单点登陆集成

张雄彪

Version 1.0, 2016-07-26

Table of Contents

dmcas-client 作为 *sso* 校验中的客户端，用来拦截各子系统发送的请求，在请求抵达子系统前完成用户信息校验工作。

正因为如此， *dmcas-client* 必须部署在各子系统中，通常也是以一个 *jar* 包的方式存在。

Chapter 1. 部署使用说明

1.1. 获取程序包

第三方需要联系 **湖北公安云** 项目组获取 *dmcas-client* 的程序 *jar* 包。

1.2. 编写配置文件

如新建 *cas-client.properties* 文件。编写如下内容：

cas-client.properties

```
casURL=http://localhost:8080/dmcas ①
loginURL=http://localhost:8080/portal/login.do ②
exceptPathPrefix=/ui/;/services/ ③
publicPathPrefix=/common/;/portal/;/wmf/casAuth ④
```

① 指定 *dmcas* 验证中心的访问地址，请填写实际地址。

② 指定用户未登录时访问应用需要跳转到的地址，即配置登陆地址。

③ 希望加入例外的 *url* 匹配规则。 **注意，不会获取用户信息，检测为例外请求后，会直接放行。**

④ 配置为公共请求 *url* 的匹配规则。 **注意，会尝试获取用信息，不论是否获取得到，只要是公共请求，都会放行。**

1.3. 配置过滤器

在子系统的 *web.xml* 文件中添加如下过滤器，请注意 *filter* 的顺序。具体该 *filter* 放在哪个位置视各系统而定。

web.xml

```
<!-- 单点登录客户端 -->
<filter>
  <filter-name>CASAuthFilter</filter-name>
  <filter-class>com.dm.cas.client.filter.AuthFilter</filter-class>
  <init-param>
    <param-name>configMethod</param-name>
    <param-value>file</param-value> ①
  </init-param>
  <init-param>
    <param-name>configFilePath</param-name>
    <param-value>/WEB-INF/config/cas-client.properties</param-value> ②
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CASAuthFilter</filter-name>
  <url-pattern>*.do</url-pattern> ③
</filter-mapping>
<filter-mapping>
  <filter-name>CASAuthFilter</filter-name>
  <url-pattern>*.html</url-pattern> ④
</filter-mapping>
```

① 指定为 *file* 表示从配置文件中读取配置信息。

② 指定配置文件的路径，此处应该配 *web* 工程的相对路径。

③ 指定请求拦截的 *url* 规则，例如此处为 *.do

④ 指定请求拦截的 *url* 规则，例如此处为 *.html

1.4. 工具类

获取登陆用户信息

```
User user = CASUtil.getUserInfo(CASAuthID); ①
```

① *CASAuthID* 为 *Cookie* 中的票据信息

Chapter 2. 开发认证节点

在子系统集成至公安云单点系统时，如果需要自己实现登陆过程中的校验逻辑。则需要实现以下接口。

2.1. 接口

在 *dmcas* 独立出来以后，暴露了如下一些接口，子系统实现这些接口后， *dmcas* 将通过 *rpc* 的方式调用子系统的具体实现来完成校验及其它业务逻辑。

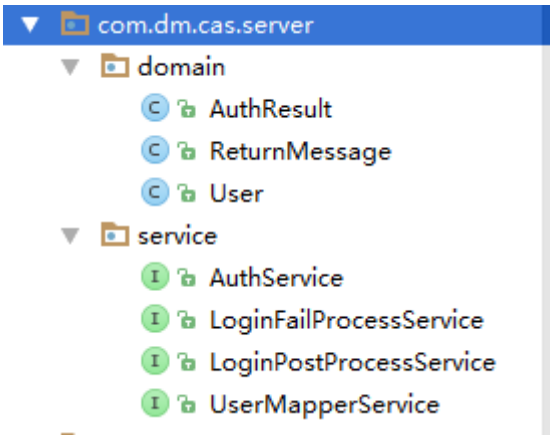


Figure 1. interfaces

AuthService

校验服务接口，根据用户登录请求的用户名密码，子系统提供校验服务即可。注意，在子系统自己的登陆页面登陆时才会调用该接口。 必须实现

AuthService

```
/**
 * 检验用户信息服务
 *
 * @author zxb
 * @version 1.0.1
 */
public interface AuthService {

    /**
     * 校验用户名密码是否正确
     *
     * @param username 用户名
     * @param password 用户明文密码
     * @return AuthResult 校验结果
     * @see com.dm.cas.server.domain.AuthResult
     */
    public AuthResult validate(String username, String password);

}
```

