

[\(https://www.zifangsky.cn/\)](https://www.zifangsky.cn/)

zifangsky的个人博客

🏠 首页 (<https://www.zifangsky.cn/>) » Java (<https://www.zifangsky.cn/java/>) » Spring (<https://www.zifangsky.cn/java/spring/>) » 正文

OAuth2.0协议入门（三）：OAuth2.0授权与单点登录（SSO）的区别以及单点登录服务端从设计到实现

📅 2018/09/11 | 📁 Spring (<https://www.zifangsky.cn/java/spring/>) |
👤 admin (<https://www.zifangsky.cn/author/zifangskyw/>)
| 💬 8 条评论 (<https://www.zifangsky.cn/1327.html#comments>) | 👁 4297 views

一 OAuth2.0授权与单点登录（SSO）的区别

在前两篇文章中我介绍了OAuth2.0协议的基本概念 (<https://www.zifangsky.cn/1309.html> (<https://www.zifangsky.cn/1309.html>)) 以及OAuth2.0授权服务端从设计到实现

(<https://www.zifangsky.cn/1313.html> (<https://www.zifangsky.cn/1313.html>))。这篇文章中我将介绍OAuth2.0授权与单点登录的区别，这两个概念看似很相似，实际上却有很大区别，而很多人往往把二者混为一谈。

什么是单点登录？

单点登录的英文名是 **Single Sign On**，因此一般简称为**SSO**。它的用途在于，不管多么复杂的应用群，只要在用户权限范围内，那么就可以做到，用户只需要登录一次就可以访问权限范围内的所有应用子系统。对于用户而言，访问多个应用子系统只需要登录一次，同样在需要注销的时候也只需要注销一次。举个简单的例子，你在百度首页登录成功之后，你再访问百度百科、百度知道、百度贴吧等网站也会处于登录状态了，这就是一个单点登录的真实案例。

OAuth2.0授权与单点登录的区别

根据OAuth2.0授权与单点登录的概念，我们可以得知二者至少存在以下几点区别：

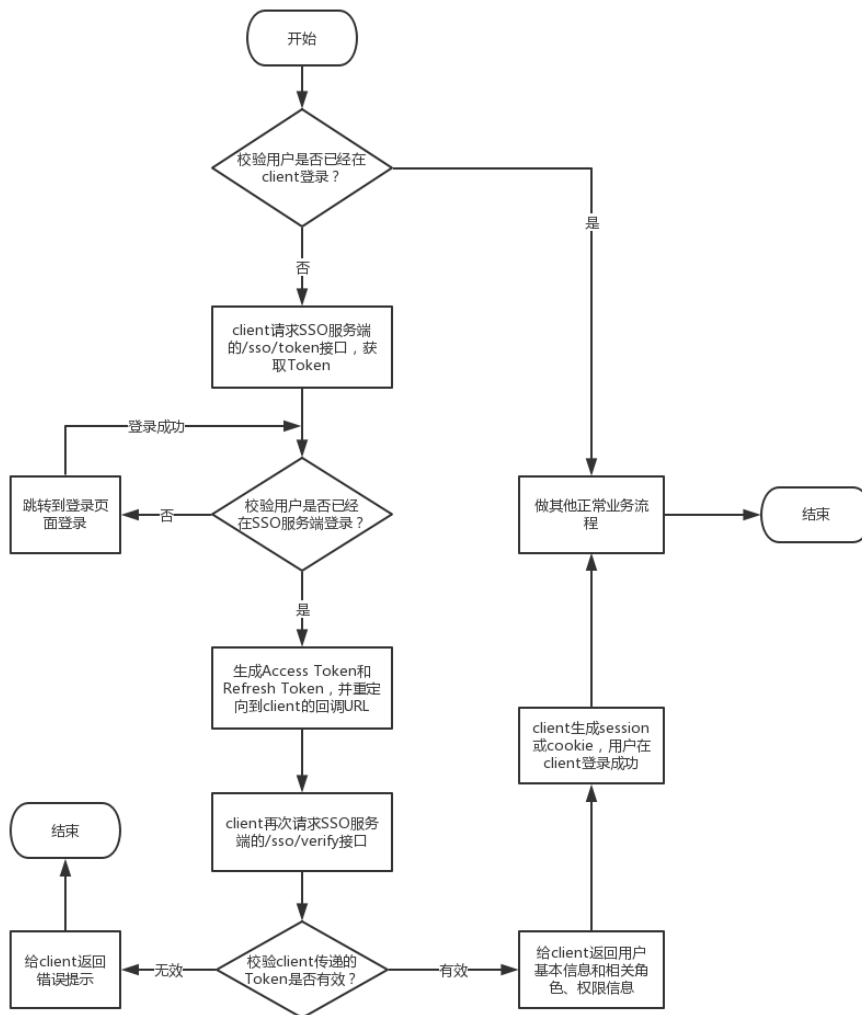
1. **从信任角度来看**。OAuth2.0授权服务端和第三方客户端不属于一个互相信任的应用群（通常都不是同一个公司提供的服务），第三方客户端的用户不属于OAuth2.0授权服务端的官方用户；而单点登录的服务端和接入的客户端都在一个互相信任的应用群（通常是同一个公司提供的服务），各个子系统的用户属于单点登录服务端的官方用户。
2. **从资源角度来看**。OAuth2.0授权主要是让用户自行决定——“我”在OAuth2.0服务提供方的个人资源是否允许第三方应用访问；而单点登录的资源都在客户端这边，单点登录的服务端主要用于登录，以及管理用户在各个子系统的权限信息。
3. **从流程角度来看**。OAuth2.0授权的时候，第三方客户端需要拿预先“商量”好的密码去获取Access Token；而单点登录则不需要。

