32 模拟真实APP完成消息推送

更新时间：2019-09-30 10:15:41

> 读一本好书，就是和许多高尚的人谈话。
>
> ——歌德

消息推送，顾名思义就是你在手机上收到的某个 APP 的消息推送，相较于移动端 Native 应用，web 应用是缺少这一项常用的功能。而借助 PWA 的 Push 特性，就是用户在打开浏览器时，不需要进入特定的网站，就能收到该网站推送而来的消息，例如：新评论，新动态等等，而借助于 Android 的 Chrome，我们可以实现在用户不打开任何浏览器或者应用的情况下，收到我们项目的推送，就像一个真实的手机推送。

本章节完整源代码地址，大家可以事先浏览一下：

Github-registerServiceWorker.js
Github-sw-push.js
Github-push.js

什么是Web Push

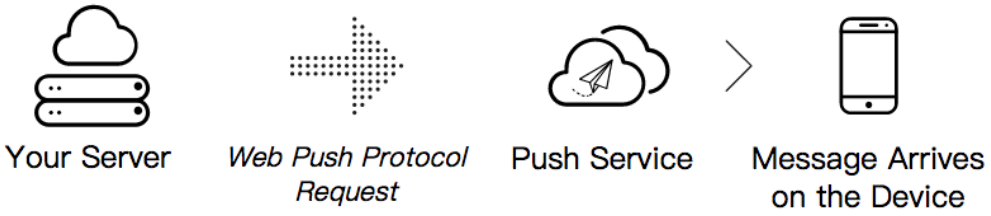
Web Push是一个基于客户端，服务端和推送服务器三者组成的一种流程规范，可以分为三个步骤：

1. 客户端完成请求订阅一个用户的逻辑。
2. 服务端调用遵从 web push 协议的接口，传送消息推送（push message）到推送服务器（该服务器由浏览器决定，开发者所能做的只有控制发送的数据）。
3. 推送服务器将该消息推送至对应的浏览器，用户收到该推送。

下图展示了一个用户订阅的过程：

所谓用户订阅，就是说我想要收到你的网站或者你的 APP 的推送通知，我就需要告诉你我是谁，我要把我的标识传给你，否则你怎么知道要给我推送。

下图展示了服务端收到用户订阅请求后如何推送：



1. 首先，在你项目的后台(Your Server)要存储一下用户订阅时传给你的标识。
2. 在后台需要给你推送的时候，找到这个标识，然后联系推送服务器(Push Service)将内容和标识传给推送服务，然后让推送服务将消息推送给用户端。（iOS和Android各自有自己的推送服务器，这个和操作系统相关）。
3. 这里就有一个约定，用户的标识，要和推送服务达成一致，例如使用Chrome浏览器，那么推送服务就是谷歌的推送服务(FCM)。

Web Push前端逻辑

那么对应到代码中，我们如何获取到用户标识呢，这就要借助与 Service Worker了(基于 Web Push 的推送和通知相关全部是基于Service Worker，现在知道有多管用了把)。

