

小程序面试真题（8题）

1. 说说你对微信小程序的理解？优缺点？



1.1. 是什么

2017年，微信正式推出了小程序，允许外部开发者在微信内部运行自己的代码，开展业务

截至目前，小程序已经成为国内前端的一个重要业务，跟 Web 和手机 App 有着同等的重要性



手机APP



微信小程序



H5微网站

小程序是一种不需要下载安装即可使用的应用，它实现了应用“触手可及”的梦想，用户扫一扫或者搜一下即可打开应用

也体现了“用完即走”的理念，用户不用关心是否安装太多应用的问题。应用将无处不在，随时可用，但又无需安装卸载

注意的是，除了微信小程序，还有百度小程序、微信小程序、支付宝小程序、抖音小程序，都是每个平台自己开发的，都是有针对性平台的应用程序

1.2. 背景

小程序并非凭空冒出来的一个概念，当微信中的 `WebView` 逐渐成为移动 `Web` 的一个重要入口时，微信就有相关的 `JS-SDK`

`JS-SDK` 解决了移动网页能力不足的问题，通过暴露微信的接口使得 `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

因此需要设计一个比较好的系统，使得所有开发者在微信中都能获得比较好的体验：

- 快速的加载
- 更强大的能力
- 原生的体验
- 易用且安全的微信数据开放
- 高效和简单的开发

这些是 `JS-SDK` 做不到的，需要设计一个全新的小程序系统

对于小程序的开发，提供一个简单、高效的应用开发框架和丰富的组件及 `API`，帮助开发者开发出具有原生体验的服务

其中相比 `H5`，小程序与它的区别有如下：

- 运行环境：小程序基于浏览器内核重构的内置解析器
- 系统权限：小程序能获得更多的系统权限，如网络通信状态、数据缓存能力等
- 渲染机制：小程序的逻辑层和渲染层是分开的

小程序可以视为只能用微信打开和浏览的 `H5`，小程序和网页的技术模型是一样的，用到的 `JavaScript` 语言和 `CSS` 样式也是一样的，只是网页的 `HTML` 标签被稍微修改成了 `WXML` 标签

因此可以说，小程序页面本质上就是网页

其中关于微信小程序的实现原理，我们在后面的文章讲到

1.3. 优缺点

优点：

- 随搜随用，用完即走：使得小程序可以代替许多APP，或是做APP的整体嫁接，或是作为阉割版功能的承载体

- 流量大，易接受：小程序借助自身平台更加容易引入更多的流量
- 安全
- 开发门槛低
- 降低兼容性限制

缺点：

- 用户留存：及相关数据显示，小程序的平均次日留存在13%左右，但是双周留存骤降到仅有1%
- 体积限制：微信小程序只有2M的大小，这样导致无法开发大型一些的小程序
- 受控微信：比起APP，尤其是安卓版的高自由度，小程序要面对很多来自微信的限制，从功能接口，甚至到类别内容，都要接受微信的管控

2. 说说微信小程序的生命周期函数有哪些？



2.1. 是什么

跟 `vue`、`react` 框架一样，微信小程序框架也存在生命周期，实质也是一堆会在特定时期执行的函数

小程序中，生命周期主要分成了三部分：

- 应用的生命周期
- 页面的生命周期
- 组件的生命周期

2.1.1. 应用的生命周期

小程序的生命周期函数是在 `app.js` 里面调用的，通过 `App(Object)` 函数用来注册一个小程序，指定其小程序的生命周期回调

2.1.2. 页面的生命周期

页面生命周期函数就是当你每进入/切换到一个新的页面的时候，就会调用的生命周期函数，同样通过 `App(Object)` 函数用来注册一个页面

2.1.3. 组件的生命周期

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，通过 `Component(Object)` 进行注册组件

2.2. 有哪些

2.2.1. 应用的生命周期

| 生命周期 | 说明 |
|------------------------|------------------------|
| onLaunch | 小程序初始化完成时触发，全局只触发一次 |
| onShow | 小程序启动，或从后台进入前台显示时触发 |
| onHide | 小程序从前台进入后台时触发 |
| onError | 小程序发生脚本错误或 API 调用报错时触发 |
| onPageNotFound | 小程序要打开的页面不存在时触发 |
| onUnhandledRejection() | 小程序有未处理的 Promise 拒绝时触发 |
| onThemeChange | 系统切换主题时触发 |

2.2.2. 页面的生命周期

| 生命周期 | 说明 | 作用 |
|--------|---------------|----------|
| onLoad | 生命周期回调—监听页面加载 | 发送请求获取数据 |
| onShow | 生命周期回调—监听页面显示 | 请求数据 |

| | | |
|----------|-------------------|-----------------|
| onReady | 生命周期回调—监听页面初次渲染完成 | 获取页面元素（少用） |
| onHide | 生命周期回调—监听页面隐藏 | 终止任务，如定时器或者播放音乐 |
| onUnload | 生命周期回调—监听页面卸载 | 终止任务 |

2.2.3. 组件的生命周期

| 生命周期 | 说明 |
|----------|-------------------|
| created | 生命周期回调—监听页面加载 |
| attached | 生命周期回调—监听页面显示 |
| ready | 生命周期回调—监听页面初次渲染完成 |
| moved | 生命周期回调—监听页面隐藏 |
| detached | 生命周期回调—监听页面卸载 |
| error | 每当组件方法抛出错误时执行 |

注意的是：

- 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，此时不能调用 setData
- 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

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，如下：

| 生命周期 | 说明 |
|------|---------------|
| show | 组件所在的页面被展示时执行 |
| hide | 组件所在的页面被隐藏时执行 |

代码如下：

```
JavaScript | 复制代码
1 Component({
2   pageLifetimes: {
3     show: function() {
4       // 页面被展示
5     },
6     hide: function() {
7       // 页面被隐藏
8     },
9   }
10 })
```

2.3. 执行过程

2.3.1. 应用的生命周期执行过程：

- 用户首次打开小程序，触发 onLaunch (全局只触发一次)
- 小程序初始化完成后，触发onShow方法，监听小程序显示
- 小程序从前台进入后台，触发 onHide方法
- 小程序从后台进入前台显示，触发 onShow方法
- 小程序后台运行一定时间，或系统资源占用过高，会被销毁

2.3.2. 页面生命周期的执行过程：

- 小程序注册完成后，加载页面，触发onLoad方法
- 页面载入后触发onShow方法，显示页面
- 首次显示页面，会触发onReady方法，渲染页面元素和样式，一个页面只会调用一次
- 当小程序后台运行或跳转到其他页面时，触发onHide方法
- 当小程序有后台进入到前台运行或重新进入页面时，触发onShow方法
- 当使用重定向方法 wx.redirectTo() 或关闭当前页返回上一页wx.navigateBack()，触发onUnload

当存在也应用生命周期和页面周期的时候，相关的执行顺序如下：

- 打开小程序：(App)onLaunch --> (App)onShow --> (Pages)onLoad --> (Pages)onShow --> (pages)onRead

- 进入下一个页面：(Pages)onHide --> (Next)onLoad --> (Next)onShow --> (Next)onReady
- 返回上一个页面：(curr)onUnload --> (pre)onShow
- 离开小程序：(App)onHide
- 再次进入：小程序未销毁 --> (App)onShow(执行上面的顺序)，小程序被销毁，(App)onLaunch重新开始执行。

3. 说说微信小程序的登录流程？



3.1. 背景

传统的 web 开发实现登陆功能，一般的做法是输入账号密码、或者输入手机号及短信验证码进行登录。服务端校验用户信息通过之后，下发一个代表登录态的 token 给客户端，以便进行后续的交互。每当 token 过期，用户都需要重新登录。