LoginPostProcessService

登录成功后的处理接口，子系统实现该接口以便完成一些登陆成功后的处理逻辑。例如，记录登陆日志等。 可选实现

LoginPostProcessService

```
/**
 * 用户登录成功后，子系统的处理服务。
 * <p/>
 * 子系统可以实现该接口，以便在用户登录成功后，计入登录日志等信息。
 * @author zxb
 * @version 1.0.1
 * Created by zxb on 2016/3/13.
 */
public interface LoginPostProcessService {

    /**
     *
cas验证用户登录成功后，会调用该接口。子类实现该接口后可加入自己的一些处理逻辑。
     * @param params 参数列表
     *               <ul>
     *                  <li>user 登录用户信息</li>
     *                  <li>ip 登录用户客户端ip地址</li>
     *               </ul>
     * @throws Exception
     * @return ReturnMessage 返回的消息，子类如果不返回消息及状态码。
cas将使用默认消息及状态码。
     */
    ReturnMessage process(Map<String, Object> params) throws Exception;

}
```

LoginFailProcessService

登陆失败后的处理逻辑，子系统实现该接口后可在用户登陆失败后加入自己的一些处理逻辑。 可选实现

LoginFailProcessService

```
/**
 * @author zxb
 * @version 1.0.1
 * Created by zxb on 2016/3/14.
 */
public interface LoginFailProcessService {

    /**
     *
cas验证用户登录失败后，会调用该接口。子类实现该接口后可加入自己的一些处理逻辑。
     * @param params 参数列表
     *               <ul>
     *                  <li>user 登录用户信息</li>
     *                  <li>ip 登录用户客户端ip地址</li>
     *               </ul>
     * @throws Exception
     * @return ReturnMessage 返回的消息，子类如果不返回消息及状态码。
cas将使用默认消息及状态码。
     */
    ReturnMessage process(Map<String, Object> params) throws Exception;
}
```

UserMapperService

用户映射接口，用户在登陆本系统或第三方系统后，再去访问其它第三方系统时，需要提供一个用户映射的服务。子系统必须实现该接口，且提供用户的匹配映射。 必须实现

UserMapperService

```
/**
 * 用户接口
 * @author zxb
 * @version 1.0.1
 * Created by zxb on 2016/3/16.
 */
public interface UserMapperService {

    /**
     *
根据用户账号获取该用户的信息，子系统实现该接口，便于各系统之间完成用户映射。此处只
会被 dmcas 调用，且只会用于各系统间匹配校验。
     * @param userCode 用户账号
     * @return user 用户信息
     * @throws Exception
     */
    public User getUser(String userCode) throws Exception;

    /**
     * 根据其它系统提供的用户信息，返回本系统对应的用户信息。
     * @param user
     * @return
     * @throws Exception
     */
    public User getUser(User user) throws Exception;
}
```

2.2. 实现示例

2.2.1. AuthService

默认的用户名密码校验，采用的是对密码 *md5* 后进行一个比较的校验。实现逻辑很简单，把密码 *md5* 一把后和库中的 *md5* 后的密码进行一个等值比较。

