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()...

明天又是充满希望的一天