ip：192.168.4.33，port：7878，ctxPath：/

**单点登录后台对接**

1. 用户中心提供登录页面地址

请求地址：http://192.168.4.33:7878/bg/api/systemUsers/login

请求类型：GET请求

请求参数：redirectUrl：（非必需如果不传这个参数，登录成功后跳转到用户中心默认页面）

返回结果：{code=1,msg=”消息内容”}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

1. 获取用户基本信息

请求地址：http://192.168.4.33:7878/bg/api/systemUsers/userInfo

请求类型：GET请求

请求参数：token(必填)

返回结果：{code=1,msg=消息内容”result:}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

1. 用户中心提供退出登录接口

请求地址：http://192.168.4.33:7878/bg/api/systemUsers/logout

请求类型：POST请求

请求参数：token (必填)

返回结果：{code=1,msg=”消息内容”}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

1. 用户中心提供二次校验接口

子系统无法辨别token的有效性，所以当用户访问敏感数据时需要做二次校验

请求地址：http://192.168.4.33:7878/bg/api/systemUsers/checkToken

请求类型：POST请求

请求参数：token (必填)

返回结果：{code=1,msg=”消息内容”}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

1. 用户中心提供获取用户列表接口

请求地址：http://192.168.4.33:7878/bg/api/systemUsers

请求类型：GET请求

请求参数：token (必填)，systemCode（必填，由用户中心分配），pageSize：分页（选填）, currentPage：分页（选填）

返回结果：{code=1,msg=”消息内容”,result:”具体内容”}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容，result:{

[ id:用户id，

account：账号，

name：姓名，

displayName**:**显示名称

phone：手机号，

mail：邮箱，

createTime：创建时间

]，…[]

}

1. 子系统提供接收用户数据接口

针对接口4：属于拉模型，子系统主动获取会存在数据延时。为了解决数据延时所带来的问题，需要有一个推模型接口。

需要子系统提供数据接收地址，数据格式为

jsonArray：:{[id:用户id，

account：账号，

name：姓名，

phone：手机号，

mail：邮箱，

createTime：创建时间，

crud:c|u|d 对应新增、修改、删除

]，…[]

}

7.子系统提供登录后的首页地址

用户中心会显示子系统列表，方便用户点击直接跳转到各个子系统

**单点登录前台对接**

1. 用户中心统一登录页面地址

请求地址：http://192.168.4.33:7878/api/security/login

请求类型：GET请求

请求参数：redirectUrl：（非必需如果不传这个参数，登录成功后跳转到用户中心默认页面）

1. 用户中心提供退出登录接口

请求地址：http://192.168.4.33:7878/api/security/logout

请求类型：POST请求

请求参数：token (必填)

返回结果：{code=1,msg=”消息内容”}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

1. 用户中心统获取用户基本信息

请求地址：http://192.168.4.33:7878/api/users/userInfo

请求类型：GET请求

请求参数：token (必填)

返回结果：{code=1,msg=”消息内容”, result}

返回结果解释：code:状态码，code==1表示成功、其他值可以参考msg查看具体错误信息，msg：消息内容。

result:{

id:用户id,

mobile:手机号,  
 avatar:头像地址相对路径,  
 nickname:昵称,  
 realName:真实姓名, cardType:证件类型（*"0">身份证、"1">港澳台、"2">护照*）  
 cardNumber:证件号码, gender:性别（*0：女，1：男，2:保密，3：其他*）

}  
(如果需要存会员信息，这个接口就够了)

错误状态码以及相关说明：

{

code: 1,

msg:”操作成功”

}

-1008,"token有误！"

-1009,"token已过期，请重新获取token"

-1012,"签名被篡改，请确认token是否泄露！"

-1014,"权限不够，禁止访问"

如何获取token？

1. 基于Spring框架开发的项目，可以用文档中附上的代码，稍做修改就能用。
2. 修改原有代码中判断登录逻辑，具体思路可阅读代码注释部分。

附上Java代码（基于Spring Interceptor实现）

**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.util.StringUtils;  
**import** org.springframework.web.method.HandlerMethod;  
**import** org.springframework.web.servlet.AsyncHandlerInterceptor;  
**import** org.springframework.web.servlet.ModelAndView;  
**import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;  
**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.BufferedReader;  
**import** java.io.DataOutputStream;  
**import** java.io.InputStreamReader;  
**import** java.net.ConnectException;  
**import** java.net.HttpURLConnection;  
**import** java.net.URL;  
**import** java.util.Enumeration;  
  
