商户服务端对接文档

聚合支付 开放接口

版本 1.0 · 2026-10-21 · 5 个接口 · 请求签名 HMAC-SHA256

接口根地址	`https://qcsj88888.com/api/v1`	请求方式	`POST`，`application/json`
金额单位	分，整数（`100` = 1.00 元）	成功标志	响应体 `code = 0`

1 请求约定

所有接口用 POST 提交 JSON，响应也是 JSON。每个请求带三个请求头：

请求头	值
`Content-Type`	`application/json`
`Authorization`	`ApiKey <你的 api_key>`，注意 `ApiKey` 后面有一个空格
`Signature`	对请求体算出的 HMAC-SHA256，64 位小写十六进制，算法见 §2

响应包为 `{"code":0,"msg":"成功","data":{...},"status_code":200,"success":true}`。只看 `code` 和 `data`：`code=0` 成功，业务数据在 `data`；`code≠0` 失败，原因看 `msg`。`status_code`、`success` 与 `code` 重复，可忽略，下文示例省略这两个字段。**HTTP 200 不代表成功。**

用途	方法 + 路径（拼在根地址后面）
统一下单	`POST /payment/openapi/unified/order/create`
查询订单	`POST /payment/openapi/unified/order/query`
发起退款	`POST /payment/openapi/unified/refund`
查询退款	`POST /payment/openapi/unified/refund/query`
获取渠道用户 ID（仅 JSAPI / 小程序）	`POST /payment/openapi/unified/channel-user/jump-url`

商户号 `mch_no`：租户只有一个商户号时不传。有多个商户号时，**下单**和**获取渠道用户 ID** 必须传；查单、退款、查退款按单号寻址，不必传。

2 签名

签名只算请求体的顶层字段，与 URL、请求头、时间戳无关。

1. 取请求体的顶层字段。
2. 去掉 `sign`、`null`、空字符串、空对象 `{}`、空数组 `[]`。
3. 按字段名 ASCII 升序排列。
4. 每个值先 URL 编码（RFC 3986：空格 `%20`、星号 `%2A`、波浪号保留）。对象和数组先按键排序输出紧凑 JSON（无空格、中文不转义），再整体编码。
5. 用 `&` 拼成 `k=v&k=v`。
6. 以 `api_secret` 为密钥做 HMAC-SHA256，取小写十六进制，放入 `Signature` 头。

```
from urllib.parse import quote
import hashlib, hmac, json


def sign(body: dict, api_secret: str) -> str:
    parts = []
    for key in sorted(body):
        value = body[key]
        if key == "sign" or value is None or value == "" or value == {} or value == []:
            continue
        if isinstance(value, (dict, list)):
            text = json.dumps(value, sort_keys=True, separators=(",", ":"), ensure_ascii=False)
        elif isinstance(value, bool):
            text = "true" if value else "false"
        else:
            text = str(value)
        parts.append(key + "=" + quote(text, safe=""))
    raw = "&".join(parts)
    return hmac.new(api_secret.encode(), raw.encode(), hashlib.sha256).hexdigest()
```

密钥只在服务端使用。不要把 `api_secret` 写进浏览器、App 或前端仓库。签名算的是你实际发出的那个请求体，改了任何字段都要重算。

3 下单

3.1 必填参数

字段	类型	说明
`out_trade_no`	string(64)	你的订单号，**幂等键**。同一租户内唯一。
`amount`	integer	金额，单位**分**，必须大于 0。
`way_code`	string(16)	支付方式，见 §4。
`subject`	string(128)	商品标题。
`mch_no`	string(64)	多商户时必传；单商户可空。

3.2 可选参数

字段	类型	说明
`body`	string(512)	商品描述。不传时用 `subject`。
`notify_url`	string(512)	异步通知地址，必须是公网可访问的 http/https 地址。见 §10。
`return_url`	string(512)	支付完成后浏览器跳回的地址（部分支付方式使用）。
`expired_time`	integer	订单有效期，单位秒，默认 7200（2 小时）。
`client_ip`	string(64)	用户终端 IP。不传时取请求来源 IP。
`ext_param`	string(512)	透传参数，上游原样返回。
`channel_extra`	object	支付方式附加参数，见 §4。

