# 前后端实现二维码扫码登录-深度剖析

## 简介

还是老样子，先把产品的需求设计给理一理先。如果没有做过这一块的前端小伙伴，可能一脸懵逼。没事，我也一样，万事开头都不简单，所以需要我们花点时间精力梳理一下

### 需求：

1. 登录二维码有效期默认2分钟，超过2分钟提示用户刷新
2. 扫码成功后，跳转扫码成功页面，用户在移动端点击确认登录后，登录进入应用

### 理解

1. 登录二维码有效期默认2分钟，超过2分钟提示用户刷新： 在生成登录二维码时，后端服务器将生成一个二维码标识符，并设置一个2分钟的有效期。后端可以通过生成时间戳或其他方法来跟踪二维码的有效性。前端在展示二维码的同时也开始计时，在二维码有效期结束前，如果用户未完成扫码登录，可以提示用户刷新二维码来获取一个新的二维码和标识符。
2. 扫码成功后，跳转扫码成功页面，用户在移动端点击确认登录后，登录进入应用： 一旦用户在移动设备上成功扫描二维码，并点击确认登录按钮，前端应轮询向后端发送请求，确认用户的登录操作。

## 梳理

通过上面简介中的初步分析，可以大致画一下下面的流程图，这样方便我们一步步去实现

### 流程图

### 接口设计

根据上面分析出来的流程图，我们了解到

登录二维码有效期默认2分钟，超过2分钟提示用户刷新

在生成登录二维码时，后端服务器将生成一个二维码标识符，并设置一个2分钟的有效期。后端可以通过生成时间戳或其他方法来跟踪二维码的有效性，与此同时，在前端展示出二维码图之后，前端就需要进行轮询监听调后端的接口，**而后端也是通过该接口来反馈给前端是否有人扫码成功、或者失败、或者过期**

后端接口设计需要将这几个状态标记出来，比如

**reason: QRCODE_SUCCESS、QRCODE_ERROR、QRCODE_EXPIRE；**

前端通过轮询的接口接收到这些不同状态做不同的处理逻辑即可

那么后端需要设计的就以下这三个接口，基本满足市面上大部分扫码需求了：

## 1. 二维码生成

如果有不同端扫码的需求，可以加上一个platform平台参数，如下

### 接口命名

POST: /v1/accounts/qrcode/

获取二维码

platform:

GKOL_PC // xxpc端

GKOL_WEB // xxweb端

GKOL_WEB_MANAGER // xx管理端

GKOL_WEB_OPERATE // xx后台

GKOL_WEB_ORGANIZE // xx后台

### body参数：

```
1 ts
2 复制代码
3 export interface Request {    /**     * 重新生成/取消时需要给     */    code?:
  string;    platform: string;}
```

### 响应结构：

```
1 ts
2 复制代码
3 export interface Response {    /**     * 二维码，前缀+base64串，格式:
  rk://scanforpclogin/{qrcode}     * 二维码json解析结构     * {     * "id":"",
  * "expire":1685696389,     * "prefix":"rk://scanforpclogin/"     *
  "platform":"Rxx_PC"     * }     */    png: string;}
```

## 2. 扫描二维码

主要给移动端用来扫码调的， 移动端先扫码web端的二维码图片，解析得到id，然后将id作为参数传进该**扫描二维码**接口

### 接口命名

POST: /v1/accounts/qrcode_fill

### body参

可以省略，用户信息是放在header的token中的

### 响应结构

```ts
ts
复制代码
export interface Response {    /**     * 身份卡ID，为空就不限制     */    card_id: string;    /**     * 二维码令牌，二维码解析结果     */    id: string;    /**     * 状态，SCAN 已扫描     * VERIFY 已确认     * CANCEL 取消     */    step: string;}
```

