

UD06. DESARROLLO DE MÓDULOS. HERENCIA

Sistemas de Gestión Empresarial

2 Curso // CFGS DAM // Informática y Comunicaciones

Profesor: Alfredo Oltra

**Cicles
Formatius**

ÍNDIX

1 HERENCIA DEL MODELO	4
2 HERENCIA DE LAS VISTAS	7
3 BIBLIOGRAFIA	9
4 AUTORES	9

Versión: 240124.1035


Licencia




Reconocimiento – NoComercial – CompartirIgual (by-nc-sa). No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Atención.** Importante prestar atención a esta información.

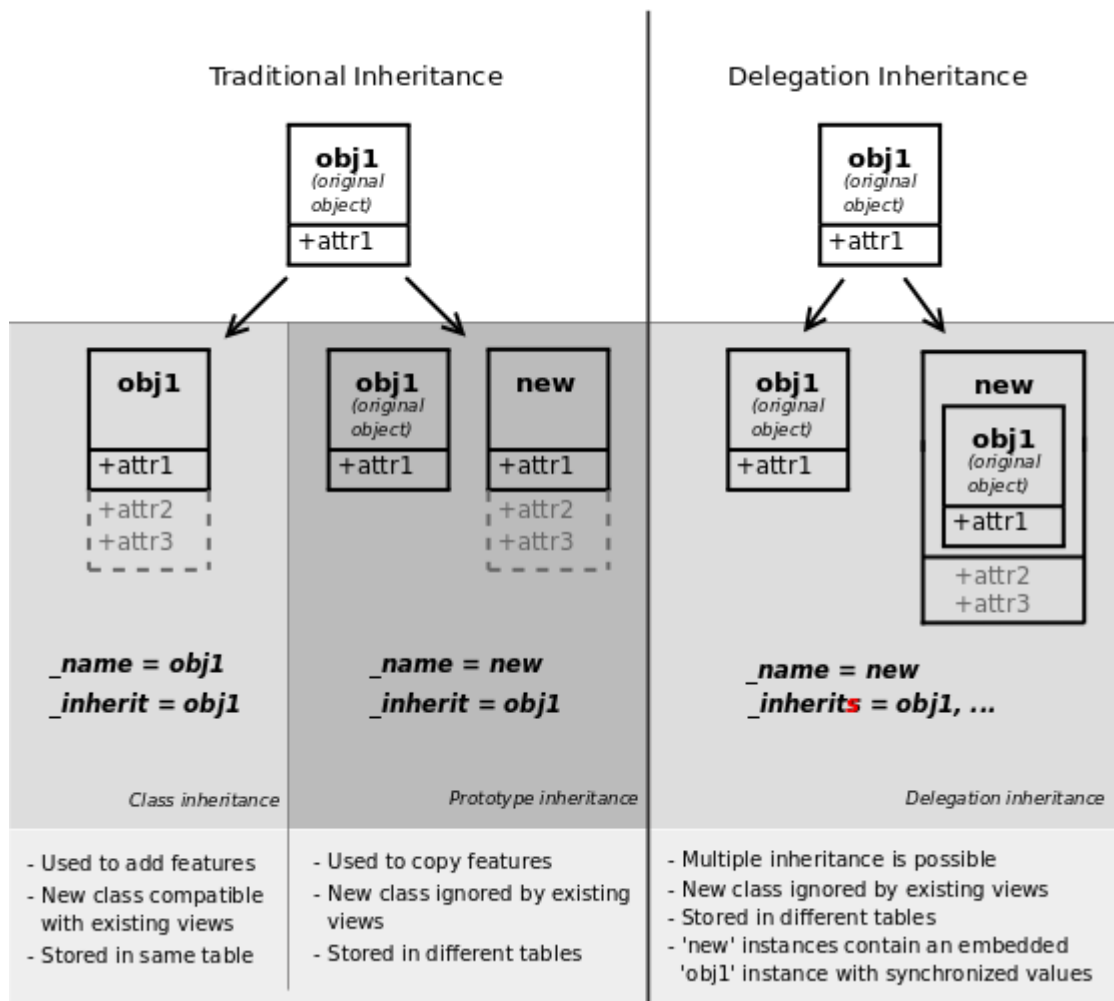
 **Interesante.** Ofrece información sobre algún detalle a tener en cuenta.