3.3 完整调用示例

```
import requests, uuid

BASE = "https://qcsj88888.com/api/v1"
API_KEY = "你的 api_key"
API_SECRET = "你的 api_secret"


def post(path: str, body: dict) -> dict:
    headers = {
        "Content-Type": "application/json",
        "Authorization": "ApiKey " + API_KEY,
        "Signature": sign(body, API_SECRET),   # sign() 见 §2
    }
    resp = requests.post(BASE + path, json=body, headers=headers, timeout=15)
    return resp.json()


body = {
    "out_trade_no": "T" + uuid.uuid4().hex[:16],
    "amount": 100,                      # 分
    "way_code": "QR_CASHIER",
    "subject": "测试商品",
    "notify_url": "https://example.com/pay/notify",
}
print(post("/payment/openapi/unified/order/create", body))
```

3.4 成功响应

```
{
  "code": 0,
  "msg": "成功",
  "data": {
    "order_no": "UP1960000000000000001",
    "out_trade_no": "T20261020001",
    "way_code": "QR_CASHIER",
    "amount": 100,
    "refundable_amount": 100,
    "trade_status": "0",
    "pay_data_type": "codeUrl",
    "pay_data": "https://qr.example.com/xxx",
    "created_time": "2026-10-20T12:00:00+08:00"
  }
}
```

`order_no` 是平台订单号。`out_trade_no` 是你的单号，后续查单、退款都用它。把 `pay_data` 交给用户完成支付，渲染方式见 §4.2。

4 支付方式与 pay_data

4.1 支持的 way_code

way_code	场景	channel_extra
`QR_CASHIER`	聚合扫码，用户扫你	可选 `payDataType`： `codeUrl` / `codeImgUrl`
`AUTO_BAR` `ALI_BAR` `WX_BAR` `YSF_BAR`	条码支付，你扫用户（聚合 / 支付宝 / 微信 / 云闪付）	必传 `{"auth_code":"付款码"}`
`ALI_JSAPI`	支付宝生活号	必传 `{"buyerUserId":"..."}`
`WX_JSAPI`	微信公众号	必传 `{"openid":"..."}`；服务商模式加 `wxAppId`
`WX_LITE`	微信小程序	必传 `{"openid":"..."}`；服务商模式加 `wxAppId`
`YSF_JSAPI`	云闪付 JSAPI	可选 `{"openid":"..."}`
`ALI_APP` `WX_APP`	原生 APP 支付（支付宝 / 微信）	不用传
`ALI_WAP`	支付宝手机网站	可选 `payDataType`： `form` / `codeImgUrl` / `payUrl`
`ALI_PC`	支付宝电脑网站	可选 `payDataType`： `form` / `payUrl`
`ALI_QR`	支付宝二维码	可选 `payDataType`： `codeUrl` / `codeImgUrl`
`WX_NATIVE`	微信扫码	可选 `payDataType`： `codeUrl` / `codeImgUrl`
`WX_H5`	微信 H5	可不传；已拿到 `openid` 时可传

`way_code` 大小写敏感。条码类的付款码可写 `auth_code`，也可写 `authCode`。付款码、 `openid`、 `buyerUserId` 只用于本次请求：平台不落库、不写日志。

4.2 pay_data 怎么用

pay_data_type	pay_data 内容	用法
`codeUrl`	支付链接	生成二维码给用户扫
`codeImgUrl`	二维码图片 URL	放进 `<img src>`
`payUrl`	支付页链接	浏览器跳转
`form`	HTML 表单字符串	作为独立页面提交；不要插进现有页面
`wxapp`	JSON 字符串，微信支付参数	`JSON.parse` 后传给 `WeixinJSBridge.invoke('getBrandWCPayRequest', ...)`

> **只渲染 http/https 地址。**把 `pay_data` 当数据，不要当代码执行。 `form` 是完整表单 HTML，插入页面会破坏现有 DOM。