二 单点登录服务端的设计

对于一个接入单点登录的子系统而言，进行单点登录需要以下两个步骤：

1. client请求单点登录服务端，**获取Access Token**；
2. client因为不能判断给它的Access Token是单点登录服务端返回还是用户伪造，所以需要再次请求单点登录服务端，**校验Access Token是否有效**，如果有效则返回用户基本信息以及相应的用户在client上所属的角色、权限等信息。

因此，单点登录服务端的设计主要围绕这两个接口展开，其主要流程是这样的：



<https://www.zifangsky.cn>

数据库的表结构设计

提示：我在下面只介绍一些表的主要字段，这个Demo中使用的完整的表结构可以参考：
https://gitee.com/zifangsky/OAuth2.0Demo/blob/master/rbac_db.sql
 (https://gitee.com/zifangsky/OAuth2.0Demo/blob/master/rbac_db.sql)

(1) sso_client_details:

接入单点登录的子系统详情表。类似于百度的百度百科、百度知道、百度贴吧等应用子系统，每个想要接入单点登录的子系统都需要事先在服务端这里“备案”。一方面是为了防止用户在服务端登录成功生成的Access Token被重定向到非法网站，从而导致用户的Access Token被窃取；另一方面是记录接入的子系统的注销URL，便于开发单点注销功能。所以主要需要以下几个字段：

- client_name: 子系统的名称
- redirect_url: 获取Access Token成功后的回调URL
- logout_url: 用户在子系统的注销URL（用户登录状态可以分为：**全局登录**——在单点登录服务端的登录状态；**局部登录**——在子系统的登录状态，注销的时候需要同时注销用户在单点登录服务端和应用子系统的登录状态）

(2) sso_access_token:

单点登录的Access Token信息表。这个表主要体现出哪个用户在哪个子系统登录，以及生成的令牌的结束日期是哪天。所以主要需要以下几个字段：

- access_token: **Access Token**字段
- user_id: 表明是哪个用户登录
- client_id: 表明是在哪个子系统登录
- expires_in: 过期时间戳，表明这个Token在哪一天过期

(3) sso_refresh_token:

单点登录的Refresh Token信息表。这个表主要用来记录Refresh Token，在设计表结构的时候需要关联它对应的sso_access_token表的记录。所以主要需要以下几个字段：

- refresh_token: **Refresh Token**字段
- token_id: 它对应的sso_access_token表的记录
- expires_in: 过期时间戳

三 单点登录服务端主要接口的代码实现

这个Demo的单点登录服务端的完整可用源码可以参

考: <https://gitee.com/zifangsky/OAuth2.0Demo/tree/master/ServerDemo>

(<https://gitee.com/zifangsky/OAuth2.0Demo/tree/master/ServerDemo>)

(1) 获取Access Token:

client请求单点登录服务端，**获取Access Token**，获取完成之后重定向到请求中的回调URL。

接口地址：http://127.0.0.1:7000/sso/token?redirect_uri=http://192.168.197.130:6080/login

```
1 /**
2  * 获取Token
3  * @author zifangsky
4  * @date 2018/8/30 16:30
5  * @since 1.0.0
6  * @param request HttpServletRequest
7  * @return org.springframework.web.servlet.ModelAndView
8  */
9 @RequestMapping("/token")
10 public ModelAndView authorize(HttpServletRequest request){
11     HttpSession session = request.getSession();
12     User user = (User) session.getAttribute(Constants.SESSION_USER);
13
14     //过期时间
15     Long expiresIn = DateUtils.dayToSecond(ExpireEnum.ACCESS_TOKEN.getTime());
16     //回调URL
17     String redirectUri = request.getParameter("redirect_uri");
18     //查询接入客户端
19     SsoClientDetails ssoClientDetails = ssoService.selectByRedirectUrl(redirectUri);
20
21     //获取用户IP
22     String requestIp = SpringContextUtils.getRequestIp(request);
23
24     //生成Access Token
25     String accessTokenStr = ssoService.createAccessToken(user, expiresIn, requestIp, ssoClientDetails);
26     //查询已经插入到数据库的Access Token
27     SsoAccessToken ssoAccessToken = ssoService.selectByAccessToken(accessTokenStr);
28     //生成Refresh Token
29     String refreshTokenStr = ssoService.createRefreshToken(user, ssoAccessToken);
30     logger.info(MessageFormat.format("单点登录获取Token: username: [{0}], channel: [{1}], Access Token: [{2}], Refresh Token: [{3}]",
31         user.getUsername(), ssoClientDetails.getClientName(), accessTokenStr, refreshTokenStr));
32
33     String params = "?code=" + accessTokenStr;
34     return new ModelAndView("redirect:" + redirectUri + params);
35 }
```

