Module - Overriding a filter in a custom module

A mechanism for dynamic registry and composing of filters has been developed in the application for searching the data in IdM (identities, roles, tree structure components, etc.). A new filter can be registered for existing REST services in any module.

The aim of this tutorial is to show how to override some actual filter behavior. We chosen **filter on identity's username**, which is represented by `username` parameter in search (url parameter). This filter is implemented in core module and search all identities, which have the same username as given parameter value (equals). We want to override this filter ⇒ we want to search all identities with username, which contain a given parameter value (like).

Source codes for this tutorial can be found in the example module

What you need before you start

- You need to install CzechIdM 7.5.0 (and higher). We have CzechIdM installed for this tutorial on server `http://localhost:8080/idm-backend`.
- Create identity to call tests with `curl`, which will have permission to read identities. We are using demo `admin:admin` identity.
- Create test identity with username `test12345`. This identity will be used in next step for testing.

01 Test search identity by username

At start, we will search identity by username - this way you can see how the core filter works. Only the core filter is implemented now, and as a result this command:

```
 $curl -u admin:admin
http://localhost:8080/idm-backend/api/v1/identities?username=test12345
```

returns our prepared test identity, whereas the command

```
 $curl -u admin:admin
http://localhost:8080/idm-backend/api/v1/identities?username=test1234
```

returns nothing (unless we have previously created an identity with the username `test1234`)).

02 Filter implementation

We are going to create a new filter in our module which searches all identities by username, which contains a given parameter value (like).

```
@Component("exampleUsernameIdentityFilter")
@Description("Filter by identity's username - search as \"like\" in username
- case insensitive")
public class UsernameIdentityFilter extends
AbstractFilterBuilder<IdmIdentity, IdmIdentityFilter> {

    @Autowired
    public UsernameIdentityFilter(IdmIdentityRepository repository) {
        super(repository);
    }

    @Override
    public String getName() {
        return IdmIdentityFilter.PARAMETER_USERNAME;
    }

    @Override
    public Predicate getPredicate(Root<IdmIdentity> root, CriteriaQuery<?>
query, CriteriaBuilder builder, IdmIdentityFilter filter) {
        if (filter.getUsername() == null) {
            return null;
        }
        return builder.like(builder.lower(root.get(IdmIdentity_.username)),
"%" + filter.getUsername() + "%");
    }

    @Override
    public int getOrder() {
        // 0 => default, we don't want to override filter as default, but is
possible when order is less than 0.
        return 10;
    }
}
```

The new filter is registered to search for identities (see FilterBuilder template), when parameter `IdmIdentityFilter.PARAMETER_USERNAME` is given in search parameters. Filter behavior is implemented in `getPredicate` method. The method `getOrder` says which filter is used by default. For example, if we want to add a new filter, and set it as default without any additional configuration, then we return value e.g. `-10`. Once set up this way, the filter will be used, when application starts as default (⇒ order before core filters), respectively before all filters with order greater than `-10`.

03 Configure filter usage

We added new filter with order `10`, so a default filter is still used, because all core filters have order `0` (**the smallest order wins** - see `@Ordered`). That is all we need in most cases. We don't intend to change default behavior, but we want to be able to configure which one of the filters is used explicitly ⇒ choose one implementation.

Go to the application configuration page (or to application.property file) and set the property value:

```
idm.sec.core.filter.IdmIdentity.username.impl=exampleUsernameIdentityFilter
```

We've added a new filter with order `10`, so the default filter is still used, since all core filters have order `0` (**the smallest order wins** - see `@Ordered`).

Read more about filter configuration.

04 Test search identity by username

Now, execute test commands again:

```
$curl -u admin:admin
http://localhost:8080/idm-backend/api/v1/identities?username=test12345
$curl -u admin:admin
http://localhost:8080/idm-backend/api/v1/identities?username=test1234
```

Congratulations, both command will find your test identity, because new filter is used.

05 Create test for filter

```
@Transactional
public class UsernameIdentityFilterTest extends AbstractIntegrationTest {

    @Autowired
    private UsernameIdentityFilter usernameIdentityFilter;

    @Test
    public void testFilteringFound() {
        String username = getHelper().createName();
        IdmIdentityDto identityOne = getHelper().createIdentity(username);
        IdmIdentityDto identityTwo =
getHelper().createIdentity(getHelper().createName() + username +
getHelper().createName());
        IdmIdentityDto identityThree =
getHelper().createIdentity(getHelper().createName() + username +
getHelper().createName());

        IdmIdentityFilter filter = new IdmIdentityFilter();
        filter.setUsername(username);
        List<IdmIdentity> identities = usernameIdentityFilter.find(filter,
null).getContent();

        assertEquals(3, identities.size());
```

```
        IdmIdentity identity = identities.stream().filter(ident ->
ident.getId().equals(identityOne.getId())).findFirst().get();
        assertNotNull(identity);

        identity = identities.stream().filter(ident ->
ident.getId().equals(identityTwo.getId())).findFirst().get();
        assertNotNull(identity);

        identity = identities.stream().filter(ident ->
ident.getId().equals(identityThree.getId())).findFirst().get();
        assertNotNull(identity);
    }

    @Test
    public void testFilteringNotFound() {
        String username = "usernameValue" + System.currentTimeMillis();
        getHelper().createIdentity(username);
        getHelper().createIdentity("123" + username +
getHelper().createName());
        getHelper().createIdentity(getHelper().createName() + username +
getHelper().createName());

        IdmIdentityFilter filter = new IdmIdentityFilter();
        filter.setUsername("username1Value"); // value is different than
variable username
        List<IdmIdentity> identities = usernameIdentityFilter.find(filter,
null).getContent();

        assertEquals(0, identities.size());
    }
}
```