微信小程序学习摘要系列《二》逻辑层

逻辑层(App Service)

MINA 的逻辑层是由 JavaScript 编写。

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。

在 JavaScript 的基础上，我们做了一些修改，以方便地开发 MINA 程序。

增加 App 和 Page 方法，进行程序和页面的注册。

提供丰富的 API，如扫一扫，支付等微信特有能力。

每个页面有独立的作用域，并提供模块化能力。

由于 MINA 并非运行在浏览器中，所以 JavaScript 在 web 中一些能力都无法使用，如 document，window 等。

开发者写的所有代码最终将会打包成一份 JavaScript，并在小程序启动的时候运行，直到小程序销毁。类似

ServiceWorker，所以逻辑层也称之为 App Service。

注册程序

App()函数用来注册一个小程序，接受一个 Object 参数，其指定小程序

生命周期函数等。

object 参数说明

属性	类型	描述	触发时机
onLaunch	Function	生命周期函数-监听小程序初始化	当小程序初始化完成时，全局只触发一次
onShow	Function	生命周期-监听小程序显示	小程序启动或者从后台进入到前台
onHide	Function	生命周期-监听小程序隐藏	小程序从前台进入到后台
其他	Any	可以添加任意的函数或数据到 Object 参数中，用 this 可以访问	

前台、后台定义:当用户点击左上角关闭，或者按了设备 Home 键离开微信，

小程序并没有正在销毁，而是进入了后台，当再次启动微信或再次打开小程序，又会从后台进入前台。

只有当小程序进入后台一定时间，或者系统资源占用过高，才会被真正销毁。

示例代码：

```
App({
  onLaunch: function() {
    // Do something initial when launch.
  },
  onShow: function() {
      // Do something when show.
  },
  onHide: function() {
      // Do something when hide.
  },
  globalData: 'I am global data'
})
```

App.prototype.getCurrentPage()获取当前页面实例

getApp()获取小程序实例

注意:App()必须在 app.js 中注册，且不能注册多个。

不要在定义 App（ ）内的函数中调用 getApp()，使用 this

不要在 onLaunch 的时候调用 getCurrentPage()，此时 page 还没有生成

通过 getApp 获取实例之后，不要私自调用生命周期的函数。

注册页面

page()函数用来注册一个页面，接受一个 object 参数，其指定页面初始数据，生命周期函数，事件处理函数等。

object 参数说明:

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
onReady	Function	生命周期函数-监听页面渲染完成
onShow	Function	生命周期函数-监听页面显示
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
其他	Any	可以添加任意的函数或数据到 Object 参数中，用 this 可以访问

示例代码：

```
//index.js

Page({

  data: {

    text: "This is page data."

  },

  onLoad: function(options) {

    // Do some initialize when page load.

  },

  onReady: function() {

    // Do something when page ready.

  },

  onShow: function() {

    // Do something when page show.

  },

  onHide: function() {

    // Do something when page hide.
```

```
  },
  onUnload: function() {
    // Do something when page close.
  },
  // Event handler.
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    })
  }
})
```

初始化数据

初始化数据将作为页面的第一次渲染。data 将会以 JSON 的形式由逻辑层传至渲染层，所以其数据必须是可以转成 JSON 的格式：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'}]
  }
})
```

事件处理函数

除了初始化数据和生命周期函数，Page 中还可以定义一些特殊的函数：事件处理函数，在渲染层可以在组件中加入事件绑定，

当达到触发事件时，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.prototype.setData()

setData 函数用于将数据从逻辑层发送到视图层，同时改变对应的 this.data 的值。

注意：

1.直接修改 this.data 无效，无法改变页面的状态，还会造成数据不一致。

2.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。

setData()参数格式

接受一个对象，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 可以非常灵活，以数据路径的形式给出，如 array[2].message，a.b.c.d，并且不需要在 this.data 中预先定义。

示例代码：

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

<view>{{text}}</view>

<button bindtap="changeText"> Change normal data </button>

<view>{{array[0].text}}</view>

<button bindtap="changeItemInArray"> Change Array data </button>

<view>{{obj.text}}</view>

<button bindtap="changeItemInObject"> Change Object data </button>

<view>{{newField.text}}</view>

<button bindtap="addNewField"> Add new data </button>

//index.js

Page({

  data: {

    text: 'init data',

    array: [{text: 'init data'}],

    object: {

      text: 'init data'
```

```
    }
},
changeText: function() {
    // this.data.text = 'changed data'    // bad, it can not work
    this.setData({
        text: 'changed data'
    })
},
changeItemInArray: function() {
    // you can use this way to modify a danamic data path
    var changedData = {}
    var index = 0
    changedData['array[' + index + '].text'] = 'changed data'
    this.setData(changedData)
},
changeItemInObject: function(){
```

```
    this.setData({
      'object.text': 'changed data'
    });
  },
  addNewField: function() {
    this.setData({
      'newField.text': 'new data'
    })
  }
})
```

页面路由

在小程序中所有页面的路由全部由 MINA 进行管理，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	路由后页面	路由前页面
初始化	小程序打开的第一个页面	onLoad，onShow	
打开新页面	调用 APIwx.nativateTo 或使用组件 Natigator	onLoad，onShow	onHide
页面重定向	调用 APIwx.redirectTo 或使用组件 Natigator	onLoad，onShow	onUnload
页面返回	调用 APIwx.navigatBack 或用户按左上角返回按钮	onShow	onUnload
Tab 切换	多 Tab 模式下用户切换 Tab	第一次打开 onLoad，onshow；否则 onShow	onHide

模块化

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++
// b.js
```

```
// You can redefine localValue in file b.js, without interference with the localValue in a.js.

var localValue = 'b'

// If a.js it run before b.js, now the globalData shoule be 2.

console.log(getApp().globalData)
```

模块化

我们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 才能对外暴露接口。

```
// common.js

function sayHello(name) {

   console.log('Hello ' + name + '!')

}
```

```
module.exports = {
  sayHello: sayHello
}
```

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')
  }
})
```