相应地，调用的cn/zifangsky/service/impl/SsoServiceImpl.java类里面的生成逻辑：

```
1 @Override
2 public String createAccessToken(User user, Long expiresIn, String requestIP, SsoClientDetails
3     Date current = new Date();
4     //过期的时间戳
5     Long expiresAt = DateUtils.nextDaysSecond(ExpireEnum.ACCESS_TOKEN.getTime(), null);
6
7     //1. 拼装待加密字符串 (username + 渠道CODE + 当前精确到毫秒的时间戳)
8     String str = user.getUsername() + ssoClientDetails.getClientName() + String.valueOf(DateUtil
9
10    //2. SHA1加密
11    String accessTokenStr = "11." + EncryptUtils.sha1Hex(str) + "." + expiresIn + "." + expire
12
13    //3. 保存Access Token
14    SsoAccessToken savedAccessToken = ssoAccessTokenMapper.selectByUserIdAndClientId(user.getId()
15    //如果存在匹配的记录, 则更新原记录, 否则向数据库中插入新记录
16    if(savedAccessToken != null){
17        savedAccessToken.setAccessToken(accessTokenStr);
18        savedAccessToken.setExpiresIn(expiresAt);
19        savedAccessToken.setUpdateUser(user.getId());
20        savedAccessToken.setUpdateTime(current);
21        ssoAccessTokenMapper.updateByPrimaryKeySelective(savedAccessToken);
22    }else{
23        savedAccessToken = new SsoAccessToken();
24        savedAccessToken.setAccessToken(accessTokenStr);
25        savedAccessToken.setUserId(user.getId());
26        savedAccessToken.setUserName(user.getUsername());
27        savedAccessToken.setIp(requestIP);
28        savedAccessToken.setClientId(ssoClientDetails.getId());
29        savedAccessToken.setChannel(ssoClientDetails.getClientName());
30        savedAccessToken.setExpiresIn(expiresAt);
31        savedAccessToken.setCreateUser(user.getId());
32        savedAccessToken.setUpdateUser(user.getId());
33        savedAccessToken.setCreateTime(current);
34        savedAccessToken.setUpdateTime(current);
35        ssoAccessTokenMapper.insertSelective(savedAccessToken);
36    }
37
38    //4. 返回Access Token
39    return accessTokenStr;
40 }
41
42 @Override
43 public String createRefreshToken(User user, SsoAccessToken ssoAccessToken) {
44     Date current = new Date();
45     //过期时间
46     Long expiresIn = DateUtils.dayToSecond(ExpireEnum.REFRESH_TOKEN.getTime());
47     //过期的时间戳
48     Long expiresAt = DateUtils.nextDaysSecond(ExpireEnum.REFRESH_TOKEN.getTime(), null);
49
50    //1. 拼装待加密字符串 (username + access token + 当前精确到毫秒的时间戳)
51    String str = user.getUsername() + ssoAccessToken.getAccessToken() + String.valueOf(DateUtil
52
53    //2. SHA1加密
54    String refreshTokenStr = "12." + EncryptUtils.sha1Hex(str) + "." + expiresIn + "." + expir
55
56    //3. 保存Refresh Token
57    SsoRefreshToken savedRefreshToken = ssoRefreshTokenMapper.selectByTokenId(ssoAccessToken.g
58    //如果存在tokenId匹配的记录, 则更新原记录, 否则向数据库中插入新记录
59    if(savedRefreshToken != null){
60        savedRefreshToken.setRefreshToken(refreshTokenStr);
61        savedRefreshToken.setExpiresIn(expiresAt);
62        savedRefreshToken.setUpdateUser(user.getId());
63        savedRefreshToken.setUpdateTime(current);
64        ssoRefreshTokenMapper.updateByPrimaryKeySelective(savedRefreshToken);
65    }else{
66        savedRefreshToken = new SsoRefreshToken();
67        savedRefreshToken.setTokenId(ssoAccessToken.getId());
68        savedRefreshToken.setRefreshToken(refreshTokenStr);
69        savedRefreshToken.setExpiresIn(expiresAt);
70        savedRefreshToken.setCreateUser(user.getId());
71        savedRefreshToken.setUpdateUser(user.getId());
72        savedRefreshToken.setCreateTime(current);
73        savedRefreshToken.setUpdateTime(current);
74        ssoRefreshTokenMapper.insertSelective(savedRefreshToken);
75    }
76 }
```

```
77 //4. 返回Refresh Tokens
78 return refreshTokenStr;
79 }
```

(2) 校验Access Token，并返回用户信息：

client在获取到Access Token后，再次调用单点登录服务端接口，用于“校验Access Token，并返回用户信息”。

接口地址：http://127.0.0.1:7000/sso/verify?

