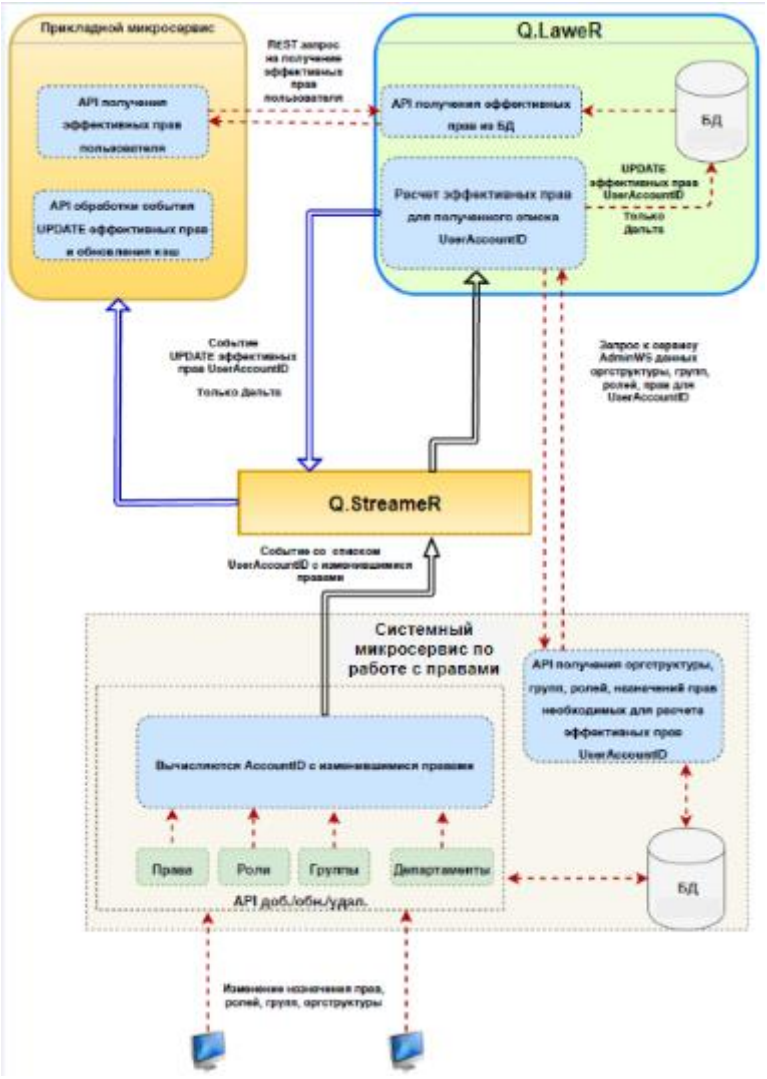
Функциональное описание Q.LawyeR

Exported on Jul 08, 2019

«Q.LawyeR» предназначен для работы с правами пользователей системы и подразделений.

Компонент обеспечивает функционал расчета и хранения эффективных прав пользователей системы. Прикладные бизнес-модули могут получитm доступ к интерфейсам компонента *«Q.LawyeR»* посредством подключения статической библиотеки **permission-right**. Библиотека ***permission-right*** предоставляет *API* для явной проверки прав пользователя, получения значений их атрибутов и генерации фильтров доступа. В качестве мастер-системы ведения прав на объекты учета могут выступать как системы собственной разработки, так и системы других вендоров.



Права определяют правила и условия доступа пользователей к объектам системы. Право рассматривается как именованный объект, имеющий два возможных значения: разрешено или запрещено, Для права можно декларировать набор атрибутов, которые могут иметь одно или несколько значений, кроме того значение атрибута может быть не задано. Значения права и его атрибутов используется для ограничения доступа к объектам информационной системы. Права определяются в прикладных модулях и регистрируются в репозитории метаданных. Права предоставляются и настраиваются администратором системы в рамках установленных в организации процедур управления доступом.

