

后端低代码开发平台解决方案

为写更少代码而奋斗



目录

CONTENTS

>> 什么是低代码

>> 实现技术选型

>> 实现步骤

>> 讨论



1 什么是低代码

对应用程序编写的模式或套路进行抽象，使之模板化

把可变的逻辑变成配置数据

继而让配置可视化



什么是低代码

- 》 不是无代码或零代码，以配置来代替代码
- 》 代码的模板化、套路化的复用
- 》 通过配置来实现不同业务场景的需求
- 》 Django程序员已经接触过一种低代码应用（ Django Admin ）



低代码第一阶：模板化

Django administration Welcome, orzel Documentation / Change password / Log out

Home > Dynamic > Projects

Select project to change Add project +

Name	Homepage	Status	Parent	Identifier	Is public	Filter
DjangoRedmineAdmin	http://labs.freehackers.org/wiki/djangoredmineadmin	Open	(None)	djangoredmineadmin	True	By status All Open Archived By is public All False True
Vim development		Open	(None)	vimdev	True	
Zeta GCC	http://www.freehackers.org/Zeta	Open	Zeta	zeta-gcc	True	
Zeta binutils	http://www.freehackers.org/Zeta	Open	Zeta	zeta-binutils	True	
Yzis	http://www.yzis.org	Open	(None)	yzis	True	
Python Mesh Viewer	http://labs.freehackers.org/wiki/pythonmeshviewer	Open	(None)	pythonmeshviewer	True	
Gentoo Dependencies Browser	http://labs.freehackers.org/wiki/gentoo-dep-browser	Open	(None)	gentoo-dep-browser	True	
Convex Processing	http://www.freehackers.org/Capricelli/Academic	Open	(None)	cvx-processing	True	
Zeta	http://www.freehackers.org/Zeta	Open	(None)	zeta	True	
Opale	http://www.freehackers.org/Opale	Open	(None)	opale	True	

10 projects

```
class ProjectAdmin(admin):  
    list_display = ['name', 'homepage', 'status',  
                    'parent', 'identifier', 'is_public']  
    list_filter = ['status', 'is_public']
```

低代码第二阶：配置化

二阶：配置化界面：供应商账号

供应商账号列表

供应商: 姓名: 性别:

手机号码:

[新增](#) [删除](#)

供应商	姓名	头像	手机号码	角色	状态	注册时间	操作
江苏龙马供应有限公司	陈国新		18514441002	一级供应商	正常	2019-09-28 11:14:34	编辑 删除
江苏龙马供应有限公司	谷永刚		18114441002	一级供应商	待审核	2019-09-27 21:02:03	编辑 通过审核 驳回申请
江苏龙马供应有限公司	陈国新		18514441002	二级供应商	正常	2019-09-10 09:29:09	编辑 删除

```
{
  'provider_accountn': {
    'display': ["provider", "name", "avatar", "mobile", "role", "status", "create_time"],
    'filter': ["provider", "name", "role", "mobile"]
  }
}
```

低代码第三阶：配置可视化

筛选条件: 供应商 × 姓名 × 角色 ×

表单类型: ☒ 数据编辑 ☐ 页面展示

列

字段	名称	操作
供应商	供应商	删除
姓名	姓名	删除
头像	头像	删除
手机号码	手机号码	删除
角色	角色	删除
状态	状态	删除
注册时间	注册时间	删除
添加一列		

操作集

删除 ×

通过审批 ×

驳回申请 ×

审批 ×

删除审批 ×

操作类型:

操作名称:

我们的理想开发过程

- 》 梳理清楚业务，进行可视化建模。
- 》 默认一套管理后台出现了，同时可以配置界面细节
- 》 默认增删改查API生成了，可以配置逻辑、新增接口、自动生成OpenAPI文档
- 》 使用触发器及云函数机制编写额外的业务逻辑



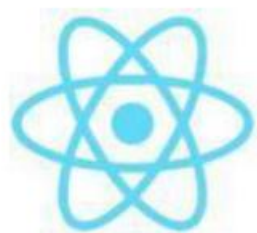
2 实现技术选型

NoSQL : MongoDB ? 大部份低代码开发平台的选择。

关系数据库 ? 兼容传统技术栈的实际需要。



开发框架



React

django



数据库选型：NoSQL



mongoDB



数据库选型：关系数据库

ORM Rule them all.