*/\*\*  
 \** ***@author*** *CREATE BY TENG FEI  
 \** ***@date*** *2020/10/25 15:58  
 \** ***@desc*** *拦截登录、并校验token  
 \* 核心思想就是一个拦截器， 拦截所有请求  
 \*/*@Configuration  
**public class** SSOConfig **implements** WebMvcConfigurer {  
 *用户中心登录地址 修改成实际地址* **private static final** String ***LOGIN\_URL*** = **"http://http://221.224.82.156:8080/szmuseum/bg/api/systemUsers/login"**;  
  *用户中心登二次校验地址 修改成实际地址* **private static final** String ***CHECK\_URL*** = **"http://http://221.224.82.156:8080/szmuseum/bg/api/systemUsers/checkToken"**;  
 *自己的系统前缀 修改成实际地址* **private static final** String ***PREFIX*** = **"http://localhost:7777/myapp/"**;  
 @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 *//拦截后台* registry.addInterceptor(**new** AsyncHandlerInterceptor() {  
 */\*\*  
 \* 基于cookie要判断3种情况，  
 \* 1：请求参数中无token，并且cookie中无token 未登录无疑 直接拦截  
 \* 2：请求参数中无token，cookie中有token 表示已经登录，此时需要做二次校验 校验通过：放行，校验失败：当作未登录处理  
 \* 3：请求参数中有token，像用户中心发起二次校验请求，如果校验通过：存cookie，如果失败：则当作未登录处理  
 \*  
 \* 基于session相对简单 只需判断2种情况  
 \* 1：请求参数中无token，并且session中无token 未登录无疑 直接拦截  
 \* 2：请求参数中有token，并且session中无token 此时需要做二次校验 校验通过：存session并放行，校验失败：当作未登录处理  
 \*  
 \** ***@param request*** *\** ***@param response*** *\** ***@param handler*** *\** ***@return*** *\** ***@throws*** *Exception  
 \*/* @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  
 HandlerMethod hm;  
 **try** {  
 hm = (HandlerMethod) handler;  
 } **catch** (Exception e) {  
 **return true**;  
 }  
 System.***out***.println(request.getRequestURI());  
 String token = request.getParameter(**"token"**);  
 String redirectUrl = getLoginUrl(request);  
 */\*\*\*\*\*\*\*\*\* 方式1 基于cookie 开始\*\*\*\*\*\*\*\*\*\*\*\*\*/  
// String cookieToken = CookieUtil.getCookie(request,"token");  
// // 情况1  
// if(StringUtils.isEmpty(token) && StringUtils.isEmpty(cookieToken)){  
// response.sendRedirect(redirectUrl);  
// return false;  
// }  
// // 情况2  
// if(StringUtils.isEmpty(token) && StringUtils.hasText(cookieToken)){  
// //至于是否二次校验token 根据自身情况决定  
// if(!checkToken(cookieToken)){  
// response.sendRedirect(redirectUrl);  
// return false;  
// }  
// }  
// // 情况3  
// if(StringUtils.hasText(token)){  
// if(!checkToken(token)){  
// response.sendRedirect(redirectUrl);  
// return false;  
// } else {  
// CookieUtil.addCookie(response,"token",token);  
// }  
// }  
 /\*\*\*\*\*\*\*\*\* 方式1 基于cookie 结束\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 /\*\*\*\*\*\*\*\*\* 方式2 基于session 开始\*\*\*\*\*\*\*\*\*\*\*\*\*/* Object sessionToken = request.getSession().getAttribute(**"token"**);  
 *// 情况1* **if**(StringUtils.*isEmpty*(token) && sessionToken == **null**){  
 response.sendRedirect(redirectUrl);  
 **return false**;  
 }  
 *// 情况2* **if**(StringUtils.*hasText*(token) && sessionToken == **null**){  
 **if**(!checkToken(token)){  
 response.sendRedirect(redirectUrl);  
 **return false**;  
 }  
 request.getSession().setAttribute(**"token"**,token);  
 }  
 *// 情况3  
// 如果前面两个情况都不满 则表示session中已经有值了（已经走过一遍情况2了）  
 /\*\*\*\*\*\*\*\*\* 方式2 基于session 结束\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
 // 3个判断全部通过，现在可以做自己的后续处理了，比方说拆token存session等操作等* ***todo 登录成功后的逻辑可以从这里开始写***

