Authentication Solution Design

TODO

- [ ] Test Plan
- [ ] my-ifastpay-ws 改动
- [ ] my-ifast-pay 改动
- [ ] 验证是否用过 fsm-client 就可以了，还是有其他需要配置的？（需要了解底层机制）
- [ ] 验证当前登录 & 验证 (2FA, Authentication)；验证“只要持有 Token，任何人都可以通过 FSMOne 的应用程序进入某用户的 iFastPay”

1. 需求 Requirement

1. 提供登录/验证（Login/Authentication）机制
2. 确保可以同时登出（my-web-fsmone 和 my-ifastpay-ws）
3. 最大登录次数（只要在 Device A 登录了 my-ifast-pay，那 Device B 就会自动登出；my-web-fsmone 同理）

2. 系统设计 System Design

2.1. 系统概览

由于 ifastpay 具有以下特点：

1. 必须先登录 FSMOne 才能登录 iFastPay
2. iFastPay 的 Session Lifecycle 必须和 FSMOne 一致
3. iFastPay 要允许用户使用 FSMOne 的账户登录（伪 SSO）

所以宏观设计暂定如下：

1. 不为 iFastPay 设计登录功能；只通过 FSMOne 的 Token 登录，共用 FSMOne 的 Session（通过共享 client.id）
2. iFastPay 会和 FSMOne 共用 Token；iFastPay 会通过 Token 向 my-oauth-ws 发起查询，确认 Session 合法性，并通过 Refresh Token 刷新 Access Token。

安全风险与解决方案

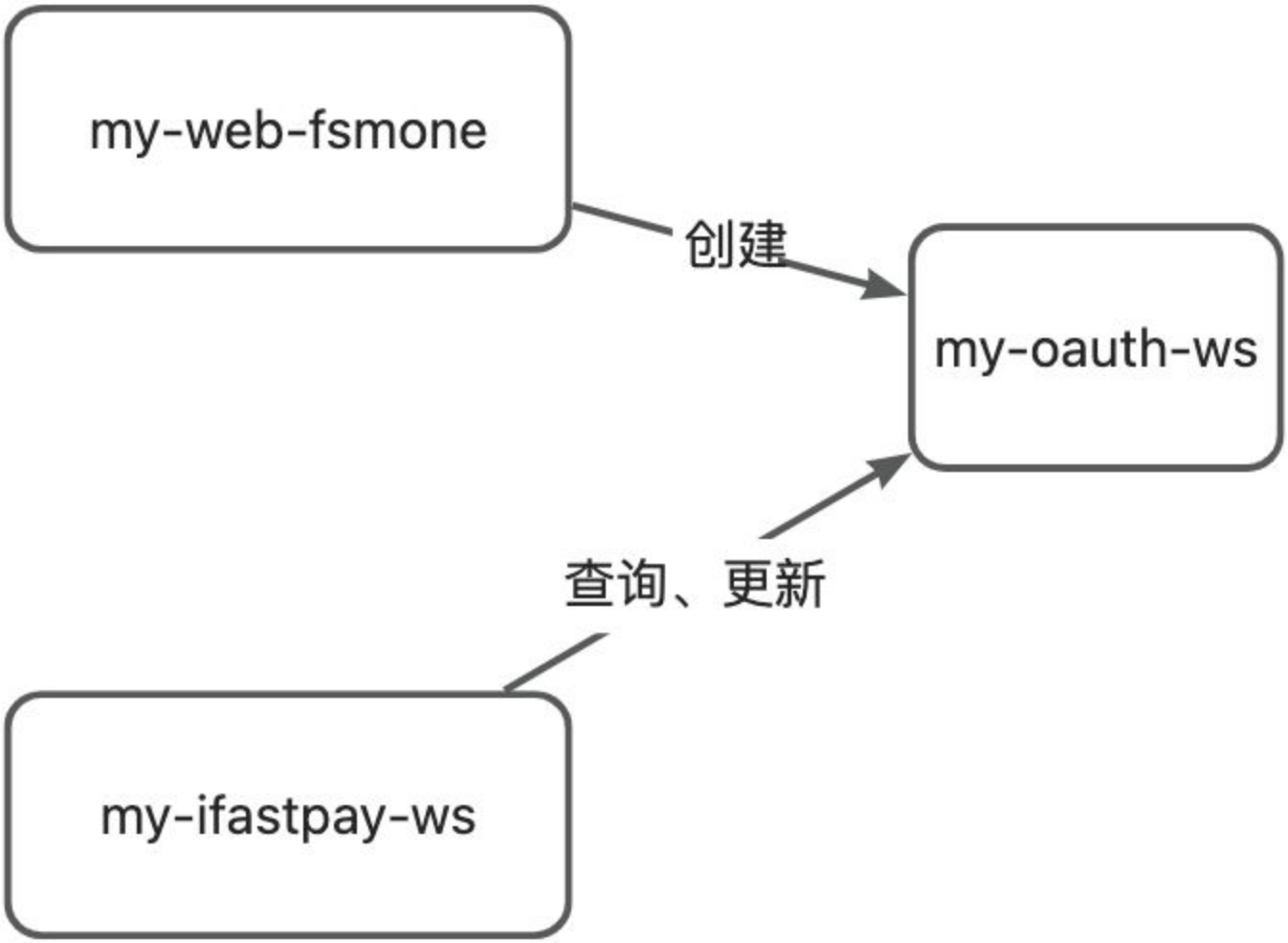
风险	解决方案

只要持有 Token，任何人都可以通过 FSMOne 的应用程序进入某用户的 iFastPay	1. 需要确认是否存在着风险；有的话，现有的 APP 又是怎么处理风险的。 2. 使用 DPoP + Device Binding (Truested Device) 进行验证，对用户身份进行验证。

