微信公众号开发

课堂目标

- 本地环境可以调试公众号接口
- 能够通过程序对公众号消息进行应答
- 能够调用服务器端接口对接 - 比如：查看关注者列表

> 参考资料
>
> 微信开发者工具说明 https://mp.weixin.qq.com/wiki?t=resource/res_main&id=mp1455784140
>
> npm 库 wechat和wechat-api，以及微信开发者文档。

知识要点

1. 开发环境搭建

- 注册微信订阅号
- 注册小程序测试号
- 安装sunny-ngrok实现外网的映射

  - 下载地址 https://www.ngrok.cc/

2. 客服消息 - (你问我答) 完成

- npm库： https://github.com/node-webot/co-wechat
- 开通公众号测试账号

    - https://mp.weixin.qq.com
    - 选择 [开发者工具] -> [公众号平台测试账号]

- 编写配置文件

```
// conf.js
module.exports = {
    appid: 'wxfc60xxxxxx',
    appsecret: '23c57e1xxxx',
    token: 'kaikeba',
 };

// index.js
const conf = require('./conf')
```

3. 消息接口源码讲解

- co-wechat实现

```
// server.js
// npm i co-wechat
const wechat = require('co-wechat')
router.all('/wechat', wechat(conf).middleware(
    async message => {
        console.log('wechart', message)
        return 'Hello world! '+message.Content;
    }
))
```

> 扫描二维码测试 微信发送信息测试

- 验证部分

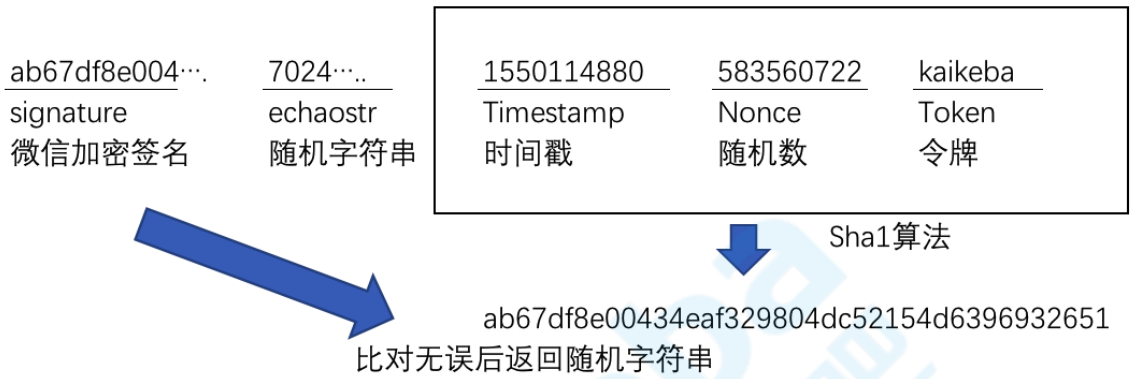
> crypto类
>
> https://www.liaoxuefeng.com/wiki/001434446689867b27157e896e74d51a89c25cc8b43bdb3000/001434501504929883d11d84a1541c6907eefd792c0da51000
>
> crypto模块的目的是为了提供通用的加密和哈希算法。用纯JavaScript代码实现这些功能不是不可能，但速度会非常慢。Nodejs用C/C++实现这些算法后，通过cypto这个模块暴露为JavaScript接口，这样用起来方便，运行速度也快。

```
  // index.js
  const Koa = require('koa')
const Router = require('koa-router')
const static = require('koa-static')
const xml2js = require('xml2js')
const app = new Koa()
const url = require('url')
const conf = require('./conf')


const crypto = require('crypto')
```

```
const xmlParser = require('koa-xml-body')
app.use(xmlParser())

const router = new Router()
app.use(static(__dirname + '/'))

// 验证
router.get('/wechat', ctx => {
    console.log('微信认证...', ctx.url)
    const {
        query
    } = url.parse(ctx.url, true)
    const {
        signature, // 微信加密签名，signature结合了开发者填写的token参数和请求中的timestamp参
数、nonce参数。
        timestamp, // 时间戳
        nonce, // 随机数
        echostr // 随机字符串
    } = query
    console.log('wechat', query)

    // 将 token timestamp nonce 三个参数进行字典序排序并用sha1加密


    let str = [conf.token, timestamp, nonce].sort().join('');
    console.log('str',str)
    let strSha1 = crypto.createHash('sha1').update(str).digest('hex');

    console.log(`自己加密后的字符串为：${strSha1}`);
    console.log(`微信传入的加密字符串为：${signature}`);
    console.log(`两者比较结果为：${signature == strSha1}`);

    // 签名对比，相同则按照微信要求返回echostr参数值
    if (signature == strSha1) {
        ctx.body = echostr
    } else {
        ctx.body = "你不是微信"
    }
})

// 接受信息
router.post('/wechat', ctx => {
    const {
        xml: msg
    } = ctx.request.body
    console.log('Receive:', msg)
    const builder = new xml2js.Builder()
    const result = builder.buildObject({
        xml: {
            ToUserName: msg.FromUserName,
            FromUserName: msg.ToUserName,
            CreateTime: Date.now(),
            MsgType: msg.MsgType,
```

```
            Content: 'Hello ' + msg.Content
        }
    })
    ctx.body = result
})

app.use(router.routes());
app.use(router.allowedMethods());
app.listen(3000);
```

- 验证过程



SHA1算法

> 安全哈希算法（Secure Hash Algorithm）主要适用于数字签名标准 （Digital Signature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA）。对于长度小于2^64位的消息，SHA1会产生一个160位的消息摘要。当接收到消息的时候，这个消息摘要可以用来验证数据的完整性。在传输的过程中，数据很可能会发生变化，那么这时候就会产生不同的消息摘要。 SHA1有如下特性：不可以从消息摘要中复原信息；两个不同的消息不会产生同样的消息摘要,(但会有1x10 ^ 48分之一的机率出现相同的消息摘要,一般使用时忽略)。
>
> 哈希： 不可变长 -> 摘要固定长度

