<https://blog.csdn.net/TreeShu321/article/details/99177115>

**二、Session 超时**

当用户登录后，我们可以设置 session 的超时时间，当达到超时时间后，自动将用户退出登录。

Session 超时的配置是 SpringBoot 原生支持的，我们只需要在 application.properties 配置文件中配置：

# session 过期时间，单位：秒

server.servlet.session.timeout=60

**Tip：  
从用户最后一次操作开始计算过期时间。  
过期时间最小值为 60 秒，如果你设置的值小于 60 秒，也会被更改为 60 秒。**

我们可以在 Spring Security 中配置处理逻辑，在 session 过期退出时调用。修改 WebSecurityConfig 的 configure() 方法，添加：

.sessionManagement()

// 以下二选一

//.invalidSessionStrategy()

//.invalidSessionUrl();

Spring Security 提供了两种处理配置，一个是 invalidSessionStrategy()，另外一个是 invalidSessionUrl()。

这两个的区别就是一个是前者是在一个类中进行处理，后者是直接跳转到一个 Url。简单起见，我就直接用 invalidSessionUrl()了，跳转到 /login/invalid，我们需要把该 Url 设置为免授权访问， 配置如下：

@RequestMapping("/login/invalid")

@ResponseStatus(HttpStatus.UNAUTHORIZED)

@ResponseBody

public String invalid() {

return "Session 已过期，请重新登录";

}

## 三、限制最大登录数

接下来实现限制最大登陆数，原理就是限制单个用户能够存在的最大 session 数。

在上一节的基础上，修改 configure() 为：

增加了下面三行代码，其中：

maximumSessions(int)：指定最大登录数  
maxSessionsPreventsLogin(boolean)：是否保留已经登录的用户；为true，新用户无法登录；为 false，旧用户被踢出  
expiredSessionStrategy(SessionInformationExpiredStrategy)：旧用户被踢出后处理方法

**maxSessionsPreventsLogin()可能不太好理解，这里我们先设为 false，效果和 QQ 登录是一样的，登陆后之前登录的账户被踢出。**

在 onExpiredSessionDetected() 方法中，处理相关逻辑，我这里只是简单的返回一句话。

执行程序，打开两个浏览器，登录同一个账户。因为我设置了 maximumSessions(1)，也就是单个用户只能存在一个 session，因此当你刷新先登录的那个浏览器时，被提示踢出了。

## 四、踢出用户

下面来看下如何主动踢出一个用户。

首先需要在容器中注入名为 SessionRegistry 的 Bean，这里我就简单的写在 WebSecurityConfig 中：

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

// 如果有允许匿名的url，填在下面

.antMatchers("/login/invalid").permitAll()

.anyRequest().authenticated().and()

// 设置登陆页

.formLogin().loginPage("/login")

.successHandler(customAuthenticationSuccessHandler)

.failureHandler(customAuthenticationFailureHandler)

.permitAll()

.and()

// .failureUrl("/login/error")

// .defaultSuccessUrl("/")

.sessionManagement()

.invalidSessionUrl("/login/invalid")

//指定最大登录数

.maximumSessions(1)

//当达到最大值时，是否保留已经登录的用户

.maxSessionsPreventsLogin(false)

//当达到最大值时，旧用户被踢出后的操作

.expiredSessionStrategy(customExpiredSessionStrategy)

.sessionRegistry(sessionRegistry());

http.csrf().disable();

}

@Bean

public SessionRegistry sessionRegistry() {

return new SessionRegistryImpl();

}

编写一个接口用于测试踢出用户：

@GetMapping("/click")

@ResponseBody

public String removeUserSessionByUsername(@RequestParam String username) {

int count = 0;

List<Object> users = sessionRegistry.getAllPrincipals();

for (Object user:users) {

if (user instanceof User) {

String principalName = ((User) user).getUsername();

if (principalName.equals(username)) {

List<SessionInformation> allSessions = sessionRegistry.getAllSessions(user, false);

if (allSessions != null && allSessions.size() > 0) {

for (SessionInformation sessionInformation:allSessions) {

sessionInformation.expireNow();

count++;

}

}

}

}

}

return "操作成功，清理session共" + count + "个";

}

1. sessionRegistry.getAllPrincipals(); 获取所有 principal 信息
2. 通过 principal.getUsername 是否等于输入值，获取到指定用户的 principal
3. sessionRegistry.getAllSessions(principal, false)获取该 principal 上的所有 session
4. 通过 sessionInformation.expireNow() 使得 session 过期

运行程序，分别使用 admin 和 zhao账户登录，admin 访问 /kick?username=zhao 来踢出用户 zhao，zhao 刷新页面，发现被踢出。

**五、退出登录**

补充一下退出登录的内容，在之前，我们直接在 WebSecurityConfig 的 configure() 方法中，配置了：

http.logout();

这就是 Spring Security 的默认退出配置，Spring Security 在退出时候做了这样几件事：

1. 使当前的 session 失效
2. 清除与当前用户有关的 remember-me 记录
3. 清空当前的 SecurityContext
4. 重定向到登录页  
   Spring Security 默认的退出 Url 是 /logout，我们可以修改默认的退出 Url，例如修改为 /signout：  
   WebSecurityConfig配置如下：

@Autowired

private CustomLogoutSuccessHandler customLogoutSuccessHandler;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

// 如果有允许匿名的url，填在下面

.antMatchers("/login/invalid").permitAll()

.anyRequest().authenticated().and()

// 设置登陆页

.formLogin().loginPage("/login")

.successHandler(customAuthenticationSuccessHandler)

.failureHandler(customAuthenticationFailureHandler)

.permitAll()

.and()

.logout()

.logoutUrl("/signout")

.deleteCookies("JSESSIONID")

.logoutSuccessHandler(customLogoutSuccessHandler)

.and()

// .failureUrl("/login/error")

// .defaultSuccessUrl("/")

.sessionManagement()

.invalidSessionUrl("/login/invalid")

//指定最大登录数

.maximumSessions(1)

//当达到最大值时，是否保留已经登录的用户

.maxSessionsPreventsLogin(false)

//当达到最大值时，旧用户被踢出后的操作

.expiredSessionStrategy(customExpiredSessionStrategy)

.sessionRegistry(sessionRegistry());

http.csrf().disable();

}

@Component

public class CustomLogoutSuccessHandler implements LogoutSuccessHandler {

private final Logger log = LoggerFactory.getLogger(CustomLogoutSuccessHandler.class);

@Override

public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {

String username = ((User)authentication.getPrincipal()).getUsername();

log.info("退出成功，用户名：{}", username);

// 重定向到登录页

response.sendRedirect("/login");

}

}

## 六、Session 共享

在最后补充下关于 Session 共享的知识点，一般情况下，一个程序为了保证稳定至少要部署两个，构成集群。那么就牵扯到了 Session 共享的问题，不然用户在 8080 登录成功后，后续访问了 8060 服务器，结果又提示没有登录。

这里就简单实现下 Session 共享，采用 Redis 来存储。

### 6.1 导入依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-data-redis</artifactId>

</dependency>

在 application.xml 中新增配置指定 redis 地址以及 session 的存储方式

spring.redis.host=localhost

spring.redis.port=6379

spring.session.store-type=redis

@EnableRedisHttpSession

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}