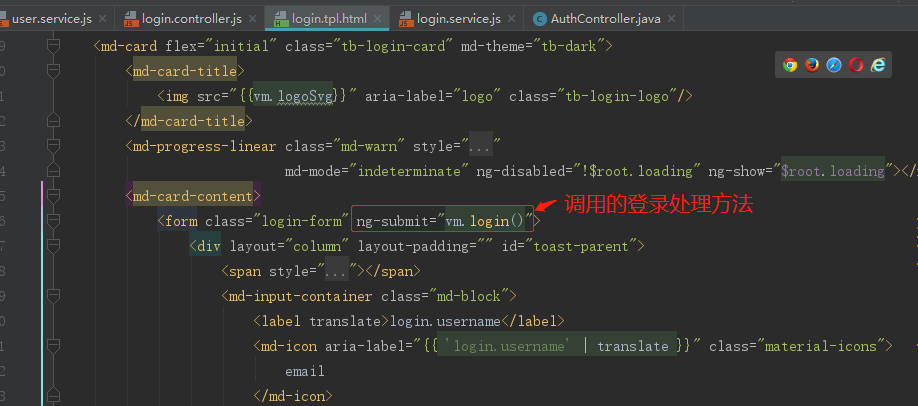
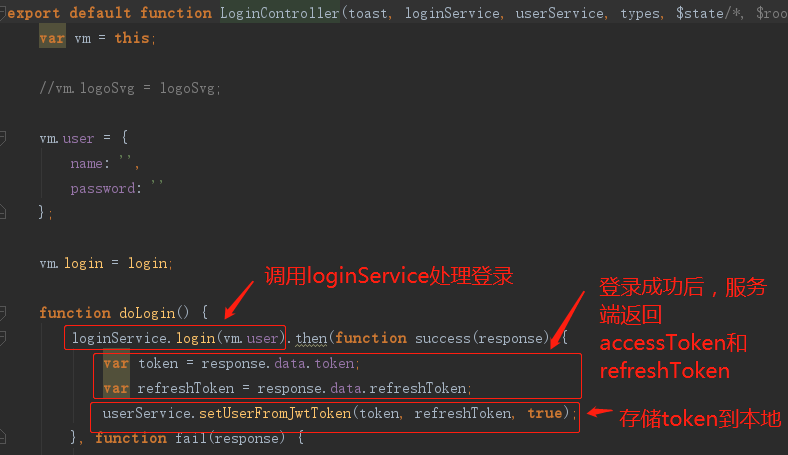
Thingsboard jwt Token登录验证流程