***todo为了保证统一退出 使用session需要对token做二次校验* return true**;  
 }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {  
  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {  
  
 }  
 }).addPathPatterns(**"/\*\*"**);  
 }  
  
 */\*\*  
 \* 校验token 校验通过返回true 否则返回false  
 \** ***@param token*** *\** ***@return*** *\** ***@throws*** *Exception  
 \*/* **private boolean** checkToken(String token) **throws** Exception {  
 String str = *sendPost*(***CHECK\_URL***,**"token="**+token);  
 JsonObject json = GsonUtil.*gsonObj*(str); 替换成自己的json包  
 **int** code = json.get(**"code"**).getAsInt(); 替换成自己的json包  
 **return** code == 1;  
 }  
  
  
 **private** String getLoginUrl(HttpServletRequest request) {  
 **return *LOGIN\_URL*** + **"?redirectUrl="**+***PREFIX*** + getRelativelyUrl(request);  
 }  
  
 *// 获取相对路径，去掉了192.168.4.33:7878后的路径  
 // 例如 原路径是 http://localhost:7999/szmuseum/bg/api/systemUsers/login  
 // 返回 bg/api/systemUsers/login* **private** String getRelativelyUrl(HttpServletRequest request){  
 String ctxPath = request.getContextPath();  
 String uri = request.getRequestURI();  
 **if**(StringUtils.*hasText*(ctxPath)){  
 uri = uri.replace(ctxPath+**"/"**,**""**) ;  
 }  
 String retVal = uri;  
 String params = getRequestParams(request);  
 **if**(StringUtils.*hasText*(params)){  
 retVal += **"?"**+ params;  
 }  
 **return** retVal;  
 }  
  
 *// 获取请求路径上的所有参数  
 // 例如：http://localhost:7999/szmuseum/bg/api/systemUsers/login?a=1&b=2&c=3  
 // 返回 a=1&b=2&c=3* **private** String getRequestParams(HttpServletRequest request){  
 String result = **""**;  
 Enumeration paramNames = request.getParameterNames();  
 **while** (paramNames.hasMoreElements()) {  
 String paramName = (String) paramNames.nextElement();  
 String[] paramValues = request.getParameterValues(paramName);  
 **if** (paramValues.**length** >0) {  
 String paramValue = paramValues[0];  
 **if** (paramValue.length() != 0) {  
 **if**(StringUtils.*isEmpty*(result)){  
 result +=paramName+**"="**+paramValue;  
 }**else** {  
 result += **"&"**+paramName+**"="**+paramValue;  
 }  
 }  
 }  
 }  
 **return** result;  
 }  
  
 *//替换成自己的发送post请求的工具类* **public static** String sendPost(String targetURL, String params) **throws** Exception {  
 String json = **null**;  
 **try** {  
 URL restServiceURL = **new** URL(targetURL);  
 HttpURLConnection httpConnection = (HttpURLConnection) restServiceURL.openConnection();httpConnection.setConnectTimeout(300000);  
 httpConnection.setReadTimeout(300000);  
 httpConnection.setRequestMethod(**"POST"**);  
 httpConnection.setDoOutput(**true**);  
 httpConnection.setUseCaches(**false**);  
 DataOutputStream out = **new** DataOutputStream(httpConnection.getOutputStream());  
 out.writeBytes(params);  
 out.flush();  
 out.close();  
 BufferedReader responseBuffer = **new** BufferedReader(**new** InputStreamReader((httpConnection.getInputStream())));  
 String output;  
 StringBuffer buffer = **new** StringBuffer();  
 **while** ((output = responseBuffer.readLine()) != **null**) {  
 buffer.append(output);  
 System.***out***.println(buffer);  
 }  
 json = buffer.toString();  
 } **catch** (ConnectException e) {  
 System.***out***.println(targetURL);  
 System.***out***.println(params);  
 **throw new** ConnectException(**"发送POST请求出现异常！"** + e);  
 }  
 **return** json;  
 }  
  
}