归纳

- 摘要
- 雪崩效应
- 类似MD5 SHA256

- 收发信息

```
// 接受信息

// 将bodyparser更换xmlParser
const xmlParser = require('koa-xml-body')
app.use(xmlParser())
const xml2js = require('xml2js')

// 接受信息
```

```
router.post('/wechat', ctx => {
    const {
        xml: msg
    } = ctx.request.body
    console.log('Receive:', msg)
    const builder = new xml2js.Builder()
    const result = builder.buildObject({
        xml: {
            ToUserName: msg.FromUserName,
            FromUserName: msg.ToUserName,
            CreateTime: Date.now(),
            MsgType: msg.MsgType,
            Content: 'Hello ' + msg.Content
        }
    })
    console.log('xml:',result)
    ctx.body = result
})
```

3. 服务器端API调用

> 官方文档：https://mp.weixin.qq.com/wiki?t=resource/res_main&id=mp1421140183
>
> npm库 https://github.com/node-webot/co-wechat

- 功能
  - 用户管理
    - 用户分组
    - 备注姓名
    - 获取用户基本资料
    - 获取用户列表
    - 获取用户地理位置
  - 界面设置-定制菜单
  - 素材管理
  - 推广支持 生成二维码
- 源码实现

```
// index.html
// 获取关注着列表
getTokens: async function () {
    let res = await axios.get(`/getTokens`)
    console.log('res',res)
},
```

```
// index.js
const tokenCache = {
    access_token: '',
    updateTime: Date.now(),
    expires_in: 7200,
};
```

```
router.get('/getToken', async ctx => {

    const wxDomain = `https://api.weixin.qq.com`;
    const path = `/cgi-bin/token`;
    const params = `?
grant_type=client_credential&appid=${conf.appid}&secret=${conf.appsecret}`;
    const url = `${wxDomain}${path}${params}`;
    const res = await axios.get(url);
    Object.assign(tokenCache, res.data, {
        updateTime: Date.now()
    });
    ctx.body = res.data
})

router.get('/getFollowers', async ctx => {
    const url = `https://api.weixin.qq.com/cgi-bin/user/get?
access_token=${tokenCache.access_token}`
    const res = await axios.get(url)
    console.log('getFollowers', res.data)
    ctx.body = res.data
})
```

- co-wechat-api库实现

```
// index.js

const WechatAPI = require('co-wechat-api');
const api = new WechatAPI(conf.appid, conf.appsecret);

// 获取关注者列表
router.get('/getFollowers', async ctx => {
    var res = await api.getFollowers();
    ctx.body = res
})
```

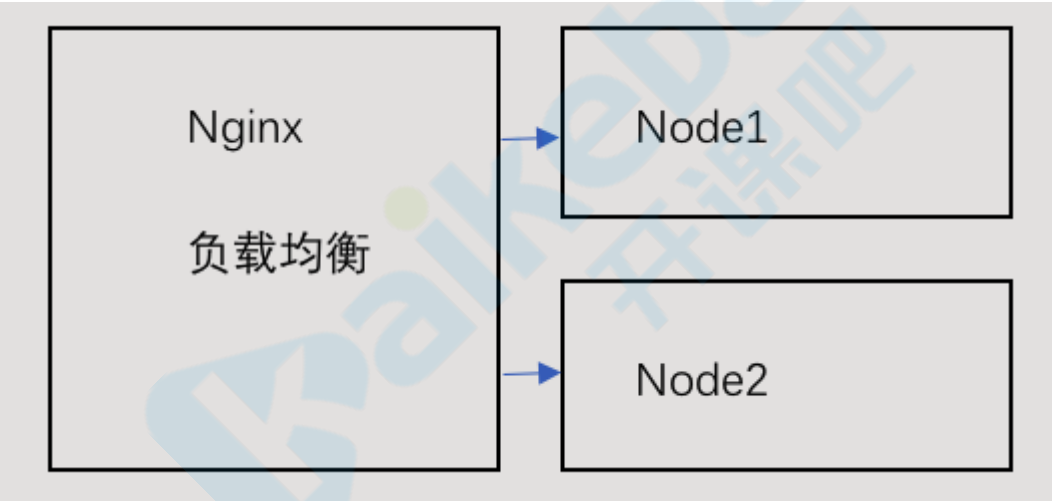
```
// index.html
// 获取关注着列表
getFollowers: async function () {
    let res = await axios.get(`/getFollowers`)
    console.log('res',res)
},
```

```
// 获取关注者列表
router.get('/getFollowers', async ctx => {
    var res = await api.getFollowers();
    console.log('res', res)
    res = await api.batchGetUsers(res.data.openid, 'zh_CN');
    console.log('res', res)
    ctx.body = res
})
```

1. 多进程下保存token 全局票据

> docker-compose提供MongoDB服务

> 当多进程时，token需要全局维护，以下为保存token的接口




```
// mongoose.js
const mongoose = require('mongoose')
const {
    Schema
} = mongoose
mongoose.connect('mongodb://localhost:27017/weixin', {
    useNewUrlParser: true
}, () => {
    console.log('Mongodb connected..')
})
exports.ServerToken = mongoose.model('ServerToken', {
    accessToken: String
});
```

```
// index.js
const api = new WechatAPI(conf.appid,
    conf.appsecret,
    async function () {
            return await ServerToken.findOne()
        },
        async function (token) {
                const res = await ServerToken.updateOne({}, token, { upsert: true })
        }
)
```