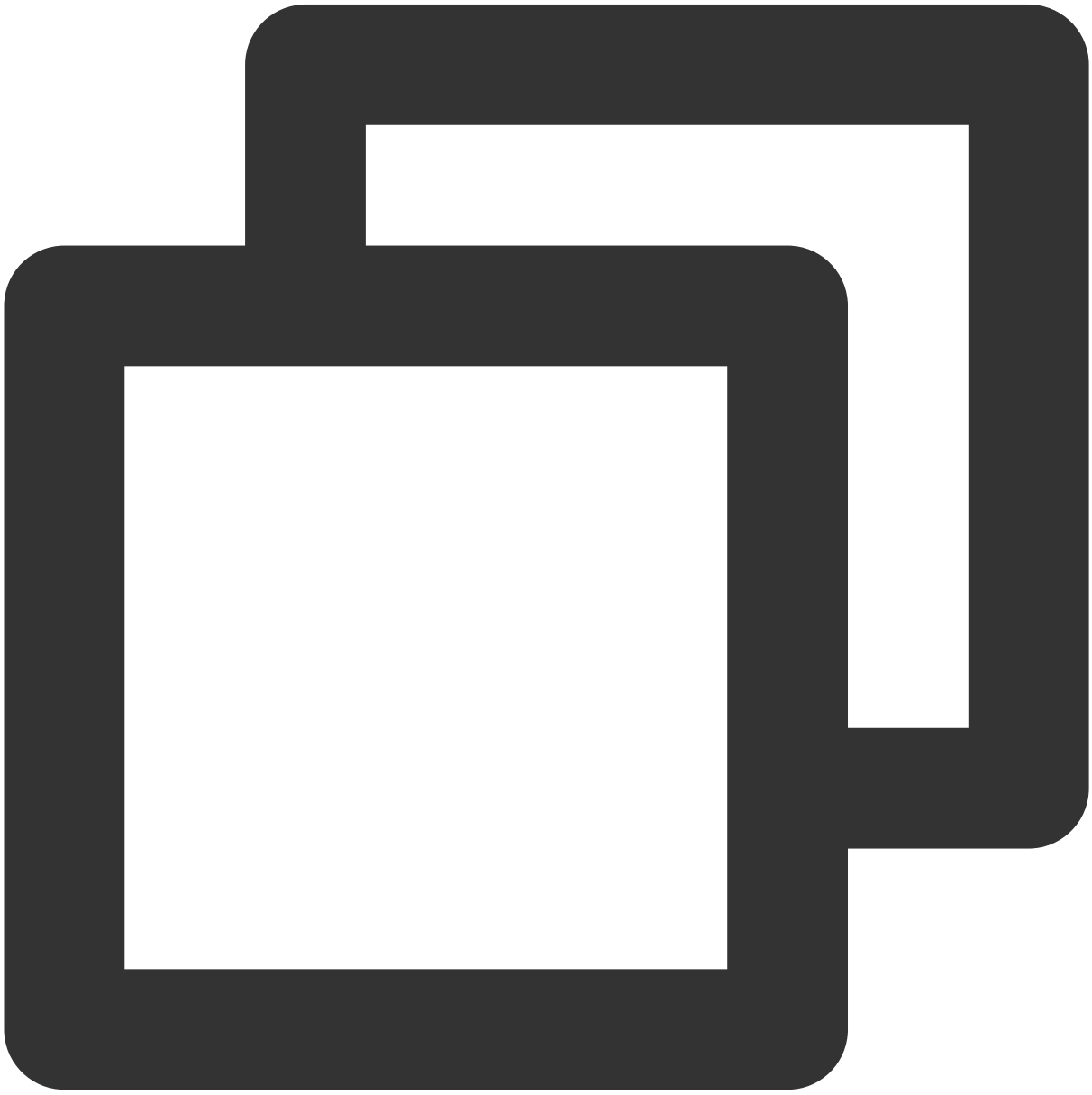
JAVASCRIPT



```
<view class="section">
  <textarea bindblur="bindTextAreaBlur" auto-height placeholder="自动变高" />
</view>
<view class="section">
  <textarea placeholder="placeholder颜色是红色的" placeholder-style="color:red;"  />
</view>
<view class="section">
  <textarea placeholder="这是一个可以自动聚焦的textarea" auto-focus />
</view>
<view class="section">
  <textarea placeholder="这个只有在按钮点击的时候才聚焦" focus="{{focus}}" />
```

