**移动端H5开发及联调方案**

|  |  |  |  |
| --- | --- | --- | --- |
| **编写人员** | 靳立祥 | **编写部门** | APP开发部 |

目录

[一、 H5与原生通信 3](#_Toc20495783)

[1. H5与原生通信方式 3](#_Toc20495784)

[2. jsBridge与原生通信原理 4](#_Toc20495785)

[3. jsBridge源码解析 6](#_Toc20495786)

[4. 如何使用jsBridge 6](#_Toc20495787)

[5. 问题及解决方案 8](#_Toc20495788)

[二、 调试 8](#_Toc20495789)

[1. 浏览器调试H5 8](#_Toc20495790)

[2. vConsole调试 10](#_Toc20495791)

[3. 抓包工具调试 11](#_Toc20495792)

[4. Chrome调试Android App内H5页面（需翻墙） 12](#_Toc20495793)

[三、 适配方案 13](#_Toc20495794)

[1. rem适配方案 13](#_Toc20495795)

[2. vw适配方案 13](#_Toc20495796)

[四、 UI还原度注意事项 15](#_Toc20495797)

[1. 金融超市设计规范 15](#_Toc20495798)

[2. UI还原度注意事项 15](#_Toc20495799)

[五、 项目常用组件 16](#_Toc20495800)

[1. hyapp-utils 16](#_Toc20495801)

[2. hyapp-ui 16](#_Toc20495802)

[3. 说明文档 16](#_Toc20495803)