上一节前端项目创建的registerServiceWorker.js中，增加如下代码：

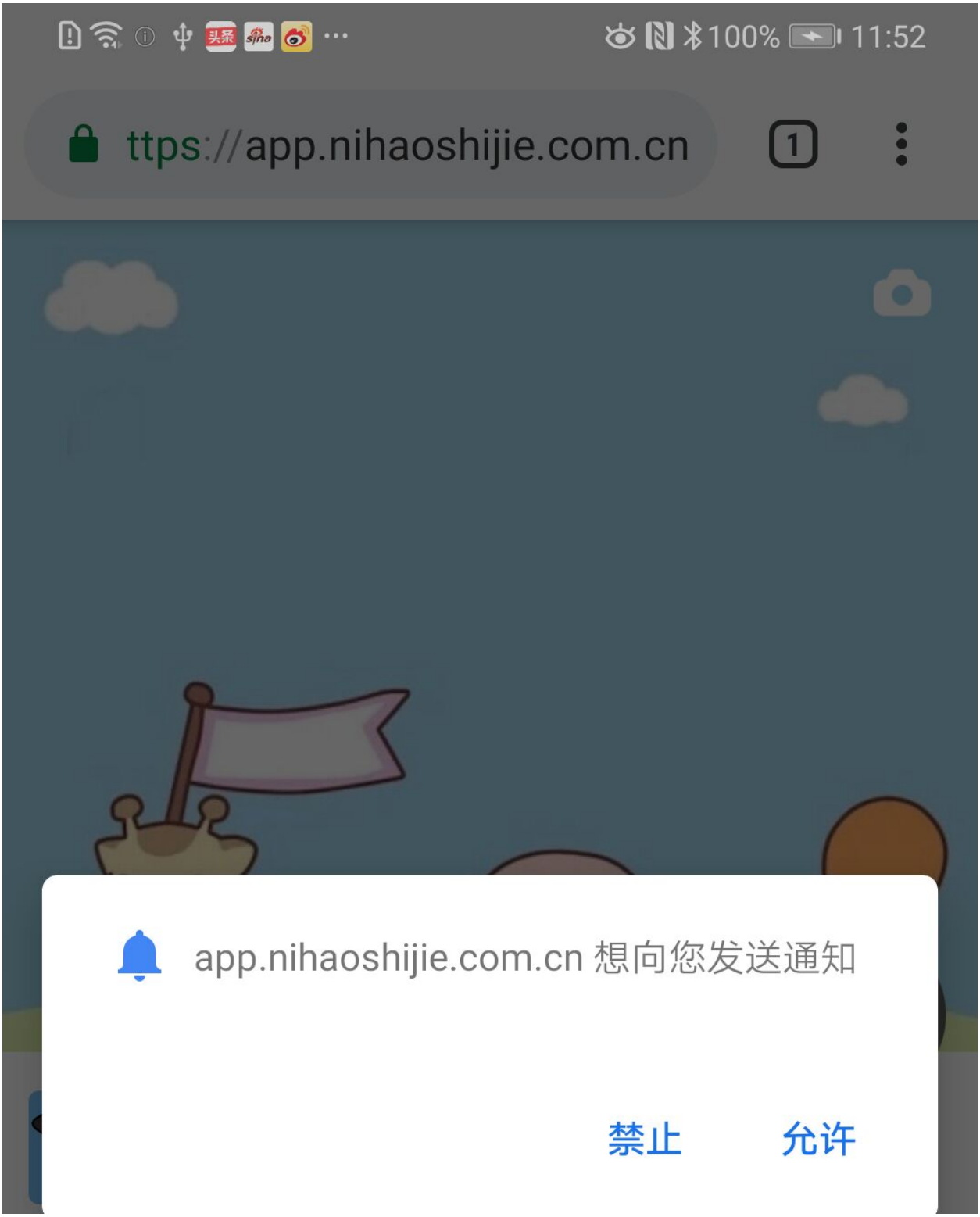
```
navigator.serviceWorker.ready.then((registration) => {
 //publicKey和后台的publicKey对应保持一致

 const publicKey = 'BAWz0cMW0hw4yYH-DwPrwylVU0ee3f4oMrt6YLGPaDn3k5MNZtqjpYwUkD7nLz3AJwtgo-kZhB_1pbcmzyTVAxA';//web-push定义的客户端的公钥，用来和后端的web-push对应

 //获取订阅请求（浏览器会弹出一个确认框，用户是否同意消息推送）
 try {

    if (window.PushManager) {
       registration.pushManager.getSubscription().then(subscription => {

          // 如果用户没有订阅 并且是一个登录用户
          if (subscription && window.localStorage.getItem('cuser')) {

            const subscription = registration.pushManager.subscribe({
              userVisibleOnly: true,//表明该推送是否需要显性地展示给用户，即推送时是否会有消息提醒。如果没有消息提醒就表明是进行“静默”推送。在Chrome中，必须要将其设置为true，否则浏览器就会在控制台报错
              applicationServerKey: urlBase64ToUint8Array(publicKey)//web-push定义的客户端的公钥，用来和后端的web-push对应
            })

            //用户同意
            .then(function(subscription) {
              console.log(subscription)
              alert(subscription)
              if (subscription && subscription.endpoint) {

                // 存入数据库
                let resp = service.post('users/addsubscription', {
                  subscription: JSON.stringify(subscription)
                })
              }
            })

            //用户不同意或者生成失败
            .catch(function(err) {
               alert(1)
               alert(err)
               console.log("No it didn't. This happened: ", err)
             });



          } else {//用户已经订阅过
            alert('subscriptioned')
            console.log("You have subscribed our notification");
          }
       });
    }


   }catch(e){
     alert(e)
     console.log(e)
   }


});
```