access_token=11.ad51132688b5be3f476592356c78aef71d235f07.2592000.1539143183

返回如下：

```
1 {
2   "access_token": "11.ad51132688b5be3f476592356c78aef71d235f07.2592000.1539143183",
3   "refresh_token": "12.c10cb9001bf0e2c7f808580318715fc089673279.31536000.1568087183",
4   "expires_in": 2592000,
5   "user_info": {
6     "id": 2,
7     "username": "zifangsky",
8     "password": "$5$t0BSeX2$hSnSDyhJmVVRpbmKuIY4SxDgyeGRGacQaBYGrTEBnZA",
9     "mobile": "110",
10    "email": "admin@zifangsky.cn",
11    "createTime": "2017-12-31T16:00:00.000+0000",
12    "updateTime": "2017-12-31T16:00:00.000+0000",
13    "status": 1,
14    "roles": [{
15      "id": 2,
16      "roleName": "user",
17      "description": "普通用户",
18      "funcs": null
19    }]
20  }
21 }
22 }
```

首先在一个拦截器里校验Access Token是否有效：

```
1 package cn.zifangsky.interceptor;
2
3 import cn.zifangsky.enums.ErrorCodeEnum;
4 import cn.zifangsky.model.SsoAccessToken;
5 import cn.zifangsky.service.SsoService;
6 import cn.zifangsky.utils.DateUtils;
7 import cn.zifangsky.utils.JsonUtils;
8 import org.apache.commons.lang3.StringUtils;
9 import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
10
11 import javax.annotation.Resource;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14 import java.time.LocalDateTime;
15 import java.util.HashMap;
16 import java.util.Map;
17
18 /**
19  * 用于校验Access Token是否为空以及Access Token是否已经失效
20  *
21  * @author zifangsky
22  * @date 2018/8/30
23  * @since 1.0.0
24  */
25 public class SsoAccessTokenInterceptor extends HandlerInterceptorAdapter {
26     @Resource(name = "ssoServiceImpl")
27     private SsoService ssoService;
28
29     /**
30      * 检查Access Token是否已经失效
31      */
32     @Override
33     public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
34         String accessToken = request.getParameter("access_token");
35
36     if(StringUtils.isNotBlank(accessToken)){
```

```

37 //查询数据库中的Access Token
38 SsoAccessToken ssoAccessToken = ssoService.selectByAccessToken(accessToken);
39
40 if(ssoAccessToken != null){
41     Long savedExpiresAt = ssoAccessToken.getExpiresIn();
42     //过期日期
43     LocalDateTime expiresDateTime = DateUtils.ofEpochSecond(savedExpiresAt, null);
44     //当前日期
45     LocalDateTime nowDateTime = DateUtils.now();
46
47     //如果Access Token已经失效, 则返回错误提示
48     return expiresDateTime.isAfter(nowDateTime) || this.generateErrorResponse(response);
49 }else{
50     return this.generateErrorResponse(response, ErrorCodeEnum.INVALID_GRANT);
51 }
52 }else{
53     return this.generateErrorResponse(response, ErrorCodeEnum.INVALID_REQUEST);
54 }
55 }
56
57 /**
58  * 组装错误请求的返回
59  */
60 private boolean generateErrorResponse(HttpServletRequest response, ErrorCodeEnum errorCode) {
61     response.setCharacterEncoding("UTF-8");
62     response.setHeader("Content-type", "application/json;charset=UTF-8");
63     Map<String, String> result = new HashMap<>(2);
64     result.put("error", errorCode.getError());
65     result.put("error_description", errorCode.getErrorDescription());
66
67     response.getWriter().write(JsonUtils.toJson(result));
68     return false;
69 }
70
71 }
72

```

然后返回用户信息:

```

1 /**
2  * 校验Access Token, 并返回用户信息
3  * @author zifangsky
4  * @date 2018/8/30 16:07
5  * @since 1.0.0
6  * @param request HttpServletRequest
7  * @return java.util.Map<java.lang.String, java.lang.Object>
8  */
9 @RequestMapping(value = "/verify", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
10 @ResponseBody
11 public Map<String, Object> verify(HttpServletRequest request) {
12     Map<String, Object> result = new HashMap<>(8);
13
14     //获取Access Token
15     String accessToken = request.getParameter("access_token");
16
17     try {
18         //过期时间
19         Long expiresIn = DateUtils.dayToSecond(ExpireEnum.ACCESS_TOKEN.getTime());
20         //查询Access Token
21         SsoAccessToken ssoAccessToken = ssoService.selectByAccessToken(accessToken);
22         //查询Refresh Token
23         SsoRefreshToken ssoRefreshToken = ssoService.selectByTokenId(ssoAccessToken.getId());
24         //查询用户信息
25         UserBo userBo = userService.selectUserBoByUserId(ssoAccessToken.getUserId());
26
27         //组装返回信息
28         result.put("access_token", ssoAccessToken.getAccessToken());
29         result.put("refresh_token", ssoRefreshToken.getRefreshToken());
30         result.put("expires_in", expiresIn);
31         result.put("user_info", userBo);
32         return result;
33     } catch (Exception e) {
34         logger.error(e.getMessage());
35         this.generateErrorResponse(result, ErrorCodeEnum.UNKNOWN_ERROR);
36         return result;
37     }
38 }