## H5与原生通信

### H5与原生通信方式

H5与Native交互，本质上就是两种调用：

* JavaScript 调用 Native
* Native 调用 JavaScript

1. Native调用 JavaScript

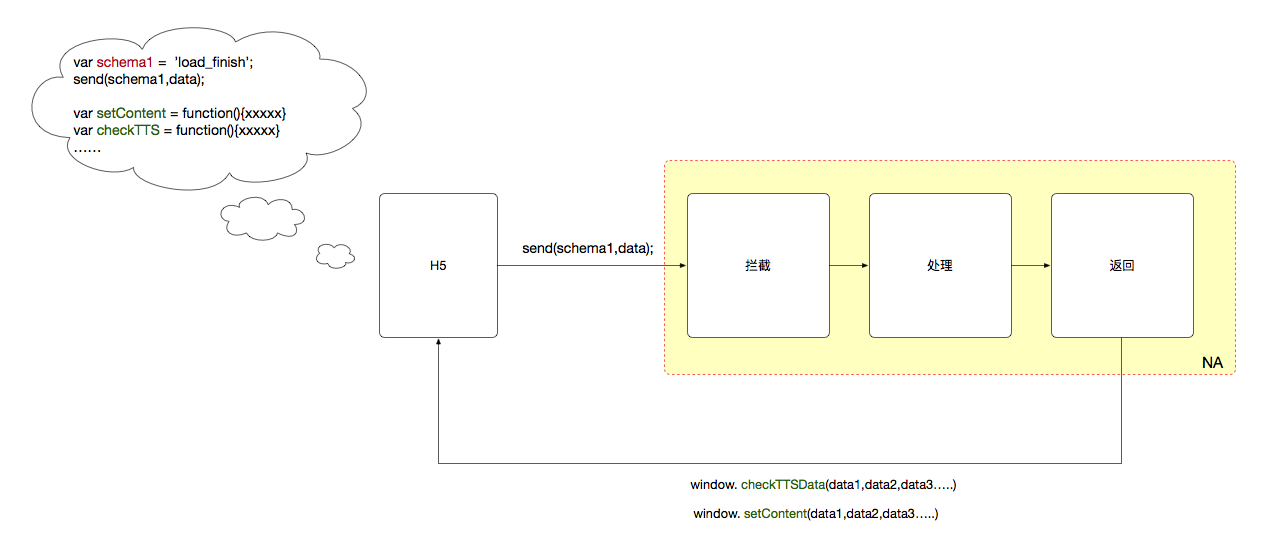
H5暴露全局的JS对象/方法，原生可以直接调用。



1. JavaScript 调用 Native

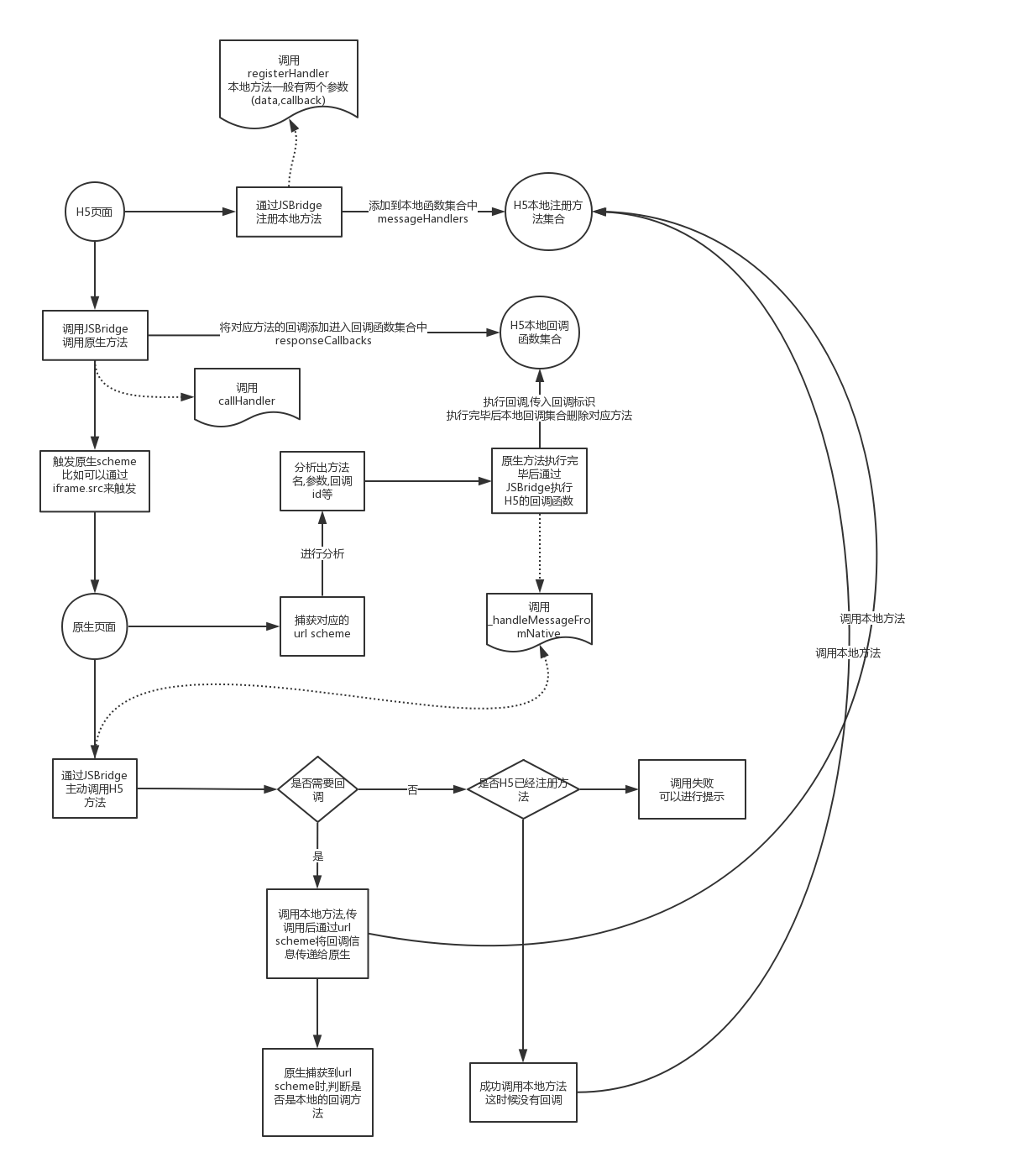
主要有以下两种方式：

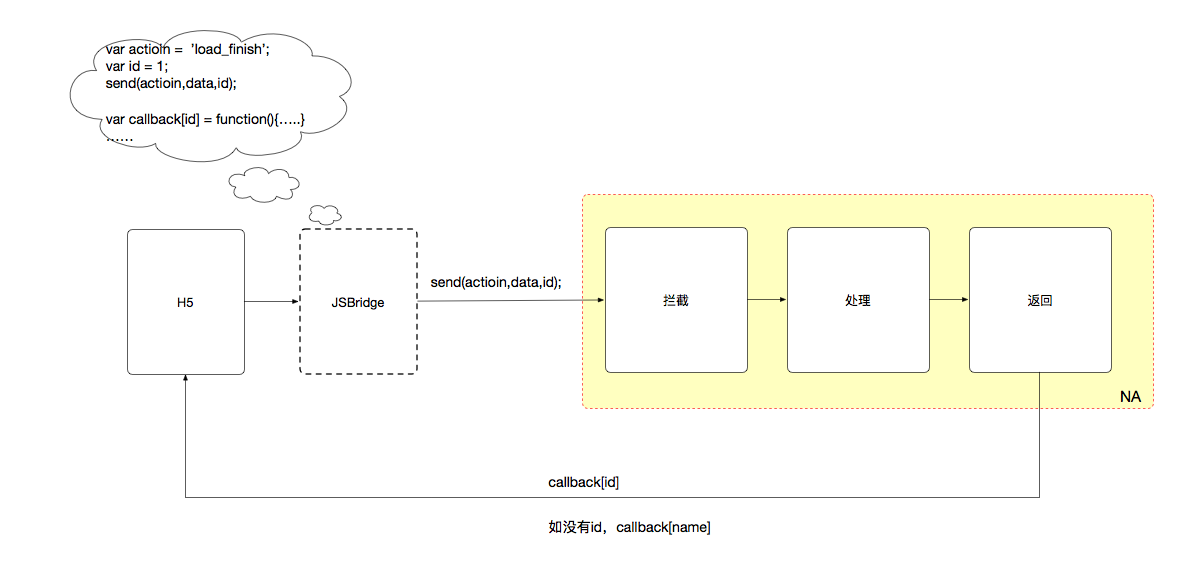
* 原生注入全局js对象（兼容性；安全性；不支持回调）
* 原生拦截js请求（不支持回调；对应关系不明确；全局函数冗杂；APP代码冗杂）



### jsBridge与原生通信原理

参考资料：[JSBridge的原理](https://juejin.im/post/5abca877f265da238155b6bc)





### jsBridge源码解析

// 这段代码是固定的，必须要放到js中

// 初始化jsBridge

export function setupWebViewJavascriptBridge (callback) {

// Android

if (isAndroid) {

if (window.WebViewJavascriptBridge) {

callback(window.WebViewJavascriptBridge)

} else {

document.addEventListener(

'WebViewJavascriptBridgeReady',

function () {

callback(window.WebViewJavascriptBridge)

},

false

)

}

} else if (isIos) {

// iOS

if (window.WebViewJavascriptBridge) {

return callback(window.WebViewJavascriptBridge)

}

if (window.WVJBCallbacks) {

return window.WVJBCallbacks.push(callback)

}