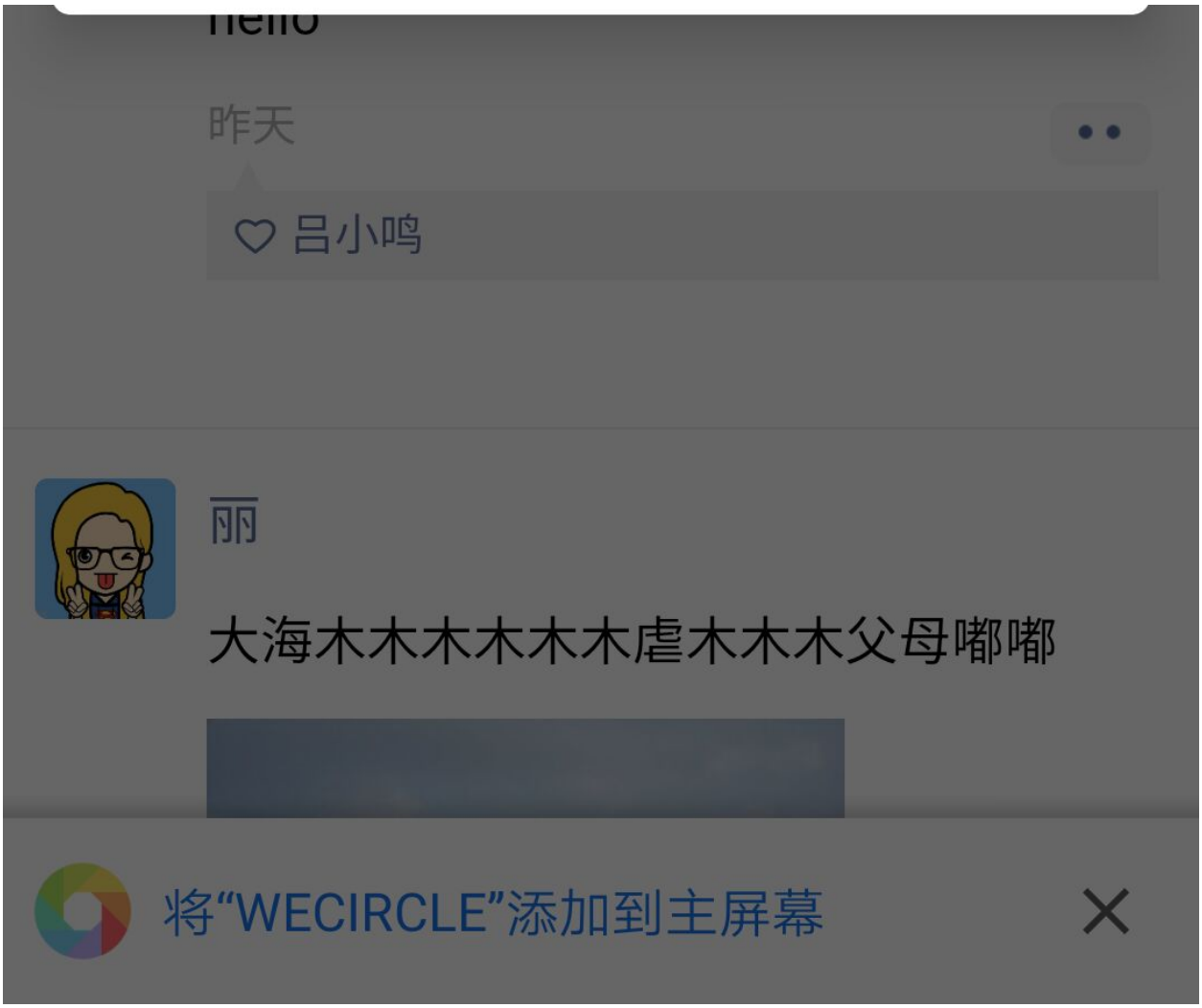
看见这么大一坨代码懵逼了么，下面给大家解释一些，在 Service Worker 准备就绪之时，我们就可以开始获取用户标识了。

