本章笔记将学习如何建设用户图像交互界面.本章节将会涉及到各种不同的视图和部件类型.理解什么是**上下文(context)和值域(domain).**并学习如何合理的利用他们创造更好的用户体验

我们将继续在todo\_ui模块的基础上操作.它的模型层已经全部完成,现在需要的是它的视图层

**使用XML文件定义用户界面**

用户界面的每一个部分都是作为记录存储在数据库中的,就像普通数据那样.模块将XML文件的中的数据加载到数据库中形成UI元素

这就意味着我们的todo\_ui模块中必须创建新的xml文件.我们首先预加载下面的文件路径到\_\_manifest\_\_.py文件的data属性中

'data': ['security/ir.model.access.csv',  
 'views/todo\_view.xml',  
 'views/todo\_menu.xml',],

**注意**

数据文件会按照开发人员编写的顺序加载.这个顺序非常重要,因为系统只允许引用已定义的xml数据id.例:有配置文件A,B B文件中含有对A文件中某个数据的引用,那么文件加载的顺序为A,B 如果为B,A那么数据将会无法被找到

在views目录中创建todo\_view.xml和todo\_menu.xml

在第三章中,我们的应用已经有了一个最基本的菜单,但是我们现在需要改进它.所以我们将会添加新的菜单目录以及对应的窗口操作.

**窗口操作**

窗口操作是浏览器端的用户界面.当以一个用户点击一个菜单元素或者一个打开表单的按钮时,窗口操作会告诉服务器接下来要做出什么反应

我们先开始创建一个菜单的窗口操作,来打开todo tasks 和 stages视图

在todo\_menu.xml中添加如下代码:

<act\_window id="action\_todo\_stage"  
 name="To-Do Task Stages"  
 res\_model="todo.task.stage"  
 view\_mode="tree,form,calender"  
 target="current"  
 context="{'default\_state':'Open'}"  
 domain="[]"  
 limit="80"/>  
  
<act\_window id="todo\_app.action\_todo\_task"  
 name="To-Do Tasks"  
 res\_model="todo.task"  
 view\_mode="tree,form,calender"  
 target="current"  
 context="{'search\_default\_filter\_my\_task':True}"/>  
  
<!--src\_model ：指定了在哪个模型中放置More菜单-->  
<act\_window id="action\_todo\_task\_stage"  
 name="To-Do Task Stages"  
 res\_model="todo.task.stage"  
 src\_model="todo.task"  
 multi="False"/>

所有的窗口动作存储在ir,actions.act\_window模型中,可以使用<act\_window>标签简写.这一点我们之前的笔记中有过介绍

第一个动作是打开task stages模型,这个动作只使用了最基本的属性

第二个动作使用了在todo\_app模块中已经定义的窗口动作id并重写了原来的动作.这个动作中包含了最常用的窗口动作属性:

1. Name 通过串口动作打开的视图标题
2. Res\_model 目标模型的唯一标识
3. view\_model 可以使用的视图类型,顺序是有相关的,列表的第一个视图类型就是默认打开的视图类型
4. Target 如果这个属性的值为new 那么该动作会重新打开一个对话窗口.默认值为current,系统则不会开新窗口
5. Context 这个属性设置目标视图的上下文,可以设置字段的默认值,或者激活过滤器.我们将会在本章的后面详细解释
6. Domain 过滤器值域
7. Limit 每个页面显示的记录条数限制

上面的代码还包含了其他的视图类型例如:日历,甘特,和图表.一旦这些改动被安装,相应的按钮也会出现在列表的右上角.然而只有创建对应的视图记录后,这些按键才会有效

第三个窗口动作示范了如何在视图顶部的**More按钮**下添加选项,一下是经常使用的动作属性

1. Src\_model

**菜单项目**

菜单项目都存储在ir.ui.menu模型中.而且可以通过设置中的Menu Items查看里面的所有元素

在todo\_app这个模块中,我们创建了一个能够打开todo\_app的顶部菜单.现在我们希望为其增加一个二级菜单和其他菜单选项.