而在微信小程序中，可以通过微信官方提供的登录能力方便地获取微信提供的用户身份标识，快速建立小程序内的用户体系，从而实现登陆功能。

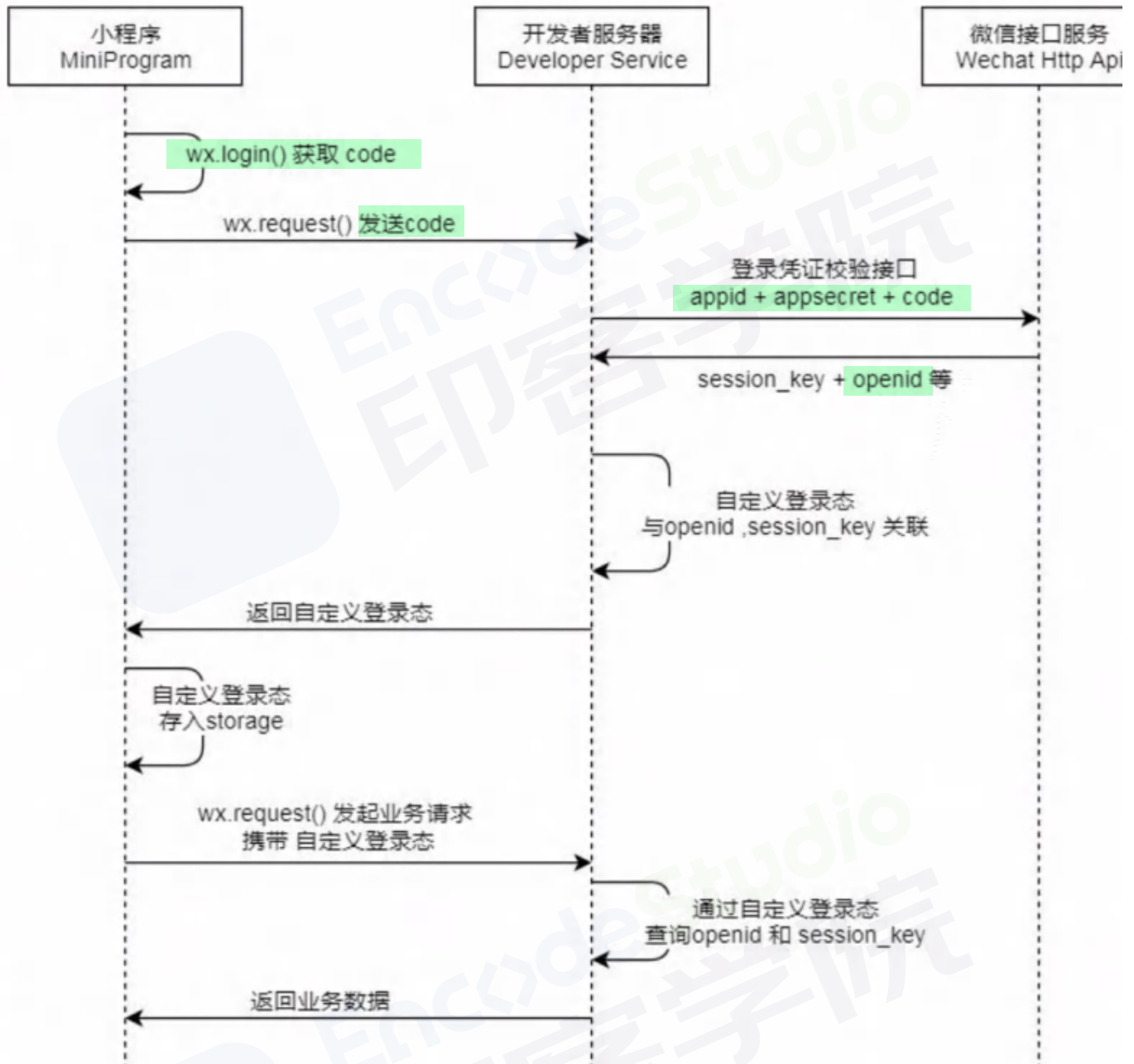
实现小程序用户体系主要涉及到 openid 和 code 的概念：

- 调用 wx.login() 方法会生成 code，将 code 作为参数传递给微信服务器指定接口，就可以获取用户的 openid。

对于每个小程序，微信都会将用户的微信 ID 映射出一个小程序 openid，作为这个用户在这个小程序的唯一标识。

3.2. 流程

微信小程序登陆具体实现的逻辑如下图所示：

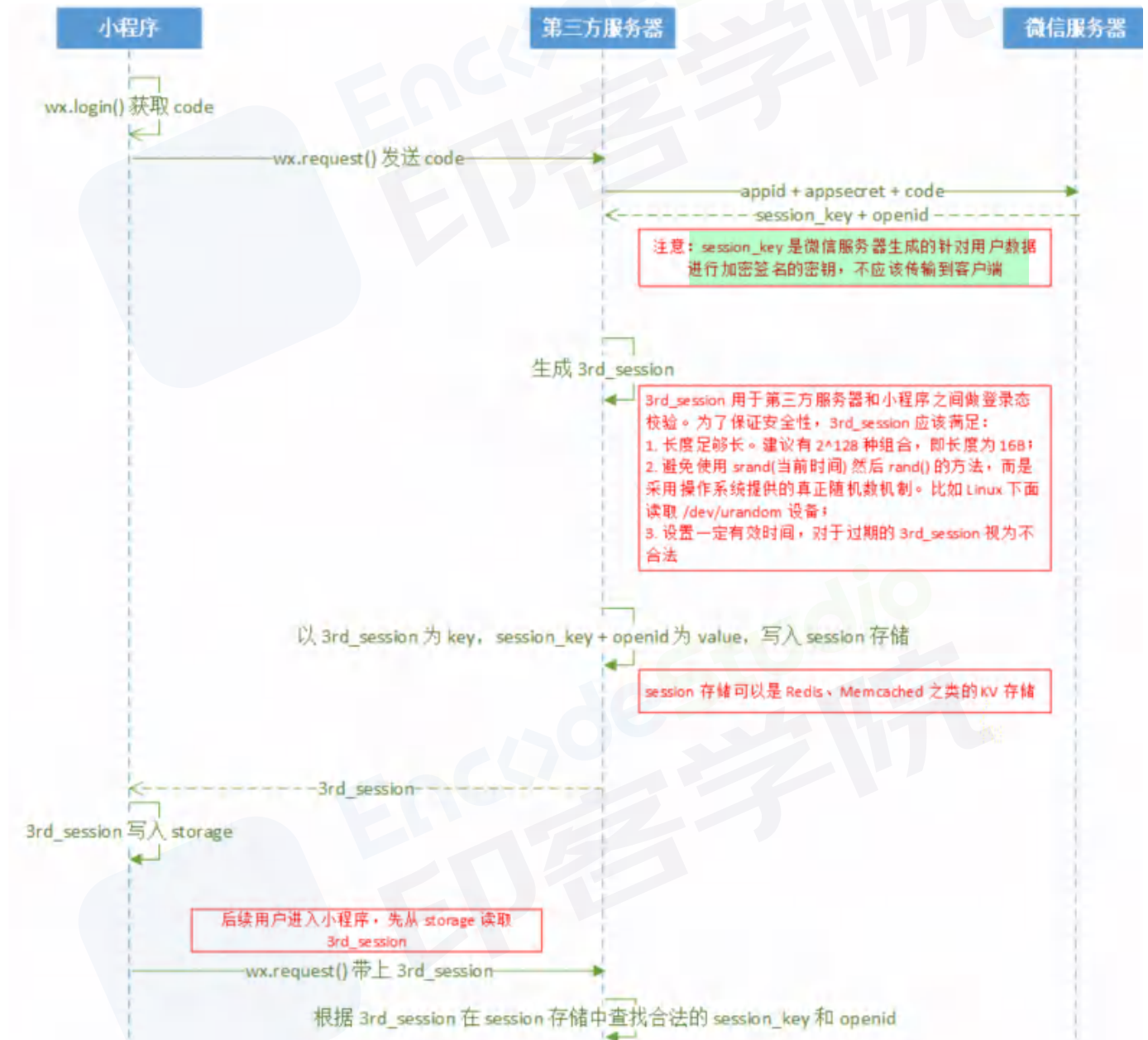


- 通过 wx.login() 获取到用户的code判断用户是否授权读取用户信息，调用wx.getUserInfo 读取用户数据
- 由于小程序后台授权域名无法授权微信的域名，所以需要自身后端调用微信服务器获取用户信息
- 通过 wx.request() 方法请求业务方服务器，后端把 appid , appsecret 和 code 一起发送到微信服务器。appid 和 appsecret 都是微信提供的，可以在管理员后台找到
- 微信服务器返回了 openid 及本次登录的会话密钥 session_key
- 后端从数据库中查找 openid ，如果没有查到记录，说明该用户没有注册，如果有记录，则继续往下走
- session_key 是对用户数据进行加密签名的密钥。为了自身应用安全，session_key 不应该在网络上

传输

- 然后生成 session 并返回给小程序
- 小程序把 session 存到 storage 里面
- 下次请求时，先从 storage 里面读取，然后带给服务端
- 服务端对比 session 对应的记录，然后校验有效期

更加详细的功能图如下所示：



3.3. 扩展

实际业务中，我们还需要登录态是否过期，通常的做法是在登录态（临时令牌）中保存有效期数据，该有效期数据应该在服务端校验登录态时和约定的时间（如服务端本地的系统时间或时间服务器上的标准

