# 6.3.1 解析Token刷新机制

以下通过一个 **超市会员卡** 的类比，结合简明代码案例，解析Token刷新机制：

### 一、核心原理类比

|  |  |  |
| --- | --- | --- |
| **概念** | **超市场景类比** | **技术实现对应** |
| **Access Token** | 当日有效入场券（2小时） | 短期令牌（JWT 2小时） |
| **Refresh Token** | 7天续券卡（可换新入场券） | 长期令牌（JWT 7天） |
| **Redis** | 会员卡管理中心（记录卡状态） | 存储令牌与设备绑定信息 |

### 二、完整流程代码示例（PHP+JS）

#### 1. 登录接口生成双令牌

// app/controller/Auth.php   
public function login() {  
 // 验证用户身份   
 $user = UserModel::where('username', $this->request->post('username'))->find();  
 if (!$user || !password\_verify($this->request->post('password'), $user->password)) {  
 return json(['code' => 401, 'msg' => '认证失败']);  
 }  
   
 // 生成双令牌   
 $accessToken = Jwt::encode([  
 'uid' => $user->id,  
 'exp' => time() + 7200 // 2小时过期   
 ], env('JWT\_SECRET'));  
   
 $refreshToken = Jwt::encode([  
 'uid' => $user->id,  
 'exp' => time() + 604800 // 7天过期   
 ], env('JWT\_SECRET'));  
   
 // Redis存储关联关系（关键！）  
 Redis::setex("user:token:{$user->id}", 604800, json\_encode([  
 'access' => $accessToken,  
 'refresh' => $refreshToken,  
 'device' => md5($this->request->header('User-Agent') . $this->request->ip())  
 ]));  
   
 return json([  
 'access\_token' => $accessToken,  
 'refresh\_token' => $refreshToken,  
 'expires\_in' => 7200   
 ]);  
}

#### 2. 令牌刷新接口（核心逻辑）

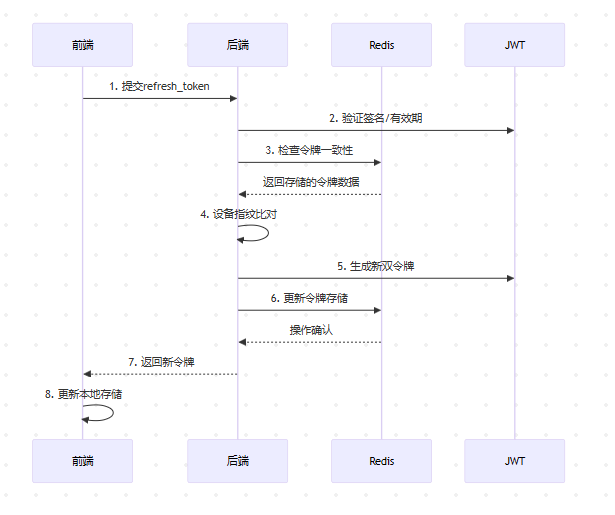
// app/controller/Auth.php   
public function refreshToken() {  
 $refreshToken = $this->request->post('refresh\_token');  
   
 try {  
 // 1. 解码验证Refresh Token   
 $payload = Jwt::decode($refreshToken, env('JWT\_SECRET'));  
 $userId = $payload['uid'];  
   
 // 2. 验证Redis存储一致性   
 $storedData = json\_decode(Redis::get("user:token:{$userId}"));  
 if (!$storedData || $storedData->refresh !== $refreshToken) {  
 throw new Exception('令牌已失效');  
 }  
   
 // 3. 设备一致性校验（防盗用）  
 $currentDevice = md5($this->request->header('User-Agent') . $this->request->ip());  
 if ($storedData->device !== $currentDevice) {  
 Redis::del("user:token:{$userId}"); // 清除异常设备令牌   
 return json(['code' => 403, 'msg' => '设备变更需重新登录']);  
 }  
   
 // 4. 生成新令牌并更新存储   
 $newAccess = Jwt::encode(['uid' => $userId, 'exp' => time()+7200], env('JWT\_SECRET'));  
 $newRefresh = Jwt::encode(['uid' => $userId, 'exp' => time()+604800], env('JWT\_SECRET'));  
   
 Redis::setex("user:token:{$userId}", 604800, json\_encode([  
 'access' => $newAccess,  
 'refresh' => $newRefresh,  
 'device' => $currentDevice   
 ]));  
   
 return json([  
 'access\_token' => $newAccess,  
 'refresh\_token' => $newRefresh,  
 'expires\_in' => 7200   
 ]);  
 } catch (Exception $e) {  
 return json(['code' => 401, 'msg' => $e->getMessage()]);  
 }  
}

#### 3. 前端自动刷新（Vue3示例）

// utils/request.js   
const service = axios.create({   
 baseURL: '/api',  
 timeout: 5000   
});  
   
// 请求拦截器（添加Access Token）  
service.interceptors.request.use(config => {  
 const token = localStorage.getItem('access\_token');   
 if (token) config.headers.Authorization = `Bearer ${token}`;  
 return config;  
});  
   
// 响应拦截器（自动刷新Token）  
service.interceptors.response.use(response => response, async error => {  
 const originalRequest = error.config;   
   
 if (error.response.status === 401 && !originalRequest.\_retry) {  
 originalRequest.\_retry = true;  
   
 try {  
 const { data } = await axios.post('/auth/refresh', {  
 refresh\_token: localStorage.getItem('refresh\_token')   
 });  
   
 localStorage.setItem('access\_token', data.access\_token);   
 localStorage.setItem('refresh\_token', data.refresh\_token);   
 originalRequest.headers.Authorization = `Bearer ${data.access\_token}`;   
 return service(originalRequest);  
 } catch (refreshError) {  
 localStorage.removeItem('access\_token');   
 localStorage.removeItem('refresh\_token');   
 window.location.href = '/login';  
 }  
 }  
   
 return Promise.reject(error);   
});

### 三、关键流程图示

sequenceDiagram   
 participant 前端   
 participant 后端   
 participant Redis   
   
 前端->>后端: 1. 提交refresh\_token   
 后端->>JWT: 2. 验证签名/有效期   
 后端->>Redis: 3. 检查令牌一致性   
 Redis-->>后端: 返回存储的令牌数据   
 后端->>后端: 4. 设备指纹比对   
 后端->>JWT: 5. 生成新双令牌   
 后端->>Redis: 6. 更新令牌存储   
 Redis-->>后端: 操作确认   
 后端-->>前端: 7. 返回新令牌   
 前端->>前端: 8. 更新本地存储



### 四、安全增强要点

1. **设备指纹算法**通过User-Agent + IP哈希生成唯一设备标识，防止跨设备刷新：

$deviceHash = hash('sha256',   
 $request->header('User-Agent') .   
 $request->ip()  
);

1. **令牌黑名单机制**主动登出时标记旧令牌：

// 登出接口   
public function logout() {  
 $userId = $this->request->user->id;  
 $accessToken = $this->request->header('Authorization');  
 Redis::sAdd("user:blacklist:{$userId}", $accessToken);  
 Redis::del("user:token:{$userId}");  
}

1. **刷新频率限制**Nginx层限制接口调用频率：

location /auth/refresh {  
 limit\_req zone=auth burst=3 nodelay;  
 proxy\_pass http://backend;  
}

通过此案例可清晰理解：**Refresh Token本质是通过长期令牌换取短期令牌的安全凭证**，配合Redis实现状态管理，既保障了无状态架构特性，又增强了系统安全性。