由于 my-oauth-ws 已经提供了以下功能：

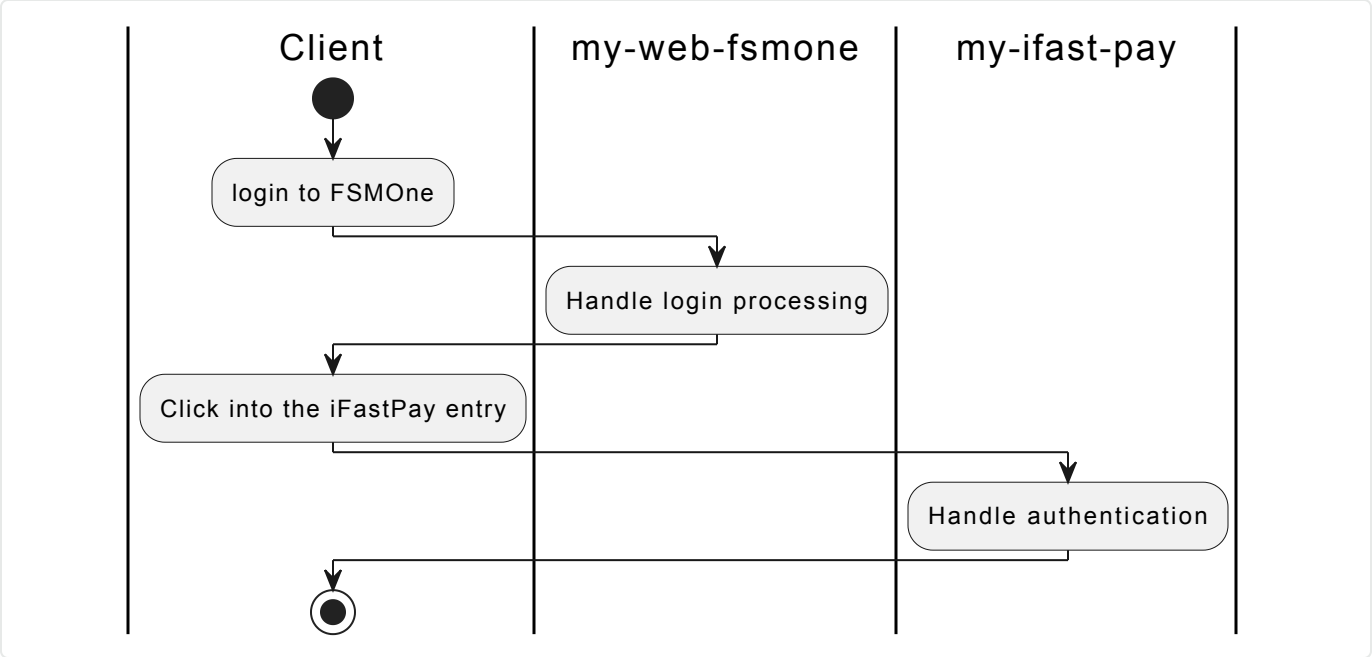
1. 最大登录次数（Maximum Concurrent Login）
2. 会话管理（Session Management）如：创建 Token、验证 Token、刷新 Token、Revoke token。

所以本次设计重点将是借鉴现有的代码库（my-web-fsmone），并将代码移植到 my-ifastpay-ws 上。接下来将会列出需要/不需要配置的条目及其原因，并且为上面的推断提供确实证据。