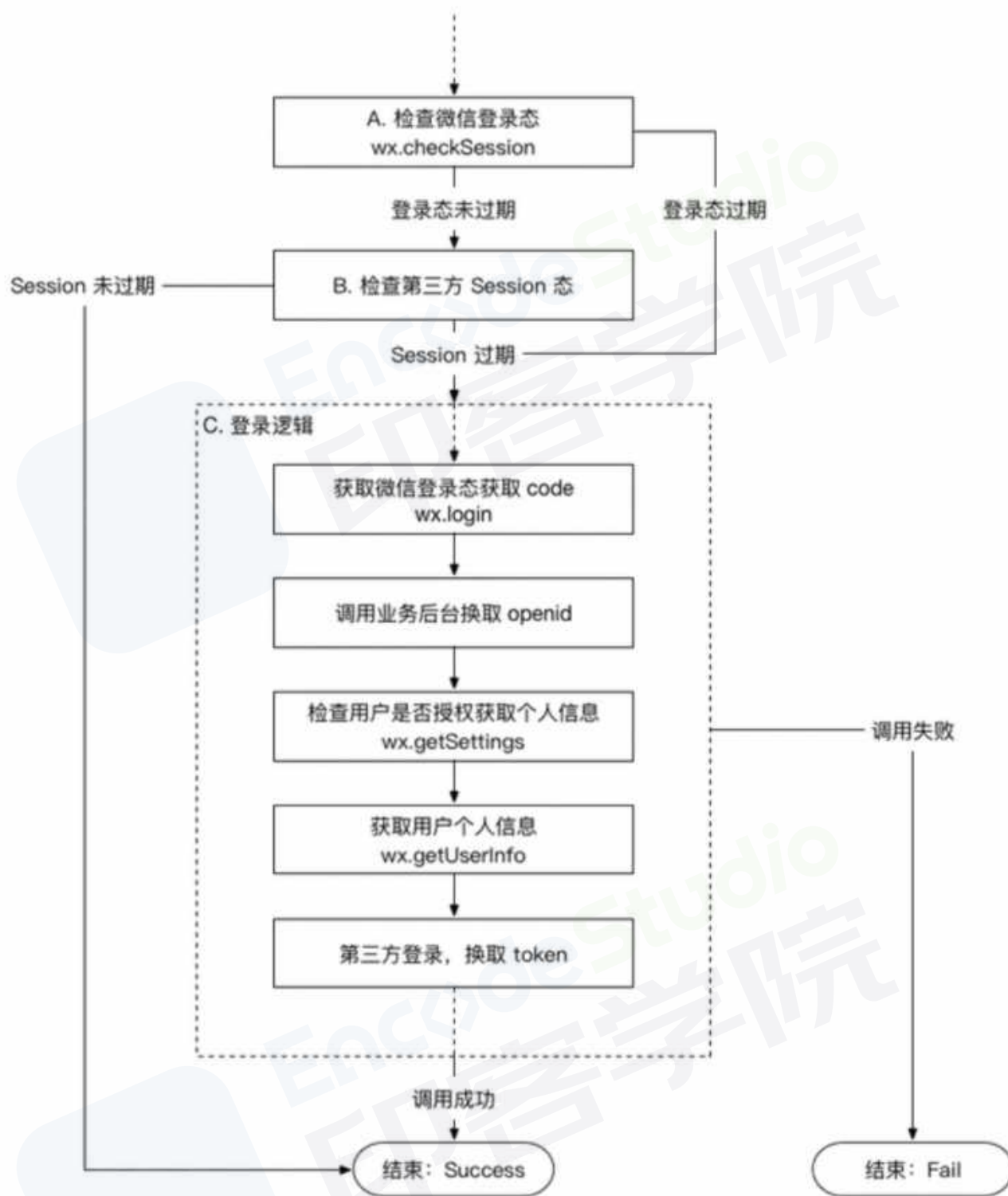
时间) 做对比

这种方法需要将本地存储的登录态发送到小程序的服务端，服务端判断为无效登录态时再返回需重新执行登录过程的消息给小程序

另一种方式可以通过调用 `wx.checkSession` 检查微信登陆态是否过期：

- 如果过期，则发起完整的登录流程
- 如果不过期，则继续使用本地保存的自定义登录态

这种方式的好处是不需要小程序服务端来参与校验，而是在小程序端调用API，流程如下所示：



4. 说说微信小程序中路由跳转的方式有哪些？区别？



4.1. 是什么

微信小程序拥有 `web` 网页和 `Application` 共同的特征，我们的页面都不是孤立存在的，而是通过和其他页面进行交互，来共同完成系统的功能

在微信小程序中，每个页面可以看成是一个 `pageModel`，`pageModel` 全部以栈的形式进行管理

4.2. 有哪些

常见的微信小程序页面跳转方式有如下：

- `wx.navigateTo(Object)`
- `wx.redirectTo(Object)`
- `wx.switchTab(Object)`
- `wx.navigateBack(Object)`
- `wx.reLaunch(Object)`

4.2.1. `wx.navigateTo(Object)`

`wx.navigateTo()` 用于保留当前页面、跳转到应用内的某个页面，使用 `wx.navigateBack` 可以返回到原页面

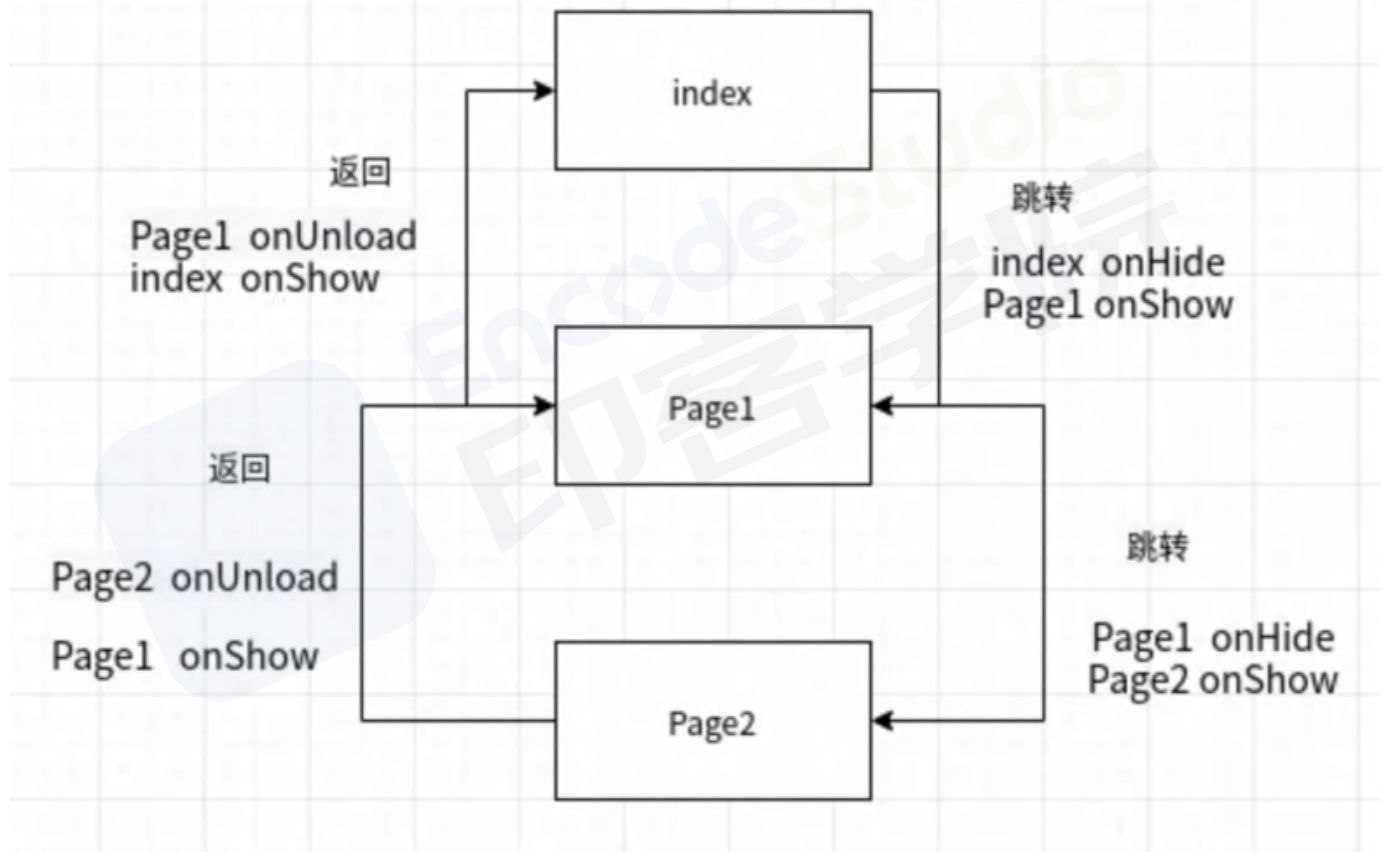
对于页面不是特别多的小程序，通常推荐使用 `wx.navigateTo` 进行跳转，以便返回原页面，以提高加载速度。当页面特别多时，则不推荐使用

参数表如下所示：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|----------|----------|-----|----|---|
| url | string | | 是 | 需要跳转的应用内非 tabBar 的页面的路径 (代码包路径), 路径后可以带参数。参数与路径之间使用 ? 分隔, 参数键与参数值用 = 相连, 不同参数用 & 分隔; 如 'path?key=value&key2=value2' |
| events | Object | | 否 | 页面间通信接口, 用于监听被打开页面发送到当前页面的数据。基础库 2.7.3 开始支持。 |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数 (调用成功、失败都会执行) |

流程图如下：

wx.navigateTo 跳转流程及状态



4.2.2. wx.redirectTo(Object)