理想中的低代码平台

- 》 使用主流的技术，可维护可扩展性高
- 》 支持个性化需求，提供扩展开发的能力
- 》 即配即得，配置后马上看到运行效果 (运行时)
- 》 能够生成代码，以便进行二次开发，配置及开发可无缝连接



3 实现步骤

以模型驱动周边，当业力模型建立起来，其他的如API、管理界面待应当自然就拥有，基于可配置达到实现个性化需求的目的。

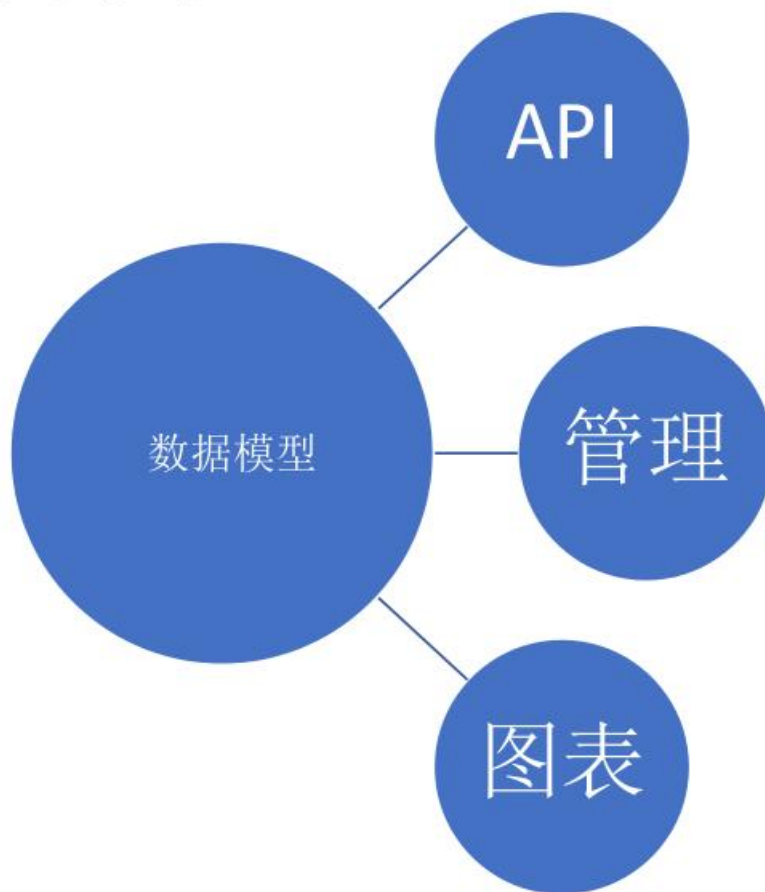


使用Django实现

- 》》 数据模型驱动一切
- 》》 设计中立的配置层，描述数据结构及各种行为
- 》》 通过元编程，从配置生成运行时代码
- 》》 触发器及云函数满足可扩展性

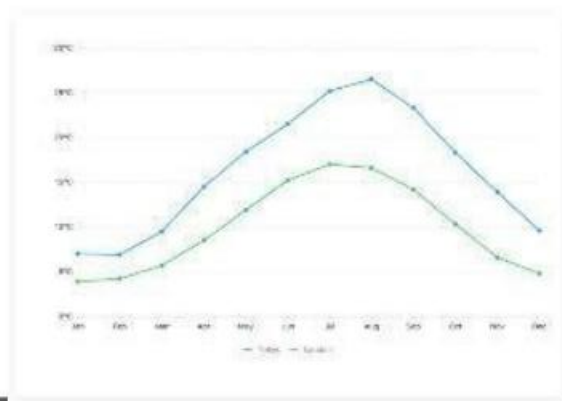


数据模型驱动一切

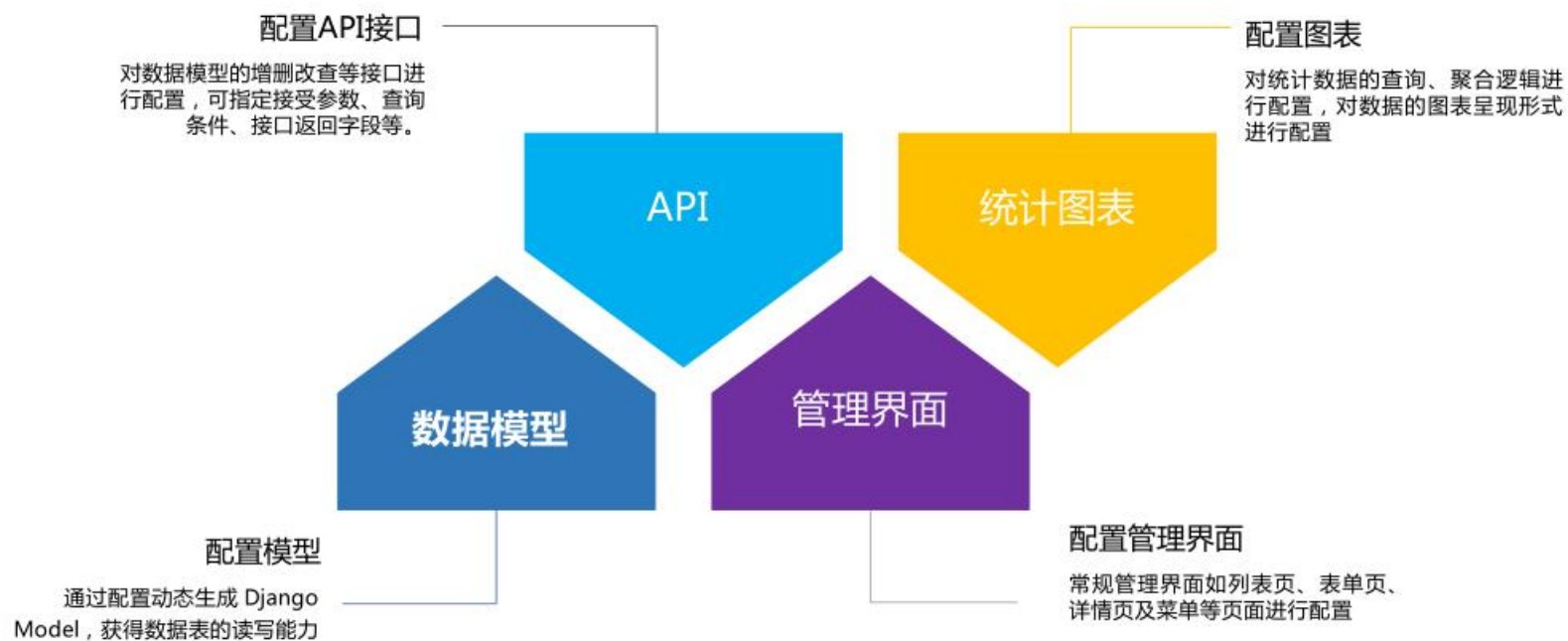


GET <http://gitmen.cn/member/>
PUT <http://gitmen.cn/member/1/>
POST <http://gitmen.cn/member/>
DELETE <http://gitmen.cn/member/1/>

列表页
表单页
详情页
导航菜单



设计中立的配置语言



模型定义配置

```
{
  "provider__account": {
    "name": "provider__account",
    "displayName": "供应商帐号",
    "fields": [
      {"name": "id", "displayName": "ID", "required": false, "type": "integer"},
      {"name": "provider", "displayName": "供应商", "required": true, "type": "ref", "ref": "provider__provider"},
      {"name": "user", "displayName": "user", "required": true, "type": "ref", "ref": "member__user"},
      {"name": "name", "displayName": "姓名", "required": false, "type": "string", "maxLength": 30},
      {"name": "avatar", "displayName": "头像", "required": false, "type": "image", "validators": [
        {"type": "url"}
      ]},
      {"name": "role", "displayName": "角色", "required": true, "type": "integer", "default": 0, "choices": [
        [1, "一级供应商"],
        [0, "二级供应商"]