window.WVJBCallbacks = [callback]

var WVJBIframe = document.createElement('iframe')

WVJBIframe.style.display = 'none'

WVJBIframe.src = 'wvjbscheme://\_\_BRIDGE\_LOADED\_\_'

document.documentElement.appendChild(WVJBIframe)

setTimeout(function () {

document.documentElement.removeChild(WVJBIframe)

}, 0)

}

}

### 如何使用jsBridge

原理：JS通过callHandler调用Native方法，通过registerHandler注册事件供Native调用。

// js调用Native方法

window.WebViewJavascriptBridge.callHandler(name, params, callback)

// Native调用js方法

window.WebViewJavascriptBridge.registerHandler(name, callback)

在项目中使用jsBridge：

* 安装依赖

npm install hyapp-utils -D

* main.js引入

import hyapp from 'hyapp-utils'

Vue.use(hyapp.Tools) // 自动初始化jsBridge（大部分项目使用）

Vue.use(hyapp.ToolsManual) // 需项目中手动初始化jsBridge（金融超市使用）

* vue文件中使用

this.$appInvoked("appGetBundleId", {}, (rst) => {}) // 调用Native的appGetBundleId方法

this.$appGetInvoked('htmlNotifyLoginSuccess', () => {}) // 注册htmlNotifyLoginSuccess事件，原生登录成功后通知H5执行回调

* js文件中使用

window.vm = new Vue({

el: '#app',

router,

store,

components: { App },

template: '<App/>'