要达成这个目的,我们将会添加一个新的顶部菜单,向todo\_menu.xml中添加以下代码:

<!--重写todo.task模型的顶部菜单-->  
<menuitem id="todo\_app.menu\_todo\_task" name="To-Do"/>  
<!--  
 创建顶部菜单的子菜单  
 打开todo-task  
-->  
<menuitem id="menu\_todo\_task\_view"  
 name="Tasks"  
 parent="todo\_app.menu\_todo\_task"  
 sequence="10"  
 action="todo\_app.action\_todo\_task"/>  
  
<menuitem id="menu\_todo\_config"  
 name="Configuration"  
 parent="todo\_app.menu\_todo\_task"  
 sequence="100"  
 groups="base.group\_system"/>  
  
<menuitem id="menu\_todo\_task\_stage"  
 name="Stages"  
 parent="menu\_todo\_config"  
 sequence="10"  
 action="action\_todo\_stage"/>

以上代码中的第一条记录是顶部菜单内容,只有一个name属性.但是它会作为接下来两条记录的父记录

要注意到第一条记录沿用了todo\_app.menu\_todo\_task这个XMLID.这就表示本条记录江将会重写原来在todo\_app模块中定义的菜单项目.本条记录并没有绑定任何动作,这是因为我们之后将会添加子菜单项目,而打开Task视图的功能会绑定在子菜单上

第二条菜单项目是顶部菜单下的二级菜单,重要属性为parent=””.通过设置action=””绑定窗口动作.它负责打开Task视图窗口.在实例中,它绑定的是在todo\_app中设定好的窗口动作

第三条菜单项目是为我们的应用添加设置功能.我们希望这个功能之能被超级管理员使用.所以我们使用**group**属性将其设置为只能被系统管理员发现

第四条菜单项目是在设置菜单下添加Stages的子菜单.我们将使用这个菜单维护之后将被加入的KanBan功能来管理Stages

这个时候,如果我们尝试去升级模块一定会遇到错误,那是因为我们还有没定义action属性的XMLID,我们将会在下面定义他们

**上下文和值域(Context and Domain)**

之前我们已经多次接触到了上下文和值域,我们已经了解了在窗口动作中可是设置他们,在模型的关系字段中也可以将他们设置为自有属性

**上下文数据(Context Data)**

**上下文(Context)是一个存有会话数据(Session Data)的字典.**它可以使用在客户端的用户界面中,也可以使用在服务端的ORM和业务逻辑中

在客户端,它可以通过一个连接或者按键将信息从一个视图传递到另一个视图,比如记录的ID.或者为下一个视图提供默认值

在服务端,一些记录集的值会依赖由上下文提供的本地设置.特别会影响能过被翻译的字段.上下文同样可以为服务端的代码提供信号.

Web端的初始上下文是这样的: {'lang':'en\_US','tz':'Europe/Brussels','uid':1}

Lang键对应用户语言,tz对应时区,uid对应当前用户id

当在客户端操作时,例如点击按键,选中的记录信息将会被加载入上下文:

1. Active\_id 是选中记录的id
2. Active\_model是当前记录的模型
3. Active\_ids 是选列表试图中批量选择记录的ids

Context也可以为字段提供默认值,或者对视图开启过滤器.如果将user\_id字段的默认值设置为当前会话用户的id我们可以添加一下代码:

{'default\_user\_id':uid}

如果目标视图包含一个名为filter\_my\_task的过滤器,我们可通过以下代码开启:

{'search\_default\_filter\_my\_task':True}

**域表达式(Domain expressions)**

域表达式用于过滤数据.odoo系统将表达式转换为SQL的WHERE语句来实现数据库的查询

当使用窗口动作打开视图时,domain将视图内的所有数据经过过滤后呈现给用户. 例如:只显示当前用户的任务:

domain=[('user\_id','=',uid)]

