大多数的odoo模块定义,像用户界面和安全规则,实际上是存储在特定数据库表中的数据.而存放在模块目录中的XML和CSV文件并不在odoo应用运行时使用.而是将他们当中的数据提前录入数据库表中

因此,odoo模块的一个重要部分就是文件数据的**序列化(serializing),**使得文件能延时加载入数据库

模块可以拥有默认和演示两种数据.数据的序列化允许将上述的两种数据存放入模块中. 所以,理解odoo数据的序列化格式非常重要,这有助于我们日后在实施模块的时候导入和导出业务数据

在我们学习实际应用的案例前,我们将会先了解一个在odoo系统数据序列化中非常重要的内容:外部识别id

**理解外部识别ID**

在odoo数据库存储的所有数据都有一个唯一的识别:id字段.它是一系列由数据库自动生成的数字.然而,这种自动生成的id在加载相互关联的数据的时候将是一种挑战:我们如何引用一个数据库还未生成id的记录?

Odoo对这个问题的答案就是:外部标识(external identifier). 通过给将要加载的数据分配字符串格式的命名标识来解决这个问题.一个命名标识可以被其他任意的数据引用.Odoo系统会负责将这些标识映射成数据库分配给他们的id.

**这一机制其实非常简单,odoo存有一个用来映射数据库id和外部标识的表,这个表就是ir.model.data模型 类似数据库中第三范式**

要查看现有的映射,需要

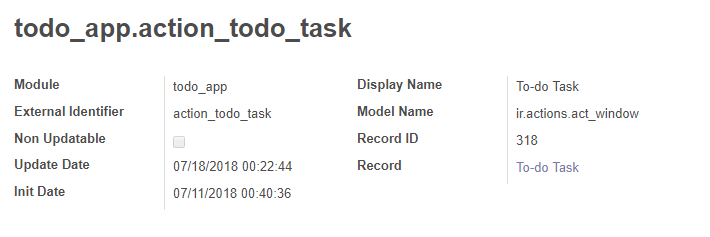
1. 开启开发者模式
2. 依次进入: Settings 🡪 Sequences & Identifiers 🡪 External Identifiers 即可

例如我们在搜索框内输入todo, 系统将会自动为我们列出所有todo模块中的外部标识

可以看到所有的外部标识都包含一个Complete ID字段. 这个字段由模块名字加点加标识名组成.例如:todo\_app.action\_todo\_task

由于Complete ID是全局唯一标识,所以模块名就相当于一个命名空间.这就意味着不同的模块中允许含有相同名称的标识,这样我们就不必担心不同模块之间外部标识冲突的问题

点击可进入详细表单,我们可以发现在todo\_app模块中的action\_todo\_task映射成为在ir.action.act\_window中的一个Record ID 本例中为318



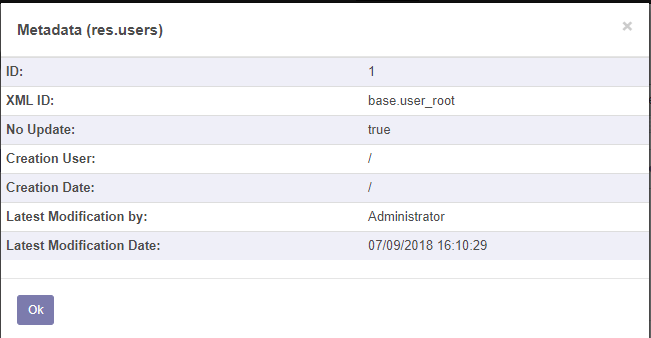
除了可以更加简便的引用其他记录,外部标识还可以在数据导入时避免重复.如果外部标识已经存在,那么原来的数据将会被更新,而不是创建一个新的记录.这就是为什么在随后的模块升级中,之前已经加载的记录会被升级而不重复的原因.

**查找外部标识**

当准备配置模块的配置和演示数据时,我们将会很频繁的查找现有的依赖的外部标识.

我们当然可以使用之前介绍的查找方法.但是**开发者菜单(Developer Menu)**可以提供更方便的方法.之前在第一章笔记中我们已经介绍过如何开启开发模式.

