# 3.2 程序与页面

从逻辑组成来说，一个小程序是由多个“页面”组成的“程序”。这里要区别一下“小程序”和“程序”的概念，往往我们需要在“程序”启动或者退出的时候存储数据或者在“页面”显示或者隐藏的时候做一些逻辑处理，了解程序和页面的概念以及它们的生命周期是非常重要的。

## 3.2.1 程序

“小程序”指的是产品层面的程序，而“程序”指的是代码层面的程序实例，为了避免误解，下文采用 App 来代替代码层面的“程序”概念。

## 1. 程序构造器 App()

宿主环境提供了 App() 构造器用来注册一个程序 App，需要留意的是 App() 构造器必须写在项目根目录的 app.js 里，App 实例是单例对象，在其他 JS 脚本中可以使用宿主环境提供的 getApp() 来获取程序实例。

代码清单 3-3 getApp() 获取 App 实例

```
// other.js
var appInstance = getApp()
```

App() 的调用方式如代码清单 3-4 所示，App 构造器接受一个 Object 参数，参数说明如表 3-1 所示，其中 onLaunch / onShow / onHide 三个回调是 App 实例的生命周期函数，我们会在后文展开；onError 我们暂时不在本章展开，我们会在第 8 章里详细讨论；App 的其他参数我们也放在后文进行展开。

代码清单 3-4 App 构造器

```
App({
  onLaunch: function(options) {},
  onShow: function(options) {},
  onHide: function() {},
  onError: function(msg) {},
  globalData: 'I am global data'
})
```

表 3-1 App 构造器的参数

| 参数属性 | 类型 | 描述 |
| --- | --- | --- |
| onLaunch | Function | 当小程序初始化完成时，会触发 onLaunch（全局只触发一次） |
| onShow | Function | 当小程序启动，或从后台进入前台显示，会触发 onShow |
| onHide | Function | 当小程序从前台进入后台，会触发 onHide |
| onError | Function | 当小程序发生脚本错误，或者 API 调用失败时，会触发 onError 并带上错误信息 |
| 其他字段 | 任意 | 可以添加任意的函数或数据到 Object 参数中，在 App 实例回调用 this 可以访问 |

# 2. 程序的生命周期和打开场景

初次进入小程序的时候，微信客户端初始化好宿主环境，同时从网络下载或者从本地缓存中拿到小程序的代码包，把它注入到宿主环境，初始化完毕后，微信客户端就会给 App 实例派发 onLaunch 事件，App 构造器参数所定义的 onLaunch 方法会被调用。

进入小程序之后，用户可以点击右上角的关闭，或者按手机设备的 Home 键离开小程序，此时小程序并没有被直接销毁，我们把这种情况称为“小程序进入后台状态”，App 构造器参数所定义的 onHide 方法会被调用。

当再次回到微信或者再次打开小程序时，微信客户端会把“后台”的小程序唤醒，我们把这种情况称为“小程序进入前台状态”，App 构造器参数所定义的 onShow 方法会被调用。

我们可以看到，App 的生命周期是由微信客户端根据用户操作主动触发的。为了避免程序上的混乱，我们不应该从其他代码里主动调用 App 实例的生命周期函数。

在微信客户端中打开小程序有很多途径：从群聊会话里打开，从小程序列表中打开，通过微信扫一扫二维码打开，从另外一个小程序打开当前小程序等，针对不同途径的打开方式，小程序有时需要做不同的业务处理，所以微信客户端会把打开方式带给 onLaunch 和 onShow 的调用参数 options，示例代码以及详细参数如代码清单 3-5 和表 3-2 所示。需要留意小程序的宿主环境在迭代更新过程会增加不少打开场景，因此要获取最新的场景值说明请查看官方文档：https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/app.html。

代码清单 3-5 onLaunch 和 onShow 带参数

```
App({
  onLaunch: function(options) { console.log(options) },
  onShow: function(options) { console.log(options) }
})
```