## 客户端请求登录

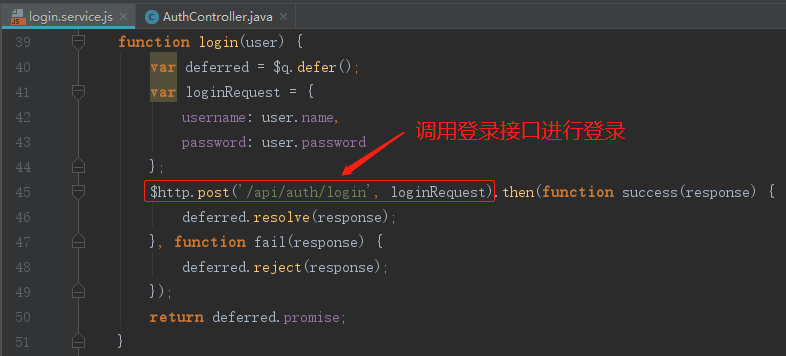
### 1.1、登录页面入口



### 1.2、login.controller.js处理登录



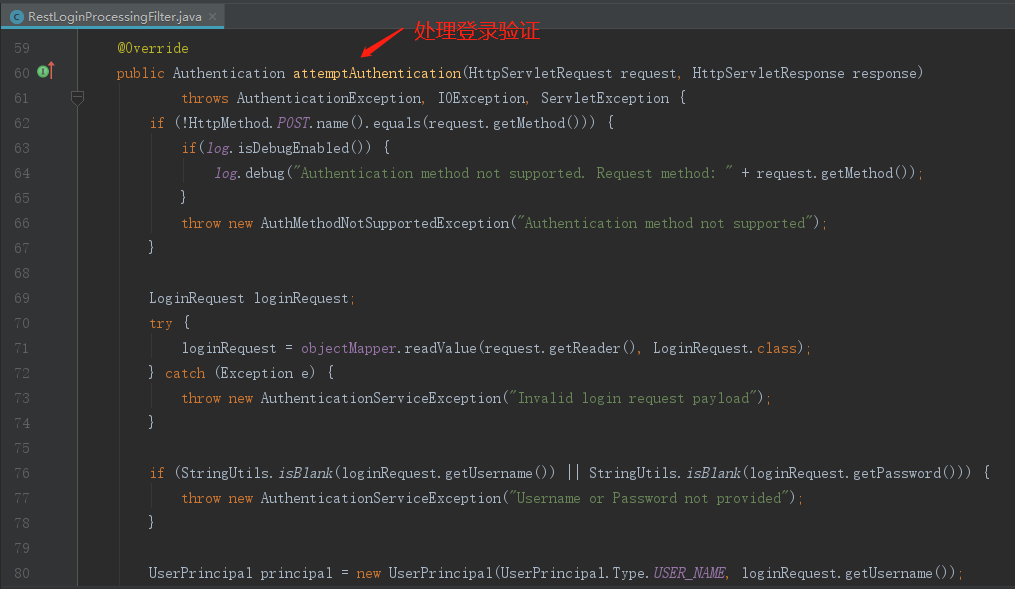
login.controller.js内部调用了login.service.js的login方法处理登录，login方法内部组装请求参数，调用登录URL接口发起请求，如下图所示：



## 服务端处理登录

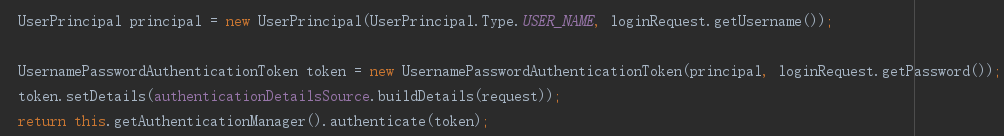
### 2.1、RestLoginProcessingFilter处理登录请求

1.在attemptAuthentication方法中进行登录用户名和密码的验证

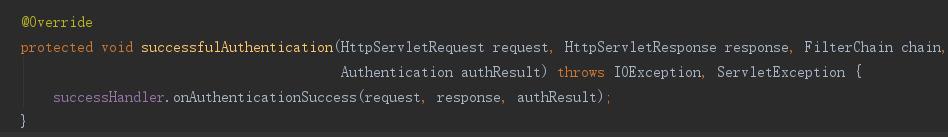


attemptAuthentication方法中验证用户名和密码的核心代码如下：

|  |
| --- |
| UserPrincipal principal = new UserPrincipal(UserPrincipal.Type.*USER\_NAME*, loginRequest.getUsername());  UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(principal, loginRequest.getPassword()); token.setDetails(authenticationDetailsSource.buildDetails(request)); return this.getAuthenticationManager().authenticate(token); |

