Kayıt Düzeyinde Yetkilendirme (ACL)

ACL Nedir?

- **Access Control List** (ACL) **kayıt düzeyinde yetkilendirme** yapmayı sağlar
- **Aynı rol**e sahip iki kullanıcının erişebildiği bir sayfada **kullanıcıya göre farklı verile**rin gösterilmesi ihtiyacı kayıt düzeyinde yetkilendirmeye örnek gösterilebilir
- Spring Security'nin **metot düzeyinde yetkilendirme** kabiliyeti üzerine kuruludur

ACL'in Özellikleri

- Uygulamadaki domain nesnelerine ait **ACL verileri** tanımlanır
- Bu ACL verilerinde **kullanıcı veya rol düzeyinde** yapılabilecek veya yapılamayacak işlemler belirtilir
- Bunlara **permission** adı verilir
- ACL verileri arasında **parent-child ilişki** de olabilir
- Bazı ACL verileri diğer bazı **parent ACL** verilerinden inherit edebilir

ACL'in Özellikleri

- Uygulama tarafı veri modeli ile ACL tarafı veri modeli arasında hiçbir **foreign key constraint ilişkisi** söz konusu değildir
- Farklı türde domain nesnelerinin hepsinin **ACL verisi ortak bir yapı üzerinde** yönetilmektedir
- Spring Security **ACL verileri üzerinde efektif biçimde çalışma**yı sağlayan bir altyapı sunmaktadır

ACL'in Özellikleri

- Domain nesnelerinin ACL bilgilerine bakarak aktif kullanıcının veya sahip olduğu rollerin **metot çağrısı öncesinde nesneleri üzerinde ilgili işlemi yapıp yapamayacağı**na karar verilebilir
- Ya da aktif kullanıcının veya sahip olduğu rollerin **metot çağrısı sonunda dönülen domain nesneleri üzerinde erişim yetkisi**nin olup olmadığı kontrol edilebilir

Uygulama Tarafı Veri Modeline Örnek

T_OWNER	
ID	USERNAME
101	ayucel
102	vguclu
103	admin

T_PET		
ID	OWNER_ID	NAME
10	101	maviş
20	101	karabaş
30	102	cingöz

ACL Tarafı Veri Modeline Örnek

ACL_CLASS	
ID	DOMAIN_CLASS
1	com.javaegitimleri.petclinic.model.Pet

ACL_SID	
ID	USER_SID
1	admin
2	ayucel
3	vguclu

ACL_OBJECT_IDENTITY			
ID	DOMAIN_CLASS_ID	OBJECT_ID	ACL_OWNER_SID_ID
1000	1	10	1
2000	1	20	1
3000	1	30	1

ACL_ENTRY			
ID	ACL_OBJECT_IDENTITY_ID	ACL_SID_ID	MASK
1	1000	2	3
2	2000	2	3
3	3000	3	3

ACL Yetkileri (Permission)

JAVA Eğitimleri

- **Permission** arayüzü ile tanımlanırlar
- **BasePermission** sınıfında built-in yetkiler tanımlanmıştır
- Yetkiler veritabanında **integer bit masking** yöntemi ile tutulur
  - Bit 0:READ mask değeri :1
  - Bit 1:WRITE mask değeri :2
  - Bit 2:CREATE mask değeri :4
  - Bit 3:DELETE mask değeri :8
  - Bit 4:ADMINISTRATION mask değeri :16

ACL Verisinin Yönetimi


```
TestingAuthenticationToken authToken = new
TestingAuthenticationToken(
"admin", "secret");
authToken.setAuthenticated(true);
SecurityContextHolder.getContext()
.setAuthentication(authToken);
```

ACL verisinin yönetmek için **SecurityContext**'de geçerli bir **Authentication** nesnesinin olması gerekir

```
ObjectIdentity objectIdentity = new
ObjectIdentityImpl(Pet.class, 10L);
```

ObjectIdentity domain nesnesini ifade eder. İçerisinde domain class'ı ve id'si yer alır. Domain class'ında getId() isimli bir metodun olması gerekir

```
MutableAcl acl = (MutableAcl)
aclService.readAclById(objectIdentity);
```

AclService ve **MutableAclService** sınıfları ile domain nesnesinin **Acl** verisine varsa ulaşılır

