onnet

2021
VIE

Odoo domain filter & role-base authorization

www.on.net.vn
www.arrowhitech.com

onnet
BY AHT TECH JSC

Table of contents

- Prefix notation
- Odoo domain
- Odoo role-based authorization
- Practice

onnet
BY AHT TECH JSC

Prefix notation (polish notation)

Prefix notation is a notation form for expressing arithmetic, logic and algebraic equations. **Prefix notation** was invented in **1924** by **Jan Lukasiewicz**, a Polish logician and philosopher, in order to simplify sentential logic.

The idea is simply to have a **parenthesis-free notation** that makes each equation **shorter** and **easier to parse** in terms of defining the evaluation priority of the operators.

Its most basic distinguishing feature is that **operators are placed on the left of their operands**. If the operator has a defined fixed number of operands, the syntax does **not** require brackets or parenthesis to lessen ambiguity.

Prefix notation (polish notation)

Example:

Infix notation with parenthesis : (3 + 2) * (5 – 1)

Prefix notation: * + 3 2 – 5 1

When used as the syntax for programming language interpreters, **prefix notation** can be readily parsed into an **abstract syntax tree** and **stored in a stack**. In traditional **infix notation** with brackets, the equation has to be parsed, the brackets removed, and the operator and operands repositioned. This is not the case with **prefix notation**.

Prefix notation (polish notation)

Steps to convert **infix** expression to **prefix**:

1. First, **reverse** the given infix expression. While reversing, **change open parenthesis to closed parenthesis** and **vice versa**.
2. Scan the characters one by one.
3. If the character is an **operand**, copy it to the **prefix notation output**.
4. If the character is a **opening parenthesis**, then push it to the **stack**.
5. If the character is an **closing parenthesis**, **pop the elements in the stack** until we find the corresponding **opening parenthesis**.
6. If the character scanned is an **operator**:

   - If the **operator** has **precedence greater than** or **equal** to **the top** of the stack or **the stack is empty**, push the **operator** to the stack.

   - If the **operator** has **precedence lesser than** the **top of the stack or stack only have operators left**, pop the operator and output it to the prefix notation output and then check the above condition again with the new top of the stack.

7. After all the characters are scanned, **reverse** the prefix notation output.

Odoo domain

A **domain** is a list of **criteria**, each criterion being **a triple** (either a **list** or a **tuple**) of (**field_name**, **operator**, **value**) where:

- **`field_name` (str)**: a **field name** of the **current model**, or a **relationship traversal** through a Many2one using **dot-notation** e.g. `'street'` or `'partner_id.country'`
- **`operator` (str):** an operator used to compare the **`field_name`** with the value
- **`value` (str):** value to **compare**

Odoo domain

Domain is used in Odoo to **select records** from **a Model** (database table) – in many different places:

- **Windows Action**
- **Form Views** – to select records from a one2many or many2many
- **Record Rules**
- **Filters**

Odoo domain

Valid **operators** are:

- **=** : equals to
- **!=** : not equals to
- **>** : greater than
- **>=** : greater than or equal to
- **<** : less than
- **<=** : less than or equal to
- **=?** : unset or equals to (returns true if value is either None or False, otherwise behaves like =)
- **=like** : matches **field_name** against the value pattern. An **underscore _** in the pattern stands for (matches) any single character; a **percent sign %** matches any string of zero or more characters.
- **like** : matches **field_name** against **the %value% pattern**. Similar to **=like** but wraps value with '%' before matching
- **not like** : doesn't match against **the %value% pattern**
- **Ilike** : case **insensitive** like
- **not ilike** : case **insensitive** not like
- **=ilike :** case **insensitive** =like
- **in** : is equal to any of the items from value, value should be **a list of items**

Odoo domain

Valid **operators** are:

- **not in:** is **unequal** to **all of the items** from value
- **child_of:** is a child (descendant) of a value record. Takes the semantics of the model into account (i.e following the relationship field named by **_parent_name**).
- **parent_of**: is a parent (ascendant) of a value record. Takes the semantics of the model into account (i.e following the relationship field named by **_parent_name**).