版权所有：腾讯云计算（北京）有限责任公司

```
  <view class="btn-area">
    <button bindtap="bindButtonTap">使得输入框获取焦点</button>
  </view>
</view>
<view class="section">
  <form bindsubmit="bindFormSubmit">
    <textarea placeholder="form 中的 textarea" name="textarea"/>
    <button form-type="submit"> 提交 </button>
  </form>
</view>
```



```
Page({
  data: {
    height: 20,
    focus: false
  },
  bindButtonTap: function() {
    this.setData({
      focus: true
    })
  },
  bindTextAreaBlur: function(e) {
    console.log(e.detail.value)
  },
  bindFormSubmit: function(e) {
    console.log(e.detail.value.textarea)
  }
})
```

版权所有：腾讯云计算（北京）有限责任公司

版权所有：腾讯云计算（北京）有限责任公司

API

API 概览

最近更新时间：2024-07-12 19:09:16

1、基础

1.1、系统能力

名称	功能说明	最低版本
canIUse	判断小程序的 API，回调，参数，组件等是否在当前版本可用	1.5.0
env	环境变量	1.5.1
base64ToArrayBuffer	将 Base64 字符串转成 ArrayBuffer 对象	1.5.1
arrayBufferToBase64	将 ArrayBuffer 对象转成 Base64 字符串	1.5.1

1.2、系统

名称	功能说明	最低版本
getSystemInfo	异步调用获取系统信息	1.5.0
getSystemInfoSync	同步调用获取系统信息	1.5.1
getSystemInfoAsync	异步获取系统信息	1.5.1
getWindowInfo	获取窗口信息	1.5.1
getSystemSetting	获取设备设置	1.5.0
openSystemBluetoothSetting	跳转系统蓝牙设置页	1.5.1
openAppAuthorizeSetting	跳转系统宿主客户端授权管理页	1.5.1
getRendererUserAgent	获取 Webview 小程序的 UserAgent	1.5.1
getDeviceInfo	获取设备基础信息	1.5.1
getAppBaseInfo	获取宿主客户端基础信息	1.5.1
getAppAuthorizeSetting	获取宿主客户端授权设置	1.5.1

版权所有：腾讯云计算（北京）有限责任公司

1.3、更新

名称	功能说明	最低版本
getUpdateManager	获取全局唯一的版本更新管理器，用于管理小程序更新	1.5.1
UpdateManager.applyUpdate	强制小程序重启并使用新版本	1.5.0
UpdateManager.onCheckForUpdate	监听向微信后台请求检查更新结果事件	1.5.0
UpdateManager.onUpdateFailed	监听小程序更新失败事件	1.5.0
UpdateManager.onUpdateReady	监听小程序有版本更新事件	1.5.0

1.4、小程序

名称	功能说明	最低版本
getLaunchOptionsSync	获取小程序启动时的参数	1.5.0
getEnterOptionsSync	获取本次小程序启动时的参数	1.5.1
onError	监听小程序错误事件	1.5.0
offError	移除小程序错误事件的监听函数	1.5.0
onThemeChange	监听系统主题改变事件	1.5.0
offThemeChange	移除系统主题改变事件的监听函数	1.5.0
onPageNotFound	监听小程序要打开的页面不存在事件	1.5.1
offPageNotFound	移除小程序要打开的页面不存在事件的监听函数	1.5.1
onAppShow	监听小程序切前台事件	1.5.1
offAppShow	移除小程序切前台事件的监听函数	1.5.1
onAppHide	监听小程序切后台事件	1.5.1
offAppHide	移除小程序切前台事件的监听函数	1.5.1
onUnhandledRejection	监听未处理的 Promise 拒绝事件	1.5.1
offUnhandledRejection	移除未处理的 Promise 拒绝事件的监听函数	1.5.1