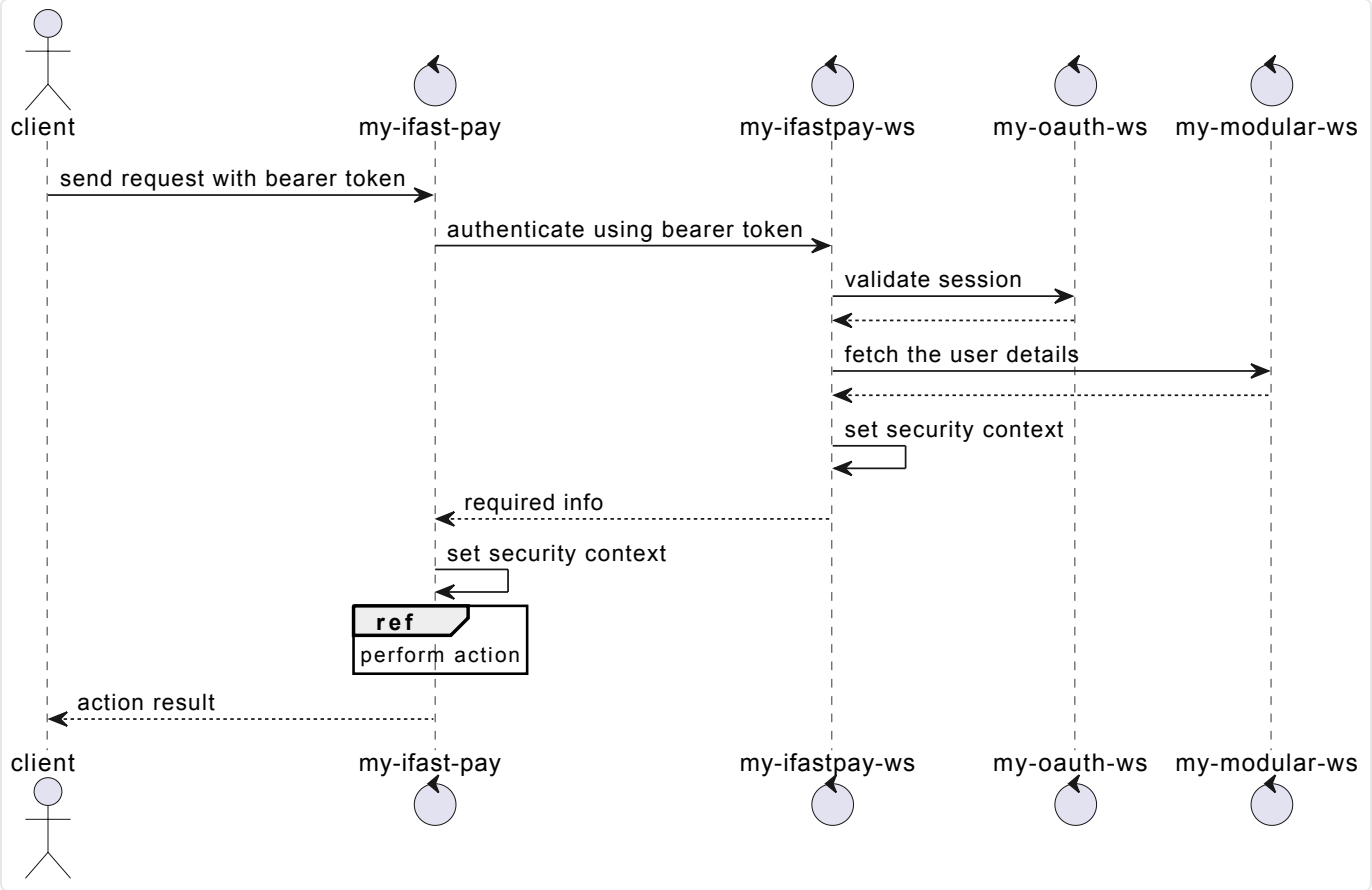


2.1.1. 活动图 Activity Diagram

对用户来说，他只需要登录 FSMOne 就可以了；其他的鉴权认证（Authentication/Authorization）都是不被感知的。



2.1.2. 流程图 Sequence Diagram



2.2. `my-ifastpay-ws` 改动

- my-oauth-ws 信息配置：无需改动；复用 fsm-client。文件：my-oauth-ws/create.sql
- **其他配置项**：参考 my-web-fsmone/ build-properties 和 my-web-fsmone/src/main/resources/application.properties

需要配置	原因
my-web-fsmone/src/main/java/com/fmy/config	
WebApplicationConfig	
SSLConfig	
CorsFilter	
AkamaiResponseFilter	
my-web-fsmone/src/main/java/com/fmy/config/security	
WebSecurityConfig	
SetRequestFilter	
FsmUserDetails	
FsmUserAuthenticationModel	
FsmLogoutSuccessHandler	
FsmLoginFailureHandler	
FsmLoginSuccessHandler	
FsmGrantedAuthority	
DefaultExceptionHandler	
my-web-fsmone/src/main/java/com/fmy/service	
RestTemplateServiceImpl	
AuthenticateService	
PasswordService	
AccountService	
my-web-fsmone/src/main/java/com/fmy/util	

AccountLoginValidator	
LoginUserDetailGetter	
ResponseUtil	
SecurityUtil	
my-web-fsmone/src/main/java/com/fmy/aop/security	
PasswordValidatorAop	
ValidatatePassword	
ValidateType	
ParameterValidatorAop	
ResponseValidatorAop	
AopConfig	

不需要配置	原因
SwaggerConfig	
RegionalMobileConfig	

2.3. `my-ifast-pay` 改动

挑战有二：

1. 基本机制 & 接入思路
2. 如何不影响日常开发?
3. 如何兼容 my-ifastpay-ws 和 my-ifast-pay 的 Java 版本（8/21）
4. 具体要怎么接入?

