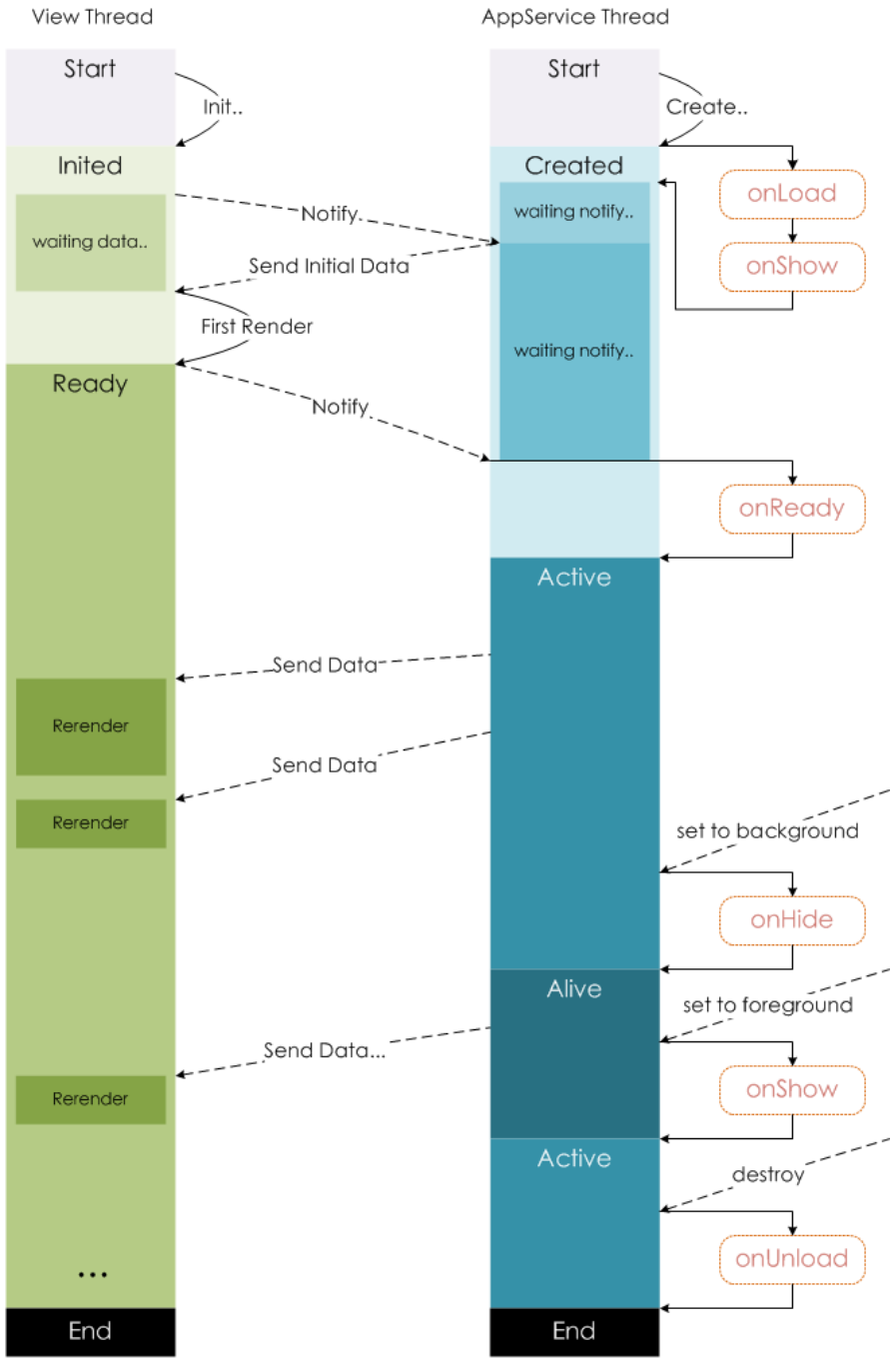
微信小程序把玩《二》：页面生命周期，模块化，数据绑定，view 组件 ...