5 幂等与失败重试

- **同一 `out_trade_no`**，金额、 `way_code`、 `subject` 全部相同：返回原订单， `code=0`，不会重复扣款。
- **同一 `out_trade_no`**，以上任一项不同：返回 `code=409`「out_trade_no 已存在且参数不一致」。换单号再下。
- **`code=500` 且 msg 以「本地订单已保留」开头**：上游调用失败或超时，这笔**可能已经成功**。不要换单号重下，先查单。确认未支付且没有可用的 `pay_data` 后，用**新的** `out_trade_no` 重新下单。
- 退款同理： `out_refund_no` 是幂等键，同一单号参数不同返回 `code=409`。
- 调用前先把 `out_trade_no` 持久化。所有网络超时、连接中断，一律先查单再决定。

6 查询订单

```
{"out_trade_no": "T20261020001"}
```

也可以传 `{"order_no": "UP1960000000000000001"}`。两个都传时以 `order_no` 为准。响应的 `data` 与下单相同。

trade_status	含义
`0`	待支付
`1`	支付中
`2`	支付成功（之后可能变成 `5` 已退款）
`3`	支付失败
`4`	已撤销
`5`	已退款
`6`	已关闭

`3` / `4` / `5` / `6` 是终态。建议每 3 秒查一次，到终态停止。支付结果以查单或异步通知中的 `trade_status` 为准。

7 发起退款

```
{
  "out_trade_no": "T20261020001",
  "out_refund_no": "R20261020001",
  "refund_amount": 100,
  "refund_reason": "测试退款",
  "notify_url": "https://example.com/refund/notify"
}
```

定位订单用 `order_no` 或 `out_trade_no`，至少传一个，都传时以 `order_no` 为准。`mch_no` 可选，传了必须与订单一致。

字段	类型	说明
`out_refund_no`	string(64)	你的退款单号，**幂等键**。
`refund_amount`	integer	退款金额，分。
`refund_reason`	string(64)	退款原因，必填。
`notify_url`	string(512)	本笔退款的通知地址，可选。

- 只有 `trade_status=2` 的订单能退款。
- `refund_amount` 不能超过 `refundable_amount`。在途退款也占用额度。
- 同一订单可以多次部分退款，每次换一个 `out_refund_no`。

```
{
  "code": 0,
  "msg": "成功",
  "data": {
    "refund_no": "UR1960000000000000001",
    "out_refund_no": "R20261020001",
    "order_no": "UP1960000000000000001",
    "mch_no": "M202610200001",
    "amount": 100,
    "state": "1",
    "created_time": "2026-10-20T12:05:00+08:00"
  }
}
```

state	含义
`0`	退款生成
`1`	退款中
`2`	退款成功
`3`	退款失败
`4`	退款关闭

`2` / `3` / `4` 是终态。

8 查询退款

```
{"out_refund_no": "R20261020001"}
```

也可以传 `{"refund_no": "UR1960000000000000001"}`。两个都传时以 `refund_no` 为准。响应与退款相同。不要只等通知：主动查到 `state=2` 再确认退款成功。

9 获取渠道用户 ID（JSAPI / 小程序）

`WX_JSAPI`、`WX_LITE`、`ALI_JSAPI` 需要先拿渠道用户 ID。`YSF_JSAPI` 可选。上游签名要用 `api_secret`，所以这一步必须由服务端换取 URL，浏览器不接触密钥。

```
{
  "mch_no": "M202610200001",
  "redirect_url": "https://example.com/channel/callback",
  "ext_param": "order=1"
}
```

```
{"code": 0, "msg": "成功", "data": {"url": "https://...签名后的上游地址"}}
```

把用户浏览器跳到返回的 `url`。授权完成后，`redirect_url` 会带上 `channelUserId`（微信还会带 `channelAppId`）：

- 微信：把 `channelUserId` 作为 `openid` 放进下单的 `channel_extra`；服务商模式再把 `channelAppId` 作为 `wxAppId`。
- 支付宝：把 `channelUserId` 作为 `buyerUserId` 放进下单的 `channel_extra`。

`if_code` 不用传，平台固定使用 `fuioupay`。

10 异步通知

平台用 **form POST**（`application/x-www-form-urlencoded`）推送，固定三个字段：

字段	说明
`notify_id`	事件 ID。同一事件重试时不变，用它去重。
`timestamp`	毫秒时间戳。
`content`	JSON 字符串，结构见下。