为了查找一条数据记录的外部标识,在相关的表单视图中,在开发工具中打开View Metadata 选项.它将会打开一个含有纪录id和外部id的窗口



**导入和导出数据**

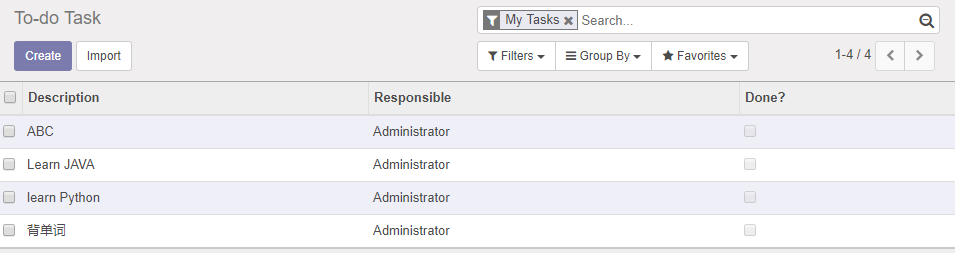
我们将开始学习研究如何从odoo的用户界面导入和导出数据. 而且从现在开始,我们将会学习更多的技术性的细节

**导出数据**

数据导出功能是任何列表视图自带的特性.要使用这一特性,我们需要:

1. 选择需要导出的数据条目
2. 在出现的动作条目中点击导出

这里我们将会联系到处todo模块中的数据

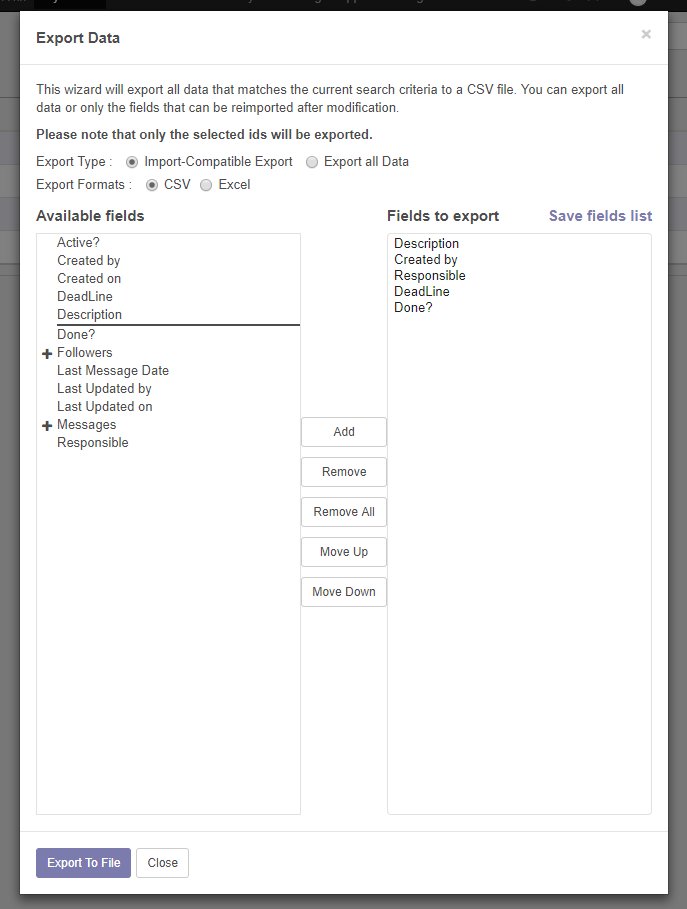


**小知识**

在旧的odoo版本中,只有显示在列表中的字段才可以被导出.但是自从odoo9版本,不显示在列表中的字段也可以有选择性的导出.

导出的按键会打开一个新的窗口让用户选择要导出的字段.而导出模式选择中的Import-Compatible Export模式将允许用户将导出后的CSV文件再重新导入到系统中.而我们将使用这一功能

窗口见下图:



Odoo系统支持两种导出文件格式:CSV和EXCEL. 这里更推荐CSV格式,因为可以更快的理解导出格式的设置.接下来,我们将选择要导出的字段,再点击导出到文件.这时,浏览器将开始自动下载导出文件到文件夹

**\*当导出的文件为CSV格式时, 如果文件中含有中文,则必须将阅读编码改为utf-8 否则会出现乱码.当文件为xlsx格式时则不出现这个问题**

**注意**

Odoo在导出文件时,会自动生成一条id字段.这个字段就是每一条记录的外部标识.如果记录没有被指定标识,那么系统会使用\_\_export\_\_字符串的格式自动生成.新的外部标识只会分配并绑定那些没有外部标识的记录.这也表示序列化导出将不会出现相同的外部标识