五：页面生命周期

这里只要熟悉页面的基本生命周期即可，业务在指定生命周期函数内书写。

以下是官网给出的生命周期函数方法和状态图

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	开发者可以添加任意的函数或数据到 OBJECT 参数中，用 `this` 可以访问


View Thread
AppService Thread
Start
Init..
Inited
waiting data..
Notify..
Send Initial Data
First Render
Ready
Notify..
Rerender
Rerender
Rerender
...
End
Start
Create..
Created
waiting notify..
waiting notify..
onLoad
onShow
onReady
Active
Send Data
Send Data
Send Data...
set to background
onHide
Alive
set to foreground
onShow
Active
destroy
onUnload
End

- 上面的生周期函数图对于做 Android 或者 IOS 的来书理解起来应该不是难事，具体怎么掌握只有慢慢尝试和摸索

代码处理:

这里的代码主需要对使用创建项目时 index 目录下文件处理下就行,至于跳转后的页面用的还是 logs 不需要更改！下面贴

下代码注释也比较详细

index.wxml

```
<!--index.wxml-->
<view class="container">
<!--绑定点击事件-->
  <view   bindtap="bindViewTap" class="userinfo">
  </view>
  <view class="usermotto">
  <!--数据绑定-->
    <text class="user-motto">{{motto}}</text>
  </view>
</view>
```

index.wxss

```
/**index.wxss**/


.container {
  width: 800;
  height: 800;
}
.userinfo {
  width: 120rpx;
  height: 120rpx;
  background: red;
}
```

index.js

```
//index.js
//获取应用实例
var app = getApp()
Page({
```

```
/**
 *  通过 data 初始化数据
 */
data: {
   motto: '点击上面 View 跳转',
   // userInfo: {}
},
//事件处理函数
bindViewTap: function() {
   //通过调用 API 进行跳转
   wx.navigateTo({
      url: '../logs/logs'
   })
},
/**
 *  监听页面开在加载的状态
```

```
 *      页面加载完成之后就不会在执行
 */
onLoad: function () {
  console.log('index---------onLoad()')
  // //this 指的就是本页面对象
  // var that = this
  // //调用应用实例的方法获取全局数据
  // app.getUserInfo(function(userInfo){
  //    //更新数据
  //    that.setData({
  //       userInfo:userInfo
  //    })
  //    //更新本页面
  //    that.update()
  // })
},
```

```
/**
 *  监听页面显示，
 *     当从当前页面调转到另一个页面
 *     另一个页面销毁时会再次执行
 */
onShow: function() {
  console.log('index---------onShow()')
},
/**
 * 监听页面渲染完成
 *     完成之后不会在执行
 */
onReady: function() {
  console.log('index---------onReaday()');
},
/**
```

```
  * 监听页面隐藏
  *     当前页面调到另一个页面时会执行
  */
 onHide: function() {
   console.log('index---------onHide()')
 },
 /**
  * 当页面销毁时调用
  */
 onUnload: function() {
   console.log('index---------onUnload')
 }


})
```

六：模块化

模块化也就是将一些通用的东西抽出来放到一个文件中，通过 module.exports 去暴露接口。我们在最初新建项目时就有

个 util.js 文件就是被模块化处理时间的

```
/**
 *  处理具体业务逻辑
 */
function formatTime(date) {
  //获取年月日
  var year = date.getFullYear()
  var month = date.getMonth() + 1
  var day = date.getDate()


  //获取时分秒
  var hour = date.getHours()
  var minute = date.getMinutes()
  var second = date.getSeconds();
```

```
  //格式化日期
  return [year, month, day].map(formatNumber).join('/') + ' ' + [hour, minute,
second].map(formatNumber).join(':')
}


function formatNumber(n) {
  n = n.toString()
  return n[1] ? n : '0' + n
}


/**
 * 模块化导出暴露接口
 */
module.exports = {
  formatTime: formatTime
```