1.5、调试

版权所有：腾讯云计算（北京）有限责任公司

名称	功能说明	最低版本
setEnableDebug	设置是否打开调试开关	1.5.0
getLogManager	获取日志管理器对象	1.5.1
console.debug	向调试面板中打印 debug 日志	1.5.0
console.error	向调试面板中打印 error 日志	1.5.0
console.group	在调试面板中创建一个新的分组	1.5.0
console.groupEnd	结束由 console.group 创建的分组	1.5.0
console.info	向调试面板中打印 info 日志	1.5.0
console.log	向调试面板中打印 log 日志	1.5.0
console.warn	向调试面板中打印 warn 日志	1.5.0
LogManager.debug	写 debug 日志	1.5.0
LogManager.info	写 info 日志	1.5.0
LogManager.log	写 log 日志	1.5.0
LogManager.warn	写 warn 日志	1.5.0
RealtimeLogManager.addFilterMsg	添加过滤关键字	1.5.16
RealtimeLogManager.error	写 error 日志	1.5.16
RealtimeLogManager.getCurrentState	可以获取当前缓存剩余空间	1.5.16
RealtimeLogManager.in	设置实时日志page参数所在的页面	1.5.16
RealtimeLogManager.info	写 info 日志	1.5.16
RealtimeLogManager.setFilterMsg	设置过滤关键字	1.5.16
RealtimeLogManager.warn	写 warn 日志	1.5.16
RealtimeTagLogManageraddFilterMsg	添加过滤关键字	1.5.16
RealtimeTagLogManager.error	写 error日志	1.5.16
RealtimeTagLogManager.info	写 info 日志	1.5.16
RealtimeTagLogManager.setFilterMsg	设置过滤关键字	1.5.16

版权所有：腾讯云计算（北京）有限责任公司

RealtimeTagLogManager.warn	写 warn 日志	1.5.16

2、路由

名称	功能说明	最低版本
switchTab	跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面	1.5.0
reLaunch	关闭所有页面，打开到应用内的某个页面	1.5.0
redirectTo	关闭当前页面，跳转到应用内的某个页面	1.5.0
navigateTo	保留当前页面，跳转到应用内的某个页面	1.5.1
navigateBack	关闭当前页面，返回上一页面或多级页面	1.5.0
EventChannel	页面间事件通信通道	1.5.1

3、跳转

名称	功能说明	最低版本
exitMiniProgram	退出当前小程序	1.5.0
navigateToMiniProgram	打开另一个小程序	1.5.0
navigateBackMiniProgram	返回到上一个小程序。只有在当前小程序是被其他小程序打开时可以调用成功	1.5.1

4、转发

名称	功能说明	最低版本

版权所有：腾讯云计算（北京）有限责任公司

hideShareMenu	隐藏当前页面的转发按钮	1.5.0
showShareMenu	显示当前页面的转发按钮	1.5.0
updateShareMenu	更新转发属性	1.5.0
showShareImageMenu	打开分享图片弹窗	1.5.35
onCopyUrl	监听用户点击右上角菜单的「复制链接」按钮时触发的事件	1.5.35
offCopyUrl	移除用户点击右上角菜单的「复制链接」按钮时触发的事件的全部监听函数	1.5.35

5、界面

5.1、交互

名称	功能说明	最低版本
hideLoading	隐藏 loading 提示框	1.5.0
hideToast	隐藏消息提示框	1.5.0
showActionSheet	显示操作菜单	1.5.0
showLoading	显示 loading 提示框	1.5.0
showModal	显示模态对话框	1.5.0
showToast	显示消息提示框	1.5.0
enableAlertBeforeUnload	开启小程序页面返回询问对话框	1.5.1
disableAlertBeforeUnload	关闭小程序页面返回询问对话框	1.5.1

5.2、导航栏

名称	功能说明	最低版本
setNavigationBarTitle	动态设置当前页面的标题	1.5.0
setNavigationBarColor	设置页面导航条颜色	1.5.0

版权所有：腾讯云计算（北京）有限责任公司

hideHomeButton	隐藏返回首页按钮	1.5.0
showNavigationBarLoading	在当前页面显示导航条加载动画	1.5.1
hideNavigationBarLoading	在当前页面隐藏导航条加载动画	1.5.1