**导入数据**

首先我们要确认系统的导入功能是否被开启.但是自从odoo9版本开始,导入功能的是默认开启的.如果没有,在设置中可以开启

导入乱码等,之后需要详细学习

**模块数据**

模块使用数据文件向数据库内加载:配置信息,初始数据和演示数据.这些可以使用CSV或者XML文件实现.CSV文件的使用方式和导入数据的方式相同.当我们在模块中使用时,唯一的额外条件是文件名必须和模型中定义的文件名相同

一个非常典型的例子就是访问安全,需要加载入ir.model.acess模型.我们之前已经接触过,这是由CSV文件实现,并且必须命名为ir.model.acess.csv

**XML数据文件**

CSV提供了一个简单而紧凑的方法去序列化数据,XML文件则功能更强,而且在加载过程中更容易控制

在之前的笔记中我们已经使用过XML数据文件了.用户界面,比如表单和列表,实际上是系统模型中存储的数据.XML正是这些数据加载入模块的媒介

为了展示,我们将创建为todo\_user模块创建一个数据文件:todo\_data\_xml:

<?xml version="1.0" encoding="UTF-8" ?>  
<odoo>  
 <record id="todo\_task\_c" model="todo.task">  
 <field name="name">Reinstall Odoo</field>  
 <field name="res\_user\_id" ref="base.user\_root"/>  
 <field name="deadline">2018-07-15</field>  
 </record>  
</odoo>

这个文件的作用和CSV文件一致

XML文件中一个record标签,代表列表中的一条数据

**注意**

最外部的标签<odoo>必须存在. 而<data>标签虽然支持但是已经变为一个可选的标签.<record>标签含有两个主要属性id 和 model. Id代表本记录的外部标识,model代表本条记录所属的模型.<field>标签对应表中的每一个字段

注意斜杠符号是不可以在name属性中使用的.例如:我们不能使用<field name=”res\_user\_id/id”>. 然而,我们可以使用<field>标签中的ref属性去引用外部标识.我们将在本章笔记后面讨论关系型字段

**数据不更新属性**

当数据被重复加载时,之前已加载的数据将会被重写.所以必须要记住:**模块的升级会重写之前在数据库中做出的任何手动修改.**如果系统中包含订制视图,当模块升级时这些订制将会丢失.正确的步骤是我们在第三章笔记中学习的:使用视图继承特性来实现

这样的重复加载的特性是系统默认的,但是可以被修改.修改后当模块升级时,一些数据文件记录是不会被修改的.通过在<odoo>或<data>标签中追加noupdate=1属性即可.这些记录会在模块安装的时候创建,但是在模块升级的时候将不会生效

这可以确保人工定制的修改不会因为模块升级而改变.这样的方法经常被用在设定数据访问权限中.

在同一个XML文件中可以同时存在多个<data>标签.我们可以利用这一特性区分那些需要/不需要更新的数据

Noupdate属性同时可以在外部标识列表中修改,位置同之前介绍的外部标识

**建议**

在模块的开发过程中,使用noupdate属性是不方便的,因为做出的修改会被忽略,并且odoo也不会在后面自动更新.解决方法是在开发过程中设置为0,而在开发结束后设置为1

**使用XML定义数据**

每一个<record>标签都含有两个最基本的属性:id和model.一个<field>对应一个字段. 正如之前提到的,id属性对应的数据中的外部标识而model属性是本条数据将被写入的模型.<field>字段赋值的方法有很多种,接下来详细介绍

**设置字段值(<filed>)**

<record>定义了一整条数据,其中嵌套多个<field>标签代表数据中的每一条字段

<field>标签中的name属性的值指向python文件中的变量

属性对应的值写在开闭标签内.对日期格式的数据,将会被系统自动转换,但是对于Boolean类型的变量,任何非空值都会被认定为True.而”0”和”False”被认定为False.