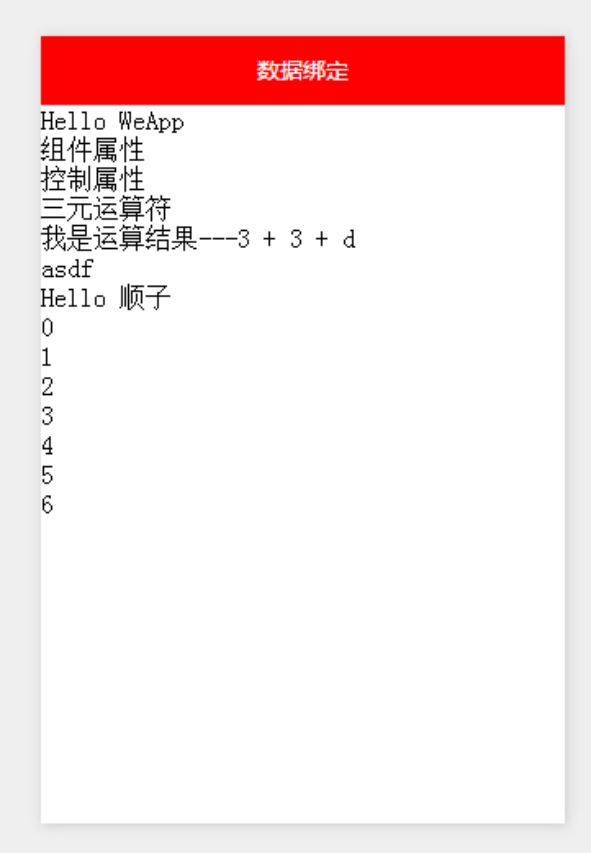
```
}
```

使用方式：

```
//导入模块化方式
var util = require('../../utils/util.js')
Page({
  data: {
    logs: []
  },
  onLoad: function () {
    this.setData({
      logs: (wx.getStorageSync('logs') || []).map(function (log) {
        // 通过暴露的接口调用模块化方法
        return util.formatTime(new Date(log))
      })
    })
  }
```

})

七：数据绑定



数据绑定有一部分前几个看着还行，后面的几个可能有几个不理解，界面展示的数据有的也因为条件没法显示。看不懂的

可以先记着，后面真正用到时就会明白，反正我是这样想的。这里先记录下

data.wxml

<!--数据绑定使用对象---内容-->

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


<!--数据绑定使用对象---组件属性---需要在双引号之内-->

<view id="item-{{id}}">组件属性</view>


<!--数据绑定使用对象---控制属性---需要在双引号之内-->

<view wx:if="{{condition}}">控制属性</view>


<!--数据绑定使用对象---三元运算-->

<view hindden="{{flag ? true : false}}">三元运算符</view>


<!--数据绑定使用对象---算数运算-->

<view>我是运算结果---{{a + b}} + {{c}} + d</view>
```

```
<!--数据绑定使用对象---逻辑判断-->

<view wx:if="{{length > 5}}">asdf</view>



<!--数据绑定使用对象---字符串运算-->

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



<!--数据绑定使用对象---数组组合-->

<view wx:for="{{[zero, 1, 2, 3, 4, 5, 6]}}">{{item}}</view>



<!--数据绑定使用对象---对象-->

<template is="objectCombine" data="{{for: x, bar: y}}"></template>



<!--数据绑定使用对象---扩展运算符对象  ...   将一个对象展开-->

<template is="objectCombine" data="{{...obj1, ...obj2, p: 5}}"></template>
```

```
<!--数据绑定使用对象---对象的 key 和 value 相同时-->
<template is="objectCombine" data="{{foo, bar}}"></template>
```

data.js

```
Page({
  data:{

    //内容绑定
    message: 'Hello WeApp',

    //组件属性绑定
    id: 0,

    //控制属性绑定
```

condition: true,

//三元运算

flag:false,

//算数运算

a: 1,

b: 2,

c: 3,

//逻辑判断

length: 6,

//字符串运算