1 HERENCIA DEL MODELO

En el caso de la herencia en el modelo, el ORM permite 3 tipos: de clase, por prototipo y por delegación:

De clase	<p>Herencia simple.</p> <p>La clase original queda ampliada por la nueva clase.</p> <p>Añade nuevas funcionalidades (atributos y / o métodos) en la clase original.</p> <p>Las vistas definidas sobre la clase original continúan funcionando.</p> <p>Permite sobrescribir métodos de la clase original.</p> <p>En PostgreSQL, continúa mapeada en la misma tabla que la clase original, ampliada con los nuevos atributos que pueda incorporar.</p>	<p>Se utiliza el atributo <code>_inherit</code> en la definición de la nueva clase Python:</p> <pre><code>_inherit = obj</code></pre> <p>El nombre de la nueva clase debe seguir siendo el mismo que el de la clase original:</p> <pre><code>_name = obj</code></pre>
Por prototipo	<p>Herencia simple.</p> <p>Aprovecha la definición de la clase original (como si fuera un prototipo).</p> <p>La clase original continúa existiendo.</p> <p>Añade nuevas funcionalidades (atributos y / o métodos) a las aportadas por la clase original.</p> <p>Las vistas definidas sobre la clase original no existen (hay que diseñar de nuevo).</p> <p>Permite sobrescribir métodos de la clase original.</p> <p>En PostgreSQL, queda mapeada en una nueva tabla.</p>	<p>Se utiliza el atributo <code>_inherit</code> en la definición de la nueva clase Python:</p> <pre><code>_inherit = obj</code></pre> <p>Hay que indicar el nombre de la nueva clase: <code>_name = nuevo_nombre</code></p>
Por delegación	<p>Herencia simple o múltiple.</p> <p>La nueva clase delega ciertos funcionamientos a otras clases que incorpora en su interior.</p> <p>Los recursos de la nueva clase contienen un recurso de cada clase de la que derivan.</p> <p>Las clases base continúan existiendo.</p> <p>Añadir las funcionalidades propias (atributos y / o métodos) que corresponda.</p> <p>Las vistas definidas sobre las clases bases no existen en la nueva clase.</p> <p>En PostgreSQL, queda mapeada en diferentes tablas:</p>	<p>Se utiliza el atributo <code>_inherits</code> (¡ojo! no <code>_inherit</code>) en la definición de la nueva clase Python: <code>_inherits = obj</code></p> <p>Hay que indicar el nombre de la nueva clase: <code>_name = nuevo_nombre</code></p>

una tabla para los atributos propios, mientras que los recursos de las clases derivadas residen en las tablas correspondientes a dichas clases.



¿Cuándo hay que usar cada tipo de herencia?

- *Odoo* es un programa que ya existe, por tanto, a la hora de programar es diferente a cuando lo hacemos desde 0 aunque usemos un framework. Por lo general no necesitamos crear cosas totalmente nuevas, tan solo ampliar algunas funcionalidades de *Odoo*. Por tanto, la herencia más utilizada es la herencia de clase. Esta amplía una clase existente, pero esta clase sigue funcionando como antes y todas las vista y relaciones permanecen.
- Usaremos la herencia por prototipo cuando queremos hacer algo más parecido a la herencia de los lenguajes de programación. Esta no modifica el original, pero obliga a crear las vistas y las relaciones desde cero.
- La herencia por delegación sirve para aprovechar los *fields* y funciones de otros modelos en los nuestros. Cuando creamos un registro de un modelo heredado de

esta manera, se crea también en el modelo padre un registro al que está relacionado con un *Many2one*.

- El funcionamiento es parecido a poner manualmente ese *Many2one* y todos los *fields* como *related*.
- Un ejemplo fácil de entender es el caso entre *product.template* y *product.product*, que hereda por delegación del primero. Con esta estructura, se puede hacer un producto base y luego con *product.product* se puede hacer un producto para cada talla y color, por ejemplo.