重定向，当页面过多时，被保留页面会挤占微信分配给小程序的内存，或是达到微信所限制的 10 层页面栈的情况下，我们应该考虑选择 `wx.redirectTo`

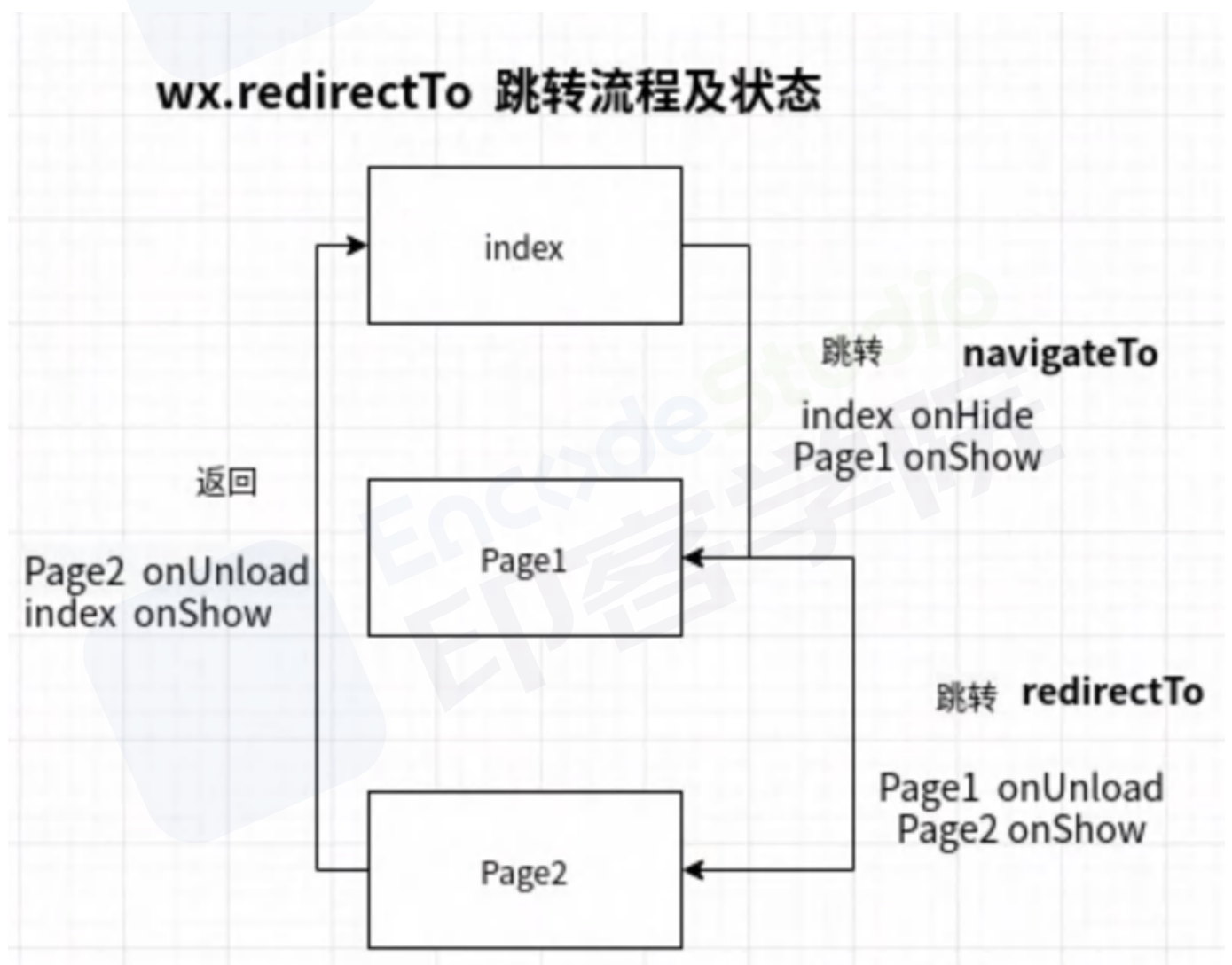
`wx.redirectTo()` 用于关闭当前页面，跳转到应用内的某个页面

这样的跳转，可以避免跳转前页面占据运行内存，但返回时页面需要重新加载，增加了返回页面的显示时间

参数表如下所示：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|----------|----------|-----|----|---|
| url | string | | 是 | 需要跳转的应用内非 tabBar 的页面的路径 (代码包路径), 路径后可以带参数。参数与路径之间使用 ? 分隔, 参数键与参数值用 = 相连, 不同参数用 & 分隔; 如 'path?key=value&key2=value2' |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数 (调用成功、失败都会执行) |

流程图如下所示：



4.2.3. wx.switchTab(Object)

跳转到 `tabBar` 页面，并关闭其他所有非 `tabBar` 页面

参数表如下所示：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|----------|----------|-----|----|---|
| url | string | | 是 | 需要跳转的 tabBar 页面的路径 (代码包路径) (需在 app.json 的 tabBar 字段定义的页面)，路径后不能带参数。 |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数 (调用成功、失败都会执行) |

4.2.4. wx.navigateBack(Object)

`wx.navigateBack()` 用于关闭当前页面，并返回上一页面或多级页面，开发者可通过 `getCurrentPages()` 获取当前的页面栈，决定需要返回几层则设置对象的 `delta` 属性即可

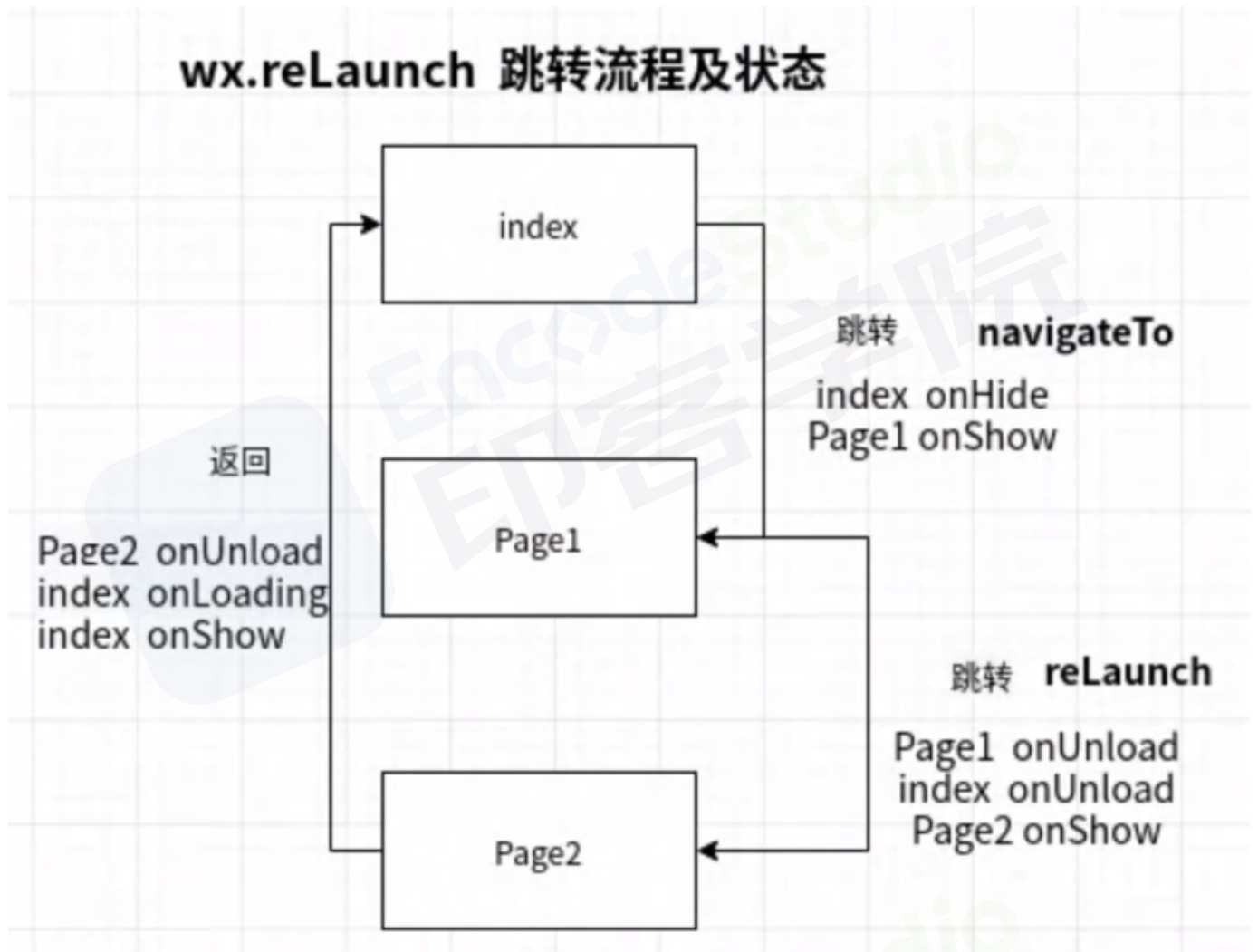
参数表如下：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|----------|----------|-----|----|---------------------------------|
| delta | number | 1 | 否 | 返回的页面数，如果 delta 大于现有页面数，则返回到首页。 |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数 (调用成功、失败都会执行) |

4.2.5. wx.reLaunch(Object)

关闭所有页面，打开到应用内的某个页面，返回的时候跳到首页

流程图如下所示：



参数表如下所示：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|----------|----------|-----|----|--|
| url | string | | 是 | 需要跳转的应用内页面路径 (代码包路径)，路径后可以带参数。参数与路径之间使用?分隔，参数键与参数值用=相连，不同参数用&分隔；如 'path?key=value&key2=value2' |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数（调用成功、失败都会执行） |

4.3. 总结

关于上述五种跳转方式，做下总结：

- navigateTo 保留当前页面，跳转到应用内的某个页面，使用 wx.navigateBack 可以返回到原页
- redirectTo 关闭当前页面，跳转到应用内的某个页面
- switchTab 跳转到 tabBar 页面，同时关闭其他非 tabBar 页面
- navigateBack 返回上一页面
- reLaunch 关闭所有页面，打开到应用内的某个页面

其中关于它们的页面栈的关系如下：

- navigateTo 新页面入栈
- redirectTo 当前页面出栈，新页面入栈
- navigateBack 页面不断出栈，直到目标返回页，新页面入栈
- switchTab 页面全部出栈，只留下新的 Tab 页面
- reLaunch 页面全部出栈，只留下新的页面

5. 说说微信小程序的发布流程？



5.1. 背景

在中大型的公司里，人员的分工非常仔细，一般会有不同岗位角色的员工同时参与同一个小程序项目。为此，小程序平台设计了不同的权限管理使得项目管理者可以更加高效管理整个团队的协同工作