这里的uid值是由对话上下文(session context)提供的

当使用一个关系字段时,他将会限制字段的选项.domain过滤器还可以使用当前视图其他字段的值.有了它我们可以根据其他字段值来决定本字段可以拥有哪些选项.例如:联系人字段可以由公司字段决定.可是只显示A公司的联系人

Domain是一个含有多个条件的列表,而每一个条件由一个元组组成.格式为:(‘field’,’operator’,value)

最左边的是字段名,可以使用带点号的关系字段

中间的比运算符可以使用:

1. 常见的比较运算符:<,>,<=,>=,=,!=
2. Like 模糊匹配,可以使用通配符,”\_”下划线匹配一个字符,%百分号匹配零或者多个字符,区分大小写
3. Ilike 类似like, 不区分大小写
4. Not like 模糊不匹配
5. In 判断是否包含,后面的值要是列表
6. Not in 判断是否不包含,后面的值要是列表
7. Child of 判断是否为value的子记录

右边的值可以是一个常量,也可以是一个python表达式.表达式如何写依赖于上下文(Context).domain有两个可能赋值的上下文(Context):客户端或服务端

对于字段domain和窗口动作来说,赋值实在客户端进行的.这里的赋值上下文包括当前视图的字段,带点符号的表达式不能够使用.会话上下文(Session Context)的值,例如uid或者active\_id同样可以被使用.python模块中的datetime和time变量也可以被使用,context\_today()函数返回客户端的当前日期.

用于访问规则的domain的python代码是在服务端赋值审核的.赋值上下文同样可以访问当前数据的所有字段,带点符号的表达式也可以使用.并且当前会话的用户诗句也是可以访问的.**服务端的user.id等同于在客户端使用uid**

**Domian中可以同时存在多个条件,条件之间也可以使用逻辑运算**

**常见的逻辑运算符:**

1. |(OR) 管道符表示或
2. &(AND) 与
3. !(NO) 非

例如:找出所有不属于当前用户的记录

['!',('res\_user\_id','=',uid)]

**AND 和 OR会影响接下来的两个条件**

例如: 找出当前用户或者没有负责人的所有任务

['|',('res\_user\_id','=',uid),('res\_user\_id','=',False)]

下面是一个更复杂的,使用在服务端的记录规则:

['|',('message\_follower\_ids','in',[user.partner\_id.id]),  
 '|',('res\_user\_id','=',user.id),('res\_user\_id','=',False)]

上面的domain可以简化为[|,A,|,B,C],计算顺序为:

1. |, B, C 🡪 D
2. |, A, D

**表单视图**

在之前的笔记中我们已经学习了,表单视图的布局可以简单也可以复杂

我们将会设计一个商业文档类型的视图,同时学习如何使用元素和部件.我们将通过继承扩展之前todo\_app中的视图.但是为了完全学习,我们将完全创建新视图,并覆盖之前的旧视图.

**处理相同种类的多个视图**

在同一个模型中可以包含同一种类的多个视图.通过XMLID,一个窗口动作可以对应一个指定的视图.所以我们就有了这样的灵活性:两个不同的菜单开打同一模型中的不同视图.这就是通过在窗口动作标签中添加view\_id属性完成的.例如:我们可以将视图view\_form\_todo\_task\_ui绑定到窗口动作:todo\_app.action\_todo\_task

**业务文档视图**

业务应用通常是用于记录产品库存,会计开票等等用于实际业务的系统.绝大多数的数据都可以用纸质文档表达.为了提供一个更好的用户体验,表单视图可以包含这些文档.例如,在我们的应用中,我们应该把To-Do Task看作成一个简单纸质表单可以完整容纳的.我们将以此为基准设计一个表单视图.

