



链滴

使用 NodeJS 接入支付宝网站支付的 Demo 开发

作者: [xiluotop](#)

原文链接: <https://ld246.com/article/1572500884644>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



开发介绍

因为后期的一些需求需要使用到支付宝网站支付业务，而近期又学习了 NodeJS 后端的开发，于是乎网上找了一些资料，而支付宝开放平台又没有现成的 Demo 案例，也只有 NodeJS 开发的 SDK 所以自己花了一些时间尝试使用 NodeJS 开发一个示例 Demo 便于后面开发项目时去使用。

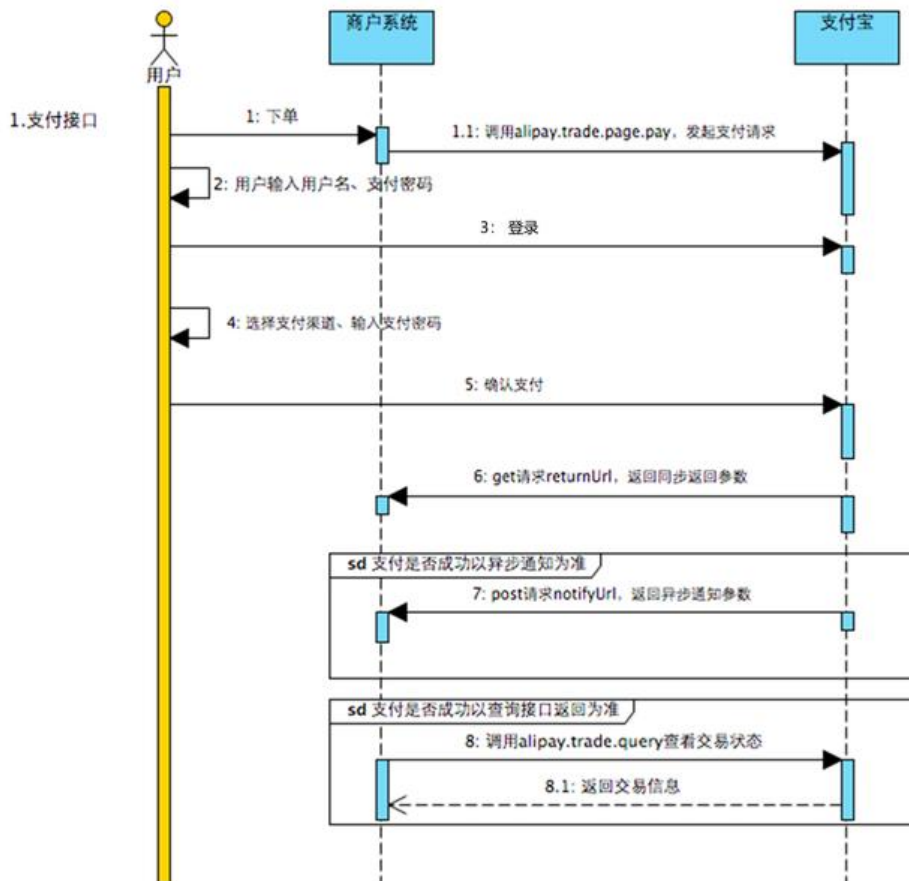
测试 DEMO: <https://github.com/xiluotop/NodeJS-Alipay-Demo>

开发步骤

- 一个订单的创建到支付成功的文字过程:

前端页面 -> 向服务端发送订单信息 -> 服务端确认信息, 向客户端发送确认信息 -> 客户端确认信息
服务端发送订单请求 -> 服务端验证订单请求信息 -> 服务端像支付宝发送订单生成 -> 支付宝向服务
返回订单数据 -> 服务端向客户端发送支付宝的表单信息 -> 客户端跳转到支付宝支付页面 -> 客户端
支付成功 -> 支付宝让客户端同步跳转到服务端指定的页面 -> 支付宝异步通知服务端订单支付结果 ->
务端接收异步通知做相应的业务处理。