      ]},
      {"name": "mobile", "displayName": "手机号码", "required": false, "type": "string", "maxLength": 20},
      {"name": "status", "displayName": "状态", "required": true, "type": "integer", "help": "", "choices": [
        [0, "待审核"],
        [1, "正常"],
        [2, "已屏蔽"]
      ]}
    ]
  },
},
}
```

API配置

```
{
  "slug": "list_category",
  "app": "baseshop",
  "model": "category",
  "operation": "list",
  "summary": "获取商品分类",
  "filter": [
    {"field": "category_page_show", "operator": "=", "value": "#{true}"},
    {"field": "slug", "operator": "contains", "value": "gmhome"}
  ],
  "ordering": "",
  "fields": ["name", "slug", "parent", "gmhome"],
  "parameter": [
    {"name": "page", "type": "PAGE_IDX", "required": false, "default": "", "desc": "页码"},
    {"name": "size", "type": "PAGE_SIZE", "required": false, "default": "", "desc": "每页数量"}
  ]
}
```

管理界面配置

```
{
  "provider__account": {
    "display": ["provider", "name", "avatar", "mobile", "role", "status", "create_time"],
    "filter": ["provider", "name", "role", "mobile"],
    "formFields": ["provider", "name", "user", "role", "mobile"],
    "inlineActions": ["delete", {
      "type": "action",
      "title": "通过审核",
      "show": "${status} === 0",
      "params": {
        "action": "update_status",
        "payload": {
          "status": 1
        }
      }
    }],
  },
}
```

