1. signin-up

当前组件的一个属性props，一般用来传递给子组件使用的，home应该是一个data或funciton。

属于vue中的父子通信的方法。

父传给子： props（也就是你这里的signin-up）

子传给父： event

remember it: props down, event up!

1. goBack

用来返回上一页

从 list 进入 update时先传递一个callback函数作为参数  
update在goback前 调用callback方法，在callback里面刷新list页面。

比如在 list 跳转到 update 时携带一个参数去下个页面

navigate('update ',{ callback: (data)=>{ console.log(data); // 打印值为：'回调参数' } });

在 update 页面 goback 之前，取到并回传参数，这样回传参数会重走 render 方法。

**const** {navigate,goBack,state} = **this**.props.navigation;

state.params.callback('回调参数');

goBack();

1. @click.prevent

@click.stop与@click.prevent

@click.stop 阻止事件冒泡

@click.prevent 阻止事件的默认行为，

<a href="http://www.baidu.com" @click.prevent="test4">百度一下</a> //阻止a标签跳转，仅执行函数test4

<form action="/xxx" @submit.prevent="test5"> //阻止表单提交，仅执行函数test5

<input type="submit" value="注册">

</form>

1. H5标签

用作文档中插图的图像，带有一个标题

<figure>

<img :src="imgBaseUrl + foodItem.image\_url">

<figcaption>{{foodItem.title}}</figcaption>

</figure>

1. Async
   1. async其实是ES7的才有的关键字，放在这里说，其实是和我们前面所说的Promise，Generator有很大关联的。async的意思是"异步"，顾名思义是有关异步操作有关的关键字。

昨天看了一篇vue的教程，作者用async/ await来发送异步请求，从服务端获取数据，代码很简洁，同时async/await 已经被标准化，是时候学习一下了。

　　先说一下async的用法，它作为一个关键字放到函数前面，用于表示函数是一个异步函数，因为async就是异步的意思， 异步函数也就意味着该函数的执行不会阻塞后面代码的执行。 写一个async 函数

async function timeout() {

　　return 'hello world';

}

　　 语法很简单，就是在函数前面加上async 关键字，来表示它是异步的，那怎么调用呢？async 函数也是函数，平时我们怎么使用函数就怎么使用它，直接加括号调用就可以了，为了表示它没有阻塞它后面代码的执行，我们在async 函数调用之后加一句console.log;

async function timeout() {

return 'hello world'

}

timeout();

console.log('虽然在后面，但是我先执行');

参考：http://www.cnblogs.com/leungUwah/p/7932912.html

1. await

[await使用中的阻塞和并发](https://www.cnblogs.com/manupstairs/p/3532879.html)

好吧，不加点陈述不让发首页。那我们来陈述一下本篇提到的问题和对应的方法。

在.NET4.5中，我们可以配合使用async和await两个关键字，来以写同步代码的方式，实现异步的操作。

好处我目前看来有两点：

1.不会阻塞UI线程。一旦UI线程不能及时响应，会极大的影响用户体验，这点在手机和平板的APP上尤为重要。

2.代码简洁。

* 相对基于event的异步方式，在多次回调的情况下（比如需要多次调web service，且后续调用基于前次调用的结果）特别明显。可以将多个+=Completed方法合并到一起。
* 相对于Begin/End的异步方式，避免了N重且不能对齐的大括号。

在同一个方法里存在多个await的情况下，如后续Async方法无需等待之前的Aysnc方法返回的结果，会隐式的以并行方式来运行后面的Async方法。

值得注意的是错误的写法会导致非预期的阻塞，下文会以简单的例子来讨论在使用await的情况下，怎样实现多个Task的并发执行。

static async Task Delay3000Async()

{

await Task.Delay(3000);

Console.WriteLine(3000);

Console.WriteLine(DateTime.Now);

}

1. FastClick

原理: 在检测到touchend事件的时候，会通过DOM自定义事件立即出发模拟一个click事件，

并把浏览器在300ms之后真正的click事件阻止掉

缺点: 脚本相对较大

没使用jquery得时候

if ('addEventListener' in document) {

document.addEventListener('DOMContentLoaded', function() {

FastClick.attach(document.body);

}, false);

}

??// 安装? ? npm install fastclick -S?

? // 引入? ? import FastClick from 'fastclick'?

? // 使用? ? FastClick.attach(document.body);? ?

为啥要使用

移动设备上的浏览器默认会在用户点击屏幕大约延迟300毫秒后才会触发点击事件，

这是为了检查用户是否在做双击。为了能够立即响应用户的点击事件，才有了FastClick。

1. Aside

<aside> 的内容可用作文章的侧栏。

1. Fetch发送数据

# Fetch发送数据

使用Fetch发送也很简单，只需要配置三个参数。

fetch('some-url', options);

第一个参数是设置请求方法（如post、put或del），Fetch会自动设置方法为get。

第二个参数是设置头部。因为一般使用JSON数据格式，所以设置ContentType为application/json。

第三个参数是设置包含JSON内容的主体。因为JSON内容是必须的，所以当设置主体时会调用JSON.stringify。

实践中，post请求会像下面这样：

let content = {some: 'content'};

// The actual fetch request

fetch('some-url', {

method: 'post',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(content)

})

// .then()...