## 3. 账号登陆

web端通过轮询该接口，监听移动端的扫码状态，如扫描二维码的三个状态：

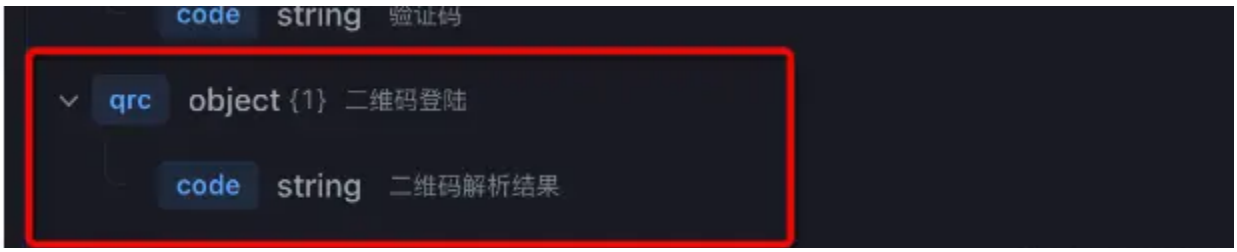
- SCAN 已扫描
- VERIFY 已确认
- CANCEL 取消

然后就是另外的三个状态：QRCODE_SUCCESS、QRCODE_ERROR、QRCODE_EXPIRE 通过这些状态，基本都能实现市面上的关于二维码的扫码业务逻辑了；