- 支付宝调用支付接口的常规流程:



准备工作

- 安装 alipay 的 SDK工具: <https://www.npmjs.com/package/alipay-sdk>
`npm install alipay-sdk`
- 获取 alipay 的应用相关信息
 - appid
 - 秘钥
 - 应用私钥
 - 应用公钥
 - 支付宝公钥 (从支付宝开放平台上进行秘钥查询那里可以获取)
 - 网关
 - 这几步的获取方式可查看 <https://jiangck.com/articles/2019/08/10/1565412139037.html>
- 准备工作后的目录应如下:

```

> node_modules
└─ sandbox-pem
   ├── private_pem2048.txt
   ├── public_pem2048.txt
   ├── package-lock.json
   └── package.json
  
```

开发工作

- 配置 SDK

- 创建一个 alipay_config.js 此 js 脚本用于设置 SDK 的相关信息

```
const fs = require('fs');
const path = require('path');

// 这里配置基本信息
const AlipayBaseConfig = {
  appId: '', // 应用 ID
  privateKey: fs.readFileSync(path.join(__dirname, './sandbox-pem/private_pem2048.txt'), 'ascii'), // 应用私钥
  alipayPublicKey: '', // 支付宝公钥
  gateway: 'https://openapi.alipaydev.com/gateway.do', // 支付宝的应用网关, 此时为沙箱环境的网关
  charset: 'utf-8', // 字符集编码
  version: '1.0', // 版本, 默认 1.0
  signType: 'RSA2' // 秘钥的解码版本
};

module.exports = {
  AlipayBaseConfig: AlipayBaseConfig, // 将配置模块暴露供初始化调用
}
```

- 创建 createOrder.js 文件, 封装生成支付宝订单的模块

- 导入 SDK 环境: `const AlipaySDK = require("alipay-sdk").default;`
- 导入配置: `const alipayConfig = require(path.join(__dirname, './alipay_config.js'));` // 先定义好的 alipay_config.js 配置模块
- 获取 alipaySDK 的实例化对象并初始化: `const alipay = new AlipaySDK(alipayConfig.AlipayBaseConfig)`
- 封装 createOrder
 - 根据官方示例: https://www.yuque.com/chenqiu/alipay-node-sdk/page_api, alipay.trade.page.pay(统一收单下单并支付页面接口) 最后返回的是一个 html 片段, 所以需要使用 formData 进行数据的封装请求
 - 引入 alipayFormData 构造函数, 用来创建网站支付需要的 form 表单

```
const AlipayFormData = require('alipay-sdk/lib/form').default;
```

- 封装好的 createOrder 以及详细代码如下:

```
// 编写一个创建支付订单的函数, 异步等待执行的函数
async function createOrder(goods) {
  let method = 'alipay.trade.page.pay'; // 统一收单下单并支付页面接口
  // 公共参数 可根据业务需要决定是否传入, 当前不用
  // let params = {
  //   app_id: '2016101000654289', // 应用 id
  //   method: method, // 调用接口
  //   format: 'JSON', // 返回数据
  //   charset: 'utf-8', // 字符编码
  //   sign_type: 'RSA2', // 验签类型
  // }
```

```

// timestamp: getFormatDate(), // 请求时间戳
// version: '1.0', // 版本
//}
// 根据官方给的 API 文档提供的一个参数集合
let bizContent = {
  out_trade_no: Date.now(), // 根据时间戳来生成一个订单号,
  product_code: 'FAST_INSTANT_TRADE_PAY', // 商品码, 当前只支持这个
  total_amount: goods.cost, // 商品价格
  subject: goods.goodsName, // 商品名称
  timeout_express: '5m', // 超时时间
  passback_params: JSON.stringify(goods.pack_params), // 将会返回的一个参数, 可用于自定义商品信息最后做通知使用
}
const formData = new AlipayFormData(); // 获取一个实例化对象
formData.addField('returnUrl', 'http://jiangck.com:9999/payresult'); // 客户端支付成功后会步跳回的地址
formData.addField('notifyUrl', 'http://jiangck.com:9999/notify.html'); // 支付宝在用户支付成后会异步通知的回调地址, 必须在公网 IP 上才能收到
formData.addField('bizContent', bizContent); // 将必要的参数集合添加进 form 表单

// 异步向支付宝发送生成订单请求, 第二个参数为公共参数, 不需要的话传入空对象就行
const result = await alipay.exec(method, {}, {
  formData: formData
});
// 返回订单的结果信息
return result;
}