5.3、背景

名称	功能说明	最低版本
setBackgroundColor	动态设置窗口的背景色	1.5.0
setBackgroundTextStyle	动态设置下拉背景字体、loading 图的样式	1.5.0

5.4、TabBar

名称	功能说明	最低版本
showTabBar	显示 tabBar	1.5.0
hideTabBar	隐藏 tabBar	1.5.0
setTabBarStyle	动态设置 tabBar 的整体样式	1.5.0
setTabBarItem	动态设置 tabBar 某一项的内容，2.7.0 起图片支持临时文件和网络文件	1.5.0
showTabBarRedDot	显示 tabBar 某一项的右上角的红点	1.5.0
hideTabBarRedDot	隐藏 tabBar 某一项的右上角的红点	1.5.0
setTabBarBadge	为 tabBar 某一项的右上角添加文本	1.5.0
removeTabBarBadge	移除 tabBar 某一项右上角的文本	1.5.0

5.5、下拉刷新

名称	功能说明	最低版本
startPullDownRefresh	开始下拉刷新	1.5.0
stopPullDownRefresh	停止当前页面下拉刷新	1.5.0

5.6、滚动

名称	功能说明	最低版本

版权所有：腾讯云计算（北京）有限责任公司

pageScrollTo	将页面滚动到目标位置，支持选择器和滚动距离两种方式定位	1.5.0
ScrollViewContext.scrollIntoView	滚动至指定位置	1.5.0
ScrollViewContext.scrollTo	滚动至指定位置	1.5.0

5.7、动画

名称	功能说明	最低版本
createAnimation	创建一个动画实例 animation	1.5.0
Animation.backgroundColor	设置背景色	1.5.0
Animation.bottom	设置 bottom 值	1.5.0
Animation.export	导出动画队列	1.5.0
Animation.height	设置高度	1.5.0
Animation.left	设置 left 值	1.5.0
Animation.matrix	同 transform-function matrix	1.5.0
Animation.matrix3d	同 transform-function matrix3d	1.5.0
Animation.opacity	设置透明度	1.5.0
Animation.right	设置 right 值	1.5.0
Animation.rotate	从原点顺时针旋转一个角度	1.5.0
Animation.rotate3d	从 固定 轴顺时针旋转一个角度	1.5.0
Animation.rotateX	从 X 轴顺时针旋转一个角度	1.5.0
Animation.rotateY	从 Y 轴顺时针旋转一个角度	1.5.0
Animation.rotateZ	从 Z 轴顺时针旋转一个角度	1.5.0
Animation.scale	缩放	1.5.0
Animation.scale3d	缩放	1.5.0
Animation.scaleX	缩放 X 轴	1.5.0
Animation.scaleY	缩放 Y 轴	1.5.0

版权所有：腾讯云计算（北京）有限责任公司

Animation.scaleZ	缩放 Z 轴	1.5.0
Animation.skew	对 X、Y 轴坐标进行倾斜	1.5.0
Animation.skewX	对 X 轴坐标进行倾斜	1.5.0
Animation.skewY	对 Y 轴坐标进行倾斜	1.5.0
Animation.step	表示一组动画完成	1.5.0
Animation.top	设置 top 值	1.5.0
Animation.translate	平移变换	1.5.0
Animation.translate3d	对 xyz 坐标进行平移变换	1.5.0
Animation.translateX	对 X 轴平移	1.5.0
Animation.translateY	对 Y 轴平移	1.5.0
Animation.translateZ	对 Z 轴平移	1.5.0
Animation.width	设置宽度	1.5.0

5.8、自定义组件

名称	功能说明	最低版本
nextTick	延迟一部分操作到下一个时间片再执行	1.5.0

5.9、菜单

名称	功能说明	最低版本
getMenuButtonBoundingClientRect	获取菜单按钮（右上角胶囊按钮）的布局位置信息	1.5.0

5.10、窗口

名称	功能说明	最低版本
onWindowResize	监听窗口尺寸变化事件	1.5.0
offWindowResize	移除窗口尺寸变化事件的监听函数	1.5.1