2.4. 测试分析 Testing Plan

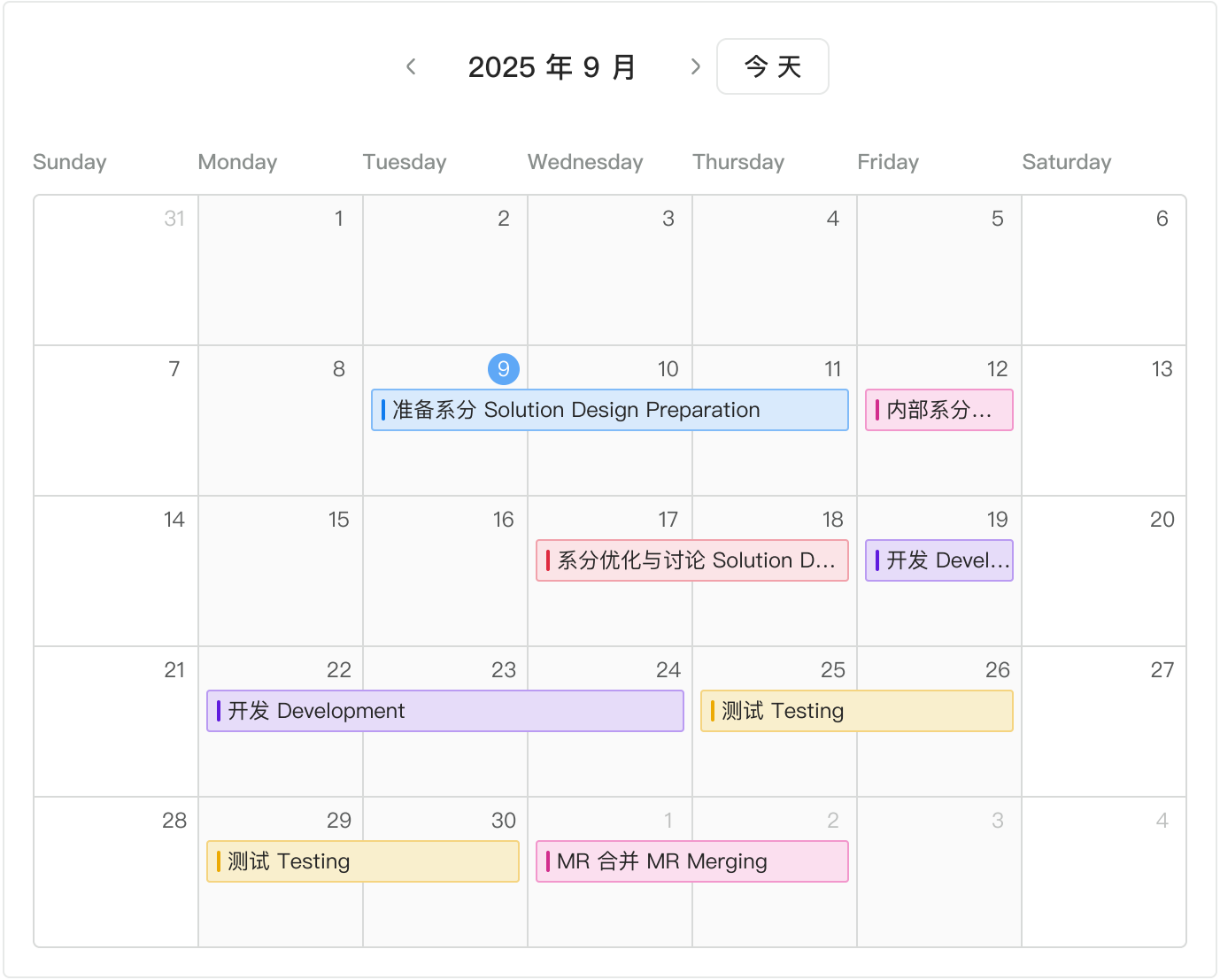
2.4.1. 宏观分析

1. my-ifastpay-ws 本地测试：MockMvc
2. my-ifast-pay 本地测试：MockMvc
3. 联调测试

2.4.2. 具体测分

1. 提供登录/验证（Login/Authentication）机制
2. 确保可以同时登出（my-web-fsmone 和 my-ifastpay-ws）
3. 最大登录次数（只要在 Device A 登录了 my-ifast-pay，那 Device B 就会自动登出；my-web-fsmone 同理）

3. 日程安排



4. 代码学习

4.1. 会话管理 Session Management

4.1.1. 同步登录/登出

据目前了解，my-web-fsmone 的确是 Stateless 的，因为他不负责管理 Session。

com.fmy.config.security.WebSecurityConfig Java

```java
http.cors()
    // Other configuration
    .and()
    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
```

但 my-web-fsmone 之所以是 Stateless，并非是因为他使用了 JWT 这种 Self-contained 的验证方法；相反，他把 Session Management 交给了 my-oauth-ws 这个服务进行集中管理。my-oauth-ws 会根据某个 Identifier 对 Token/Session 进行验证，代码如下：

com.fmy.oauth2.config.endpoint.CustomCheckTokenEndpoint Java

```java
private Map<String, ?> checkTokenCommon(String tokenRequestHeader,
                                        String tokenRequestParam,
                                        boolean checkSecret,
                                        boolean checkClient) {
    // Validation...
    if (checkClient) {
        AuthenticationUtil.verifyClientDetails(checkSecret);
    }

    // The value will be either tokenRequestHeader/ tokenRequestParam
    OAuth2AccessToken token = resourceServerTokenServices.readAccessToken(value);

    // Validate token...
    OAuth2Authentication authentication = resourceServerTokenServices.loadAuthentication(
        token.getValue());

    // Response Handling...
    return response;
}
```

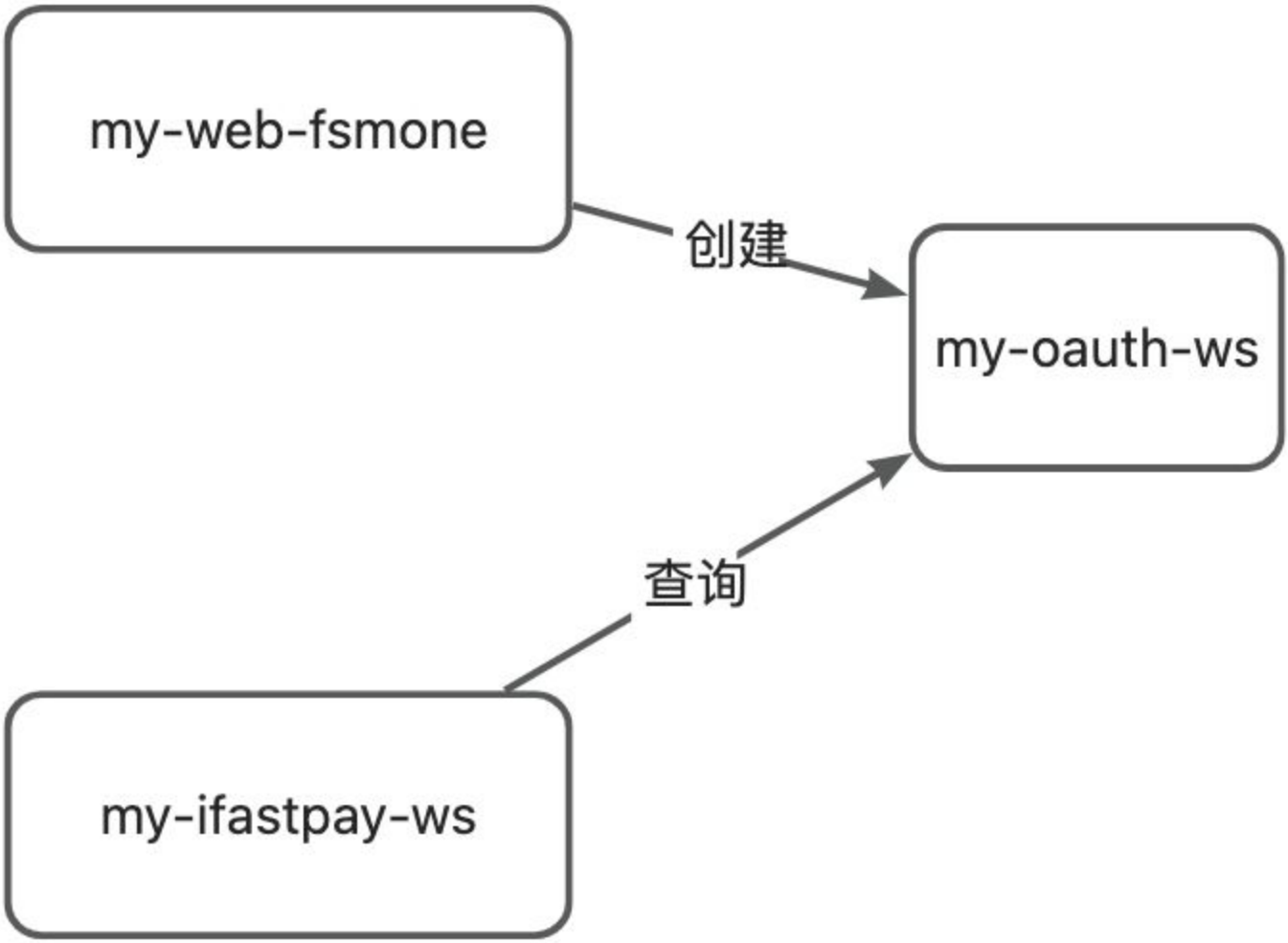
com.fmy.shared.oauth2.service.credential.ClientDetailsHelperService Java

```java
public ClientDetails getClientDetailsFromAuthHeader(String authHeader, boolean checkSecret) throws InvalidClientException {
    // Parameter handling...
    String[] clientIdInfo = decodedHeader.split(":");
    ClientDetails clientDetails = clientDetailsService.loadClientByClientId(clientIdInfo[0]);

    // Validate ClientDetails...

    // Validate client's secret...
    return clientDetails;
}
```