```

```
37 }
38 }
```

(3) 通过Refresh Token刷新Access Token接口：

如果客户端的Access Token过期了，那么就可以通过这个接口刷新Access Token。

接口地址：http://127.0.0.1:7000/sso/refreshToken?

refresh_token=12.c10cb9001bf0e2c7f808580318715fc089673279.31536000.1568087183

返回如下：

```
1 {
2   "access_token": "11.40f0270697c37db4570e41e0f6f335bf6c2f8902.2592000.1539164947",
3   "refresh_token": "12.c10cb9001bf0e2c7f808580318715fc089673279.31536000.1568087183",
4   "expires_in": 2592000,
5   "user_info": {
6     "id": 2,
7     "username": "zifangsky",
8     "password": "$5$t008SeX2$hSnSDyhJmVVRpbmKuIY4SxDgyeGRGacQaBYGrTEBnZA",
9     "mobile": "110",
10    "email": "admin@zifangsky.cn",
11    "createTime": "2017-12-31T16:00:00.000+0000",
12    "updateTime": "2017-12-31T16:00:00.000+0000",
13    "status": 1,
14    "roles": [{
15      "id": 2,
16      "roleName": "user",
17      "description": "普通用户",
18      "funcs": null
19    }]
20  }
21 }
22 }
```

```
1 /**
2  * 通过Refresh Token刷新Access Token
3  * @author zifangsky
4  * @date 2018/8/30 16:07
5  * @since 1.0.0
6  * @param request HttpServletRequest
7  * @return java.util.Map<java.lang.String,java.lang.Object>
8  */
9 @RequestMapping(value = "/refreshToken", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
10 @ResponseBody
11 public Map<String, Object> refreshToken(HttpServletRequest request) {
12     Map<String, Object> result = new HashMap<>(8);
13
14     //获取Refresh Token
15     String refreshTokenStr = request.getParameter("refresh_token");
16     //获取用户IP
17     String requestIp = SpringContextUtils.getRequestIp(request);
18
19     try {
20         SsoRefreshToken ssoRefreshToken = ssoService.selectByRefreshToken(refreshTokenStr);
21
22         if(ssoRefreshToken != null) {
23             Long savedExpiresAt = ssoRefreshToken.getExpiresIn();
24             //过期日期
25             LocalDateTime expiresDateTime = DateUtils.ofEpochSecond(savedExpiresAt, null);
26             //当前日期
27             LocalDateTime nowDateTime = DateUtils.now();
28
29             //如果Refresh Token已经失效，则需要重新生成
30             if (expiresDateTime.isBefore(nowDateTime)) {
31                 this.generateErrorResponse(result, Enum.valueOf(ErrorCodeEnum.class, EXPIRED_TOKEN));
32                 return result;
33             } else {
34                 //获取存储的Access Token
35                 SsoAccessToken ssoAccessToken = ssoService.selectByAccessId(ssoRefreshToken.getAccessId());
36                 //查询接入客户端
37                 SsoClientDetails ssoClientDetails = ssoService.selectByPrimaryKey(ssoAccessToken.getClientId());
38                 //获取对应的用户信息
39                 User user = userService.selectById(ssoAccessToken.getUserId());
```

```
40
41 //新的过期时间
42 Long expiresIn = DateUtils.dayToSecond(ExpireEnum.ACCESS_TOKEN.getTime());
43 //生成新的Access Token
44 String newAccessTokenStr = ssoService.createAccessToken(user, expiresIn, requestIp);
45 //查询用户信息
46 UserBo userBo = userService.selectUserBoByUserId(ssoAccessToken.getUserId());
47
48 logger.info(MessageFormat.format("单点登录重新刷新Token: username: [{0}], requestIp: [{1}], user: {2}, requestIp: {3}, ssoAccessToken: {4}, newAccessToken: {5}",
49     user.getUsername(), requestIp, ssoAccessToken.getAccessToken(), newAccessTokenStr));
50
51 //组装返回信息
52 result.put("access_token", newAccessTokenStr);
53 result.put("refresh_token", ssoRefreshToken.getRefreshToken());
54 result.put("expires_in", expiresIn);
55 result.put("user_info", userBo);
56 return result;
57 }
58 }else {
59     this.generateErrorResponse(result, ErrorCodeEnum.INVALID_GRANT);
60     return result;
61 }
62 }catch (Exception e){
63     e.printStackTrace();
64     this.generateErrorResponse(result, ErrorCodeEnum.UNKNOWN_ERROR);
65     return result;
66 }
67 }
68
69 /**
70  * 组装错误请求的返回
71  */
72 private void generateErrorResponse(Map<String, Object> result, ErrorCodeEnum errorCodeEnum) {
73     result.put("error", errorCodeEnum.getError());
74     result.put("error_description", errorCodeEnum.getErrorDescription());
75 }
```