ACL Verisinin Yönetimi

Her domain için en fazla bir Acl nesnesi dönülebilir. Eğer Acl verisi mevcut ise **isGranted()** metodu ile belirtilen kullanıcı veya rollerin, belirtilen yetkilere (permission) sahip olup olmadıkları kontrol edilebilir

```
acl.isGranted(
Arrays.asList(BasePermission.READ),
Arrays.asList((Sid) new PrincipalSid("ayucel")), false);

acl.isGranted(
Arrays.asList(BasePermission.WRITE),
Arrays.asList((Sid)new
GrantedAuthoritySid("ROLE_OWNER")),false);

List<AccessControlEntry> entries = acl.getEntries();

for (AccessControlEntry accessControlEntry : entries) {
    Permission p = accessControlEntry.getPermission();
}
```

İhtiyaç olursa **Acl** nesnesi içinden **AclControlEntry** nesnelerine de erişmek mümkündür

ACL Verisinin Yönetimi

MutableAclService kullanılarak bir domain nesnesi için **MutableAcl** nesnesi yaratılabilir. Bu **Acl** nesnesinin **insertAce()** metotları vasıtası ile belirli bir kullanıcı veya rol için yetkiler tanımlanabilir

```
acl = aclService.createAcl(objectIdentity);

acl.insertAce(acl.getEntries().size(), BasePermission.READ,
new PrincipalSid("ayucel"), true);

acl.insertAce(acl.getEntries().size(),BasePermission.WRITE,
new GrantedAuthoritySid("ROLE_OWNER"), true);

aclService.updateAcl(acl);
```

AclControlEntry'lerinin insert, update veya delete işlemlerinin ardından **MutableAclService** ile **MutableAcl** nesnesi update edilmelidir

ACL Verisinin Kullanımı

```
@PostFilter("hasPermission(filterObject,read) or
hasPermission(filterObject,write)")
Collection<Owner> findOwners(String lastName);

@PostAuthorize("hasPermission(returnObject,read) or
hasPermission(returnObject,write)")
Owner loadOwner(long id);

@PreAuthorize("hasPermission(#owner,delete)")
void deleteOwner(Owner owner);
```

PostFilter, **PreAuthorize** ve **PostAuthorize** annotasyonlarını metot düzeyinde kullanarak, ilgili metotlar çağrılmadan önce veya çağrıldıktan sonra yetki kontrolü yapmak mümkündür. Annotasyonların içerisine Spring Security'ye özel **SpEL expression**'ları yazılmaktadır

ACL Verisinin Kullanımı

```
public interface BusinessService {

@PreAuthorize("#owner.name == authentication.name")
public String secureMethod1(Owner owner);

@PreAuthorize("#o.name == authentication.name")
public String secureMethod2(@P("o") Owner owner);

@PostAuthorize("returnObject.ownerName ==
authentication.name")
public Pet secureMethod3();

@PostFilter("filterObject.ownerName ==
authentication.name")
public List<Pet> secureMethod4();

}
```

ACL Verisinin Kullanımı

- Mevcut **Authentication** token üzerinden kullanıcı ve rollerinin ilgili domain nesnesi üzerinde belirtilen yetkiye sahip olup olmadığı kontrol edilir
- **filterObject**, **returnObject**, **#owne**r gibi placeholder'lar expression içerisinde o andaki domain nesnesini ifade eder
- Yetkiler **BasePermission** sınıfındaki statik değişken isimleri ile veya mask değerleri ile belirtilebilir

ACL Verisinin Kullanımı

- Anotasyonlardaki **hasPermission** ifadesi arka tarafta işi **PermissionEvaluator**'a delege etmektedir
- İstenirse **PermissionEvaluator** kullanılarak da yetki kontrolü programatik biçimde gerçekleştirilebilir

ACL Verisinin Kullanımı

```
public Owner loadOwner(long id) {
    Owner owner = (Owner) sessionFactory
            .getCurrentSession().get(Owner.class, id);

    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

    boolean hasReadPermission =
               permissionEvaluator.hasPermission(authentication,
               owner,BasePermission.READ);

    if(!hasReadPermission) {
        throw new AccessDeniedException("Owner cannot be read");
    }

    return owner;

}
```

ACL Security Tag

- **JSP sayfaları**nın içerisinde **kayıt düzeyinde yetkilendirme** amaçlı kullanılabilir
- JSP **security taglib**'in sayfada tanımlı olması gerekir

```
<sec:accesscontrollist hasPermission="1,2"
                       domainObject="${pet}">

  <a href="detailView.jsp?id=${pet.id}"/>

</sec:accesscontrollist>
```