以往我们在开发完网页之后，需要把网页的代码和资源放在服务器上，让用户通过互联网来访问

在小程序的平台里，开发者完成开发之后，需要在开发者工具提交小程序的代码包，然后在小程序后台发布小程序



5.2. 流程

关于发布的流程，主要分成了三个部分：

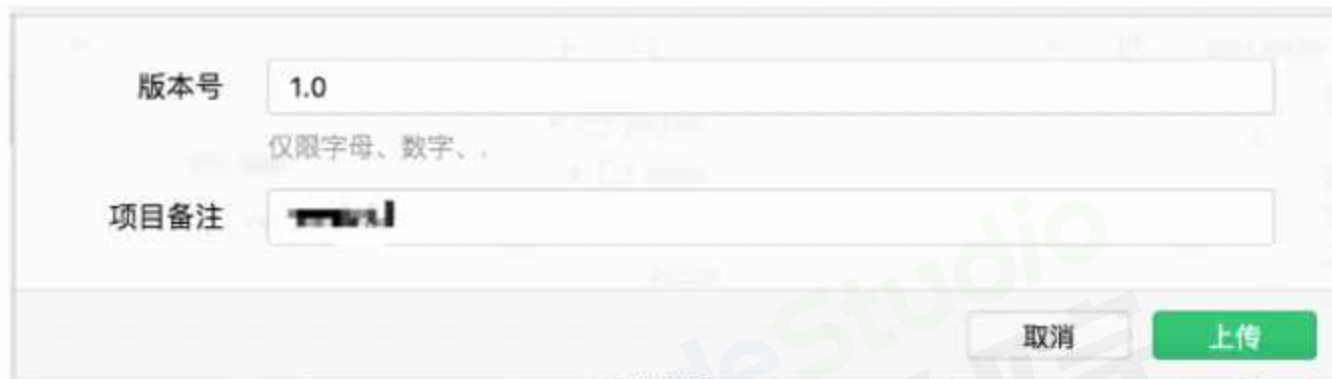
- 上传代码
- 提交审核
- 发布版本

5.2.1. 上传代码

在开发者工具中，可以点击代码上传功能：



然后就可以填写版本信息：



版本号 1.0

仅限字母、数字、.

项目备注

取消 上传

然后点击上传，编译器则会提示上传代码成功

5.2.2. 提交审核

代码上传完毕，就可以登陆微信公众号的官网首页，点击【开发管理】，查看应用详情：

首页

开发管理

用户身份

数据分析

模板消息

客服消息

附近门店

微信支付

设置

开发管理

线上版本

| | | | | |
|--------|------|----------------------|----|---|
| 版本号 | 发布者 | M | 详情 | ▼ |
| v0.9.2 | 发布时间 | 2017-02-13 09:59:00 | | |
| | 描述 | 收藏; 会员页面配置; 会员头部背景图; | | |

审核版本

| | | | | |
|--------|--------|----------------------|--|--|
| 版本号 | 开发者 | M | | |
| v0.9.2 | 提交审核时间 | 2017-02-13 09:59:00 | | |
| 线上版本 | 描述 | 收藏; 会员页面配置; 会员头部背景图; | | |

开发版本

| | | | | |
|--------|------|---------------------|------|---|
| 版本号 | 开发者 | 黄赐印 | 提交审核 | ▼ |
| V0.9.0 | 提交时间 | 2017-01-14 03:51:00 | | |
| | 描述 | 增加分享, 客服, 直接注册 | | |

| | | | | |
|--------|------|----------------------|------|---|
| 版本号 | 开发者 | M | 提交审核 | ▼ |
| v0.9.2 | 提交时间 | 2017-02-08 11:34:00 | | |
| 待审核 | 描述 | 收藏; 会员页面配置; 会员头部背景图; | | |

| | | | | |
|-------|------|---------------------|------|---|
| 版本号 | 开发者 | 韦先敏 | 提交审核 | ▼ |
| 0.9.3 | 提交时间 | 2017-02-13 14:43:00 | | |
| | 描述 | 更新首页布局, 线上发布由实施部门发布 | | |

提交审核过程需要填写审核信息，如下图：

提交审核

配置功能页面 至少填写一组，填写正确的信息有利于用户快速搜索出你的小程序

功能页面1

功能页面

请选择

标题

0/32

所在服务类目

请选择

功能页面和服务类目必须一一对应，且功能页面提供的内容必须符合该类目范围

标签

标签用回车分开，填写与页面功能相关的标签，更易于被搜索

+

 添加功能页面

提交审核

提交审核成功之后如下图：

微信公众平台 | 小程序

文档 社区

首页

开发管理

用户身份

数据分析

模板消息

客服消息

附近门店

微信支付

设置

开发管理

线上版本

| | | | | |
|--------|------|----------------------|----|---|
| 版本号 | 发布者 | M | 详情 | ✓ |
| v0.9.2 | 发布时间 | 2017-02-13 09:59:00 | | |
| | 描述 | 收藏; 会员页面配置; 会员头部背景图; | | |

审核版本

| | | | |
|-------|--------|---------------------|---|
| 版本号 | 开发者 | 韦先敬 | ✓ |
| 0.9.3 | 提交审核时间 | 2017-02-13 14:56:00 | |
| 审核中 | 描述 | 更新首页布局，线上发布由实施部门发布 | |

5.2.3. 发布版本

当审核通过之后，即可提交发布



发布成功之后则如下:



5.3. 扩展

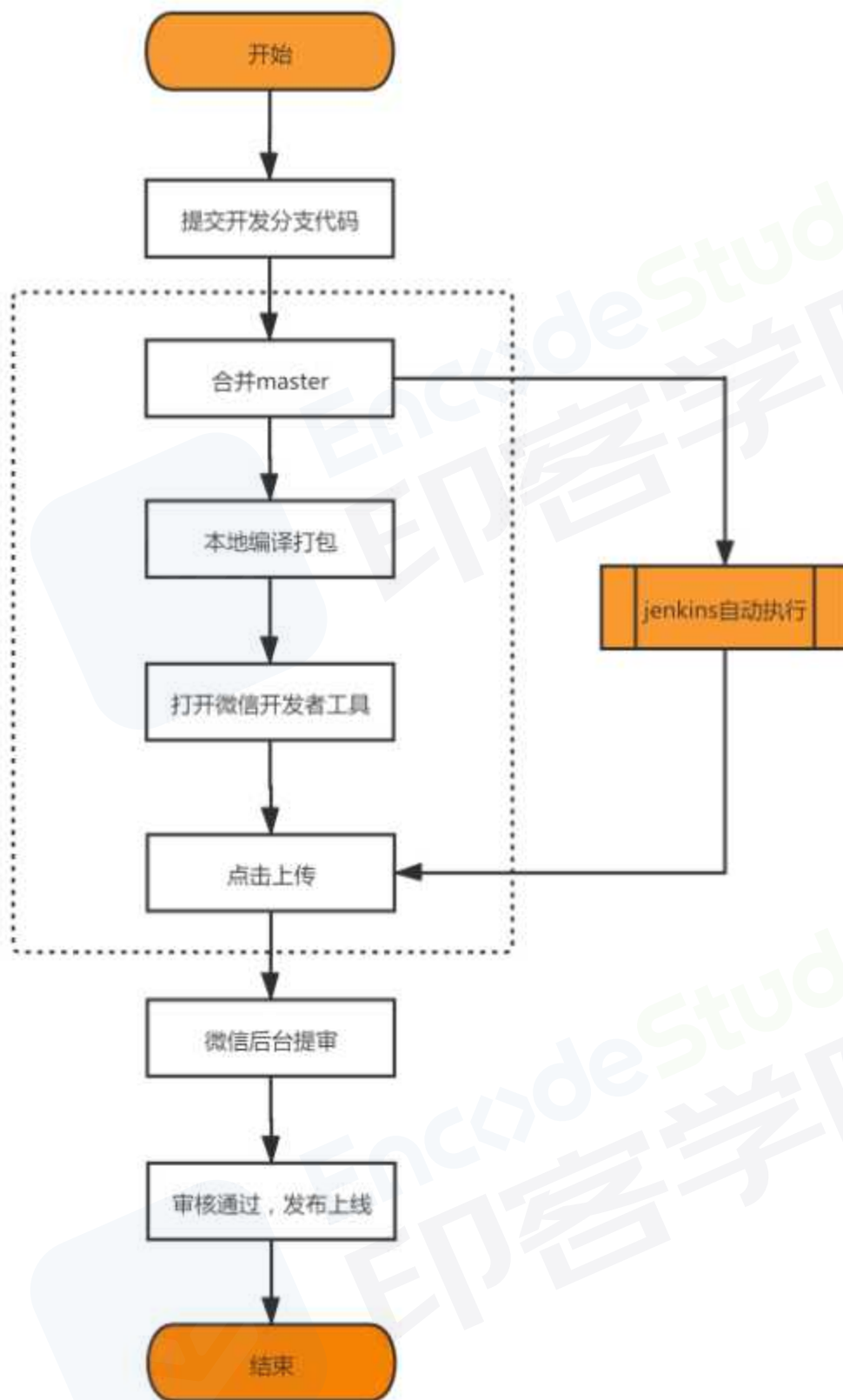
上述是最简单的小程序代码发布的流程，通常的流程如下：

- 代码管理服务上新建分支
- 开发测试新需求
- 测试完成后，将本地分支合并到 master 分支
- 拉取 master 分支最新代码，执行 build 命令生成小程序可执行文件