要创建一个业务文档视图类型的配置文件,我们将在views/todo\_views.xml文件中添加以下代码:

<?xml version="1.0" encoding="UTF-8" ?>  
<odoo>  
 <record id="view\_form\_todo\_task\_ui" model="ir.ui.view">  
 <field name="model">todo.task</field>  
 <field name="=priority">15</field>  
 <field name="arch" type="xml">  
 <form>  
 <!--表头,可以存放按键,工作流状态等-->  
 <header>  
 </header>  
 <!--表单内容-->  
 <sheet>  
 </sheet>  
 <!--表单下方的社交模块-->  
 <div class="oe\_chatter">  
 <field name="message\_follower\_ids" widget="mail\_followers"/>  
 <field name="message\_ids" widget="mail\_thread"/>  
 </div>  
 </form>  
 </field>  
 </record>  
</odoo>

视图的名称是可选属性,如果没有设置系统就会自动生成.为了简化,我们省略了<field name=”name”></field>标签,选择让系统自动生成

我们可以发现,业务文档视图通常含有**三个主要区域:表头(header),表单(sheet)用于存放主要内容,和表单下方的历史纪录与社交区域**

历史纪录和社交区域是通过mail模块添加的.在第三章笔记中,我们已经详细学习了如何嵌入这一功能

**头部标签(Header)**

表单的头部通常的功能是显示当前记录的生命周期阶段或者是动作按钮

这些按钮和通常的不同,最关键的步骤就是高亮他们,使用属性class=”oe\_highlight”

**文档的生命周期使用状态条小部件来表达当前文档的状态.通常使用名state的选择字段或者stage的多对一字段.**这两个字段存在于odoo很多的模块中.

Stage是一个引用辅助模型的多对一字段来表达过程中的各个阶段.它可以被终端用户动态调整来配合指定的业务逻辑,并且它可以完美契合Kanban功能

State 是一个含有各个阶段的可选择列表,例:New,In progress Done.它无法被终端用户编辑,但是由于它是静态的,所以在业务逻辑中使用更简单.

实际上,stages要比states更晚一些进入系统.两种方式可以共存.但是目前的趋势是odoo核心就是要stages替换states.但是states依旧能提供一些stages没有的功能

通过将stages映射入states可以获得两全其美的效果.这就是我们在之前章节中做的,通过将一个state字段添加入Stages模块中,并且通过计算可以使该字段在To-do Task文档中访问

在todo\_view.xml文件中,我们可以在header标签中添加一个状态栏:

<header>  
 <field name="stage\_state" invisible="True"/>  
  
 <button name="do\_toggle\_done"  
 type="object"  
 string="Toggle Done"  
 attrs="{'invisible':[('stage\_state','in',['Draft'])]}"  
 class="oe\_highlight"/>  
  
 <field name="stage\_id"  
 widget="statusbar"  
 clickable="True"  
 options="{'fold\_field':'fold'}"/>  
</header>

在这里我们将stage\_state字段设置为不可见.我们需要这样设置来强制客户端将该数据内容发送给服务端 否则该字段的值无法在表达式中使用

**注意**

在开发过程中非常重要是的:如果要使用任何字段的值,不论是在domain或attr表达式中,必须提前被加载入视图.

下面是一个位于状态栏的按键,使用户可以来回切换Task的Done标签

状态栏中的按键会根据当前文档所处的生命周期来决定是否显示

我们使用attr属性设置当文档处于draft状态时,按键为不可见.

这些可视化的特性同样也可以在其他视图元素中使用,比如字段.我们将在本章后面更深入了解他们

可点击属性允许用户通过点击状态栏目来改变当前文档的阶段(Stage).通常我们会这样设定,但是有些情况不会,例如我们想提高对工作流的可控性.或者要求用户只能通过特定的按键才能使工作流进入下一阶段.这样我们就可以在两个阶段中添加审批过程

在options属性中,我们可以设置详细:

1. Fold\_field,当使用stages的时候,字段名需要折叠显示
2. Statusbar\_visible,当使用states的时候,设定有哪些阶段是要一直可见的.

**表单内容(The sheet)**

Sheet是整个表单视图的主要区域,用于存放数据.它被设计成类似纸质文档的样式.odoo也将记录称之为文档(documents).通常,一个文档表单的结构包含以下元素:

1. 文档的标题和副标题
2. 右上角的按键盒(button-box)
3. 其他文档头标题
4. 分页表单(Notebook)

我们将会逐一学习

**标题和副标题**

在<group>以外的字段不会自动呈现标签.但是我们可以通过使用<label>来迫使它显示字段标签

1. For 属性识别将要显示标签的字段
2. String属性可以重写原字段的string值
3. Class属性可以更改CSS样式
   1. Oe\_edit\_only 当表单为编辑模式下字段才会显示
   2. Oe\_read\_only 当表单为读取模式下字段才会显示

通常的HTML和CSS样式同样可以使标题多样化.最好的结果是标题必须在<div>标签内 class=oe\_title

这里的<sheet>包含了标题和被当作副标题的其他字段

<div class="oe\_title">  
 <label for="name" class="oe\_edit\_only"/>  
 <h1><field name="name"/></h1>  
 <h3>  
 <span class="oe\_read\_only">By </span>  
 <label for="res\_user\_id" class="oe\_edit\_only"/>  
 <field name="res\_user\_id" class="oe\_inline"/>  
 </h3>  
</div>

以上的代码中出现了很多基本的HTML标签:div,span,h1,h3.而<label>能控制HTML元素呈现的时间和位置.

**表单中的组(group)**

表单中的主要内容应当使用<group>来组织。Group标签在主界面中插入两列，而每一列中的字段默认会有label显示

字段和其对应的标签占用两列，所以在group中添加字段会以堆叠的形式呈现。如果我们在一个group中再嵌套两个group,那么我们会得到两列带有字段和标签的内容并排呈现

继续编辑表单视图，我们现在可以向主内容中添加按键盒

<group name="group\_top">  
 <group name="group\_left">  
 <field name="deadline"/>  
 <separator string="Reference"/>  
 <field name="refers\_to"/>  
 </group>  
 <group name="group\_right">  
 <field name="tag\_ids" widget="many2many\_tags"/>  
 </group>  
</group>

为group标签中的name属性赋值是一个好习惯。它将会为以后的视图扩展提供很多便利。同样string属性也可以设置，它将会被当作段落标题

代码中的<separator>标签可以添加组内的分段标签

**提示**

通过使用col和colspan标签,我们能够更好的控制组与组之间的布局

Col属性可以控制<group>的默认列数.默认值为2,但是它可以将其设置为任意数据.最好将其设置为偶数因为每个字段需要占用两列

对于<field>,col属性可以设置其占用的列数,默认为1

**字段(Field)**

视图字段标签拥有许多属性.大多数属性的值源于定义他们的模型,但是这些值同样可以在视图中重写

通用的,不受字段类型的影响的属性有:

1. Name 字段名 在模型文件中定义的变量名
2. String 字段的标签名,可以重写在模型中预定义的string值
3. Help 帮助信息,可以重写在模型中预定义的help值
4. Placeholder 字段中显示的提示信息
5. Widget 允许用户重写默认的字段样式
6. Options JSON数据格式
7. Class CSS样式
8. Nolabel=”True” 避免显示自动生成的字段标签,适用于<group>内的带有<label>的字段
9. Invisible”True” 不显示字段内容.但是服务器还是会发送字段内容,并且表内其他字段可以引用
10. Readonly=”True” 字段只读
11. Required=”True” 字段是必填字段

只用于特殊字段的属性:

1. Password=True 只能用于text字段,输入内容会全部变为星号
2. Filename 只能用于二进制字段
3. Mode 只能用于一对多字段.用于设置数据的默认视图类型.默认是列表类型,但是也可以设置为表单,kanban或者图表
4. Sum,avg 在列表视图中的数字类型字段,会在视图尾部显示总数或平均值

对于所有的布尔类型属性,True和1对等, False和0对等.