}).$mount('#app')

window.vm.$appInvoked("appGetBundleId", {}, (rst) => {})

window.vm.$appGetInvoked('htmlNotifyLoginSuccess', () => {})

### 问题及解决方案

1. WebViewJavascriptBridge未初始化（iOS）（框架已集成）

js调用Native及注册事件时，若jsBridge未初始化完成，则重新调用一次初始化方法。

appInvoked('appWVJBCompleted', { completed: true }) // jsBridge初始化成功通知Native

1. 频繁调用（安卓白屏）

原因：所有接口请求均调用 `appGetAjaxHeader` 方法获取header，同时请求大量接口频繁调用Native，导致部分请求丢失，得不到Native响应，导致后续回调无法执行。

解决：缓存需频繁请求的结果。

## 调试

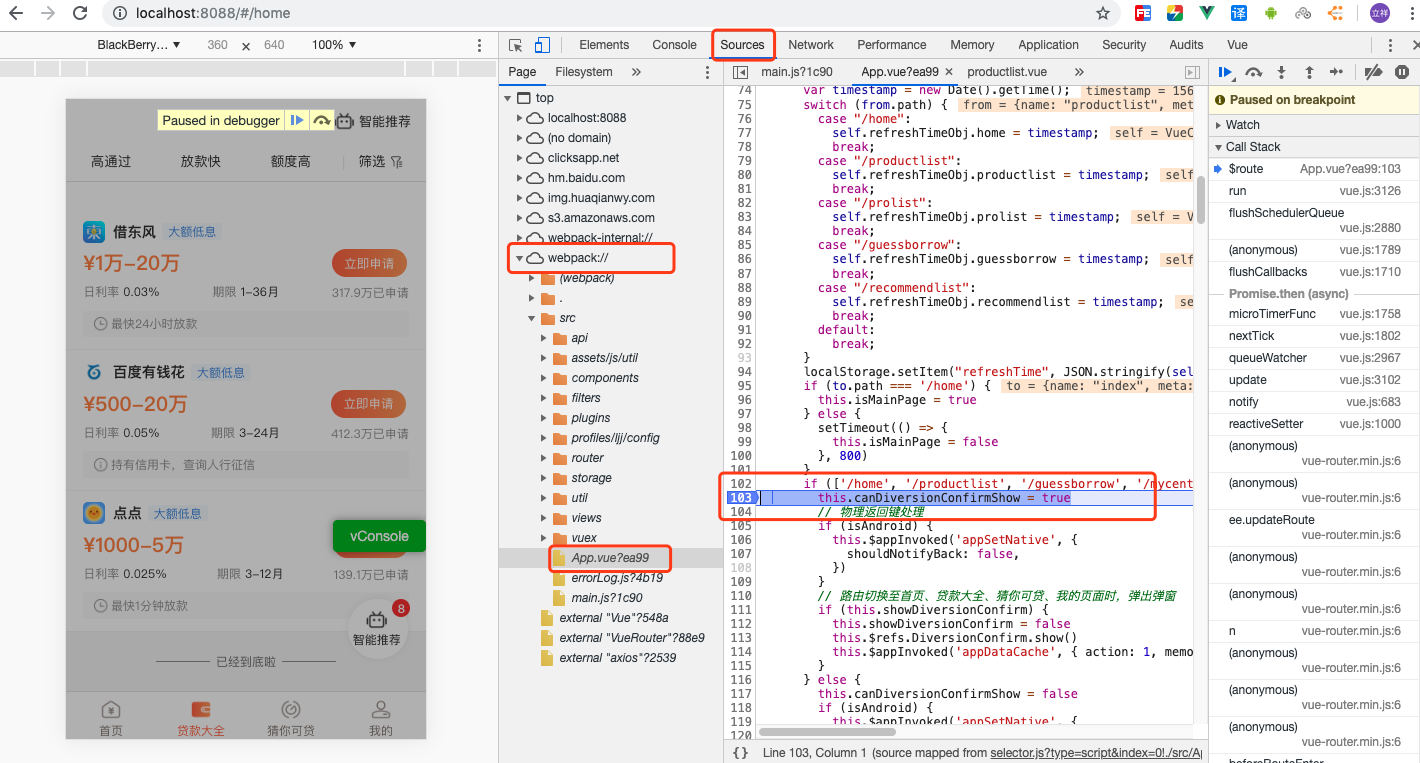
### 浏览器调试H5

1. 工具

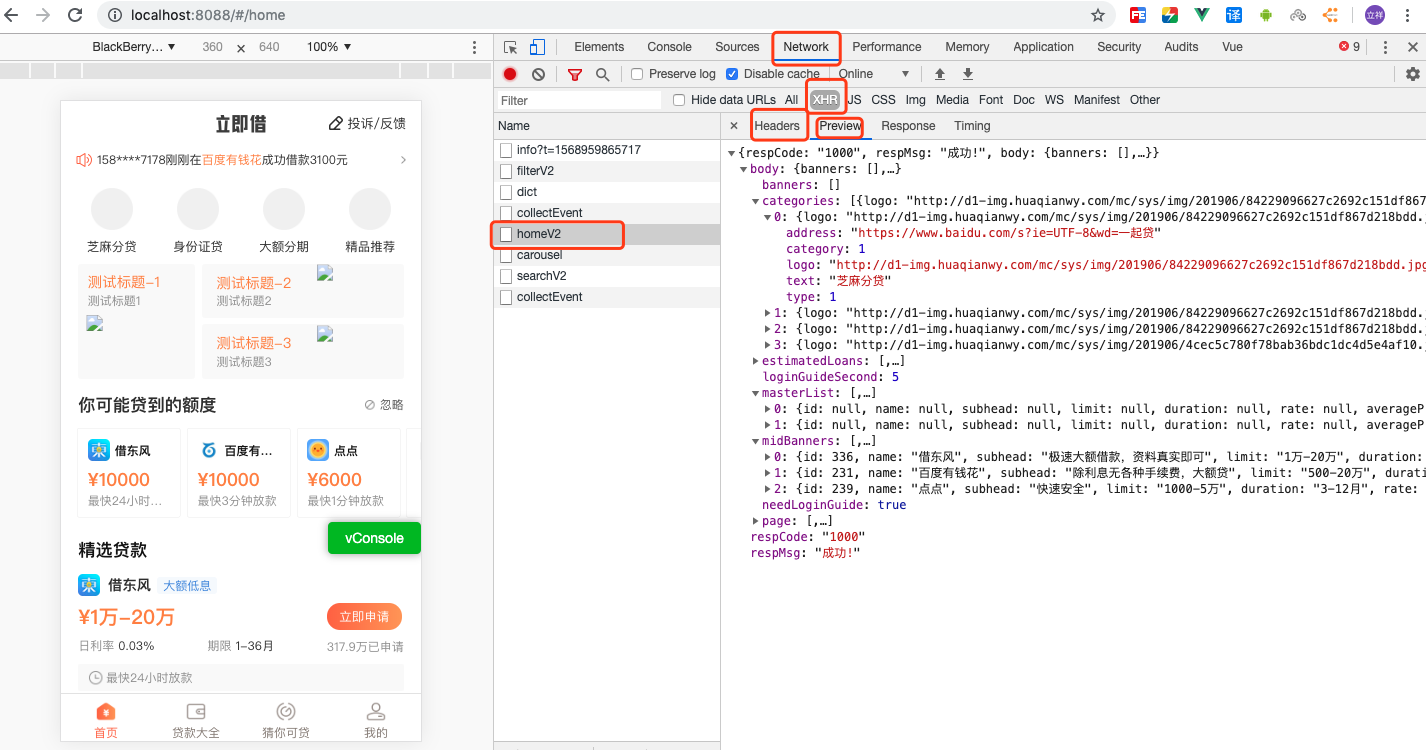
* 浏览器：Chrome