1. 代码通过 `registration.pushManager.getSubscription()` 先要确定用户是否已经订阅过，就是是否已经获取过标识，然后得到的 `subscription` 就是我们要的标识。（后面将 `subscription` 代替标识）。
2. 如果用户没有订阅过，通过 `registration.pushManager.subscribe()` 可以拿到 `subscription`，在调用这个方法的时

候，浏览器就会询问用户是否接受订阅，也就是会弹一个框：

如果是在手机端，前提是使用Android的Chrome，会收到这样的提示，如图：





3. 当我们点击同意，就会获取到`subscription`，然后通过 service 发请求到后台存储。这个`subscription`其实是一个对象，长这样：

```
{
  "endpoint": "https://fcm.googleapis.com/fcm/send/eekuJ6272vI:APA91bEdnUY1cpyTfRFVMUJBx2CNQdA6Qg2FwP0oPibqltHxgZz__2ggmgSpE5bGRol81cginuT2clRDuqmmmtiqgYiG_WXQtvw83Mv41bJxJj89y1rglr5mvyiyHpBRmI_y07uq1pVlc",
  "expirationTime": null,
  "keys": {
    "p256dh": "BLcOaaco6_dIjflo3uiR6nDqERiCUwOuVT1mD5W45V99hvuYoqJxJZzKrKLsgE16zI_DA7o5PXXa8HVZvNz8PHg1",
    "auth": "vRqwuyij2AR9qkzUOwP3Pwx"
  }
}
```

对于每一个客户端来说`subscription`都是唯一的。

到此我们就完成了前端关于用户订阅的逻辑，那么接下来让我们看看后端的逻辑。

Web Push后端逻辑

首先存储`subscription`我们需要新建一张表`Subscription`。
在后端项目的models文件夹下新建`Subscription.js`代码如下：

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var SubscriptionSchema = new mongoose.Schema({
  subscription: { type: String,required:true },
  userid:{ type: String, unique: true },//注意这里不用ref外键
  update: { type: Date, default: Date.now },
  create: { type: Date, default: Date.now },
},{timestamps:{createdAt: 'create',updatedAt:'update'}});

module.exports = mongoose.model('Subscription', SubscriptionSchema);
```

subscription:字段是一个字符串，我们会将前端传的对象`JSON.stringify()`一下。

userid：这个字段是标识那个用户，采用`unique:true`表明唯一性，这里不用 ref 外键是为了可空，为了后续可能会给没登录过的用户也推送一些消息，当然我们项目只会在发送聊天消息时推送，就要求用户必须是登录过的。

在后端项目的 routes 文件夹下的users.js文件的路由里面新增一个方法：

