Spring-Security

由于之前写的《Spring-security》比较杂乱，这里重启炉灶！

1. 授权方式

1.1 prePostEnabled

打开配置：

```
//@PreAuthorize、@PostAuthorize、@PreFilter、@PostFilter都需要此配置来开启
@EnableGlobalMethodSecurity(prePostEnabled=true)
```

1.1.1 @Pre/PostAuthorize

@PreAuthorize

> 在方法执行之前执行，而且这里可以调用方法的参数，也可以得到参数值，这是利用JAVA8的参数名反射特性，如果没用JAVA8，那么也可以利用Spring Security的@P标注参数，或者Spring Data的@Param标注参数

> @PreAuthorize is checked on the basis of **role** or the **argument** which is passed to the method.

```
@PreAuthorize("#userId == authentication.principal.userId or hasAuthority('ADMIN')")
public void changePassword(@P("userId") long userId ){

}
```

@PostAuthorize

> 在方法执行之后执行，而且这里可以调用方法的返回值，如果EL为false，那么该方法也已经执行完了，可能会回滚。returnObject（EL变量）表示返回的对象。

> @PostAuthorize can be authorized on the basis of logged in **roles**, **return object** (returnObject`) by method and **passed argument** to the method.

```
@PostAuthorize
public User getUser("returnObject.userId == authentication.principal.userId or hasPermission(returnObject,
'ADMIN')"){

}
```

注解使用示例

service层：

```
package com.concretepage.service;

import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.security.access.prepost.PreAuthorize;
import com.concretepage.bean.Book;

public interface IBookService {
        @PreAuthorize ("hasRole('ROLE_WRITE')")
```

```
        public void addBook(Book book);

        @PostAuthorize ("returnObject.owner == authentication.name")
        public Book getBook();

        @PreAuthorize ("#book.owner == authentication.name")
        public void deleteBook(Book book);
}
```

> authentication and principal keyword can directly be used to access user information. # is used to access argument of the method. Take attention on @PostAuthorize, built-in keyword *returnObject* has been used. Here returnObject is equivalent to Book instance returned by the method. 原文

意思：@PreAuthorize和@PostAuthorize都可以直接使用**authentication**和**principal**来获取与用户信息，@PostAuthorize中可以使用关键字**returnObject**，表示方法返回的对象；

1.1.2 @Pre/PostFilter

- 这2个注解可以使用内建对象`filterObject`，此对象代表列表或数组的元素, 其他内建对象及表达式

```
// 此处的filterObject就代表Task元素
@PostFilter("filterObject.assignee == authentication.name")
List<Task> findAll() {
    ...
}
```

@PostFilter

> *@PostFilter* defines a rule for filtering the return list of a method, by **applying that rule to every element in the list**. If the evaluated value is true, the item will be kept in the list. Otherwise, the item will be removed.

> 在方法执行之后执行，而且这里可以调用方法的返回值，然后对返回值进行过滤或处理或修改并返回。EL变量returnObject表示返回的对象。只有方法返回的是集合或数组类型的才可以使用。（与分页技术不兼容）

```
@postFilter
public User getUser("hasPermission(returnObject, 'ADMIN')"){

}
```

@PreFilter

> *@PreFilter* works in a very similar fashion, however, the filtering is applied to a list that is being passed as an input parameter to the annotated method.

> 在方法执行之前执行，而且这里可以调用方法的参数，然后对参数值进行过滤或处理或修改，EL变量filterObject表示参数，如有多个参数，使用filterTarget注解参数。只有方法参数是集合或数组才行。（很少会用到，与分页技术不兼容）

2. session

2.1 设置超时时间

2.1.1 springBoot

```
server.session.timeout=60
```

2.2 设置session

```
protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()
                        // 最多同时登陆几个客户端，后面的会挤掉前面的
                .maximumSessions(1)
                        // 配合maximumSessions，达到最大值后，拒绝登陆
                // .maxSessionsPreventsLogin(true)
                .expiredUrl("/login")
                // 当再次请求的时候，如果检测到session失效，如何处理
                .expiredSessionStrategy(sessionInformationExpiredStrategy)
                        // session保存策略，默认保存在内存中，所以重启系统需要重新登录
                .sessionRegistry(new SessionRegistryImpl());
}
```

3. 获取当前用户

3.1 静态方法

```
public class SpringSecurityUtil {

    //session 由controller 注入参数传入
    public static String currentUser(HttpSession session) {
        SecurityContextImpl securityContext = (SecurityContextImpl) session
            .getAttribute("SPRING_SECURITY_CONTEXT");
        return ((UserDetails)securityContext.getAuthentication()
                .getPrincipal()).getUsername();
    }
}
```