2-14 创建OAuth2Filter类

> 注意事项：
>
> 因为在 `OAuth2Filter` 类中要读写 `ThreadLocal` 中的数据，所以 `OAuth2Filter` 类必须要设置成多例的，否则 `ThreadLocal` 将无法使用。

在配置文件中，添加JWT需要用到的密钥、过期时间和缓存过期时间。

```
emos:
  jwt:
    #密钥
    secret: abc123456
    #令牌过期时间（天）
    expire:  5
    #令牌缓存时间（天数）
    cache-expire: 10
```

在 `com.example.emos.wx.config.shiro` 中创建 `OAuth2Filter` 类。

```java
package com.example.emos.wx.config.shiro;
import com.auth0.jwt.exceptions.JWTDecodeException;
import com.auth0.jwt.exceptions.TokenExpiredException;
import org.apache.commons.lang3.StringUtils;
import org.apache.http.HttpStatus;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.web.filter.authc.AuthenticatingFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Scope;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.RequestMethod;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
@Component
@Scope("prototype")
public class OAuth2Filter extends AuthenticatingFilter {
    @Autowired
    private ThreadLocalToken threadLocalToken;
    @Value("${emos.jwt.cache-expire}")
    private int cacheExpire;
    @Autowired
    private JwtUtil jwtUtil;
    @Autowired
    private RedisTemplate redisTemplate;
    /**
     * 拦截请求之后，用于把令牌字符串封装成令牌对象
     */
    @Override
    protected AuthenticationToken createToken(ServletRequest request,
        ServletResponse response) throws Exception {
        //获取请求token
        String token = getRequestToken((HttpServletRequest) request);
        if (StringUtils.isBlank(token)) {
            return null;
        }
        return new OAuth2Token(token);
    }
    /**
     * 拦截请求，判断请求是否需要被Shiro处理
     */
    @Override
    protected boolean isAccessAllowed(ServletRequest request,
        ServletResponse response, Object mappedValue) {
        HttpServletRequest req = (HttpServletRequest) request;
        // Ajax提交application/json数据的时候，会先发出Options请求
        // 这里要放行Options请求，不需要Shiro处理
```

```
56.         if (req.getMethod().equals(RequestMethod.OPTIONS.name())) {
57.             return true;
58.         }
59.         // 除了Options请求之外，所有请求都要被Shiro处理
60.         return false;
61.     }
62.     /**
63.      * 该方法用于处理所有应该被Shiro处理的请求
64.      */
65.     @Override
66.     protected boolean onAccessDenied(ServletRequest request,
67.         ServletResponse response) throws Exception {
68.         HttpServletRequest req = (HttpServletRequest) request;
69.         HttpServletResponse resp = (HttpServletResponse) response;
70.         resp.setHeader("Content-Type", "text/html;charset=UTF-8");
71.         //允许跨域请求
72.         resp.setHeader("Access-Control-Allow-Credentials", "true");
73.         resp.setHeader("Access-Control-Allow-Origin", req.getHeader("Origin"));
74.         threadLocalToken.clear();
75.         //获取请求token，如果token不存在，直接返回401
76.         String token = getRequestToken((HttpServletRequest) request);
77.         if (StringUtils.isBlank(token)) {
78.             resp.setStatus(HttpStatus.SC_UNAUTHORIZED);
79.             resp.getWriter().print("无效的令牌");
80.             return false;
81.         }
82.         try {
83.             jwtUtil.verifierToken(token); //检查令牌是否过期
84.         } catch (TokenExpiredException e) {
85.             //客户端令牌过期，查询Redis中是否存在令牌，如果存在令牌就重新生成一个令牌给客户端
86.             if (redisTemplate.hasKey(token)) {
87.                 redisTemplate.delete(token);//删除令牌
88.                 int userId = jwtUtil.getUserId(token);
89.                 token = jwtUtil.createToken(userId);  //生成新的令牌
90.                 //把新的令牌保存到Redis中
91.                 redisTemplate.opsForValue().set(token, userId + "", cacheExpire,
    TimeUnit.DAYS);
92.                 //把新令牌绑定到线程
93.                 threadLocalToken.setToken(token);
94.             } else {
95.                 //如果Redis不存在令牌，让用户重新登录
96.                 resp.setStatus(HttpStatus.SC_UNAUTHORIZED);
97.                 resp.getWriter().print("令牌已经过期");
98.                 return false;
99.             }
100.         } catch (JWTDecodeException e) {
101.             resp.setStatus(HttpStatus.SC_UNAUTHORIZED);
102.             resp.getWriter().print("无效的令牌");
103.             return false;
104.         }
105.         boolean bool = executeLogin(request, response);
106.         return bool;
107.     }
108.     @Override
109.     protected boolean onLoginFailure(AuthenticationToken token,
```

```
110.          AuthenticationException e, ServletRequest request, ServletResponse response) {
111.          HttpServletRequest req = (HttpServletRequest) request;
112.          HttpServletResponse resp = (HttpServletResponse) response;
113.          resp.setStatus(HttpStatus.SC_UNAUTHORIZED);
114.          resp.setContentType("application/json;charset=utf-8");
115.          resp.setHeader("Access-Control-Allow-Credentials", "true");
116.          resp.setHeader("Access-Control-Allow-Origin", req.getHeader("Origin"));
117.          try {
118.              resp.getWriter().print(e.getMessage());
119.          } catch (IOException exception) {
120.          }
121.          return false;
122.      }
123.      /**
124.       * 获取请求头里面的token
125.       */
126.      private String getRequestToken(HttpServletRequest httpRequest) {
127.          //从header中获取token
128.          String token = httpRequest.getHeader("token");
129.          //如果header中不存在token，则从参数中获取token
130.          if (StringUtils.isBlank(token)) {
131.              token = httpRequest.getParameter("token");
132.          }
133.          return token;
134.      }
135.      @Override
136.      public void doFilterInternal(ServletRequest request,
137.          ServletResponse response, FilterChain chain) throws ServletException, IOException
      {
138.          super.doFilterInternal(request, response, chain);
139.      }
140. }
```