Для использования прав, в прикладном модуле необходимо добавить в pom.xml зависимость:

```
<dependency>
    <groupId>ru.diasoft.micro</groupId>
    <artifactId>permission-right</artifactId>
    <version>1.0.1</version>
</dependency>
```

Для главного класса приложения нужно указать аннотацию *@EnablePermissionControl*:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import ru.diasoft.micro.permission.annotation.EnablePermissionControl;

@SpringBootApplication
@EnableDiscoveryClient
@EnablePermissionControl
public class TemplateApplication {
 public static void main(String[] args) {
     SpringApplication.run(TemplateApplication.class, args);
 }
}
```

При использовании *Spring JPA*, ограничение доступа к объектам настраивается на классе сущности. Правила доступа к объектам разделены на четыре типа и определяются соответствующей аннотацией: создание объекта - аннотация *@CreatePermissions*, изменение объекта - *@UpdatePermissions*, удаление объекта - *@DeletePermissions*, и, собственно, доступ - *@AccessPermissions*. Если не заданы какие-то из *DML* аннотаций, то для соответствующих действий применяются правила *@AccessPermissions*. Аннотации типов доступа имеют один параметр, значением которого является список аннотаций *@Permission*, в каждой из которых определяется используемое право и условия применения атрибутов этого права. Пример использования *@AccessPermissions* приведен ниже:

```
@Getter
@Setter
@Entity
@Table(name="t_Account")
@AccessPermissions({
 @Permission(name = "BranchPermission", conditions = {
     @Condition(property = "branchID", attribute = "BranchID")
 }),
 @Permission(name = "AccountPermission", conditions = {
     @Condition(property = "areaID", attribute = "AreaID"),
     @Condition(property = "currencyID", attribute = "CurrencyID")
 })
})
public class AccountEntity {
   @Id
   @Column(name = "f_AccountID")
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Integer id;

   @Column(name = "f_BranchID")
   private Integer branchID;

   @Column(name = "f_BranchExtID")
   private Integer branchExtID;

   @Column(name = "f_AreaID")
   private Integer areaID;

   @Column(name = "f_FundID")
   private Integer currencyID;

   @Column(name = "f_Name", length = 240)
   private String name;

   @Column(name = "f_ClientID")
   private Integer clientID;

   @Column(name = "f_UniqueBrief", length = 20)
   private String uniqueBrief;

   @Column(name = "f_IsActive")
   private Short isActive;
}
```

Остальные аннотации доступа используются аналогичным образом. Аннотация @Permission имеет два параметра: *name*, в котором определяется системное наименование права, и *conditions*, который содержит список аннотаций *@Condition*. *@Condition* задает соответствие между атрибутом права, определяемым в параметре *attribute*, и полем сущности, определяемым в параметре *property*. Эти условия применяются в виде SQL-фильтров, автоматически встраиваемых в запросы к БД, при выполнении методов репозитория. Ниже приведен пример репозитория для приведенной выше сущности *AccountEntity*:

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.querydsl.QuerydslPredicateExecutor;
import org.springframework.stereotype.Repository;

@Repository
public interface AccountRepository extends JpaRepository<AccountEntity, Integer>,
QuerydslPredicateExecutor<AccountEntity> {
   Iterable<AccountEntity> getByClientID(Integer clientID);
}
```

Вызов *getByClientID* в API-методе контроллера сервиса:

```
@RestController
public class TemplateController {
   @Autowired
   private AccountRepository accountRepository;

   @PostMapping("/v1/AcccountByClientID")
   @ApiOperation(value = "Получение счетов по идентификатору клиента",
response=AccountEntity.class, produces = "application/json")
   public Iterable<AccountEntity> getAccountByClientID(@ApiParam(name = "ClientID",
value="Идентификатор клиента") @RequestParam Integer ClientID) {
      Iterable<AccountEntity> accountEntities = accountRepository.getByClientID(ClientID);
      return accountEntities;
   }
}
```

Метод *getByClientID* получает список счетов клиента по идентификатору клиента. При вызове метода будет выполнен следующий запрос:

```
select f_AccountAnlID, f_AreaID, f_BranchExtID, f_BranchID, f_ClientID, f_FundID, f_IsActive,
f_Name, f_UniqueBrief
from t_Account
where f_ClientID=? and f_BranchID in(?,?,?,?,?,?,?,?,?,?) and f_AreaID=? and f_FundID=?
```

В условных операторах фильтров применяются значения из соответствующих атрибутов права, настроенных администратором системы для текущего пользователя, выполняющего метод. Условия фильтров соединяются оператором *and*. Условный оператор фильтра зависит от количества значений соответствующего атрибута. Вследствие того что, количество параметров, которые можно передать в оператор *in*, ограничено, причем, это ограничение зависит от платформы, при наличии определенного числа значений атрибута, превышающего некоторый порог, фильтр приобретает более сложную форму, использующую специальную кэш-таблицу со значениями атрибута.