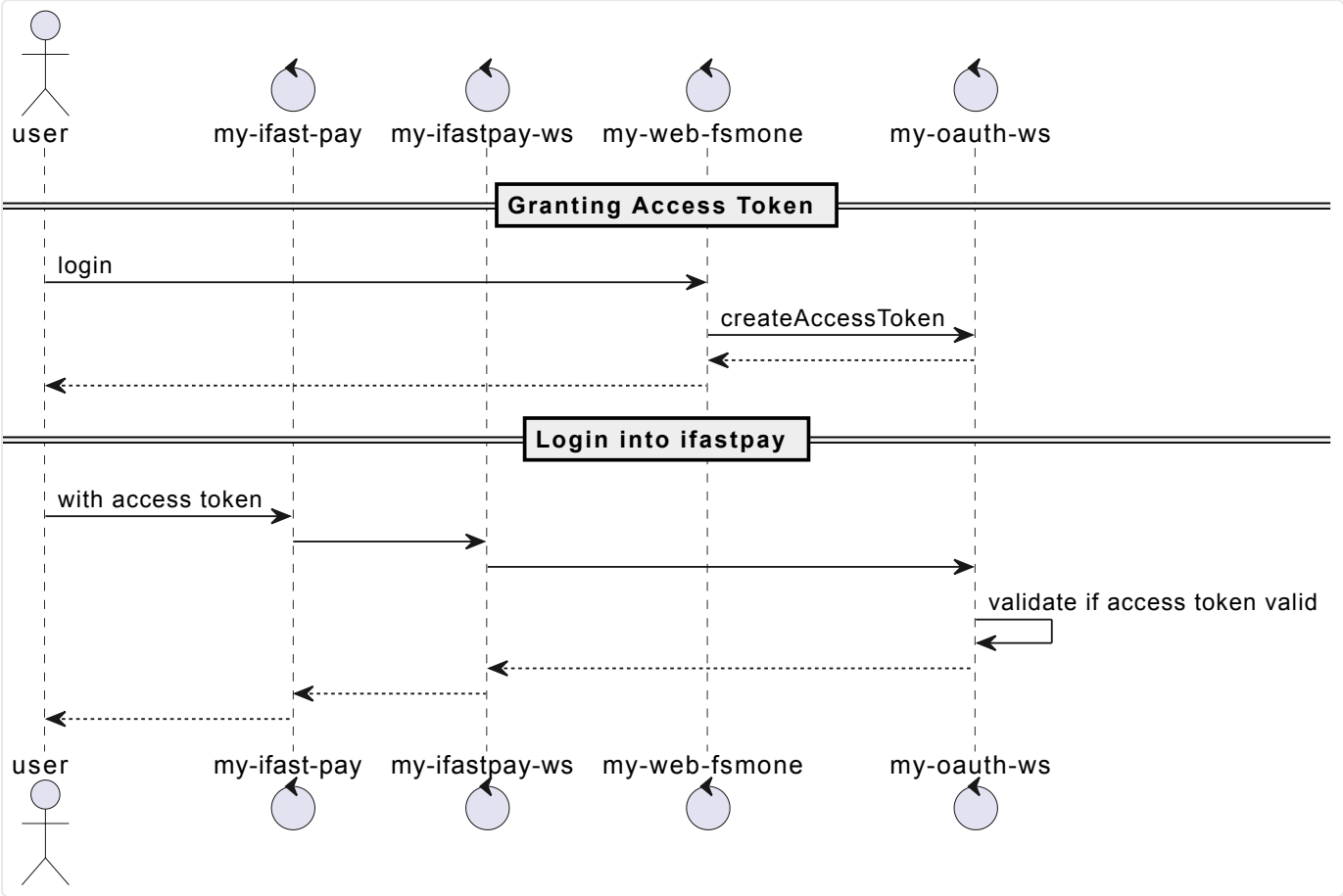
根据这两段代码，我们可以得知两件事：

1. my-oauth-ws 会对 Client（内部服务）进行身份验证
2. readAccessToken 只会用到 acces token

从中我们能推断出，my-oauth-ws 本身就已经提供了 Session 同步 Login/Logout 的解法，概念如下：



流程图：



大概这样；无论是 Revocation 还是 Expirarion，my-ifastpay-ws/ my-web-fsmone 都可以同步更新。

4.1.2. 账号单一登录

第二个问题，就是一个账号只可以登录一次，比如在 Device A 登录之后，Device B 就会登出。

目前来说，对于单一服务而言，是可以做大 Max Concurrent Login 的，也就是如果我在 Device A 登录了 my-ifastpay，那我原本的 Device B 就会自动 Logout