```

- 将生成订单的方法暴露出去

```

module.exports = {
  createOrder: createOrder
}

```

- 创建 checkSign.js 验签模块

• 验签的作用是在当用于支付成功后, 支付宝异步向服务器设置好的回调地址发送 post 请求, 用告知服务器用户的支付信息, 而且服务器最终也是依据这个异步通知来处理逻辑业务, 而不是靠支付的同步跳转, 并且当支付宝发送了异步通知请求后, 服务端需要做验证检查, 这一步是必须的, 因为及金钱交易必须要严谨, 一定要核实请求过来的信息真实性。合法后才能做逻辑处理, 封装的验签功也是基于 alipaySDK 的工具, 只是用来方便调用, 代码如下:

```

const path = require('path');

// 用于通知验签
// -----配置 alipay SDK 环境
// 导入 SDK
const AlipaySDK = require("alipay-sdk").default;
// 导入配置
const alipayConfig = require(path.join(__dirname, './alipay_config.js'));
// 初始化
const alipaySdk = new AlipaySDK(alipayConfig.AlipayBaseConfig);

async function checkNotify(obj) {
  const result = await alipaySdk.checkNotifySign(obj);
}

```

```
    return result;
  }
}
```

```
module.exports = checkNotify;
```

- 创建 mysql.js 模块，用于对数据库的存储工作进行简单的封装

- 代码如下：

```
/*
  这个自定义模块用来进行 mysql 数据库订单的增加和查询功能
*/
```

```
// 引入 mysql 模块
const mysql = require('mysql');
```

```
// 配置 mysql
const mysqlConfig = {
  host: 'localhost', // 数据库主机名
  port: '3306', // 端口号
  user: 'root', // 用户名
  password: '123456', // 密码
  database: 'alipay', // 数据库名
}
```

```
// 封装查询函数
function selectSql(sqlstr, callback) {
  // 建立数据库连接
  let sql = mysql.createConnection(mysqlConfig);
  let result = null;
  if (sql) {
    sql.query(sqlstr, callback);
    // 关闭数据库连接
    sql.end();
  }
}
```

```
// 封装添加函数
function addSql(sqlstr, callback) {
  return selectSql(sqlstr, callback);
}
```

```
// 将两个数据库操作方法暴露
module.exports = {
  selectSql: selectSql,
  addSql: addSql
}
```