- 开发者工具点击“上传”
- 提审
- 发布

但是面对多人协调开发的时候，有可能出现已经上线的代码还没合并到 `master` 的情况

因此可以考虑自动化构建部署，就是将从开发到部署的一系列流程变成自动化，衔接连贯，在构建失败时能够告知开发者，构建成功后能够告知测试和实施人员，可参考如下流程图：



6. 说说微信小程序的支付流程？

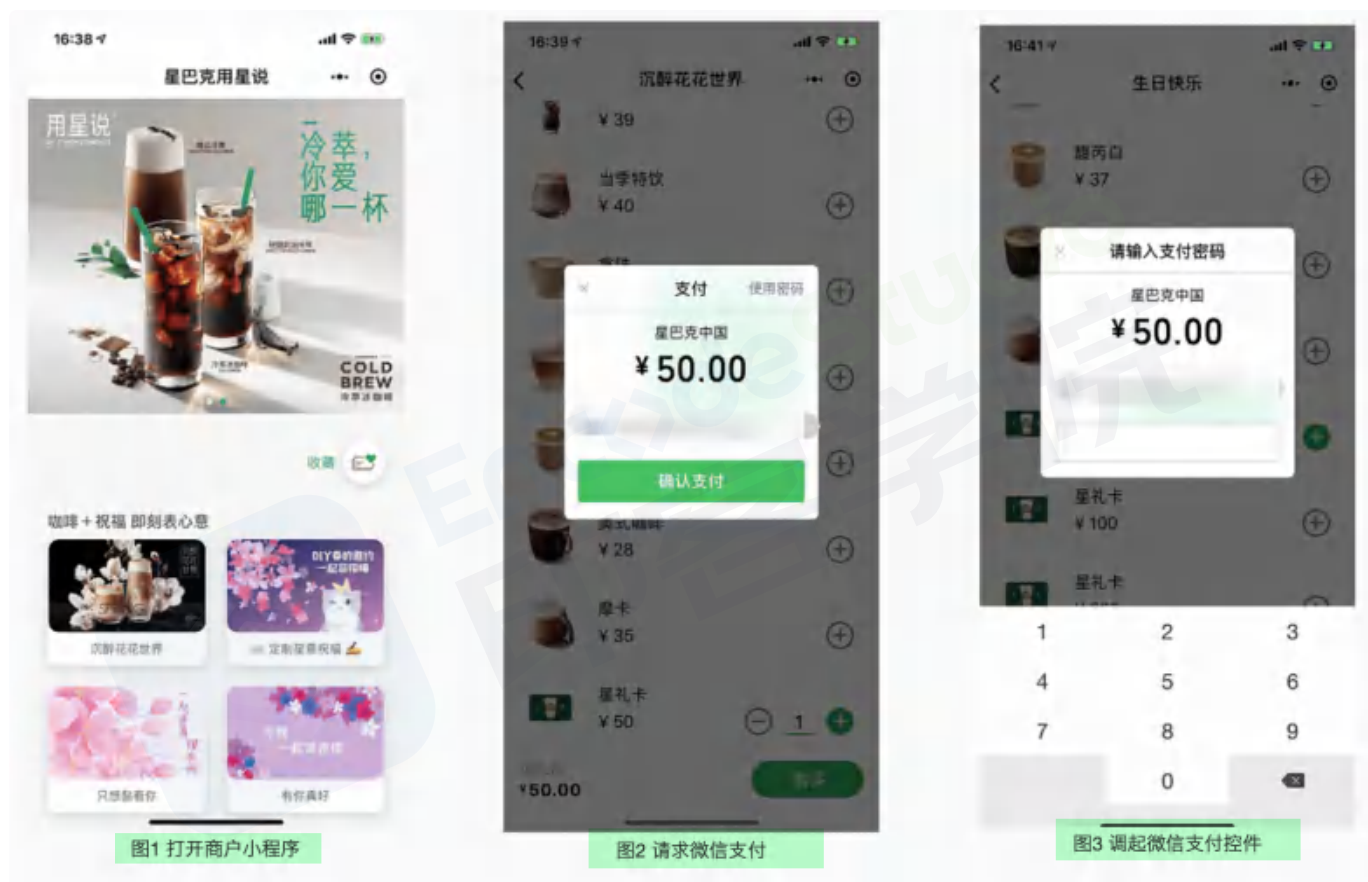


6.1. 前言

微信小程序为电商类小程序，提供了非常完善、优秀、安全的支付功能

在小程序内可调用微信的 API 完成支付功能，方便、快捷

场景如下图所示：



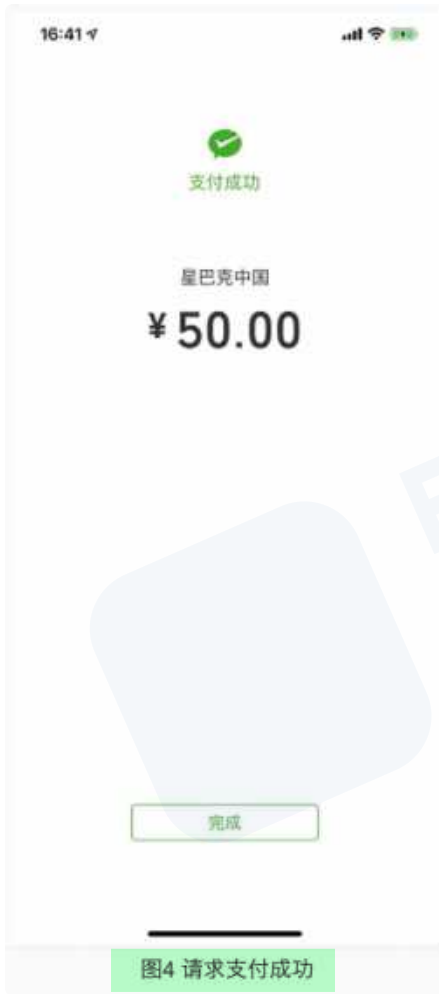


图4 请求支付成功



图5 返回商户小程序



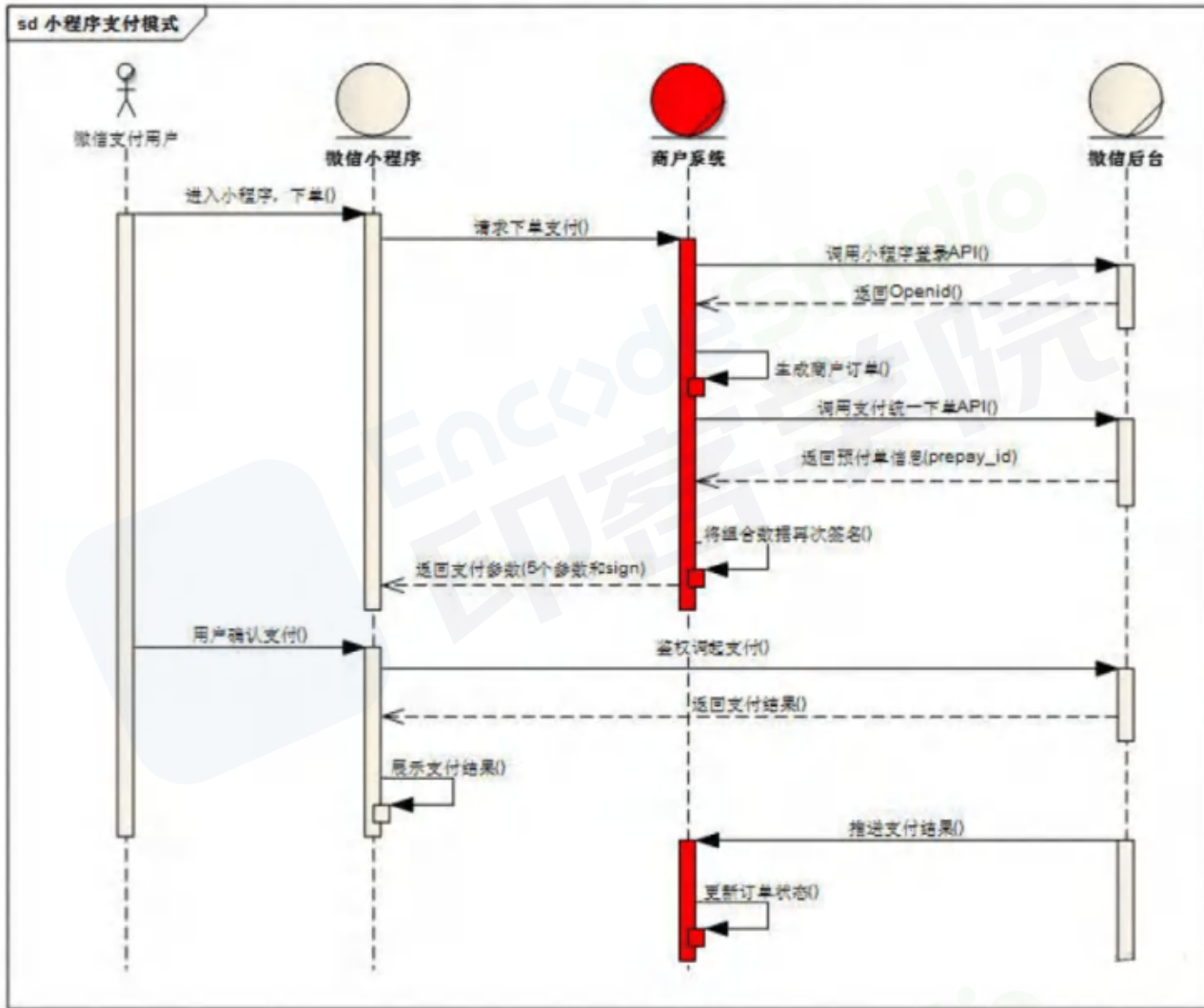
图6 下发支付凭证

- 用户通过分享或扫描二维码进入商户小程序，用户选择购买，完成选购流程
- 调起微信支付控件，用户开始输入支付密码
- 密码验证通过，支付成功。商户后台得到支付成功的通知
- 返回商户小程序，显示购买成功
- 微信支付公众号下发支付凭证