```
name: '顺子',


//数组组合
zero: 0,


//对象


x: 0,
y: 1,


//对象展开
obj1: {
    a: 1,
    b: 2
```

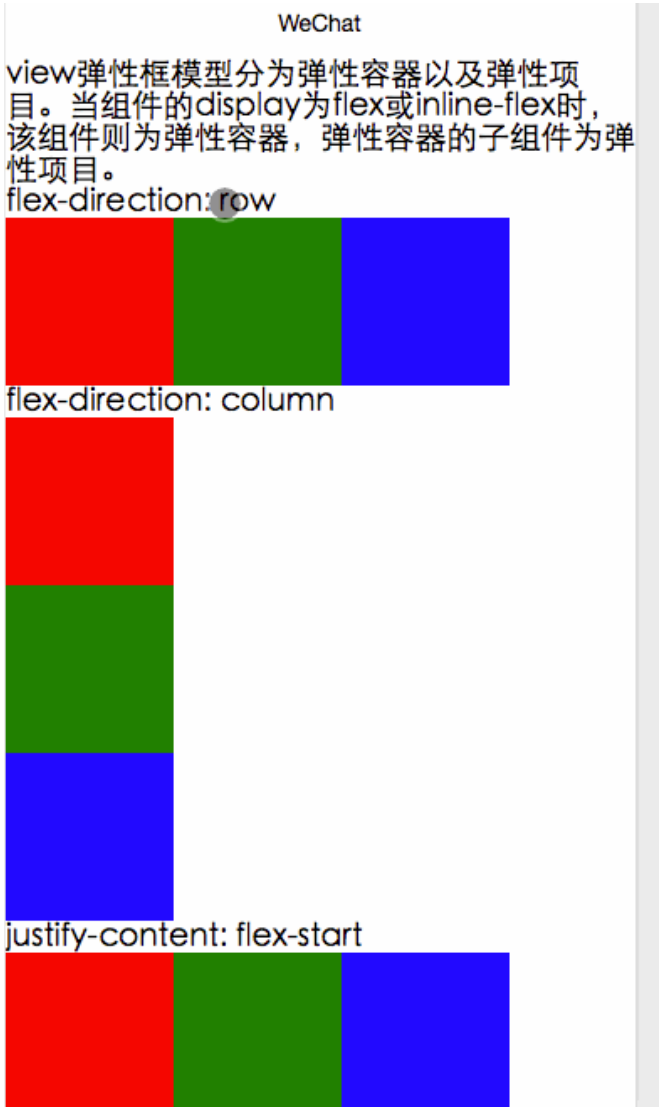
```
  },
  obj2: {
      c: 3,
      d: 4
  },
  p: 5,

  //对象 key 和 value 形同时
  foo: 'my-foo',
  bar: 'my-bar'

},
```

})

八：view 组件



刚看到这个效果的时候还真是和 ReactNative 的效果一致，属性也基本的一样.

view 这个组件就是一个视图组件使用起来非常简单。

主要属性：

flex-direction： 主要两个特性”row”横向排列”column”纵向排列

justify-content 主轴的对齐方式（如果 flex-direction 为 row 则主轴就是水平方向）

- 可选属性（‘flex-start’，‘flex-end’，‘center’，‘space-between’，‘space-around’）

align-items 侧轴对齐方式如果 flex-direction 为 row 则侧轴就是垂直方向）

- 可选属性（‘flex-start’，‘flex-end’，‘center’ ）

wxml

```
<view class="page">
    <view class="page__hd">
        <text class="page__title">view</text>
        <text class="page__desc">弹性框模型分为弹性容器以及弹性项目。当组件的 display 为 flex 或 inline-flex 时，
该组件则为弹性容器，弹性容器的子组件为弹性项目。</text>
    </view>
    <view class="page__bd">
        <view class="section">
```

```
    <view class="section__title">flex-direction: row</view>
    <view class="flex-wrp" style="flex-direction:row;">
        <view class="flex-item" style="background: red"></view>
        <view class="flex-item" style="background: green"></view>
        <view class="flex-item" style="background: blue"></view>
    </view>
</view>
<view class="section">
    <view class="section__title">flex-direction: column</view>
    <view class="flex-wrp" style="height: 300px;flex-direction:column;">
        <view class="flex-item" style="background: red"></view>
        <view class="flex-item" style="background: green"></view>
        <view class="flex-item" style="background: blue"></view>
    </view>
</view>
<view class="section">
```

```
<view class="section__title">justify-content: flex-start</view>

<view class="flex-wrp" style="flex-direction:row;justify-content: flex-start;">

    <view class="flex-item" style="background: red"></view>
```