MD5AuthService

```
public AuthResult validate(String username, String password) {
logger.info("begin validate user login info! username:" + username);
AuthResult authResult = new AuthResult();
if (!StringUtils.hasText(username) || !StringUtils.hasText(password)) {
    authResult.setValid(false);
    authResult.setMessage("用户名或密码不能为空！");
    return authResult;
}

Connection conn = null;
PreparedStatement ps = null;
```

```
ResultSet rs = null;
try {
    conn = dataSource.getConnection();
    ps = conn.prepareStatement(sql);
    ps.setString(1, username);
    rs = ps.executeQuery();
    String dbPassword = null;
    String flag = null;
    while (rs.next()) {
        dbPassword = rs.getString(1);
        flag = rs.getString(2);
    }
    if (!StringUtils.hasText(dbPassword)) {
        authResult.setValid(false);
        authResult.setMessage("用户不存在！");
        return authResult;
    }
    if (!dbPassword.equalsIgnoreCase(EncryptUtil.md5Degest(password))) {
        authResult.setValid(false);
        authResult.setMessage("用户或密码不正确！");
        return authResult;
    }
    if (!("1".equals(flag))) {
        authResult.setValid(false);
        authResult.setMessage("帐号已被禁用！");
        return authResult;
    }
    //获取登录用户信息
    User user = userService.queryByCodePassword(username, password);
    authResult.setUser(user);
    //验证是否成功
    authResult.setValid(true);
    return authResult;
} catch (Exception e) {
    logger.error("validate user:" + username + " error!", e);
    // 返回信息
    authResult.setValid(false);
    authResult.setMessage("验证失败！");
    return authResult;
} finally {
    this.release(rs, ps, conn);
    logger.info("begin validate user login info! username:" + username + ", validate:"
+ authResult.isValid() + ",msg:" + authResult.getMessage());
}
}
```

2.2.2. LoginPostProcessService

在用户登陆成功后，往往需要记录用户登陆行为的日志，刷新用户访问量等等。

LoginPostProcessServiceImpl 实现了该接口，这里则是添加了一些子系统独有的一些逻辑。

接口方法返回的 *ReturnMessage* 可以决定登陆成功后最终返回给客户端的信息，如 *ReturnCode* 和 *ReturnMessage*

LoginPostProcessServiceImpl

```
public class LoginPostProcessServiceImpl implements LoginPostProcessService {

    private Logger logger = Logger.getLogger(this.getClass());

    /**
     * 登录日志类型
     */
    private final String LOGIN_LOG_TYPE = "1001";

    /**
     * 日志服务接口
     */
    private LogService logService;

    /**
     * 参数列表
     */
    private ParamCache paramCache;

    public void setLogService(LogService logService) {
        this.logService = logService;
    }

    public void setParamCache(ParamCache paramCache) {
        this.paramCache = paramCache;
    }

    @Override
    public ReturnMessage process(Map<String, Object> params) throws Exception {
        logger.info("begin login success process!");
        ReturnMessage returnMessage = null;
        if (params != null && params.size() > 0) {
            // 记录用户登录日志
            User user = (User) params.get("user");
            String ip = params.get("ip").toString();

            logger.debug("create login log, user:" + user.getUser_code() + ", ip:" +
ip);
            this.logService.createLog(user, ip, LOGIN_LOG_TYPE);

            // 更新门户登陆统计信息
            this.logService.update();
```

```
            // 验证用户密码是否为初始化密码
            String username = params.get("username").toString();
            String password = params.get("password").toString();
            String initial_password = paramCache.getValue("INITIAL_PASSWORD");
            if (!StringUtils.hasText(initial_password)) {
                logger.error("initial_password is empty,verify password will not
effect!please check the table wmf_param!");
            } else {
                if (EncryptUtil.md5Degest(initial_password).equalsIgnoreCase(password
)) {
                    returnMessage = new ReturnMessage();
                    returnMessage.setReturnCode("1");
                    returnMessage.setReturnMsg("您的密码还是初始化密码，请及时修改！"
);
                    logger.info("verify success!warn, the user " + username + "'s
password is initial_password!");
                }
            }
        }
        logger.info("end login success process!");
        return returnMessage;
    }
}
```