Odoo10判定Boolean类型的方法在10版本中改进:在之前版本中”0”和”False”会被认定为True. 接下来我们会介绍另一种推荐辨别Bool类型的方法:eval

**使用表达式赋值**

另一个更详细的赋值方式是使用eval属性.它将会判定eval中的python代码并通过执行这段代码赋值.这段代码除了包含python代码外,还含有一些其他的特殊标识,我们来认识一下

要处理日期类型的数据, 可以使用下面的几个模块: time,datetime,timedelta和relativedelta.例如,我们将一个日期的值设置为一天前:

<field name=”date” eval=”(datetime.now()+timedelta(-1).strftime(%Y-%M-%D))” />

同样在eval的上下文中可以使用ref()函数 ref函数可以将外部标识翻翻译为对应的数据库ID.这可以为关系字段赋值.例如我们之前设定的res\_user\_id值

<field name="res\_user\_id" eval="ref('base.group\_user')"/>

**为关系字段赋值**

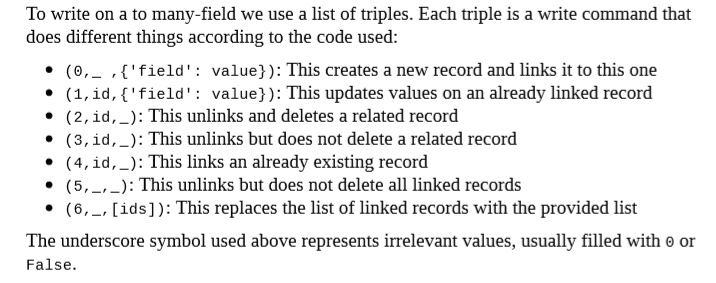
我们刚刚学习了如何给多对一关系型字段赋值,例如res\_user\_id,使用eval属性+ref(0方法,但是还有一种简单的方法

<field>标签还有一个ref属性. 通过外部标识,可以直接多对一字段赋值,我们可以将上面的代码改写为:

<field name="res\_user\_id" ref="base.user\_demo"/>

对于一对多和多对多字段,相关的IDs列表是必须的,同时一个不同的代码格式也是必须的. Odoo提供一种特殊的语法去写这种类型的字段

<field name="tag\_ids" eval="[(6,0,[ref('vehicle\_tag\_leasing'),  
 ref('fleet.vehicle\_tag\_compact'),  
 ref('fleet.vehicle\_tag\_senior')]  
 )]"/>



**常用标签简写**

如果我们回看第二章的笔记,我们会发现在xml配置文件中不只有<record>标签,同样也有<act\_window>和<menuitem>标签

这些就是常用的模型标签的简写方式并且和普通<record>标签一样.如何通过载入数据修改用户界面我们将在第六章的笔记中详细介绍

以下就是一些常用的标签:

1. <act\_window> 🡪窗口动作模型ir.actions.act\_window
2. <menuitem>🡪菜单模型ir.ui.menu
3. <report>🡪报告模型ir.actions.report.xml
4. <templete>QWeb报表模型ir.ui.view
5. <url> URL动作模型ir.actions.act\_url

**其他的XML数据文件动作**

到目前为止,我们已经接触到如何使用XML配置文件添加或更新数据.但是XML文件同时允许执行其他任务.例如:可以删除数据,执行任意的模型方法,和触发工作流事件

**删除记录**

想要删除一条记录,我们要使用<delete>标签,并且要提供要删除的记录id或者一个能够找到记录的查找域(Search domain). 例如:使用一个查找域去删除一条记录:

<delete id="todo\_app.todo\_task\_user\_rule" model="ir.rule"/>

**触发函数和工作流**

XML文件也可以在加载的过程中执行函数.通过<function>标签,可以设置测试数据.例如:CRM应用就是用这种方式设置演示数据

<function model="crm.lead" name="action\_set\_lost"  
 eval="[ref('crm\_case\_7')],  
 ref('crm\_case\_9')],  
 ref('crm\_case\_11')],  
 ref('crm\_case\_12')],{'install\_mode':True}"/>

以上代码会调用crm.lead模型里的action\_set\_lost函数,然后通过eval属性传入两个参数.一个参数是要使用的ID列表,接下来是要使用的上下文

XML数据还可以通过<workflow>标签触发工作流. 例如:工作流可以更改销售订单状态,或者将其转换为发票.销售模块不再使用工作流,但是这个例子仍然可以在测试数据中找到:

<workflow model="sale.order" ref="sale\_order\_4" action="order\_confirm"/>

**总结**

我们已经学习了有关数据系列化所有的核心知识并对XML内容有了一个更好的理解.我们也花了时间理解外部标识和如何通过标签删除或更新数据.我们同时也学习了如何使用CSV文件导入和导出数据.

在接下来的章节中,我们将了解如何更深层的开发odoo模型和如何开发用户界面