表 3-2 onLaunch,onShow 参数

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| path | String | 打开小程序的页面路径 |
| query | Object | 打开小程序的页面参数 query |
| scene | Number | 打开小程序的场景值，详细场景值请参考小程序官方文档 |
| shareTicket | String | shareTicket，详见小程序官方文档 |
| referrerInfo | Object | 当场景为由从另一个小程序或公众号或 App 打开时，返回此字段 |
| referrerInfo.appId | String | 来源小程序或公众号或 App 的 appId，详见下方说明 |
| referrerInfo.extraData | Object | 来源小程序传过来的数据，scene=1037 或 1038 时支持 |

表 3-3 以下场景支持返回 referrerInfo.appId

| 场景值 | 场景 | appId 信息含义 |
| --- | --- | --- |
| 1020 | 公众号 profile | 页相关小程序列表 返回来源公众号 appId |
| 1035 | 公众号自定义菜单 | 返回来源公众号 appId |
| 1036 | App 分享消息卡片 | 返回来源应用 appId |
| 1037 | 小程序打开小程序 | 返回来源小程序 appId |

| 场景值 | 场景 | appId 信息含义 |
| --- | --- | --- |
| 1038 | 从另一个小程序返回 | 返回来源小程序 appId |
| 1043 | 公众号模板消息 | 返回来源公众号 appId |

# 3. 小程序全局数据

我们在 3.1.2 节说到小程序的 JS 脚本是运行在 JsCore 的线程里，小程序的每个页面各自有一个 WebView 线程进行渲染，所以小程序切换页面时，小程序逻辑层的 JS 脚本运行上下文依旧在同一个 JsCore 线程中。

在上文中说道 App 实例是单例的，因此不同页面直接可以通过 App 实例下的属性来共享数据。App 构造器可以传递其他参数作为全局属性以达到全局共享数据的目的。

代码清单 3-6 小程序全局共享数据

```
// app.js
App({
  globalData: 'I am global data' // 全局共享数据
})
// 其他页面脚本 other.js
var appInstance = getApp()
console.log(appInstance.globalData) // 输出: I am global data
```

与此同时，我们要特别留意一点，所有页面的脚本逻辑都跑在同一个 JsCore 线程，页面使用 setTimeout 或者 setInterval 的定时器，然后跳转到其他页面时，这些定时器并没有被清除，需要开发者自己在页面离开的时候进行清理。

## 3.2.2 页面

一个小程序可以有很多页面，每个页面承载不同的功能，页面之间可以互相跳转。为了叙述简洁，我们之后讨论所涉及的"页面"概念特指"小程序页面"。

# 1. 文件构成和路径

一个页面是分三部分组成：界面、配置和逻辑。界面由 WXML 文件和 WXSS 文件来负责描述，配置由 JSON 文件进行描述，页面逻辑则是由 JS 脚本文件负责。一个页面的文件需要放置在同一个目录下，其中 WXML 文件和 JS 文件是必须存在的，JSON 和 WXSS 文件是可选的。

页面路径需要在小程序代码根目录 app.json 中的 pages 字段声明，否则这个页面不会被注册到宿主环境中。例如两个页面的文件的相对路径分别为 pages/index/page. *和 pages/other/other.(表示 wxml/wxss/json/js 四个文件)，在 app.json 的 pages 字段的代码路径需要去除*.后缀，如代码清单 3-7 所示，默认 pages 字段的第一个页面路径为小程序的首页。

代码清单 3-7 app.json 声明页面路径

```
{
  "pages":[
    "pages/index/page", // 第一项默认为首页
```

```
    "pages/other/other"
  ]
}
```

为了叙述方便，下文使用 page.wxml / page.wxss / page.json / page.js 来分别代表特定页面的 4 个文件。

## 2. 页面构造器 Page()

宿主环境提供了 Page() 构造器用来注册一个小程序页面，Page()在页面脚本 page.js 中调用，Page() 的调用方式如代码清单 3-8 所示。Page 构造器接受一个 Object 参数，参数说明如表 3-4 所示，其中 data 属性是当前页面 WXML 模板中可以用来做数据绑定的初始数据，我们会在后文展开讨论；onLoad / onReady / onShow / onHide /onUnload 5 个回调是 Page 实例的生命周期函数，我们在后文展开；onPullDownRefresh / onReachBottom / onShareAppMessage / onPageScroll 4 个回调是页面的用户行为，我们也会在后文展开。