2.2.3. LoginFailProcessService

LoginFailProcessService 同 *LoginPostProcessService* 接口类似，它在用户登陆失败时会调用。

LoginFailProcessServiceImpl 实现了该接口，添加了一些日志记录的逻辑。

LoginFailProcessServiceImpl

```
/**
 * 登录失败后的处理逻辑
 * Created by zxb on 2016/3/14.
 */
public class LoginFailProcessServiceImpl implements LoginFailProcessService {

    private Logger logger = Logger.getLogger(this.getClass());

    private MistakeLogService mistakeLogService;

    /**
     * 登录日志类型
     */
    private final String LOGIN_LOG_TYPE = "1001";

    public void setMistakeLogService(MistakeLogService mistakeLogService) {
        this.mistakeLogService = mistakeLogService;
    }

    @Override
    public ReturnMessage process(Map<String, Object> params) throws Exception {
        logger.info("begin login fail process!");
        if (params != null && params.size() > 0) {
            // 创建登录失败日志
            User user = (User) params.get("user");
            String ip = params.get("ip").toString();

            logger.debug("login fail user:" + (user == null ? "" : user.getUser_code(
)) + ", ip:" + ip);
            mistakeLogService.createLog(user, ip, LOGIN_LOG_TYPE);
        }
        logger.info("end login fail process!");
        return null;
    }
}
```

2.2.4. UserMapperService

UserMapperService 作为各子系统间互相访问时的一个服务接口。它要求子系统实现它时，需要提供用户映射的服务。

UserMapperServiceImpl 实现了该接口的两个方法。

UserMapperServiceImpl

```
/**
 * 用户映射服务实现
 *
```

```
 * @author zxb
 * @version 1.0.1
 *          Created by zxb on 2016/3/17.
 */
public class UserMapperServiceImpl implements UserMapperService {

    private Logger logger = Logger.getLogger(this.getClass());

    /**
     * 用户操作接口
     */
    private UserService userService;

    /**
     * 设置用户操作接口
     */
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Override
    public User getUser(String userCode) throws Exception {
        if (StringUtils.hasText(userCode)) {
            User user = this.userService.queryByCode(userCode);
            return user;
        }
        return null;
    }

    @Override
    public User getUser(User user) throws Exception {
        if (user == null || !StringUtils.hasText(user.getUser_code())) {
            logger.error("user is null or user_code is empty!");
            return null; ①
        }

        String userCode = user.getUser_code();
        String idCardNo = user.getUser_sfzh();
        logger.debug("other system's user, user_code " + userCode + ", idCardNo " +
idCardNo);

        // 非普通用户，如一般的管理用户
        if (!StringUtils.hasText(idCardNo)) {
            logger.error("current user " + userCode + " does'nt has idCardNo!");
            return null; // 阻止无身份证号的用户进行访问
        }

        // 本系统中映射用户不存在时，则创建用户
        User mappingUser = this.userService.queryByNameSfzh(userCode, idCardNo);
        if (mappingUser == null) {
            mappingUser = this.userService.createPkiUser(userCode, userCode, idCardNo,
```

```
"PKI001");
        }
        return mappingUser; ②
    }
}
```

① 没有匹配用户时，直接返回 *null* 则可以。

② 存在用户，则返回匹配的用户即可。

2.3. 发布服务实现

子系统提供的实现在统一单点登录系统中是通过 *rpc* 调用的，且采用的是较为成熟的 *dubbo* 框架完成。子系统需要通过 *dubbo* 将上述的实现类服务发布至注册中心即可。

2.3.1. 注册中心地址

注册中心采用的为 *zookeeper* 集群。实际地址请联系湖北公安云项目组获取。

2.3.2. 发布服务

引入依赖

1. 手动引入

   手动将以下 *jar* 包加入工程 *classpath* 下。

   Table 1. dubbo依赖jar包

jar名称	版本号	备注
dubbo	2.5.3	
netty	3.2.5.Final	
javassist	3.15.0-GA	
log4j	1.2.16	
commons-logging	1.1.1	
spring	2.5.6.SEC03	可替换为spring3