目前 my-oauth-ws 已经提供同个 client（my-web-fsmone/ my-ifastpay-ws）内最大登录数的能力，如下：

com.fmy.shared.oauth2.core.token.CustomDefaultTokenService Java

```java
@Override
    @Transactional
    public OAuth2AccessToken createAccessToken(OAuth2Authentication authentication) throws AuthenticationException {

        String clientId = AuthenticationUtil.getClientId();
        String refreshTokenValue = authentication.getOAuth2Request().getRequestParameters().get("update_validity_refresh_token");
        if (StringUtils.isNotEmpty(refreshTokenValue)) {
            // refresh token 的相关处理逻辑...
        } else {
            oauthTokenService.blacklist(authentication, clientMaxConcurrencyLoginResolverService.resolveConcurrentLogin(clientId));
            ehcacheTokenCacheService.blacklist(authentication, clientMaxConcurrencyLoginResolverService.resolveConcurrentLogin(clientId));
        }
        DefaultOAuth2AccessToken token = (DefaultOAuth2AccessToken) super.createAccessToken(authentication);
        postTokenCreation(authentication, token);
        if(StringUtils.isNotEmpty(refreshTokenValue)) {
            OAuth2Authentication oAuth2Authentication = tokenStoreInjected.readAuthentication(token);
            if(oAuth2Authentication == null) {
                throw new InvalidTokenException(IfastError.Auth.TOKEN_INVALID.getMessage());
            }
        }
        return token;

    }
```

可以看到 oauthTokenService 和 ehcacheTokenCacheService 都有 blacklist 的操作。由于两者操作逻辑相似，我们只看 oauthTokenService 的处理逻辑：

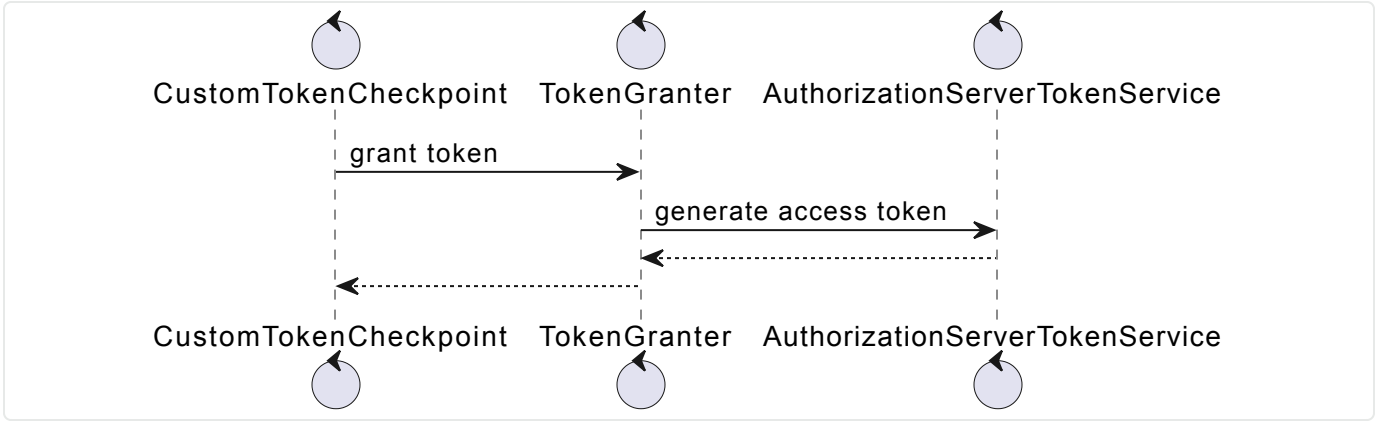
om.fmy.shared.oauth2.service.oauth.OauthTokenServiceImpl Java

```java
@Override
@Transactional(propagation = Propagation.REQUIRES_NEW)
public void blacklist(OAuth2Authentication authentication, int maxConcurrentLogin) {

    log.info("oauth maxConcurrentLogin: {}", maxConcurrentLogin);

    List<OauthTokenBean> oauthTokenBeans = this.findAllActiveToken(authentication);
    log.info("Number of concurrent login: {}", oauthTokenBeans.size());
    if(CollectionUtils.isEmpty(oauthTokenBeans)) {
        return;
    }

    // simple counter to skip current login sessions
    int loginsToSkip = maxConcurrentLogin - 1;
    for(OauthTokenBean oauthTokenBean: oauthTokenBeans) {
        if(loginsToSkip > 0) {
            loginsToSkip --;
            continue;
        }
        cacheToken.invalidate(oauthTokenBean.getJti());
        oauthTokenBean.setIsRevoked(true);
        update(oauthTokenBean);
    }
}
```

这里，findAllActiveToken 会根据返回所有的 ActiveToken，并保留前几个选项，也就是比较迟 Expired 的，然后其他的都一律 Invalidate/Revoke，通过这样的方式进行 Maximum Concurrent Login control。

结论一：经过验证，证实 CustomTokenCheckpoint 会用到 CustomTokenService。其调用链路如下：

当前 Access Token 的分发方式：



备注：CustomTokenCheckpoint 用的是 AuthorizationServerEndpointsConfigurer，有需要参考的话可以看下面配置：

com.fmy.shared.oauth2.config.DefaultAuthServerConfig Java

```java
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    WebResponseExceptionTranslator webResponseExceptionTranslator = applicationContext.getBean(WebResponseExceptionTranslator.class);
    log.info("Registering WebResponseExceptionTranslator: {}", webResponseExceptionTranslator);
    customDefaultTokenServices.setClientDetailsService(clientDetailsService);
    endpoints.tokenStore(tokenStore)
        .accessTokenConverter(accessTokenConverter)
        .authenticationManager(authenticationManager)
        .userDetailsService(customUserDetailService)
        .tokenServices(customDefaultTokenServices)
        .tokenGranter(new CompositeTokenGranter(getTokenGranters()))
        .exceptionTranslator(webResponseExceptionTranslator);
}
```