代码清单 3-8 Page 构造器

```
Page({
  data: { text: "This is page data." },
  onLoad: function(options) { },
  onReady: function() { },
  onShow: function() { },
  onHide: function() { },
  onUnload: function() { },
  onPullDownRefresh: function() { },
  onReachBottom: function() { },
  onShareAppMessage: function () { },
  onPageScroll: function() { }
})
```

表 3-4 Page 构造器的参数

| 参数属性 | 类型 | 描述 |
| --- | --- | --- |
| data | Object | 页面的初始数据 |
| onLoad | Function | 生命周期函数--监听页面加载，触发时机早于 onShow 和 onReady |
| onReady | Function | 生命周期函数--监听页面初次渲染完成 |
| onShow | Function | 生命周期函数--监听页面显示，触发事件早于 onReady |
| onHide | Function | 生命周期函数--监听页面隐藏 |
| onUnload | Function | 生命周期函数--监听页面卸载 |
| onPullDownRefresh | Function | 页面相关事件处理函数--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 | 用户点击右上角转发 |
| onPageScroll | Function | 页面滚动触发事件的处理函数 |
| 其他 | Any | 可以添加任意的函数或数据，在 Page 实例的其他函数中用 this 可以访问 |

# 3. 页面的生命周期和打开参数

页面初次加载的时候，微信客户端就会给 Page 实例派发 onLoad 事件，Page 构造器参数所定义的 onLoad 方法会被调用，onLoad 在页面没被销毁之前只会触发 1 次，在 onLoad 的回调中，可以获取当前页面所调用的打开参数 option，关于打开参数我们放在这一节的最后再展开阐述。

页面显示之后，Page 构造器参数所定义的 onShow 方法会被调用，一般从别的页面返回到当前页面时，当前页的 onShow 方法都会被调用。

在页面初次渲染完成时，Page 构造器参数所定义的 onReady 方法会被调用，onReady 在页面没被销毁前只会触发 1 次，onReady 触发时，表示页面已经准备妥当，在逻辑层就可以和视图层进行交互了。

以上三个事件触发的时机是 onLoad 早于 onShow，onShow 早于 onReady。

页面不可见时，Page 构造器参数所定义的 onHide 方法会被调用，这种情况会在使用 wx.navigateTo 切换到其他页面、底部 tab 切换时触发。

当前页面使用 wx.redirectTo 或 wx.navigateBack 返回到其他页时，当前页面会被微信客户端销毁回收，此时 Page 构造器参数所定义的 onUnload 方法会被调用。

我们可以看到，Page 的生命周期是由微信客户端根据用户操作主动触发的。为了避免程序上的混乱，我们不应该在其他代码中主动调用 Page 实例的生命周期函数。

最后我们说一下页面的打开参数 query，让我们来设想这样一个场景，我们实现一个购物商城的小程序，我们需要完成一个商品列表页和商品详情页，点击商品列表页的商品就可以跳转到该商品的详情页，当然我们不可能为每个商品单独去实现它的详情页。我们只需要实现一个商品详情页的 pages/detail/detail.*(代表 WXML/WXSS/JS/JSON 文件*即可，在列表页打开商品详情页时把商品的 id 传递过来，详情页通过刚刚说的 onLoad 回调的参数 option 就可以拿到商品 id，从而绘制出对应的商品，代码如代码清单 3-9 所示。

代码清单 3-9 页面的打开参数 Page 构造器

```
// pages/list/list.js
// 列表页使用 navigateTo 跳转到详情页
wx.navigateTo({ url: 'pages/detail/detail?id=1&other=abc' })

// pages/detail/detail.js
Page({
  onLoad: function(option) {
        console.log(option.id)
        console.log(option.other)
  }
})
```