5.11、字体

版权所有：腾讯云计算（北京）有限责任公司

名称	功能说明	最低版本
loadFontFace	动态加载网络字体	1.5.0

6、网络

6.1、发起请求

名称	功能说明	最低版本
request	发起 HTTPS 网络请求	1.5.0
RequestTask.abort	中断请求任务	1.5.0
RequestTask.onChunkReceived	监听 Transfer-Encoding Chunk Received 事件	1.5.1
RequestTask.offChunkReceived	移除 Transfer-Encoding Chunk Received 事件的监听函数	1.5.1
RequestTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.0
RequestTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.0

6.2、上传

名称	功能说明	最低版本
uploadFile	将本地资源上传到服务器	1.5.0
UploadTask.abort	中断上传任务	1.5.0
UploadTask.onProgressUpdate	监听上传进度变化事件	1.5.0
UploadTask.offProgressUpdate	取消监听上传进度变化事件	1.5.0
UploadTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.0
UploadTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.0

6.3、下载

名称	功能说明	最低版本
downloadFile	下载文件资源到本地。客户端直接发起一个 HTTPS	1.5.1

版权所有：腾讯云计算（北京）有限责任公司

	GET 请求，返回文件的本地临时路径 (本地路径)，单次下载允许的最大文件为200MB	
DownloadTask.abort	中断下载任务	1.5.1
DownloadTask.onProgressUpdate	监听下载进度变化事件	1.5.1
DownloadTask.offProgressUpdate	移除下载进度变化事件的监听函数	1.5.1
DownloadTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.1
DownloadTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.1

6.4、WebSocket

名称	功能说明	最低版本
sendSocketMessage	通过 WebSocket 连接发送数据。需要先 wx.connectSocket，并在 wx.onSocketOpen 回调之后才能发送	1.5.1
onSocketOpen	监听 WebSocket 连接打开事件	1.5.1
onSocketMessage	监听 WebSocket 接收到服务器的消息事件	1.5.1
onSocketError	监听 WebSocket 错误事件	1.5.1
onSocketClose	监听 WebSocket 连接关闭事件	1.5.1
connectSocket	创建一个 WebSocket 连接	1.5.1
closeSocket	关闭 WebSocket 连接	1.5.1
SocketTask.close	关闭 WebSocket 连接	1.5.0
SocketTask.onClose	监听 WebSocket 连接关闭事件	1.5.0
SocketTask.onError	监听 WebSocket 错误事件	1.5.0
SocketTask.onMessage	监听 WebSocket 接收到服务器的消息事件	1.5.0
SocketTask.onOpen	监听 WebSocket 连接打开事件	1.5.0
SocketTask.send	通过 WebSocket 连接发送数据	1.5.0

6.5、UDP 通信

名称	功能描述	最低版本

版权所有：腾讯云计算（北京）有限责任公司

createUDPSocket	创建一个 UDP Socket 实例	1.5.1
UDPSocket.bind	绑定一个系统随机分配的可用端口，或绑定一个指定的端口号	1.5.1
UDPSocket.close	关闭 UDP Socket 实例	1.5.1
UDPSocket.connect	预先连接到指定的 IP 和 port，需要配合 write 方法一起使用	1.5.1
UDPSocket.onClose	监听关闭事件	1.5.1
UDPSocket.offClose	移除关闭事件的监听函数	1.5.1
UDPSocket.onError	监听错误事件	1.5.1
UDPSocket.offError	移除错误事件的监听函数	1.5.1
UDPSocket.onListening	监听开始监听数据包消息的事件	1.5.1
UDPSocket.offListening	移除开始监听数据包消息的事件的监听函数	1.5.1
UDPSocket.onMessage	监听收到消息的事件	1.5.1
UDPSocket.offMessage	移除收到消息的事件的监听函数	1.5.1
UDPSocket.send	向指定的 IP 和 port 发送消息	1.5.1
UDPSocket.setTTL	设置 IP_TTL 套接字选项，用于设置一个 IP 数据包传输时允许的最大跳步数	1.5.16
UDPSocket.write	用法与 send 方法相同	1.5.16