其中 TokenGranter 可以参考 AbstractTokenGranter，其他的 TokenGranter 也是从 AbstractTokenGranter 演变出来的。

org.springframework.security.oauth2.provider.token.AbstractTokenGranter Java

```java
public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {

    if (!this.grantType.equals(grantType)) {
        return null;
    }

    String clientId = tokenRequest.getClientId();
    ClientDetails client = clientDetailsService.loadClientByClientId(clientId);
    validateGrantType(grantType, client);

    if (logger.isDebugEnabled()) {
        logger.debug("Getting access token for: " + clientId);
    }

    return getAccessToken(client, tokenRequest);

}

protected OAuth2AccessToken getAccessToken(ClientDetails client, TokenRequest tokenRequest) {
    return tokenServices.createAccessToken(getOAuth2Authentication(client, tokenRequest));
}
```

这是 DefaultTokenService 的默认逻辑，但是 CustomDefaultTokenService 重写了这部分逻辑，优化了提取和检查逻辑，提高效率。

org.springframework.security.oauth2.provider.token.DefaultTokenServices Java

```java
@Transactional
public OAuth2AccessToken createAccessToken(OAuth2Authentication authentication) throws AuthenticationException {

    OAuth2AccessToken existingAccessToken = tokenStore.getAccessToken(authentication);
    OAuth2RefreshToken refreshToken = null;
    if (existingAccessToken != null) {
        if (existingAccessToken.isExpired()) {
            if (existingAccessToken.getRefreshToken() != null) {
                refreshToken = existingAccessToken.getRefreshToken();
                // The token store could remove the refresh token when the
                // access token is removed, but we want to
                // be sure...
                tokenStore.removeRefreshToken(refreshToken);
            }
            tokenStore.removeAccessToken(existingAccessToken);
        }
        else {
            // Re-store the access token in case the authentication has changed
            tokenStore.storeAccessToken(existingAccessToken, authentication);
            return existingAccessToken;
        }
    }

    // Omitting...
    return accessToken;
}
```

结论二：验证 Token 时只需要是合法的 Client 就可以了；但如果是刷新/创建，则需要 Client Authenticated = Client ID。

后者的验证分布在两个地方，一个是 Spring Security 框架的 DefaultOAuth2RequestFactory，另一个则是 my-oauth-ws 的 com.fmy.oauth2.config.endpoint.CustomTokenEndpoint#getAccessToken

org.springframework.security.oauth2.provider.request.DefaultOAuth2RequestFactory Java

```java
public TokenRequest createTokenRequest(Map<String, String> requestParameters, ClientDetails authenticatedClient) {

    String clientId = requestParameters.get(OAuth2Utils.CLIENT_ID);
    if (clientId == null) {
        // if the clientId wasn't passed in in the map, we add pull it from the authenticated client object
        clientId = authenticatedClient.getClientId();
    }
    else {
        // otherwise, make sure that they match
        if (!clientId.equals(authenticatedClient.getClientId())) {
            throw new InvalidClientException("Given client ID does not match authenticated client");
        }
    }
    String grantType = requestParameters.get(OAuth2Utils.GRANT_TYPE);

    Set<String> scopes = extractScopes(requestParameters, clientId);
    TokenRequest tokenRequest = new TokenRequest(requestParameters, clientId, scopes, grantType);

    return tokenRequest;
}
```

com.fmy.oauth2.config.endpoint.CustomTokenEndpoint Java

```java
TokenRequest tokenRequest = authorizationServerEndpointsConfiguration.getEndpointsConfigurer()
    .getOAuth2RequestFactory().createTokenRequest(parameters, authenticatedClient);

if (tokenRequest.getGrantType().equals("password")) {
    authoritiesFacade.recacheAuthoritiesByUser(clientId, parameters.get("username"));
}

// Only validate client details if a client is authenticated during this request.
// Double check to make sure that the client ID is the same in the token request and authenticated client.
if (StringUtils.hasText(clientId) && !clientId.equals(tokenRequest.getClientId())) {
    throw new InvalidClientException("Given client ID does not match authenticated client");
}
```

4.2. 当前 my-web-fsmone 的配置

my-web-fsmone/src/main/java/com/fmy/config/security/WebSecurityConfig.java

my-web-fsmone/src/main/java/com/fmy/config/security/WebSecurityConfig.java Java

```java
http.cors()
    .and().csrf().disable()
    .formLogin()
    .loginProcessingUrl("/rest/login")
    .successHandler(fsmLoginSuccessHandler)
    .failureHandler(fsmLoginFailureHandler)
    .and().logout().logoutUrl("/rest/logout").logoutSuccessHandler(fsmLogoutSuccessHandler())
    .and().exceptionHandling().authenticationEntryPoint(this.restAuthenticationEntryPoint)
    .and().authorizeRequests()
    .antMatchers("/**").permitAll()
    .and().requiresChannel().anyRequest().requiresSecure()
    .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(new SetRequestFilter(), UsernamePasswordAuthenticationFilter.class);
http.addFilterBefore(new AkamaiResponseFilter(), UsernamePasswordAuthenticationFilter.class);
http.addFilterBefore(this.oauthAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
```