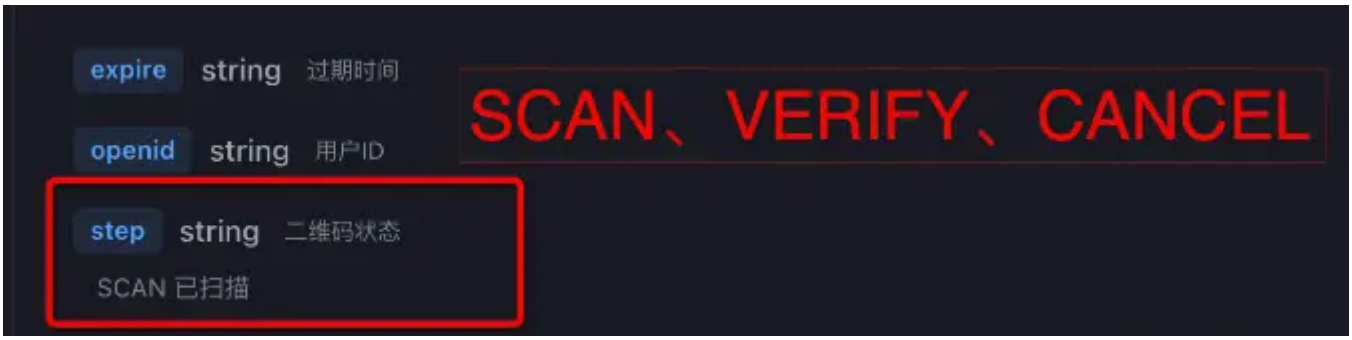
### 接口命名

POST /v1/passport/guest

### body参数




### 响应结构




## 前端实现

这里主要解说上面所提的环节以及接口返回的格式做一下简略的处理总结

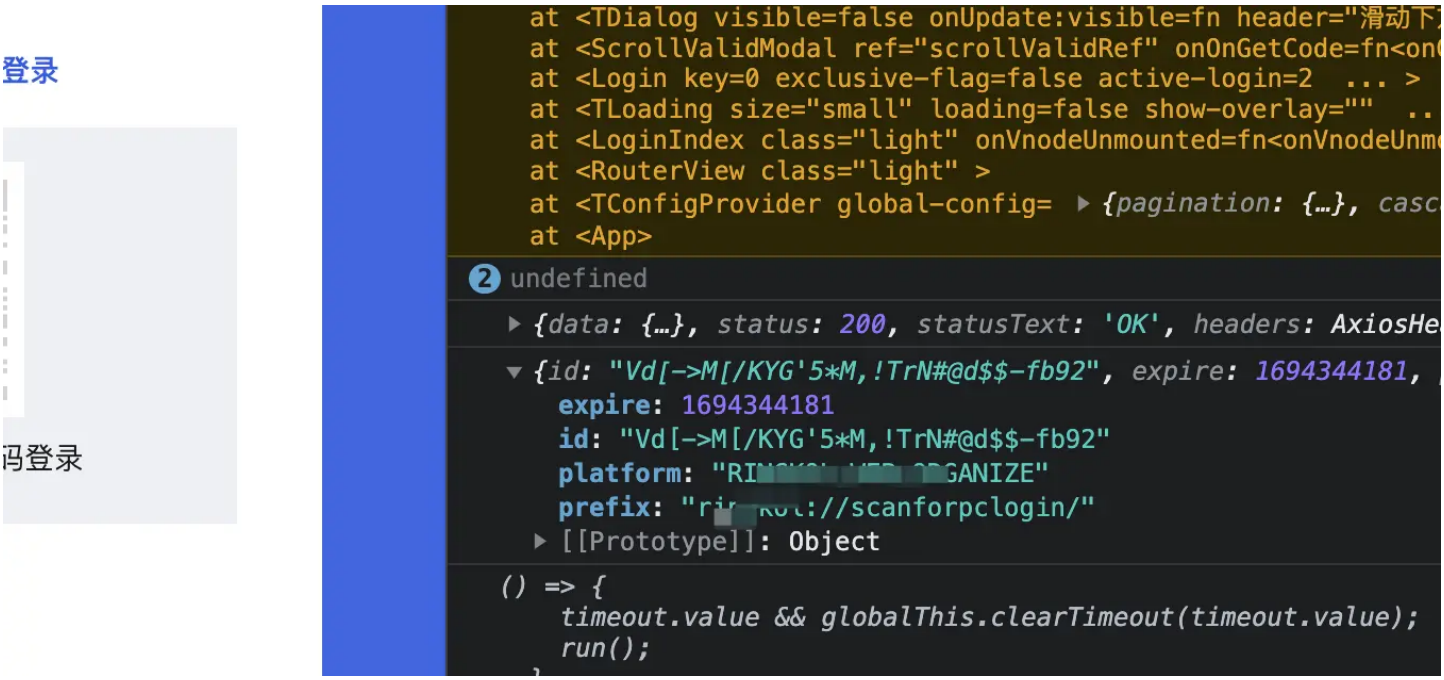
### 1. 二维码生成接口返回

将图中的png结果，按照接口文档拼接成base64格式后，通过qrcode-parser解析数据

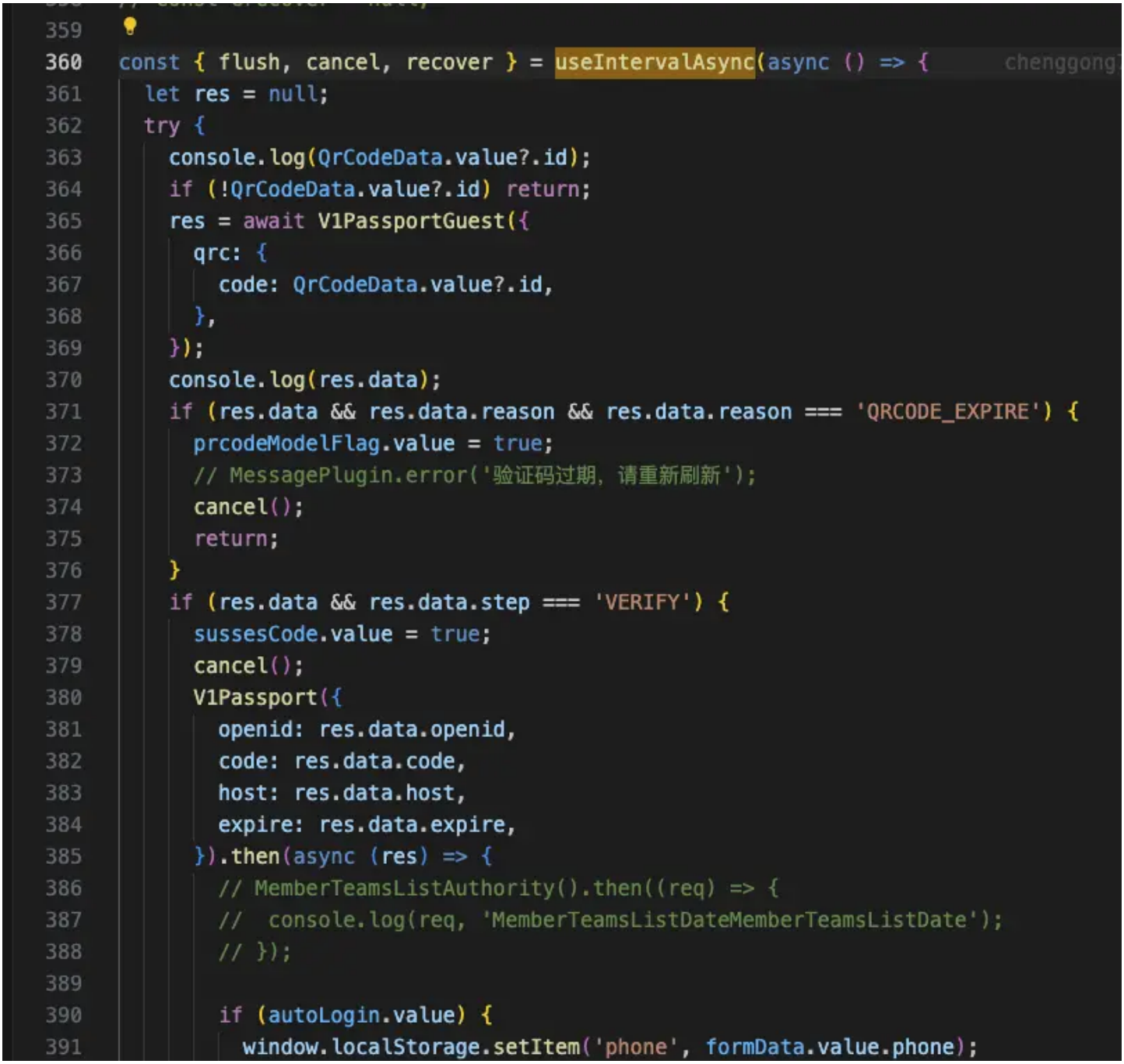
`import qrcodeParser from 'qrcode-parser';`

```
ts
复制代码
    V1CreateQRcode().then((res) => {        qrcodeUrl.value =
`${res.data?.png}`;       qrcodeParser(res.data?.png)
        .then((result: any) => {
console.log(JSON.parse(result));              QrCodeData.value = result ?
JSON.parse(result) : null;               // setUseAsyncHookInit();
    console.log(flush);               // flush();               recover();
            // qrcodeUrl.value = JSON.stringify(res.data?.png);          })
          .catch((err) => {               console.log(err);          });
 });
```

解析后即可得到如下结果：




1. 接口轮询Hook的封装 调轮询的hook接口实现监听和逻辑处理，而关于hook的实现，将在下一篇文章详细介绍



```
359
360 const { flush, cancel, recover } = useIntervalAsync(async () => {
361   let res = null;
362   try {
363     console.log(QrCodeData.value?.id);
364     if (!QrCodeData.value?.id) return;
365     res = await V1PassportGuest({
366       qrc: {
367         code: QrCodeData.value?.id,
368       },
369     });
370     console.log(res.data);
371     if (res.data && res.data.reason && res.data.reason === 'QRCODE_EXPIRE') {
372       prcodeModelFlag.value = true;
373       // MessagePlugin.error('验证码过期，请重新刷新');
374       cancel();
375       return;
376     }
377     if (res.data && res.data.step === 'VERIFY') {
378       sussesCode.value = true;
379       cancel();
380       V1Passport({
381         openid: res.data.openid,
382         code: res.data.code,
383         host: res.data.host,
384         expire: res.data.expire,
385       }).then(async (res) => {
386         // MemberTeamsListAuthority().then((req) => {
387         //  console.log(req, 'MemberTeamsListDateMemberTeamsListDate');
388         // });
389
390         if (autoLogin.value) {
391           window.localStorage.setItem('phone', formData.value.phone);
```

