CS544 EA

Applications

Spring Security: Method Security

Method Security

- Method security is important: defense in depth!
- 3 types of security annotations supported:
  - @Secured
  - JSR-250 annotations
  - @PreAuthorize and @PostAuthorize

Nice tutorial at: https://www.baeldung.com/spring-security-method-security

Enabling Method Security

Maven dependency for Method Security

```
<dependency>
     <groupId>org.springframework.security</groupId>
     <artifactId>spring-security-config</artifactId>
</dependency>
```

```
@Configuration
@EnableWebMvc
@EnableGlobalMethodSecurity(
          securedEnabled = true,
          jsr250Enabled = true,
          prePostEnabled = true)
@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
```

Can be added to any @Configuration class

Enable which annotations you want

```
 <sec:global-method-security
     secured-annotations="enabled"
     jsr250-annotations="enabled"
     pre-post-annotations="enabled"/>
```

Or with XML config

@Secured

- Spring Security’s original annotation
  - Specify which Roles are allowed to execute

```
@Service
@Transactional
public class ContactService {
     @Resource
     private ContactDao contactDao;

     @Secured({ "ROLE_USER", "ROLE_ADMIN" })
     public Contact get(Long id) {
          return contactDao.getOne(id);
     }

     @Secured("ROLE_ADMIN")
     public void add(Contact contact) {
          contactDao.save(contact);
     }
}
```

As always you can also add them to the class level to apply to all methods

JSR-250

- Very similar to @Secured (but Java standard)

```
@Service
@Transactional
public class ContactService {
     @Resource
     private ContactDao contactDao;

     @RolesAllowed({ "ROLE_USER", "ROLE_ADMIN" })
     public Contact get(Long id) {
          return contactDao.getOne(id);
     }

     @RolesAllowed("ROLE_ADMIN")
     public void add(Contact contact) {
          contactDao.save(contact);
     }
}
```

@Pre / @PostAuthorize

- Modern Spring annotations
  - Can use security expressions
  - Can access arguments / return values

```
@Service
@Transactional
public class ContactService {
        @Resource
        private ContactDao contactDao;

        @PreAuthorize("hasRole('USER')")
        public Contact get(Long id) {
                return contactDao.getOne(id);
        }

        @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
        public void add(Contact contact) {
                contactDao.save(contact);
        }
}
```

Pre or Post

- @PreAuthorize has access to incoming params to make an authorization decision

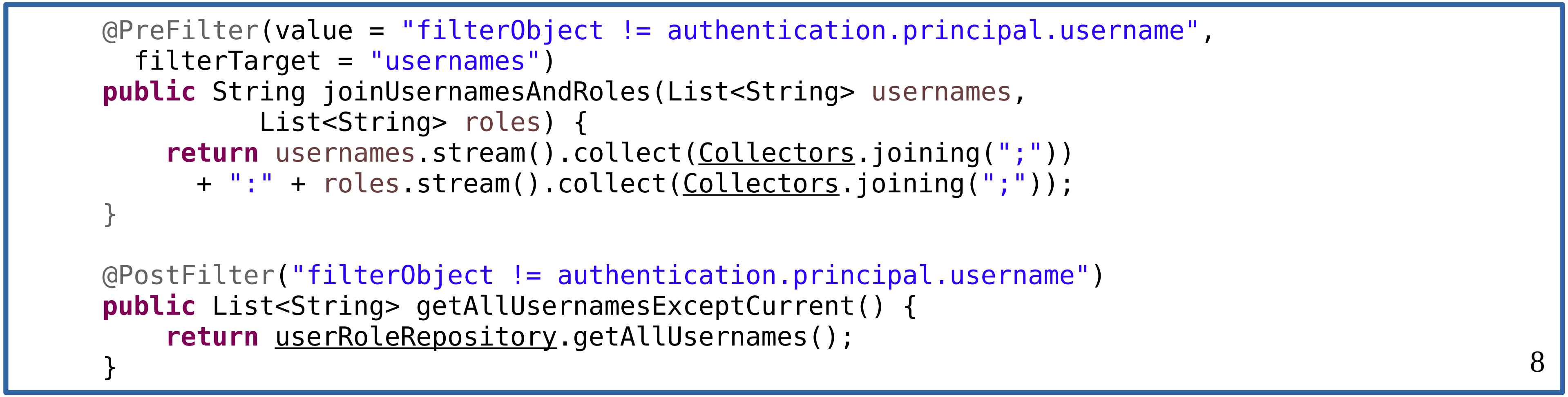
```
@PreAuthorize("#id < 100")
public Contact get(Long id) {
     return contactDao.getOne(id);
}
```

- @PostAuthorize executes method and then has access to the return (to make a decision)

```
@PostAuthorize("returnObject.name != 'bob'")
public Contact get(Long id) {
     contactDao.save(contact);
}
```

@Pre / @PostFilter

- Filter lets you remove items from a collection
  - @PreFilter can remove from parameter collection
  - @PostFilter can remove from returned collection



```
@PreFilter(value = "filterObject != authentication.principal.username",
  filterTarget = "usernames")
public String joinUsernamesAndRoles(List<String> usernames,
          List<String> roles) {
    return usernames.stream().collect(Collectors.joining(";"))
      + ":" + roles.stream().collect(Collectors.joining(";"));
}

@PostFilter("filterObject != authentication.principal.username")
public List<String> getAllUsernamesExceptCurrent() {
    return userRoleRepository.getAllUsernames();
}
```