cas 单点登录与 spring boot 关联使用

1.cas单点登录介绍

单点登录（ Single Sign-On , 简称 SSO ）是目前比较流行的服务于企业业务整合的解决方案之一， SSO 使得在多个应用系统中，用户只需要 登录一次 就可以访问所有相互信任的应用系统。

从结构体系看， CAS 包括两部分： CAS Server 和 CAS Client 。

CAS Server

CAS Server 负责完成对用户的认证工作 , 需要独立部署 , CAS Server 会处理用户名 / 密码等凭证(Credentials) 。

CAS Client

负责处理对客户端受保护资源的访问请求，需要对请求方进行身份认证时，重定向到 CAS Server 进认证。（原则上，客户端应用不再接受任何的用户名密码等 Credentials ）。

CAS Client 与受保护的客户端应用部署在一起，以 Filter 方式保护受保护的资源。

2.cas下载编译

jasig cas下载官网地址 https://www.apereo.org/projects/cas
github项目下载地址 https://github.com/apereo/cas/releases

我使用下载的是cas-4.1.10是maven构建的项目，如果你要下cas-4.2以上的版本可能就是gradle构的项目，如果你常用的是gradle，那就下载cas-4.2以上的版本，这里比较坑的就是cas的编译。

将下载的项目解压后，进入到项目目录里使用mvn clean install 进行编译，这里会等待很时间，如果没用翻墙可能有些jar包下载不下来，如果编译不了就在网上找别人编译后的项目，我在编译的时候就直编译不了，下载太慢，坑了我很长时间。

编译后将cas-server-webapp项目下的targer的war包放到tomcat的里，启动tomcat。这里的详细骤可以参考
Jasig cas 单点登录系统Server&Java Client配置

这里的搭建cas server一端就不详细说了，下面主要说说与spring boot的结合配置

3.spring boot的配置

这里需要你有一个spring boot的项目，如何搭建可以参考网上的教程。

引入jar包

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-cas</artifactId>
</dependency>
<dependency>
```

```
            <groupId>javax.inject</groupId>
            <artifactId>javax.inject</artifactId>
            <version>1</version>
        </dependency>
```

有过spring security的经验的应该知道关于权限登陆的用法， 实现AuthenticationUserDetailsService接口或UserDetailsService 接口，用作登陆验证。

```
import org.springframework.security.cas.authentication.CasAssertionAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.AuthenticationUserDetailsService;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import java.util.ArrayList;
import java.util.List;

/**
 * Authenticate a user from the database.
 */
public class CustomUserDetailsService implements AuthenticationUserDetailsService<CasAsse
tionAuthenticationToken> {

    @Override
    public UserDetails loadUserDetails(CasAssertionAuthenticationToken token) throws Usern
meNotFoundException {
        String login = token.getPrincipal().toString();
        String username = login.toLowerCase();

        List<GrantedAuthority> grantedAuthorities = new ArrayList<GrantedAuthority>();
        grantedAuthorities.add(new SimpleGrantedAuthority("ROLE_ADMIN"));

        return new AppUserDetails(username, grantedAuthorities);
    }

}
```

然后就用到了userDetails 接口，我们这里实现这个接口，并定义一些关于用户的信息和权限。如果在clien要使用这些信息，比如用户名，角色，权限等信息，你可以在这里处理一下。

```
package com.shang.spray.security;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

public class AppUserDetails implements UserDetails {
    /** */
    private static final long serialVersionUID = -4777124807325532850L;

    private String username;

    private String password;

    private boolean accountNonExpired;

    private boolean accountNonLocked;

    private boolean credentialsNonExpired;

    private boolean enabled;

    private Collection<? extends GrantedAuthority> authorities;

    private List<String> roles;

    public AppUserDetails() {
        super();
    }

    public AppUserDetails(String username, Collection<? extends GrantedAuthority> authoritie
) {
        super();
        this.username = username;
        this.password = "";
        this.accountNonExpired = true;
        this.accountNonLocked = true;
        this.credentialsNonExpired = true;
        this.enabled = true;
        this.authorities = authorities;
        this.roles = new ArrayList<>();
        this.roles.addAll(authorities.stream().map((Function<GrantedAuthority, String>) Granted
uthority::getAuthority).collect(Collectors.toList()));
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        /*
         * List<GrantedAuthority> l = new ArrayList<GrantedAuthority>(); l.add(new
         * GrantedAuthority() { private static final long serialVersionUID = 1L;
         *
         * @Override public String getAuthority() { return "ROLE_AUTHENTICATED"; } }); return l;
         */
        return authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }
    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return accountNonExpired;
    }

    @Override
    public boolean isAccountNonLocked() {
        return accountNonLocked;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return credentialsNonExpired;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }


}
```

接下来就是最重要的一个步骤，继承实现WebSecurityConfigurerAdapter 配置类，并配置关于单点录服务器的一些信息和拦截页面。这里先放上这个实现类

