微信小程序开发初体验--教你开发小程序

文 / Ruter（已获授权）

原文链接：http://www.jianshu.com/p/7a94a6b6be96/comments/5078500

微信小程序面世以来受到的关注颇多，直到最近我才动手尝试进行了小程序的开发，总体上感觉还是不错的，有一点不适应的就是要摆脱 Web APP 开发对 DOM 的操作。在这里我就把我是如何利用 API 开发微信小程序的过程写成教程，教大家快速上手体验一次微信小程序的开发。

补充: 之前忘了把源码发上来，完成之后就已经放在 Github 上了 点我下载>>

在开始之前我们先来看下成品的效果图

准备工作

我们先确定想要开发一款什么样的小程序，首先要符合「小」，因为我们这次是要体验小程序的开发，所以尽量不要弄得太复杂；其次是「快」，小程序里需要的数据啊、资源啊，最好是现成就有的，自己写个 API 什么的这就太耗时了，就不叫快速上手了。

所以呢，如果能调用现成的 API 那是极好的，经过一番挑选，我选择了聚合数据的历史上的今天这个 API，

调用这个 API 获取数据，我们只要做 2 个页面就可以完全展示出来了，又「小」又「快」哈 XD

- API 一枚: 「历史上的今天」
- 微信小程序开发者工具

注: API 需要注册之后获得 KEY 才能使用，具体请查看聚合数据官方文档，这里默认各位已经注册并拥有相应 API 所需的 KEY

工程结构

微信开发者工具的安装和使用在这里就不多作介绍了，有疑问的话可以看微信官方的简易教程

先创建一个工程，依次点击「添加项目」--「无 AppID」，然后填好「项目名称」并选择「项目目录」，点击「添加项目」

添加项目

然后我们来清理一下默认工程的目录结构，删除以下目录和文件

pages/logs/

pages/index/index.wxss

创建以下目录和文件

pages/detail/

pages/detail/detail.js

pages/detail/detail.wxml

pages/templates/

pages/templates/item.wxml

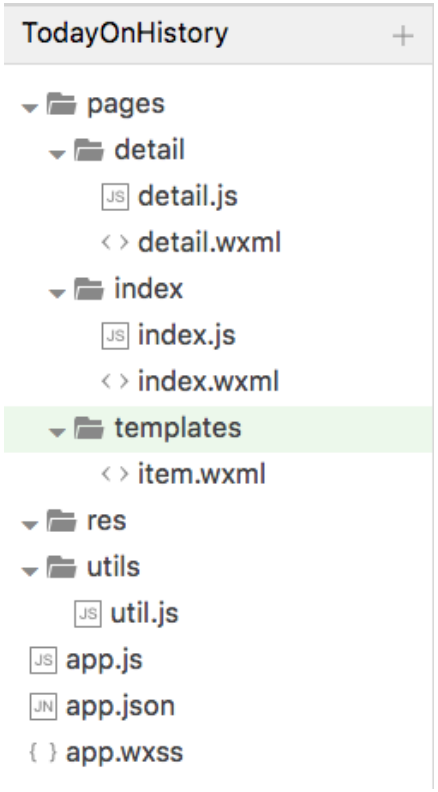
res/

现在你看到的目录结构应该是这样子的

.

```
├── app.js
├── app.json
├── app.wxss
├── pages
│   ├── detail
│   │   ├── detail.js
│   │   └── detail.wxml
│   ├── index
│   │   ├── index.js
│   │   └── index.wxml
│   └── templates
```

```
|           └── item.wxml
├── res
└── utils
    └── util.js
```



目录结构

这就是我们工程目录的最终结构了，后面还会添加资源进去，但是整体结构还是这样不会改变的

修改配置

微信小程序是通过修改 app. json 文件改变全局配置的，具体的可配置项请各位自行查阅小程序文档的配置一节

打开 app. json，修改成

```
{
    "pages":[
        "pages/index/index",
        "pages/detail/detail"
    ]
        "pages/index/index",
        "pages/detail/detail"
    ],
    "window":{
        "backgroundTextStyle":"light",
        "navigationBarBackgroundColor": "#3e3e3e",
        "navigationBarTitleText": "历史今日",
        "navigationBarTextStyle":"white"
```

```
  }
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#3e3e3e",
    "navigationBarTitleText": "历史今日",
    "navigationBarTextStyle":"white"

  },
  "debug": true
}
```

