一、数据模型

- 数据模型存放在 **models** 目录下，是一个python文件，描述业务对象
- 位于odoo的ORM层，用python来描述数据字段，避免手写sql，并提供可扩展性和安全服务
- **_name** 属性定义模型的名称，必须存在
- 定义模型字段，odoo根据需要保存在数据库中

1 字段类型

常用字段类型:

字段类型	说明	备注
Char	字符型	size限制字符长度
Boolean	布尔型	
Float	浮点型	digits=(9,3) 限制大小和精度

		这里表示数字总长度9，小数部分3，不足补0
Integer	整形	
Date	日期	年月日
Datetime	时间戳	可以使用fields.Date.today获取当前日期
Text	文本	可用于多行文本
Html	文本	类似Text，自带html编辑器，自定解析
Selection	下拉列表	枚举类型
Monetary	货币数	类似浮点数，带有货币特殊处理
Binary	二进制	可有python的base64编码字符串处理

属性	说明
string	unicode，字段名称
required	bool，True表示NOT NULL
help	unicode，帮助文字，鼠标悬停提示
index	True则添加数据库索引，搜索快，但写操作慢
default	默认值
readonly	True则在用户界面无法编辑，但是可以用API修改
copy	False则使得字段在使用ORM的copy()方法复制时忽略该字段
groups	限制某些字段对一些group的可见性，用户组用,隔开
states	

3 保留字段

odoo在根据字段创建数据表的时候，会额外创建几个保留字段，这些字段由odoo系统管理，不应该由用户写入，但可以读取

保留字段	说明
id	一个数据模型的全局唯一标识符
create_date	数据的创建时间

create_uid	数据的创建用户
write_date	数据的最后修改时间
write_uid	最后修改数据的用户

4 特殊字段

- 在默认情况下，odoo使用 **name** 字段来进行模型的显示和搜索
- 可以使用 **_rec_name** 重新指定存在的字段来替代 **name** 的功能

代码示例

model.py

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api

class TestModule(models.Model):
    _name = 'test_module.model' #模型名称，必须，后面用的着
    _description = u'测试模型'

    name = fields.Char(string='Name', size=50)
    value = fields.Integer(string='Data', default=99)
     description = fields.Text('Description')
```

数据模型添加完成后，需要在目录下的**__init__.py** 文件中import

```
from . import models
```

该数据模型将会保存在数据库的 **test_module_model** 表中

二、字段的计算

1、 计算字段

类似vue的compute，在每次调用时计算得出

```
people_count = fields.Integer(string=u'人数', compute='_get_peop
e_count', store=True)

@api.multi
def _get_people_count(self):
 for model in self:
 model.people_count = len(model.name)
```

- 计算字段不会保存在数据库内，但是可以使用 **store** 参数，设为True，则可以保存在数据库内

2、depends依赖字段

```
@api.multi
@api.depends('value', 'name')
def _get_people_count(self):
 for model in self:
 model.people_count = len(model.value)
```

- **@api.depends** 指定依赖，依赖被修改，则重新触发计算字段更新，可以同时传多个依赖字段。使用依赖，则修改数据；
- 和 **onchange** 不同的是，点击保存后，才会触发；

3、onchange监测字段

```
@api.onchange('price', 'count')
def _onchange_price(self):
 self.totalPrice = self.count * self.price
```

- 当被监测的字段更改后，触发方法；
- 和 **depends** 不同的是，这里的触发不需要点击保存，界面实时修改，实时触发；

4、模型约束

（1）constrains

```
@api.constrains('count')
def _check_count(self):
 for product in self:
 if product.count < 0:
 raise ValidationError('Input count cannot less 0 :%s' %
product.count)
```

- 点击保存，保存数据库之前，会触发方法，检查特定字段

（2）数据库约束

```
_sql_constraints = [
  ('name_description_check',
    'CHECK(name != description)',
    '名称和描述不能相同'),
  ('name_check',
    'UNIQUE(name)',
```

```
    '名称不能重复')
]
```

- 数据库约束即使用pg语句进行数据检查
- 使用 **_sql_constraints** 定义，这是一个三元素的元组列表，三个元素依次是：sql约束的名称、约束规则和违反规则的警告

pg的约束函数可以参考：https://www.postgresql.org/docs/9.3/ddl-constraints.html

三、代码示例

model.py

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api

class TestModule(models.Model):
    _name = 'test_module.model' #模型名称，必须，后面用的着
    _description = u'测试模型'

    name = fields.Char(string='Name', size=50)
    value = fields.Integer(string='Data', default=99)
  description = fields.Text('Description')
```

数据模型添加完成后，需要在目录下的**__init__.py** 文件中import

```
from . import models
```

该数据模型将会保存在数据库的 **test_module_model** 表中