```
/*
* 添加订阅信息
*/
router.post('/addsubscription', async (req, res, next)=> {
  var userid = req.user ? req.user._id : '';

  try {
    var result = await Subscription.create({
      subscription: req.body.subscription,
      userid:userid
    })

    res.json({
      code:0,
      data:result
    })
  }catch(e){
    // console.log(e)
    res.json({
      code:0,
      data: e.errmsg.indexOf('dup key') ? 'has scription' : e.errmsg // 说明用户已经订阅过
    })
  }

});
```

上面代码通过`Subscription.create()`就完成了对一个`subscription`的存储。当存储时发现 userid 已经有过，就会抛出一个错误，就说明这个用户已经订阅过了。

然后，在后端项目的 utils 文件夹下新建`push.js`工具方法，来实现后台推送逻辑：

首先安装web-push，是一个基于Node.js的web-push封装，当然还有基于Java或者Php的：

```
npm install web-push --save
```

然后在`push.js`新增代码，首先需要生成`vapidKeys`，这个就是我们在前端用的那个key，要和这里保持一致，代码如下：

```
var vapidKeys = webpush.generateVAPIDKeys();
```

只需要生成一次，然后设置一下，得到之后后面一只用这个就可以，代码如下：

```
var vapidKeys = { publicKey:
  'BAWz0cMW0hw4yYH-DwPrwylVU0ee3f4oMrt6YLGPaDn3k5MNZ1tqjpYwUkD7nLz3AJwtgo-kZhB_1pbcmzyTVAxA',
 privateKey: 'BJ_V2wtPYaVCl7Ef2GAkVxXB2ft9cTgw-b5lM2ggc8lo' };

 webpush.setVapidDetails(
 'mailto:example@yourdomain.org',//不需要邮箱通知的话这里可以随意填
 vapidKeys.publicKey,
 vapidKeys.privateKey
);
```

完成这些设置之后，就可以和 Push Service 通信，来实现推送了，代码如下：

```
module.exports = async function(userid,data){

 //国内使用的话，需要设置代理才行
 var option = {
  proxy: 'http://113.10.152.92:3128'   //http://www.freeproxylists.net/zh/hk.html
 }

 //从数据库中找到subscription
 var obj = await Subscription.findOne({
  userid: userid
 }).exec();
 console.log('检查是否有可推送的subscription')
 console.log(obj)
 if (obj && obj.subscription) {
  console.log('找到subscription 可以推送')
  // 调用webpush的sendNotification来发起推送通知
  webpush.sendNotification(JSON.parse(obj.subscription), JSON.stringify(data),option).catch(function(err) {
   console.error(err)
  });
 }

}
```

这里解释一下，由于我们使用的推送服务器是基于谷歌的FCM，这个服务在国内是无法使用的（或者说有时可用有时不可用），所以我们需要设置一个代理，当然网上有很多免费的国外代理，可以在http://www.freeproxylists.net/zh/hk.html找找，如果想要稳定一点的可以掏钱买一个 VPN 服务。

Web Notification

前面说了那么多，好像全是Push相关的，那么对于我们的 APP 来说，如和在收到 Push 之后提示呢，这就涉及到Notification相关的API了。

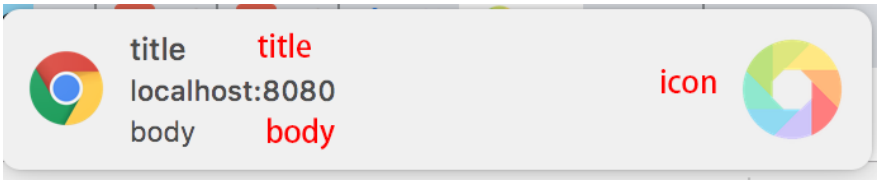
接下来，在 Service Worker 里注册 `push` 事件来接收 `push` 请求。
在前端项目的 public 目录下新建一个 `sw-push.js`：