我们这里修改了导航栏的背景颜色(navigationBarBackgroundColor)、标题颜色(navigationBarTextStyle)以及标题内容(navigationBarTitleText)，为了方便查看调试信息，我还开启了 debug 模式

全局配置以及搞定啦，接下来正式开始编码

首页布局

前面已经说了我们要做的是「历史上的今天」这样的一个小程序，所以同一个日期会有很多条目，最常见的布局就是做成列表

列表里会有很多的条目，数量是不确定的，所以我们不能在页面里写死了布局，这时候就要用到模板

(template)了，我们可以在模板中定义代码片段，然后在不同的地方调用

我们来定义一个模板，打开 pages/templates/item.wxml ，添加代码

```
<template name="tItem">
    <navigator url="../detail/detail?id={{item.e_id}}">
        <view class="ui-list-item ui-pure-item ui-border-b">
            <view class="ui-item-span"><text>{{item.date}}</text></view>
            <view class="ui-item-content ui-nowrap"><text>{{item.title}}</text></view>
        </view>
    </navigator>
</template>
```

注: 模板的使用细节请参考官方文档模板一节

接下来打开 pages/index/index.wxml 删掉里面的内容，我们要在这里编写列表页，这里会使用到我们上面定义的模板

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


<scroll-view scroll-y="true" class="flex-row ui-list ui-border-t">
```

```
        <template is="tItem" data="{{item}}" wx:for="{{events}}"/>
        <view class="ui-tips">
            <view wx:if="{{hidden}}"><text>没有更多内容了</text></view>
            <view wx:else><text>内容加载中...</text></view>
        </view>
</scroll-view>

<loading hidden="{{hidden}}">Loading...</loading>
```

可以注意到第一行使用了 import 将模板引入到页面中，然后再使用 is 属性，声明需要使用的模板，用 data 属性传入数据供模板使用

注: 模板拥有自己的作用域，只能使用 data 传入的数据

为了测试和查看布局效果，我们打开 pages/index/index.js 删除里面的代码，然后添加以下代码手动创建数据传入给页面渲染

```
Page({
    data: {
        hidden: true,
```

```
events: [
    {
        date: "2016-10-14",
        title: "TodayOnHistory，历史上的今天"
    },
    {
        date: "2016-10-14",
        title: "TodayOnHistory，历史上的今天"
    },
    {
        date: "2016-10-14",
        title: "TodayOnHistory，历史上的今天"
    },
    {
        date: "2016-10-14",
        title: "TodayOnHistory，历史上的今天"
    },
    {
        date: "2016-10-14",
        title: "TodayOnHistory，历史上的今天"
```

```
        },
        {
            date: "2016-10-14",
            title: "TodayOnHistory，历史上的今天"
        },
        {
            date: "2016-10-14",
            title: "TodayOnHistory，历史上的今天"
        }
    ]
  }
})
```

保存后点击开发工具左侧的编译，即可查看到效果

首页布局效果

注: 布局会用到图标字体，导入到 res/下，样式则写在 app.wxss 全局样式表中，图标字体文件和样式请从源码中获取，这篇教程不作样式的说明

详细页面

首页的布局已经完成了，暂时放下首页列表，接下来开始编写详细内容的页面

打开 pages/detail/detail.wxml，添加内容如下

```
<view class="container">
    <view class="ui-title ui-border-b"><text>{{detail.title}}</text></view>
    <view class="ui-content"><text>{{detail.content}}</text></view>
    <block wx:for="{{detail.picUrl}}">
        <view>
            <view><image mode="aspectFit" src="{{item.url}}"/></view>
            <view class="ui-pic-title"><text>{{item.pic_title}}</text></view>
        </view>
    </block>
</view>
<loading hidden="{{hidden}}">Loading...</loading>
```

搞定，这个页面就这么简单就 OK 了，现在我们打开 pages/detail/detail.js 手动添加数据到这个页面

中查看效果