图表配置

```
{
  model: 'baseshop__order',
  filters: [{field: 'status', operator: '=', value: 'success'}],
  filterFields: ['create_time'],
  metric: {
    totalAmount: {field: 'real_total', aggregate: 'Sum', displayName: '交易总金额'},
    productCount: {field: 'items__quantity', aggregate: 'Sum', displayName: '卖出商品总量'},
  },
  dimension: [
    'time', 'agencyId',
    'time': {'field': 'confirm_time', 'aggregate': 'TruncMonth', 'displayName': '下单时间'},
  ],
  charts: [
    {
      y: 'totalAmount',
      yFormat: '{{totalAmount}}元',
      x: 'time',
      legend: 'agencyName',
      geom: [
        { type: 'intervalStack' }
      ],
    },
  ],
}
```

元编程

- 》 动态生成模型 (Django)
- 》 动态生成ORM查询逻辑 (Django)
- 》 动态生成模型的序列化类 (Django)
- 》 动态生成Web组件 (React)



动态生成模型

```
Meta = type("Meta", (), {
    'app_label': 'myapp',
    'verbose_name': '动物',
    'verbose_name_plural': '动物'
})

attrs = {
    'name': models.CharField(max_length=32),
    '__module__': 'myapp.models',
    'Meta': Meta
}

Animal = type("Animal", (models.Model,), attrs)
```

动态生成模型

```
FIELDS = {
    'char': models.CharField,
    'int': models.IntegerField,
    'bool': models.BooleanField
}

Meta = type("Meta", (), {
    'app_label': 'myapp',
    'verbose_name': '动物',
    'verbose_name_plural': '动物'
})

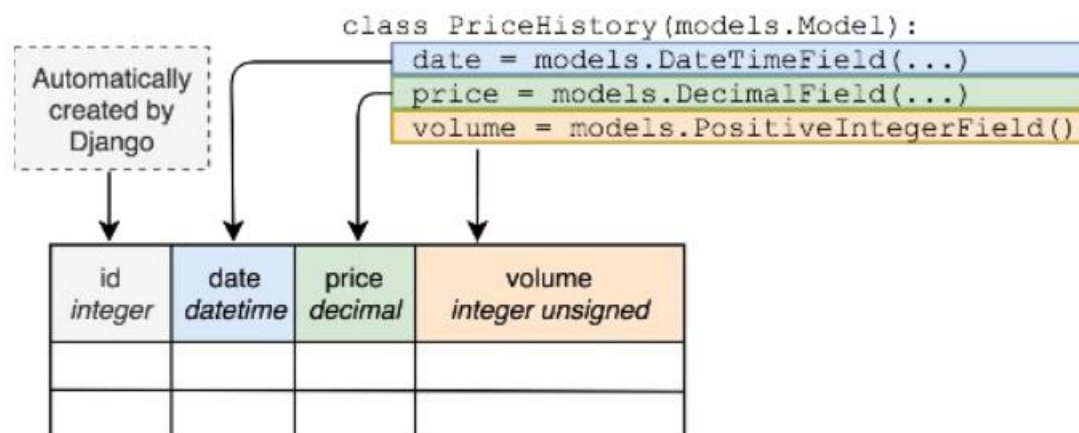
attrs = {
    'name': FIELDS['char'](**{'max_length': 32}),
    '__module__': 'myapp.models',
    'Meta': Meta
}

Animal = type("Animal", (models.Model,), attrs)
```

动态生成模型后续问题



进程中模型更新



Migrations

触发器（Signal）—— 对接口调用前后的扩展方式

```
@receiver(post_bsm_create, sender=Order, dispatch_uid='connect_order_providers')
def connect_order_providers(sender, instance, create, **kwargs):
    """把订单中所有关联的供应商连接起来
    """
    if not create:
        return
    items = instance.items.all().select_related('product')
    providers = []
    for item in items:
        providers.append(item.product.provider_id)
    providers = set([p for p in providers if p])
    if providers:
        instance.providers.set(providers)
```

云函数 —— 以纯函数方式编写个性化接口

```
@bsm_func(staff_required=True, name='update_status', model=Account)
def update_status(user, id=None, status=None):
    account = Account.objects.get(id=id)
    account.status = status
    account.save()
    user = account.user
    user.is_staff = status == 1
    user.save()
```

生成可交付代码

- 》 生成Django代码（服务端）
- 》 生成Spring Boot代码（服务端）
- 》 生成React代码（前端）
- 》 可基于生成代码进行二次开发，且兼容未来的生成代码





谢谢聆听！

