4.6 本地数据缓存

本地数据缓存是小程序存储在当前设备上硬盘上的数据，本地数据缓存有非常多的用途，我们可以利用本地数据缓存来存储用户在小程序上产生的操作，在用户关闭小程序重新打开时可以恢复之前的状态。我们还可以利用本地缓存一些服务端非实时的数据提高小程序获取数据的速度，在特定的场景下可以提高页面的渲染速度，减少用户的等待时间。

4.6.1 读写本地数据缓存

小程序提供了读写本地数据缓存的接口，通过 wx.getStorage/wx.getStorageSync 读取本地缓存，通过 wx.setStorage/wx.setStorageSync 写数据到缓存，其中 Sync 后缀的接口表示是同步接口[9]，执行完毕之后会立马返回，示例代码和参数说明如下所示。

代码清单 4-13 wx.getStorage/wx.getStorageSync 读取本地数据缓存

```
wx.getStorage({
  key: 'key1',
  success: function(res) {
    // 异步接口在 success 回调才能拿到返回值
    var value1 = res.data
  },
  fail: function() {
    console.log('读取 key1 发生错误')
  }
```

```
})


try{
  // 同步接口立即返回值
  var value2 = wx.getStorageSync('key2')
}catch (e) {
  console.log('读取 key2 发生错误')
}
```

表 4-4 wx.getStorage/wx.getStorageSync 详细参数

参数名	类型	必填	描述
key	String	是	本地缓存中指定的 key
success	Function	否	异步接口调用成功的回调函数，回调参数格式：{data: key 对应的内容}
fail	Function	否	异步接口调用失败的回调函数

参数名	类型	必填	描述
complete	Function	否	异步接口调用结束的回调函数（调用成功、失败都会执行）

代码清单 4-14 wx.setStorage/wx.setStorageSync 写入本地数据缓存

```
// 异步接口在 success/fail 回调才知道写入成功与否
wx.setStorage({
  key:"key",
  data:"value1"
  success: function() {
    console.log('写入 value1 成功')
  },
  fail: function() {
    console.log('写入 value1 发生错误')
  }
})

try{
  // 同步接口立即写入
  wx.setStorageSync('key', 'value2')
  console.log('写入 value2 成功')
```

```
}catch (e) {
  console.log('写入 value2 发生错误')
}
```

表 4-5 wx.setStorage/wx.setStorageSync 详细参数

参数名	类型	必填	描述
key	String	是	本地缓存中指定的 key
data	Object/String	是	需要存储的内容
success	Function	否	异步接口调用成功的回调函数
fail	Function	否	异步接口调用失败的回调函数
complete	Function	否	异步接口调用结束的回调函数（调用成功、失败都会执行）

4.6.2 缓存限制和隔离

小程序宿主环境会管理不同小程序的数据缓存，不同小程序的本地缓存空间是分开的，每个小程序的缓存空间上限为 10MB，如果当前缓存已经达到 10MB，再通过 wx.setStorage 写入缓存会触发 fail 回调。

小程序的本地缓存不仅仅通过小程序这个维度来隔离空间，考虑到同一个设备可以登录不同微信用户，宿主环境还对不同用户的缓存进行了隔离，避免用户间的数据隐私泄露。

由于本地缓存是存放在当前设备，用户换设备之后无法从另一个设备读取到当前设备数据，因此用户的关键信息不建议只存在本地缓存，应该把数据放到服务器端进行持久化存储。

4.6.3 利用本地缓存提前渲染界面

讨论一个需求：我们要实现了一个购物商城的小程序，首页是展示一堆商品的列表。一般的实现方法就是在页面 onLoad 回调之后通过 wx.request 向服务器发起一个请求去拉取首页的商品列表数据，等待 wx.request 的 success 回调之后把数据通过 setData 渲染到界面上，如下代码所示。