```
Page({
```

```
  data:{
      hidden: true,
      detail: {
          title: "历史上的今天",
          content: "历史上的今天历史上的今天历史上的今天历史上的今天历史",
          picUrl: [
              {
                  url: "http://sjbz.fd.zol-img.com.cn/t_s320x510c/g5/M00/00/04/
ChMkJlfJWJCIYePaAAPdCld59MEAAU-KAP0U3gAA90i450.jpg",
                  pic_title: "这是图片标题"
              }
          ]
      }
  }
})
```

详细页面效果

填充数据

现在页面布局都已经完成了，是时候调用 API 编写逻辑层的代码来填充数据到页面上了，在开始之前我们

先清理一下无用的代码

打开 app. js 删掉无用的函数和属性

App({

})

})

以上这步是额外步骤，并不影响我们接下来要做的事情

打开 utils/util.js 并清空里面的代码，添加如下内容

```
// 查询事件列表的 Base URL
const API_URL_L = "http://v.juhe.cn/todayOnhistory/queryEvent.php"
// 查询详细信息的 Base URL
const API_URL_D = "http://v.juhe.cn/todayOnhistory/queryDetail.php"
// 申请 API 获得的 KEY
const API_KEY = "YOUR API KEY"

// 获取事件列表
function fetchEvents(today) {
    var promise = new Promise(function(resolve, reject){
```

```
        wx.request({
            url: API_URL_L,
            data: {
                key: API_KEY,
                date: today
            },
            header: {
                'Content-Type': 'application/json'
            },
            success: resolve,
            fail: reject
        })
    })
    return promise
}
```

```
function getEvents() {
    var tmpDate = new Date()
    var today = tmpDate.getMonth() + 1
    today = today + '/' + tmpDate.getDate()
    return fetchEvents(today)
        .then(function(res) {
            // console.log(res.data.result)
            return res.data.result
        })
        .catch(function(err) {
            console.log(err)
            return []
        })
}

// 获取详细信息
```

```
function fetchDetail(e_id) {
    var promise = new Promise(function(resolve, reject){
        wx.request({
            url: API_URL_D,
            data: {
                key: API_KEY,
                e_id: e_id
            },
            header: {
                'Content-Type': 'application/json'
            },
            success: resolve,
            fail: reject
        })
    })
    return promise
```

```
}

function getDetail(e_id) {
    return fetchDetail(e_id)
        .then(function(res) {
            // console.log(res.data.result)
            return res.data.result
        })
        .catch(function(err) {
            console.log(err)
            return []
        })
}

module.exports = {
    getEvents: getEvents,
```

```
    getDetail: getDetail
}
```

注: 请将 API_KEY 的值替换为你申请到的 KEY

我们要利用 API 获取的数据有两种，一是「事件列表」，另一种是事件对应的「详细信息」，上面使用到了 ES6 原生提供的 Promise 对象 ,具体请参考阮一峰的《JavaScript 标准参考教程(alpha)》中「Promise 对象」一节

最后还用到了 module.exports 对外暴露两个函数，使外部可以调用

我们继续打开 pages/index/index.js 文件，修改成这样

```
const util = require('../../utils/util.js')


Page({
  data: {
    hidden: false,
    events: []
  },
  onLoad:function(options){
```

```
    // 页面初始化  options 为页面跳转所带来的参数
    var self = this
    util.getEvents().then(function(data) {
      self.setData({
        hidden: true,
        events: data
      })
    })
  }
})
```

然后打开 pages/detail/detail.js，修改如下

```
const util = require('../../utils/util.js')

Page({
  data:{
      hidden: false,
```

```
        detail: {}
},
onLoad:function(param){
    // 页面初始化  param 为页面跳转所带来的参数
    var self = this
    util.getDetail(param.id).then(function(result){
            self.setData({
                    detail: result[0]
            })
    })
},
onReady:function(){
    // 页面渲染完成
    wx.setNavigationBarTitle({
            title: this.data.detail.title
    })
```

```
        this.setData({
            hidden: true
        })
    }
})
```

这里我们调用了 wx.setNavigationBarTitle 方法动态设置导航栏的标题内容，需要注意的是必须在页面渲染完成之后，即 onReady 之后才能调用该方法修改导航栏标题

这次教程就到这里结束啦～涉及到的部分知识点并没有详细介绍和说明，如果有不明白的地方请大家根据我给出的链接去查看详细的介绍，此文权当快速上手的一个引子，更加深入的内容以及小程序的其他 API 的使用，还需要各位亲自去实践，欢迎交流～

参考链接

- 微信小程序简易教程
- 微信小程序框架
- 微信小程序 API
- Promise 对象