1. scrollToElement（el,time,offsetX,offset,easing）

el选项可以是选择器或者元素，如果是选择器就会在id='scroller'元素下选择该元素。iScroll会移动到该元素具有的left/top处

time选项表示transition duration，是可选的

offsetX,offsetY采用像素定义偏移，因此你可以滚动到一个元素的时候加上这个偏移量。如果你设置为true那么就会在视口居中

easing表示动画曲线

1. nextClick

在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中

在created()钩子函数执行的时候DOM 其实并未进行任何渲染，而此时进行DOM操作无异于徒劳，所以此处一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。

与之对应的就是mounted()钩子函数，因为该钩子函数执行时所有的DOM挂载和渲染都已完成，此时在该钩子函数中进行任何DOM操作都不会有问题 。

在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的DOM结构的时候，这个操作都应该放进Vue.nextTick()的回调函数中。

具体原因在Vue的官方文档中详细解释：

Vue 异步执行 DOM 更新。只要观察到数据变化，Vue 将开启一个队列，并缓冲在同一事件循环中发生的所有数据改变。

如果同一个 watcher 被多次触发，只会被推入到队列中一次。这种在缓冲时去除重复数据对于避免不必要的计算和 DOM 操作上非常重要。然后，在下一个的事件循环 “tick”中，Vue 刷新队列并执行实际 (已去重的) 工作。Vue 在内部尝试对异步队列使用原生的 Promise.then 和MessageChannel，如果执行环境不支持，会采用 setTimeou t(fn, 0)代替。

例如，当你设置vm.someData = 'new value'，该组件不会立即重新渲染。当刷新队列时，组件会在事件循环队列清空时的下一个“tick”更新。多数情况我们不需要关 心这个过程，但是如果你想在 DOM 状态更新后做点什么，这就可能会有些棘手。虽然 Vue.js 通常鼓励开发人员沿着“数据驱动”的方式思考，避免直接接触 DOM，但是有时我 们确实要这么做。为了在数据变化之后等待 Vue 完成更新 DOM ，可以在数据变化之后立即使用Vue.nextTick(callback) 。这样回调函数在 DOM 更新完成后就会调用。

1. mixin

vue中提供了一种混合机制--mixins，用来更高效的实现组件内容的复用。最开始我一度认为这个和组件好像没啥区别。。后来发现错了。下面我们来看看mixins和普通情况下引入组件有什么区别？

组件在引用之后相当于在父组件内开辟了一块单独的空间，来根据父组件props过来的值进行相应的操作，单本质上两者还是泾渭分明，相对独立。

而mixins则是在引入组件之后，则是将组件内部的内容如data等方法、method等属性与父组件相应内容进行合并。相当于在引入后，父组件的各种属性方法都被扩充了。

单纯组件引用：

父组件 + 子组件 >>> 父组件 + 子组件

mixins：

父组件 + 子组件 >>> new父组件

值得注意的是，在使用mixins时，父组件和子组件同时拥有着子组件内的各种属性方法，但这并不意味着他们同时共享、同时处理这些变量，两者之间除了合并，是不会进行任何通信的。最开始看到mixins的时候，天真的我似乎看到了一种向下的类似vuex的数据共享方案，心情十分激动啊。但是仔细一研究官方api和一些技术博客，才发现自己。。。天真。哈哈

有个很强大的博客：<http://www.deboy.cn/Vue-mixins-advance-tips.html>

可以参考

1. Markdown基本语法

Markdown是一种纯文本格式的标记语言。通过简单的标记语法，它可以使普通文本内容具有一定的格式。

相比WYSIWYG编辑器

**优点：**  
1、因为是纯文本，所以只要支持Markdown的地方都能获得一样的编辑效果，可以让作者摆脱排版的困扰，专心写作。  
2、操作简单。比如:WYSIWYG编辑时标记个标题，先选中内容，再点击导航栏的标题按钮，选择几级标题。要三个步骤。而Markdown只需要在标题内容前加#即可

**缺点：**  
1、需要记一些语法（当然，是很简单。五分钟学会）。  
2、有些平台不支持Markdown编辑模式。

1. Object.keys方法之详解

* 传入对象，返回属性名

var obj = {'a':'123','b':'345'};

console.log(Object.keys(obj)); //['a','b']

var obj1 = { 100: "a", 2: "b", 7: "c"};

console.log(Object.keys(obj1)); // console: ["2", "7", "100"]

var obj2 = Object.create({}, { getFoo : { value : function () { return this.foo } } });

obj2.foo = 1;

console.log(Object.keys(obj2)); // console: ["foo"]

* 传入字符串，返回索引

var str = 'ab1234';

console.log(Object.keys(obj)); *//[0,1,2,3,4,5]*

* 构造函数 返回空数组或者属性名

function Pasta(name, age, gender) {

this.name = name;

this.age = age;

this.gender = gender;

this.toString = function () {

return (this.name + ", " + this.age + ", " + this.gender);

}

}

console.log(Object.keys(Pasta)); //console: []

var spaghetti = new Pasta("Tom", 20, "male");

console.log(Object.keys(spaghetti)); //console: ["name", "age", "gender", "toString"]

* 数组 返回索引
* var arr = ["a", "b", "c"];
* console.log(Object.keys(arr)); *// console: ["0", "1", "2"]*