服务器响应令牌 access 和 refresh 在调用接口的时候如何使用

JWT Access Token 和 Refresh Token 使用指南

在 RESTful API 开发中，JWT 的 Access Token 和 Refresh Token 是协同工作的，它们的使用方式如下：

令牌的基本使用流程

1. 获取初始令牌

```Http
POST /api/token/
Content-Type: application/json

{
  "username": "your_username",
  "password": "your_password"
}
```

2. 服务器响应

```Json
{
  "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  // Access Token
  "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." // Refresh Token
}
```

Access Token 的使用方式

Access Token 用于访问受保护的 API 资源：

1. 在请求头中添加 Authorization

```Http
GET /api/protected-resource/
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

2. 后端验证流程

1. 解析 Authorization 头
2. 验证 JWT 签名
3. 检查令牌是否过期
4. 检查用户权限
5. 返回请求的资源

Refresh Token 的使用方式

当 Access Token 过期时，使用 Refresh Token 获取新的 Access Token：

1. 发送刷新请求

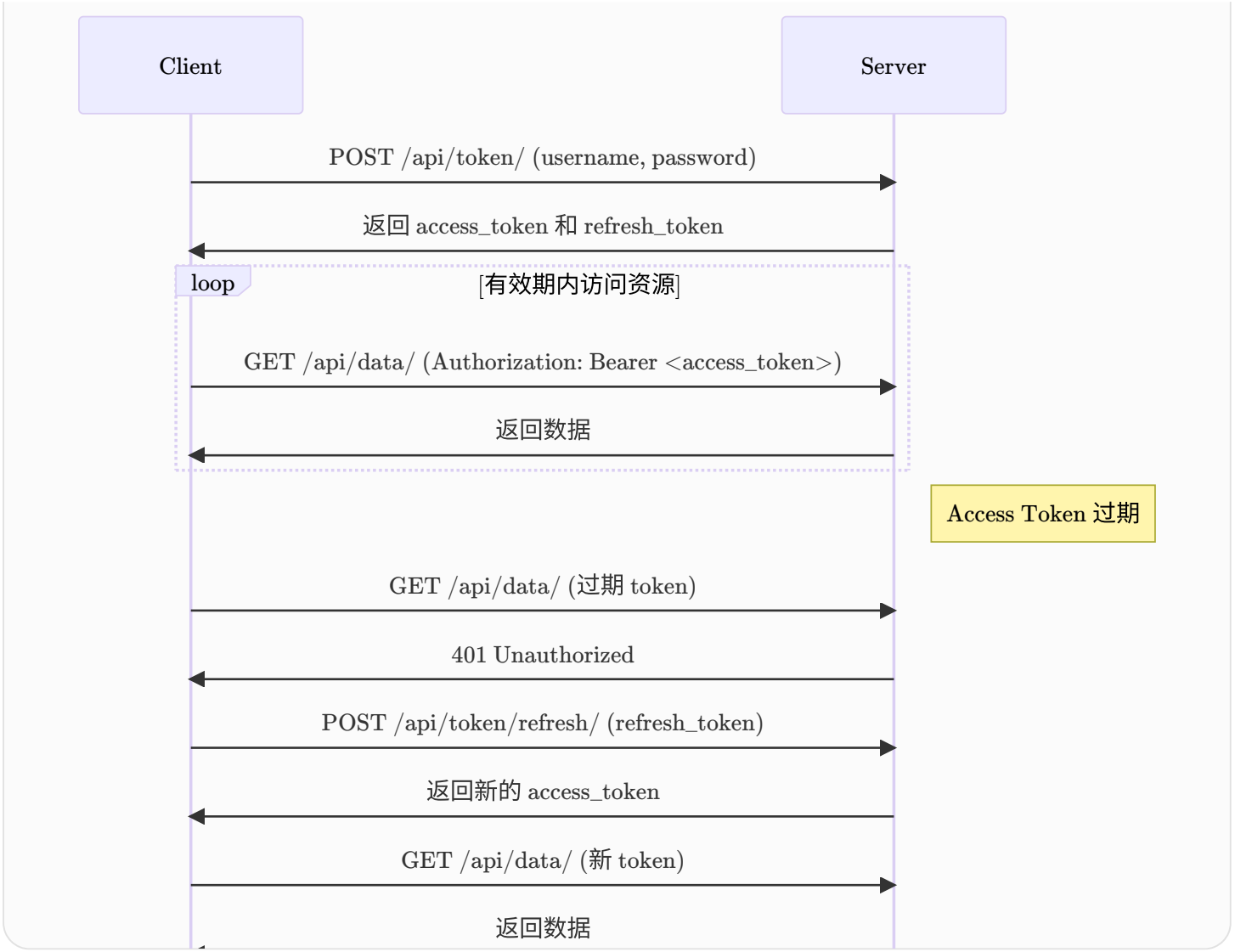
```Http
POST /api/token/refresh/
Content-Type: application/json

{
  "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

2. 服务器响应

```Json
{
  "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." // 新的 Access Token
}
```

完整生命周期示例

图表 代码 下载



前端实现示例

使用 Axios 封装请求

JavaScript

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://your-api.com/api/',
});

// 添加请求拦截器
api.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
}, error => {
  return Promise.reject(error);
});