**字段标签**

<label>可以更好的控制字段的显示标签.其中一个实例就是让标签只在编辑模式中显示,之前的代码已经有提示

**关系字段**

对于关系字段,我们可以进一步控制用户的操作.默认情况下,用户可以通过关系字段快速创建新值,和打开对应字段的表单.这种功能可以通过一下代码关闭:

options={'no\_open':True,'no\_create':True}

context和domain同样对关系字段十分有用.context可以设定关系字段的默认值,domain可以过滤候选记录的.一个常见的例子就是A字段的可选记录依赖于B字段的值.domain可以在模型中定义,也可以在视图中重写

**字段工具**

每一种类型的字段,都以其默认的样式呈现在表单上.但是系统还提供其他样式可供选择.

对于text类型字段,可以选用下列样式:

1. Email 确定e-mail是否可用
2. url 确定是否是合法的url
3. HTML

对于数字型类型,可以选用下列样式:

1. Handle 为列表视图中的序列字段设计,允许用户按照自己的习惯排列
2. Float\_time 将一个浮点型字段转换为时间

对于关系型和选择字段,我们可以选用下列样式:

1. Many2many\_tags 像按钮类标签一样显示值列表
2. Selection 多对一字段使用选择字段样式
3. Radio 列表字段使用单选按钮选择
4. Kanban\_state\_selection Kanban的状态显示为指示灯.通常状态下为灰色,已完成的为绿色,其他的状态都是红色
5. Priority 选择字段的优先级

**按键(Buttons)**

按键标签可用的属性有:

1. Icon 修改按键图标,但是只能使用系统中定义的图标
2. String 按键名
3. Type 指代按键将要执行的动作
   1. Workflow 按键触发工作流
   2. Object 按键调用python函数
   3. Action 按键触发窗口动作
4. name 根据按键要执行的动作类型,识别对应的动作id.例如:工作流注册id,模型中的函数名,或者一个窗口动作的数据库id.使用%(‘XMLID’)可以将XMLID转换成该记录在数据库中的id
5. args 当字段种类为object,为调用的python函数传入参数
6. context
7. confirm
8. special=cancel

**快捷按键**

当设计表单结构时,我们嵌入了一个快捷按键区域.现在我们将向里面添加按键

对于我们的应用来说,我们设计一个可以显示当前任务的所有者的所有任务,并且点击按键会导向那些任务的列表

首先我们需要添加对应的计算字段,在todo\_model.py中的TodoTask类中添加以下代码:

def compute\_user\_todo\_count(self):  
 for task in self:  
 task.user\_todo\_count = task.search\_count([('res\_user\_id','=',task.res\_user\_id.id)])  
  
user\_todo\_count = fields.Integer('User To-Do Count',compute='compute\_user\_todo\_count')

接下来我们将向按键盒中添加按键.在oe\_title div下面:

<!--button-box-->  
<div name="buttons" class="oe\_right\_oe\_button\_box">  
 <button class="oe\_stat\_button"  
 type="action"  
 icon="fa-tasks"  
 name="%(action\_todo\_task\_button)d"  
 context="{'default\_res\_user\_id':res\_user\_id}"  
 help="All to-dos for this user">  
 <field string="To-Dos" name="user\_todo\_count" widget="statinfo"/>  
 </button>  
</div>

这个按键将显示当前任务负责人的所有任务数量.这个数量是通过user\_todo\_count字段计算出的.

以下是快捷按键可以使用的属性:

1. class=”oe\_stat\_button” 修改按键的CSS样式
2. icon 改变按键图标
3. type&name 按键的类型和对应的动作触发器.通常来说,快捷按键的类型为action.对于一个窗口动作来说,name是动作的XMLID.它应该是一个数据库的id,所以我们将使用表达式将XMLID转换成数据库ID.
4. String 修改按键的标签,快捷按键一般不需要
5. Context 设定目标视图的默认值.这个使用在通过按键创建新记录上
6. Help 鼠标悬停在按键上的帮助信息

按键本事也是一个容器,同样有字段显示.这些字段通常使用statinfo样式.这个字段应该是所属模型中的一个计算字段.按键中还可以使用静态文本,例如:

<div>User’s To-dos</div>

当点击按钮时,我们希望看到一个只显示负责人任务的列表.这是通过action\_todo\_task\_button动作达成,但是现在并没有写入.但是它需要知道当前的负责人,并在过滤器中使用.我们需要使用按键的上下文(Context)属性来存储这个值.