- 创建 server.js 用来建立一个 http 服务器，并且做好各接口的请求处理

代码如下：

```
const path = require("path");
const bp = require('body-parser');
// 引入自定义 mysql 工具
const mysql = require(path.join(__dirname, './mysql.js'));
```

```

// 引入 express
const express = require('express');
// 获取 express 实例对象
let app = express();

// 设置托管静态资源
app.use(express.static(path.join(__dirname, './public')));
// 处理 post 请求参数
app.use(bodyParser.urlencoded({
  extended: false
}));

// 前端响应要创建订单的数据对象
app.get('/payinfo', (req, res) => {
  let data = req.query;
  // 做一个简单的商品判断
  if (data && (data.goodsName === '大卫龙' || data.goodsName === '冰阔咯' || data.goodsName === '雪碧' || data.goodsName === 'QQB') && data.count && data.cost) {
    res.send(Object.assign(data, {
      code: 200,
    }));
  } else {
    res.setHeader('content-type', 'application/javascript');
    res.send('alert("信息有误，请重新尝试!!!");');
  }
})

// 获取创建订单的自定义模块
const createOrder = require(path.join(__dirname, './createOrder.js')).createOrder;
// 获取验签自定义模块
const checkSign = require(path.join(__dirname, './checkSign.js'));

// 生成订单请求
app.post('/createOrder', (req, res) => {
  console.log(req.body.price);
  req.body.pack_params = {
    payName: req.body.payName,
    goodsName: req.body.goodsName,
    price: req.body.price,
    count: req.body.count,
    cost: req.body.cost,
  }
  async function asyncCreate() {
    const result = await createOrder(req.body);
    res.send(result);
  }
  asyncCreate();
});

// 支付的信息展示
app.get('/payresult', (req, res) => {
  let htmlStr = '';
  htmlStr += '<p>' + '商户订单号' + ':' + req.query.out_trade_no + '</p>'
  htmlStr += '<p>' + '支付宝交易订单号' + ':' + req.query.trade_no + '</p>'

```

```

    htmlStr += `<p>` + '交易金额' + ':' + req.query.total_amount + '¥ </p>'
    htmlStr += `<p>` + '交易时间' + ':' + req.query.timestamp + '¥ </p>'
    htmlStr += '<h1 style:"text-align:center;">支付成功!!! <a href="/index.html">返回首页!
/a></h1>'
    res.send(htmlStr);
  })

app.post('/notify.html', (req, res) => {
  // 输出验签结果
  async function checkResult(postData) {
    let result = await checkSign(postData);
    if (result) {
      // console.log('订单成功支付!!! 请做处理')
      // console.log(req.body);
      let data = req.body;
      let goods = JSON.parse(data.passback_params);
      let sqlStr = `
insert into order_list value("${data.out_trade_no}",
    "${data.trade_no}",
    "${goods.goodsName}",
    ${goods.price},
    ${goods.count},
    ${data.total_amount},
    "支付成功",
    "${goods.payName}");
`;
      // 响应支付宝处理成功, 否则支付宝会一直定时发送异步通知
      res.end('success');
      mysql.addSql(sqlStr)
    }
  }
  checkResult(req.body);
})

// 查询订单接口
app.get('/getorder', (req, res) => {
  mysql.selectSql('select * from order_list', (err, result) => {
    result = Object.assign({
      code: 200,
      msg: '获取成功',
      list: JSON.stringify(result),
    })
    res.send(result);
  });
})

app.listen(9999, () => {
  console.log('server start with 9999...');
})

```

- public 静态资源

- 静态资源只是前端用来让客户端进行操作的展示, 具体文件可以访问后面的 github 开源 Demo 获取, 里面含有一个 alipay.sql 数据库的转储文件, 也可自行创建。

总结

- 以上步骤就是一个支付宝网站支付接口的调用 Demo 简单开发。
- 开发过程会遇到的问题有如下几点：
 - SDK 配置中的公钥配置，看清楚是填写 **支付宝公钥**，而不是应用公钥
 - 秘钥算法一定要对应 RSA 与 RSA2 一定要弄清楚，是什么加密就写什么
 - 验签失败时要检查 **加密算法，支付宝公钥，应用私钥/公钥加密类型**
 - 验签通过后一定要想支付响应成功信息的字符串 "success"，不然会一直受到异步通知从而出现任务多次处理。
- 本测试 DEMO 已经开源 Github， [可点击我进行测试 Demo 的获取](#)。如果您有帮助的还请 Star 下O(∩_∩)O。
- 我也是刚进行学习的，这也是我的第一个支付测试 demo，肯定会有很 bug，后续会添加更多的接口测试 demo 和一步步的完善，有什么问题的话可以留言一起交流喔。