```
// 添加service worker对push的监听
self.addEventListener('push', function (e) {
  var data = e.data
  if (e.data) {
    data = data.json()
      e.waitUntil(
        self.registration.showNotification(data.title, {
          body: data.body || '',
          icon: data.img || "https://app.nihaoshijie.com.cn/img/icons/apple-touch-icon-180x180-1-touming.png",
          actions: [{
              action: 'go-in',
              title: '进入程序'
          }]
        })
      );
    ;
  } else {
    console.log('push没有任何数据')
  }
})
```

当浏览器收到推送通知时，就会进入这个事件里，我们通过`self.registration.showNotification()`

- title：消息的标题，属于必传的值。
- body：消息的实体，可以不传。
- icon：配置消息的图片，会出现在消息里面。
- actions：配置消息的操作项，在结合`notificationclick`事件可以实现消息的点击交互。

就可以弹出一个通知框，前提是你在之前的通知允许中点击了确定：



在手机端是这个样子：

如果想要在手机端或者是PC端收到提示，前端必须满足下面条件：

1. PC端的chrome要可以翻墙，也就是能够使用谷歌相关的服务。
2. 手机端的chrome要内置了chrome服务(GMS)，据笔者实验国内的华为，vivo，小米系列基本是没有内置chrome服务的，而Nexus系列的手机则可以正常使用。能够连接Google Play则代表可以。

你要问我为什么这么多条件？原因是web-push是基于谷歌的FCM(云消息机制)实现的推送，而FCM包含在GMS里面。知道谷歌禁止国外的华为手机使用谷歌服务有多大影响了吧，连消息都收不到啊。

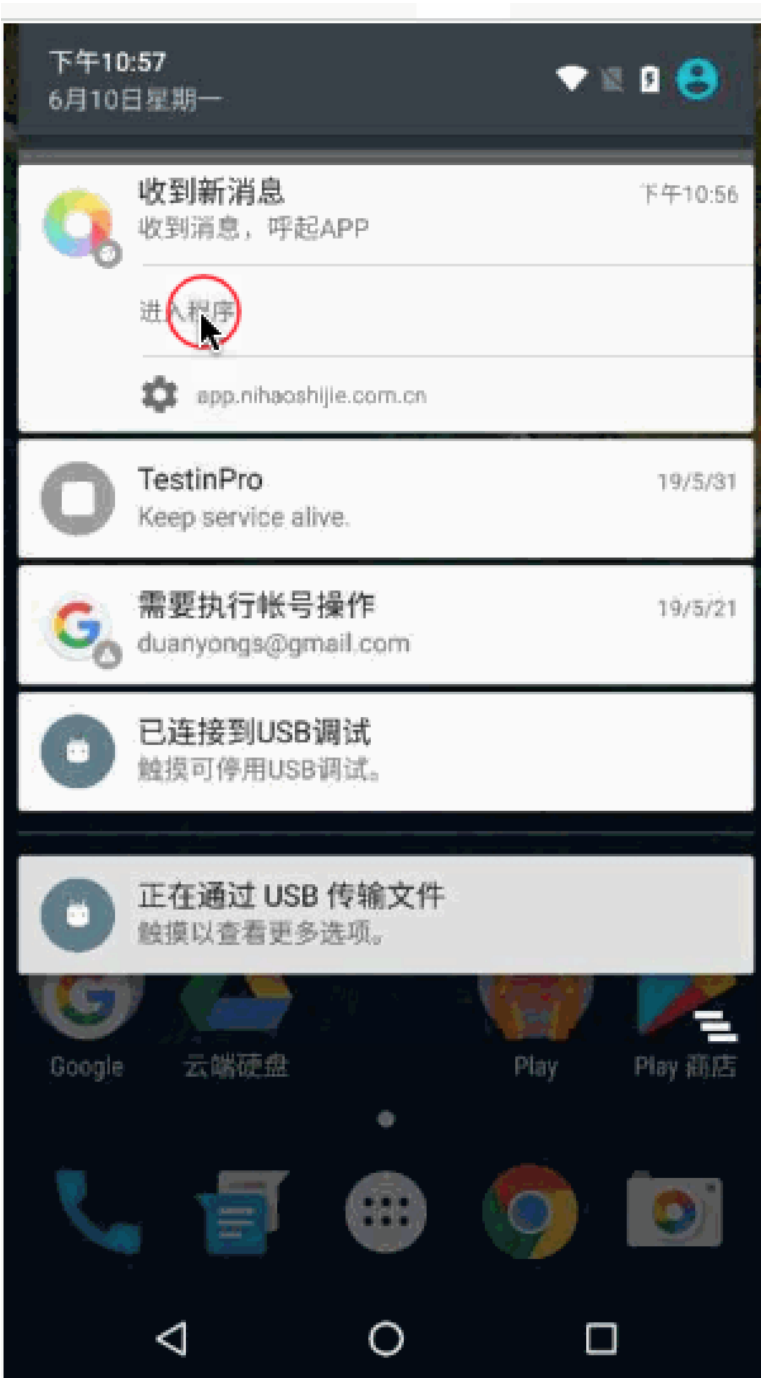
在 Notification 添加点击事件实现完整消息流程，在 `sw-push.js` 增加如下代码：