(4) 单点注销接口：

在这个Demo项目中，我没有提供单点注销功能的示例代码，但是我可以简单说一下单点注销功能的主要流程，如果需要这个功能可以自行使用代码实现：

1. 用户在应用子系统请求注销登录；
2. 用户在应用子系统注销完成后，应用子系统后台请求单点登录服务端的注销接口；
3. 单点登录服务端的注销接口根据用户的Token在数据库中查询当前用户登录的所有应用子系统的注销接口，然后依次调用注销即可。

四 接入单点登录的子系统的关键代码

这个Demo的单点登录子系统的完整可用源码可以参

考：<https://gitee.com/zifangsky/OAuth2.0Demo/tree/master/SsoClientDemo>

(<https://gitee.com/zifangsky/OAuth2.0Demo/tree/master/SsoClientDemo>)

其实，对于接入单点登录的子系统来说，登录模块调用单点登录服务端提供的接口就可以了。

登录校验过滤器：

```
1 package cn.zifangsky.interceptor;
2
3 import cn.zifangsky.common.Constants;
4 import cn.zifangsky.common.SpringContextUtils;
5 import cn.zifangsky.model.bo.UserBo;
6 import org.apache.commons.lang3.StringUtils;
7 import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
8
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11 import javax.servlet.http.HttpSession;
12
```



```
13 /**
14  * 定义一些页面需要做登录检查
15  *
16  * @author zifangsky
17  * @date 2018/7/26
18  * @since 1.0.0
19  */
20 public class LoginInterceptor extends HandlerInterceptorAdapter{
21
22     /**
23      * 检查是否已经登录
24      */
25     @Override
26     public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
27         HttpSession session = request.getSession();
28
29     //获取session中存储的用户信息
30     UserBo user = (UserBo) session.getAttribute(Constants.SESSION_USER);
31
32     if(user != null){
33         return true;
34     }else{
35         //如果token不存在，则跳转等登录页面
36         response.sendRedirect(request.getContextPath() + "/login?redirectUrl=" + SpringCon
37
38         return false;
39     }
40 }
41 }
42 }
```

登录相关的代码逻辑:

```
1 package cn.zifangsky.controller;
2
3 import cn.zifangsky.common.Constants;
4 import cn.zifangsky.model.SsoResponse;
5 import cn.zifangsky.model.User;
6 import cn.zifangsky.utils.CookieUtils;
7 import org.apache.commons.lang3.StringUtils;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.beans.factory.annotation.Value;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.client.RestTemplate;
13 import org.springframework.web.servlet.ModelAndView;
14
15 import javax.servlet.http.HttpServletRequest;
16 import javax.servlet.http.HttpServletResponse;
17 import javax.servlet.http.HttpSession;
18
19 /**
20  * 登录
21  * @author zifangsky
22  * @date 2018/7/9
23  * @since 1.0.0
24  */
25 @Controller
26 public class LoginController {
27
28     @Autowired
29     private RestTemplate restTemplate;
30
31     @Value("${own.sso.access-token-uri}")
32     private String accessTokenUri;
33
34     @Value("${own.sso.verify-uri}")
35     private String verifyUri;
36
37     /**
38      * 登录验证（实际登录调用认证服务器）
39      * @author zifangsky
40      * @date 2018/8/30 18:02
41      * @since 1.0.0
42      * @param request HttpServletRequest
```

```
43     * @return org.springframework.web.servlet.ModelAndView
44     */
45     @RequestMapping("/login")
46     public ModelAndView login(HttpServletRequest request, HttpServletResponse response){
47         //当前系统登录成功之后的回调URL
48         String redirectUrl = request.getParameter("redirectUrl");
49         //当前系统请求认证服务器成功之后返回的Token
50         String code = request.getParameter("code");
51
52         //最后重定向的URL
53         String resultUrl = "redirect:";
54         HttpSession session = request.getSession();
55
56         //1. code为空, 则说明当前请求不是认证服务器的回调请求, 则重定向URL到认证服务器登录
57         if(StringUtils.isBlank(code)){
58             //如果存在回调URL, 则将这个URL添加到session
59             if(StringUtils.isNoneBlank(redirectUrl)){
60                 session.setAttribute(Constants.SESSION_LOGIN_REDIRECT_URL, redirectUrl);
61             }
62
63             //拼装请求Token的地址
64             resultUrl += accessTokenUri;
65         }else{
66             //2. 验证Token, 并返回用户基本信息、所属角色、访问权限等
67             SsoResponse verifyResponse = restTemplate.getForObject(verifyUri, SsoResponse.class, code);
68
69             //如果正常返回
70             if(StringUtils.isNoneBlank(verifyResponse.getAccess_token())){
71                 //2.1 将用户信息存到session
72                 session.setAttribute(Constants.SESSION_USER, verifyResponse.getUser_info());
73
74                 //2.2 将Access Token和Refresh Token写到cookie
75                 CookieUtils.addCookie(response, Constants.COOKIE_ACCESS_TOKEN, verifyResponse.getAccess_token());
76                 CookieUtils.addCookie(response, Constants.COOKIE_REFRESH_TOKEN, verifyResponse.getRefresh_token());
77             }
78
79             //3. 从session中获取回调URL, 并返回
80             redirectUrl = (String) session.getAttribute(Constants.SESSION_LOGIN_REDIRECT_URL);
81             session.removeAttribute("redirectUrl");
82             if(StringUtils.isNoneBlank(redirectUrl)){
83                 resultUrl += redirectUrl;
84             }else{
85                 resultUrl += "/user/userIndex";
86             }
87         }
88     }
89
90     return new ModelAndView(resultUrl);
91 }
92
93 }
```