6.6、TCP 通信

名称	功能说明	最低版本
createTCPSocket	创建一个 TCP Socket 实例	1.5.1
TCPSocket.bindWifi	将 TCP Socket 绑定到当前 Wi-Fi 网络，成功后会触发 onBindWifi 事件（仅 Android 支持）	1.5.1
TCPSocket.close	关闭连接	1.5.1
TCPSocket.connect	在给定的套接字上启动连接	1.5.1
TCPSocket.onClose	监听一旦 socket 完全关闭就发出该事件	1.5.1

版权所有：腾讯云计算（北京）有限责任公司

TCPSocket.offClose	移除一旦 socket 完全关闭就发出该事件的监听函数	1.5.1
TCPSocket.onConnect	监听当一个 socket 连接成功建立的时候触发该事件	1.5.1
TCPSocket.offConnect	移除当一个 socket 连接成功建立的时候触发该事件的监听函数	1.5.1
TCPSocket.onError	监听当错误发生时触发	1.5.1
TCPSocket.offError	移除当错误发生时触发的监听函数	1.5.1
TCPSocket.onMessage	监听当接收到数据的时触发该事件	1.5.1
TCPSocket.offMessage	移除当接收到数据的时触发该事件的监听函数	1.5.1
TCPSocket.onBindWifi	监听当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件	1.5.1
TCPSocket.offBindWifi	移除当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件的监听函数	1.5.1
TCPSocket.write	在 socket 上发送数据	1.5.1

6.7、mDNS

名称	功能说明	最低版本
stopLocalServiceDiscovery	停止搜索 mDNS 服务	1.5.1
startLocalServiceDiscovery	开始搜索局域网下的 mDNS 服务	1.5.1
offLocalServiceResolveFail	移除 mDNS 服务解析失败的事件的监听函数	1.5.1
onLocalServiceResolveFail	监听 mDNS 服务解析失败的事件	1.5.1
offLocalServiceLost	移除 mDNS 服务离开的事件的监听函数	1.5.1
onLocalServiceLost	监听 mDNS 服务离开的事件	1.5.1
offLocalServiceFound	移除 mDNS 服务发现的事件的监听函数	1.5.1
onLocalServiceFound	监听 mDNS 服务发现的事件	1.5.1
offLocalServiceDiscoveryStop	移除 mDNS 服务停止搜索的事件的监听函数	1.5.1
onLocalServiceDiscoveryStop	监听 mDNS 服务停止搜索的事件	1.5.1

版权所有：腾讯云计算（北京）有限责任公司

7、数据缓存

名称	功能说明	最低版本
setStorage	将数据存储在本地缓存中指定的 key 中	1.5.0
setStorageSync	将数据存储在本地缓存中指定的 key 中	1.5.0
revokeBufferURL	根据 URL 销毁存在内存中的数据	1.5.35
removeStorage	从本地缓存中移除指定 key	1.5.0
removeStorageSync	removeStorage的同步版本	1.5.0
getStorage	从本地缓存中异步获取指定 key 的内容	1.5.0
createBufferURL	根据传入的 buffer 创建一个唯一的 URL 存在内存中	1.5.35
getStorageSync	从本地缓存中同步获取指定 key 的内容	1.5.0
getStorageInfo	异步获取当前 storage 的相关信息	1.5.0
getStorageInfoSync	getStorageInfo的同步版本	1.5.0
clearStorage	清理本地数据缓存	1.5.0
clearStorageSync	clearStorage的同步版本	1.5.0
batchSetStorage	将数据批量存储在本地缓存中指定的 key 中	1.5.16
batchSetStorageSync	batchSetStorage 的同步版本	1.5.16
batchGetStorage	从本地缓存中异步批量获取指定 key 的内容	1.5.16
batchGetStorageSync	batchGetStorage 的同步版本	1.5.16

8、数据分析

名称	功能说明	最低版本
reportEvent	事件上报	1.5.16

版权所有：腾讯云计算（北京）有限责任公司

9、画布

9.1、画布概要

版权所有：腾讯云计算（北京）有限责任公司