6.2. 流程

以电商小程序为例

支付流程图如下所示：



具体的做法：

- 打开某小程序，点击直接下单
- wx.login获取用户临时登录凭证code，发送到后端服务器换取openid
- 在下单时，小程序需要将购买的商品Id，商品数量，以及用户的openid传送到服务器
- 服务器在接收到商品Id、商品数量、openid后，生成服务期订单数据，同时经过一定的签名算法，向微信支付发送请求，获取预付单信息(prepay_id)，同时将获取的数据再次进行相应规则的签名，向小程序端响应必要的信息
- 小程序端在获取对应的参数后，调用wx.requestPayment()发起微信支付，唤醒支付工作台，进行支付
- 接下来的一些列操作都是由用户来操作的包括了微信支付密码，指纹等验证，确认支付之后执行鉴权调起支付
- 鉴权调起支付：在微信后台进行鉴权，微信后台直接返回给前端支付的结果，前端收到返回数据后对支付结果进行展示
- 推送支付结果：微信后台在给前端返回支付的结果后，也会向后台也返回一个支付结果，后台通过

这个支付结果来更新订单的状态

其中后端响应数据必要的信息则是 `wx.requestPayment` 方法所需要的参数，大致如下：

JavaScript | 复制代码

```
1  wx.requestPayment({  
2    // 时间戳  
3    timeStamp: '',  
4    // 随机字符串  
5    nonceStr: '',  
6    // 统一下单接口返回的 prepay_id 参数值  
7    package: '',  
8    // 签名类型  
9    signType: '',  
10   // 签名  
11   paySign: '',  
12   // 调用成功回调  
13   success () {},  
14   // 失败回调  
15   fail () {},  
16   // 接口调用结束回调  
17   complete () {}  
18 })
```

参数表如下所示：

| 属性 | 类型 | 默认值 | 必填 | 说明 |
|-----------|----------|-----|----|---|
| timeStamp | string | | 是 | 时间戳，从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间 |
| nonceStr | string | | 是 | 随机字符串，长度为32个字符以下 |
| package | string | | 是 | 统一下单接口返回的 prepay_id 参数值，提交格式如：prepay_id=*** |
| signType | string | MD5 | 否 | 签名算法，应与后台下单时的值一致 |
| paySign | string | | 是 | 签名，具体见微信支付文档 |
| success | function | | 否 | 接口调用成功的回调函数 |
| fail | function | | 否 | 接口调用失败的回调函数 |
| complete | function | | 否 | 接口调用结束的回调函数（调用成功、失败都会执行） |

6.3. 结束

小程序支付和以往的网页、APP微信支付大同小异，可以说小程序的支付变得更加简洁，不需要设置支付目录、域名授权等操作

7. 说说微信小程序的实现原理？



7.1. 背景

网页开发，渲染线程和脚本是互斥的，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的原因，本质就是我们常说的 JS 是单线程的

而在小程序中，选择了 Hybrid 的渲染方式，将视图层和逻辑层是分开的，双线程同时运行，视图层的界面使用 WebView 进行渲染，逻辑层运行在 JSCore 中

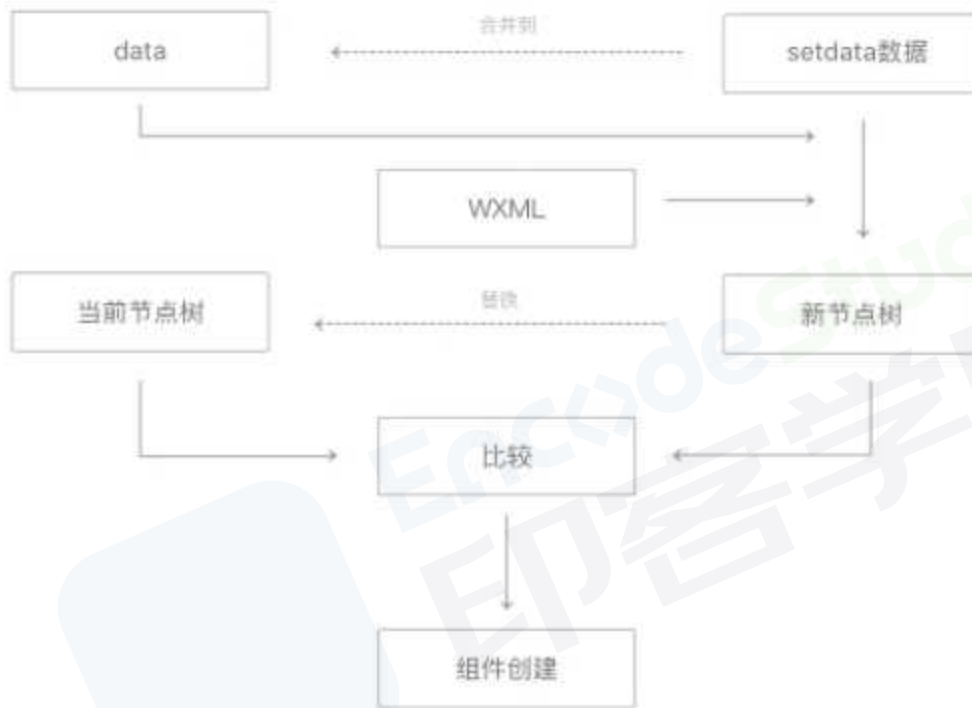


- 渲染层：界面渲染相关的任务全都在 WebView 线程里执行。一个小程序存在多个界面，所以渲染层存在多个 WebView 线程
- 逻辑层：采用 JsCore 线程运行 JS 脚本，在这个环境下执行的都是有关小程序业务逻辑的代码

7.2. 通信

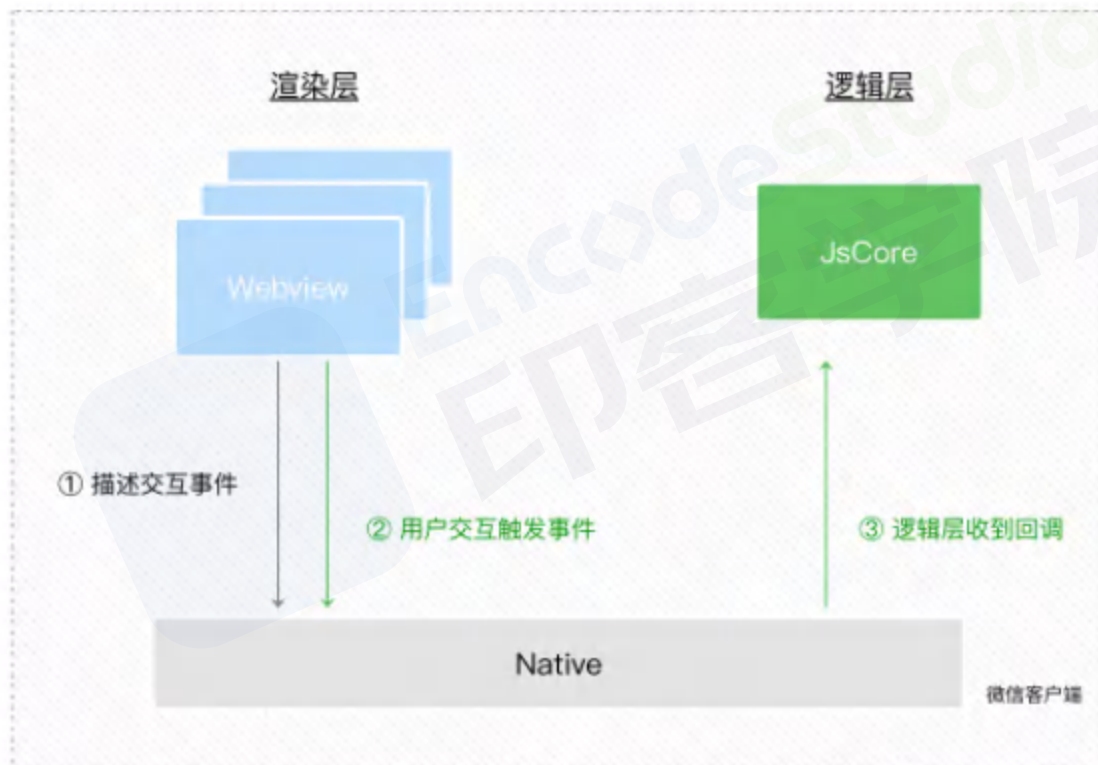
小程序在渲染层，宿主环境会把 `wxml` 转化成对应的 `JS` 对象

在逻辑层发生数据变更的时候，通过宿主环境提供的 `setData` 方法把数据从逻辑层传递到渲染层，再经过对比前后差异，把差异应用在原来的 `Dom` 树上，渲染出正确的视图