当然，上面代码中使用到的一些配置就是我们单点登录服务端的接口地址：

```
1 own.sso.access-token-uri=http://10.0.5.22:7000/sso/token?redirect_uri=http://192.168.197.130:6080
2 own.sso.verify-uri=http://10.0.5.22:7000/sso/verify?access_token={1}
```

测试：

1. 将SsoClientDemo项目部署在跟ServerDemo项目不同的服务器；
2. 第一次启动SsoClientDemo项目并访问需要登录的页面，比如：
`http://192.168.197.130:6080/user/userIndex`；
3. 可以发现这时跳转到ServerDemo项目，在服务端登录成功之后，跳转到SsoClientDemo项目的/user/userIndex，说明客户端也登录成功了；
4. 重启SsoClientDemo项目，并再次访问`http://192.168.197.130:6080/user/userIndex`，可以发现这次是直接登录了（当然也可以把SsoClientDemo项目部署到多个服务器上面，先后登录查看效果），这说明单点登录功能已经实现。

本篇文章到此结束，感谢大家的阅读。

参考：

亲，帮我喂喂鱼呗(^_^)

Missing Plug-in

🔥 热门文章

🔄 最新文章

🔀 随机文章

读《苏菲的世界》有感
(<https://www.zifangsky.cn/1419.html>) 4

《时间回旋》读后杂想
(<https://www.zifangsky.cn/1425.html>) 0

标签

Ant (<https://www.zifangsky.cn/tag/ant>) CXF
(<https://www.zifangsky.cn/tag/cxf>) Hibernate
(<https://www.zifangsky.cn/tag/hibernate>) interceptor

(<https://www.zifangsky.cn/tag/interceptor>) Java

(<https://www.zifangsky.cn/tag/tomcat>)
Java基础

(<https://www.zifangsky.cn/tag/tomcat>)

JDBC (<https://www.zifangsky.cn/tag/jdbc>) JQuery

(<https://www.zifangsky.cn/tag/jquery>) JSON

(<https://www.zifangsky.cn/tag/json>) JSP

(<https://www.zifangsky.cn/tag/jsp>) Keepalived

(<https://www.zifangsky.cn/tag/keepalived>) Linux

(<https://www.zifangsky.cn/tag/lir>)

memcached (<https://www.zifangsky.cn/tag/memcached>)

Mybatis (<https://www.zifangsky.cn/tag/mybatis>)

mysql (<https://www.zifangsky.cn/tag/mysql>) Nginx

(<https://www.zifangsky.cn/tag/nginx>)

Quartz (<https://www.zifangsky.cn/tag/quartz>) Redis

(<https://www.zifangsky.cn/tag/redis>) RESTful

(<https://www.zifangsky.cn/tag/restful>)

Spring

- OAuth认证实现机制及单点登录原理 (https://blog.csdn.net/u013444046/article/details/51508897)

👍 赞 (10)

#OAuth (https://www.zifangsky.cn/tag/oauth) #OAuth2.0 (https://www.zifangsky.cn/tag/oauth2-0)
#单点登录 (https://www.zifangsky.cn/tag/%e5%8d%95%e7%82%b9%e7%99%bb%e5%bd%95)
©版权声明：原创作品，允许转载，转载时请务必以超链接形式标明文章 原始出处 (https://www.zifangsky.cn/1327.html)、作者信息和本声明。否则将追究法律责任。
转载请注明来源： OAuth2.0协议入门（三）： OAuth2.0授权与单点登录（SSO）的区别以及单点登录服务端从设计到实现 (https://www.zifangsky.cn/1327.html) - zifangsky的个人博客 (https://www.zifangsky.cn)

上一篇 (https://www.zifangsky.cn/1326.html)

下一篇 (https://www.zifangsky.cn/1343.html)

你可能也喜欢：

- OAuth2.0协议入门（二）： OAuth2.0授权服务端从设计到实现 (https://www.zifangsky.cn/1313.html)
- OAuth2.0协议入门（一）： OAuth2.0协议的基本概念以及使用授权码模式（authorization code）实现百度账号登录 (https://www.zifangsky.cn/1309.html)
- 如何在普通Spring项目中手动实现类似Spring Boot中有条件生成Bean？ (https://www.zifangsky.cn/1416.html)
- 如何在Spring中动态设置controller中方法的请求路径？ (https://www.zifangsky.cn/1415.html)
- 基于Spring的项目中Redis存储对象使用Jackson序列化方式 (https://www.zifangsky.cn/1366.html)

本文共 8 个回复



陈小水 2019/04/04 11:38

谢谢，是篇好文章

(https://www.zifangsky.cn/1327.html?replytocom=4196#respond)