* 浏览器插件：Vue.js devtools

1. 方法

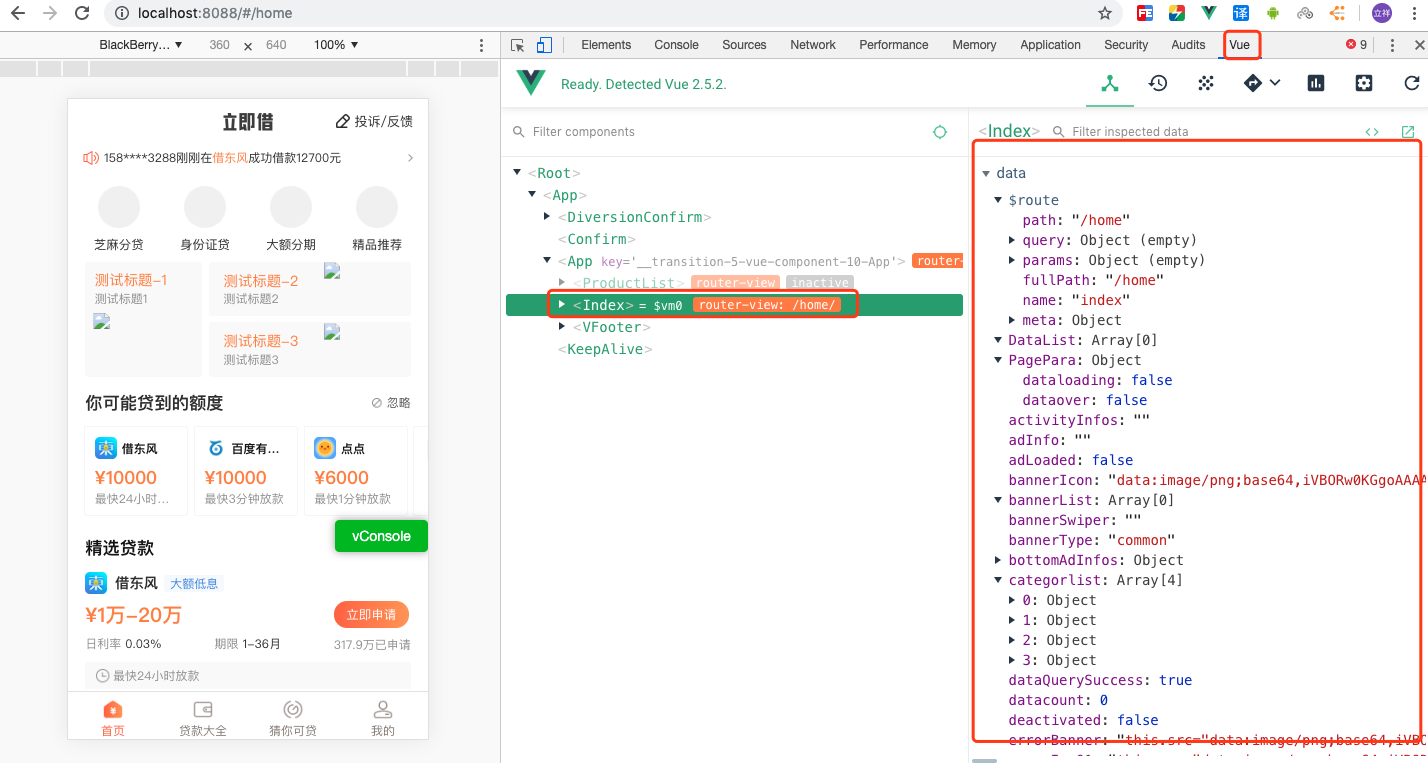
* 断点调试



* 接口调试



* Vue变量



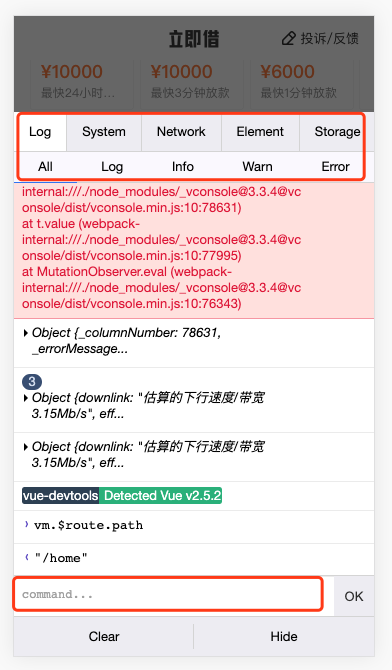
### vConsole调试

cnpm install vconsole // 安装vconsole依赖

// main.js引入vconsole

import Vconsole from 'vconsole'

let vConsole = new Vconsole()



