步骤

```
# 打包成App
1.使用的路由必须是hash路由的方式
mode:hash
2.项目中的请求，必须是完整的路径，这个路径必须允许跨域
3.打包前需要设置打包出来的路径是相对路径
vue.config.js
module.exports={
 publicPath:'./'
}
4.执行打包
这样的话无需服务器浏览器就直接能打开页面
5.将项目包给Android和ios的开发人员

问题解决方案:
1、兼容齐刘海屏，在index.html的meta content添加viewport-fit=cover(不推荐)
2、使用safeAreaInsets插件，使用的条件是，使用的是WKWebview
 安装: npm install safe-area-insets --save
 引用: var safeAreaInsets = require('safe-area-insets');
 api: safeAreaInsets.support/top/left/right/bottom 算安全区域的大小,然后手动抬高
 监听安全区域的变化:
 function callback(style){
 console.log("change")
 }
 //add
 safeAreaInsets.onchange(callback)
 //remove
 safeAreaInsets.offchange(callback)
 因为最初safeAreaInsets为0，所以在created或者mounted
 safeAreaInsets.onchange(style=>{
 top= style.top;
 重新设置top，bottom的值
 })
# Android和ios的开发人员
IOS:
/*窗口宽高获取*/
CGTFloat width = self.view.frame.size.width;
/*创建一个显示网页的视图*/
WKWebView替代UIWebView
```

```
UIWebView *webview = [[UIWebView alloc] initWithFrame:CGRectMake(0,0,窗口宽,窗口高)];
/*消除弹簧效果*/
webview.scrollView.bounces = NO;
/*加载资源 得到的路径是绝对路径*/
NSString *path = [[NSBundle mainBundle] pathForResource:@"dist/index" ofType:@"html"];
/*也可以用网页地址*/
NSString *path = @"https://xxxxxxx";

/*构建请求，让webview加载请求*/
/*构建url方式一*/
NSURL *url = [NSURL URLWithString:path];
/*构建url方式二*/
NSURL *url = [[NSURL alloc] initFileURKWithPath:path];
NSURLRequest *request = [NSURLRequest requsetWithURL:url];
[webview loadRequest:request]
/*将webview覆盖到view上*/
[self.view addSubview:webview]
```

部署服务器上

```
1、路由可以使用hash也可以使用history
2、请求必须相对于环境的
3、打包路径(hash可以使用相对路径和绝对路径)(history使用绝对路径)
4、解决设计屏幕齐刘海的问题。npm i safe-area-insets -S
5、执行打包
6、项目包给后台，(history路由需要配置404重定向到index.html)
```

web网页与ios、android进行交互

```

# 原理
1、web网页调用ios和android
发送一个假请求，让原生开发拦截
比如<a href="jiade://captrueImage">拍照</a>
或者js window.location.href = 'jiade://captrueImage';
ios拦截到 jiade://captrueImage 就知道干什么了
经过对拦截到的字符串进行判断
使用发布订阅模式来调用原生功能
2、ios、android调用web网页
在window上挂靠一个函数
window.xxx = (value)=>{
```

```
window.xxx = (value)=>{
  // 操作
}

当原生需要传值给网页时，动态渲染
NSString *jsMethod = [NSString stringWithFormat:@"js代码"]
[self.webview evaluateJavaScripot:@"jsMethod" completionHandler:nil]


# 插件
webviewjavascriptbridge
https://github.com/marcuswestin/WebViewJavascriptBridge
pod安装或者下载拖入

#import "WebViewJavaScriptBridge.h"
// 申明
@property WebViewJavaScriptBridge* bridge;
// 实例化
self.bridge = [WebViewJavascriptBridge bridgeForWebView:webView];

//在桥上注册事件
[self.bridge registerHandler:@"注册的ios事件" handler:^(id data, WVJBResponseCallback responseCallback) {
  NSLog(@"ObjC Echo called with: %@", data);
  responseCallback(data);
}];
//调用事件
[self.bridge callHandler:@"调用js的事件名" data:nil responseCallback:^(id responseData) {
  NSLog(@"ObjC received response: %@", responseData);
}];

前端代码:
index.html中
function setupWebViewJavascriptBridge(callback) {
  if (window.WebViewJavascriptBridge) { return callback(WebViewJavascriptBridge); }
  if (window.WVJBCallbacks) { return window.WVJBCallbacks.push(callback); }
  window.WVJBCallbacks = [callback];
  var WVJBIframe = document.createElement('iframe');
  WVJBIframe.style.display = 'none';
  WVJBIframe.src = 'https://__bridge_loaded__';
```

```
    document.documentElement.appendChild(WVJBIframe);
    setTimeout(function() { d cument.documentElement.removeChild(WVJBIframe) }, 0)
  }

  当比如点击按钮触发时调用:
  setupWebViewJavascriptBridge(function(bridge) {
    ###!注意this的问题 function函数变成箭头函数
    #注册
    bridge.registerHandler('js的事件名', function(data, responseCallback) {
    console.log("JS Echo called with:", data)
    // 返回的值传入data
    responseCallback(data)
    })
    #调用
    bridge.callHandler('调用的ios事件名', {'key':'value'}, function
responseCallback(responseData) {
    console.log("JS received response:", responseData)
    })
  })
```