小程序把页面的打开路径定义成页面 URL，其组成格式和网页的 URL 类似，在页面路径后使用英文 ？分隔 path 和 query 部分，query 部分的多个参数使用 & 进行分隔，参数的名字和值使用 key=value 的形式声明。在页面 Page 构造器里 onLoad 的 option 可以拿到当前页面的打开参数，其类型是一个 Object，其键值对与页面 URL 上 query 键值对一一对应。和网页 URL 一样，页面 URL 上的 value 如果涉及特殊字符（例如：&字符、?字符、中文字符等，详情参考 URI 的 RFC3986 说明 ），需要采用 UrlEncode 后再拼接到页面 URL 上。

# 4. 页面的数据

3.1.4 节讨论了小程序界面渲染的基本原理，我们知道小程序的页面结构由 WXML 进行描述，WXML 可以通过数据绑定的语法绑定从逻辑层传递过来的数据字段，这里所说的数据其实就是来自于页面 Page 构造器的 data 字段，data 参数是页面第一次渲染时从逻辑层传递到渲染层的数据。

代码清单 3-10 Page 构造器的 data 参数

```
<!-- page.wxml -->
<view>{{text}}</view>
<view>{{array[0].msg}}</view>

// page.js
Page({
  data: {
    text: 'init data',
    array: [{msg: '1'}, {msg: '2'}]
  }
})
```

宿主环境所提供的 Page 实例的原型中有 setData 函数，我们可以在 Page 实例下的方法调用 this.setData 把数据传递给渲染层，从而达到更新界面的目的。由于小程序的渲染层和逻辑层分别在两个线程中运行，所以 setData 传递数据实际是一个异步的过程，所以 setData 的第二个参数是一个 callback 回调，在这次 setData 对界面渲染完毕后触发。

setData 其一般调用格式是 setData(data, callback)，其中 data 是由多个 key: value 构成的 Object 对象。

代码清单 3-11 使用 setData 更新渲染层数据

```
// page.js
Page({
  onLoad: function(){
    this.setData({
      text: 'change data'
    }, function(){
      // 在这次 setData 对界面渲染完毕后触发
    })
  }
})
```

实际在开发的时候，页面的 data 数据会涉及相当多的字段，你并不需要每次都将整个 data 字段重新设置一遍，你只需要把改变的值进行设置即可，宿主环境会自动把新改动的字段合并到渲染层对应的字段中，如下代码所示。data 中的 key 还可以非常灵活，以数据路径的形式给出，例如 this.setData({"d[0]": 100}); this.setData({"d[1].text": 'Goodbye'}); 我们只要保持一个原则就可以提高小程序的渲染性能：每次只设置需要改变的最小单位数据。

代码清单 3-12 使用 setData 更新渲染层数据

```
// page.js
```

```
Page({
  data: {
    a: 1, b: 2, c: 3,
    d: [1, {text: 'Hello'}, 3, 4]
  }
  onLoad: function(){
      // a 需要变化时，只需要 setData 设置 a 字段即可
    this.setData({a : 2})
  }
})
```

此外需要注意以下 3 点：

1. 直接修改 Page 实例的 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。
2. 由于 setData 是需要两个线程的一些通信消耗，为了提高性能，每次设置的数据不应超过 1024kB。
3. 不要把 data 中的任意一项的 value 设为 undefined，否则可能会有引起一些不可预料的 bug。

# 5. 页面的用户行为

小程序宿主环境提供了四个和页面相关的用户行为回调：

1. 下拉刷新 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 page.json 中设置 enablePullDownRefresh 为 true。
   当处理完数据刷新后，wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。
2. 上拉触底 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 page.json 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。
3. 页面滚动 onPageScroll
   监听用户滑动页面事件，参数为 Object，包含 scrollTop 字段，表示页面在垂直方向已滚动的距离（单位 px）。
4. 用户转发 onShareAppMessage
   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，在用户点击转发按钮的时候会调用，此事件需要 return 一个 Object，包含 title 和 path 两个字段，用于自定义转发内容，如代码清单 3-13 所示。