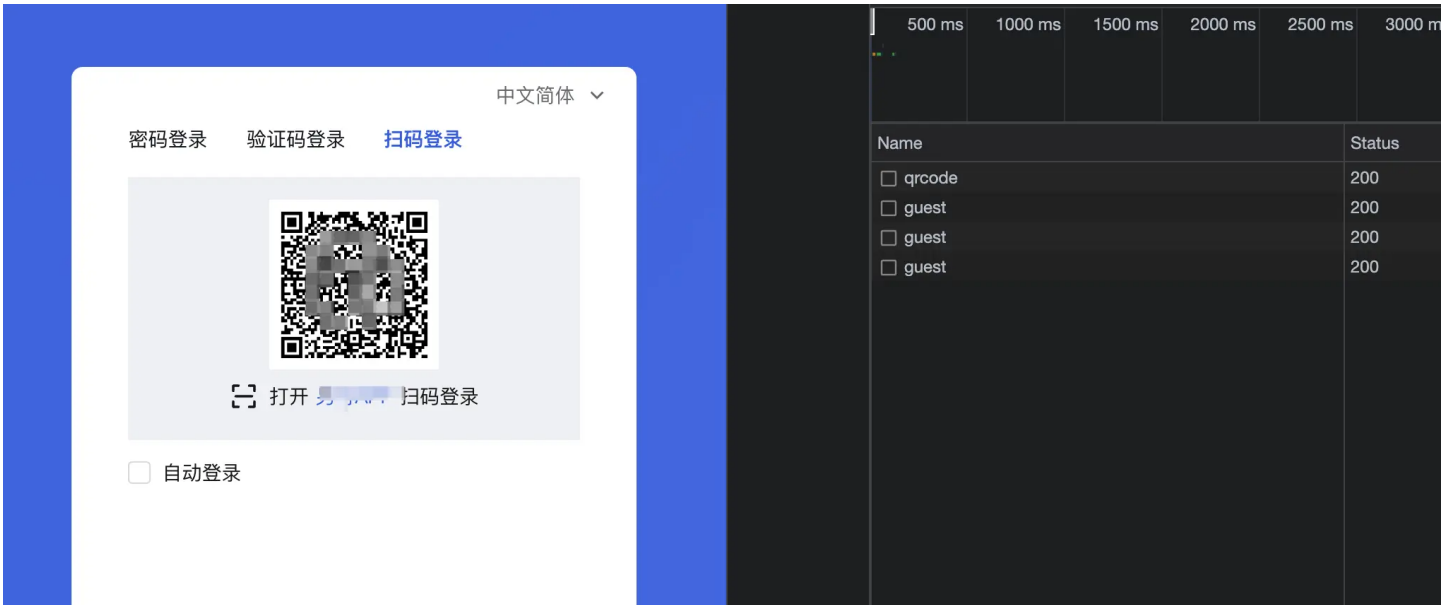
# 总结

登录二维码有效期默认2分钟，超过2分钟提示用户刷新： 在生成登录二维码时，后端服务器将生成一个二维码标识符，并设置一个2分钟的有效期，即**生成二维码中解析的数据**

```
1 js
2 复制代码
3 {    "id": "", // 生成二维码的标识符，这个要传给账号登录的code参数    "expire":
  1685696389, // 有效时间，就是生成该二维码的时间，    "prefix":
  "rk://scanforpclogin/",    "platform": "Rxx_PC"}
```

后端通过生成时间戳来expire，跟踪二维码的有效性，当然也可以通过其他方法实现。

前端在展示二维码后，开始轮询调**账号登录接口**




将二维码标识传到code字段中，在二维码有效期结束前，如果用户未完成扫码登录，可以提示用户刷新二维码来获取一个新的二维码和标识符。 如下：


关于其他的错误成功等状态，**就根据自己的业务需求进行操作**，比如扫码成功后，跳转扫码成功页面，用户在移动端点击确认登录后，登录进入应用： 一旦用户在移动设备上成功扫描二维码，并点击确认登录按钮，前端应立即向后端发送请求，确认用户的登录操作

后端服务器检查接收到的请求中的标识符是否与之前生成的相匹配，并验证标识符的有效性和状态。如果匹配和验证成功，则表示用户已经完成了登录操作。后端服务器可以更新用户的登录状态，并将用户重定向到进入应用程序的页面。