回复



admin (http://www.zifangsky.cn) 博主 2019/04/06 15:24

@ 陈小水 不客气，觉得小站还行，可以时常来看看。

(https://www.zifangsky.cn/1327.html?replytocom=4231#respond)

回复



qjia 2019/03/30 08:50

您好，我想问一下，我重启了SsoClientDemo项目，并再次访问 http://192.168.197.130:6080/user/userIndex，还是要重新输入密码呀？

(https://www.zifangsky.cn/1327.html?replytocom=4082#respond)

回复



admin (http://www.zifangsky.cn) 博主 2019/03/30 12:56

@ qjia 如果服务端和客户端都是同样的域名（比如：localhost、127.0.0.1）会出现session冲突的问题，所以我才建议用两个不同的IP分别部署啊。

(https://www.zifangsky.cn/1327.html?replytocom=4086#respond)

回复



qjia 2019/03/30 13:32

@ admin 好的，我弄的确实是同一个域名，谢谢啊

(https://www.zifangsky.cn/1327.html?replytocom=4088#respond)

回复

(https://www.zifangsky.cn/t: Spring Boot (https://www.zifangsky.cn/tag/spring boot) Spring Data Redis (https://www.zifangsky.cn/tag/spring-data-redis)

SpringMVC (https://www.zifangsky.cn/tag/s: SSH框架 (https://www.zifangsky.cn/tag/ssh%e6%a1%86%e6%9e%b6) Struts2 (https://www.zifangsky.cn/tag/struts2) WebMagic (https://www.zifangsky.cn/tag/webmagic) webservice (https://www.zifangsky.cn/tag/webservice)

WordPress (https://www.zifangsky.cn/tag/wordpress zabbix (https://www.zifangsky.cn/tag/zabbi Zookeeper (https://www.zifangsky.cn/tag/zookeeper) 不负瞎瞎说 (https://www.zifangsky.cn/tag/%e4%b8%8d%e

互联网 (https://www.zifangsky.cn/tag/%e4%ba%92%e 人工智能 (https://www.zifangsky.cn/tag/%e4%ba%ba%e5%b7%a5%e6%99%


图书分享 (https://www.zifangsky.cn/ 多线程 (https://www.zifangsky.cn/tag/%e5%a4%9a%e7%

安全 (https://www.zifangsky.cn/tag/%e5%ae%89%e 数据结构 (https://www.zifangsky.cn/tag/%e6%95%b0‘ 正则表达式 (https://www.zifangsky.cn/tag/%e6%ad%a3‘ 爬虫 (https://www.zifangsky.cn/tag/%e7%88%ac%e8%99%ab)

电影 (https://www.zifangsky.cn/tag/%e7%94%b5%e5% 算法 (https://www.zifangsky.cn/tag/%e7%a

读书总结 (https://www.zifangsky.cn/tag/%e8%af%bb%e4%b9%a6%e6‘ 资源分享 (https://www.zifangsky.cn/tag/%e8%b5%84%e6%ba‘ 软件 (https://www.zifangsky.cn/tag/%e8%bd%af%e4%bb%b6) 集群 (https://www.zifangsky.cn/tag/%e9%9b%86‘

最新评论



不客气，觉得小站还行，可以时常来...

**Robert** 2019/03/11 16:56

思路比较清楚，赞！👍看了下Demo代码，只实现了跳转到Server去登录的功能（登录转移），并没有实现SSO单点登录（只登录一次就可以访问多个客户端，<https://baike.baidu.com/item/SSO/3451380>）

(<https://www.zifangsky.cn/1327.html?replytocom=3888#respond>)

[回复](#)**admin** (<http://www.zifangsky.cn>) [博主](#) 2019/03/12 09:26

@ Robert 实现了的呦，看来你还是没看懂我的代码精髓啊，而且我猜你还没有亲手尝试过我这篇文章最后的测试步骤。

(<https://www.zifangsky.cn/1327.html?replytocom=3890#respond>)

[回复](#)**Robert** 2019/03/11 16:48

mark

(<https://www.zifangsky.cn/1327.html?replytocom=3887#respond>)

[回复](#)

发表评论

来都来了，何不留个足迹~



昵称

*



邮箱

*



网址



验证码 *

[发表评论](#)

(<https://www.zifangsky.cn/1327.html#comments>)



谢谢，是篇好文章

(<https://www.zifangsky.cn/1327.html#comments>)



好的，我弄的确实是同一个域名，谢...

(<https://www.zifangsky.cn/1327.html#comments>)



如果服务端和客户端都是同样的域名...

(<https://www.zifangsky.cn/1327.html#comments>)



感觉会代码的非常牛逼

(<https://www.zifangsky.cn/1416.html#comments>)

友情链接

iceH's Blog (<http://www.secice.cn>) 业余草 (<http://www.xttblog.com/>) 六阿哥博客 (<https://blog.6ag.cn>) 我的简书地址 (<https://www.jianshu.com/u/ccdf85cca694>)
掘金专栏 (<https://juejin.im/user/5819f202d203090055df470e>) 朴实的追梦者 (<http://www.zmzblog.com>) 青木（简书） (<https://www.jianshu.com/u/cedd62e70951>)