```
package com.shang.spray.security;

import org.jasig.cas.client.session.SingleSignOutFilter;
import org.jasig.cas.client.validation.Cas20ServiceTicketValidator;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.security.cas.ServiceProperties;
import org.springframework.security.cas.authentication.CasAssertionAuthenticationToken;
import org.springframework.security.cas.authentication.CasAuthenticationProvider;
import org.springframework.security.cas.web.CasAuthenticationEntryPoint;
import org.springframework.security.cas.web.CasAuthenticationFilter;
import org.springframework.security.config.annotation.authentication.builders.Authenticatio
ManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
rerAdapter;
import org.springframework.security.core.userdetails.AuthenticationUserDetailsService;
import org.springframework.security.web.authentication.logout.LogoutFilter;
import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandle
;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import javax.inject.Inject;

@Configuration
@EnableWebSecurity
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

    private static final String CAS_URL_LOGIN = "cas.service.login";
    private static final String CAS_URL_LOGOUT = "cas.service.logout";
    private static final String CAS_URL_PREFIX = "cas.url.prefix";
    private static final String CAS_SERVICE_URL = "app.service.security";
    private static final String APP_SERVICE_HOME = "app.service.home";

    @Inject
    private Environment env;


    @Bean
    public ServiceProperties serviceProperties() {
        ServiceProperties sp = new ServiceProperties();
        sp.setService(env.getRequiredProperty(CAS_SERVICE_URL));
        sp.setSendRenew(false);
        return sp;
    }

    @Bean
    public CasAuthenticationProvider casAuthenticationProvider() {
        CasAuthenticationProvider casAuthenticationProvider = new CasAuthenticationProvider()

        casAuthenticationProvider.setAuthenticationUserDetailsService(customUserDetailsServic
());
        casAuthenticationProvider.setServiceProperties(serviceProperties());
        casAuthenticationProvider.setTicketValidator(cas20ServiceTicketValidator());
        casAuthenticationProvider.setKey("an_id_for_this_auth_provider_only");
        return casAuthenticationProvider;
    }

    @Bean
    public AuthenticationUserDetailsService<CasAssertionAuthenticationToken> customUserD
tailsService() {
        return new CustomUserDetailsService();
    }

    @Bean
    public Cas20ServiceTicketValidator cas20ServiceTicketValidator() {
        return new Cas20ServiceTicketValidator(env.getRequiredProperty(CAS_URL_PREFIX));
    }

    @Bean
    public CasAuthenticationFilter casAuthenticationFilter() throws Exception {
        CasAuthenticationFilter casAuthenticationFilter = new CasAuthenticationFilter();
        casAuthenticationFilter.setAuthenticationManager(authenticationManager());
        casAuthenticationFilter.setFilterProcessesUrl("/j_spring_cas_security_check");
        return casAuthenticationFilter;
    }

    @Bean
    public CasAuthenticationEntryPoint casAuthenticationEntryPoint() {
        CasAuthenticationEntryPoint casAuthenticationEntryPoint = new CasAuthenticationEntry
oint();
        casAuthenticationEntryPoint.setLoginUrl(env.getRequiredProperty(CAS_URL_LOGIN));
        casAuthenticationEntryPoint.setServiceProperties(serviceProperties());
        return casAuthenticationEntryPoint;
    }

    @Bean
    public SingleSignOutFilter singleSignOutFilter() {
        SingleSignOutFilter singleSignOutFilter = new SingleSignOutFilter();
        singleSignOutFilter.setCasServerUrlPrefix(env.getRequiredProperty(CAS_URL_PREFIX));
        return singleSignOutFilter;
    }

    @Bean
    public LogoutFilter requestCasGlobalLogoutFilter() {
        LogoutFilter logoutFilter = new LogoutFilter(env.getRequiredProperty(CAS_URL_LOGOUT
 + "?service="
                + env.getRequiredProperty(APP_SERVICE_HOME), new SecurityContextLogoutHand
er());
        logoutFilter.setLogoutRequestMatcher(new AntPathRequestMatcher("/logout", "POST"));
        return logoutFilter;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(casAuthenticationProvider());
    }


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.addFilter(casAuthenticationFilter());
        http.exceptionHandling().authenticationEntryPoint(casAuthenticationEntryPoint());
        http.addFilterBefore(singleSignOutFilter(), CasAuthenticationFilter.class)
                .addFilterBefore(requestCasGlobalLogoutFilter(), LogoutFilter.class);

        http.csrf().disable();
        http.headers().frameOptions().disable();
        http.authorizeRequests().antMatchers("/assets/**").permitAll()
                .anyRequest().authenticated();

        http.logout().logoutUrl("/logout").logoutSuccessUrl("/").invalidateHttpSession(true)
                .deleteCookies("JSESSIONID");
    }
}
```

配置信息

```
#cas
```

```
app.service.security=http://localhost:8200/j_spring_cas_security_check
app.service.home=http://localhost:8200

cas.service.login=http://localhost:8080/cas/login
cas.service.logout=http://localhost:8080/cas/logout
cas.url.prefix=http://localhost:8080/cas
```

关于上面的一些说明:

1. servicePropertiesi单点登录服务器地址
2. casAuthenticationProvider 权限登录配置
3. 客户端登录地址和退出地址的配置
4. 客户端拦截的路径配置
5. 配置后需要启用配置 @Configuration 和@EnableWebSecurity

4.结束

以上单点登录cas与spring boot的配置使用，对于有多个后台的系统，写一个单独的单点登录系统来所有的平台来管理感觉非常有必要。 没有使用过的可以去尝试一下，简单好用你值的一学。

有什么问题欢迎给我来信或留言!