Veamos un ejemplo de cada tipo de herencia:

```
class res_alarm(Model.model): # Clase padre
    _name = 'res.alarm'

class res_partner(models.model): # De clase
    _name = 'res.partner'
    _inherit = 'res.partner'
    debit_limit = fields.float('Payable limit')
    ...
class calendar_alarm(Model.model): # Por prototipo
    _name = 'calendar.alarm' # cambio el name con respecto al padre
    _inherit = 'res.alarm'
    ...
class calendar_alarm(Model.model): # Por delegación
    _name = 'calendar.alarm'
    _inherits = {'res.alarm': 'alarm_id'}
    res_alarm_id = fields.Many2one('res.alarm')
    ...
```

2 HERENCIA DE LAS VISTAS

Si se hace herencia de clase y se añaden *fields* que queremos ver en pantalla, hay que ampliar la vista existente del modelo padre. Para ello usaremos un *record* en XML con una sintaxis especial. Lo primero es añadir esta etiqueta:

```
<field name="inherit_id" ref="modulo.id_xml_vista_padre"/>
```

Después, en el *<arch>* no hay que declarar una vista completa, sino una etiqueta que ya exista en la vista padre y qué hacer con esa etiqueta.

Lo que se puede hacer es:

- **inside (por defecto):** los valores se añaden *dentro* de la etiqueta.
- **after:** añade el contenido después de la etiqueta.
- **before:** añade el contenido antes de la etiqueta.
- **replace:** reemplaza el contenido de la etiqueta.
- **attributes:** modifica los atributos.

Veamos algunos ejemplos:

```
<field name="arch" type="xml">
  <field name="campo1" position="after">
    <field name="nuevo_campo1"/>
  </field>
  <field name="campo2" position="replace">
    <field name="nuevo_campo2"/>
  </field>
  <field name="camp03" position="before">
    <field name="nuevo_campo3"/>
  </field>
  <xpath expr="//field[@name='order_line']/tree/field[@name='price_unit']"
position="after">
    <field name="nuevo_campo4"/>
  </xpath>
  <xpath expr="//form/*" position="before">
    <header>
      <field name="status" widget="statusbar"/>
    </header>
  </xpath>
</field>
```

Como se puede ver en el ejemplo, se puede usar la etiqueta *<xpath>* para encontrar etiquetas más difíciles de referenciar o que estén repetidas.

Es posible que tengamos una herencia de clase en el modelo, pero queramos acceder desde dos menús diferentes a la vista original y a la heredada. Para ello podemos especificar para cada *action* las vistas a las que está asociado.

Odoo cuando va a mostrar algo que le indica un *action*, busca la vista que le corresponde. En caso de no encontrarla, busca la vista de ese modelo con más prioridad. Para ello, dentro de la *action* podemos definir el listado de vista que *Odoo* consultará para elegir la que tiene más prioridad:

```
<field name="view_ids" eval="[(5, 0, 0),(0, 0, {'view_mode': 'tree', 'view_id':  
ref('tree_external_id')})],(0, 0, {'view_mode': 'form', 'view_id':  
ref('form_external_id')})],]" />
```

Esta opción es funcional pero no es la más recomendada. En estos casos es mejor definir correctamente cada uno de los pasos que debe seguir la acción creando un registro del modelo *ir.actions.act_window.view*.

```
<record id="action_window_view" model="ir.actions.act_window.view">  
  <field name="sequence" eval="1" />  
  <field name="view_mode">form</field>  
  <field name="view_id" ref="my_module.my_form_view"/>  
  <field name="act_window_id" ref="my_module.my_action_window"/>  
</record>
```

De esta manera la acción `my_module.my_action_window` mostrará la vista `my_module.my_form_view`.

3 BIBLIOGRAFIA

1. Documentación de Odoo

4 AUTORES

A continuación ofrecemos en orden alfabético (por apellido) el listado de autores que han hecho aportaciones a este documento.

- Jose Castillo Aliaga
- Sergi García Barea
- Alfredo Oltra