Odoo domain

Domain criteria can be combined using **logical operators** in **prefix form**:

- **'&'**: logical AND, default operation to combine criteria following one another. Arity 2 (uses the next 2 criteria or combinations).
- **'|':** logical OR, arity 2.
- **'!':** logical NOT, arity 1.

Odoo role-based authorization

Aside from manually managing access using custom code, Odoo provides two main data-driven mechanisms to manage or restrict access to data.

Both mechanisms are linked to specific users through groups: a user belongs to any number of groups, and security mechanisms are associated to groups, thus applying security mechanisms to users.

class res.groups:

- ***name***: serves as user-readable identification for the group (spells out the role / purpose of the group)
- ***category_id***: The module category, serves to associate groups with an Odoo App (~a set of related business models) and convert them into an exclusive selection in the user form.
- ***implied_ids***: Other groups to set on the user alongside this one. This is a convenience pseudo-inheritance relationship: it's possible to explicitly remove implied groups from a user without removing the implier.
- ***comment***: Additional notes on the group e.g.

Odoo role-based authorization

Access Rights

Grants access to an entire model for a given set of operations. If no access rights matches an operation on a model for a user (through their group), the user doesn't have access.

Access rights are additive, a user's accesses are the union of the accesses they get through all their groups e.g. given a user who is part of group A granting read and create access and a group B granting update access, the user will have all three of create, read, and update.

class ir.model.access

- ***name***: The purpose or role of the group.
- ***model_id***: The model whose access the ACL controls.
- ***group_id***: The res.groups to which the accesses are granted, an empty group_id means the ACL is granted to every user (non-employees e.g. portal or public users).

The perm_method attributes grant the corresponding CRUD access when set, they are all unset by default.

- perm_create
- perm_read
- perm_write
- perm_unlink

Odoo role-based authorization

Access Rules

The domain is a python expression which can use the following variables:

- **time:** Python's time module.
- **user**: The current user, as a singleton recordset.
- **company_id**: The current user's currently selected company as a single company id (not a recordset).
- **company_ids**: All the companies to which the current user has access as a list of company ids (not a recordset), see Security rules for more details.

The perm_method have completely different semantics than for ir.model.access: for rules, they specify which operation the rules applies for. If an operation is not selected, then the rule is not checked for it, as if the rule did not exist.

All operations are selected by default.

- perm_create
- perm_read
- perm_write
- perm_unlink

Odoo role-based authorization

Access Rules
Global rules versus group rules

There is a large difference between global and group rules in how they compose and combine:

- **Global rules intersect**, if two global rules apply then both must be satisfied for the access to be granted, this means adding global rules always restricts access further.
- **Group rules unify,** if two group rules apply then either can be satisfied for the access to be granted. This means adding group rules can expand access, but not beyond the bounds defined by global rules.
- **The global and group rulesets intersect**, which means the first group rule being added to a given global ruleset will restrict access.

NOTE: Creating multiple global rules is risky as it's possible to create non-overlapping rulesets, which will remove all access.

Odoo role-based authorization

Access Rules

Record rules are conditions which must be satisfied in order for an operation to be allowed. Record rules are evaluated record-by-record, following access rights.

Access rules are default-allow: if access rights grant access and no rule applies to the operation and model for the user, the access is granted.

class ir.rule

- **name**: The description of the rule.
- model_id: The model to which the rule applies.
- groups: The res.groups to which access is granted (or not). Multiple groups can be specified. If no group is specified, the rule is global which is treated differently than “group” rules (see below).
- global: Computed on the basis of groups, provides easy access to the global status (or not) of the rule.
- domain_force: A predicate specified as a domain, the rule allows the selected operations if the domain matches the record, and forbids it otherwise.

Practice

Exercise:

Write following condition using Odoo's domain expression:

```
age >= 11 or age <= 18 and gender == 'male'
```

Q&A