这个动作必须在表单之前被定义,所以我们必须将其放在XML文件的顶部:

<act\_window id="action\_todo\_task\_button"  
 name="To-Do Tasks"  
 res\_model="todo.task"  
 view\_mode="tree,form,calender,graph,pivot"  
 domain="[('res\_user\_id','=',default\_res\_user\_id)]"/>

注意到我们在代码中使用了default\_res\_user\_id,这个特殊的值同样可以在创建新任务时将设置res\_user\_id的默认值

**动态视图(Dynamic Views)**

视图元素同样支持动态属性,它允许视图外观随着某个字段值的改变而变化.我们需要**on change事件**,能够使用户在编辑表单时,改变其他字段的值.或者当某些条件达成时强制改变或不显示某个字段.

**On change events**

**On change**机制可以允许用户在设置某个字段值时,同时改变其他视图的字段值.例如:Product字段的改变可以同时改变Price字段的值

在旧版本中,on change只能在视图层中定义.8.0版本后,是在模型层中定义.并且不需要在视图文件中做任何说明.这是通过创建特殊函数完成的,并且需要在函数上方添加**@api.onchange(‘field1’,’field2’)装饰器**.我们将在第七章中更深的讨论on change机制

**动态变量(Dynamic attributes)**

为了能够立刻对用户的输入作出反应,on change机制通常可以自动重新计算计算字段的值.动态变量包含一些属性来提供特定用户的访问权限:

1. Groups 使特定的组用户才能访问字段.参数为包含多个用户组XMLIDs的列表
2. States 根据记录的状态字段的值来决定其他字段是否可见.参数也为包含多个状态值的列表

除此之外,我们同样在客户端使用表达式来达到同样的隐藏字段的效果.使用**attr属性作为前后端的映射.**

例如:如果需要refers\_to字段在除了draft以外的状态都可见我们可以在XML文件中这样添加代码:

<field name="refers\_to" attrs="{'invisible':state=='draft'}"/>

Invisible属性在所有的元素中都可用,不只是字段.例:我们可以用在XML文件中的<group>标签或<notebook>页面

Attrs也可以设置其他字段属性:readonly 和required.只能用于数据字段,可以使字段只读或者为必填字段.这个功能可以让我们实施一些基本的客户端逻辑.例如:基于他字段的值,使某一个字段变为必填字段.例如基于状态字段

**列表视图**

至此,列表视图只需要简要介绍一下,但是我们仍然会了解其中可用的属性.下面是我们todo模块中的列表视图:

<record id="todo\_app.view\_tree\_todo\_task" model="ir.ui.view">  
 <field name="model">todo.task</field>  
 <field name="arch" type="xml">  
 <tree decoration-muted="is\_done"   
 decoration-bf="stage\_stage=='open'"   
 delete="False">  
 <field name="name"/>  
 <field name="res\_user\_id"/>  
 <field name="is\_done"/>  
 <field name="stage\_state" invisible="1"/>  
 </tree>  
 </field>  
</record>

列表中每一行的字体颜色和大小都可以通过python函数定义

这是通过**decoratuon-NAME**属性设置的.通过对字段值的判定.这里的NAME是个可变的,基于bootstrap样式

**注意**

颜色和字体属性8.0版本可用,9.0版本不可用.10版本必须使用的新的属性

使用表达式的字段必须使用<field>标签,这样web端才能知道需要从服务端获取数据.如果我们不希望该字段呈现给用户,只需要将invisible值设置为1即可

其他列表支持的属性有:

1. Default\_order 可以重写列表列出数据的默认顺序.
2. Create,delete,edit 如果设置为false(\***小写**),将取消列表视图的对应操作
3. Editable 可以使数据直接在列表视图进行编辑

列表视图同样可以包含字段和按键,并且大部分的表单视图属性也可以使用

在列表视图中,数字类型的字段可以在当列的末尾显示摘要值.要开启这样的功能,需要在数字类型字段中添加一下属性:sum,avg,min,max 并且提供一个标签名,例:

<field name="amount" sum="Total Amount"/>

在列表末尾?计算amount字段的所有值得和,并为其添加Total Amount标签

**查找视图**

当输入查询栏目时,可以设置哪些字段是可以被自动查找的.我们同样可以提供预定义的过滤器,以及列表视图中的分组.

下面是todo模块的查找视图代码:

<record id="todo\_app.view\_filter\_todo\_task" model="ir.ui.view">  
 <field name="model">todo.task</field>  
 <field name="arch" type="xml">  
 <search>  
 <field name="name"/>  
 <field name="res\_user\_id"/>  
 <filter name="filter\_not\_done"  
 string="Not Done"  
 domain="[('is\_done','=',False)]"/>  
 <filter name="filter\_done"  
 string="Done"  
 domain="[('is\_done','=',True)]"/>  
  
 <separtor/>  
 <filter name="group\_user"  
 string="By User"  
 context="{'group\_by':'res\_user\_id'}"/>  
 </search>  
 </field>  
</record>

从上面的代码中我们可以看到有两个字段将被查找-name和res\_user\_id.当用户开始在搜索栏中输入时,会出现一个下拉,关于上述两个字段的推荐.如果用户敲击回车,则会返回推荐列表的第一个值.

这里我们已经有了两个预定义的过滤器,过滤已完成或者未完成的任务.这些过滤器可以被分别激活,也可以使用或逻辑连接.在过滤器之间使用<separator/>标签可以使得过滤器之间用与连接.

第三个过滤器使用上下文分组.告诉视图使用res\_user\_id字段分组.相当于数据库中的GROUP\_BY

查找视图中的字段可以包含以下属性:<field>

1. Name 识别要查找的字段
2. String 修改默认标签
3. Operator 修改默认的逻辑运算符
4. Filter\_domain
5. Groups 使查找字段只能被某个工作组访问

查找视图中的过滤器可以包含一下属性:<fiter>

1. Name 过滤器识别id
2. String 过滤器标签,必填
3. Domain 当前过滤器的域表达式
4. Context 内容字典.使用某一字段对数据进行分组
5. Groups 使当前过滤器只能被某个组用户访问

**其他视图种类**

Odoo系统中最常见的视图就是表单和列表试图,也是我们刚刚学习的.除此之外,系统还提供有其他视图.Kanban视图我们会在之后的第八章学习

**日期视图**

正如它的名字,日期视图将记录汇总在一个日历中.todo-task的日历视图代码为:

<record id="view\_calender\_todo\_task" model="ir.ui.view">  
 <field name="name">todo.task</field>  
 <field name="arch" type="xml">  
 <calender date\_start="deadline"  
 color="res\_user\_id"  
 display="[name],Stage[stage\_id]">  
 <!--将在日历上显示的内容-->  
 <field name="name"/>  
 <field name="stage\_id"/>  
 </calender>  
 </field>  
</record>

可用于日历视图中的属性有:

1. Date\_start 开始日期
2. Date\_end 结束日期
3. Date\_delay 事件持续的时间,可以代替date\_end属性
4. All\_day
5. Color 将日历内容按分组颜色.每一个不相同字段都会被分配一个不同的颜色
6. Display 每一个日历元素中显示的内容.
7. Mode 日历的默认格式,可以是day,week或者month

**总结**

在本章中,我们已经学习了如何去建立一个用户界面,并介绍了最常用的几个视图种类.而在下一章中,我们将更多的学习和业务逻辑相关的知识