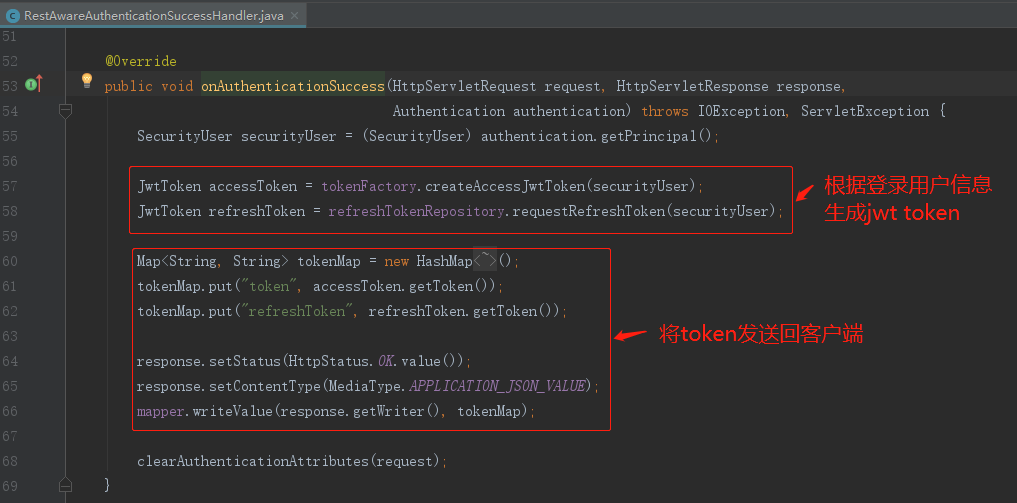


2.登录验证成功后回调successfulAuthentication方法



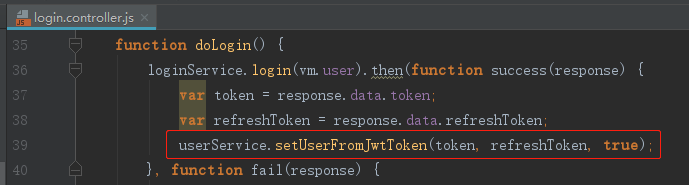
### 2.2、RestAwareAuthenticationSuccessHandler返回登录Token

successfulAuthentication方法内部调用了AuthenticationSuccessHandler的onAuthenticationSuccess方法处理登录成功后的业务处理，在onAuthenticationSuccess方法内部将根据当前登录用户生成jwt Token，然后将Token 发送回客户端，如下图所示：



RestAwareAuthenticationSuccessHandler是AuthenticationSuccessHandler接口的具体实现类。

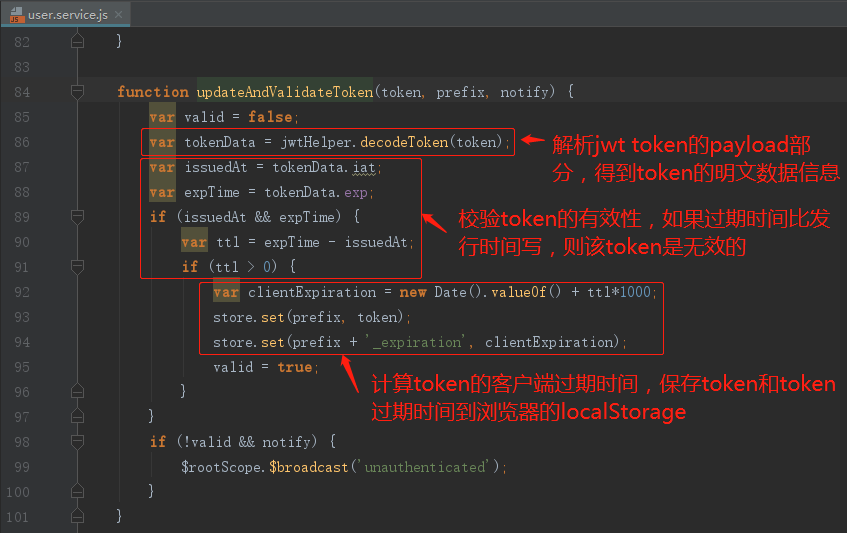
## 客户端保存登录Token

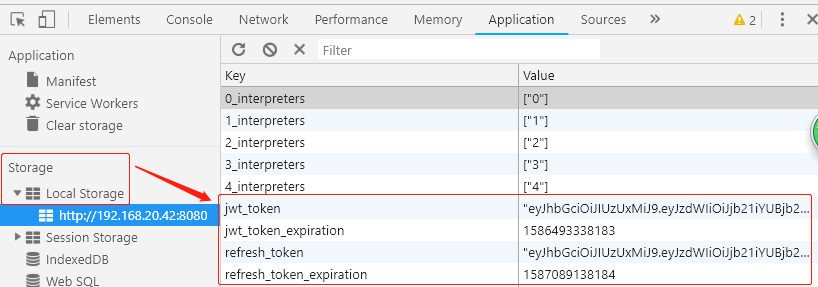


服务端返回accessToken和refreshToken之后，login.controller.js内部调用了user.service.js里面的setUserFromJwtToken方法保存Token，setUserFromJwtToken方法的代码如下：

|  |
| --- |
| **function** setUserFromJwtToken(jwtToken, refreshToken, notify, doLogout) {  currentUser = **null**;  currentUserDetails = **null**;  lastPublicDashboardId = **null**;  userTokenAccessEnabled = **false**;  allowedDashboardIds = [];  **if** (!jwtToken) {  clearTokenData();  **if** (notify) {  $rootScope.$broadcast('unauthenticated', doLogout);  }  } **else** {  updateAndValidateToken(jwtToken, 'jwt\_token', **true**);  updateAndValidateToken(refreshToken, 'refresh\_token', **true**);  **if** (notify) {  loadUser(**false**).then(**function** success() {  $rootScope.$broadcast('authenticated');  }, **function** fail() {  $rootScope.$broadcast('unauthenticated');  });  } **else** {  loadUser(**false**);  }  } } |