   如果连接 *zookeeper* 注册中心，则需要如下 *jar* 包

   Table 2. zookeeper client 依赖包

jar名称	版本号	备注
zkclient	0.1	
zookeeper	3.3.3	
log4j	1.2.15	可替换为高版本
jline	0.9.94	

junit	3.8.1	

2. maven引入

```
<!-- dubbo -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.5.3</version>
    <exclusions>
        <exclusion> ①
            <groupId>org.springframework</groupId>
            <artifactId>spring</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<!-- zookeeper client -->
<dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
    <version>0.1</version>
</dependency>
```

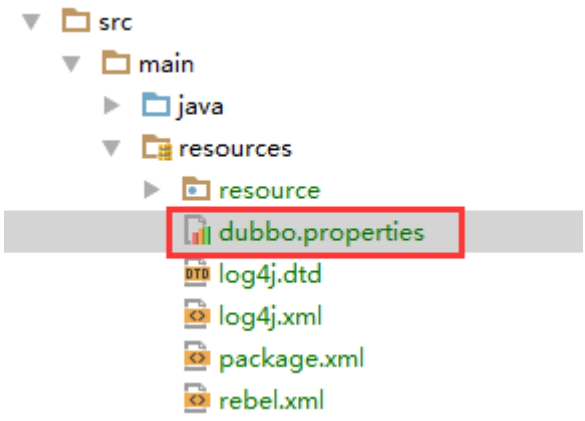
① 如果工程已有 *spring* 高版本的依赖，此处可排除 *spring* 依赖，以免引起冲突

添加配置文件

1. dubbo基础环境配置文件

   dubbo.properties 文件为 *dubbo* 的默认全局环境配置文件。

   在工程的 *classpath* 下添加 *dubbo.properties* 文件，如下图。



   编辑 *dubbo.properties* 添加如下内容

```
# 指定容器
dubbo.container=log4j,spring
dubbo.application.name=dmga-dubbo-validation
dubbo.application.owner=

# 指定注册中心地址
dubbo.registry.address=zookeeper://localhost:2181  ①
dubbo.registry.file=validation.cache

# 指定服务协议
dubbo.protocol.name=dubbo  ②
dubbo.protocol.port=-1
dubbo.protocol.accesslog=true

# 指定服务配置
dubbo.service.loadbalance=roundrobin
dubbo.service.group=jingzong  ③
dubbo.service.timeout=5000
dubbo.service.retries=0
dubbo.service.version=1.0   ④

# 指定spring文件加载位置
dubbo.spring.config=classpath:resource/beans/beans-*.xml ⑤

# 指定关闭钩子
dubbo.shutdown.hook=true
```

① 生产环境注册中心地址，请联系湖北公安云项目组获取

② 指定协议，建议使用 *dubbo*

③ 指定服务所属组，必须指定

④ 指定服务版本号，必须指定

⑤ 如果为 *web* 方式启动，此处不需要指定 *spring* 配置文件路径

具体配置以 *dubbo* 官网文档为准

1. dubbo服务配置文件

   在 *classpath* 下添加 *spring* 配置文件，例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!-- 校验服务实现,md5方式 -->
    <dubbo:service interface="com.dm.cas.server.service.AuthService" ref=
"md5AuthService"/>

    <!-- 登录成功后处理服务实现 -->
    <dubbo:service interface="com.dm.cas.server.service.LoginPostProcessService"
ref="loginPostProcessService"/>

    <!-- 登录失败后的处理服务实现 -->
    <dubbo:service interface="com.dm.cas.server.service.LoginFailProcessService"
ref="loginFailProcessService"/>

    <!-- 用户映射服务, 默认实现 -->
    <dubbo:service interface="com.dm.cas.server.service.UserMapperService" ref=
"userMappingService" />

    <!-- 用户登出处理服务实现  -->
    <dubbo:service interface="com.dm.cas.server.service.LogoutProcessService"
ref="logoutProcessService" /> ①
</beans>
```

① *ref* 引用的为具体的 *spring bean*

启动服务

启动服务时将上述配置的 *dubbo* 相关的 *spring* 配置文件加载起来即可。

启动服务成功后，需要联系 湖北公安云 项目组将需要接入单点的服务 *ip* 或域名 添加至 **信任列表** 。