```
self.addEventListener('notificationclick', function (e) {
    var action = e.action;
    e.waitUntil(
        // 获取所有clients
        self.clients.matchAll().then(function (clientList) {
          if (clientList.length > 0) {
            return clientList[0].focus();
          }

          if (action === 'go-in') {
            return self.clients.openWindow('https://app.nihaoshijie.com.cn/index.html#/mypage');
          }

        })
    );
    e.notification.close();
});
```

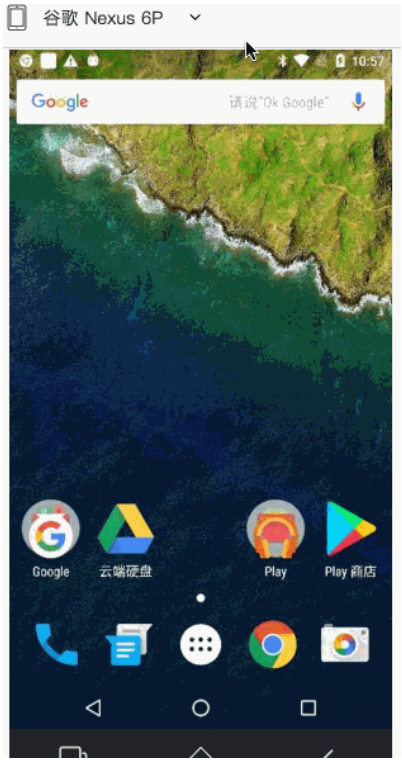
在 self.registration.showNotification() 中，我们传了一个action，这里就对应了消息弹出时，有选项可以选择：

在PC端：

在手机端：



根据上节我们讲的离线APP，收到消息通知的条件不限于你必须打开着APP，经过验证，即使APP已经关闭，同样可以收到推送消息，调用`self.clients.openWindow()`可以将APP呼起来，可以看下图的流程：



最后，我们的`sw-push.js`需要配置在 offline-plugin 插件里面进行合并，最终对于Service Worker 只有一个`sw.js`，在`vue.config.js`里修改代码，如下：

```
...
  ServiceWorker: {
    events: true,
    // push事件逻辑写在另外一个文件里面
    entry: './public/sw-push.js'
  },
...
```

小节

本章节主要讲解了使用Web Push来实现消息推送，并解释了其中的原理和具体的实现方法。
相关技术点：

1. Web Push的概念和基本流程。
2. 在Node.js里使用Web Push，并推送给前端。
3. Web Notification的API及相关的配置来提示消息。

本章节完整源代码地址：

Github-registerServiceWorker.js
Github-sw-push.js
Github-push.js

}