当视图存在交互的时候，例如用户点击你界面上某个按钮，这类反馈应该通知给开发者的逻辑层，需要将对应的处理状态呈现给用户

对于事件的分发处理，微信进行了特殊的处理，将所有的事件拦截后，丢到逻辑层交给 `JavaScript` 进行处理



由于小程序是基于双线程的，也就是任何在视图层和逻辑层之间的数据传递都是线程间的通信，会有一些的延时，因此在小程序中，页面更新成了异步操作

异步会使得各部分的运行时序变得复杂一些，比如在渲染首屏的时候，逻辑层与渲染层会同时开始初始化工作，但是渲染层需要有逻辑层的数据才能把界面渲染出来

如果渲染层初始化工作较快完成，就要等逻辑层的指令才能进行下一步工作

因此逻辑层与渲染层需要有一定的机制保证时序正确，在每个小程序页面的生命周期中，存在着若干次页面数据通信



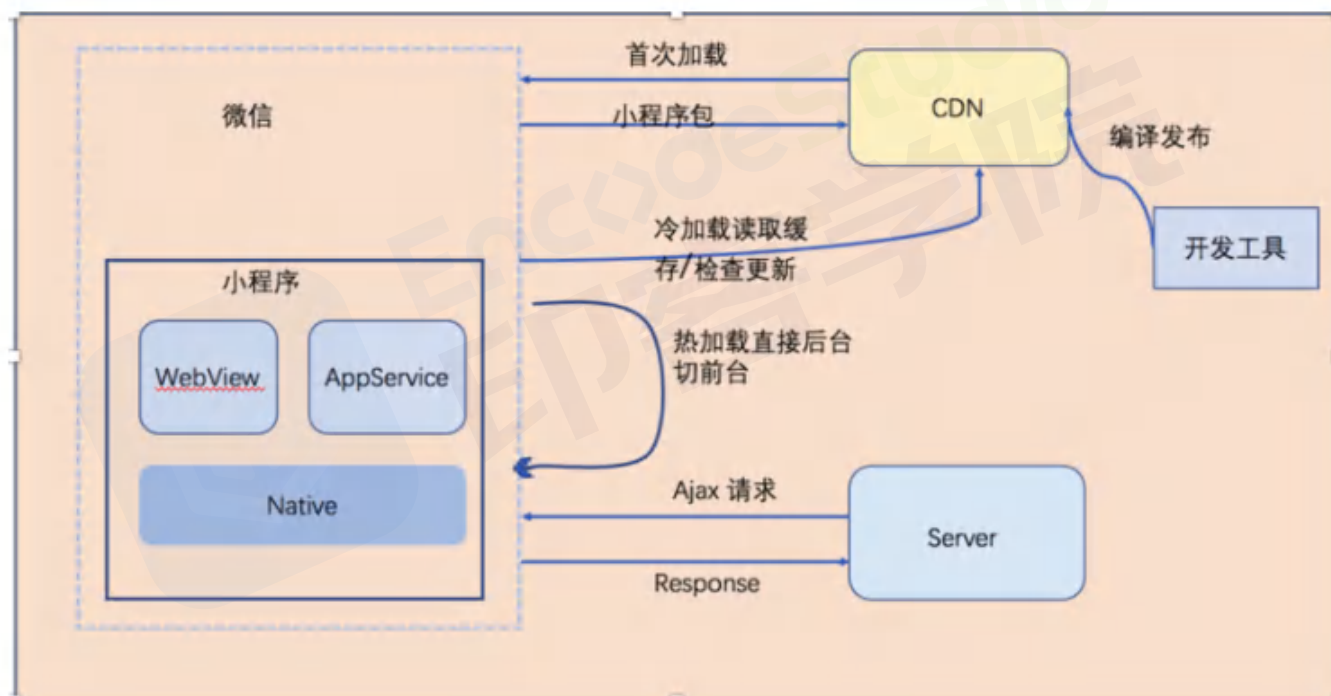
7.3. 运行机制

小程序启动运行两种情况：

- 冷启动（重新开始）：用户首次打开或者小程序被微信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，即为冷启动
- 热启动：用户已经打开过小程序，然后在一定时间内再次打开该小程序，此时无需重新启动，只需要将后台态的小程序切换到前台，这个过程就是热启动

7.3.1. 需要注意

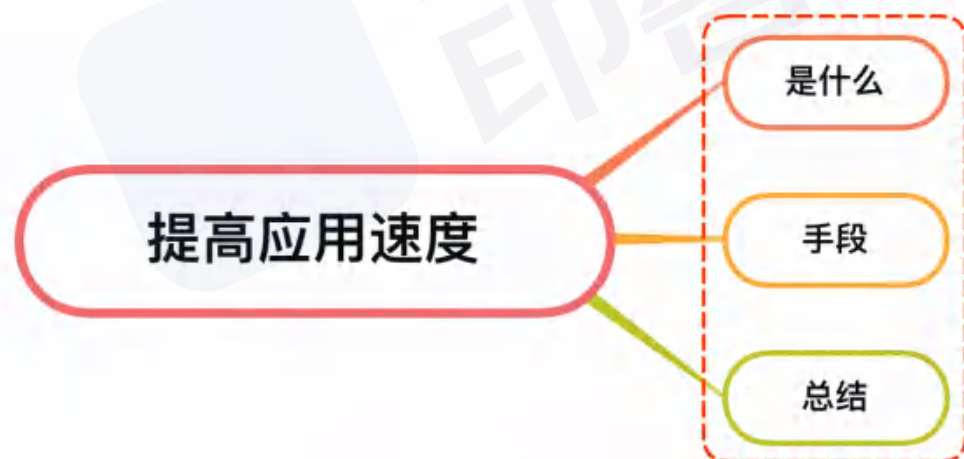
1. 小程序没有重启的概念
2. 当小程序进入后台，客户端会维持一段时间的运行状态，超过一定时间后会被微信主动销毁
3. 短时间内收到系统两次以上内存警告，也会对小程序进行销毁，这也就为什么一旦页面内存溢出，页面会奔溃的本质原因了



开发者在后台发布新版本之后，无法立刻影响到所有现网用户，但最差情况下，也在发布之后 24 小时之内下发新版本信息到用户

每次冷启动时，都会检查是否有更新版本，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

8. 说说提高微信小程序的应用速度的手段有哪些？



8.1. 是什么

小程序启动会常常遇到如下图场景：



这是因为，小程序首次启动前，微信会在小程序启动前为小程序准备好通用的运行环境，如运行中的线程和一些基础库的初始化

然后才开始进入启动状态，展示一个固定的启动界面，界面内包含小程序的图标、名称和加载提示图标。此时，微信会在背后完成几项工作：

- 下载小程序代码包
- 加载小程序代码包
- 初始化小程序首页

下载到的小程序代码包不是小程序的源代码，而是编译、压缩、打包之后的代码包

整体流程如下图：



8.2. 手段

围绕上图小程序的启动流程，我们可以从加载、渲染两个纬度进行切入：

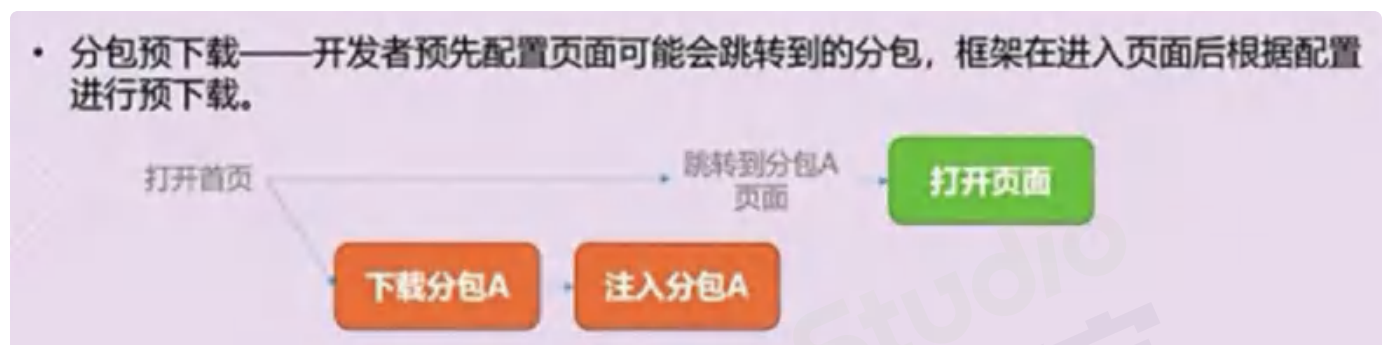
8.2.1. 加载

提升体验最直接的方法是控制小程序包的大小，常见手段有如下：

- 代码包的体积压缩可以通过勾选开发者工具中“上传代码时，压缩代码”选项
- 及时清理无用的代码和资源文件
- 减少资源包中的图片等资源的数量和大小（理论上除了小icon，其他图片资源从网络下载），图片资源压缩率有限

并且可以采取分包加载的操作，将用户访问率高的页面放在主包里，将访问率低的页面放入子包里，按需加载

当用户点击到子包的目录时，还是有一个代码包下载的过程，这会感觉到明显的卡顿，所以子包也不建议拆的太大，当然我们可以采用子包预加载技术，并不需要等到用户点击到子包页面后在下载子包



8.2.2. 渲染

关于微信小程序首屏渲染优化的手段如下：

- 请求可以在页面onLoad就加载，不需要等页面ready后在异步请求数据
- 尽量减少不必要的https请求，可使用 `getStorageSync()` 及 `setStorageSync()` 方法将数据存储在本地
- 可以在前置页面将一些有用的字段带到当前页，进行首次渲染（列表页的某些数据--> 详情页），没有数据的模块可以进行骨架屏的占位

在微信小程序中，提高页面的多次渲染效率主要在于正确使用 `setData`：

- 不要过于频繁调用`setData`，应考虑将多次`setData`合并成一次`setData`调用
- 数据通信的性能与数据量正相关，因而如果有一些数据字段不在界面中展示且数据结构比较复杂或

包含长字符串，则不应使用 `setData` 来设置这些数据

- 与界面渲染无关的数据最好不要设置在data中，可以考虑设置在page对象的其他字段下

除此之外，对于一些独立的模块我们尽可能抽离出来，这是因为自定义组件的更新并不会影响页面上其他元素的更新

各个组件也将具有各自独立的逻辑空间。每个组件都分别拥有自己的独立的数据、`setData` 调用

8.3. 总结

小程序启动加载性能：

- 控制代码包的大小
- 分包加载
- 首屏体验（预请求，利用缓存，避免白屏，及时反馈

小程序渲染性能：

- 避免不当的使用setData
- 使用自定义组件