ACL Bean Konfigürasyonu

```
<security:global-method-security
pre-post-annotations="enabled">
    <security:expression-handler ref="securityExpressionHandler" />
</security:global-method-security>

<bean id="securityExpressionHandler"
class="org.springframework.security.access.expression.method.DefaultMethodS
ecurityExpressionHandler">
    <property name="permissionEvaluator" ref="permissionEvaluator" />
</bean>

<bean id="permissionEvaluator"
class="org.springframework.security.acls.AclPermissionEvaluator">
    <constructor-arg ref="aclService" />
</bean>

<bean id="aclService"
class="org.springframework.security.acls.jdbc.JdbcMutableAclService">
    <constructor-arg ref="dataSource" />
    <constructor-arg ref="lookupStrategy" />
    <constructor-arg ref="aclCache" />
</bean>
```

ACL Bean Konfigürasyonu

```
<bean id="lookupStrategy"
class="org.springframework.security.acls.jdbc.BasicLookupStrategy">
    <constructor-arg ref="dataSource" />
    <constructor-arg ref="aclCache" />
    <constructor-arg ref="aclAuthorizationStrategy" />
    <constructor-arg ref="permissionEvaluationStrategy" />
</bean>

<bean id="aclCache"
class="org.springframework.security.acls.domain.EhCacheBasedAclCache">
    <constructor-arg>
        <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
            <property name="cacheManager">
                <bean
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean" />
            </property>
            <property name="cacheName" value="aclCache" />
        </bean>
    </constructor-arg>
    <constructor-arg ref="aclAuthorizationStrategy" />
    <constructor-arg ref="permissionEvaluationStrategy" />
</bean>
```

ACL Bean Konfigürasyonu

```
<bean id="aclAuthorizationStrategy"
class="org.springframework.security.acls.domain.AclAuthorizationStrategyImpl">
    <constructor-arg>
        <list>
            <ref bean="roleAdmin" />
            <ref bean="roleAdmin" />
            <ref bean="roleAdmin" />
        </list>
    </constructor-arg>
</bean>

<bean id="roleAdmin"
class="org.springframework.security.core.authority.GrantedAuthorityImpl">
    <constructor-arg value="ROLE_ADMINISTRATOR" />
</bean>

<bean id="permissionGrantingStrategy"
class="org.springframework.security.acls.domain.DefaultPermissionGrantingStrat
egy">
    <constructor-arg ref="auditLogger"/>
</bean>

<bean id="auditLogger"
class="org.springframework.security.acls.domain.ConsoleAuditLogger" />
```

Metot Invokasyonundan Sonra Yetkilendirme

- Metot invokasyonundan sonra return değeri üzerinde **yetkilendirme** yapmak veya **AccessDeniedException** fırlatmak mümkündür
- Bu işlem için **AfterInvocationManager** arayüzü kullanılmaktadır
- Default implemantasyonu **AfterInvocationProviderManager** sınıfıdır
- AfterInvocationProviderManager da aslında **koordinatör** rolündedir

Metot Invokasyonundan Sonra Yetkilendirme

- Asıl iş bir grup **AfterInvocationProvider** implementasyonu tarafından yapılır
- Metot invokasyonundan dönen değer(ler) **AfterInvocationProvider** bean'ları tarafından incelenir ve yetkilendirme yapılır
- Metot return değeri bir **Collection** ise içerisinde **bazı değerler filtrelenebilir**, yada tekil değer ise **AccessDeniedException** fırlatılabilir

Metot Invokasyonundan Sonra Yetkilendirme

- **PostAuthorize** ve **PostFilter** annotasyonları metot return değerlerinin yetkilendirilmesinde kullanılmaktadır
- Bu anotasyonlar ile belirtilen yetki ifadelerini **PostInvocationAdviceProvider** ele almaktadır
- Bu bir **AfterInvocationProvider** implementasyonudur
- İfadeler arka tarafta yine **SecurityExpressionHandler**'a havale edilmektedir

Metot Invokasyonundan Sonra Yetkilendirme

```
<security:global-method-security pre-post-annotations="enabled">

    <security:expression-handler ref="securityExpressionHandler"/>

    <security:after-invocation-provider ref="afterInvocationProvider"/>

</security:global-method-security>


<bean id="afterInvocationProvider"
class="org.springframework.security.access.prepost.PostInvocationAdviceProvider">
    <constructor-arg>
        <bean
class="org.springframework.security.access.expression.method.ExpressionBasedPostI
nvocationAdvice">
            <constructor-arg ref="securityExpressionHandler"/>
        </bean>
    </constructor-arg>
</bean>
```

İlave ACL Yetkileri Tanımlama

- **BasePermission** sınıfı **extend** edilerek uygulamaya özel ACL yetkileri de tanımlanabilir
- Extend edilerek oluşturulan class **DefaultPermissionFactory** bean'ine **constructor argümanı** olarak verilmelidir
- **LookupStrategy** ve **PermissionEvalutor** bean'leri de bu PermissionFactory bean'ini kullanmalıdırlar

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com