### 抓包工具调试

工具：Charles或fiddler

主要针对接口进行抓包调试。主要用途：

* 查看请求

确认请求参数及返回值是否正确。

* Breakpoints

拦截请求，修改请求参数；拦截返回值，修改服务端返回数据。模拟不同状态及返回，验证页面表现是否正确。

* Map Local

把接口请求代理到本地json。直接修改本地json文件，即可实现返回不同的值，不必重复设置Breakpoints。

* Map Remote

把接口代理到其他服务。

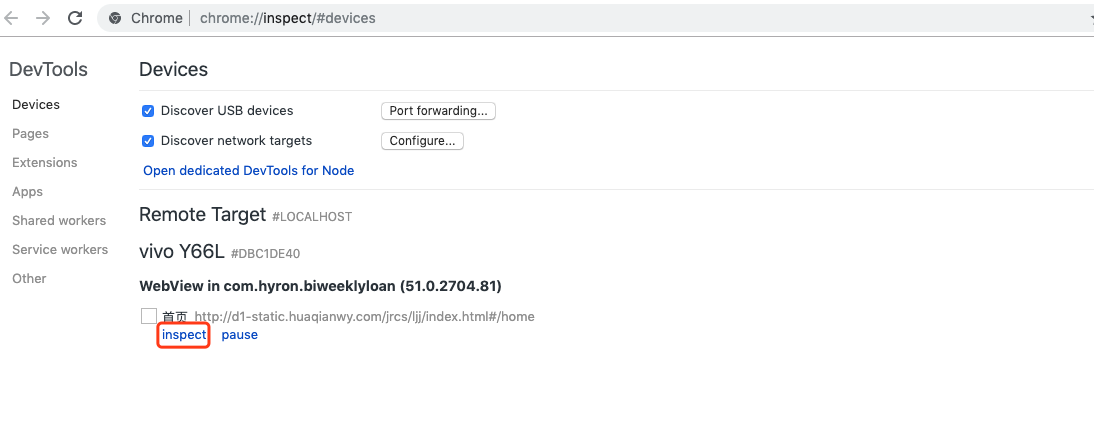
### Chrome调试Android App内H5页面（需翻墙）

1. 工具

* 浏览器：Chrome
* 浏览器插件：ADB（可选）
* 手机系统：Android

1. 方法

* 手机打开USB调试，数据线连接电脑
* Chrome打开 “chrome://inspect/#devices”或者点击浏览器插件ADB
* 手机打开要调试的页面，在浏览器点击“inspect”



* 在浏览器进行调试



## 适配方案

### rem适配方案

1. 兼容性：iOS 7、Android 4.2以上
2. 原理：rem是相对于HTML根元素为基准的相对单位，通过媒体查询为不同的屏幕尺寸，设置不同的html根元素的字体大小
3. 使用方法：

* 引入“\_mobile-mixin.scss”。（见附件rem适配）
* 配置$rem-base基准尺寸。（一般为750）
* 页面写法：rc(Number)。如：width: rc(32);表示基于750px设计稿，宽度为32px。