Соответствие между методом репозитория и типом доступа определяется правилами *Criteria API* и зависит от префикса метода. Имеется следующая зависимость: *@AccessPermissions* - применяется к методам репозитория, начинающимся на ***find***, ***read***, ***get***, ***query***, ***stream***, ***count***, ***exists***, *@DeletePermissions* - применяется к методам репозитория, начинающимся на ***delete***, ***remove***. Правила доступа, определяемые *@CreatePermissions* и *@UpdatePermissions*, применяются при выполнении методов с префиксом ***save*** в зависимости от того создается ли новый экземпляр объекта или изменяется существующий.

Если у пользователя не задано какое-либо право или не задан какой-то из используемых атрибутов права, то при вызове метода репозитория будет сгенерировано исключение *AccessDenied*.

Библиотека *permission-right* управляет процессом получения значений права и его атрибутов из системного сервиса *rightws*, Параметрами получения значений являются идентификатор пользователя и системное наименование права. Идентификатор текущего пользователя извлекается из токена доступа, с которым вызывается API-метод сервиса. Для повышения эффективности процесса, значения права и атрибутов кэшируются в прикладном сервисе. Обновление значения права в кэше выполняется через механизм сообщений, которые генерируют сервисы системной платформы при изменении администратором системы значений прав и атрибутов права.

Библиотека *permission-right* предоставляет *API* для явной проверки прав пользователя, получения значений их атрибутов и генерации фильтров доступа:

```
public interface PermissionProvider {
 // Получить идентификатор пользователя
 public Long getUserAccountID();
 // Получить значение атрибута права, заданного для пользователя, по системным
наименованиям права и атрибута
 public Object getAttributeValue(Session session, String rightSysName, String attributeSysName);
 // Проверить задано ли у пользователя право, по системному наименованию права
 public boolean checkPermission(Session session, String rightSysName);
 // Проверить заданы ли у пользователя права, по типу доступа
 public boolean checkPermission(Session session, Class<?> entityType, Class<? extends Annotation>
permissionType);
 // Проверить соответствует ли объект условиям доступа, по типу доступа
 public boolean validatePermission(Session session, Object entity, Class<? extends Annotation>
permissionType);
 // Сгенерировать текст фильтра по классу объекта и типу доступа
 public String generatePermissionFilter(Session session, Class<?> entityType, Class<? extends
Annotation> permissionType, String tableAlias);
 // Установить значения параметров фильтра по классу объекта и типу доступа
 public void setFilterParameters(Session session, NativeQuery<?> query, Class<?> entityType, Class<?
extends Annotation> permissionType);
 }
```

При конструировании специализированных запросов к БД, фильтры, ограничивающие доступ к данным, необходимо добавлять в запрос в явном виде. Условия фильтров формируются аналогично приведенному выше описанию. Значения параметров применяемых фильтров необходимо получать при помощи метода *getAttributeValue*. Примеры использования приведены ниже:

```
PermissionProvider permissionProvider =
PermissionProviderFactory.getInstance().getPermissionProvider();

NativeQuery<AccountEntity> query = session.createNativeQuery(
   "select * from t_Account with(index=t_AccountFk2) " +
   "where f_ClientID = :ClientID and f_IsActive != 0 and f_Name not like 'Счет учета%' and " +
   "f_BranchID in(:BranchID) and f_AreaID in(:AreaID) and f_FundID in(:CurrencyID)",
   AccountEntity.class);
query.setParameter("ClientID", clientID);
query.setParameter("BranchID", permissionProvider.getAttributeValue(session, "BranchPermission",
"BranchID"));
query.setParameter("AreaID", permissionProvider.getAttributeValue(session, "AccountPermission",
"AreaID"));
query.setParameter("CurrencyID", permissionProvider.getAttributeValue(session,
"AccountPermission", "CurrencyID"));

List<AccountEntity> result = query.getResultList();
```

```
PermissionProvider permissionProvider =
PermissionProviderFactory.getInstance().getPermissionProvider();

NativeQuery<AccountEntity> query = session.createNativeQuery(
   "select * from t_Account with(index=t_AccountFk2) " +
   "where f_ClientID = :ClientID and f_IsActive != 0 and f_Name not like 'Счет учета%' and " +
   permissionProvider.generatePermissionFilters(session, AccountEntity.class,
AccessPermissions.class, null),
   AccountEntity.class);
query.setParameter("ClientID", clientID);
permissionProvider.setFilterParameters(session, query, AccountEntity.class,
AccessPermissions.class);

List<AccountEntity> result = query.getResultList();
```