支付事件（`event=ORDER`）：

```
{
  "event": "ORDER",
  "order": {
    "order_no": "UP1960000000000000001",
    "out_trade_no": "T20261020001",
    "mch_no": "M202610200001",
    "way_code": "QR_CASHIER",
    "amount": 100,
    "refunded_amount": 0,
    "refundable_amount": 100,
    "trade_status": "2",
    "upstream_pay_order_id": "19522112587029827620865",
    "upstream_channel_order_no": "42000012342026102000000000000",
    "pay_time": "2026-10-20T12:00:00+08:00",
    "err_code": null,
    "err_msg": null
  },
  "upstream": {}
}
```

退款事件（`event=REFUND`）：

```
{
  "event": "REFUND",
  "refund": {
    "refund_no": "UR1960000000000000001",
    "out_refund_no": "R20261020001",
    "order_no": "UP1960000000000000001",
    "amount": 100,
    "state": "2",
    "upstream_refund_order_id": "...",
    "success_time": "2026-10-20T12:06:00+08:00",
    "err_code": null,
    "err_msg": null
  },
  "order": { "...": "订单当前快照，字段同 ORDER 事件" },
  "upstream": {}
}
```

`upstream` 是上游回调字段的白名单投影（`payOrderId`、`state`、`channelOrderNo` 等），没有快照时是空对象。

10.1 推送时机

事件	什么时候推
`ORDER`	订单状态变成 `2` / `3` / `4` / `6`
`ORDER` + `REFUND`	全额退款导致订单变成 `5`，两条都推
`REFUND`	退款状态变成 `2` / `3` / `4`
不推	订单 `0` / `1`、退款 `0` / `1`
只推 `REFUND`	部分退款成功、订单仍是 `2`；新的 `refunded_amount` 从订单快照读

10.2 接收规则

- 返回任意 **2xx** 即视为接收成功，响应体不限。
- 失败后每分钟重试一次，**最多投递 3 次**（含首次）。
- 用 `notify_id` 去重。重复通知直接返回 2xx。

- 回调**不带签名**。不放心时以查单或查退款的结果为准，不要只凭回调发货。
- `notify_url` 必须公网可达，平台会拒绝 `localhost`、`127.0.0.1`、内网地址。

11 错误码与排查

HTTP	code	含义	处理
200	`0`	成功	处理 `data`
200	`400`	业务参数不合法	看 `msg`
200	`404`	订单或退款单不存在	核对单号；确认是同一租户
200	`409`	幂等键已存在且参数不一致	换单号，或先查询确认原单
200	`500`	上游失败或超时	不要换单号，先查单
400	`400`	JSON 格式错误或字段校验失败	看 `msg` 里的字段名
401	`10401`	认证失败	检查 `ApiKey` 前缀、key、签名
404	`404`	路径不存在	检查 URL 拼写
500	`500`	服务端异常	稍后重试；持续出现联系平台

11.1 常见问题

- **一直 401**：最常见是签名对不上。确认 `Signature` 是对**实际发送的 body** 算的；`Authorization` 是 `ApiKey` + 一个空格 + key；`api_key` 与 `api_secret` 是同一对。
- **金额差 100 倍**：金额单位是分。`1.00 元` 传 `100`。请求体字段全部用 snake_case。
- **收不到通知**：`notify_url` 要公网可达，不能是 `localhost` 或内网 IP；返回非 2xx 会重试，3 次后停止。

12 联调清单

☐ 用 1 元（`100` 分）跑通「下单 → 支付 → 查单 → 全额退款 → 查退款」，确认所有金额都是**分**。
☐ `notify_url` 公网可达，返回 2xx；重复通知按 `notify_id` 去重。
☐ 同一 `out_trade_no` 原样重放返回同一笔订单；改金额重放返回 `code=409`。
☐ 按业务选对 `way_code`；条码类传 `channel_extra.auth_code`；JSAPI 类先拿 `openid`。
☐ `pay_data` 只按 http/https 渲染；`form` 不插入现有页面。
☐ 退款后主动查 `refund/query`，确认 `state=2`。
☐ 把 `out_trade_no`、`out_refund_no` 持久化，接口超时不换号，先查单。