代码清单 4-15 page.js 拉取商品列表数据展示在界面上

```
Page({
  onLoad: function() {
    var that = this
    wx.request({
```

```
    url: 'https://test.com/getproductlist',
    success: function (res) {
      if (res.statusCode === 200) {
        that.setData({
          list: res.data.list
        })
      }
    }
  })
 }
})
```

设想一下当用户退出小程序再进来，界面仍然会有白屏现象，因为我们需要等待拉取商品列表的请求回来才能渲染商品列表。当然我们还可以再做一些体验上的优化，例如在发请求前，可能我们会在界面上显示一个 Loading 提示用户在加载中，但是并没有解决这个延迟渲染的现象，这个时候我们可以利用本地缓存来提前渲染界面。

我们在拉取商品列表后把列表存在本地缓存里，在 onLoad 发起请求前，先检查是否有缓存过列表，如果有的话直接渲染界面，然后等到 wx.request 的 success 回调之后再覆盖本地缓存重新渲染新的列表，如下代码所示。

代码清单 4-16 page.js 利用本地缓存提前渲染界面

```
Page({
```

```
onLoad: function() {

  var that = this

  var list =wx.getStorageSync("list")



  if (list) { // 本地如果有缓存列表，提前渲染

    that.setData({

      list: list

    })

  }

  wx.request({

    url: 'https://test.com/getproductlist',

    success: function (res) {

      if (res.statusCode === 200) {

        list = res.data.list

        that.setData({ // 再次渲染列表

          list: list

        })

        wx.setStorageSync("list",list) // 覆盖缓存数据

      }

    }

  })

 }
```

```
})
```

这种做法可以让用户体验你的小程序时感觉加载非常快，但是你还要留意这个做法的缺点，如果小程序对渲染的数据实时性要求非常高的话，用户看到一个旧数据的界面会非常困惑。因此一般在对数据实时性/一致性要求不高的页面采用这个方法来做提前渲染，用以优化小程序体验。

4.6.4 缓存用户登录态 SessionId

在 4.4 节我们说到处理用户登录态的一般方法，通常用户在没有主动退出登录前，用户的登录态会一直保持一段时间[10]，就无需用户频繁地输入账号密码。如果我们把 SessionId 记录在 Javascript 中某个内存变量，当用户关闭小程序再进来小程序时，之前内存的 SessionId 已经丢失，此时我们就需要利用本地缓存的能力来持久化存储 SessionId。

代码清单 4-17 利用本地缓存持久存储用户登录态 SessionId

```
//page.js

var app = getApp()

Page({

  onLoad: function() {

    // 调用 wx.login 获取微信登录凭证

    wx.login({

      success: function(res) {

        // 拿到微信登录凭证之后去自己服务器换取自己的登录凭证

        wx.request({
```

```
          url: 'https://test.com/login',
          data: { code: res.code },
          success: function(res) {
            var data = res.data
            // 把 SessionId 和过期时间放在内存中的全局对象和本地缓存里边
            app.globalData.sessionId =data.sessionId
            wx.setStorageSync('SESSIONID',data.sessionId)

            // 假设登录态保持 1 天
            var expiredTime = +new Date() +1*24*60*60*1000
            app.globalData.expiredTime =expiredTime
            wx.setStorageSync('EXPIREDTIME',expiredTime)
          }
        })
      }
    })
  }
})
```

在重新打开小程序的时候，我们把上一次存储的 SessionId 内容取出来，恢复到内存。

代码清单 4-18 利用本地缓存恢复用户登录态 SessionId

```
//app.js
```

```
App({

  onLaunch: function(options) {

    var sessionId =wx.getStorageSync('SESSIONID')

    var expiredTime =wx.getStorageSync('EXPIREDTIME')

    var now = +new Date()



    if (now - expiredTime <=1*24*60*60*1000) {

      this.globalData.sessionId = sessionId

      this.globalData.expiredTime = expiredTime

    }

  },

  globalData: {

    sessionId: null,

    expiredTime: 0

  }

})
```