备注

- 由于 ifastpay 不需要 Login，所以可以忽略有关 Login/Logout 的配置。此外，由于不确定是否使用 CDN，因此可以先忽略 AkamaiResponseFilter。需要配置的选项只有 OAuthAuthenticationFilter 和 authorizeRequests，还有 Stateless 的 Session Management（因为会交给 my-oauth-ws 处理）

my-web-fsmone/src/main/java/com/fmy/config/WebApplicationConfig.java

虽然继承了 WebApplicationConfigBase，但只是定义了一些 Bean，可以忽略。

4.3. 需要用到的 my-oauth-ws endpoint

主要有三个：

- CustomCheckTokenEndpoint
- CustomTokenEndpoint
- RevokeTokenEndpoint

4.4. 学习/补充说明

- SecurityContextHolder.getContext().getAuthentication() 的 authentication 什么时候放进去的? 我需要懂他放的究竟是什么，才可以更好的理解之后的代码，比如是哪一种client；也可以了解自己需要做什么配置。
- shared.core 值得学习

OAuth2RestTemplate

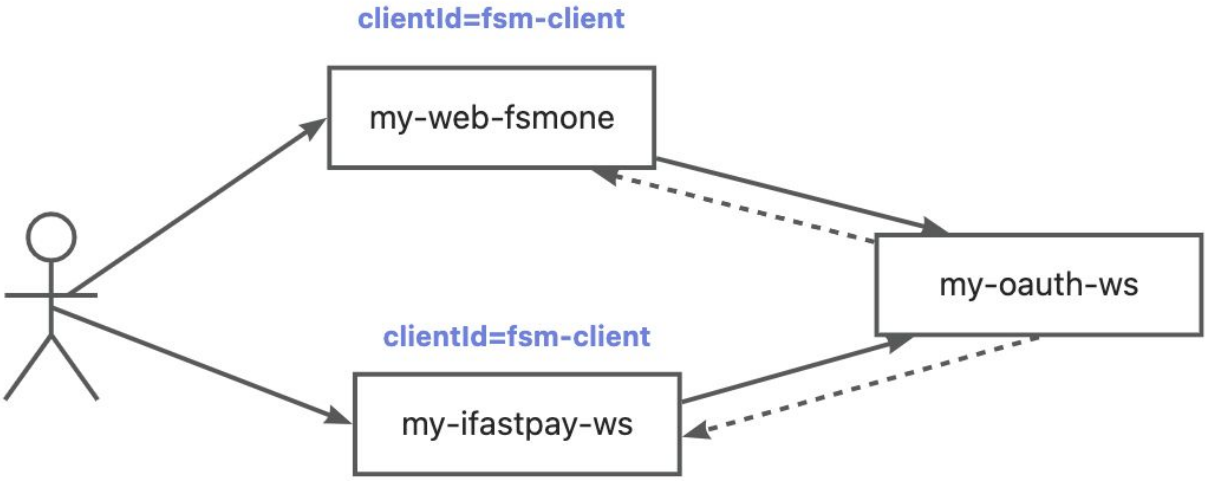
com.fmy.core.oauth.service.rest.OauthRestTemplateServiceImpl Java

```java
@Override
public RestTemplate getRestTemplate() {
    RestTemplate restTemplate = new RestTemplate( new BufferingClientHttpRequestFactory(new SimpleClientHttpRequestFactory()));
    Map<String, String> headerRequestMap = new HashMap<>();
    int refreshTokenValidity = httpServletRequest.getIntHeader(ApplicationUser.Constant.X_REFRESH_TOKEN_VALIDITY);
    //temporary workaround to prevent client logged out during US trading hour on deployment day
    if(DateTimeUtil.getCurrentDateTime().getTime() - DateTimeUtil.getDate(2023, 2, 3, 5, 00, 0).getTime() <= 0) {
        refreshTokenValidity = -1;
    }
    byte[] bytes = Base64.encodeBase64((clientId + (refreshTokenValidity != -1? String.format("-%s", refreshTokenValidity) : "") + ":" + clientSecret).getBytes());
    String code = "Basic " + new String(bytes);
    logger.info("Setting up headers Authorization: {}, Cache_Control: {}, User_Agent: {}", code, "no-cache",httpServletRequest.getHeader(ApplicationUser.Constant.USER_AGENT) );
    List<ClientHttpRequestInterceptor> interceptors = new ArrayList<>();
    headerRequestMap.put(ApplicationUser.Constant.AUTHORIZATION, code);
    headerRequestMap.put(ApplicationUser.Constant.CACHE_CONTROL, "no-cache");
    headerRequestMap.put(ApplicationUser.Constant.USER_AGENT, httpServletRequest.getHeader(ApplicationUser.Constant.USER_AGENT));
    interceptors.add(new HeaderRequestInterceptor(headerRequestMap, httpServletRequest));
    interceptors.add(new ErrorLoggingRequestResponseInterceptor("OauthRestTemplateServiceImpl"));
    interceptors.add(new CorrelationIdInterceptor());
    restTemplate.setInterceptors(interceptors);
    return restTemplate;
}
```

目前想法：可以安排一个 clientId = fsm, 实现 id 共享，这样就不需要额外配置了；

- 0909 学习重点:
  - **目的**：验证为 my-ifastpay-ws 配置 fsm.oauth.client.id=fsm-client 的想法是否可行，因此需要了解 getAccessToken 的相关验证和机制，了解是否还有其他需要配置的项目。
  - **目的**：推进 Solution Design，为 my-ifast-pay 和 my-ifastpay-ws 写出基本的配置项，并从中发现还需要往下确认/不确定的内容，然后消除模糊。

    - **目的**：验证 Solution Design/ 方案的可行性，因此需要准备好 Overall 的 Test Plan (Theoretically & Practically)
    - **目的**：了解整体的 Authentication（+Account）流程，确保整体功能的实现。
- 确认登出情况 这个操作能不能有连带关系？（my-web-fsmone 连带 my-ifast-pay）？



那 clientId 的业务语境 & 含义是什么？可以这样直接用吗？如果之后要其他配置怎么办？

也可以看看 token exchange 的思路。

5. 参考资料

无。