// 添加响应拦截器 - 处理 token 过期
api.interceptors.response.use(response => response, async error => {
  const originalRequest = error.config;

  // 如果是401错误且未重试过
  if (error.response.status === 401 && !originalRequest._retry) {
    originalRequest._retry = true;

    try {
      // 使用 refresh token 获取新的 access token
      const refreshToken = localStorage.getItem('refresh_token');
      const response = await axios.post('/api/token/refresh/', { refresh: refreshToken });

      // 保存新的 access token
      localStorage.setItem('access_token', response.data.access);

      // 更新请求头并重试原始请求
      originalRequest.headers.Authorization = `Bearer ${response.data.access}`;
      return api(originalRequest);
    } catch (refreshError) {
      // 刷新失败，跳转到登录页
      localStorage.removeItem('access_token');
      localStorage.removeItem('refresh_token');
      window.location = '/login';
      return Promise.reject(refreshError);
    }
  }

  return Promise.reject(error);
});

export default api;
```

安全最佳实践

1. 令牌存储策略

- **浏览器环境**：
  - Access Token：存储在内存或 sessionStorage（短期）
  - Refresh Token：存储在 HttpOnly Cookie 或安全存储中
- **移动应用**：
  - 使用安全存储（Keychain/Keystore）

2. 令牌传输安全

- 始终使用 HTTPS
- 避免在 URL 参数中传递令牌
- 使用合适的 CORS 策略

3. 令牌轮换策略

```python
# settings.py
SIMPLE_JWT = {
    'ROTATE_REFRESH_TOKENS': True,  # 刷新时生成新的 Refresh Token
    'BLACKLIST_AFTER_ROTATION': True,  # 将旧令牌加入黑名单
}
```

4. 令牌失效场景

- 用户主动登出
- 检测到可疑活动
- 密码更改
- 账户被禁用

令牌失效处理

1. 登出实现

```python
```

```python
# views.py
from rest_framework_simplejwt.tokens import RefreshToken
from rest_framework.response import Response
from rest_framework.views import APIView

class LogoutView(APIView):
    def post(self, request):
        try:
            refresh_token = request.data["refresh"]
            token = RefreshToken(refresh_token)
            token.blacklist()  # 将令牌加入黑名单
            return Response(status=204)
        except Exception as e:
            return Response({"error": "Invalid token"}, status=400)
```

2. 令牌黑名单管理

Python

```python
# models.py
from rest_framework_simplejwt.token_blacklist.models import OutstandingToken, BlacklistedToken

def revoke_all_tokens(user):
    # 撤销用户所有令牌
    tokens = OutstandingToken.objects.filter(user=user)
    for token in tokens:
        BlacklistedToken.objects.get_or_create(token=token)
```

令牌有效期策略

令牌类型	推荐有效期	用途
Access Token	15-30分钟	访问受保护资源
Refresh Token	7-30天	获取新的 Access Token

常见问题解决方案

1. Access Token 过期后如何自动刷新?

- 使用 Axios 拦截器（如上所示）
- 在响应 401 错误时自动尝试刷新

2. 如何同时支持多个设备?

- 每个设备维护独立的 Refresh Token
- 用户界面显示所有活动设备

- 允许用户远程撤销设备令牌

3. 如何实现”记住我”功能?

Python

```python
# 登录视图
class LoginView(APIView):
    def post(self, request):
        # ...
        remember_me = request.data.get('remember_me', False)

        if remember_me:
            # 设置较长的 Refresh Token 有效期
            refresh = RefreshToken.for_user(user)
            refresh.set_exp(lifetime=timedelta(days=30))
        else:
            # 默认有效期
            refresh = RefreshToken.for_user(user)

        # ...
```

4. 如何在前端处理令牌过期?

JavaScript

```
// 检查令牌是否即将过期
function isTokenExpiringSoon(token) {
  if (!token) return true;

  try {
    const payload = JSON.parse(atob(token.split('.')[1]));
    const exp = payload.exp * 1000; // 转换为毫秒
    const now = Date.now();
    const buffer = 30000; // 30秒缓冲

    return (exp - now) < buffer;
  } catch (e) {
    return true;
  }
}

// 在请求前检查令牌
async function makeRequest() {
  let token = localStorage.getItem('access_token');

  if (isTokenExpiringSoon(token)) {
    try {
      const newToken = await refreshToken();
      token = newToken;
    } catch (e) {
      // 处理刷新失败
    }
  }

  // 使用 token 发送请求
}
```

总结

1. **Access Token 用于访问受保护资源**：
   - 添加到 `Authorization: Bearer <token>` 头中
   - 有效期短，需要定期刷新
2. **Refresh Token 用于获取新的 Access Token**：
   - 通过专门的刷新端点使用
   - 存储安全，有效期较长
3. **最佳实践**：
   - 使用 HTTPS 传输
   - 安全存储 Refresh Token
   - 实现令牌轮换和黑名单

- 在前端处理自动刷新

4. **安全增强**：
   - 添加 IP 绑定
   - 实现设备指纹
   - 监控异常活动

通过合理使用 Access Token 和 Refresh Token，可以在保证安全性的同时提供良好的用户体验，实现无状态的身份验证机制。