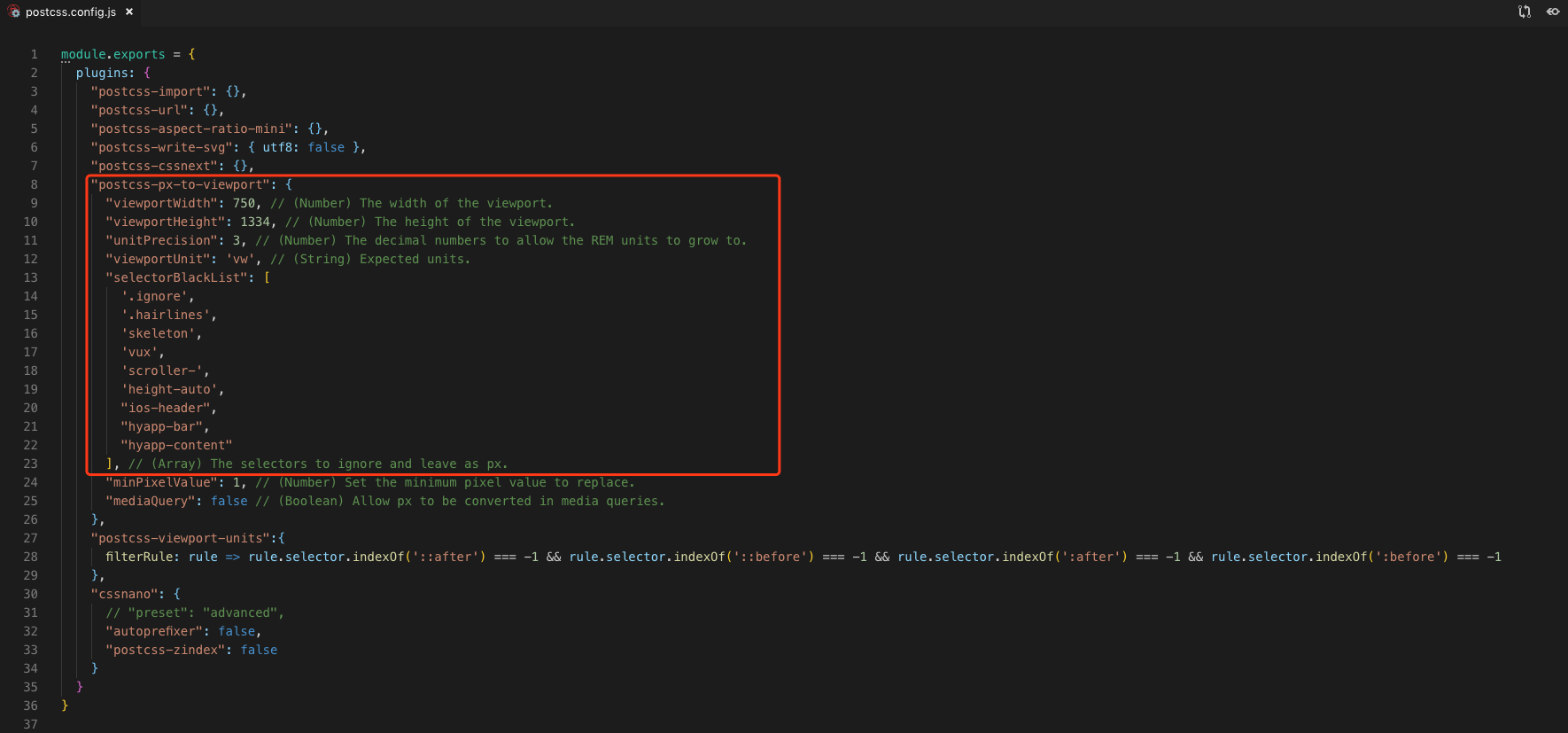
### vw适配方案

1. 兼容性：iOS 8、Android 4.4以上
2. 原理：1vw 等于视口宽度的1%
3. 使用方法：

* 安装依赖：

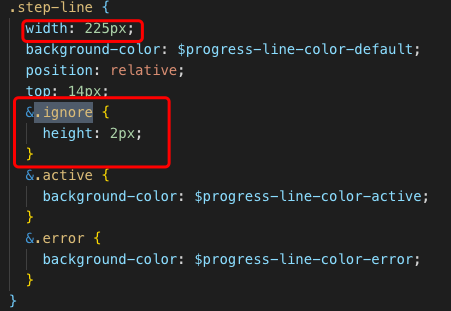
npm install postcss-px-to-viewport

* 修改配置文件postcss.config.js：



* 代码处理

根据设计提供的750px设计稿，直接写px即可。若不需将px转化为vm，可在selectorBlackList中配置黑名单，html文件中加上配置的class，sass中直接使用class。



1. DEMO：[好借款](https://wsdaikuan.2345.com/xfjr/loan/hjk/#/index)

## UI还原度注意事项

### 金融超市设计规范



### UI还原度注意事项

1. 注意页边距和模块间距
2. 字体颜色、大小、行高
3. 0.5px线
4. 按钮字体不居中问题（flex布局，line-height: normal）
5. 小图标变形

## 项目常用组件

### hyapp-utils

Git地址：<http://gitlab.jr.2345.net/finance_h5/h5_hyapp-utils>

主要功能：jsBridge、lodash、错误收集

### hyapp-ui

Git地址：<http://gitlab.jr.2345.net/finance_h5/h5_hyapp-ui>

Demo地址：<http://172.17.16.112:8888/examples/#/index>

### 说明文档

链接：<http://172.17.16.112:8888/examples/docs-examples/baseComponents/>

说明文档及hyapp-ui组件demo，均在hyapp-ui的git上维护。