   代码清单 3-13 使用 onShareAppMessage 自定义转发字段

```
// page.js
Page({
onShareAppMessage: function () {
 return {
   title: '自定义转发标题',
   path: '/page/user?id=123'
 }
}
})
```

# 6. 页面跳转和路由

一个小程序拥有多个页面，我们可以通过 wx.navigateTo 推入一个新的页面，如图 3-6 所示，在首页使用 2 次 wx.navigateTo 后，页面层级会有三层，我们把这样的一个页面层级称为页面栈。

图 3-6 使用 2 次 wx.navigateTo 后的页面栈

后续为了表述方便，我们采用这样的方式进行描述页面栈：[ pageA, pageB, pageC ]，其中 pageA 在最底下，pageC 在最顶上，也就是用户所看到的界面，需要注意在本书编写的时候，小程序宿主环境限制了这个页面栈的最大层级为 10 层，也就是当页面栈到达 10 层之后就没有办法再推入新的页面了。我们下面来通过上边这个页面栈描述以下几个和导航相关的 API。

使用 wx.navigateTo({ url: 'pageD' }) 可以往当前页面栈多推入一个 pageD，此时页面栈变成 [ pageA, pageB, pageC, pageD ]。

使用 wx.navigateBack() 可以退出当前页面栈的最顶上页面，此时页面栈变成 [ pageA, pageB, pageC ]。

使用 wx.redirectTo({ url: 'pageE' }) 是替换当前页变成 pageE，此时页面栈变成 [ pageA, pageB, pageE ]，当页面栈到达 10 层没法再新增的时候，往往就是使用 redirectTo 这个 API 进行页面跳转。

小程序提供了原生的 Tabbar 支持，我们可以在 app.json 声明 tabBar 字段来定义 Tabbar 页（注：更多详细参数见 Tabbar 官方文档 ）。

代码清单 3-14 app.json 定义小程序底部 tab

```
{
  "tabBar": {
    "list": [
      { "text": "Tab1", "pagePath": "pageA" },
      { "text": "Tab1", "pagePath": "pageF" },
      { "text": "Tab1", "pagePath": "pageG" }
    ]
  }
}
```

我们可以在刚刚的例子所在的页面栈中使用 wx.switchTab({ url: 'pageF' })，此时原来的页面栈会被清空（除了已经声明为 Tabbar 页 pageA 外其他页面会被销毁），然后会切到 pageF 所在的 tab 页面，页面栈变成 [ pageF ]，此时点击 Tab1 切回到 pageA 时，pageA 不会再触发 onLoad，因为 pageA 没有被销毁。

补充一下，wx.navigateTo 和 wx.redirectTo 只能打开非 TabBar 页面，wx.switchTab 只能打开 Tabbar 页面。

我们还可以使用 wx. reLaunch({ url: 'pageH' }) 重启小程序，并且打开 pageH，此时页面栈为 [ pageH ]。表 3-5 罗列了详细的页面路由触发方式及页面生命周期函数的对应关系。

表 3-5 页面路由触发方式及页面生命周期函数的对应关系

| 路由方式 | 触发时机 | 路由前页面生命周期 | 路由后页面生命周期 |
| --- | --- | --- | --- |
| 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 | onHide | onLoad, onShow |
| 页面重定向 | 调用 API wx.redirectTo | onUnload | onLoad, onShow |

| 路由方式 | 触发时机 | 路由前页面生命周期 | 路由后页面生命周期 |
| --- | --- | --- | --- |
| 页面返回 | 调用 API wx.navigateBack | onUnload | onShow |
| Tab 切换 | 调用 API wx.switchTab | 请参考表 3-6 | 请参考表 3-6 |
| 重启动 | 调用 API wx.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）如表 3-6 所示，注意 Tabbar 页面初始化之后不会被销毁。

表 3-6 页面路由触发方式及页面生命周期函数的对应关系

| 当前页面 | 路由后页面 | 触发的生命周期（按顺序） |
| --- | --- | --- |
| A | A | 无 |
| 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() |