重点看updateAndValidateToken这个方法，在该方法内部会校验Token的有效性，计算客户端过期时间，保存Token到localStorage，如下图所示：

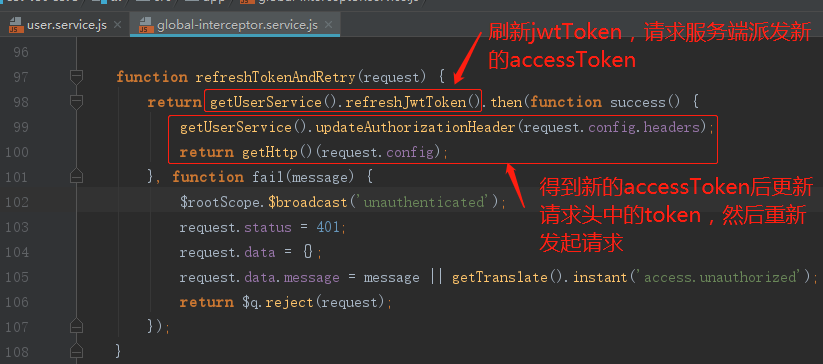


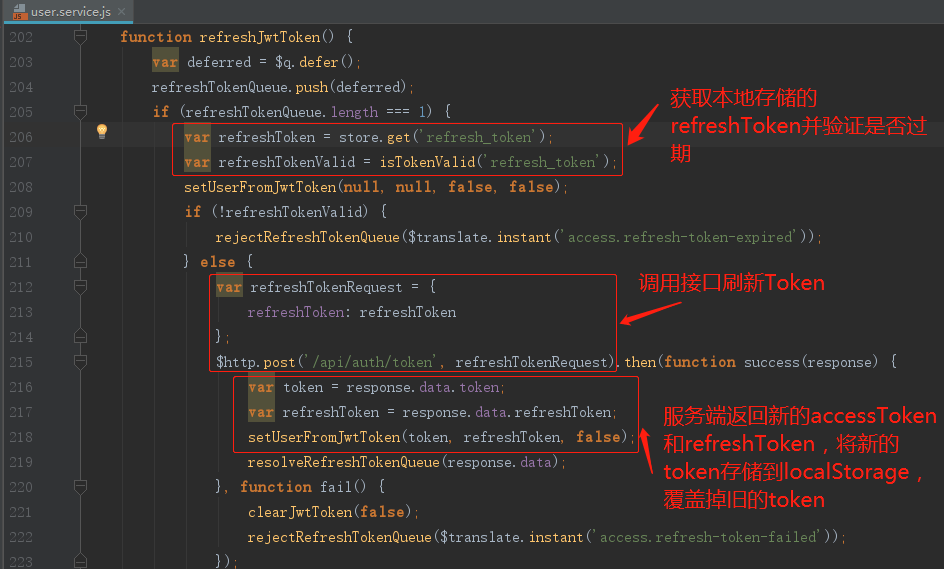


## 客户端GlobalInterceptor拦截器刷新Token

global-interceptor.service.js定义了一个GlobalInterceptor全局拦截器，拦截客户端发起的http请求以及服务端返回的响应数据，当客户端检查到AccessToken已过期或者服务端返回AccessToken过期时，GlobalInterceptor就会通过refreshToken去服务端请求新的AcessToken，得到新的AcessToken后再尝试发起上一次的Http请求，如下所示：

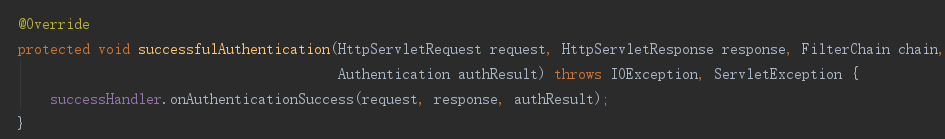






## 服务端刷新Token

在RefreshTokenProcessingFilter的attemptAuthentication方法中验证客户端refreshToken的合法性，校验通过后回调successfulAuthentication方法派发新的token给客户端。



successfulAuthentication方法内部调用的也是RestAwareAuthenticationSuccessHandler的onAuthenticationSuccess方法，由onAuthenticationSuccess方法负责生成accessToken和refreshToken，然后发送给客户端。

